Investigation of an Orthogonal Encryption Technique for Deep Learning

Thien-Ngo Nguyen Le
Department of Computer Science
Colorado School of Mines
Golden, CO, USA
thienngole@mymail.mines.edu

Abstract

Encrypted Deep Learning Services is a great solution for those who work in the AI field but limited in resources. While avoiding expensive investment and providing flexibility on scaling, encrypted deep learning services also provide the privacy and confidentiality of the data for its users. However, this study will show that it may cost users more than what they can save due to the loss of up to 18% of accuracy on the trained model. Also, this study examines possible vulnerabilities.

Keywords—Machine Learning; Encryption; Deep Learning; Deep Learning Services; Encrypted Deep Learning Services; security and privacy; Known Plaintext Attack.

1. INTRODUCTION

Deep learning service (DLS) is getting more popular every day in our computing world. It is a great solution for small companies, researchers, and individuals to avoid an expensive investment on the high-performance computing system for the AI field. Deep learning services provide users fast and stable needed infrastructure. By using DLS, it is fairly easy to scale up the system without worrying about all the aspects of scaling, and users only have to pay for what they use. It is also a great deal for all short-term users.

Orthogonal encryption is one of many methods that are invented to use in DLS to protect the privacy and confidentiality of the data when using DLS, which is called Encrypted Deep Learning Services (EDLS). EDLS encrypts rows of data before sharing for the training process. This means that we can use EDLS to train, predict, and validate our models without losing the confidentiality of the data.

This study designed an experiment to evaluate the accuracy of a deep learning model trained on encrypted data and the strength of orthogonal encryption technique by applying the orthogonal encryption technique on two datasets, MINST and CIFAR10. It turns out that the accuracy of trained models and the strength of this orthogonal encryption technique depend heavily on the characteristic on the dataset it is applied to rather than the crypto algorithm itself.

2. DEEP LEARNING SERVICES AND ORTHOGONAL ENCRYPTION

2.1 *Deep Learning Services*

Generally, every machine learning model involves three phrases: configuration, training, and deploying. The training process of any machine learning model requires high computing power, so hardware configuration including high-end CPUs, GPUs, or FPGAs is normally used to accelerate the training process. However, the cost of such a system is an expensive investment that is not affordable to many people. Therefore, machine learning/deep learning as services is a good solution. Deep learning service (DLS) is a set of services that provide machine learning tools as part of cloud computing services offered by many big cloud providers such as Microsoft, Amazon, and IBM. DLS providers will host everything and be responsible for all aspects of scaling, then provide all the necessary assets to the users and the users only have to pay for the resources that they use. By utilizing these services, users can save time, money, and resources that would have been invested into hosting their own system. This makes DLS one of the fastest growing cloud services. According to Industry Research, DLS will grow with the compound annual growth rate of over 38 percent during the period of 2019-2023[1].

One of the benefits of DLS is that it provides users fast and stable needed infrastructure. DLS will free small companies, organizations, and even individuals from the burden of building their own in-house infrastructure from scratch. By using DLS, it is fairly easy to scale up the system without worrying about all the aspects of scaling. Since users only have to pay for what they use, DLS is also a great deal for all short-term users. However, when it comes to confidentiality, it is hard for DLS to guarantee this. Traditionally, to ensure data confidentiality, data owners need to host an in-house or private deep learning service. An in-house deep learning service may not be a viable option for many people and is often an expensive solution. Ensuring confidentiality of the data is a big challenge in DLS because in many cases, it is very important on who can get access to the training data, especially when the data is sensitive. For example, when a group of researchers want to train a model to detect a new disease, they would not want any third party to have access to their training data because it may contain information about patients, and it may be illegal for them to share. Hence, Encrypted Deep Learning Services (EDLS) was introduced as a solution. In EDLS only encrypted rows of data are shared for the training process. This means that we can use EDLS to train, predict, and validate our models without any revealing of the original data. This enables researchers to get access to more sensitive data for researching and still ensures the confidentiality of the data. Figure 1 shows a sample architecture of an EDLS.

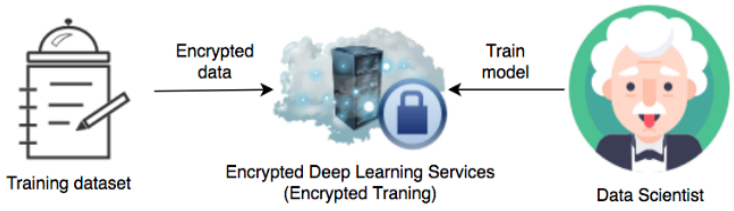


Figure 1: Encrypted Deep Learning Service Model

There have been quite a few encryption techniques introduced in EDLS such as Homomorphic Encryption and Orthogonal Encryption. This study will focus on investigating Orthogonal Encryption in EDLS.

2.2 Orthogonal Transformation based Encryption

Orthogonal transformation is a linear transformation that preserves the length of vectors. Let T be a linear transformation; if $||T(v)|| = ||v||$ where v is a vector, then T is called an orthogonal transformation. If $T(v) = A(v)$ then A is called an orthogonal matrix; matrix A columns and rows are orthogonal vectors [8]. Orthogonal transformation can use the transformational properties of the orthogonal matrix to transform an image into a new image that contains no information and usability of the original image. The original image can be retrieved by performing the dot product of the new image with the inverse of the orthogonal matrix. Orthogonal transformation has been applied to many areas such as in design of a feedforward neural network with optimum number of links and input nodes, design of a neural network operating with orthogonalized data for periodic process, and convergence assessment of networks [9].

Some researchers proposed to use orthogonal transformation as an encryption technique. They named this type of encryption as orthogonal encryption. Based on the properties of orthogonal transformation, orthogonal encryption can use an orthogonal matrix to encrypt a clear image (plain image) into a new image (cipher image) that has no information and usability of the plain image. The cipher image can be decrypted by the decryption key which is the inverse of the orthogonal matrix. A good orthogonal encryption method must be able to provide reduction in image size and the ability to be compressed after the encryption process.

This study investigates a specific orthogonal encryption technique introduced in patent US20190087689A1: METHODS AND PROCESSES OF ENCRYPTED DEEP LEARNING SERVICES [2]. There are many ways to retrieve an orthogonal matrix; the introduced schema performs a QR decomposition on an image to obtain the orthogonal matrix. Specifically, the operation decomposes an image as matrix A into a product A=Q*R where Q is an orthogonal matrix and R is an upper triangular matrix. Orthogonal matrix Q is then used as an encryption key to perform matrix multiplication with plain images to calculate cipher images. The inverse of Q will be used as the decryption key to decrypt cipher images by performing matrix multiplication with cipher images to obtain the corresponding plain image. According to [2], the proposed orthogonal encryption technique not only ensures the confidentiality of the data but also uses less memory and has faster prediction speed. The proposed orthogonal encryption technique uses 15 times less memory and significantly faster prediction speed than Crypto Net, a Neural Networks over data encrypted with Homomorphic Encryption. Figure 2 shows an image A decomposed into R (upper triangular matrix) and Q (orthogonal matrix used as encryption key). The decryption, Q^{-1}, is the inverse of Q. The cipher image is the product of the plain image from the MNIST dataset and encryption key Q. The decrypted image is the product of the cipher image and the decryption key Q^{-1}.

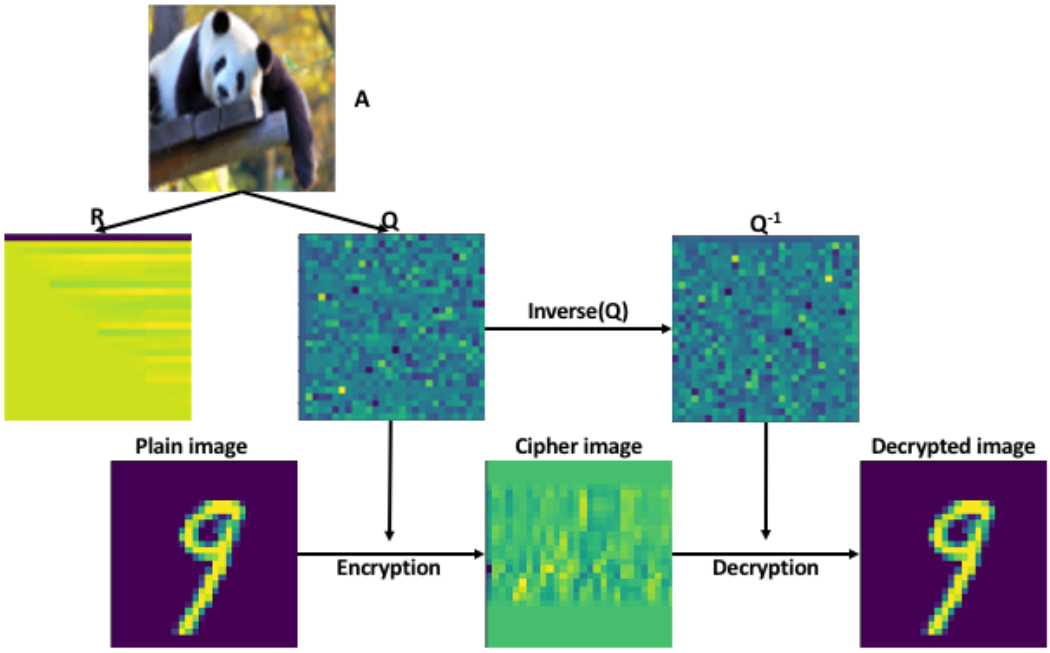


Figure 2: Orthogonal encryption

3. GOAL AND EXPERIMENTAL DESIGN

This section describes the experiments and the goals of this study. This study will use a convolutional neural network as a deep learning model to train on original versions and encrypted versions of both MNIST and CIFAR10 datasets, then measure the accuracy on models trained on encrypted data compared to clean data and analyze the strength of orthogonal encryption algorithms.

3.1 *Goals*

When it comes to machine learning, the cost of errors on any machine learning model can be really huge. For example, a false positive cancer diagnosis will cost both patient and hospital financially and mentally. Optimizing the accuracy of a machine learning model can mitigate that cost. Therefore, ***model accuracy*** is important. When it comes to cryptography, ***strength of a crypto algorithm*** is the most important characteristic because it ensures the privacy and confidentiality of the data.

The accuracy of a deep learning model depends heavily on the dataset it has been trained on. Training the same deep neural network on both un-encrypted and encrypted versions of two popular datasets in machine learning classification, MNIST and CIFAR10, will result in more reasonable accuracy evaluation. Since it involves encryption, the encryption key could be a factor that affects the training data and the accuracy of the model. Training models on datasets that are encrypted by different keys will add more reliability to the measurement process. Datasets encrypted by different keys also provide a better detail to analyze the strength of the crypto algorithm. This study has two main goals:

1. ***measure and compare the accuracy of convolutional neural network models on clean data and encrypted data.***
2. ***evaluate the strength of the orthogonal encryption technique by analyzing encrypted images' patterns and possible vulnerabilities.***

Trained models will be evaluated to report accuracies for analysis by answering questions such as, will encrypted datasets reduce the accuracy of deep learning models? If so, what could be the reason? If not, why encrypting data with orthogonal encryption does not change the accuracy of deep learning models? Will encryption keys affect the accuracy of models trained on the datasets they encrypted? If so, why do encryption keys affect model accuracy? The strength of orthogonal encryption method then will be analyzed by describing the observation of apparent patterns of encrypted images by different dimensions such as the same image encrypted by different keys and different images encrypted by the same key. Known plaintext attack vulnerability will also be analyzed to measure the strength of the orthogonal encryption method since it is a critical attack that every crypto algorithm has to protect against.

3.2 Experimental Design

This section will describe in detail the datasets used in the experiment, MNIST and CIFAR10. The structure of convolutional neural networks used in the experiment will also be described in this section. Lastly, specific training processes will be proved in this section.

3.2.1 Datasets and Tasks

MNIST and CIFAR10 are the two datasets used in this study. MNIST is a handwritten digits dataset that contains 70,000 image samples of 10 digits from 0 to 9. All the images are in gray scale and have a size of 28x28 pixel. The MNIST dataset was constructed by the National Institute of Standards and Technology (NIST) in 1999[3]. According to NIST [3], the dataset contains 70,000 digit images from 500 different writers taken from Census Bureau employees and American high school students. This is one of the most popular datasets that has been used for training in many image processing systems and also the dataset that is used in the patent mentioned above [2] to demonstrate the orthogonal encryption technique.

The CIFAR10 dataset is a subset of an 80 million tiny images dataset constructed by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton from the University of Toronto. The CIFAR10 dataset contains 60,000 color images of size 32x32 pixel divided equally into 10 classes of completely mutually exclusive objects: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck [4].

3.2.2 Convolutional Neural Network (CNN) Model

A Convolutional Neural Network (CNN) is a deep learning algorithm commonly applied to computer vision. CNN models can take in an input image, assign importance to various aspects/objects in the image and then be able to differentiate one from the other [10]. CNNs are the advanced version of fully connected deep learning networks that can reduce overfitting by taking advantage of the hierarchical pattern in data and using smaller and simpler patterns to

assemble more complex patterns. CNN reduces images into a form that is easier to process but still retains all the features of the images to result in a good prediction. Therefore, CNN required much lower pre-processing than other classification algorithms [10]. CNNs are widely used for image classification and recognition because of their high accuracy and low computing cost. Figure 6 shows a sample of a CNN structure.

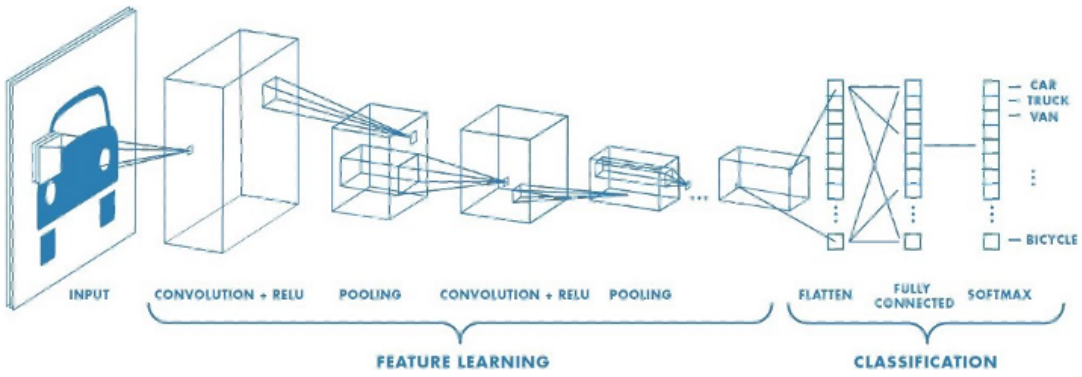


Figure 6: Sample Architecture of Convolutional Neural Network
Source: https://towardsdatascience.com

There are many choices for CNN architecture; this experiment used a typical CNN model with feature extraction by alternating convolution layers with subsampling layers. A subsampling layer is also called a Pooling which is a function that normally immediately follows a convolution layer in a CNN model to down sample the spatial size of output height and width. By doing this, the subsampling layer will reduce the number of parameters to be learned by the network to avoid overfitting during the training process. Therefore, implementing subsampling layers will help minimize the computational complexity of the CNN networks and give higher accuracy.

3.2.3 The Designed Experiments

This experiment will first train CNN models with clean images of MNIST and CIFAR10 datasets and then evaluate the models to get accuracies later used as the base to compare with accuracies from models trained by encrypted datasets. In the second task, MNIST and CIFAR10 datasets will be encrypted by 10 different keys using orthogonal encryption, then CNN models will be trained on them, and the accuracies will be analyzed. In the third task, encrypted images will be displayed for observation and analysis to focus on patterns of encrypted images. The last task in this experiment will discuss and demonstrate a possibility of known plaintext attack on orthogonal encryption.

(1) Accuracy on classifying clean examples.

In order to verify CNN models and get the accuracy on clean datasets, both clean MNIST and CIFAR10 datasets are used to train CNN models in this task. The MNIST dataset will be trained with three different CNN models: model C-PX1 is trained with one subsampling layer,

model C-PX2 is trained with two subsampling layers, and model C-PX3 is trained with three subsampling layers. These models will be trained on 60,000 train images size 28x28x1 (gray scale images size 28x28 pixels), and each model will be tested with 10,000 test images which have the same size as train images. Max Pooling [5] is used as subsampling layers for CNNs in this experiment. Figure 3 shows an example of Max Pooling implementation on CNN structure.

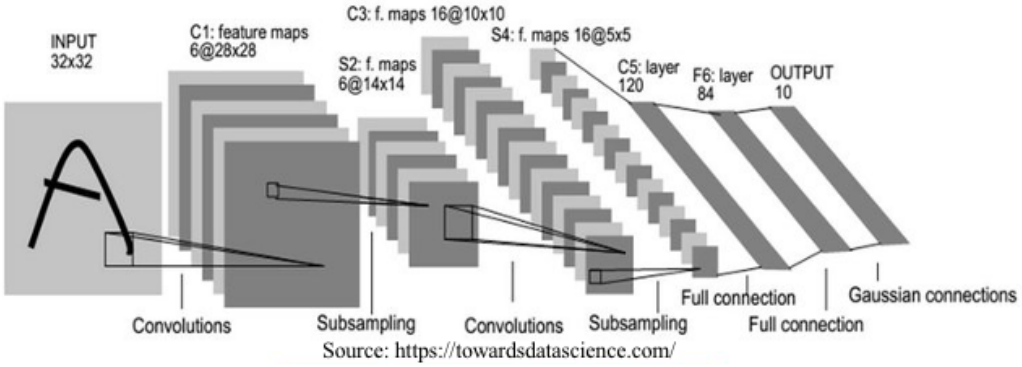


Figure 3: Max Pooling on CNN structure

The CIFAR10 dataset will be also trained on the Convolutional Neural Network that has the same architecture as models trained on the MNIST dataset but will be trained on 60,000 images of size 32x32x3 (color images size 32x32) and tested on 10,000 images having the same size as train images. The accuracy of each model will be then reported.

(2) Accuracy on classifying encrypted examples.

One goal of this study is to evaluate the accuracies of orthogonal encrypted datasets on deep learning models compared to clean datasets. This task will generate ten encrypted datasets from each MNIST and CIFAR10 dataset with ten different encryption keys, and then use them to train on the exact CNN models that trained un-encrypted datasets. The first five keys will be derived from QR decomposition of five randomly selected pictures. The other five keys will be generated by creating 32x32 matrices using the ortho_group class in a python library, SciPy. There are differences in size of images in MNIST and CIFAR10 datasets; images in MNIST are size of 28x28 and images in CIFAR10 are size of 32x32. So, the size of encryption keys for MNIST and CIFAR10 datasets also need to be 28x28 and 32x32 respectively because they have to match with the size of the images in order to perform the encryption calculation. Therefore, the five randomly selected pictures will be resized into 28x28 and 32x32 before performing QR decompositions to get five 28x28 Q matrices for MNIST dataset encryption and five 32x32 Q matrices for CIFAR10 dataset encryption. For the generated encryption matrices, five 32x32 random orthogonal matrices will be generated. For these five matrices, they will be kept the same size for CIFAR10 dataset encryptions and then be resized into five 28x28 matrices for MNIST dataset encryption. Figure 4

shows 5 selected pictures to use as the key and their corresponding encryption matrices size of 28x28 and 32x32. Figure 5 demonstrates the process of generating 28x28 matrices from 32x32 matrices to use as encryption keys for the MNIST dataset.

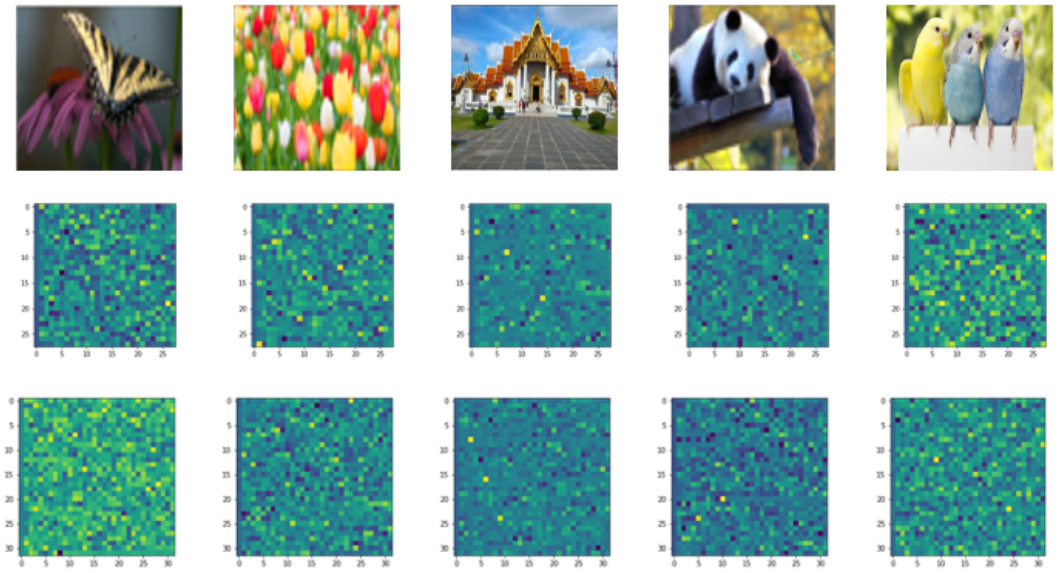


Figure 4: Pictures and their Qs matrices from QR decomposition

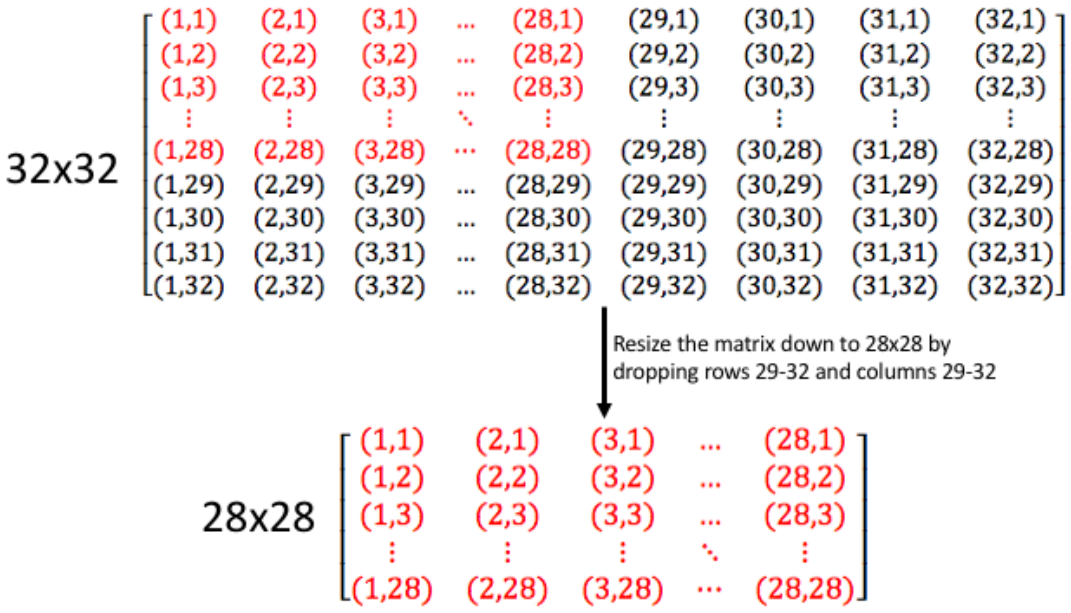


Figure 5: Matric 32x32 to 28x28

The reason for processing the key like this is to keep the keys that are used to encrypt MNIST and CIFAR10 datasets as similar as possible even if the keys have to be different in size. This will ensure that all the images are performed by the same transformation during the encryption process.

After finishing encrypting, the datasets then are used to train on CNN deep neural networks that have the same architectures as networks used to train un-encrypted datasets. The training

process will end up with 20 trained models: ten are trained with encrypted MNIST datasets and ten are trained with encrypted CIFAR10 datasets. The accuracy of each model will be reported.

(3) Visual analysis of encrypted examples.

Even with encrypted images, the patterns appearing on them can still provide a lot of information. Analyzing samples of encrypted images may show some patterns and characteristics related to the strength of orthogonal encryption. This section will use some sample encrypted images to perform some comparison and analysis.

The first task will analyze encrypted images from the same object that are encrypted by 10 different keys. Specifically, each digit in the MNIST dataset and each object in the CIFAR10 dataset will be encrypted with 10 different encryption keys and then displayed for comparison.

The second task will analyze encrypted images from different objects that are encrypted by the same key. In this task, each encryption key will be used to encrypt 10 different digits in the MNIST dataset and 10 different objects in the CIFAR10 dataset.

The third task in this section will analyze encrypted images from different objects that are encrypted by different keys. Each digit in the MNIST dataset will be encrypted by one of the 10 keys and the same process will be applied to the CIFAR10 dataset. The result then will be displayed for analysis.

(4) Possibility of known plaintext attacks

A known plaintext attack is a type of cryptanalysis in which the attacker knows at least one pair of clear images (plain images) and encrypted images (cipher images) then uses this to reveal secret information such as an encryption key. Encrypted archive ZIP files are a well-known example that is prone to known plaintext attacks. There are many types of software available on the internet that help determine the key needed to decrypt the archived files [11]. In image encryption (digital encryption), various cryptanalysis attacks have been introduced such as in [12][13][14][15][16]. According to the schema introduced in [11], a single-pixel encrypted system is totally vulnerable to known plaintext attack.

Known plaintext attacks are critical to any crypto algorithms. According to previous works on digital encryption, known plaintext attacks succeed on many digital encryption methods. This task will perform some analysis of the Orthogonal Encryption to find possible vulnerabilities. The orthogonal encryption technique being investigated in this study works based on the following function: $C = P * K$ where C is the cipher image, P is the plain image, and K is the orthogonal matrix used as the encryption key. The decryption key D will be the inverse of K. Theoretically, it is possible to derive the encryption key K if C and P are known. K can be derived by the equation $K = C * P^{-1}$ where K is the encryption key, C is the cipher image, and P^{-1} is the inverse of the plain image. We may not be able to directly find P^{-1} from P if P is a singular matrix because the singular matrix does not have any inverse [6]. According to [7], any singular matrix could be converted to a non-singular matrix. Matrices P are the presentation of images, so the singular characteristic of matrix P depends heavily on the image that it represents. If the dataset contains matrices that

represent gray scale images, there will be more singular matrices P than in the dataset that contains matrices representing color images. To test these hypotheses, the first task in this section will use a python script to analyze and report the number of singular and non-singular matrices contained on MNIST and CIFAR10 datasets. The second task will use some found non-singular matrices to derive the key by using the equation $K = C * P^{-1}$. Encryption key K' will then be reversed to find the decryption key D' to decrypt some images and compare resulting images with images decrypted by decryption key D.

4. Experimental Results and Analysis

This section will present the results of the experiment designed in section 4.2. Specifically, section 5.1 will present the results of the experiment performed on MNIST dataset; section 5.2 will present the results of the experiment on the CIFAR10 dataset; lastly section 5.3 will give a brief comparison, analysis and summary of all results.

4.1 Results on the MNIST dataset

(1) ***Accuracy on classifying clean examples***

As mentioned in section 3.2.3(1), the clear MNIST dataset is trained with three versions of CNN networks (one subsampling layer, two subsampling layers, and three subsampling layers). Table 1 below shows the models' accuracies trained on 50,000 images and tested on 10,000 images from MNIST dataset. Overall, ***all three versions got 99% on accuracy***; Two and three subsampling layer models got approximately 0.3% higher in accuracy compared to one subsampling layer model. This occurs because the two subsampling layer and three subsampling layer models can reduce overfitting during the training process.

Table 1: Accuracy on classifying clean MNIST dataset.

One subsampling layer	Two subsampling layers	Three subsampling layers
0.9900	0.9930	0.9937

(2) ***Accuracy on classifying encrypted examples***

Table 2 below shows the accuracies of ten CNN models trained on ten encrypted MNIST datasets that are encrypted by ten different encryption keys. Key1 to key5 are retrieved from performing QR decompositions on five randomly selected images; key6 to key10 are retrieved by orthogonal matrix generating using SciPy library in Python. Overall, the two and three subsampling layer models have higher accuracy than the one subsampling layer model. This is the same as the model trained on unencrypted data. The two and three subsampling layer models have approximately 0.2% higher accuracy than the one subsampling layer models. On average, models trained on datasets encrypted by QR decomposition keys have higher accuracy than those trained on datasets encrypted by orthogonal matrix generating keys (approximately 0.2%). The average accuracy of models trained on encrypted datasets is approximately 1% lower than models trained on a clean MNIST dataset. Because orthogonal encryption works based on matrix multiplication,

in each 28x28 pixel encrypted image there are 43,120 multiplication and addition operations performed. Each pixel on an encrypted image is determined by 28 multiplications and 27 additions; each image contains 28x28=784 pixel, so each encrypted image will need 43,120 operations ($[28 + 27] * 784 = 43{,}120$). Since all images in the dataset are encrypted by the same key, the more operations are performed, the more they make cipher images of the dataset closer to each other. This is a reasonable explanation for why the accuracy of models trained on encrypted datasets are a little bit lower than those models trained on clean datasets.

Table 2: Accuracy on classifying encrypted MNIST datasets.

Five QR keys	One subsampling layer	Two subsampling layers	Three subsampling layers
Key 1	0.9867	0.9885	0.9897
Key 2	0.9887	0.9883	0.9887
Key 3	0.9881	0.9904	0.989
Key 4	0.9873	0.9892	0.9893
Key 5	0.9869	0.9867	0.9903
QR Keys avg	**0.98754**	**0.98862**	**0.9894**
Five random keys	One subsampling layer	Two subsampling layers	Three subsampling layers
Key 6	0.9863	0.9887	0.9853
Key 7	0.9846	0.9878	0.9828
Key 8	0.9844	0.9878	0.9883
Key 9	0.9855	0.9881	0.9873
Key 10	0.9854	0.9854	0.9872
Randomly generated orthogonal keys avg	**0.98524**	**0.98756**	**0.98618**
Overall average over the 10 keys	**0.98639**	**0.98809**	**0.98779**

(3) ***Visual analysis of encrypted examples***

- ***Same object encrypted by different keys***

This task uses three sample digits from MNIST dataset. Figure 7 shows a clean image of digit 2. Figure 8 shows the encrypted images of digit 2 using 10 different keys designed for use in this experiment. Figure 9 and Figure 10 show un-encrypted and encrypted images of digit 3. Figure 11 and Figure 12 are the representations of digit 8. ***These samples show that the same object encrypted by different keys produces different cipher images, but there are still patterns of the un-encrypted images passed on those cipher images***. Focusing on those yellow pixels of cipher images in Figure 8, the shape of digit 2 is still presented in cipher images of digit 2 in Figure 10. This characteristic is also presented on cipher images of digit 3 and digit 8. The cipher images 2, 3, 4, 5, and 8 in Figure 10 show 2 shapes of digit 3 represented by yellow pixels; this could be generated during the orthogonal transformation process. The same characteristic and pattern are also presented in the cipher images of digit 8 in Figure 12. More demonstrations of other digits are presented in Appendix C. ***This shows that the orthogonal encryption technique does not completely hide all the information of the original image; There are common patterns from cipher images of the same object even if they are encrypted by different keys presented***.

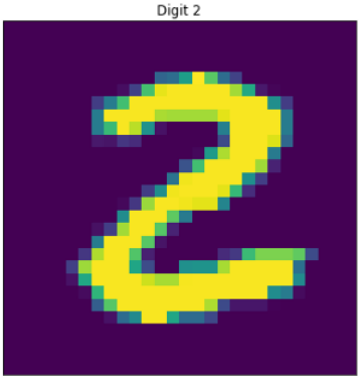


Figure 7: Un-encrypted digit 2

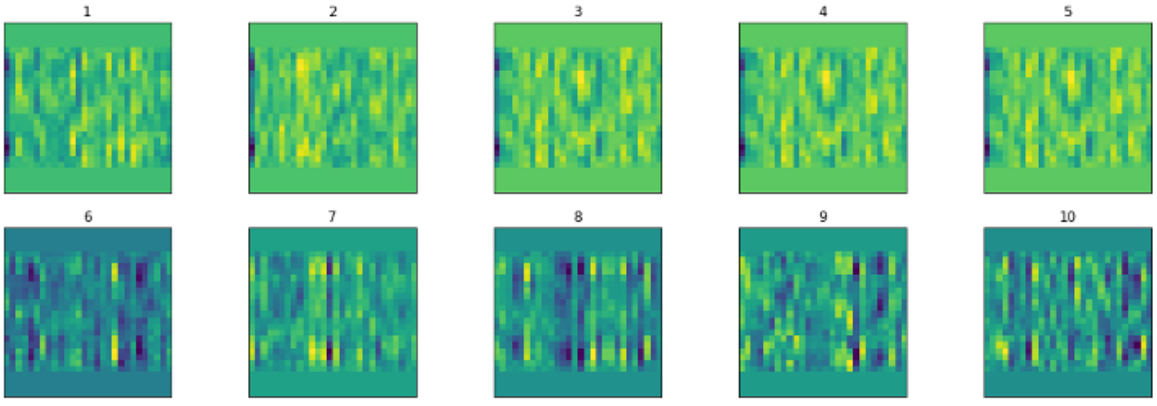


Figure 8: Digit 2 encrypted by 10 different encryption keys

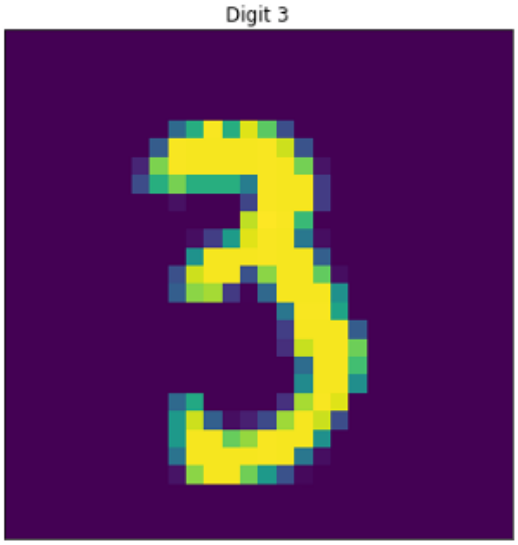


Figure 9: Un-encrypted digit 3

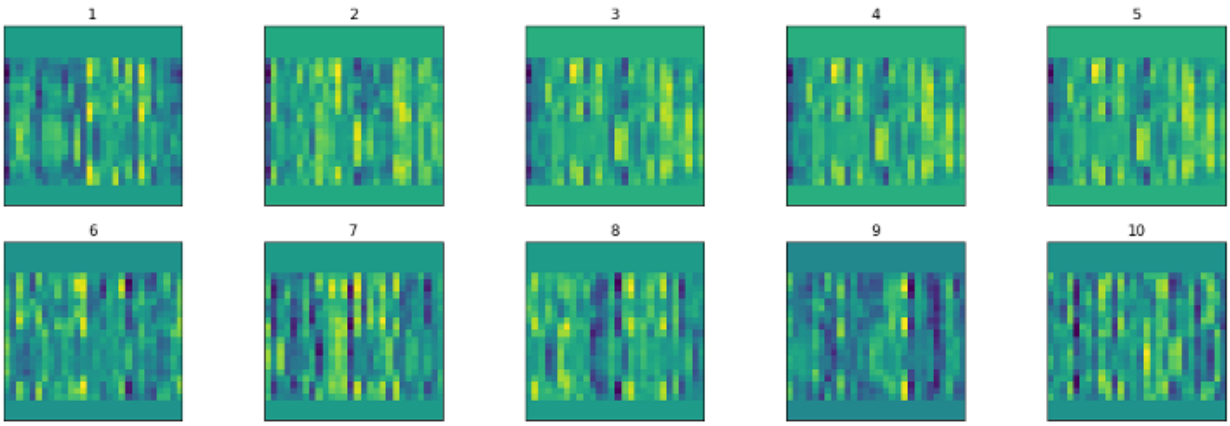


Figure 10: Digit 3 encrypted by 10 different encryption keys

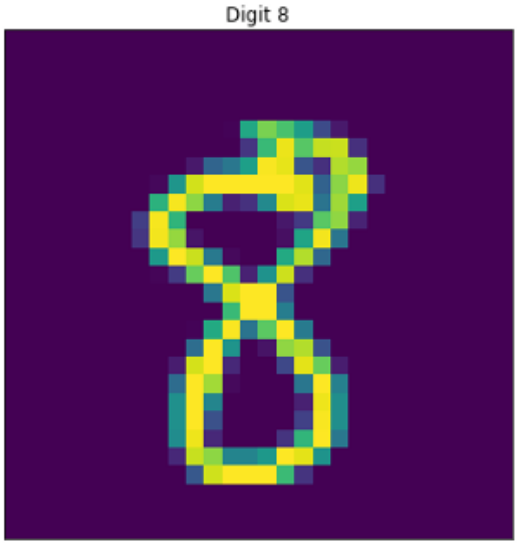


Figure 9: Un-encrypted digit 8

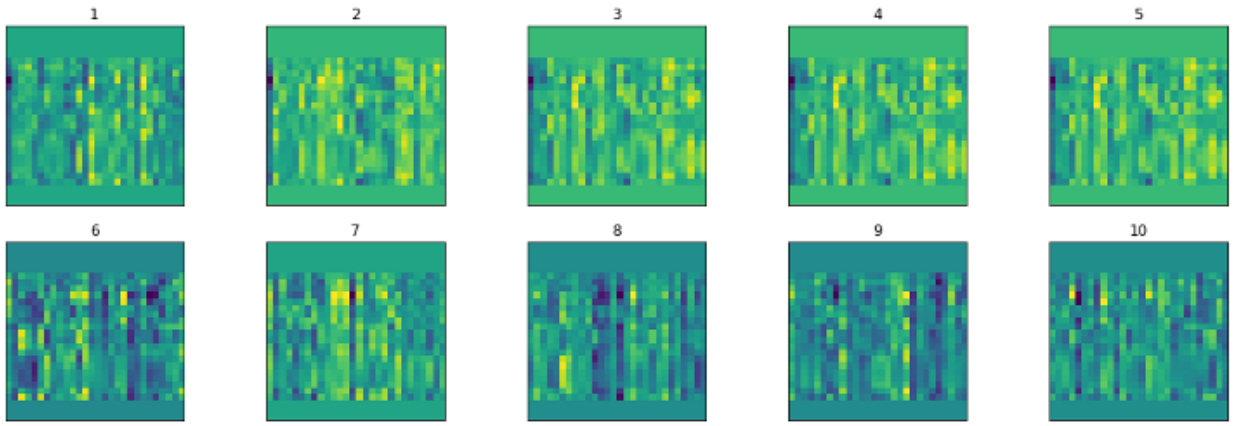


Figure 12: Digit 8 encrypted by 10 different encryption keys

- ***Different objects encrypted by the same key***

Figure 13, 14, and 15 show three examples of different objects being encrypted by the same key. Specifically, these figures show digit 0, 1, 2, 3, and 4 being encrypted by key2, key5 and key7. Key2 and key5 are derived from performing QR decomposition; key7 is retrieved from generating a random orthogonal matrix using SciPy library in Python. ***Although encrypted by the same key, each example clearly shows that different objects will have different cipher images when they are encrypted by the same key***. This provides a possibility of categorizing cipher images when an attacker can obtain a large number of cipher data which is obviously easy for the attacker. Being

able to categorize cipher images will give the attacker a lot of valuable information for cryptanalysis. ***Combining the cipher images from these three figures also proves the conclusion in the above section that cipher images of the same object have common patterns even when they are encrypted by different encryption keys***. More demonstrations of different objects encrypted by the same keys are presented in Appendix C.

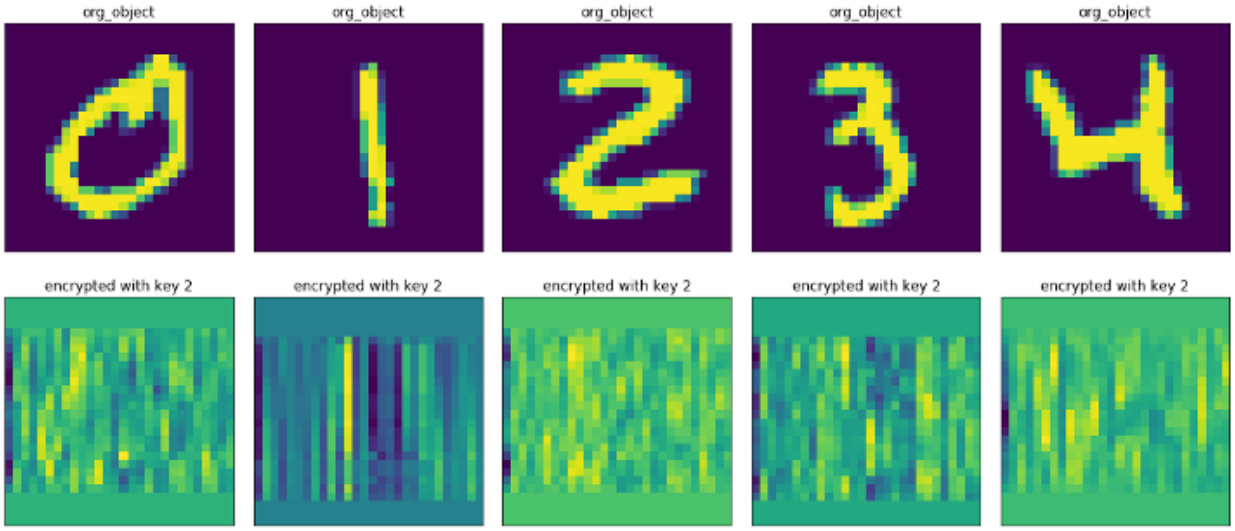


Figure 13: Digit 0, 1, 2, 3, 4 encrypted by key 2

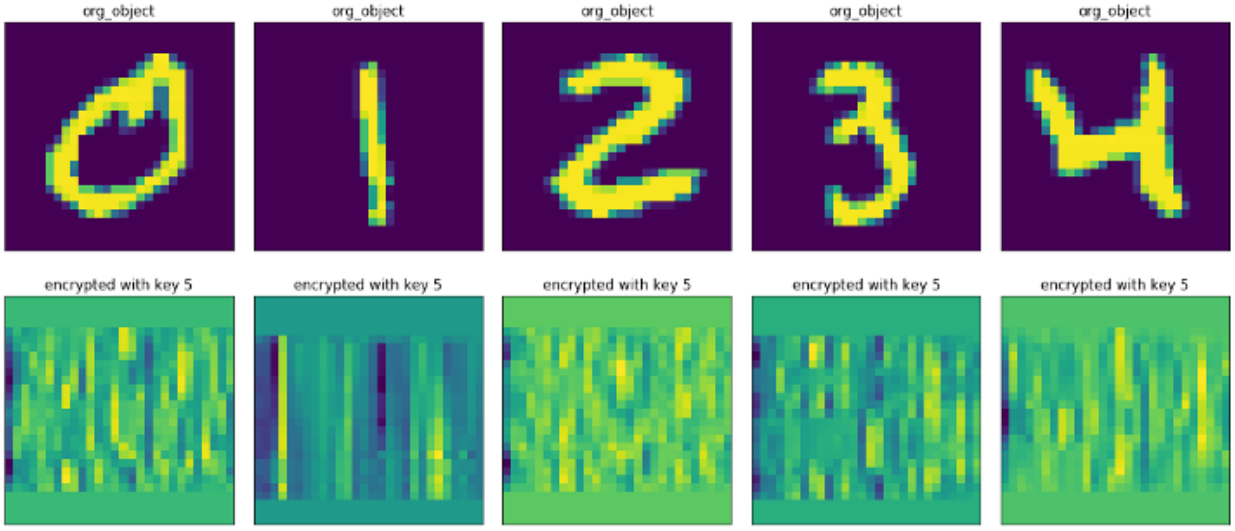


Figure 14: Digit 0, 1, 2, 3, 4 encrypted by key 5

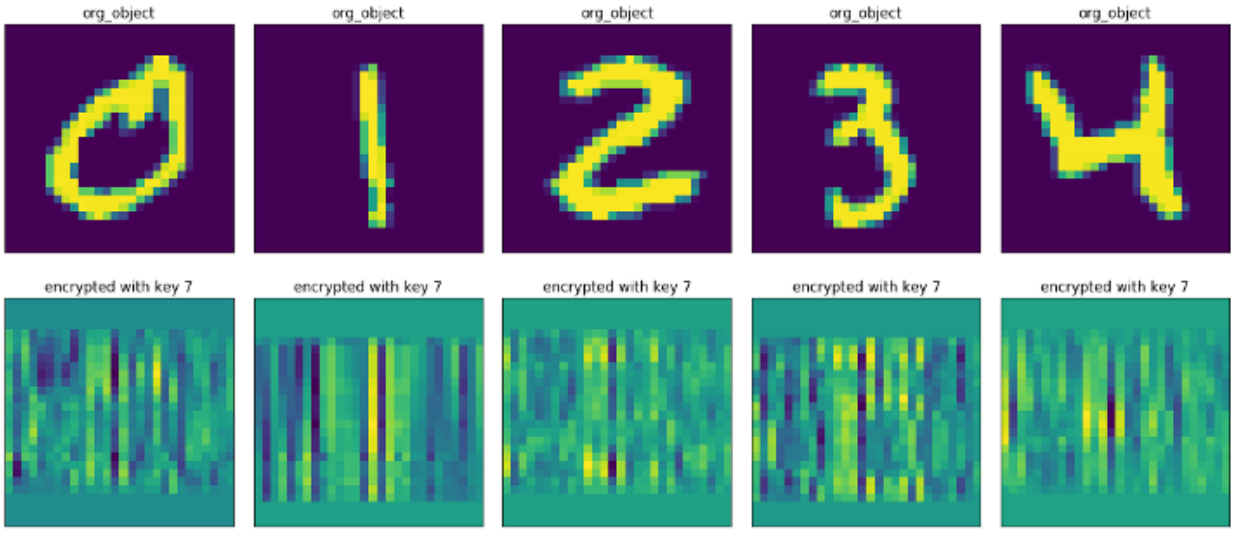


Figure 15: Digit 0, 1, 2, 3, 4 encrypted by key 7

❖ ***Different objects encrypted by different keys***

Figure 17 shows a picture of ten original digits in the MNIST dataset. Figure 18 contains the corresponding cipher images of digits in Figure 17; each digit is encrypted by a different key. It seems like when different objects are encrypted by different keys the orthogonal encryption technique can hide the most information from the original images. However, some patterns from the original images are still presented if we compare between the original image and its cipher image.

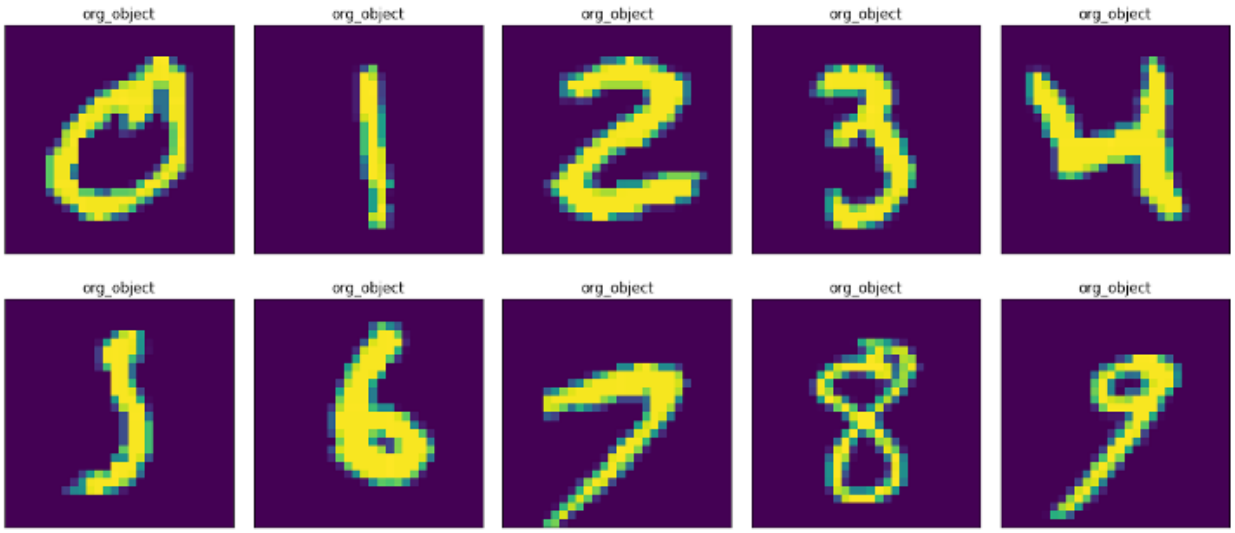


Figure 17: Original digits.

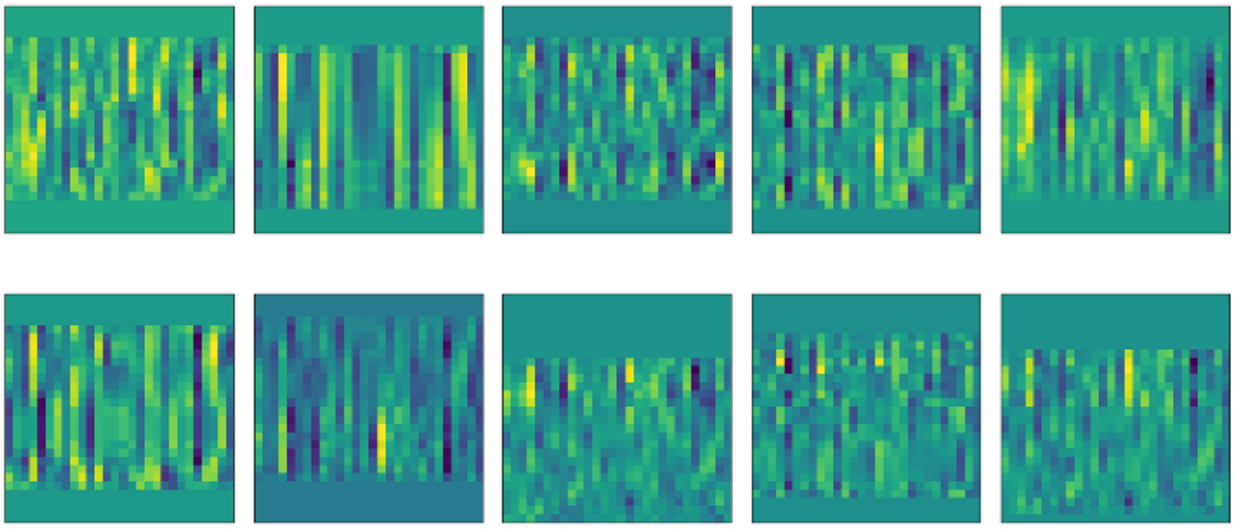


Figure 18: Digits 0-9 encrypted by key 1-10 respectively.

(4) Possibility of known plaintext attacks

As described in section 3.2.3(4), it is theoretically possible to derive the encryption key when both the cipher image and plain image are known. The encryption key can be derived by calculating the product of the cipher image and the inverse of the plain image. However, if the plain image is a singular matrix then its inverse cannot be directly derived from the plain image. This task will use a Python script to first check all the images on MNIST dataset to see if they are invertible. ***The result was that none of the images in the entire MNIST dataset (including train and test sets) is invertible. This means that the inverse of a plain image cannot be derived directly from a plain image.*** According to [6], it is possible to convert a singular matrix to a non-singular matrix, but this will be performed via shifting and transformation, so the derived key won't be absolutely the same as the real key. The similarity of the derived key and the real key depends on how much shifting we performed to convert the singular matrix to non-singular. Unfortunately, this task was not able to derive any key from any plain and cipher image pair in the dataset.

4.2 Results on the CIFAR10

From the experiment on MNIST dataset, CNN models with two and three subsampling layers produce better accuracies. Since the difference in accuracy between these two structures are very small, CIFAR10 datasets will be only trained on two subsampling layers models to save time on the training process.

(1) ***Accuracy on classifying clean examples.***

The result shows that ***the accuracy of the model trained on the clean CIFAR10 dataset is 73.51%***; this is lower than the model trained on the clean MNIST dataset. This is reasonable due to the difference in complexity of images in MNIST and CIFAR10 datasets. Specifically, the MNIST dataset is the set of gray scale images that only size 28x28 pixel, and images from the

CIFAR10 dataset are three channel (color) images, sized 32x32 pixel. Each image in the CIFAR10 dataset will have 240 pixels more than an image in the MNIST dataset.

(2) ***Accuracy on classifying encrypted examples***

Table 3 shows the accuracy of ten different models trained on ten encrypted CIFAR10 datasets. ***The overall accuracy of these ten models is approximately 18% lower than the accuracy of the model trained on the clean CIFAR10 dataset***. This is a huge difference in accuracy between models trained on un-encrypted and encrypted datasets. The difference in this experiment is a lot higher than the difference of the experiment on the MNIST dataset. However, considering the size of images in the CIFAR10 dataset, each image is the size of 32x32x3 pixel, so there will be 193,536 multiplication and addition operations performed on each encrypted image ($[(32 \text{ multiplication} + 31 \text{ addition}) * (32\text{x}32) * 3]$). This is 150,416 more operations needed than an encrypted image in the MNIST dataset. This means that more information of the original image is transformed, and this could cause confusion between cipher images of different objects during the training process. Confusion between cipher images is a good feature to have in crypto algorithms, but it may not be good for training data used in deep learning.

Table 3: Accuracy on classifying encrypted CIFAR10 datasets.

Keys	QR decomposition keys					Randomly generated orthogonal keys				
	1	2	3	4	5	6	7	8	9	10
Accuracy	0.5851	0.6248	0.5886	0.6202	0.5972	0.5341	0.5061	0.516	0.5112	0.4731
Overall average	0.55564									

(3) ***Visual analysis of encrypted examples***

- ***Same object encrypted by different keys***

Figures 19 and 21 show clean images of an airplane and cat from the CIFAR10 dataset; figures 20 and 22 show their corresponding encrypted images encrypted by ten different keys. ***Not like MNIST dataset, the cipher images of an object that is encrypted by different keys show fewer common patterns than what is shown in MNIST dataset***. However, the color pixels presented still show some imagination shape of the original object, but the cipher image hides a lot more information of the original image than in the MNIST dataset. Other sample objects encrypted by different keys are presented in Appendix C.

Figure 19: Un-encrypted Airplane

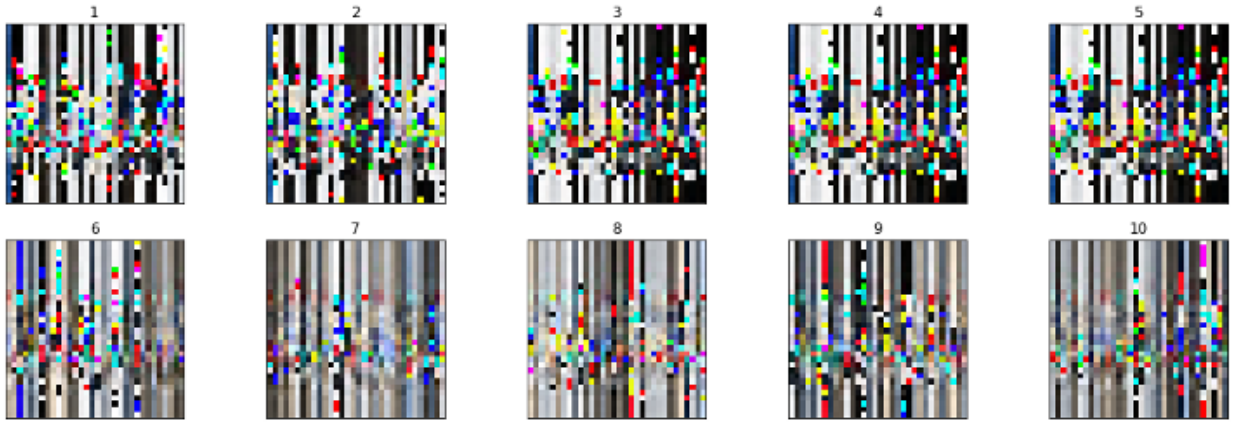


Figure 20: Airplane encrypted by 10 different encryption keys.

Figure 21: Un-encrypted Cat.

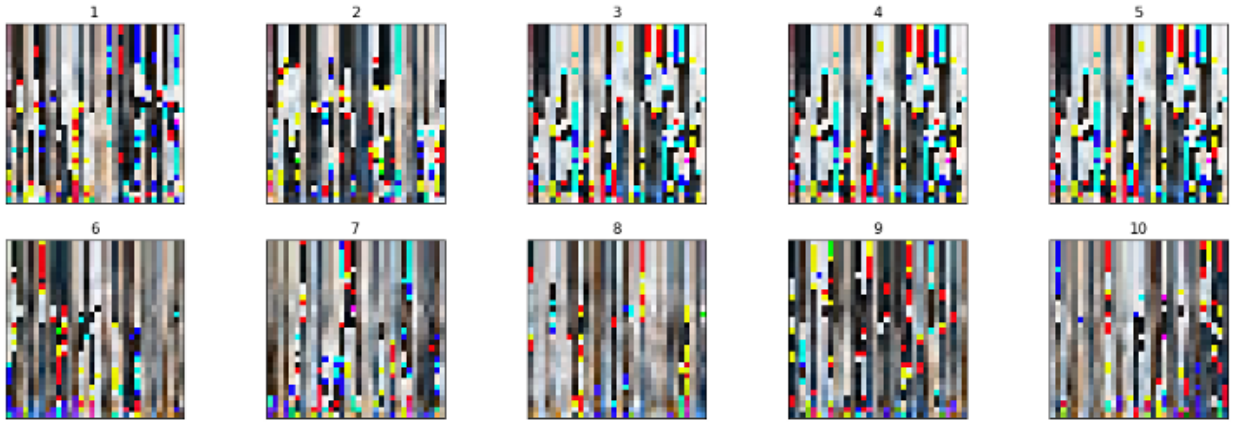


Figure 22: Cat encrypted by 10 different encryption keys.

❖ ***Different objects encrypted by the same key***

Figures 23, 24, and 25 below show the sample of a dog, frog, horse, ship, and truck encrypted by key2, key4 and key8. Key2 and key4 are derived by performing QR decomposition; key8 is from orthogonal matrix generating. ***These samples show the same characteristic shown in the MNIST dataset, a common characteristic between cipher images of the same object and a difference in cipher images of different objects encrypted by the same key.***

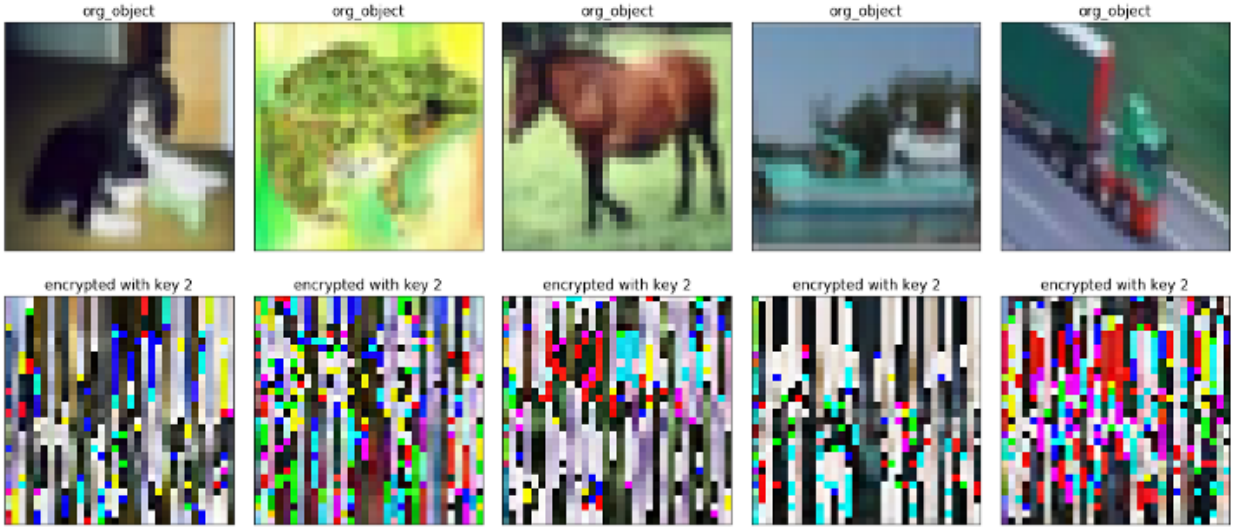


Figure 23: Dog, Frog, Horse, Ship, Truck encrypted by key 2

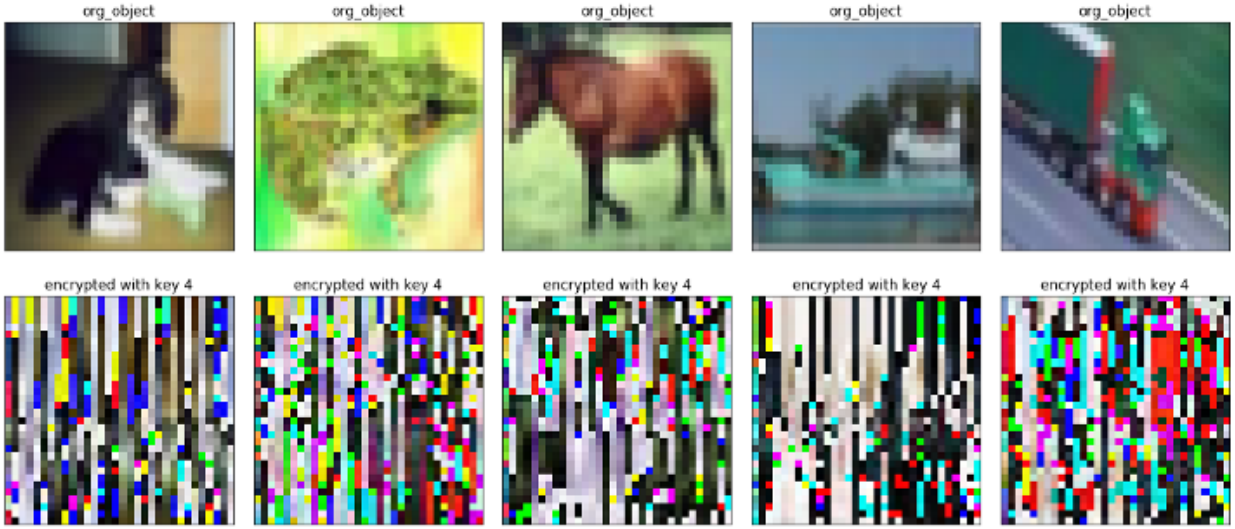


Figure 24: Dog, Frog, Horse, Ship, Truck encrypted by key 4

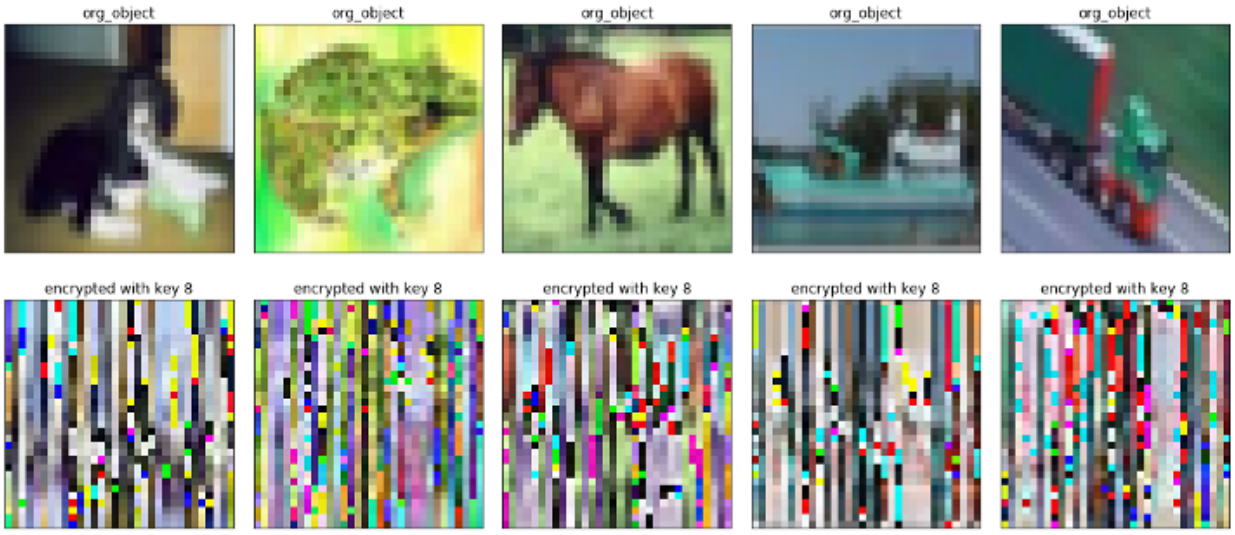


Figure 25: Dog, Frog, Horse, Ship, Truck encrypted by key 8

❖ ***Different objects encrypted by different keys***

Figure 26 shows original images of all ten objects from the CIFAR10 dataset, and figure 27 shows their corresponding cipher images. The cipher image of each object is encrypted by a different key. Similar to the MNIST dataset, when different objects are encrypted by different keys, the cipher images seem to hide more information of the original images than the same object with different keys or the same key for different objects.

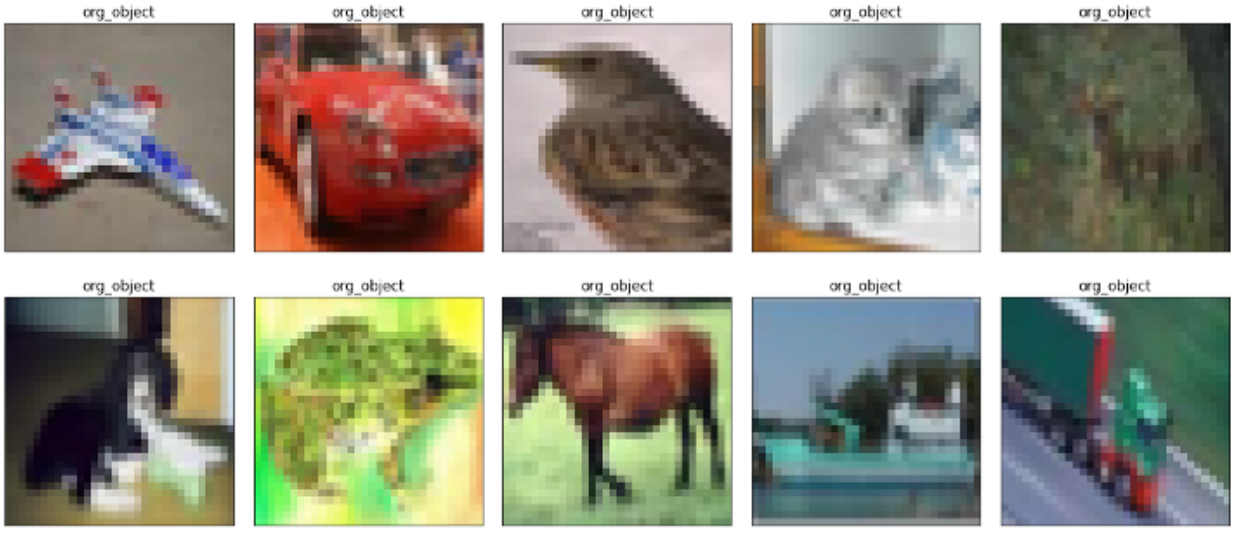


Figure 26: Un-encrypted objects.

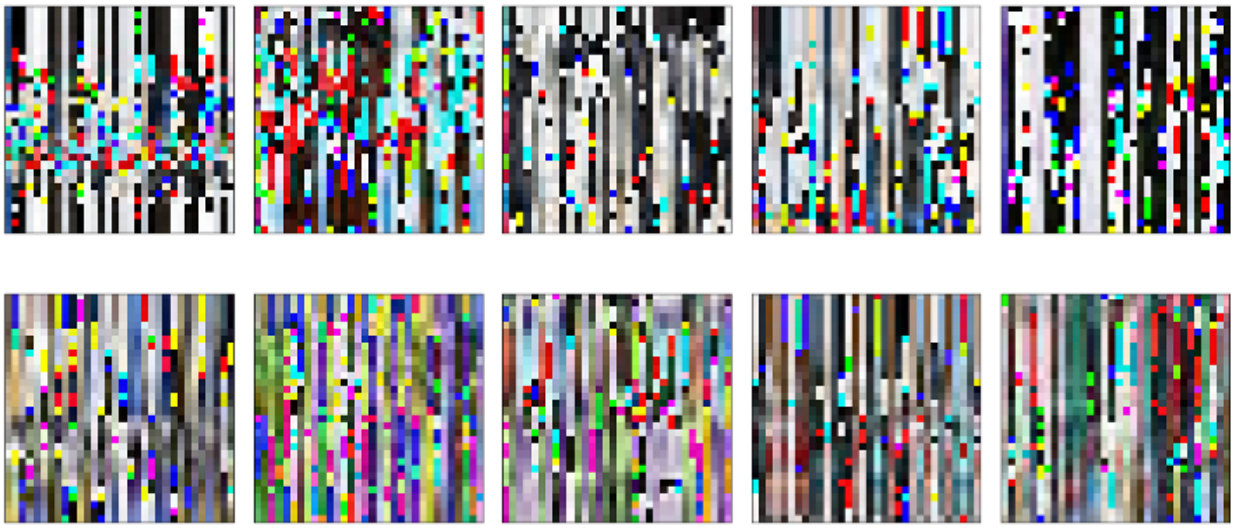


Figure 27: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck encrypted by key 1-10 respectively.

(4) ***Possibility of known plaintext attacks***

The same python script used in 4.1(4) is used to check if a matrix representing a plain image of images in the dataset is invertible. Since CIFAR10 is a set of 3 channel(color) images, this task will check for an invertible matrix on each channel. ***50000 images from the train dataset were checked; there are 49,187 black channel matrices that are invertible, 63 red channel matrices that are invertible, and 18 green channel matrices that are invertible***. This means that there is a very high possibility that the known plaintext attack could succeed on this dataset. A few demonstration attacks were performed in this task. Figure 28 below shows an example of the image derived by a known plaintext attack and its corresponding original image. Even the derived image and original image are not absolutely the same due to some shifting that may occur during the transformation and calculation processes, but the patterns of these two images are absolutely the same. Performing enough data analysis on the derived image could convert it to one that is more similar to the original image.

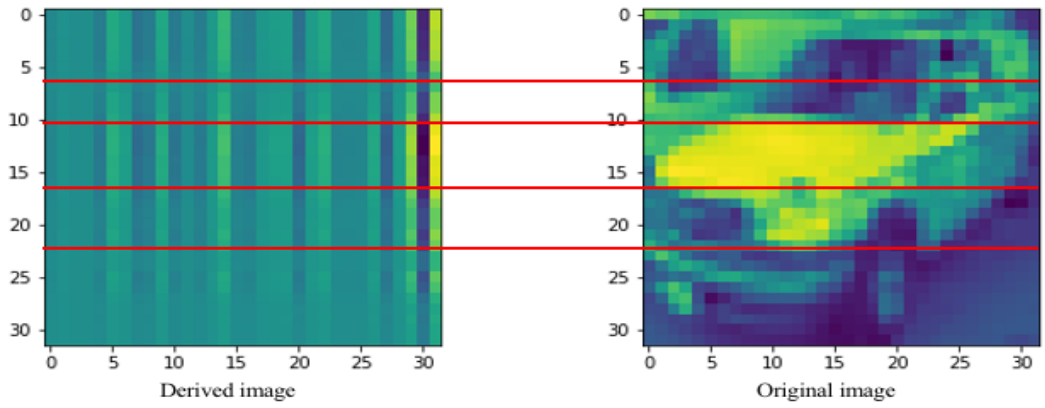


Figure 28: Derived image vs original image

4.3 Summary of the Results.

In summary, the accuracies of models trained on encrypted datasets are generally lower than those models trained on unencrypted datasets. The difference will vary from the size and quality of images that are encrypted. For example, the difference between accuracies of deep learning models trained on encrypted and unencrypted gray scale 28x28 pixel images from the MNIST dataset will be smaller than the different accuracies between models trained on encrypted and unencrypted 32x32 pixel color images from CIFAR10 dataset. This also ratios with the information each cipher image can hide from its original image. The higher information the cipher images can hide, the lower accuracy the deep learning model trained on it will have. The average accuracy of models trained on encrypted MNIST datasets is approximately 1% lower than the model that trained on unencrypted datasets. Moreover, those models trained on encrypted CIFAR10 datasets have an average of 18% lower accuracy than the model trained on unencrypted data.

It appears that different datasets present different strengths of the orthogonal encryption technique. In the MNIST dataset, encrypted images are passed on many patterns of the original images; this shows the weakness of the orthogonal encryption algorithm. Nevertheless, due to the characteristic of the images in the dataset, it is harder to perform a known plaintext attack since all the plain images in the dataset are represented by singular matrices which are uninvertible. On the other hand, the CIFAR10 dataset has higher resolution images which are represented by larger dimension matrices; this helps the cipher images because they can hide more information of the original images due to more transformation operators being performed. It will reduce the accuracy of the model train on them. Most of the images in the CIFAR10 dataset are represented by non-singular matrices, so it is fairly easy to derive their inverse for known plaintext attacks. So orthogonal encryption on the CIFAR10 dataset is more vulnerable to known plaintext attacks than the MNIST dataset.

5. CONCLUSION AND FUTURE WORK

Overall, the orthogonal encryption method will reduce the accuracy of the model trained on the encrypted data. Depending on the characteristics of each dataset, the reducing accuracy will vary. This study shows that the MNIST dataset with orthogonal encryption will reduce approximately 1% in accuracy compared to the unencrypted datasets, and the encrypted CIFAR10 datasets will reduce up to 18% accuracy compared to its unencrypted dataset. This result is affected by multiple factors; the biggest factor is the resolution of images in the dataset. The higher resolution image will require a higher pixel to represent; therefore, a higher number of transformation operations will be performed when the image is encrypted. This will make the cipher image contain no information of the original image that it represents, and the crypto algorithm is more secure. However, it will also reduce the accuracy of the model that is trained on this data. According to this study, accuracy and crypto algorithm strength are inverse ratios. Comparing MNIST and CIFAR10 dataset, MNIST is less secure, as it has a higher accuracy. CIFAR10, on the other hand, seems more secure than MNIST but has a lower accuracy.

Both MNIST and CIFAR10 datasets only contain 10 classes of objects. In the future, I would like to experiment more with the orthogonal encryption technique with a dataset that contains a larger number of classes. This study shows that the resolution of images in the dataset has significantly impacted the accuracy of deep learning models trained on encrypted data. It would give more confidence and reliability on that conclusion if a larger dataset with higher resolution images were used to experiment on orthogonal encryption. Nonetheless, this will also require more resources for the experiment.

6. REFERENCES

[1] Global Machine Learning-as-a-Service (MLaaS) Market 2019-2023. (n.d.). Retrieved November 27, 2020, from https://www.industryresearch.co/global-machine-learning-as-a-service-mlaas-market-14588435

[2] Chen, M. (2019). *U.S. Patent No. US2019008768 9A1*. Washington, DC: U.S. Patent and Trademark Office.

[3] THE MNIST DATABASE. (1999). Retrieved November 29, 2020, from http://yann.lecun.com/exdb/mnist/

[4] Krizhevsky, Alex (2009). "Learning Multiple Layers of Features from Tiny Images"

[5] Max-pooling / Pooling. (n.d.). Retrieved November 29, 2020, from https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

[6] Singular Matrix. (n.d.). Retrieved December 01, 2020, from https://mathworld.wolfram.com/SingularMatrix.html

[7] John. (2012, June 13). Making a singular matrix non-singular. Retrieved December 01, 2020, from https://www.johndcook.com/blog/2012/06/13/matrix-condition-number/

[8] Rowland, Todd. "Orthogonal Transformation." From MathWorld--A Wolfram Web Resource, created by Eric W. Weisstein. https://mathworld.wolfram.com/OrthogonalTransformation.html

[9] Kanjilal, Partha & Banerjee, Debarag. (1995). On the Application of Orthogonal Transformation for the Design and Analysis of Feedforward Networks. Neural Networks, IEEE Transactions on. 6. 1061 - 1070. 10.1109/72.410351.

[10] Saha, S. (2018, December 17). A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. Retrieved December 04, 2020, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[11] S. Jiao, T. Lei, Y. Gao, Z. Xie and X. Yuan, "Known-Plaintext Attack and Ciphertext-Only Attack for Encrypted Single-Pixel Imaging," in *IEEE Access*, vol. 7, pp. 119557-119565, 2019, doi: 10.1109/ACCESS.2019.2936119.

[12] M. Li, D. Lu, W. Wen, H. Ren and Y. Zhang, "Cryptanalyzing a color image encryption scheme based on hybrid hyper-chaotic system and cellular automata", *IEEE Access*, vol. 6, pp. 47102-47111, 2018.

[13] M. Li, H. J. Fan, Y. Xiang, Y. Li and Y. Zhang, "Cryptanalysis and improvement of a chaotic image encryption by first-order time-delay system", *IEEE Multimedia*, vol. 25, no. 3, pp. 92-101, Sep. 2018.

[14] M. Li, D. D. Lu, Y. Xiang, Y. Zhang and H. Ren, "Cryptanalysis and improvement in a chaotic image cipher using two-round permutation and diffusion", *Nonlinear Dyn.*, vol. 96, no. 1, pp. 31-47, Apr. 2019.

[15] L. Y. Zhang, Y. Liu, C. Wang, J. Zhou, Y. Zhang and G. Chen, "Improved known-plaintext attack to permutation-only multimedia ciphers", *Inf. Sci.*, vol. 430, pp. 228-239, Nov. 2017.

[16] Y. Zhang, D. Xiao, W. Wen and H. Nan, "Cryptanalysis of image scrambling based on chaotic sequences and Vigenère cipher", *Nonlinear Dyn.*, vol. 78, no. 1, pp. 235-240, Oct. 2014.

Appendix_A

December 7, 2020

1 MNIST Orthogonal encryption on Convolutional Neural Network(CNN)

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        from keras.utils.np_utils import to_categorical
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, AvgPool2D, BatchN
        from keras.preprocessing.image import ImageDataGenerator
        from keras.callbacks import LearningRateScheduler
        import matplotlib.pyplot as plt
        from keras.datasets import mnist
        from PIL import Image
        import scipy
        from scipy.stats import ortho_group
```

```
Using TensorFlow backend.
```

```
In [2]: #Load MNIST data
        (x_train, y_train), (x_test, y_test) = mnist.load_data()
        print("Done loading data!")
```

```
Done loading data!
```

2 Train with clear MNIST dataset

```
In [59]: # reshape dataset to have a single channel
         X_train = x_train / 255.0
         X_test = x_test / 255.0
         X_train = X_train.reshape(-1,28,28,1)
         X_test = X_test.reshape(-1,28,28,1)
         # one hot encode target values
         Y_train = to_categorical(y_train, num_classes = 10)
```

```
Y_test = to_categorical(y_test, num_classes = 10)

print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("Y_train shape: ", Y_train.shape)
print("Y_test shape: ", Y_test.shape)
```

```
X_train shape:  (60000, 28, 28, 1)
X_test shape:  (10000, 28, 28, 1)
Y_train shape:  (60000, 10)
Y_test shape:  (10000, 10)
```

Build Convolutional Neural Networks

In [60]:
```
nets = 3
org_model = [0] *nets

for j in range(3):
    org_model[j] = Sequential()
    org_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
            input_shape=(28,28,1)))
    org_model[j].add(MaxPool2D())
    if j>0:
        org_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
        org_model[j].add(MaxPool2D())
    if j>1:
        org_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
        org_model[j].add(MaxPool2D(padding='same'))
    org_model[j].add(Flatten())
    org_model[j].add(Dense(256, activation='relu'))
    org_model[j].add(Dense(10, activation='softmax'))
    #compile
    org_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=['
```

Train Convolutional Neural Networks

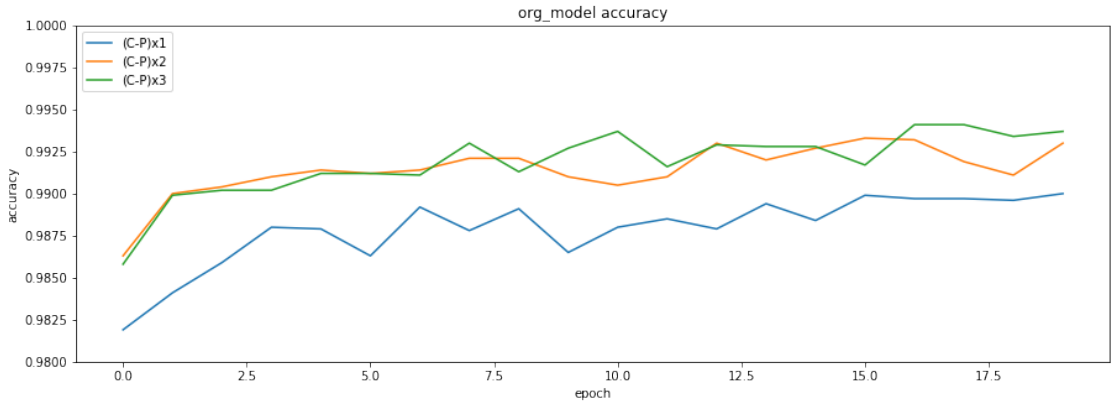
In [61]:
```
#Decrease learning rate by 0.95 each epoch
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)
```

In [62]:
```
# train
org_history = [0] * nets
names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
epochs = 20
for j in range(nets):
    org_history[j] = org_model[j].fit(X_train, Y_train, batch_size=80, epochs = epochs
        validation_data = (X_test, Y_test), callbacks=[annealer], verbose=0)
    print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}'
        names[j],epochs,max(org_history[j].history['acc']),max(org_history[j].history
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.99000
CNN (C-P)x2: Epochs=20, Train accuracy=0.99985, Validation accuracy=0.99330
CNN (C-P)x3: Epochs=20, Train accuracy=0.99983, Validation accuracy=0.99410
```

Plot accuracy

```
In [63]: plt.figure(figsize=(15,5))
         for i in range(nets):
             plt.plot(org_history[i].history['val_acc'])
         plt.title('org_model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(names, loc='upper left')
         axes = plt.gca()
         axes.set_ylim([0.98,1])
         plt.show()
```



Test

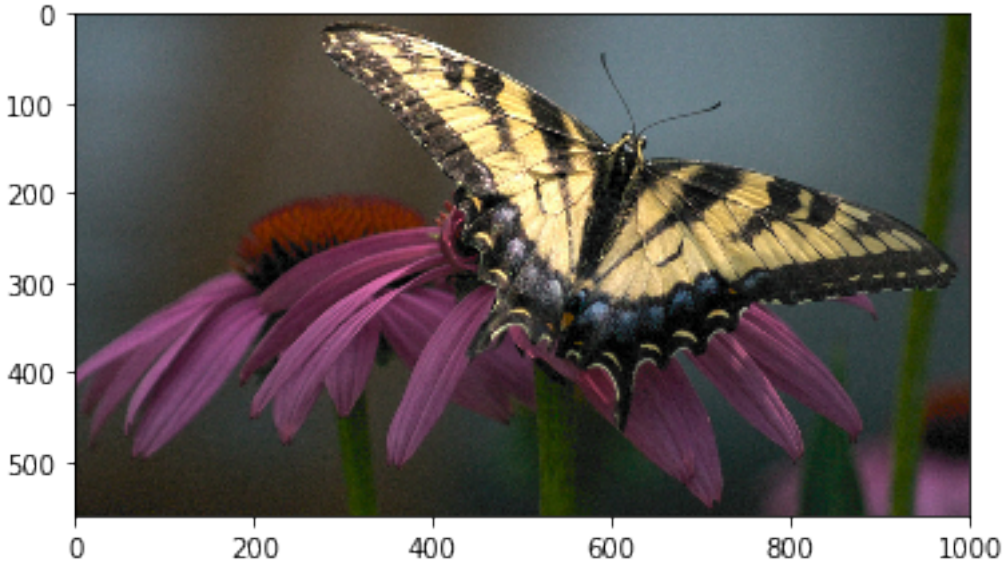
```
In [64]: for j in range(nets):
             test_loss, test_acc = org_model[j].evaluate(X_test,  Y_test, verbose=2)
             print('Acc on test dataset: ',test_acc)
```

```
Acc on test dataset:  0.99
Acc on test dataset:  0.993
Acc on test dataset:  0.9937
```

3 Train with encrypted MNIST dataset using key 1

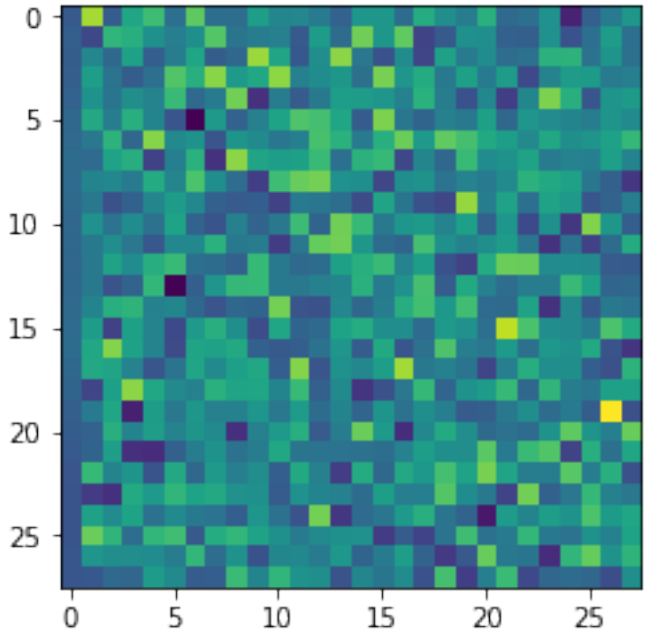
Drive orthogonal matrix to get encryption key Q1 and decryption key D1

```
In [140]: #Read key image as RGB
          key1 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key1.png
          #Show RGB key image
          plt.imshow(key1)
          #gray scale key image
          key1 = key1.convert('L')
          #resize key image
          key1 = np.resize(key1,(28,28))
          #convert to 2D array
          key1 = np.asarray(key1)
          #QR decomposition
          Q1, R1 = scipy.linalg.qr(key1)
          #Drive the decryption matrix
          D1 = np.linalg.inv(Q1)
```



Encryption key Q1

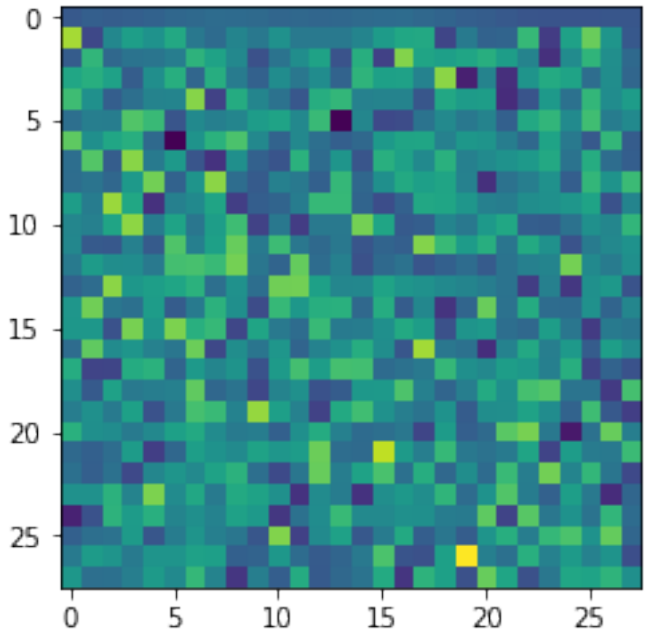
```
In [141]: plt.imshow(Q1)

Out[141]: <matplotlib.image.AxesImage at 0x1aa0a5cc0>
```



Decryption key D1

```
In [142]: plt.imshow(D1)
```

```
Out[142]: <matplotlib.image.AxesImage at 0x1aa163a58>
```



Encrypt the dataset

```
In [143]: #encrypt train set
          enc1_X_train = []
          for an_image in x_train:
              an_image = np.dot(an_image, Q1)
              enc1_X_train.append(an_image)
          print("Done encrypting ",len(enc1_X_train),"train images!")
          #encrypt test set
          enc1_X_test = []
          for an_image in x_test:
              an_image = np.dot(an_image, Q1)
              enc1_X_test.append(an_image)
          print("Done encrypting ", len(enc1_X_test), "test images!")
```

```
Done encrypting  60000 train images!
Done encrypting  10000 test images!
```

```
In [144]: enc1_X_train = np.array(enc1_X_train, dtype='float32') / 255.0
          enc1_X_test = np.array(enc1_X_test, dtype='float32') / 255.0
          enc1_X_train = enc1_X_train.reshape(-1,28,28,1)
          enc1_X_test = enc1_X_test.reshape(-1,28,28,1)
          # one hot encode target values
          enc1_y_train = to_categorical(y_train, num_classes = 10)
          enc1_y_test = to_categorical(y_test, num_classes = 10)

          print("enc1_X_train shape", enc1_X_train.shape)
          print("enc1_y_train shape", enc1_y_train.shape)
          print("enc1_X_test shape", enc1_X_test.shape)
          print("enc1_y_test shape", enc1_y_test.shape)
```

```
enc1_X_train shape (60000, 28, 28, 1)
enc1_y_train shape (60000, 10)
enc1_X_test shape (10000, 28, 28, 1)
enc1_y_test shape (10000, 10)
```

Build Convolutional Neural Networks

```
In [145]: nets = 3
          enc1_model = [0] *nets

          for j in range(3):
              enc1_model[j] = Sequential()
              enc1_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                      input_shape=(28,28,1)))
              enc1_model[j].add(MaxPool2D())
              if j>0:
```

```
            enc1_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
            enc1_model[j].add(MaxPool2D())
        if j>1:
            enc1_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
            enc1_model[j].add(MaxPool2D(padding='same'))
        enc1_model[j].add(Flatten())
        enc1_model[j].add(Dense(256, activation='relu'))
        enc1_model[j].add(Dense(10, activation='softmax'))
        enc1_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

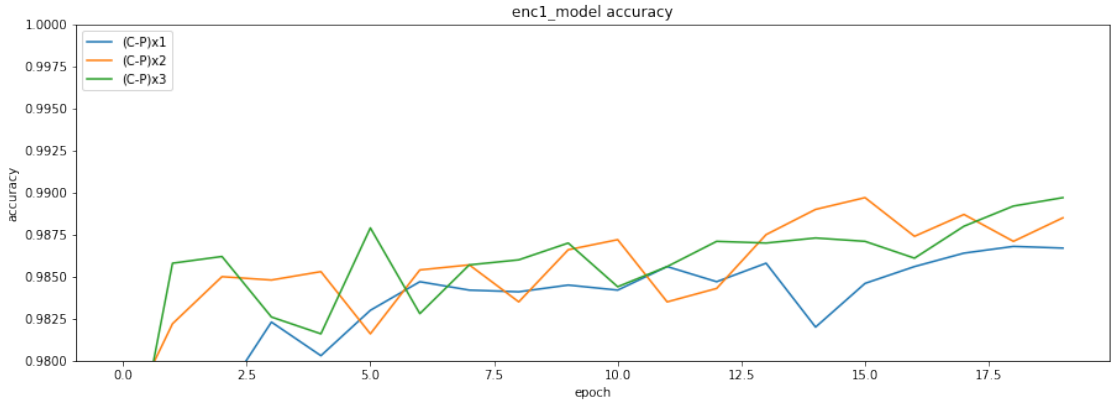
Train Convolutional Neural Networks

```
In [146]: #Decrease learning rate by 0.95 each epoch
          annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)

In [147]: # train
          enc1_history = [0] * nets
          names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
          epochs = 20
          for j in range(nets):
              enc1_history[j] = enc1_model[j].fit(enc1_X_train, enc1_y_train, batch_size=80, ep
                  validation_data = (enc1_X_test, enc1_y_test), callbacks=[annealer], verbose=(
              print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}
                  names[j],epochs,max(enc1_history[j].history['acc']),max(enc1_history[j].histo
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98680
CNN (C-P)x2: Epochs=20, Train accuracy=0.99960, Validation accuracy=0.98970
CNN (C-P)x3: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98970
```

Plot accuracy

```
In [148]: plt.figure(figsize=(15,5))
          for i in range(nets):
              plt.plot(enc1_history[i].history['val_acc'])
          plt.title('enc1_model accuracy')
          plt.ylabel('accuracy')
          plt.xlabel('epoch')
          plt.legend(names, loc='upper left')
          axes = plt.gca()
          axes.set_ylim([0.98,1])
          plt.show()
```



Test

```
In [149]: for j in range(nets):
              enc1_test_loss, enc1_test_acc = enc1_model[j].evaluate(enc1_X_test,  enc1_y_test
              print('Acc on encrypted test dataset: ',enc1_test_acc)
```

```
Acc on encrypted test dataset:  0.9867
Acc on encrypted test dataset:  0.9885
Acc on encrypted test dataset:  0.9897
```

4 Train with encrypted MNIST dataset using key 2

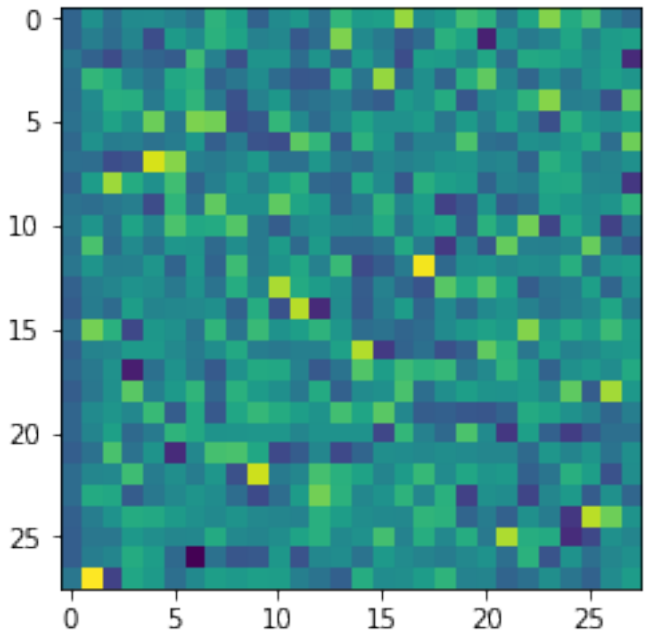
Drive orthogonal matrix to get encryption key Q2 and decryption key D2

```
In [77]: #Read key image as RGB
         key2 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key2.png'
         #Show RGB key image
         plt.imshow(key2)
         #gray scale key image
         key2 = key2.convert('L')
         #resize key image
         key2 = np.resize(key2,(28,28))
         #convert to 2D array
         key2 = np.asarray(key2)
         #QR decomposition
         Q2, R2 = scipy.linalg.qr(key2)
         #Drive the decryption matrix
         D2 = np.linalg.inv(Q2)
```

Encryption key Q2

```
In [78]: plt.imshow(Q2)
```

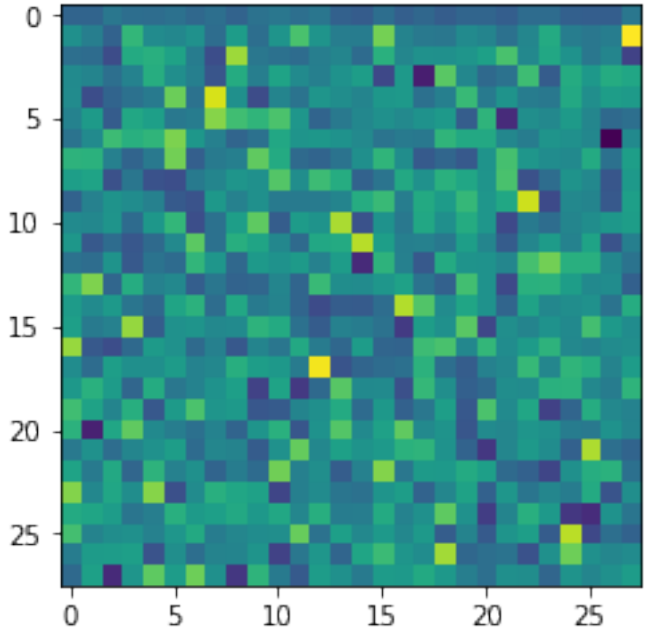
```
Out[78]: <matplotlib.image.AxesImage at 0x13ee7cef0>
```



Decryption key D2

```
In [79]: plt.imshow(D2)
```

```
Out[79]: <matplotlib.image.AxesImage at 0x148f95dd8>
```



Encrypt the dataset

```
In [80]: #encrypt train set
         enc2_X_train = []
         for an_image in x_train:
             an_image = np.dot(an_image, Q2)
             enc2_X_train.append(an_image)
         print("Done encrypting ",len(enc2_X_train),"train images!")
         #encrypt test set
         enc2_X_test = []
         for an_image in x_test:
             an_image = np.dot(an_image, Q2)
             enc2_X_test.append(an_image)
         print("Done encrypting ", len(enc2_X_test), "test images!")
```

```
Done encrypting  60000 train images!
Done encrypting  10000 test images!
```

```
In [81]: enc2_X_train = np.array(enc2_X_train, dtype='float32') / 255.0
         enc2_X_test = np.array(enc2_X_test, dtype='float32') / 255.0
         enc2_X_train = enc2_X_train.reshape(-1,28,28,1)
         enc2_X_test = enc2_X_test.reshape(-1,28,28,1)
         # one hot encode target values
         enc2_y_train = to_categorical(y_train, num_classes = 10)
```

```
enc2_y_test = to_categorical(y_test, num_classes = 10)

print("enc2_X_train shape", enc2_X_train.shape)
print("enc2_y_train shape", enc2_y_train.shape)
print("enc2_X_test shape", enc2_X_test.shape)
print("enc2_y_test shape", enc2_y_test.shape)
```

```
enc2_X_train shape (60000, 28, 28, 1)
enc2_y_train shape (60000, 10)
enc2_X_test shape (10000, 28, 28, 1)
enc2_y_test shape (10000, 10)
```

Build Convolutional Neural Networks

```
In [82]: nets = 3
         enc2_model = [0] *nets

         for j in range(3):
             enc2_model[j] = Sequential()
             enc2_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                     input_shape=(28,28,1)))
             enc2_model[j].add(MaxPool2D())
             if j>0:
                 enc2_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
                 enc2_model[j].add(MaxPool2D())
             if j>1:
                 enc2_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
                 enc2_model[j].add(MaxPool2D(padding='same'))
             enc2_model[j].add(Flatten())
             enc2_model[j].add(Dense(256, activation='relu'))
             enc2_model[j].add(Dense(10, activation='softmax'))
             enc2_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

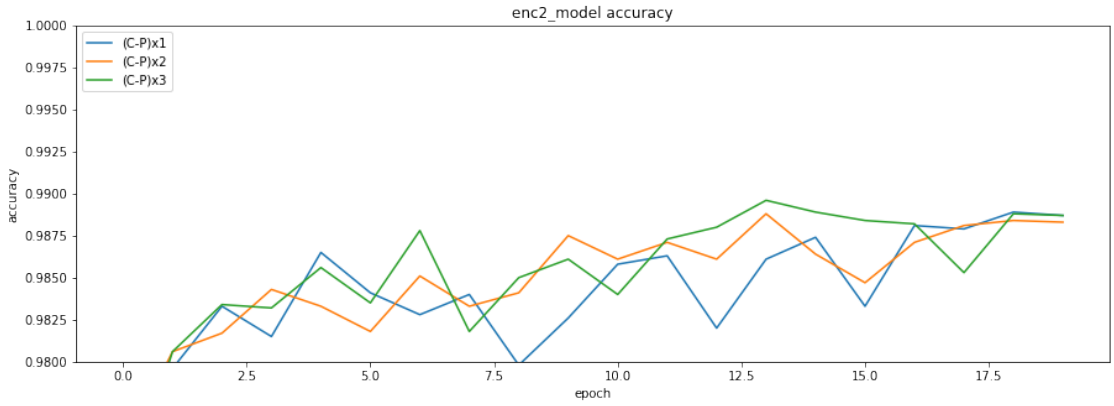
Train Convolutional Neural Networks

```
In [83]: #Decrease learning rate by 0.95 each epoch
         annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)

In [84]: # train
         enc2_history = [0] * nets
         names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
         epochs = 20
         for j in range(nets):
             enc2_history[j] = enc2_model[j].fit(enc2_X_train, enc2_y_train, batch_size=80, epo
                 validation_data = (enc2_X_test, enc2_y_test), callbacks=[annealer], verbose=0)
             print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}"
                 names[j],epochs,max(enc2_history[j].history['acc']),max(enc2_history[j].histo
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98890
CNN (C-P)x2: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98880
CNN (C-P)x3: Epochs=20, Train accuracy=0.99970, Validation accuracy=0.98960
```

Plot accuracy

```
In [85]: plt.figure(figsize=(15,5))
         for i in range(nets):
             plt.plot(enc2_history[i].history['val_acc'])
         plt.title('enc2_model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(names, loc='upper left')
         axes = plt.gca()
         axes.set_ylim([0.98,1])
         plt.show()
```



Test

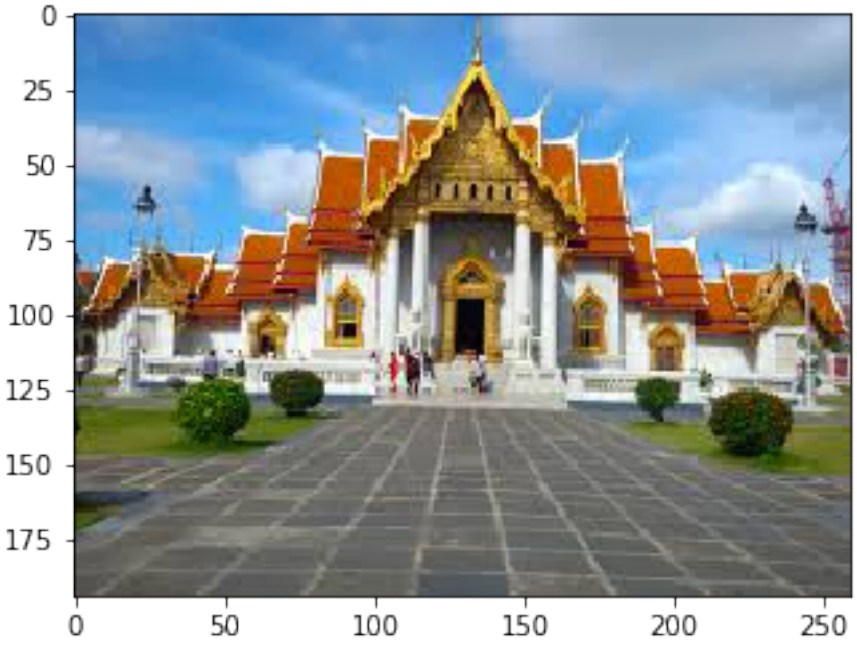
```
In [86]: for j in range(nets):
             enc2_test_loss, enc2_test_acc = enc2_model[j].evaluate(enc2_X_test,  enc2_y_test,
             print('Acc on encrypted test dataset: ',enc2_test_acc)
```

```
Acc on encrypted test dataset:  0.9887
Acc on encrypted test dataset:  0.9883
Acc on encrypted test dataset:  0.9887
```

5 Train with encrypted MNIST dataset using key 3

Drive orthogonal matrix to get encryption key Q3 and decryption key D3

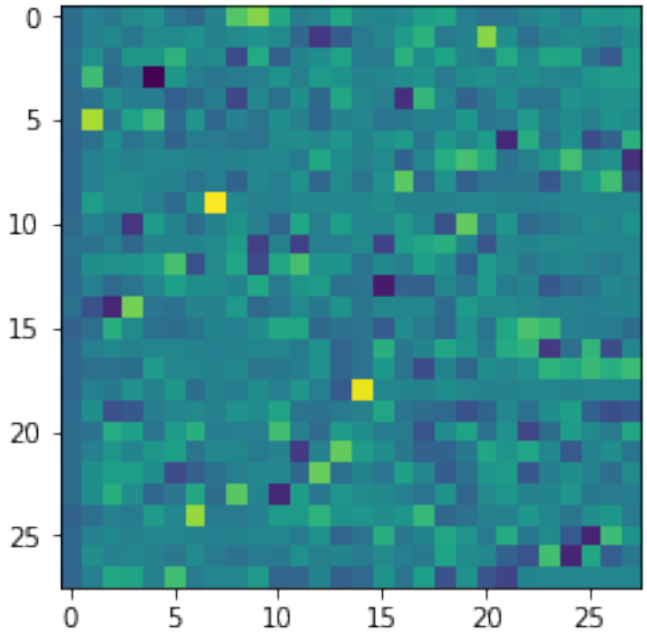
In [87]:
```
#Read key image as RGB
key3 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key3.png"
#Show RGB key image
plt.imshow(key3)
#gray scale key image
key3 = key3.convert('L')
#resize key image
key3 = np.resize(key3,(28,28))
#convert to 2D array
key3 = np.asarray(key3)
#QR decomposition
Q3, R3 = scipy.linalg.qr(key3)
#Drive the decryption matrix
D3 = np.linalg.inv(Q3)
```



Encryption key Q3

In [88]:
```
plt.imshow(Q3)
```

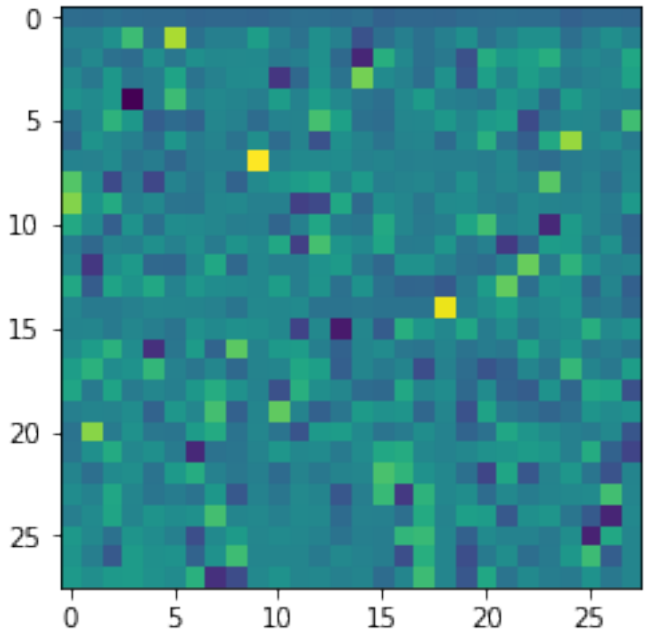
Out[88]:
```
<matplotlib.image.AxesImage at 0x14ec234e0>
```



Decryption key D3

```
In [89]: plt.imshow(D3)
```

```
Out[89]: <matplotlib.image.AxesImage at 0x14ef50320>
```



Encrypt the dataset

```
In [90]: #encrypt train set
         enc3_X_train = []
         for an_image in x_train:
             an_image = np.dot(an_image, Q3)
             enc3_X_train.append(an_image)
         print("Done encrypting ",len(enc3_X_train),"train images!")
         #encrypt test set
         enc3_X_test = []
         for an_image in x_test:
             an_image = np.dot(an_image, Q3)
             enc3_X_test.append(an_image)
         print("Done encrypting ", len(enc3_X_test), "test images!")
```

```
Done encrypting  60000 train images!
Done encrypting  10000 test images!
```

```
In [91]: enc3_X_train = np.array(enc3_X_train, dtype='float32') / 255.0
         enc3_X_test = np.array(enc3_X_test, dtype='float32') / 255.0
         enc3_X_train = enc3_X_train.reshape(-1,28,28,1)
         enc3_X_test = enc3_X_test.reshape(-1,28,28,1)
         # one hot encode target values
         enc3_y_train = to_categorical(y_train, num_classes = 10)
         enc3_y_test = to_categorical(y_test, num_classes = 10)

         print("enc3_X_train shape", enc3_X_train.shape)
         print("enc3_y_train shape", enc3_y_train.shape)
         print("enc3_X_test shape", enc3_X_test.shape)
         print("enc3_y_test shape", enc3_y_test.shape)
```

```
enc3_X_train shape (60000, 28, 28, 1)
enc3_y_train shape (60000, 10)
enc3_X_test shape (10000, 28, 28, 1)
enc3_y_test shape (10000, 10)
```

Build Convolutional Neural Networks

```
In [93]: nets = 3
         enc3_model = [0] *nets

         for j in range(3):
             enc3_model[j] = Sequential()
             enc3_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                     input_shape=(28,28,1)))
             enc3_model[j].add(MaxPool2D())
             if j>0:
```

```
            enc3_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
            enc3_model[j].add(MaxPool2D())
        if j>1:
            enc3_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
            enc3_model[j].add(MaxPool2D(padding='same'))
        enc3_model[j].add(Flatten())
        enc3_model[j].add(Dense(256, activation='relu'))
        enc3_model[j].add(Dense(10, activation='softmax'))
        enc3_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

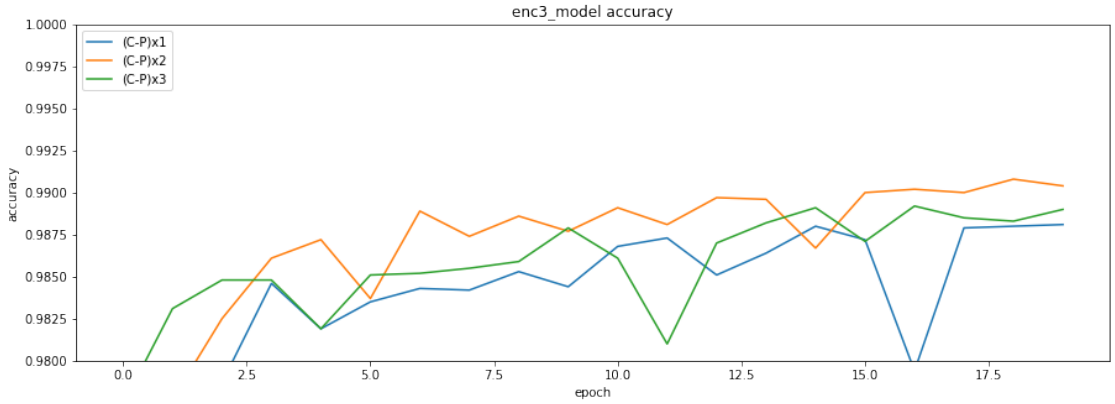
Train Convolutional Neural Networks

```
In [115]: #Decrease learning rate by 0.95 each epoch
          annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)

In [116]: # train
          enc3_history = [0] * nets
          names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
          epochs = 20
          for j in range(nets):
              enc3_history[j] = enc3_model[j].fit(enc3_X_train, enc3_y_train, batch_size=80, ep
                  validation_data = (enc3_X_test, enc3_y_test), callbacks=[annealer], verbose=(
              print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}
                  names[j],epochs,max(enc3_history[j].history['acc']),max(enc3_history[j].histo
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98810
CNN (C-P)x2: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.99080
CNN (C-P)x3: Epochs=20, Train accuracy=0.99937, Validation accuracy=0.98920
```

Plot accuracy

```
In [117]: plt.figure(figsize=(15,5))
          for i in range(nets):
              plt.plot(enc3_history[i].history['val_acc'])
          plt.title('enc3_model accuracy')
          plt.ylabel('accuracy')
          plt.xlabel('epoch')
          plt.legend(names, loc='upper left')
          axes = plt.gca()
          axes.set_ylim([0.98,1])
          plt.show()
```



Test

```
In [118]: for j in range(nets):
              enc3_test_loss, enc3_test_acc = enc3_model[j].evaluate(enc3_X_test,  enc3_y_test
              print('Acc on encrypted test dataset: ',enc3_test_acc)
```

```
Acc on encrypted test dataset:  0.9881
Acc on encrypted test dataset:  0.9904
Acc on encrypted test dataset:  0.989
```

6 Train with encrypted MNIST dataset using key 4

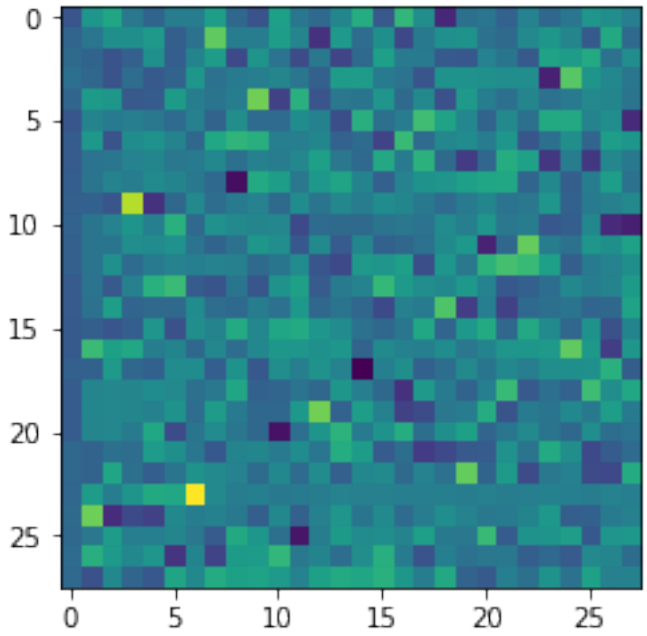
Drive orthogonal matrix to get encryption key Q4 and decryption key D4

```
In [94]: #Read key image as RGB
         key4 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key4.png'
         #Show RGB key image
         plt.imshow(key4)
         #gray scale key image
         key4 = key4.convert('L')
         #resize key image
         key4 = np.resize(key4,(28,28))
         #convert to 2D array
         key4 = np.asarray(key4)
         #QR decomposition
         Q4, R4 = scipy.linalg.qr(key4)
         #Drive the decryption matrix
         D4 = np.linalg.inv(Q4)
```

Encryption key Q4

```
In [95]: plt.imshow(Q4)
```

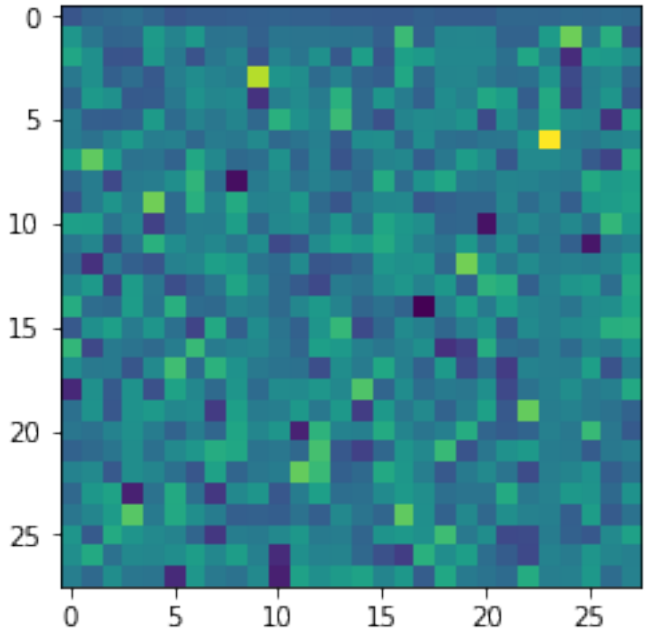
```
Out[95]: <matplotlib.image.AxesImage at 0x15256f080>
```



Decryption key D4

```
In [96]: plt.imshow(D4)
```

```
Out[96]: <matplotlib.image.AxesImage at 0x151c096a0>
```



Encrypt the dataset

```
In [97]: #encrypt train set
         enc4_X_train = []
         for an_image in x_train:
             an_image = np.dot(an_image, Q4)
             enc4_X_train.append(an_image)
         print("Done encrypting ",len(enc4_X_train),"train images!")
         #encrypt test set
         enc4_X_test = []
         for an_image in x_test:
             an_image = np.dot(an_image, Q4)
             enc4_X_test.append(an_image)
         print("Done encrypting ", len(enc4_X_test), "test images!")
```

```
Done encrypting  60000 train images!
Done encrypting  10000 test images!
```

```
In [98]: enc4_X_train = np.array(enc4_X_train, dtype='float32') / 255.0
         enc4_X_test = np.array(enc4_X_test, dtype='float32') / 255.0
```

```
        enc4_X_train = enc4_X_train.reshape(-1,28,28,1)
        enc4_X_test = enc4_X_test.reshape(-1,28,28,1)
        # one hot encode target values
        enc4_y_train = to_categorical(y_train, num_classes = 10)
        enc4_y_test = to_categorical(y_test, num_classes = 10)

        print("enc4_X_train shape", enc4_X_train.shape)
        print("enc4_y_train shape", enc4_y_train.shape)
        print("enc4_X_test shape", enc4_X_test.shape)
        print("enc4_y_test shape", enc4_y_test.shape)
```

```
enc4_X_train shape (60000, 28, 28, 1)
enc4_y_train shape (60000, 10)
enc4_X_test shape (10000, 28, 28, 1)
enc4_y_test shape (10000, 10)
```

Build Convolutional Neural Networks

```
In [99]: nets = 3
         enc4_model = [0] *nets

         for j in range(3):
             enc4_model[j] = Sequential()
             enc4_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                     input_shape=(28,28,1)))
             enc4_model[j].add(MaxPool2D())
             if j>0:
                 enc4_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
                 enc4_model[j].add(MaxPool2D())
             if j>1:
                 enc4_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
                 enc4_model[j].add(MaxPool2D(padding='same'))
             enc4_model[j].add(Flatten())
             enc4_model[j].add(Dense(256, activation='relu'))
             enc4_model[j].add(Dense(10, activation='softmax'))
             enc4_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

Train Convolutional Neural Networks

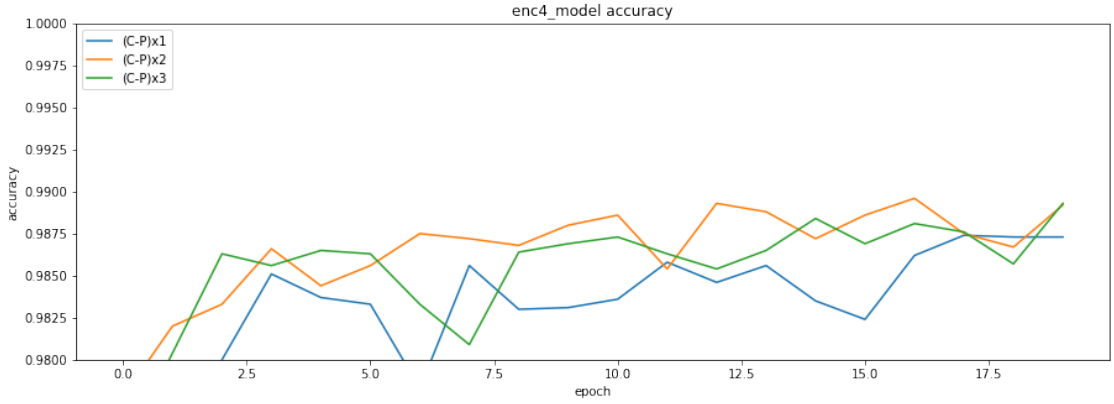
```
In [100]: #Decrease learning rate by 0.95 each epoch
          annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)

In [119]: # train
          enc4_history = [0] * nets
          names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
          epochs = 20
          for j in range(nets):
              enc4_history[j] = enc4_model[j].fit(enc4_X_train, enc4_y_train, batch_size=80, e
```

```
                validation_data = (enc4_X_test, enc4_y_test), callbacks=[annealer], verbose=
            print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f
                names[j],epochs,max(enc4_history[j].history['acc']),max(enc4_history[j].histo
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98740
CNN (C-P)x2: Epochs=20, Train accuracy=0.99990, Validation accuracy=0.98960
CNN (C-P)x3: Epochs=20, Train accuracy=0.99982, Validation accuracy=0.98930
```

Plot accuracy

```
In [120]: plt.figure(figsize=(15,5))
          for i in range(nets):
              plt.plot(enc4_history[i].history['val_acc'])
          plt.title('enc4_model accuracy')
          plt.ylabel('accuracy')
          plt.xlabel('epoch')
          plt.legend(names, loc='upper left')
          axes = plt.gca()
          axes.set_ylim([0.98,1])
          plt.show()
```



Test

```
In [121]: for j in range(nets):
              enc4_test_loss, enc4_test_acc = enc4_model[j].evaluate(enc4_X_test,  enc4_y_test
              print('Acc on encrypted test dataset: ',enc4_test_acc)
```

```
Acc on encrypted test dataset:  0.9873
Acc on encrypted test dataset:  0.9892
Acc on encrypted test dataset:  0.9893
```

7 Train with encrypted MNIST dataset using key 5

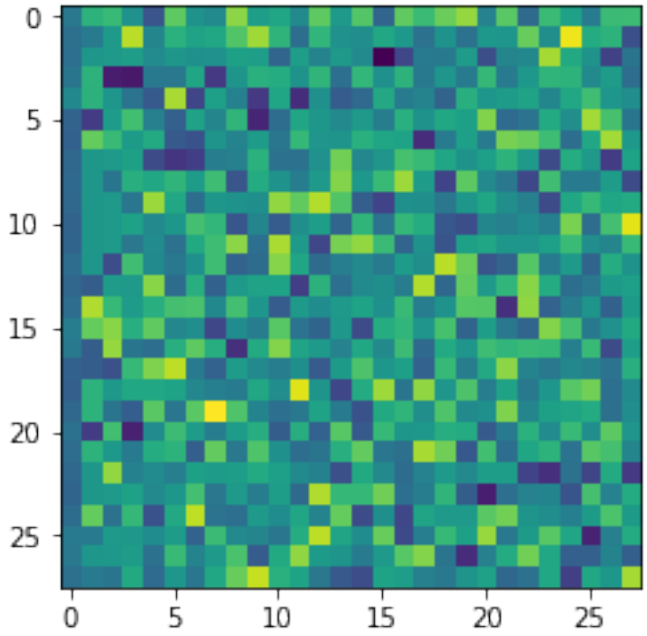
Drive orthogonal matrix to get encryption key Q5 and decryption key D5

```
In [101]: #Read key image as RGB
          key5 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key5.png
          #Show RGB key image
          plt.imshow(key5)
          #gray scale key image
          key5 = key5.convert('L')
          #resize key image
          key5 = np.resize(key5,(28,28))
          #convert to 2D array
          key5 = np.asarray(key5)
          #QR decomposition
          Q5, R5 = scipy.linalg.qr(key5)
          #Drive the decryption matrix
          D5 = np.linalg.inv(Q5)
```

Encryption key Q5

```
In [102]: plt.imshow(Q5)
```

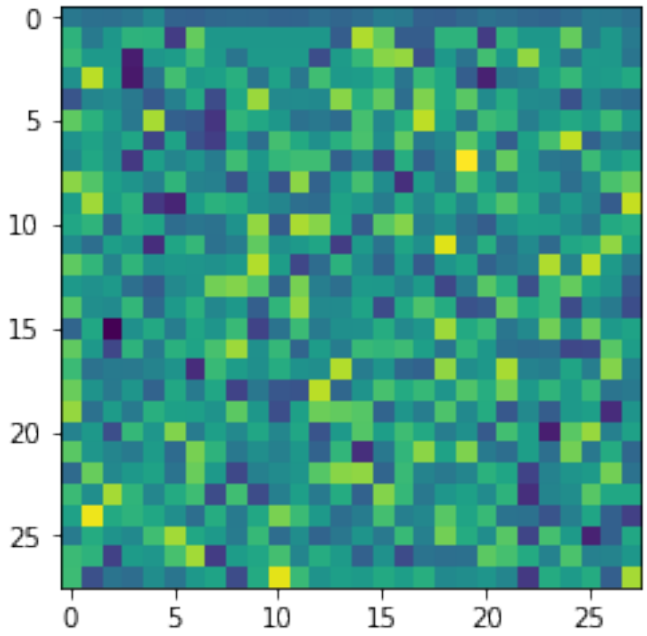
```
Out[102]: <matplotlib.image.AxesImage at 0x151da8710>
```



Decryption key D5

```
In [103]: plt.imshow(D5)
```

```
Out[103]: <matplotlib.image.AxesImage at 0x15492dd30>
```



Encrypt the dataset

```
In [104]: #encrypt train set
          enc5_X_train = []
          for an_image in x_train:
              an_image = np.dot(an_image, Q5)
              enc5_X_train.append(an_image)
          print("Done encrypting ",len(enc5_X_train),"train images!")
          #encrypt test set
          enc5_X_test = []
          for an_image in x_test:
              an_image = np.dot(an_image, Q5)
              enc5_X_test.append(an_image)
          print("Done encrypting ", len(enc5_X_test), "test images!")
```

```
Done encrypting  60000 train images!
Done encrypting  10000 test images!
```

```
In [105]: enc5_X_train = np.array(enc5_X_train, dtype='float32') / 255.0
          enc5_X_test = np.array(enc5_X_test, dtype='float32') / 255.0
          enc5_X_train = enc5_X_train.reshape(-1,28,28,1)
          enc5_X_test = enc5_X_test.reshape(-1,28,28,1)
          # one hot encode target values
          enc5_y_train = to_categorical(y_train, num_classes = 10)
          enc5_y_test = to_categorical(y_test, num_classes = 10)

          print("enc5_X_train shape", enc5_X_train.shape)
          print("enc5_y_train shape", enc5_y_train.shape)
          print("enc5_X_test shape", enc5_X_test.shape)
          print("enc5_y_test shape", enc5_y_test.shape)
```

```
enc5_X_train shape (60000, 28, 28, 1)
enc5_y_train shape (60000, 10)
enc5_X_test shape (10000, 28, 28, 1)
enc5_y_test shape (10000, 10)
```

Build Convolutional Neural Networks

```
In [106]: nets = 3
          enc5_model = [0] *nets

          for j in range(3):
              enc5_model[j] = Sequential()
              enc5_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                      input_shape=(28,28,1)))
              enc5_model[j].add(MaxPool2D())
              if j>0:
```

```
            enc5_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
            enc5_model[j].add(MaxPool2D())
        if j>1:
            enc5_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
            enc5_model[j].add(MaxPool2D(padding='same'))
        enc5_model[j].add(Flatten())
        enc5_model[j].add(Dense(256, activation='relu'))
        enc5_model[j].add(Dense(10, activation='softmax'))
        enc5_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

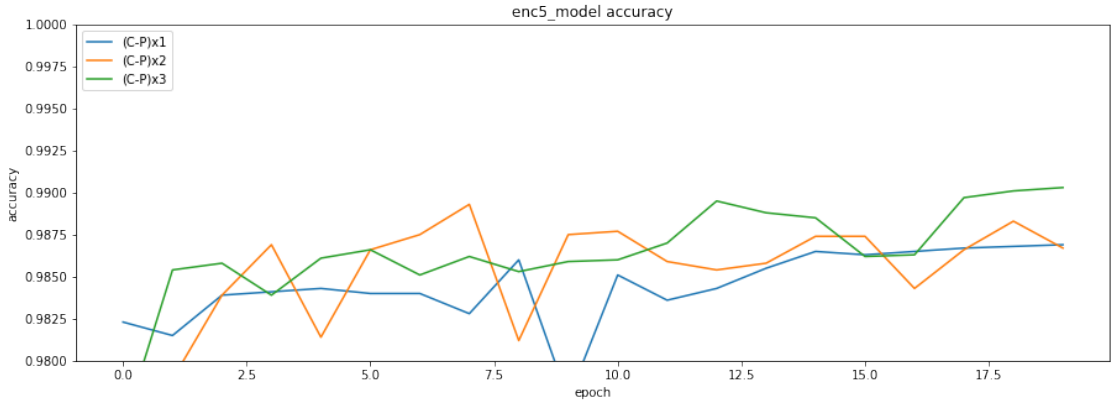
Train Convolutional Neural Networks

```
In [107]: #Decrease learning rate by 0.95 each epoch
          annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)

In [123]: # train
          enc5_history = [0] * nets
          names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
          epochs = 20
          for j in range(nets):
              enc5_history[j] = enc5_model[j].fit(enc5_X_train, enc5_y_train, batch_size=80, ep
                  validation_data = (enc5_X_test, enc5_y_test), callbacks=[annealer], verbose=(
              print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}
                  names[j],epochs,max(enc5_history[j].history['acc']),max(enc5_history[j].histo
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98690
CNN (C-P)x2: Epochs=20, Train accuracy=0.99967, Validation accuracy=0.98930
CNN (C-P)x3: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.99030
```

Plot accuracy

```
In [124]: plt.figure(figsize=(15,5))
          for i in range(nets):
              plt.plot(enc5_history[i].history['val_acc'])
          plt.title('enc5_model accuracy')
          plt.ylabel('accuracy')
          plt.xlabel('epoch')
          plt.legend(names, loc='upper left')
          axes = plt.gca()
          axes.set_ylim([0.98,1])
          plt.show()
```



Test

```
In [125]: for j in range(nets):
              enc5_test_loss, enc5_test_acc = enc5_model[j].evaluate(enc5_X_test,  enc5_y_test
              print('Acc on encrypted test dataset: ',enc5_test_acc)
```

```
Acc on encrypted test dataset:  0.9869
Acc on encrypted test dataset:  0.9867
Acc on encrypted test dataset:  0.9903
```

8 Generate Keys

#generate a random 32x32 matrices to use as the orthogonal matix for encryption.

f1 = ortho_group.rvs(dim=32) np.save("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f1", f1)

f2 = ortho_group.rvs(dim=32) np.save("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f2", f2)

f3 = ortho_group.rvs(dim=32) np.save("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f3", f3)

f4 = ortho_group.rvs(dim=32) np.save("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f4", f4)

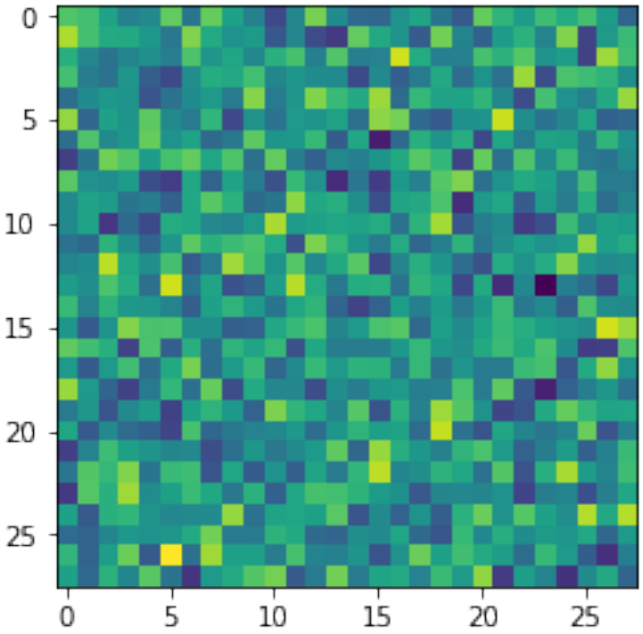
f5 = ortho_group.rvs(dim=32) np.save("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f5", f5)

9 Train with encrypted MNIST dataset using fixed key 1

```
In [3]: #load key f1
        F1 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f1.npy")
        F1 = np.resize(F1, (28,28))
        #Drive the decryption matrix from the fixed encryption key.
        DF1 = np.linalg.inv(F1)
```

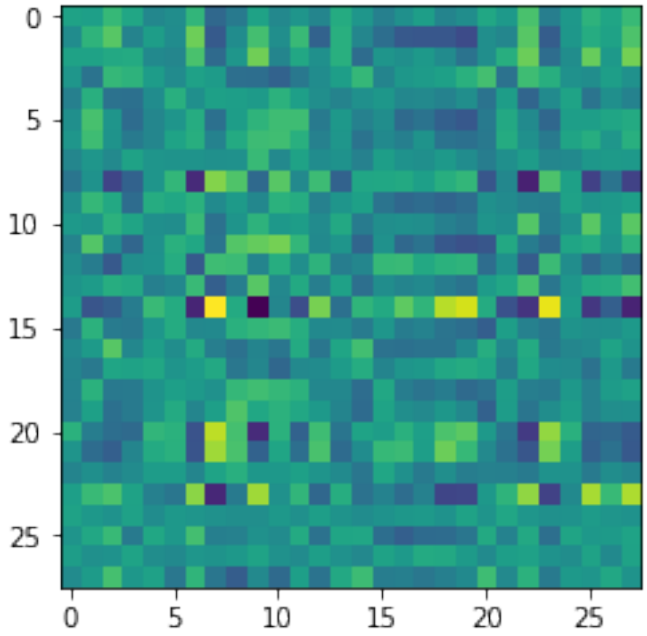
Encryption key F1

```
In [4]: plt.imshow(F1)

Out[4]: <matplotlib.image.AxesImage at 0x113293358>
```



Decryption key DF1

```
In [5]: plt.imshow(DF1)

Out[5]: <matplotlib.image.AxesImage at 0x13b10f550>
```



Encrypt the dataset

```
In [6]: #encrypt train set
        encf1_X_train = []
        for an_image in x_train:
            an_image = np.dot(an_image, F1)
            encf1_X_train.append(an_image)
        print("Done encrypting ",len(encf1_X_train),"train images with F1")
        #encrypt test set
        encf1_X_test = []
        for an_image in x_test:
            an_image = np.dot(an_image, F1)
            encf1_X_test.append(an_image)
        print("Done encrypting ", len(encf1_X_test), "test images with F1")
```

```
Done encrypting  60000 train images with F1
Done encrypting  10000 test images with F1
```

```
In [7]: # reshape encrypted dataset to have a single channel
        encf1_X_train = np.array(encf1_X_train, dtype='float32') / 255.0
        encf1_X_test = np.array(encf1_X_test, dtype='float32') / 255.0
        encf1_X_train = encf1_X_train.reshape(-1,28,28,1)
        encf1_X_test = encf1_X_test.reshape(-1,28,28,1)
        # one hot encode target values
        encf1_y_train = to_categorical(y_train, num_classes = 10)
        encf1_y_test = to_categorical(y_test, num_classes = 10)
```

```
        print("encf1_X_train shape: ", encf1_X_train.shape)
        print("encf1_X_test shape: ", encf1_X_test.shape)
        print("encf1_y_train shape: ", encf1_y_train.shape)
        print("encf1_y_test shape: ", encf1_y_test.shape)
```

```
encf1_X_train shape:  (60000, 28, 28, 1)
encf1_X_test shape:  (10000, 28, 28, 1)
encf1_y_train shape:  (60000, 10)
encf1_y_test shape:  (10000, 10)
```

Build Convolutional Neural Networks

```
In [8]: nets = 3
        encf1_model = [0] *nets

        for j in range(3):
            encf1_model[j] = Sequential()
            encf1_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                    input_shape=(28,28,1)))
            encf1_model[j].add(MaxPool2D())
            if j>0:
                encf1_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
                encf1_model[j].add(MaxPool2D())
            if j>1:
                encf1_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
                encf1_model[j].add(MaxPool2D(padding='same'))
            encf1_model[j].add(Flatten())
            encf1_model[j].add(Dense(256, activation='relu'))
            encf1_model[j].add(Dense(10, activation='softmax'))
            encf1_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

```
WARNING:tensorflow:From /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pa
Instructions for updating:
Colocations handled automatically by placer.
```

Train

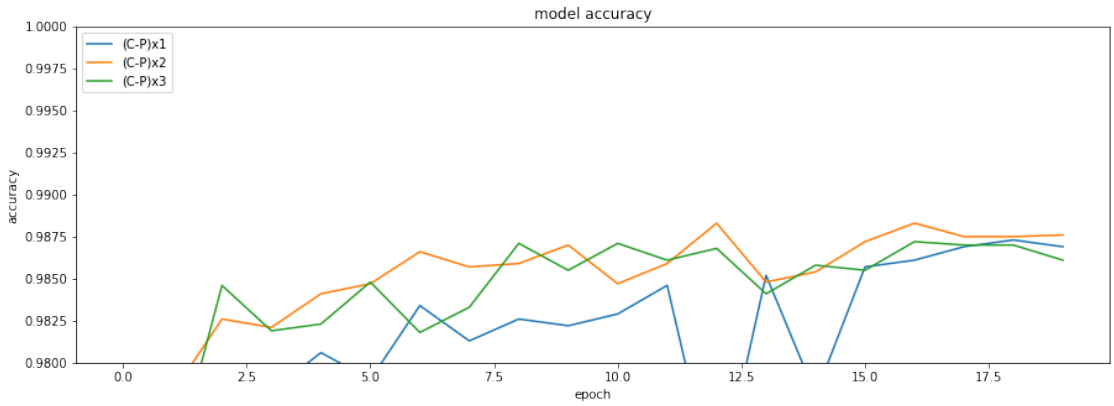
```
In [9]: #Decrease learning rate by 0.95 each epoch
        annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)
        # train
        encf1_history = [0] * nets
        names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
        epochs = 20
        for j in range(nets):
            encf1_history[j] = encf1_model[j].fit(encf1_X_train, encf1_y_train, batch_size=80,
                validation_data = (encf1_X_test, encf1_y_test), callbacks=[annealer], verbose=
```

```
            print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}"
                names[j],epochs,max(encf1_history[j].history['acc']),max(encf1_history[j].histo
```

```
WARNING:tensorflow:From /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pa
Instructions for updating:
Use tf.cast instead.
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98730
CNN (C-P)x2: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98830
CNN (C-P)x3: Epochs=20, Train accuracy=0.99960, Validation accuracy=0.98720
```

Plot accuracy

```
In [10]: # PLOT ACCURACIES
         plt.figure(figsize=(15,5))
         for i in range(nets):
             plt.plot(encf1_history[i].history['val_acc'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(names, loc='upper left')
         axes = plt.gca()
         axes.set_ylim([0.98,1])
         plt.show()
```



Test result

```
In [11]: for j in range(nets):
             encf1_test_loss, encf1_test_acc = encf1_model[j].evaluate(encf1_X_test, encf1_y_te
             print('Accuracy on test dataset: ',encf1_test_acc)
```

```
Accuracy on test dataset:  0.9869
Accuracy on test dataset:  0.9876
Accuracy on test dataset:  0.9861
```

```
In [ ]:
```

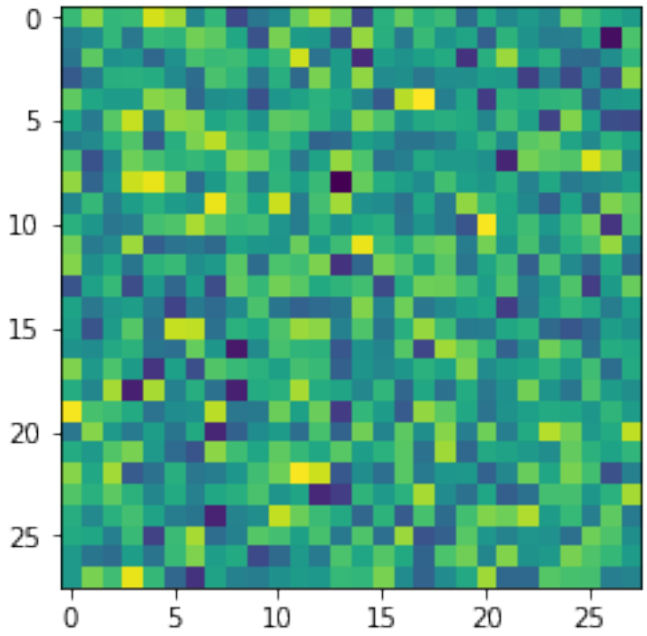
10 Train with encrypted MNIST dataset using fixed key 2

```
In [12]: #load key f2
         F2 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f2.npy")
         F2 = np.resize(F2, (28,28))
         #Drive the decryption matrix from the fixed encryption key.
         DF2 = np.linalg.inv(F2)
```

Encryption key F2

```
In [13]: plt.imshow(F2)
```

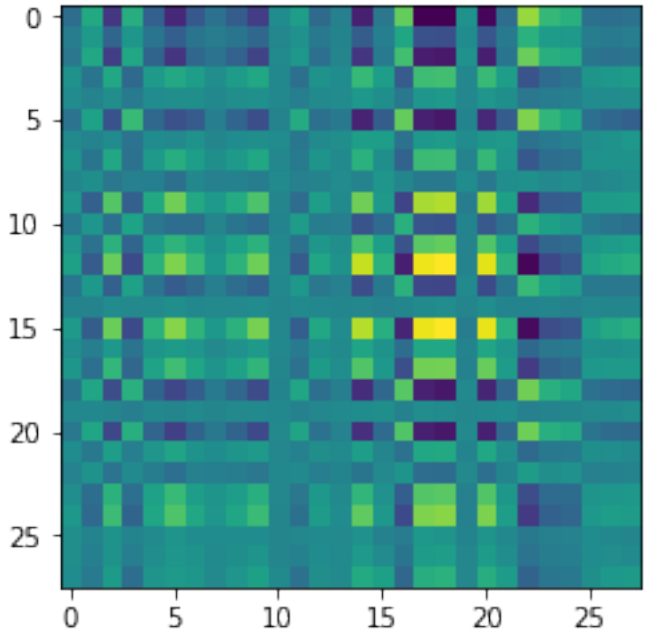
```
Out[13]: <matplotlib.image.AxesImage at 0x149eb65f8>
```



Decryption key DF2

```
In [14]: plt.imshow(DF2)
```

```
Out[14]: <matplotlib.image.AxesImage at 0x149d99240>
```



Encrypt the dataset

```
In [15]: #encrypt train set
         encf2_X_train = []
         for an_image in x_train:
             an_image = np.dot(an_image, F2)
             encf2_X_train.append(an_image)
         print("Done encrypting ",len(encf2_X_train),"train images with F2")
         #encrypt test set
         encf2_X_test = []
         for an_image in x_test:
             an_image = np.dot(an_image, F2)
             encf2_X_test.append(an_image)
         print("Done encrypting ", len(encf2_X_test), "test images with F2")
```

```
Done encrypting  60000 train images with F2
Done encrypting  10000 test images with F2
```

```
In [16]: # reshape encrypted dataset to have a single channel
         encf2_X_train = np.array(encf2_X_train, dtype='float32') / 255.0
         encf2_X_test = np.array(encf2_X_test, dtype='float32') / 255.0
         encf2_X_train = encf2_X_train.reshape(-1,28,28,1)
         encf2_X_test = encf2_X_test.reshape(-1,28,28,1)
         # one hot encode target values
         encf2_y_train = to_categorical(y_train, num_classes = 10)
         encf2_y_test = to_categorical(y_test, num_classes = 10)
```

```
        print("encf2_X_train shape: ", encf2_X_train.shape)
        print("encf2_X_test shape: ", encf2_X_test.shape)
        print("encf2_y_train shape: ", encf2_y_train.shape)
        print("encf2_y_test shape: ", encf2_y_test.shape)
```

```
encf2_X_train shape:  (60000, 28, 28, 1)
encf2_X_test shape:  (10000, 28, 28, 1)
encf2_y_train shape:  (60000, 10)
encf2_y_test shape:  (10000, 10)
```

Build Convolutional Neural Networks

```
In [17]: nets = 3
         encf2_model = [0] *nets

         for j in range(3):
             encf2_model[j] = Sequential()
             encf2_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                     input_shape=(28,28,1)))
             encf2_model[j].add(MaxPool2D())
             if j>0:
                 encf2_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
                 encf2_model[j].add(MaxPool2D())
             if j>1:
                 encf2_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
                 encf2_model[j].add(MaxPool2D(padding='same'))
             encf2_model[j].add(Flatten())
             encf2_model[j].add(Dense(256, activation='relu'))
             encf2_model[j].add(Dense(10, activation='softmax'))
             encf2_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

Train

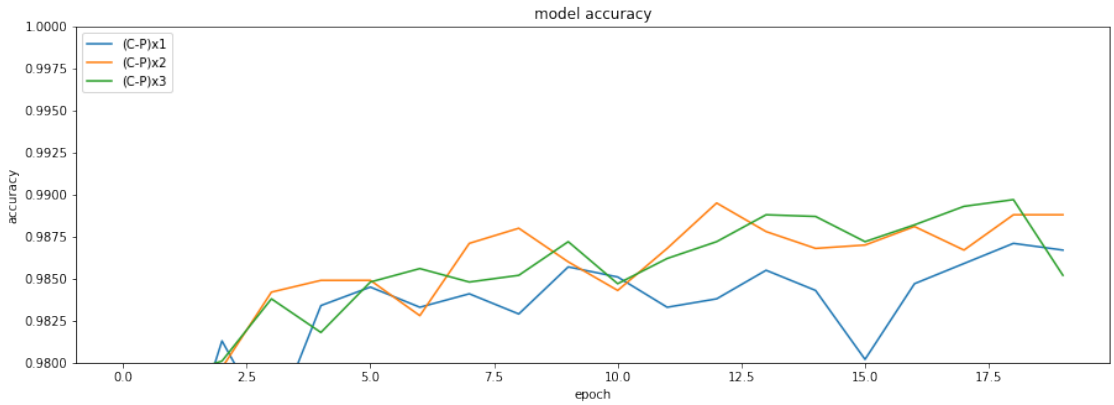
```
In [18]: #Decrease learning rate by 0.95 each epoch
         annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)
         # train
         encf2_history = [0] * nets
         names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
         epochs = 20
         for j in range(nets):
             encf2_history[j] = encf2_model[j].fit(encf2_X_train, encf2_y_train, batch_size=80
                 validation_data = (encf2_X_test, encf2_y_test), callbacks=[annealer], verbose=
             print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}"
                 names[j],epochs,max(encf2_history[j].history['acc']),max(encf2_history[j].hist
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98710
CNN (C-P)x2: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98950
```

```
CNN (C-P)x3: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98970
```

Plot accuracy

```
In [19]: # PLOT ACCURACIES
         plt.figure(figsize=(15,5))
         for i in range(nets):
             plt.plot(encf2_history[i].history['val_acc'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(names, loc='upper left')
         axes = plt.gca()
         axes.set_ylim([0.98,1])
         plt.show()
```



Test result

```
In [20]: for j in range(nets):
             encf2_test_loss, encf2_test_acc = encf2_model[j].evaluate(encf2_X_test, encf2_y_te
             print('Accuracy on test dataset: ',encf2_test_acc)
```

```
Accuracy on test dataset:  0.9867
Accuracy on test dataset:  0.9888
Accuracy on test dataset:  0.9852
```

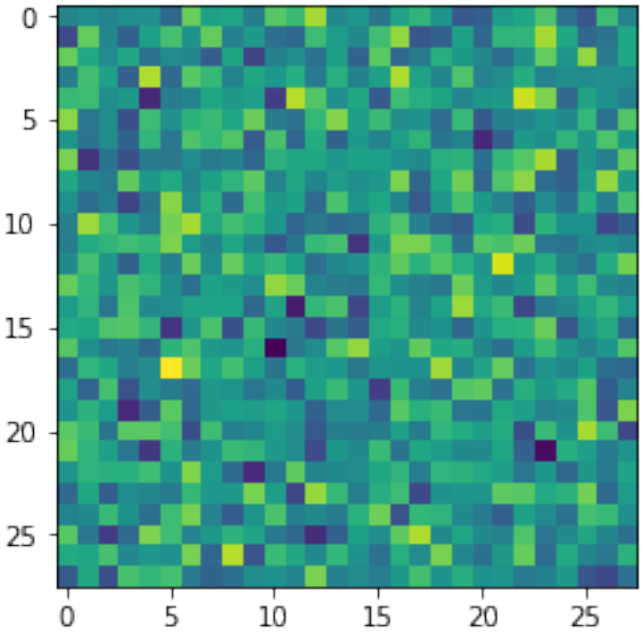
11 Train with encrypted MNIST dataset using fixed key 3

```
In [21]: #load key f3
         F3 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f3.npy")
         F3 = np.resize(F3, (28,28))
         #Drive the decryption matrix from the fixed encryption key.
         DF3 = np.linalg.inv(F3)
```

Encryption key F3

```
In [22]: plt.imshow(F3)
```

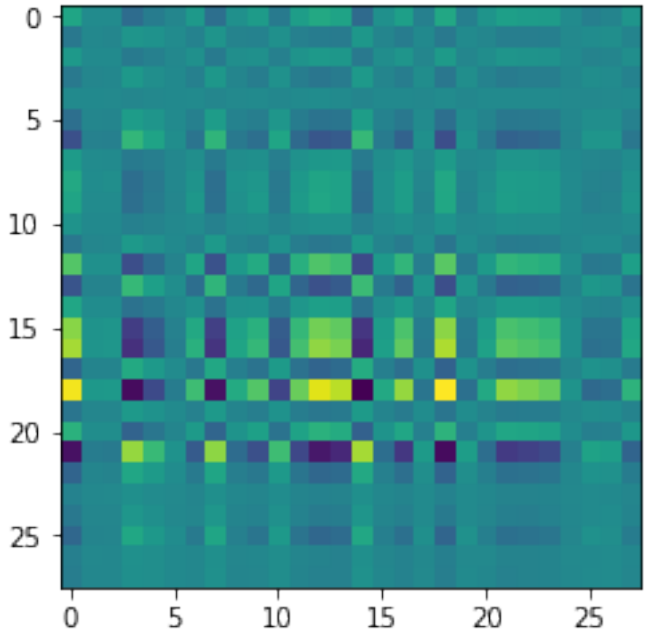
```
Out[22]: <matplotlib.image.AxesImage at 0x150bd39b0>
```



Decryption key DF3

```
In [23]: plt.imshow(DF3)
```

```
Out[23]: <matplotlib.image.AxesImage at 0x14d901828>
```



Encrypt the dataset

```
In [24]: #encrypt train set
         encf3_X_train = []
         for an_image in x_train:
             an_image = np.dot(an_image, F3)
             encf3_X_train.append(an_image)
         print("Done encrypting ",len(encf3_X_train),"train images with F3")
         #encrypt test set
         encf3_X_test = []
         for an_image in x_test:
             an_image = np.dot(an_image, F3)
             encf3_X_test.append(an_image)
         print("Done encrypting ", len(encf3_X_test), "test images with F3")
```

```
Done encrypting  60000 train images with F3
Done encrypting  10000 test images with F3
```

```
In [25]: # reshape encrypted dataset to have a single channel
         encf3_X_train = np.array(encf3_X_train, dtype='float32') / 255.0
         encf3_X_test = np.array(encf3_X_test, dtype='float32') / 255.0
         encf3_X_train = encf3_X_train.reshape(-1,28,28,1)
         encf3_X_test = encf3_X_test.reshape(-1,28,28,1)
         # one hot encode target values
         encf3_y_train = to_categorical(y_train, num_classes = 10)
         encf3_y_test = to_categorical(y_test, num_classes = 10)
```

```
        print("encf3_X_train shape: ", encf3_X_train.shape)
        print("encf3_X_test shape: ", encf3_X_test.shape)
        print("encf3_y_train shape: ", encf3_y_train.shape)
        print("encf3_y_test shape: ", encf3_y_test.shape)
```

```
encf3_X_train shape:  (60000, 28, 28, 1)
encf3_X_test shape:  (10000, 28, 28, 1)
encf3_y_train shape:  (60000, 10)
encf3_y_test shape:  (10000, 10)
```

Build Convolutional Neural Networks

```
In [26]: nets = 3
         encf3_model = [0] *nets

         for j in range(3):
             encf3_model[j] = Sequential()
             encf3_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                     input_shape=(28,28,1)))
             encf3_model[j].add(MaxPool2D())
             if j>0:
                 encf3_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
                 encf3_model[j].add(MaxPool2D())
             if j>1:
                 encf3_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
                 encf3_model[j].add(MaxPool2D(padding='same'))
             encf3_model[j].add(Flatten())
             encf3_model[j].add(Dense(256, activation='relu'))
             encf3_model[j].add(Dense(10, activation='softmax'))
             encf3_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

Train

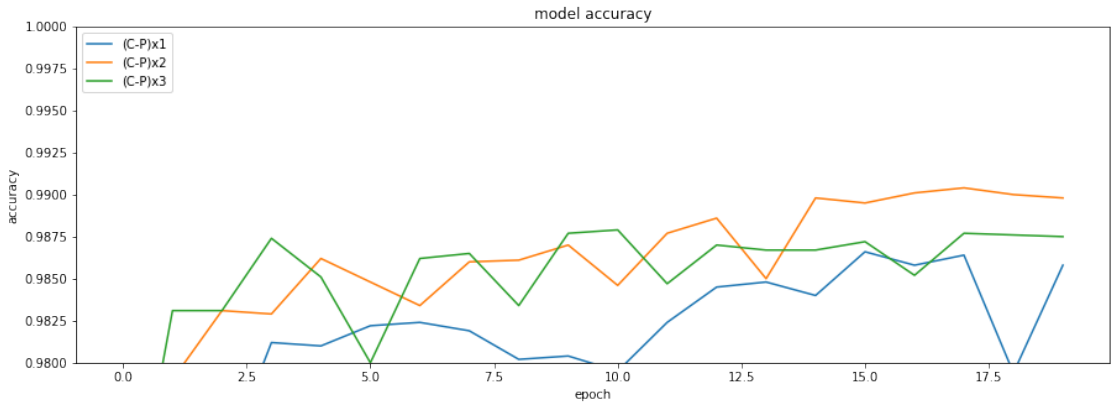
```
In [27]: #Decrease learning rate by 0.95 each epoch
         annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)
         # train
         encf3_history = [0] * nets
         names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
         epochs = 20
         for j in range(nets):
             encf3_history[j] = encf3_model[j].fit(encf3_X_train, encf3_y_train, batch_size=80
                 validation_data = (encf3_X_test, encf3_y_test), callbacks=[annealer], verbose=
             print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}"
                 names[j],epochs,max(encf3_history[j].history['acc']),max(encf3_history[j].hist
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99997, Validation accuracy=0.98660
CNN (C-P)x2: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.99040
```

```
CNN (C-P)x3: Epochs=20, Train accuracy=0.99983, Validation accuracy=0.98790
```

Plot accuracy

```
In [28]: # PLOT ACCURACIES
         plt.figure(figsize=(15,5))
         for i in range(nets):
             plt.plot(encf3_history[i].history['val_acc'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(names, loc='upper left')
         axes = plt.gca()
         axes.set_ylim([0.98,1])
         plt.show()
```



Test result

```
In [29]: for j in range(nets):
             encf3_test_loss, encf3_test_acc = encf3_model[j].evaluate(encf3_X_test, encf3_y_te
             print('Accuracy on test dataset: ',encf3_test_acc)
```

```
Accuracy on test dataset:  0.9858
Accuracy on test dataset:  0.9898
Accuracy on test dataset:  0.9875
```

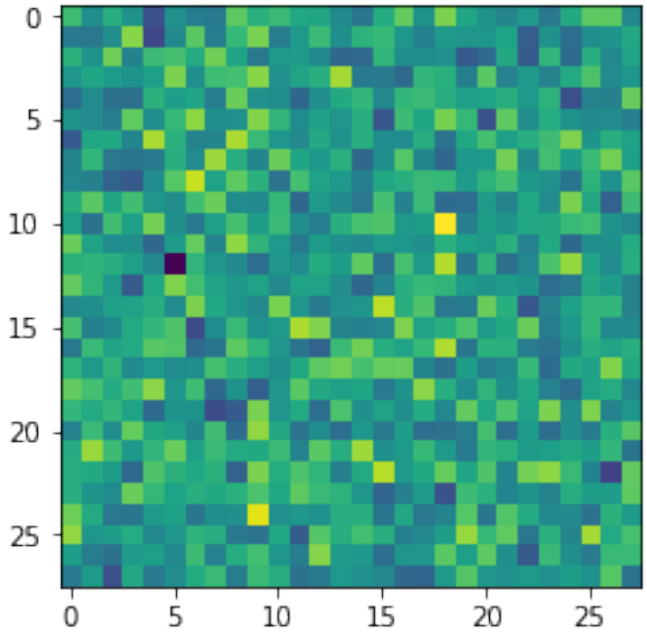
12 Train with encrypted MNIST dataset using fixed key 4

```
In [30]: #load key f4
         F4 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f4.npy")
         F4 = np.resize(F4, (28,28))
         #Drive the decryption matrix from the fixed encryption key.
         DF4 = np.linalg.inv(F4)
```

Encryption key F4

```
In [31]: plt.imshow(F4)
```

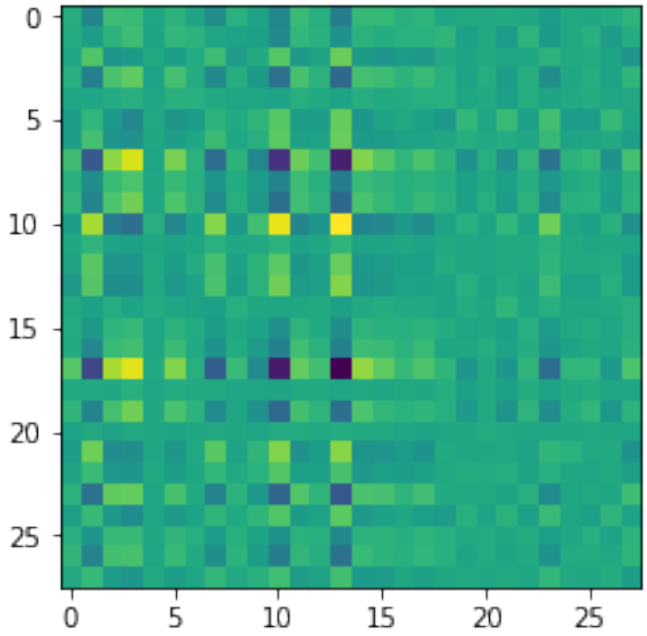
```
Out[31]: <matplotlib.image.AxesImage at 0x160cc5da0>
```



Decryption key DF4

```
In [32]: plt.imshow(DF4)
```

```
Out[32]: <matplotlib.image.AxesImage at 0x1618e2ba8>
```



Encrypt the dataset

```
In [33]: #encrypt train set
         encf4_X_train = []
         for an_image in x_train:
             an_image = np.dot(an_image, F4)
             encf4_X_train.append(an_image)
         print("Done encrypting ",len(encf4_X_train),"train images with F4")
         #encrypt test set
         encf4_X_test = []
         for an_image in x_test:
             an_image = np.dot(an_image, F4)
             encf4_X_test.append(an_image)
         print("Done encrypting ", len(encf4_X_test), "test images with F4")
```

```
Done encrypting  60000 train images with F4
Done encrypting  10000 test images with F4
```

```
In [34]: # reshape encrypted dataset to have a single channel
         encf4_X_train = np.array(encf4_X_train, dtype='float32') / 255.0
         encf4_X_test = np.array(encf4_X_test, dtype='float32') / 255.0
         encf4_X_train = encf4_X_train.reshape(-1,28,28,1)
         encf4_X_test = encf4_X_test.reshape(-1,28,28,1)
         # one hot encode target values
         encf4_y_train = to_categorical(y_train, num_classes = 10)
         encf4_y_test = to_categorical(y_test, num_classes = 10)
```

```
        print("encf4_X_train shape: ", encf4_X_train.shape)
        print("encf4_X_test shape: ", encf4_X_test.shape)
        print("encf4_y_train shape: ", encf4_y_train.shape)
        print("encf4_y_test shape: ", encf4_y_test.shape)
```

```
encf4_X_train shape:  (60000, 28, 28, 1)
encf4_X_test shape:  (10000, 28, 28, 1)
encf4_y_train shape:  (60000, 10)
encf4_y_test shape:  (10000, 10)
```

Build Convolutional Neural Networks

```
In [35]: nets = 3
         encf4_model = [0] *nets

         for j in range(3):
             encf4_model[j] = Sequential()
             encf4_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                     input_shape=(28,28,1)))
             encf4_model[j].add(MaxPool2D())
             if j>0:
                 encf4_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
                 encf4_model[j].add(MaxPool2D())
             if j>1:
                 encf4_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
                 encf4_model[j].add(MaxPool2D(padding='same'))
             encf4_model[j].add(Flatten())
             encf4_model[j].add(Dense(256, activation='relu'))
             encf4_model[j].add(Dense(10, activation='softmax'))
             encf4_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

Train

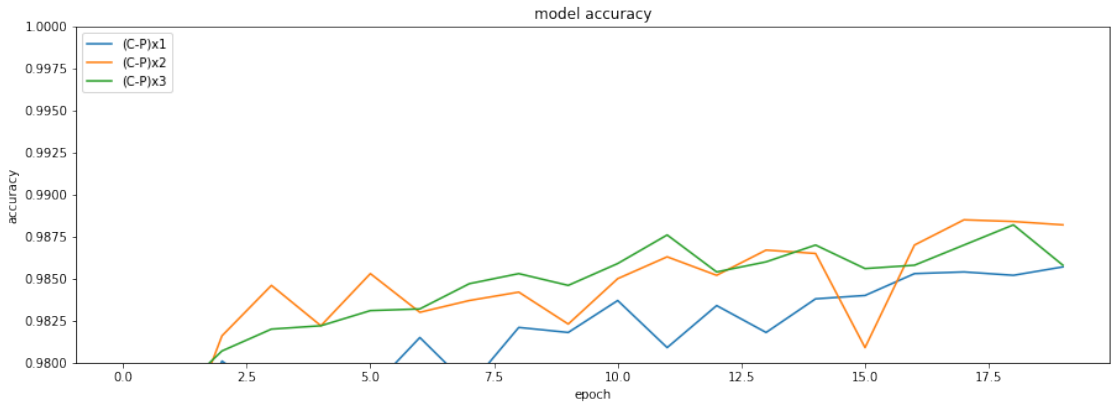
```
In [36]: #Decrease learning rate by 0.95 each epoch
         annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)
         # train
         encf4_history = [0] * nets
         names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
         epochs = 20
         for j in range(nets):
             encf4_history[j] = encf4_model[j].fit(encf4_X_train, encf4_y_train, batch_size=80
                 validation_data = (encf4_X_test, encf4_y_test), callbacks=[annealer], verbose=
             print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}"
                 names[j],epochs,max(encf4_history[j].history['acc']),max(encf4_history[j].hist
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98570
CNN (C-P)x2: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98850
```

```
CNN (C-P)x3: Epochs=20, Train accuracy=0.99942, Validation accuracy=0.98820
```

Plot accuracy

```
In [37]: # PLOT ACCURACIES
         plt.figure(figsize=(15,5))
         for i in range(nets):
             plt.plot(encf4_history[i].history['val_acc'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(names, loc='upper left')
         axes = plt.gca()
         axes.set_ylim([0.98,1])
         plt.show()
```



Test result

```
In [38]: for j in range(nets):
             encf4_test_loss, encf4_test_acc = encf4_model[j].evaluate(encf4_X_test, encf4_y_te
             print('Accuracy on test dataset: ',encf4_test_acc)
```

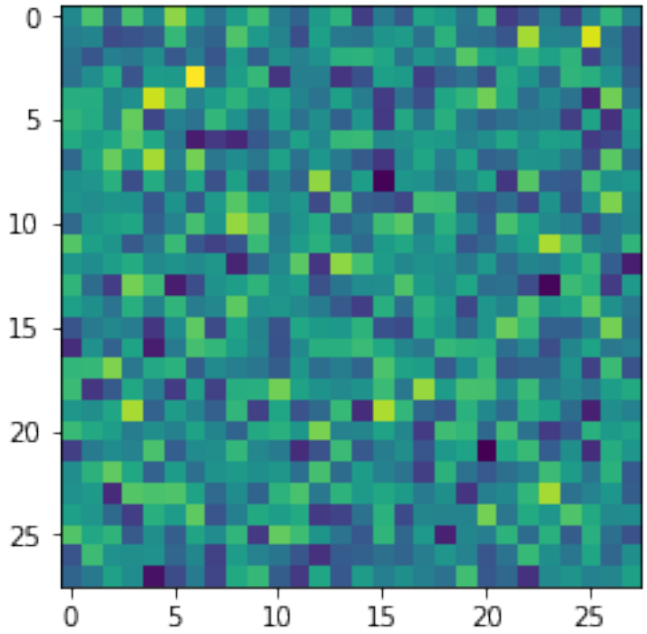
```
Accuracy on test dataset:  0.9857
Accuracy on test dataset:  0.9882
Accuracy on test dataset:  0.9858
```

13 Train with encrypted MNIST dataset using fixed key 5

```
In [39]: #load key f5
         F5 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f5.npy")
         F5 = np.resize(F5, (28,28))
         #Drive the decryption matrix from the fixed encryption key.
         DF5 = np.linalg.inv(F5)
```

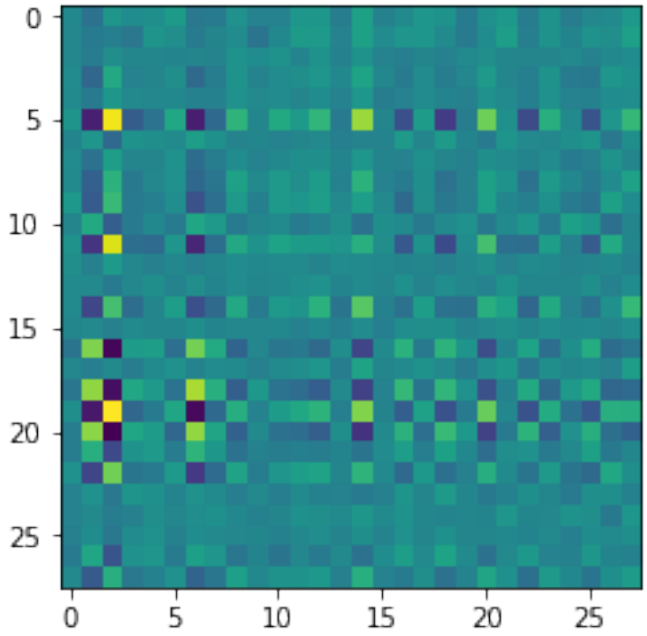
Encryption key F5

```
In [40]: plt.imshow(F5)

Out[40]: <matplotlib.image.AxesImage at 0x16465dba8>
```



Decryption key DF5

```
In [41]: plt.imshow(DF5)

Out[41]: <matplotlib.image.AxesImage at 0x165101208>
```



Encrypt the dataset

```
In [42]: #encrypt train set
         encf5_X_train = []
         for an_image in x_train:
             an_image = np.dot(an_image, F5)
             encf5_X_train.append(an_image)
         print("Done encrypting ",len(encf5_X_train),"train images with F5")
         #encrypt test set
         encf5_X_test = []
         for an_image in x_test:
             an_image = np.dot(an_image, F5)
             encf5_X_test.append(an_image)
         print("Done encrypting ", len(encf5_X_test), "test images with F5")
```

```
Done encrypting  60000 train images with F5
Done encrypting  10000 test images with F5
```

```
In [43]: # reshape encrypted dataset to have a single channel
         encf5_X_train = np.array(encf5_X_train, dtype='float32') / 255.0
         encf5_X_test = np.array(encf5_X_test, dtype='float32') / 255.0
         encf5_X_train = encf5_X_train.reshape(-1,28,28,1)
         encf5_X_test = encf5_X_test.reshape(-1,28,28,1)
         # one hot encode target values
         encf5_y_train = to_categorical(y_train, num_classes = 10)
         encf5_y_test = to_categorical(y_test, num_classes = 10)
```

```
        print("encf5_X_train shape: ", encf5_X_train.shape)
        print("encf5_X_test shape: ", encf5_X_test.shape)
        print("encf5_y_train shape: ", encf5_y_train.shape)
        print("encf5_y_test shape: ", encf5_y_test.shape)
```

```
encf5_X_train shape:  (60000, 28, 28, 1)
encf5_X_test shape:  (10000, 28, 28, 1)
encf5_y_train shape:  (60000, 10)
encf5_y_test shape:  (10000, 10)
```

Build Convolutional Neural Networks

```
In [44]: nets = 3
         encf5_model = [0] *nets

         for j in range(3):
             encf5_model[j] = Sequential()
             encf5_model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
                     input_shape=(28,28,1)))
             encf5_model[j].add(MaxPool2D())
             if j>0:
                 encf5_model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
                 encf5_model[j].add(MaxPool2D())
             if j>1:
                 encf5_model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
                 encf5_model[j].add(MaxPool2D(padding='same'))
             encf5_model[j].add(Flatten())
             encf5_model[j].add(Dense(256, activation='relu'))
             encf5_model[j].add(Dense(10, activation='softmax'))
             encf5_model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=
```

Train

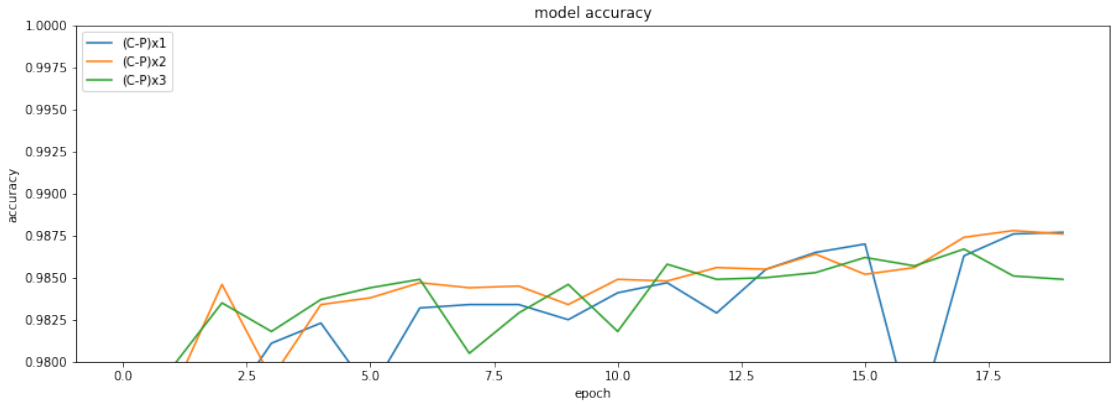
```
In [45]: #Decrease learning rate by 0.95 each epoch
         annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x)
         # train
         encf5_history = [0] * nets
         names = ["(C-P)x1","(C-P)x2","(C-P)x3"]
         epochs = 20
         for j in range(nets):
             encf5_history[j] = encf5_model[j].fit(encf5_X_train, encf5_y_train, batch_size=80
                 validation_data = (encf5_X_test, encf5_y_test), callbacks=[annealer], verbose=
             print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accuracy={3:.5f}"
                 names[j],epochs,max(encf5_history[j].history['acc']),max(encf5_history[j].hist
```

```
CNN (C-P)x1: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98770
CNN (C-P)x2: Epochs=20, Train accuracy=0.99998, Validation accuracy=0.98780
```

```
CNN (C-P)x3: Epochs=20, Train accuracy=0.99988, Validation accuracy=0.98670
```

Plot accuracy

```
In [46]: # PLOT ACCURACIES
         plt.figure(figsize=(15,5))
         for i in range(nets):
             plt.plot(encf5_history[i].history['val_acc'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(names, loc='upper left')
         axes = plt.gca()
         axes.set_ylim([0.98,1])
         plt.show()
```



Test result

```
In [47]: for j in range(nets):
             encf5_test_loss, encf5_test_acc = encf5_model[j].evaluate(encf5_X_test, encf5_y_te
             print('Accuracy on test dataset: ',encf5_test_acc)
```

```
Accuracy on test dataset:  0.9877
Accuracy on test dataset:  0.9876
Accuracy on test dataset:  0.9849
```

https://www.kaggle.com/cdeotte/how-to-choose-cnn-architecture-mnist

jupyter nbconvert –to PDF /Users/thienngole/Desktop/MINES/3Fall2020/MS-project/CNN_MNIST.ipynb

Appendix_B

December 2, 2020

[351]:
```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import itertools
import cv2
import glob
import os
%matplotlib inline
from PIL import Image
import scipy
```

[352]:
```
batch_size = 32  # The default batch size of keras.
num_classes = 10  # Number of class for the dataset
epochs = 100
data_augmentation = False
```

[353]:
```
# Load CIFAR10 data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print("Done loading data!")
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
Done loading data!
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
50000 train samples
```

```
10000 test samples
```

1 Train with clear CIFAR10 dataset

[354]:
```
# Normalize the data(connvert data type to float for computation).
X_train = x_train.astype('float32')
X_test = x_test.astype('float32')
X_train /= 255
X_test /= 255

# Convert class vectors to binary class matrices(one hot encoding).
Y_train = keras.utils.to_categorical(y_train, num_classes)
Y_test = keras.utils.to_categorical(y_test, num_classes)
```

[355]:
```
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print('x_test shape:', x_test.shape)
print('y_test shape:', y_test.shape)
```

```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

[356]:
```
#define the convnet
model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
model.add(Conv2D(32, (3, 3), padding='same',input_shape = x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# FLATTERN => DENSE => RELU => DROPOUT
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
```

```
# a softmax classifier
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

[357]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Compile
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[358]:
```
#Train model
history = model.fit(X_train, Y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(X_test, Y_test),
              shuffle=True)
```

```
Not using data augmentation.
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 265s 5ms/step - loss: 1.8464 -
acc: 0.3226 - val_loss: 1.5902 - val_acc: 0.4258
Epoch 2/100
50000/50000 [==============================] - 247s 5ms/step - loss: 1.5059 -
acc: 0.4541 - val_loss: 1.4189 - val_acc: 0.4905
Epoch 3/100
50000/50000 [==============================] - 229s 5ms/step - loss: 1.3541 -
acc: 0.5157 - val_loss: 1.2569 - val_acc: 0.5545
Epoch 4/100
50000/50000 [==============================] - 222s 4ms/step - loss: 1.2514 -
acc: 0.5561 - val_loss: 1.1224 - val_acc: 0.6034
Epoch 5/100
50000/50000 [==============================] - 222s 4ms/step - loss: 1.1706 -
acc: 0.5877 - val_loss: 1.0791 - val_acc: 0.6144
Epoch 6/100
50000/50000 [==============================] - 217s 4ms/step - loss: 1.1001 -
acc: 0.6151 - val_loss: 1.0259 - val_acc: 0.6349
Epoch 7/100
50000/50000 [==============================] - 217s 4ms/step - loss: 1.0469 -
acc: 0.6320 - val_loss: 0.9857 - val_acc: 0.6532
Epoch 8/100
50000/50000 [==============================] - 217s 4ms/step - loss: 1.0044 -
acc: 0.6475 - val_loss: 0.9287 - val_acc: 0.6776
Epoch 9/100
50000/50000 [==============================] - 215s 4ms/step - loss: 0.9667 -
```

```
acc: 0.6621 - val_loss: 0.9682 - val_acc: 0.6651
Epoch 10/100
50000/50000 [==============================] - 231s 5ms/step - loss: 0.9345 -
acc: 0.6734 - val_loss: 0.8970 - val_acc: 0.6915
Epoch 11/100
50000/50000 [==============================] - 255s 5ms/step - loss: 0.9004 -
acc: 0.6861 - val_loss: 0.8652 - val_acc: 0.7005
Epoch 12/100
50000/50000 [==============================] - 261s 5ms/step - loss: 0.8793 -
acc: 0.6927 - val_loss: 0.8467 - val_acc: 0.7032
Epoch 13/100
50000/50000 [==============================] - 226s 5ms/step - loss: 0.8617 -
acc: 0.6982 - val_loss: 0.8783 - val_acc: 0.7035
Epoch 14/100
50000/50000 [==============================] - 156s 3ms/step - loss: 0.8415 -
acc: 0.7062 - val_loss: 0.8114 - val_acc: 0.7175
Epoch 15/100
50000/50000 [==============================] - 148s 3ms/step - loss: 0.8211 -
acc: 0.7150 - val_loss: 0.8205 - val_acc: 0.7161
Epoch 16/100
50000/50000 [==============================] - 171s 3ms/step - loss: 0.8101 -
acc: 0.7205 - val_loss: 0.7708 - val_acc: 0.7356
Epoch 17/100
50000/50000 [==============================] - 174s 3ms/step - loss: 0.7983 -
acc: 0.7228 - val_loss: 0.7693 - val_acc: 0.7346
Epoch 18/100
50000/50000 [==============================] - 169s 3ms/step - loss: 0.7871 -
acc: 0.7292 - val_loss: 0.7576 - val_acc: 0.7406
Epoch 19/100
50000/50000 [==============================] - 191s 4ms/step - loss: 0.7767 -
acc: 0.7327 - val_loss: 0.7685 - val_acc: 0.7376
Epoch 20/100
50000/50000 [==============================] - 164s 3ms/step - loss: 0.7673 -
acc: 0.7337 - val_loss: 0.7402 - val_acc: 0.7463
Epoch 21/100
50000/50000 [==============================] - 169s 3ms/step - loss: 0.7592 -
acc: 0.7406 - val_loss: 0.7445 - val_acc: 0.7451
Epoch 22/100
50000/50000 [==============================] - 166s 3ms/step - loss: 0.7497 -
acc: 0.7435 - val_loss: 0.7368 - val_acc: 0.7511
Epoch 23/100
50000/50000 [==============================] - 169s 3ms/step - loss: 0.7480 -
acc: 0.7474 - val_loss: 0.7328 - val_acc: 0.7561
Epoch 24/100
50000/50000 [==============================] - 163s 3ms/step - loss: 0.7460 -
acc: 0.7447 - val_loss: 0.7480 - val_acc: 0.7449
Epoch 25/100
50000/50000 [==============================] - 169s 3ms/step - loss: 0.7349 -
```

```
acc: 0.7500 - val_loss: 0.7597 - val_acc: 0.7451
Epoch 26/100
50000/50000 [==============================] - 153s 3ms/step - loss: 0.7293 -
acc: 0.7539 - val_loss: 0.7091 - val_acc: 0.7587
Epoch 27/100
50000/50000 [==============================] - 165s 3ms/step - loss: 0.7263 -
acc: 0.7525 - val_loss: 0.7110 - val_acc: 0.7622
Epoch 28/100
50000/50000 [==============================] - 172s 3ms/step - loss: 0.7234 -
acc: 0.7553 - val_loss: 0.7248 - val_acc: 0.7539
Epoch 29/100
50000/50000 [==============================] - 170s 3ms/step - loss: 0.7155 -
acc: 0.7578 - val_loss: 0.7297 - val_acc: 0.7506
Epoch 30/100
50000/50000 [==============================] - 161s 3ms/step - loss: 0.7118 -
acc: 0.7579 - val_loss: 0.7017 - val_acc: 0.7712
Epoch 31/100
50000/50000 [==============================] - 183s 4ms/step - loss: 0.7084 -
acc: 0.7592 - val_loss: 0.6869 - val_acc: 0.7692
Epoch 32/100
50000/50000 [==============================] - 180s 4ms/step - loss: 0.7076 -
acc: 0.7616 - val_loss: 0.6996 - val_acc: 0.7673
Epoch 33/100
50000/50000 [==============================] - 187s 4ms/step - loss: 0.7026 -
acc: 0.7636 - val_loss: 0.6973 - val_acc: 0.7669
Epoch 34/100
50000/50000 [==============================] - 198s 4ms/step - loss: 0.7041 -
acc: 0.7633 - val_loss: 0.7239 - val_acc: 0.7732
Epoch 35/100
50000/50000 [==============================] - 217s 4ms/step - loss: 0.6953 -
acc: 0.7660 - val_loss: 0.6771 - val_acc: 0.7695
Epoch 36/100
50000/50000 [==============================] - 182s 4ms/step - loss: 0.6912 -
acc: 0.7659 - val_loss: 0.6835 - val_acc: 0.7711
Epoch 37/100
50000/50000 [==============================] - 203s 4ms/step - loss: 0.6844 -
acc: 0.7679 - val_loss: 0.6924 - val_acc: 0.7702
Epoch 38/100
50000/50000 [==============================] - 191s 4ms/step - loss: 0.6851 -
acc: 0.7673 - val_loss: 0.6747 - val_acc: 0.7732
Epoch 39/100
50000/50000 [==============================] - 254s 5ms/step - loss: 0.6850 -
acc: 0.7716 - val_loss: 0.7194 - val_acc: 0.7646
Epoch 40/100
50000/50000 [==============================] - 213s 4ms/step - loss: 0.6791 -
acc: 0.7730 - val_loss: 0.6767 - val_acc: 0.7751
Epoch 41/100
50000/50000 [==============================] - 226s 5ms/step - loss: 0.6762 -
```

```
acc: 0.7728 - val_loss: 0.7372 - val_acc: 0.7628
Epoch 42/100
50000/50000 [==============================] - 202s 4ms/step - loss: 0.6697 -
acc: 0.7757 - val_loss: 0.7150 - val_acc: 0.7619
Epoch 43/100
50000/50000 [==============================] - 227s 5ms/step - loss: 0.6615 -
acc: 0.7778 - val_loss: 0.6743 - val_acc: 0.7759
Epoch 44/100
50000/50000 [==============================] - 168s 3ms/step - loss: 0.6698 -
acc: 0.7757 - val_loss: 0.6565 - val_acc: 0.7793
Epoch 45/100
50000/50000 [==============================] - 174s 3ms/step - loss: 0.6629 -
acc: 0.7783 - val_loss: 0.6755 - val_acc: 0.7760
Epoch 46/100
50000/50000 [==============================] - 173s 3ms/step - loss: 0.6666 -
acc: 0.7775 - val_loss: 0.6669 - val_acc: 0.7751
Epoch 47/100
50000/50000 [==============================] - 187s 4ms/step - loss: 0.6638 -
acc: 0.7762 - val_loss: 0.6995 - val_acc: 0.7730
Epoch 48/100
50000/50000 [==============================] - 193s 4ms/step - loss: 0.6624 -
acc: 0.7781 - val_loss: 0.6934 - val_acc: 0.7806
Epoch 49/100
50000/50000 [==============================] - 175s 3ms/step - loss: 0.6601 -
acc: 0.7778 - val_loss: 0.7271 - val_acc: 0.7603
Epoch 50/100
50000/50000 [==============================] - 184s 4ms/step - loss: 0.6568 -
acc: 0.7807 - val_loss: 0.6951 - val_acc: 0.7789
Epoch 51/100
50000/50000 [==============================] - 186s 4ms/step - loss: 0.6575 -
acc: 0.7821 - val_loss: 0.7437 - val_acc: 0.7759
Epoch 52/100
50000/50000 [==============================] - 227s 5ms/step - loss: 0.6538 -
acc: 0.7812 - val_loss: 0.6431 - val_acc: 0.7831
Epoch 53/100
50000/50000 [==============================] - 194s 4ms/step - loss: 0.6461 -
acc: 0.7821 - val_loss: 0.6991 - val_acc: 0.7701
Epoch 54/100
50000/50000 [==============================] - 175s 4ms/step - loss: 0.6526 -
acc: 0.7815 - val_loss: 0.6721 - val_acc: 0.7835
Epoch 55/100
50000/50000 [==============================] - 162s 3ms/step - loss: 0.6524 -
acc: 0.7835 - val_loss: 0.7175 - val_acc: 0.7584
Epoch 56/100
50000/50000 [==============================] - 162s 3ms/step - loss: 0.6515 -
acc: 0.7841 - val_loss: 0.6738 - val_acc: 0.7781
Epoch 57/100
50000/50000 [==============================] - 169s 3ms/step - loss: 0.6499 -
```

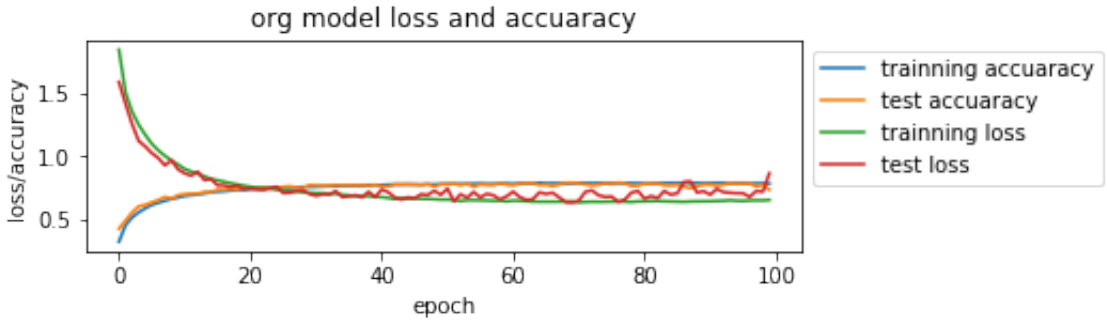
```
acc: 0.7821 - val_loss: 0.7027 - val_acc: 0.7730
Epoch 58/100
50000/50000 [==============================] - 173s 3ms/step - loss: 0.6477 -
acc: 0.7828 - val_loss: 0.6909 - val_acc: 0.7714
Epoch 59/100
50000/50000 [==============================] - 188s 4ms/step - loss: 0.6485 -
acc: 0.7843 - val_loss: 0.6580 - val_acc: 0.7868
Epoch 60/100
50000/50000 [==============================] - 179s 4ms/step - loss: 0.6430 -
acc: 0.7849 - val_loss: 0.6823 - val_acc: 0.7744
Epoch 61/100
50000/50000 [==============================] - 181s 4ms/step - loss: 0.6486 -
acc: 0.7824 - val_loss: 0.7201 - val_acc: 0.7704
Epoch 62/100
50000/50000 [==============================] - 185s 4ms/step - loss: 0.6480 -
acc: 0.7862 - val_loss: 0.6771 - val_acc: 0.7824
Epoch 63/100
50000/50000 [==============================] - 195s 4ms/step - loss: 0.6446 -
acc: 0.7858 - val_loss: 0.6551 - val_acc: 0.7861
Epoch 64/100
50000/50000 [==============================] - 207s 4ms/step - loss: 0.6413 -
acc: 0.7858 - val_loss: 0.6596 - val_acc: 0.7866
Epoch 65/100
50000/50000 [==============================] - 250s 5ms/step - loss: 0.6402 -
acc: 0.7898 - val_loss: 0.7143 - val_acc: 0.7721
Epoch 66/100
50000/50000 [==============================] - 216s 4ms/step - loss: 0.6403 -
acc: 0.7862 - val_loss: 0.7013 - val_acc: 0.7745
Epoch 67/100
50000/50000 [==============================] - 167s 3ms/step - loss: 0.6403 -
acc: 0.7870 - val_loss: 0.7111 - val_acc: 0.7641
Epoch 68/100
50000/50000 [==============================] - 158s 3ms/step - loss: 0.6409 -
acc: 0.7858 - val_loss: 0.6749 - val_acc: 0.7753
Epoch 69/100
50000/50000 [==============================] - 147s 3ms/step - loss: 0.6423 -
acc: 0.7852 - val_loss: 0.6366 - val_acc: 0.7907
Epoch 70/100
50000/50000 [==============================] - 145s 3ms/step - loss: 0.6435 -
acc: 0.7854 - val_loss: 0.6331 - val_acc: 0.7914
Epoch 71/100
50000/50000 [==============================] - 144s 3ms/step - loss: 0.6352 -
acc: 0.7901 - val_loss: 0.6487 - val_acc: 0.7865
Epoch 72/100
50000/50000 [==============================] - 143s 3ms/step - loss: 0.6380 -
acc: 0.7877 - val_loss: 0.7182 - val_acc: 0.7774
Epoch 73/100
50000/50000 [==============================] - 144s 3ms/step - loss: 0.6398 -
```

```
acc: 0.7873 - val_loss: 0.7300 - val_acc: 0.7695
Epoch 74/100
50000/50000 [==============================] - 146s 3ms/step - loss: 0.6393 -
acc: 0.7877 - val_loss: 0.6914 - val_acc: 0.7741
Epoch 75/100
50000/50000 [==============================] - 143s 3ms/step - loss: 0.6398 -
acc: 0.7881 - val_loss: 0.6779 - val_acc: 0.7864
Epoch 76/100
50000/50000 [==============================] - 143s 3ms/step - loss: 0.6411 -
acc: 0.7863 - val_loss: 0.6813 - val_acc: 0.7804
Epoch 77/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6421 -
acc: 0.7876 - val_loss: 0.6384 - val_acc: 0.7921
Epoch 78/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6410 -
acc: 0.7884 - val_loss: 0.6566 - val_acc: 0.7866
Epoch 79/100
50000/50000 [==============================] - 143s 3ms/step - loss: 0.6381 -
acc: 0.7871 - val_loss: 0.7120 - val_acc: 0.7647
Epoch 80/100
50000/50000 [==============================] - 143s 3ms/step - loss: 0.6429 -
acc: 0.7869 - val_loss: 0.7257 - val_acc: 0.7672
Epoch 81/100
50000/50000 [==============================] - 143s 3ms/step - loss: 0.6431 -
acc: 0.7847 - val_loss: 0.6647 - val_acc: 0.7825
Epoch 82/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6461 -
acc: 0.7840 - val_loss: 0.6849 - val_acc: 0.7811
Epoch 83/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6463 -
acc: 0.7869 - val_loss: 0.6692 - val_acc: 0.7844
Epoch 84/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6439 -
acc: 0.7877 - val_loss: 0.7253 - val_acc: 0.7702
Epoch 85/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6421 -
acc: 0.7885 - val_loss: 0.6964 - val_acc: 0.7689
Epoch 86/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6408 -
acc: 0.7876 - val_loss: 0.7068 - val_acc: 0.7740
Epoch 87/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6394 -
acc: 0.7894 - val_loss: 0.7973 - val_acc: 0.7584
Epoch 88/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6421 -
acc: 0.7881 - val_loss: 0.8010 - val_acc: 0.7481
Epoch 89/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6431 -
```

```
acc: 0.7884 - val_loss: 0.7127 - val_acc: 0.7647
Epoch 90/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6439 -
acc: 0.7864 - val_loss: 0.7233 - val_acc: 0.7742
Epoch 91/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6439 -
acc: 0.7871 - val_loss: 0.6962 - val_acc: 0.7769
Epoch 92/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6463 -
acc: 0.7885 - val_loss: 0.7408 - val_acc: 0.7649
Epoch 93/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6475 -
acc: 0.7884 - val_loss: 0.7162 - val_acc: 0.7750
Epoch 94/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.6513 -
acc: 0.7849 - val_loss: 0.7114 - val_acc: 0.7858
Epoch 95/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6479 -
acc: 0.7865 - val_loss: 0.7074 - val_acc: 0.7805
Epoch 96/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6465 -
acc: 0.7881 - val_loss: 0.7072 - val_acc: 0.7850
Epoch 97/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6502 -
acc: 0.7842 - val_loss: 0.6779 - val_acc: 0.7822
Epoch 98/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6508 -
acc: 0.7861 - val_loss: 0.7256 - val_acc: 0.7621
Epoch 99/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.6505 -
acc: 0.7863 - val_loss: 0.7190 - val_acc: 0.7742
Epoch 100/100
50000/50000 [==============================] - 140s 3ms/step - loss: 0.6542 -
acc: 0.7844 - val_loss: 0.8658 - val_acc: 0.7351
```

[359]:
```python
def acc_loss_graphic(model_history, title):
    fig = plt.figure()
    plt.subplot(2,1,2)
    plt.plot(model_history.history['acc'], label = 'trainning accuaracy')
    plt.plot(model_history.history['val_acc'], label = 'test accuaracy')
    plt.plot(model_history.history['loss'], label = 'trainning loss')
    plt.plot(model_history.history['val_loss'], label = 'test loss')
    plt.title(title)
    plt.ylabel('loss/accuracy')
    plt.xlabel('epoch')
    plt.legend(bbox_to_anchor=(1,1), loc='upper left', ncol =1)
```

[360]:
```
acc_loss_graphic(history, 'org model loss and accuaracy')
```



Test

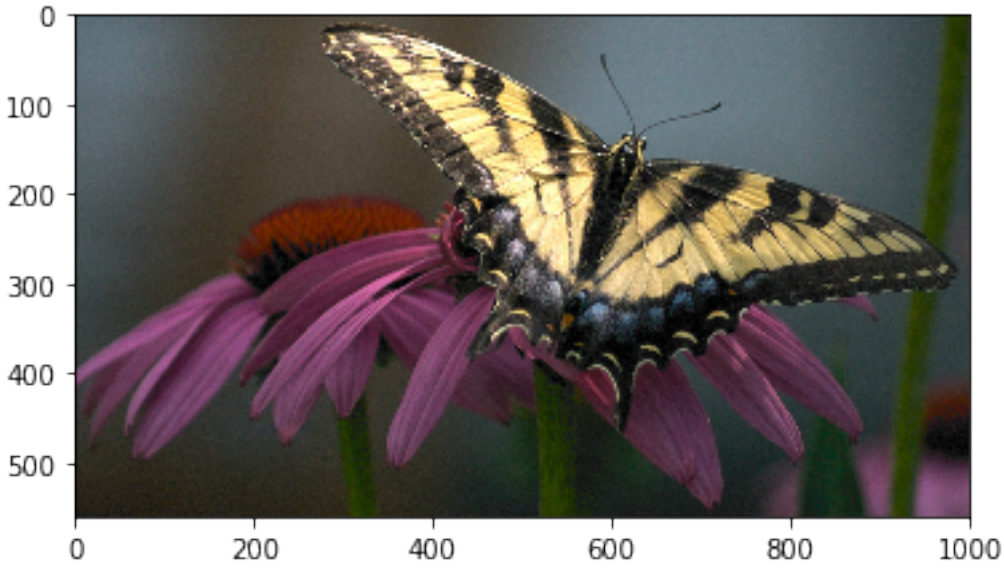
[361]:
```
# Score trained model.
scores = model.evaluate(X_test, Y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# make prediction.
pred = model.predict(x_test)
```

```
10000/10000 [==============================] - 6s 627us/step
Test loss: 0.8657635468482971
Test accuracy: 0.7351
```

2 Train with encrypted CIFAR10 dataset using key 1

Drive orthogonal matrix to get encryption key Q1 and decryption key D1

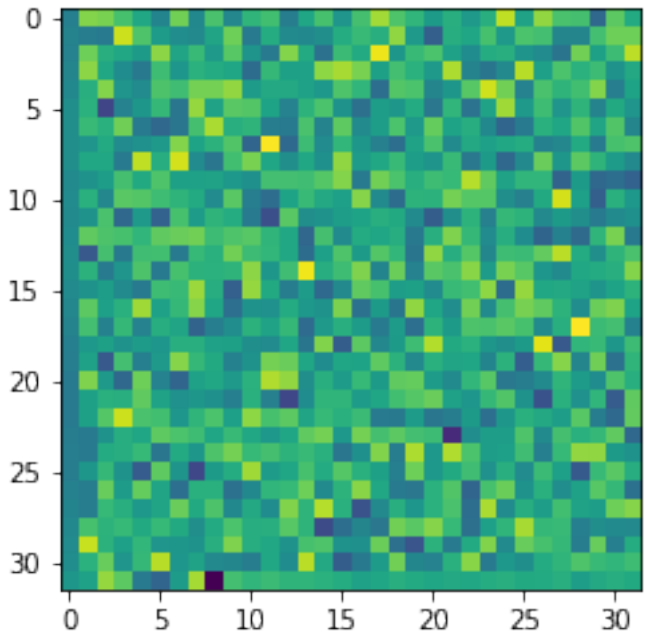
[455]:
```
#Read key image as RGB
key1 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/
 ↪key1.png")
#Show RGB key image
plt.imshow(key1)
#gray scale key image
key1 = key1.convert('L')
#resize key image
key1 = np.resize(key1,(32,32))
#convert to 2D array
key1 = np.asarray(key1)
#QR decomposition
Q1, R1 = scipy.linalg.qr(key1)
#Drive the decryption matrix
D1 = np.linalg.inv(Q1)
```



Encryption key Q1

[456]:
```
plt.imshow(Q1)
```

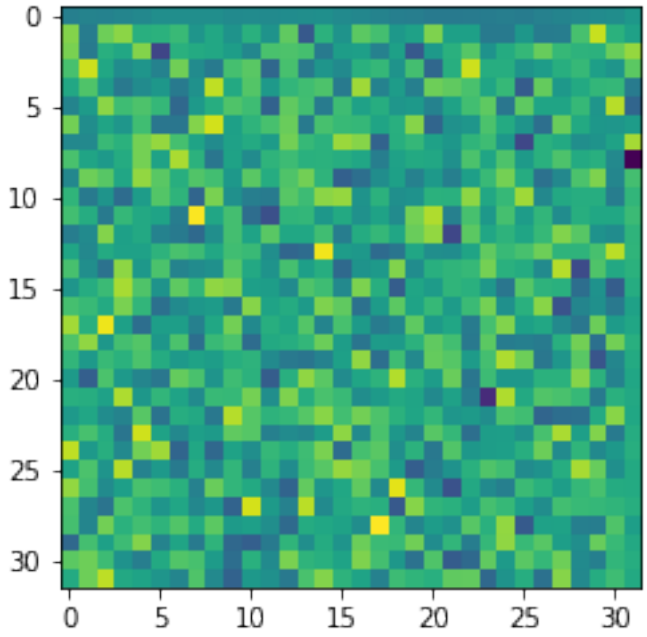
[456]:
```
<matplotlib.image.AxesImage at 0x18c448208>
```



Decryption key D1

[457]:
```
plt.imshow(D1)
```

[457]:
```
<matplotlib.image.AxesImage at 0x18c3e3668>
```



Encrypt the dataset

[458]:
```
enc1_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    enc1_b = np.dot(b, Q1)
    enc1_g = np.dot(g, Q1)
    enc1_r = np.dot(r, Q1)
    enc1_rgb = cv2.merge((enc1_b,enc1_g,enc1_r))
    enc1_X_train.append(enc1_rgb)
print("Done encrypting ",len(enc1_X_train),"train images!")
enc1_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    enc1_b = np.dot(b, Q1)
    enc1_g = np.dot(g, Q1)
    enc1_r = np.dot(r, Q1)
    enc1_rgb = cv2.merge((enc1_b,enc1_g,enc1_r))
    enc1_X_test.append(enc1_rgb)
print("Done encrypting ",len(enc1_X_test),"test images!")
```

```
Done encrypting  50000 train images!
```

```
Done encrypting  10000 test images!
```

[459]:
```
# Normalize the data.
enc1_X_train = np.array(enc1_X_train)
enc1_X_test = np.array(enc1_X_test)

enc1_X_train = enc1_X_train.astype('float32')
enc1_X_test = enc1_X_test.astype('float32')
enc1_X_train /= 255
enc1_X_test /= 255

# Convert class vectors to binary class matrices(one hot encoding).
enc1_y_train = keras.utils.to_categorical(y_train, num_classes)
enc1_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[460]:
```
print('enc1_x_train shape:', enc1_X_train.shape)
print('enc1_y_train shape:', enc1_y_train.shape)
print('enc1_x_test shape:', enc1_X_test.shape)
print('enc1_y_test shape:', enc1_y_test.shape)
```

```
enc1_x_train shape: (50000, 32, 32, 3)
enc1_y_train shape: (50000, 10)
enc1_x_test shape: (10000, 32, 32, 3)
enc1_y_test shape: (10000, 10)
```

[461]:
```
#define the convnet
enc1_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc1_model.add(Conv2D(32, (3, 3), padding='same',input_shape = enc1_X_train.
 ↪shape[1:]))
enc1_model.add(Activation('relu'))
enc1_model.add(Conv2D(32, (3, 3)))
enc1_model.add(Activation('relu'))
enc1_model.add(MaxPooling2D(pool_size=(2, 2)))
enc1_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc1_model.add(Conv2D(64, (3, 3), padding='same'))
enc1_model.add(Activation('relu'))
enc1_model.add(Conv2D(64, (3, 3)))
enc1_model.add(Activation('relu'))
enc1_model.add(MaxPooling2D(pool_size=(2, 2)))
enc1_model.add(Dropout(0.25))

# FLATTERN => DENSE => RELU => DROPOUT
enc1_model.add(Flatten())
enc1_model.add(Dense(512))
```

```
enc1_model.add(Activation('relu'))
enc1_model.add(Dropout(0.5))
# a softmax classifier
enc1_model.add(Dense(num_classes))
enc1_model.add(Activation('softmax'))
```

[369]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Compile
enc1_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[370]:
```
#Train model
enc1_history = enc1_model.fit(enc1_X_train, enc1_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(enc1_X_test, enc1_y_test),
              shuffle=True)
```

```
Not using data augmentation.
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.9419 -
acc: 0.2947 - val_loss: 1.6684 - val_acc: 0.4102
Epoch 2/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.6294 -
acc: 0.4162 - val_loss: 1.5950 - val_acc: 0.4337
Epoch 3/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.5209 -
acc: 0.4537 - val_loss: 1.4021 - val_acc: 0.4977
Epoch 4/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.4482 -
acc: 0.4833 - val_loss: 1.3579 - val_acc: 0.5105
Epoch 5/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3926 -
acc: 0.5039 - val_loss: 1.3353 - val_acc: 0.5193
Epoch 6/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3477 -
acc: 0.5206 - val_loss: 1.2702 - val_acc: 0.5494
Epoch 7/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3134 -
acc: 0.5336 - val_loss: 1.2272 - val_acc: 0.5592
Epoch 8/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.2801 -
acc: 0.5477 - val_loss: 1.2435 - val_acc: 0.5547
```

```
Epoch 9/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.2549 -
acc: 0.5550 - val_loss: 1.2146 - val_acc: 0.5657
Epoch 10/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.2301 -
acc: 0.5660 - val_loss: 1.1731 - val_acc: 0.5831
Epoch 11/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.2067 -
acc: 0.5738 - val_loss: 1.1506 - val_acc: 0.5908
Epoch 12/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1953 -
acc: 0.5808 - val_loss: 1.1481 - val_acc: 0.5959
Epoch 13/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1782 -
acc: 0.5867 - val_loss: 1.1306 - val_acc: 0.5986
Epoch 14/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1622 -
acc: 0.5901 - val_loss: 1.1453 - val_acc: 0.5961
Epoch 15/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1477 -
acc: 0.5949 - val_loss: 1.1129 - val_acc: 0.6037
Epoch 16/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.1376 -
acc: 0.5994 - val_loss: 1.1069 - val_acc: 0.6049
Epoch 17/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.1281 -
acc: 0.6043 - val_loss: 1.1095 - val_acc: 0.6019
Epoch 18/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1205 -
acc: 0.6064 - val_loss: 1.1045 - val_acc: 0.6114
Epoch 19/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1072 -
acc: 0.6109 - val_loss: 1.1143 - val_acc: 0.6015
Epoch 20/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0992 -
acc: 0.6118 - val_loss: 1.1024 - val_acc: 0.6153
Epoch 21/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0935 -
acc: 0.6190 - val_loss: 1.0753 - val_acc: 0.6209
Epoch 22/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0900 -
acc: 0.6181 - val_loss: 1.0563 - val_acc: 0.6246
Epoch 23/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0782 -
acc: 0.6206 - val_loss: 1.0555 - val_acc: 0.6250
Epoch 24/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0767 -
acc: 0.6227 - val_loss: 1.0736 - val_acc: 0.6192
```

```
Epoch 25/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0698 -
acc: 0.6260 - val_loss: 1.0980 - val_acc: 0.6189
Epoch 26/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0671 -
acc: 0.6276 - val_loss: 1.0462 - val_acc: 0.6279
Epoch 27/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0587 -
acc: 0.6293 - val_loss: 1.0628 - val_acc: 0.6261
Epoch 28/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0569 -
acc: 0.6325 - val_loss: 1.0658 - val_acc: 0.6240
Epoch 29/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0524 -
acc: 0.6320 - val_loss: 1.0684 - val_acc: 0.6293
Epoch 30/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0463 -
acc: 0.6349 - val_loss: 1.0663 - val_acc: 0.6270
Epoch 31/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0465 -
acc: 0.6352 - val_loss: 1.1218 - val_acc: 0.6304
Epoch 32/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0455 -
acc: 0.6363 - val_loss: 1.0490 - val_acc: 0.6288
Epoch 33/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0429 -
acc: 0.6343 - val_loss: 1.0714 - val_acc: 0.6302
Epoch 34/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0436 -
acc: 0.6355 - val_loss: 1.0531 - val_acc: 0.6317
Epoch 35/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0365 -
acc: 0.6374 - val_loss: 1.0505 - val_acc: 0.6321
Epoch 36/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0335 -
acc: 0.6424 - val_loss: 1.0654 - val_acc: 0.6305
Epoch 37/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0346 -
acc: 0.6391 - val_loss: 1.0337 - val_acc: 0.6363
Epoch 38/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0342 -
acc: 0.6402 - val_loss: 1.0697 - val_acc: 0.6336
Epoch 39/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0296 -
acc: 0.6398 - val_loss: 1.0612 - val_acc: 0.6358
Epoch 40/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0282 -
acc: 0.6410 - val_loss: 1.0950 - val_acc: 0.6333
```

```
Epoch 41/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0269 -
acc: 0.6453 - val_loss: 1.0533 - val_acc: 0.6340
Epoch 42/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0259 -
acc: 0.6452 - val_loss: 1.0817 - val_acc: 0.6290
Epoch 43/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0225 -
acc: 0.6440 - val_loss: 1.2600 - val_acc: 0.6270
Epoch 44/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.0225 -
acc: 0.6466 - val_loss: 1.0829 - val_acc: 0.6303
Epoch 45/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0187 -
acc: 0.6477 - val_loss: 1.0740 - val_acc: 0.6382
Epoch 46/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0266 -
acc: 0.6445 - val_loss: 1.0825 - val_acc: 0.6334
Epoch 47/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.0216 -
acc: 0.6454 - val_loss: 1.1385 - val_acc: 0.6338
Epoch 48/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0198 -
acc: 0.6446 - val_loss: 1.0622 - val_acc: 0.6409
Epoch 49/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.0178 -
acc: 0.6456 - val_loss: 1.0381 - val_acc: 0.6375
Epoch 50/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.0185 -
acc: 0.6486 - val_loss: 1.0358 - val_acc: 0.6344
Epoch 51/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0174 -
acc: 0.6464 - val_loss: 1.0815 - val_acc: 0.6379
Epoch 52/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.0173 -
acc: 0.6463 - val_loss: 1.1172 - val_acc: 0.6356
Epoch 53/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0122 -
acc: 0.6491 - val_loss: 1.0884 - val_acc: 0.6343
Epoch 54/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0169 -
acc: 0.6487 - val_loss: 1.1539 - val_acc: 0.6325
Epoch 55/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0153 -
acc: 0.6470 - val_loss: 1.0750 - val_acc: 0.6376
Epoch 56/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0097 -
acc: 0.6508 - val_loss: 1.0785 - val_acc: 0.6244
```

```
Epoch 57/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0160 -
acc: 0.6468 - val_loss: 1.0711 - val_acc: 0.6295
Epoch 58/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0137 -
acc: 0.6501 - val_loss: 1.0998 - val_acc: 0.6301
Epoch 59/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0120 -
acc: 0.6512 - val_loss: 1.0797 - val_acc: 0.6258
Epoch 60/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0129 -
acc: 0.6522 - val_loss: 1.0994 - val_acc: 0.6318
Epoch 61/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0143 -
acc: 0.6499 - val_loss: 1.1611 - val_acc: 0.6302
Epoch 62/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0131 -
acc: 0.6507 - val_loss: 1.1573 - val_acc: 0.6284
Epoch 63/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0100 -
acc: 0.6494 - val_loss: 1.0418 - val_acc: 0.6432
Epoch 64/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0078 -
acc: 0.6515 - val_loss: 1.0374 - val_acc: 0.6406
Epoch 65/100
50000/50000 [==============================] - 179s 4ms/step - loss: 1.0099 -
acc: 0.6528 - val_loss: 1.1400 - val_acc: 0.6278
Epoch 66/100
50000/50000 [==============================] - 162s 3ms/step - loss: 1.0096 -
acc: 0.6510 - val_loss: 1.0861 - val_acc: 0.6359
Epoch 67/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.0102 -
acc: 0.6520 - val_loss: 1.0807 - val_acc: 0.6299
Epoch 68/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0210 -
acc: 0.6493 - val_loss: 1.1657 - val_acc: 0.6283
Epoch 69/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0138 -
acc: 0.6493 - val_loss: 1.1880 - val_acc: 0.6270
Epoch 70/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0130 -
acc: 0.6513 - val_loss: 1.1528 - val_acc: 0.6415
Epoch 71/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0109 -
acc: 0.6509 - val_loss: 1.0816 - val_acc: 0.6385
Epoch 72/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0109 -
acc: 0.6531 - val_loss: 1.0536 - val_acc: 0.6381
```

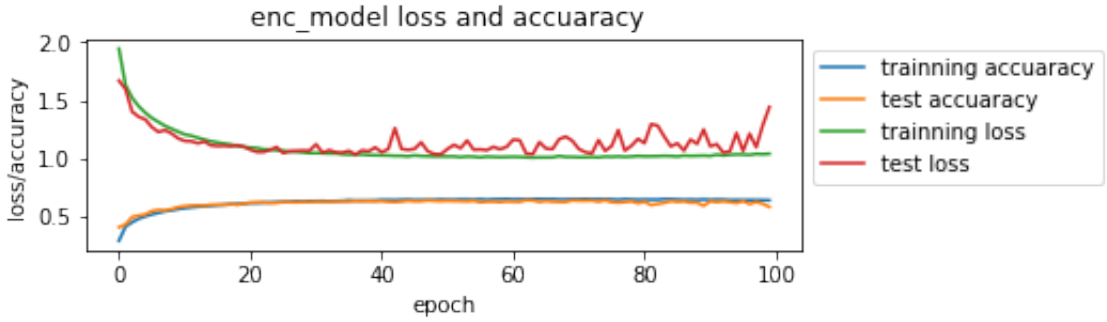
```
Epoch 73/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0130 -
acc: 0.6507 - val_loss: 1.0437 - val_acc: 0.6399
Epoch 74/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0098 -
acc: 0.6506 - val_loss: 1.1587 - val_acc: 0.6295
Epoch 75/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0164 -
acc: 0.6509 - val_loss: 1.0657 - val_acc: 0.6426
Epoch 76/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0144 -
acc: 0.6500 - val_loss: 1.1067 - val_acc: 0.6261
Epoch 77/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0176 -
acc: 0.6469 - val_loss: 1.2438 - val_acc: 0.6220
Epoch 78/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0140 -
acc: 0.6513 - val_loss: 1.0676 - val_acc: 0.6265
Epoch 79/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0170 -
acc: 0.6493 - val_loss: 1.1088 - val_acc: 0.6325
Epoch 80/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0214 -
acc: 0.6473 - val_loss: 1.1681 - val_acc: 0.6173
Epoch 81/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.0186 -
acc: 0.6483 - val_loss: 1.1315 - val_acc: 0.6295
Epoch 82/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0192 -
acc: 0.6482 - val_loss: 1.2945 - val_acc: 0.6014
Epoch 83/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.0200 -
acc: 0.6476 - val_loss: 1.2820 - val_acc: 0.6103
Epoch 84/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0223 -
acc: 0.6517 - val_loss: 1.1883 - val_acc: 0.6197
Epoch 85/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.0209 -
acc: 0.6509 - val_loss: 1.1045 - val_acc: 0.6352
Epoch 86/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0173 -
acc: 0.6490 - val_loss: 1.1296 - val_acc: 0.6284
Epoch 87/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.0250 -
acc: 0.6463 - val_loss: 1.0686 - val_acc: 0.6324
Epoch 88/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0217 -
acc: 0.6481 - val_loss: 1.1624 - val_acc: 0.6218
```

```
Epoch 89/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.0233 -
acc: 0.6463 - val_loss: 1.1256 - val_acc: 0.6224
Epoch 90/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.0248 -
acc: 0.6494 - val_loss: 1.2486 - val_acc: 0.5945
Epoch 91/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0220 -
acc: 0.6460 - val_loss: 1.1055 - val_acc: 0.6328
Epoch 92/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0332 -
acc: 0.6484 - val_loss: 1.1222 - val_acc: 0.6268
Epoch 93/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0259 -
acc: 0.6475 - val_loss: 1.0499 - val_acc: 0.6328
Epoch 94/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0322 -
acc: 0.6441 - val_loss: 1.0620 - val_acc: 0.6274
Epoch 95/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0347 -
acc: 0.6449 - val_loss: 1.2192 - val_acc: 0.6183
Epoch 96/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0342 -
acc: 0.6439 - val_loss: 1.0648 - val_acc: 0.6350
Epoch 97/100
50000/50000 [==============================] - 160s 3ms/step - loss: 1.0308 -
acc: 0.6444 - val_loss: 1.2127 - val_acc: 0.6102
Epoch 98/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0397 -
acc: 0.6442 - val_loss: 1.0968 - val_acc: 0.6275
Epoch 99/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0368 -
acc: 0.6436 - val_loss: 1.2858 - val_acc: 0.6120
Epoch 100/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0408 -
acc: 0.6420 - val_loss: 1.4412 - val_acc: 0.5851
```

[371]:
```python
def acc_loss_graphic(model_history, title):
    fig = plt.figure()
    plt.subplot(2,1,2)
    plt.plot(model_history.history['acc'], label = 'trainning accuaracy')
    plt.plot(model_history.history['val_acc'], label = 'test accuaracy')
    plt.plot(model_history.history['loss'], label = 'trainning loss')
    plt.plot(model_history.history['val_loss'], label = 'test loss')
    plt.title(title)
    plt.ylabel('loss/accuracy')
    plt.xlabel('epoch')
```

```
    plt.legend(bbox_to_anchor=(1,1), loc='upper left', ncol =1)
```

[372]:
```
acc_loss_graphic(enc1_history, ' enc_model loss and accuaracy')
```



[373]:
```
# Score trained enc_model.
enc1_scores = enc1_model.evaluate(enc1_X_test, enc1_y_test, verbose=1)
print('Test loss:', enc1_scores[0])
print('Test accuracy:', enc1_scores[1])

# make prediction.
enc_pred = enc1_model.predict(enc1_X_test)
```

```
10000/10000 [==============================] - 6s 598us/step
Test loss: 1.4412360233306885
Test accuracy: 0.5851
```

3 Train with encrypted CIFAR10 dataset using key 2

Drive orthogonal matrix to get encryption key Q2 and decryption key D2

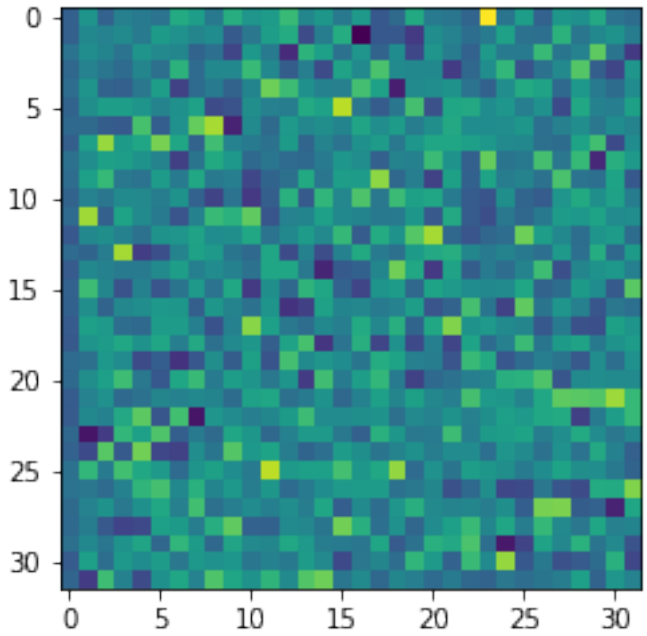
[374]:
```
#Read key image as RGB
key2 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/
 ↪key2.png")
#Show RGB key image
plt.imshow(key2)
#gray scale key image
key2 = key2.convert('L')
#resize key image
key2 = np.resize(key2,(32,32))
#convert to 2D array
key2 = np.asarray(key2)
#QR decomposition
Q2, R2 = scipy.linalg.qr(key2)
#Drive the decryption matrix
```

```
D2 = np.linalg.inv(Q2)
```

Encryption key Q2

[375]:
```
plt.imshow(Q2)
```

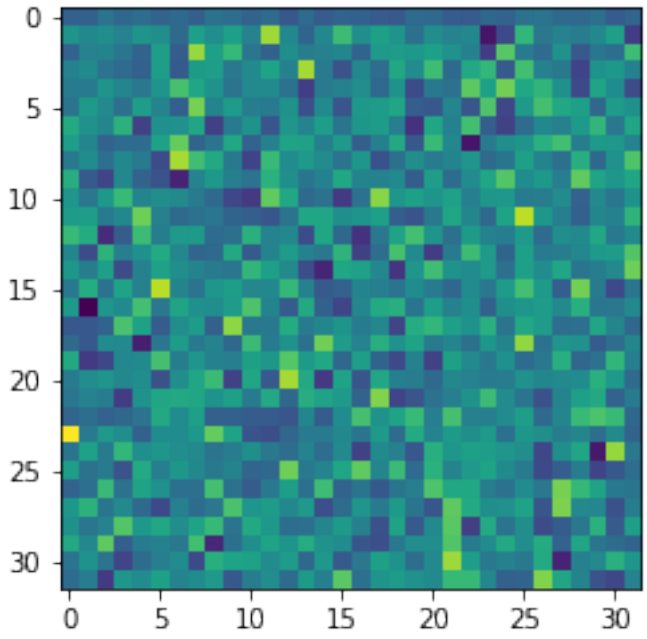
[375]:
```
<matplotlib.image.AxesImage at 0x1602113c8>
```



Decryption key D2

[376]:
```
plt.imshow(D2)
```

[376]:
```
<matplotlib.image.AxesImage at 0x161394ba8>
```



Encrypt the dataset

[377]:
```
enc2_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    enc2_b = np.dot(b, Q2)
    enc2_g = np.dot(g, Q2)
    enc2_r = np.dot(r, Q2)
    enc2_rgb = cv2.merge((enc2_b,enc2_g,enc2_r))
    enc2_X_train.append(enc2_rgb)
print("Done encrypting ",len(enc2_X_train),"train images!")
enc2_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    enc2_b = np.dot(b, Q2)
    enc2_g = np.dot(g, Q2)
    enc2_r = np.dot(r, Q2)
    enc2_rgb = cv2.merge((enc2_b,enc2_g,enc2_r))
    enc2_X_test.append(enc2_rgb)
print("Done encrypting ",len(enc2_X_test),"test images!")
```

```
Done encrypting  50000 train images!
Done encrypting  10000 test images!
```

[378]:
```
# Normalize the data.
enc2_X_train = np.array(enc2_X_train)
enc2_X_test = np.array(enc2_X_test)

enc2_X_train = enc2_X_train.astype('float32')
enc2_X_test = enc2_X_test.astype('float32')
enc2_X_train /= 255
enc2_X_test /= 255

# Convert class vectors to binary class matrices(one hot encoding).
enc2_y_train = keras.utils.to_categorical(y_train, num_classes)
enc2_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[379]:
```
print('enc2_x_train shape:', enc2_X_train.shape)
print('enc2_y_train shape:', enc2_y_train.shape)
print('enc2_x_test shape:', enc2_X_test.shape)
print('enc2_y_test shape:', enc2_y_test.shape)
```

```
enc2_x_train shape: (50000, 32, 32, 3)
enc2_y_train shape: (50000, 10)
enc2_x_test shape: (10000, 32, 32, 3)
enc2_y_test shape: (10000, 10)
```

[380]:
```
#define the convnet
enc2_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc2_model.add(Conv2D(32, (3, 3), padding='same',input_shape = enc2_X_train.
 ↪shape[1:]))
enc2_model.add(Activation('relu'))
enc2_model.add(Conv2D(32, (3, 3)))
enc2_model.add(Activation('relu'))
enc2_model.add(MaxPooling2D(pool_size=(2, 2)))
enc2_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc2_model.add(Conv2D(64, (3, 3), padding='same'))
enc2_model.add(Activation('relu'))
enc2_model.add(Conv2D(64, (3, 3)))
enc2_model.add(Activation('relu'))
enc2_model.add(MaxPooling2D(pool_size=(2, 2)))
enc2_model.add(Dropout(0.25))

# FLATTERN => DENSE => RELU => DROPOUT
enc2_model.add(Flatten())
```

```
enc2_model.add(Dense(512))
enc2_model.add(Activation('relu'))
enc2_model.add(Dropout(0.5))
# a softmax classifier
enc2_model.add(Dense(num_classes))
enc2_model.add(Activation('softmax'))
```

[381]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Compile
enc2_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[382]:
```
#Train model
enc2_history = enc2_model.fit(enc2_X_train, enc2_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(enc2_X_test, enc2_y_test),
              shuffle=True)
```

```
Not using data augmentation.
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.9538 -
acc: 0.2902 - val_loss: 1.6847 - val_acc: 0.4045
Epoch 2/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.6299 -
acc: 0.4186 - val_loss: 1.5343 - val_acc: 0.4542
Epoch 3/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.5080 -
acc: 0.4611 - val_loss: 1.4034 - val_acc: 0.4989
Epoch 4/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.4380 -
acc: 0.4860 - val_loss: 1.3460 - val_acc: 0.5219
Epoch 5/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3905 -
acc: 0.5013 - val_loss: 1.2863 - val_acc: 0.5389
Epoch 6/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3474 -
acc: 0.5201 - val_loss: 1.2531 - val_acc: 0.5501
Epoch 7/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3113 -
acc: 0.5310 - val_loss: 1.2501 - val_acc: 0.5547
Epoch 8/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.2825 -
```

```
acc: 0.5433 - val_loss: 1.2111 - val_acc: 0.5620
Epoch 9/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.2564 -
acc: 0.5533 - val_loss: 1.1773 - val_acc: 0.5761
Epoch 10/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.2352 -
acc: 0.5627 - val_loss: 1.1853 - val_acc: 0.5782
Epoch 11/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.2177 -
acc: 0.5689 - val_loss: 1.1609 - val_acc: 0.5840
Epoch 12/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1979 -
acc: 0.5783 - val_loss: 1.1439 - val_acc: 0.5920
Epoch 13/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1781 -
acc: 0.5821 - val_loss: 1.1695 - val_acc: 0.5862
Epoch 14/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1664 -
acc: 0.5881 - val_loss: 1.1249 - val_acc: 0.6081
Epoch 15/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1489 -
acc: 0.5937 - val_loss: 1.1015 - val_acc: 0.6047
Epoch 16/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1430 -
acc: 0.5950 - val_loss: 1.0934 - val_acc: 0.6116
Epoch 17/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1312 -
acc: 0.6013 - val_loss: 1.0908 - val_acc: 0.6090
Epoch 18/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1195 -
acc: 0.6067 - val_loss: 1.0773 - val_acc: 0.6116
Epoch 19/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1101 -
acc: 0.6093 - val_loss: 1.0721 - val_acc: 0.6199
Epoch 20/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1005 -
acc: 0.6105 - val_loss: 1.0608 - val_acc: 0.6238
Epoch 21/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0958 -
acc: 0.6154 - val_loss: 1.0623 - val_acc: 0.6214
Epoch 22/100
50000/50000 [==============================] - 170s 3ms/step - loss: 1.0906 -
acc: 0.6174 - val_loss: 1.0751 - val_acc: 0.6240
Epoch 23/100
50000/50000 [==============================] - 182s 4ms/step - loss: 1.0870 -
acc: 0.6172 - val_loss: 1.0453 - val_acc: 0.6311
Epoch 24/100
50000/50000 [==============================] - 167s 3ms/step - loss: 1.0781 -
```

```
acc: 0.6226 - val_loss: 1.0483 - val_acc: 0.6279
Epoch 25/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.0786 -
acc: 0.6245 - val_loss: 1.0689 - val_acc: 0.6307
Epoch 26/100
50000/50000 [==============================] - 162s 3ms/step - loss: 1.0708 -
acc: 0.6233 - val_loss: 1.0625 - val_acc: 0.6291
Epoch 27/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0737 -
acc: 0.6258 - val_loss: 1.1032 - val_acc: 0.6330
Epoch 28/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.0594 -
acc: 0.6288 - val_loss: 1.0893 - val_acc: 0.6289
Epoch 29/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.0596 -
acc: 0.6297 - val_loss: 1.0579 - val_acc: 0.6280
Epoch 30/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.0516 -
acc: 0.6330 - val_loss: 1.0375 - val_acc: 0.6369
Epoch 31/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.0506 -
acc: 0.6322 - val_loss: 1.0582 - val_acc: 0.6247
Epoch 32/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0526 -
acc: 0.6321 - val_loss: 1.0404 - val_acc: 0.6380
Epoch 33/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0452 -
acc: 0.6328 - val_loss: 1.0443 - val_acc: 0.6394
Epoch 34/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.0449 -
acc: 0.6367 - val_loss: 1.0390 - val_acc: 0.6321
Epoch 35/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0426 -
acc: 0.6378 - val_loss: 1.0764 - val_acc: 0.6417
Epoch 36/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0376 -
acc: 0.6347 - val_loss: 1.0118 - val_acc: 0.6451
Epoch 37/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0377 -
acc: 0.6402 - val_loss: 1.0464 - val_acc: 0.6352
Epoch 38/100
50000/50000 [==============================] - 160s 3ms/step - loss: 1.0354 -
acc: 0.6388 - val_loss: 1.0247 - val_acc: 0.6430
Epoch 39/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0328 -
acc: 0.6415 - val_loss: 1.0351 - val_acc: 0.6385
Epoch 40/100
50000/50000 [==============================] - 210s 4ms/step - loss: 1.0321 -
```

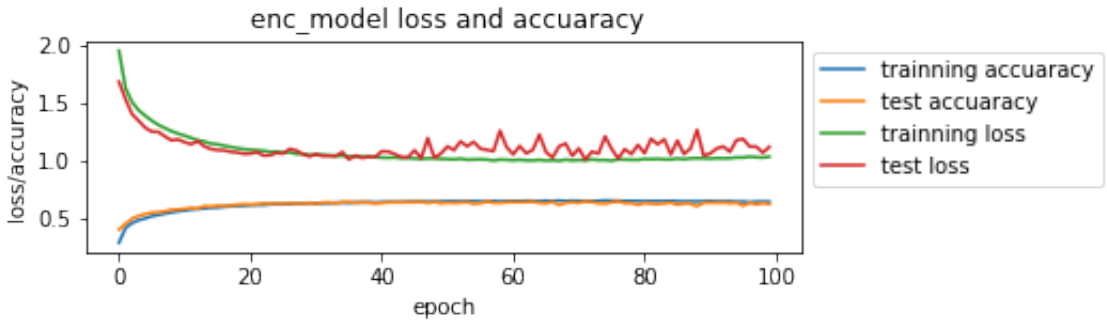
```
acc: 0.6394 - val_loss: 1.0313 - val_acc: 0.6324
Epoch 41/100
50000/50000 [==============================] - 240s 5ms/step - loss: 1.0275 -
acc: 0.6423 - val_loss: 1.0821 - val_acc: 0.6350
Epoch 42/100
50000/50000 [==============================] - 248s 5ms/step - loss: 1.0266 -
acc: 0.6433 - val_loss: 1.0801 - val_acc: 0.6401
Epoch 43/100
50000/50000 [==============================] - 160s 3ms/step - loss: 1.0286 -
acc: 0.6419 - val_loss: 1.0572 - val_acc: 0.6418
Epoch 44/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.0218 -
acc: 0.6430 - val_loss: 1.0325 - val_acc: 0.6404
Epoch 45/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0240 -
acc: 0.6451 - val_loss: 1.0299 - val_acc: 0.6413
Epoch 46/100
50000/50000 [==============================] - 240s 5ms/step - loss: 1.0194 -
acc: 0.6464 - val_loss: 1.0911 - val_acc: 0.6415
Epoch 47/100
50000/50000 [==============================] - 242s 5ms/step - loss: 1.0169 -
acc: 0.6464 - val_loss: 1.0220 - val_acc: 0.6452
Epoch 48/100
50000/50000 [==============================] - 211s 4ms/step - loss: 1.0199 -
acc: 0.6455 - val_loss: 1.1961 - val_acc: 0.6345
Epoch 49/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.0215 -
acc: 0.6461 - val_loss: 1.0272 - val_acc: 0.6454
Epoch 50/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.0166 -
acc: 0.6471 - val_loss: 1.0623 - val_acc: 0.6317
Epoch 51/100
50000/50000 [==============================] - 230s 5ms/step - loss: 1.0148 -
acc: 0.6501 - val_loss: 1.1237 - val_acc: 0.6382
Epoch 52/100
50000/50000 [==============================] - 240s 5ms/step - loss: 1.0126 -
acc: 0.6495 - val_loss: 1.0926 - val_acc: 0.6347
Epoch 53/100
50000/50000 [==============================] - 210s 4ms/step - loss: 1.0178 -
acc: 0.6489 - val_loss: 1.1679 - val_acc: 0.6376
Epoch 54/100
50000/50000 [==============================] - 205s 4ms/step - loss: 1.0091 -
acc: 0.6502 - val_loss: 1.1257 - val_acc: 0.6427
Epoch 55/100
50000/50000 [==============================] - 196s 4ms/step - loss: 1.0116 -
acc: 0.6504 - val_loss: 1.1639 - val_acc: 0.6273
Epoch 56/100
50000/50000 [==============================] - 191s 4ms/step - loss: 1.0126 -
```

```
acc: 0.6481 - val_loss: 1.1048 - val_acc: 0.6330
Epoch 57/100
50000/50000 [==============================] - 195s 4ms/step - loss: 1.0104 -
acc: 0.6474 - val_loss: 1.0948 - val_acc: 0.6461
Epoch 58/100
50000/50000 [==============================] - 193s 4ms/step - loss: 1.0023 -
acc: 0.6516 - val_loss: 1.0830 - val_acc: 0.6371
Epoch 59/100
50000/50000 [==============================] - 189s 4ms/step - loss: 1.0110 -
acc: 0.6487 - val_loss: 1.2600 - val_acc: 0.6349
Epoch 60/100
50000/50000 [==============================] - 195s 4ms/step - loss: 1.0083 -
acc: 0.6506 - val_loss: 1.1198 - val_acc: 0.6408
Epoch 61/100
50000/50000 [==============================] - 195s 4ms/step - loss: 1.0046 -
acc: 0.6519 - val_loss: 1.0553 - val_acc: 0.6421
Epoch 62/100
50000/50000 [==============================] - 208s 4ms/step - loss: 1.0084 -
acc: 0.6515 - val_loss: 1.1285 - val_acc: 0.6409
Epoch 63/100
50000/50000 [==============================] - 208s 4ms/step - loss: 1.0038 -
acc: 0.6538 - val_loss: 1.0646 - val_acc: 0.6426
Epoch 64/100
50000/50000 [==============================] - 170s 3ms/step - loss: 1.0027 -
acc: 0.6534 - val_loss: 1.0948 - val_acc: 0.6475
Epoch 65/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.0082 -
acc: 0.6496 - val_loss: 1.2264 - val_acc: 0.6355
Epoch 66/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0015 -
acc: 0.6534 - val_loss: 1.0757 - val_acc: 0.6317
Epoch 67/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.0041 -
acc: 0.6524 - val_loss: 1.0287 - val_acc: 0.6451
Epoch 68/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.0002 -
acc: 0.6554 - val_loss: 1.1259 - val_acc: 0.6167
Epoch 69/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.0099 -
acc: 0.6515 - val_loss: 1.1492 - val_acc: 0.6365
Epoch 70/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.0027 -
acc: 0.6546 - val_loss: 1.0426 - val_acc: 0.6406
Epoch 71/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0066 -
acc: 0.6529 - val_loss: 1.1074 - val_acc: 0.6358
Epoch 72/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.0034 -
```

```
acc: 0.6531 - val_loss: 1.0100 - val_acc: 0.6500
Epoch 73/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.0120 -
acc: 0.6511 - val_loss: 1.0793 - val_acc: 0.6427
Epoch 74/100
50000/50000 [==============================] - 160s 3ms/step - loss: 1.0057 -
acc: 0.6533 - val_loss: 1.0630 - val_acc: 0.6322
Epoch 75/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.0063 -
acc: 0.6561 - val_loss: 1.2115 - val_acc: 0.6297
Epoch 76/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0008 -
acc: 0.6552 - val_loss: 1.1023 - val_acc: 0.6414
Epoch 77/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0119 -
acc: 0.6545 - val_loss: 1.0195 - val_acc: 0.6487
Epoch 78/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0103 -
acc: 0.6516 - val_loss: 1.1006 - val_acc: 0.6332
Epoch 79/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0092 -
acc: 0.6519 - val_loss: 1.0584 - val_acc: 0.6339
Epoch 80/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0085 -
acc: 0.6516 - val_loss: 1.1371 - val_acc: 0.6181
Epoch 81/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0153 -
acc: 0.6494 - val_loss: 1.0646 - val_acc: 0.6344
Epoch 82/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0135 -
acc: 0.6511 - val_loss: 1.1902 - val_acc: 0.6288
Epoch 83/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0162 -
acc: 0.6514 - val_loss: 1.1432 - val_acc: 0.6257
Epoch 84/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0124 -
acc: 0.6501 - val_loss: 1.1864 - val_acc: 0.6309
Epoch 85/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0117 -
acc: 0.6529 - val_loss: 1.0553 - val_acc: 0.6379
Epoch 86/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0197 -
acc: 0.6494 - val_loss: 1.1751 - val_acc: 0.6204
Epoch 87/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0139 -
acc: 0.6480 - val_loss: 1.0586 - val_acc: 0.6305
Epoch 88/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0208 -
```

```
acc: 0.6491 - val_loss: 1.1056 - val_acc: 0.6285
Epoch 89/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0227 -
acc: 0.6483 - val_loss: 1.2673 - val_acc: 0.6069
Epoch 90/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0214 -
acc: 0.6494 - val_loss: 1.0450 - val_acc: 0.6361
Epoch 91/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0187 -
acc: 0.6517 - val_loss: 1.0629 - val_acc: 0.6377
Epoch 92/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0225 -
acc: 0.6483 - val_loss: 1.1074 - val_acc: 0.6355
Epoch 93/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0212 -
acc: 0.6486 - val_loss: 1.1236 - val_acc: 0.6387
Epoch 94/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0223 -
acc: 0.6493 - val_loss: 1.0778 - val_acc: 0.6331
Epoch 95/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0287 -
acc: 0.6457 - val_loss: 1.1787 - val_acc: 0.6366
Epoch 96/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0288 -
acc: 0.6470 - val_loss: 1.1894 - val_acc: 0.6114
Epoch 97/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0353 -
acc: 0.6413 - val_loss: 1.1233 - val_acc: 0.6354
Epoch 98/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0314 -
acc: 0.6465 - val_loss: 1.1226 - val_acc: 0.6226
Epoch 99/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0282 -
acc: 0.6463 - val_loss: 1.0705 - val_acc: 0.6328
Epoch 100/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0354 -
acc: 0.6453 - val_loss: 1.1209 - val_acc: 0.6248
```

[383]:
```
acc_loss_graphic(enc2_history, ' enc_model loss and accuaracy')
```



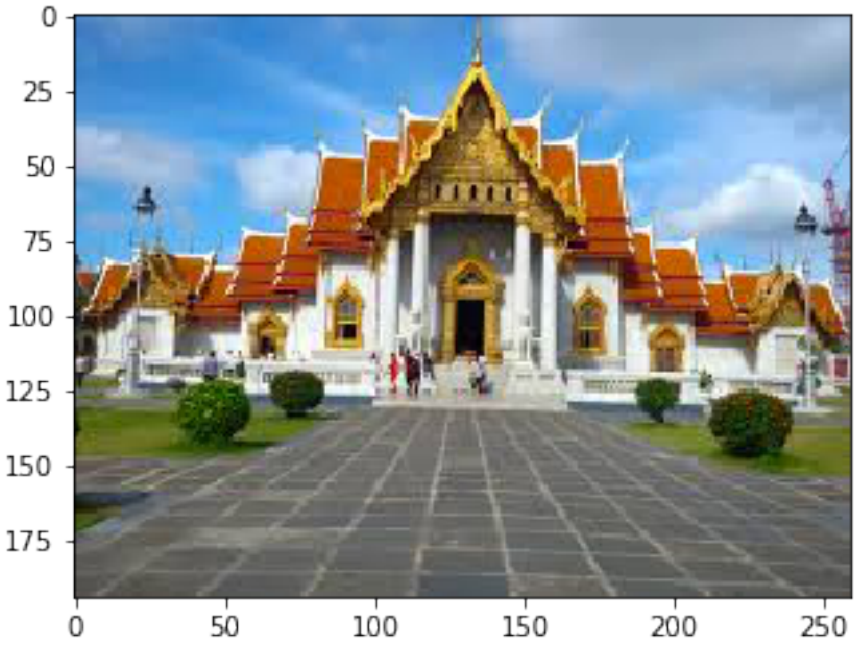
[384]:
```
# Score trained enc_model.
enc2_scores = enc2_model.evaluate(enc2_X_test, enc2_y_test, verbose=1)
print('Test loss:', enc2_scores[0])
print('Test accuracy:', enc2_scores[1])

# make prediction.
enc_pred = enc2_model.predict(enc2_X_test)
```

```
10000/10000 [==============================] - 6s 609us/step
Test loss: 1.1209170204162597
Test accuracy: 0.6248
```

4 Train with encrypted CIFAR10 dataset using key 3

Drive orthogonal matrix to get encryption key Q3 and decryption key D3

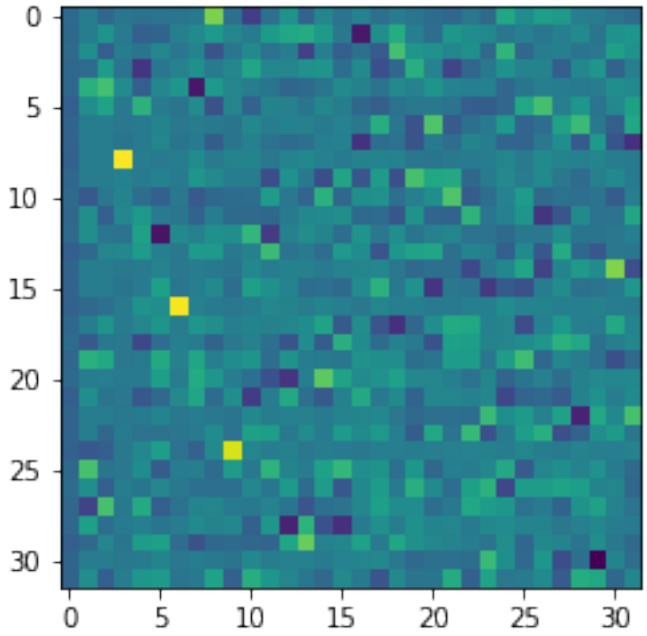
[385]:
```
#Read key image as RGB
key3 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/
 ↪key3.png")
#Show RGB key image
plt.imshow(key3)
#gray scale key image
key3 = key3.convert('L')
#resize key image
key3 = np.resize(key3,(32,32))
#convert to 2D array
key3 = np.asarray(key3)
#QR decomposition
Q3, R3 = scipy.linalg.qr(key3)
#Drive the decryption matrix
D3 = np.linalg.inv(Q3)
```



Encryption key Q3

[386]:
```
plt.imshow(Q3)
```

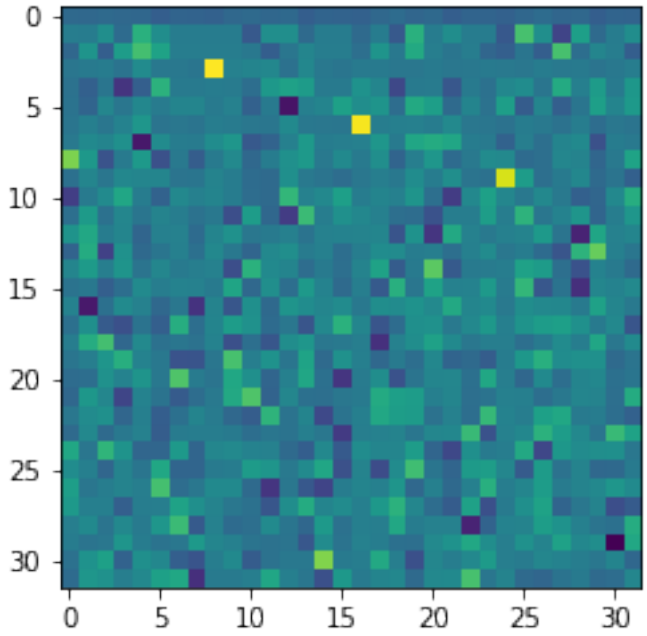
[386]:
```
<matplotlib.image.AxesImage at 0x161c80eb8>
```



Decryption key D3

[387]:
```
plt.imshow(D3)
```

[387]:
```
<matplotlib.image.AxesImage at 0x16b617748>
```



Encrypt the dataset

[388]:
```
enc3_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    enc3_b = np.dot(b, Q3)
    enc3_g = np.dot(g, Q3)
    enc3_r = np.dot(r, Q3)
    enc3_rgb = cv2.merge((enc3_b,enc3_g,enc3_r))
    enc3_X_train.append(enc3_rgb)
print("Done encrypting ",len(enc3_X_train),"train images!")
enc3_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    enc3_b = np.dot(b, Q3)
    enc3_g = np.dot(g, Q3)
    enc3_r = np.dot(r, Q3)
    enc3_rgb = cv2.merge((enc3_b,enc3_g,enc3_r))
    enc3_X_test.append(enc3_rgb)
```

```
print("Done encrypting ",len(enc3_X_test),"test images!")
```

```
Done encrypting  50000 train images!
Done encrypting  10000 test images!
```

[389]:
```
# Normalize the data.
enc3_X_train = np.array(enc3_X_train)
enc3_X_test = np.array(enc3_X_test)

enc3_X_train = enc3_X_train.astype('float32')
enc3_X_test = enc3_X_test.astype('float32')
enc3_X_train /= 255
enc3_X_test /= 255

# Convert class vectors to binary class matrices(one hot encoding).
enc3_y_train = keras.utils.to_categorical(y_train, num_classes)
enc3_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[390]:
```
print('enc3_x_train shape:', enc3_X_train.shape)
print('enc3_y_train shape:', enc3_y_train.shape)
print('enc3_x_test shape:', enc3_X_test.shape)
print('enc3_y_test shape:', enc3_y_test.shape)
```

```
enc3_x_train shape: (50000, 32, 32, 3)
enc3_y_train shape: (50000, 10)
enc3_x_test shape: (10000, 32, 32, 3)
enc3_y_test shape: (10000, 10)
```

[391]:
```
#define the convnet
enc3_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc3_model.add(Conv2D(32, (3, 3), padding='same',input_shape = enc3_X_train.
 ↪shape[1:]))
enc3_model.add(Activation('relu'))
enc3_model.add(Conv2D(32, (3, 3)))
enc3_model.add(Activation('relu'))
enc3_model.add(MaxPooling2D(pool_size=(2, 2)))
enc3_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc3_model.add(Conv2D(64, (3, 3), padding='same'))
enc3_model.add(Activation('relu'))
enc3_model.add(Conv2D(64, (3, 3)))
enc3_model.add(Activation('relu'))
enc3_model.add(MaxPooling2D(pool_size=(2, 2)))
enc3_model.add(Dropout(0.25))
```

```
# FLATTERN => DENSE => RELU => DROPOUT
enc3_model.add(Flatten())
enc3_model.add(Dense(512))
enc3_model.add(Activation('relu'))
enc3_model.add(Dropout(0.5))
# a softmax classifier
enc3_model.add(Dense(num_classes))
enc3_model.add(Activation('softmax'))
```

[392]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Compile
enc3_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[393]:
```
#Train model
enc3_history = enc3_model.fit(enc3_X_train, enc3_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(enc3_X_test, enc3_y_test),
              shuffle=True)
```

```
Not using data augmentation.
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.9081 -
acc: 0.3082 - val_loss: 1.6144 - val_acc: 0.4315
Epoch 2/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.6013 -
acc: 0.4240 - val_loss: 1.4631 - val_acc: 0.4772
Epoch 3/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.4953 -
acc: 0.4630 - val_loss: 1.3888 - val_acc: 0.5044
Epoch 4/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.4220 -
acc: 0.4921 - val_loss: 1.3249 - val_acc: 0.5241
Epoch 5/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3709 -
acc: 0.5131 - val_loss: 1.3558 - val_acc: 0.5228
Epoch 6/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3289 -
acc: 0.5275 - val_loss: 1.2476 - val_acc: 0.5566
Epoch 7/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.2919 -
acc: 0.5416 - val_loss: 1.2412 - val_acc: 0.5570
```

```
Epoch 8/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.2628 -
acc: 0.5553 - val_loss: 1.2258 - val_acc: 0.5649
Epoch 9/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.2362 -
acc: 0.5635 - val_loss: 1.1666 - val_acc: 0.5847
Epoch 10/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.2141 -
acc: 0.5695 - val_loss: 1.1637 - val_acc: 0.5846
Epoch 11/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1941 -
acc: 0.5771 - val_loss: 1.1428 - val_acc: 0.5948
Epoch 12/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1699 -
acc: 0.5872 - val_loss: 1.1317 - val_acc: 0.5962
Epoch 13/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.1558 -
acc: 0.5937 - val_loss: 1.1046 - val_acc: 0.6036
Epoch 14/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.1431 -
acc: 0.6002 - val_loss: 1.1157 - val_acc: 0.6050
Epoch 15/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.1304 -
acc: 0.6022 - val_loss: 1.0938 - val_acc: 0.6098
Epoch 16/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.1219 -
acc: 0.6056 - val_loss: 1.0926 - val_acc: 0.6145
Epoch 17/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.1102 -
acc: 0.6118 - val_loss: 1.0818 - val_acc: 0.6169
Epoch 18/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0995 -
acc: 0.6141 - val_loss: 1.0765 - val_acc: 0.6148
Epoch 19/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0957 -
acc: 0.6164 - val_loss: 1.1046 - val_acc: 0.6211
Epoch 20/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0900 -
acc: 0.6199 - val_loss: 1.0635 - val_acc: 0.6263
Epoch 21/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0791 -
acc: 0.6231 - val_loss: 1.1025 - val_acc: 0.6164
Epoch 22/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0711 -
acc: 0.6264 - val_loss: 1.0655 - val_acc: 0.6183
Epoch 23/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0733 -
acc: 0.6254 - val_loss: 1.0769 - val_acc: 0.6270
```

```
Epoch 24/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0607 -
acc: 0.6315 - val_loss: 1.0603 - val_acc: 0.6317
Epoch 25/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0591 -
acc: 0.6311 - val_loss: 1.0577 - val_acc: 0.6241
Epoch 26/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0499 -
acc: 0.6345 - val_loss: 1.0762 - val_acc: 0.6389
Epoch 27/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0563 -
acc: 0.6321 - val_loss: 1.0533 - val_acc: 0.6361
Epoch 28/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0503 -
acc: 0.6353 - val_loss: 1.0566 - val_acc: 0.6312
Epoch 29/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0471 -
acc: 0.6379 - val_loss: 1.0845 - val_acc: 0.6296
Epoch 30/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.0448 -
acc: 0.6381 - val_loss: 1.0557 - val_acc: 0.6378
Epoch 31/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.0441 -
acc: 0.6391 - val_loss: 1.0589 - val_acc: 0.6324
Epoch 32/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.0427 -
acc: 0.6390 - val_loss: 1.0699 - val_acc: 0.6269
Epoch 33/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.0392 -
acc: 0.6404 - val_loss: 1.0557 - val_acc: 0.6365
Epoch 34/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.0367 -
acc: 0.6398 - val_loss: 1.1247 - val_acc: 0.6227
Epoch 35/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0312 -
acc: 0.6432 - val_loss: 1.1201 - val_acc: 0.6307
Epoch 36/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0284 -
acc: 0.6452 - val_loss: 1.0730 - val_acc: 0.6315
Epoch 37/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.0280 -
acc: 0.6448 - val_loss: 1.0970 - val_acc: 0.6280
Epoch 38/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0278 -
acc: 0.6449 - val_loss: 1.0791 - val_acc: 0.6409
Epoch 39/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.0268 -
acc: 0.6464 - val_loss: 1.0783 - val_acc: 0.6372
```

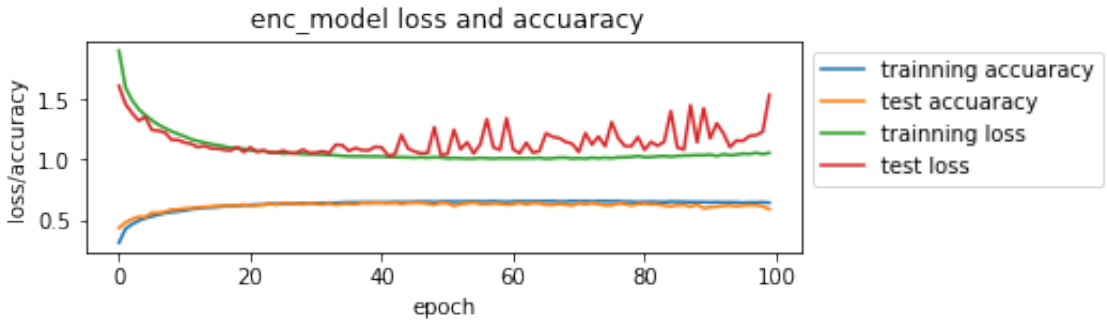
```
Epoch 40/100
50000/50000 [==============================] - 160s 3ms/step - loss: 1.0274 -
acc: 0.6434 - val_loss: 1.1074 - val_acc: 0.6423
Epoch 41/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0259 -
acc: 0.6453 - val_loss: 1.1021 - val_acc: 0.6396
Epoch 42/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.0222 -
acc: 0.6455 - val_loss: 1.0326 - val_acc: 0.6371
Epoch 43/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0210 -
acc: 0.6484 - val_loss: 1.0452 - val_acc: 0.6404
Epoch 44/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0214 -
acc: 0.6473 - val_loss: 1.2061 - val_acc: 0.6317
Epoch 45/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0193 -
acc: 0.6471 - val_loss: 1.0937 - val_acc: 0.6388
Epoch 46/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0162 -
acc: 0.6488 - val_loss: 1.0679 - val_acc: 0.6420
Epoch 47/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0205 -
acc: 0.6502 - val_loss: 1.0530 - val_acc: 0.6404
Epoch 48/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0164 -
acc: 0.6506 - val_loss: 1.0561 - val_acc: 0.6337
Epoch 49/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0207 -
acc: 0.6480 - val_loss: 1.2708 - val_acc: 0.6246
Epoch 50/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0181 -
acc: 0.6508 - val_loss: 1.0398 - val_acc: 0.6402
Epoch 51/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0109 -
acc: 0.6531 - val_loss: 1.0462 - val_acc: 0.6372
Epoch 52/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0140 -
acc: 0.6506 - val_loss: 1.2489 - val_acc: 0.6158
Epoch 53/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0122 -
acc: 0.6513 - val_loss: 1.0854 - val_acc: 0.6296
Epoch 54/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0093 -
acc: 0.6525 - val_loss: 1.1453 - val_acc: 0.6362
Epoch 55/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0101 -
acc: 0.6530 - val_loss: 1.0515 - val_acc: 0.6433
```

```
Epoch 56/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0151 -
acc: 0.6518 - val_loss: 1.1293 - val_acc: 0.6316
Epoch 57/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0117 -
acc: 0.6514 - val_loss: 1.3378 - val_acc: 0.6304
Epoch 58/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0090 -
acc: 0.6544 - val_loss: 1.1034 - val_acc: 0.6347
Epoch 59/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0129 -
acc: 0.6516 - val_loss: 1.0815 - val_acc: 0.6326
Epoch 60/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0124 -
acc: 0.6489 - val_loss: 1.3449 - val_acc: 0.6256
Epoch 61/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0120 -
acc: 0.6519 - val_loss: 1.0870 - val_acc: 0.6295
Epoch 62/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0127 -
acc: 0.6536 - val_loss: 1.0542 - val_acc: 0.6417
Epoch 63/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0128 -
acc: 0.6524 - val_loss: 1.1414 - val_acc: 0.6271
Epoch 64/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0124 -
acc: 0.6533 - val_loss: 1.0586 - val_acc: 0.6381
Epoch 65/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0159 -
acc: 0.6534 - val_loss: 1.0713 - val_acc: 0.6380
Epoch 66/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0084 -
acc: 0.6553 - val_loss: 1.2196 - val_acc: 0.6252
Epoch 67/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0116 -
acc: 0.6547 - val_loss: 1.1909 - val_acc: 0.6236
Epoch 68/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0173 -
acc: 0.6510 - val_loss: 1.1801 - val_acc: 0.6328
Epoch 69/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0173 -
acc: 0.6512 - val_loss: 1.1487 - val_acc: 0.6246
Epoch 70/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0092 -
acc: 0.6558 - val_loss: 1.1328 - val_acc: 0.6199
Epoch 71/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0129 -
acc: 0.6526 - val_loss: 1.0634 - val_acc: 0.6362
```

```
Epoch 72/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0088 -
acc: 0.6551 - val_loss: 1.2209 - val_acc: 0.6289
Epoch 73/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0169 -
acc: 0.6516 - val_loss: 1.1263 - val_acc: 0.6381
Epoch 74/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0123 -
acc: 0.6557 - val_loss: 1.1931 - val_acc: 0.6302
Epoch 75/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0169 -
acc: 0.6544 - val_loss: 1.1125 - val_acc: 0.6198
Epoch 76/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0128 -
acc: 0.6538 - val_loss: 1.3128 - val_acc: 0.6204
Epoch 77/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0199 -
acc: 0.6537 - val_loss: 1.1796 - val_acc: 0.6358
Epoch 78/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0201 -
acc: 0.6499 - val_loss: 1.1141 - val_acc: 0.6314
Epoch 79/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0232 -
acc: 0.6500 - val_loss: 1.1121 - val_acc: 0.6201
Epoch 80/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0300 -
acc: 0.6475 - val_loss: 1.1894 - val_acc: 0.6301
Epoch 81/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0217 -
acc: 0.6491 - val_loss: 1.0813 - val_acc: 0.6346
Epoch 82/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0216 -
acc: 0.6503 - val_loss: 1.1477 - val_acc: 0.6275
Epoch 83/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0278 -
acc: 0.6485 - val_loss: 1.1158 - val_acc: 0.6290
Epoch 84/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0303 -
acc: 0.6467 - val_loss: 1.1500 - val_acc: 0.6194
Epoch 85/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0253 -
acc: 0.6512 - val_loss: 1.4038 - val_acc: 0.6117
Epoch 86/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0305 -
acc: 0.6497 - val_loss: 1.1012 - val_acc: 0.6219
Epoch 87/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0339 -
acc: 0.6492 - val_loss: 1.0844 - val_acc: 0.6244
```

```
Epoch 88/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0355 -
acc: 0.6475 - val_loss: 1.4536 - val_acc: 0.6086
Epoch 89/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0379 -
acc: 0.6473 - val_loss: 1.1445 - val_acc: 0.6270
Epoch 90/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.0371 -
acc: 0.6472 - val_loss: 1.4284 - val_acc: 0.5966
Epoch 91/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0421 -
acc: 0.6445 - val_loss: 1.1773 - val_acc: 0.6041
Epoch 92/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0302 -
acc: 0.6472 - val_loss: 1.3032 - val_acc: 0.6084
Epoch 93/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0454 -
acc: 0.6449 - val_loss: 1.2202 - val_acc: 0.6163
Epoch 94/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0370 -
acc: 0.6454 - val_loss: 1.1049 - val_acc: 0.6204
Epoch 95/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0424 -
acc: 0.6451 - val_loss: 1.1594 - val_acc: 0.6111
Epoch 96/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0510 -
acc: 0.6410 - val_loss: 1.1574 - val_acc: 0.6191
Epoch 97/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0472 -
acc: 0.6445 - val_loss: 1.1984 - val_acc: 0.6221
Epoch 98/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0571 -
acc: 0.6434 - val_loss: 1.2031 - val_acc: 0.6203
Epoch 99/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0466 -
acc: 0.6455 - val_loss: 1.2335 - val_acc: 0.6105
Epoch 100/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.0563 -
acc: 0.6412 - val_loss: 1.5399 - val_acc: 0.5886
```

[394]:
```
acc_loss_graphic(enc3_history, ' enc_model loss and accuaracy')
```



[395]:
```
# Score trained enc_model.
enc3_scores = enc3_model.evaluate(enc3_X_test, enc3_y_test, verbose=1)
print('Test loss:', enc3_scores[0])
print('Test accuracy:', enc3_scores[1])

# make prediction.
enc_pred = enc3_model.predict(enc3_X_test)
```

```
10000/10000 [==============================] - 7s 653us/step
Test loss: 1.5398791343688965
Test accuracy: 0.5886
```

[ ]:
```
```

5 Train with encrypted CIFAR10 dataset using key 4

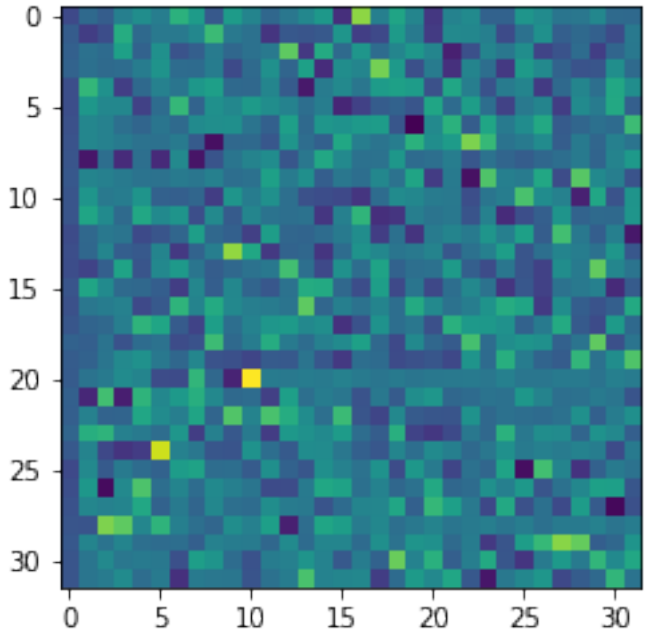
Drive orthogonal matrix to get encryption key Q4 and decryption key D4

[438]:
```
#Read key image as RGB
key4 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/
 ↪key4.png")
#Show RGB key image
plt.imshow(key4)
#gray scale key image
key4 = key4.convert('L')
#resize key image
key4 = np.resize(key4,(32,32))
#convert to 2D array
key4 = np.asarray(key4)
#QR decomposition
Q4, R4 = scipy.linalg.qr(key4)
#Drive the decryption matrix
D4 = np.linalg.inv(Q4)
```

Encryption key Q4

[439]:
```
plt.imshow(Q4)
```

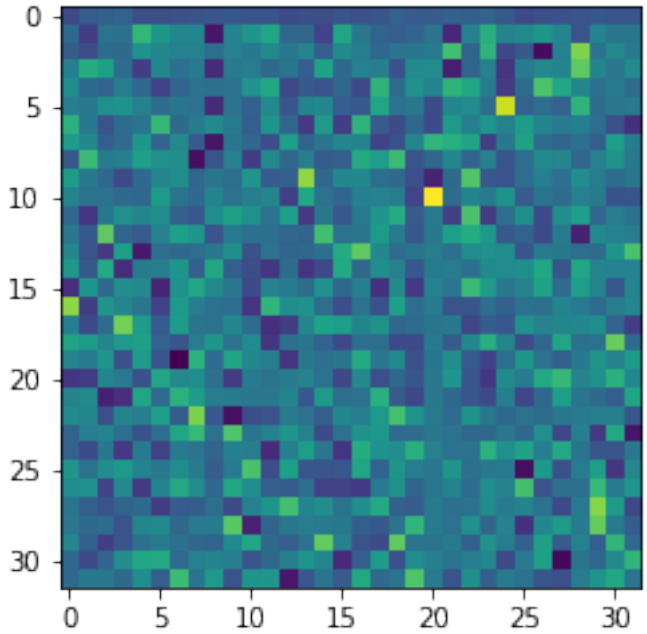
[439]:
```
<matplotlib.image.AxesImage at 0x1787482e8>
```



Decryption key D4

[440]:
```
plt.imshow(D4)
```

[440]:
```
<matplotlib.image.AxesImage at 0x16f6c7e10>
```



Encrypt the dataset

[441]:
```
enc4_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    enc4_b = np.dot(b, Q4)
    enc4_g = np.dot(g, Q4)
    enc4_r = np.dot(r, Q4)
    enc4_rgb = cv2.merge((enc4_b,enc4_g,enc4_r))
    enc4_X_train.append(enc4_rgb)
print("Done encrypting ",len(enc4_X_train),"train images!")
enc4_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    enc4_b = np.dot(b, Q4)
    enc4_g = np.dot(g, Q4)
    enc4_r = np.dot(r, Q4)
    enc4_rgb = cv2.merge((enc4_b,enc4_g,enc4_r))
    enc4_X_test.append(enc4_rgb)
```

```
print("Done encrypting ",len(enc4_X_test),"test images!")
```

```
Done encrypting  50000 train images!
Done encrypting  10000 test images!
```

[442]:
```
# Normalize the data.
enc4_X_train = np.array(enc4_X_train)
enc4_X_test = np.array(enc4_X_test)

enc4_X_train = enc4_X_train.astype('float32')
enc4_X_test = enc4_X_test.astype('float32')
enc4_X_train /= 255
enc4_X_test /= 255

# Convert class vectors to binary class matrices(one hot encoding).
enc4_y_train = keras.utils.to_categorical(y_train, num_classes)
enc4_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[443]:
```
print('enc4_x_train shape:', enc4_X_train.shape)
print('enc4_y_train shape:', enc4_y_train.shape)
print('enc4_x_test shape:', enc4_X_test.shape)
print('enc4_y_test shape:', enc4_y_test.shape)
```

```
enc4_x_train shape: (50000, 32, 32, 3)
enc4_y_train shape: (50000, 10)
enc4_x_test shape: (10000, 32, 32, 3)
enc4_y_test shape: (10000, 10)
```

[444]:
```
#define the convnet
enc4_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc4_model.add(Conv2D(32, (3, 3), padding='same',input_shape = enc4_X_train.
 ↪shape[1:]))
enc4_model.add(Activation('relu'))
enc4_model.add(Conv2D(32, (3, 3)))
enc4_model.add(Activation('relu'))
enc4_model.add(MaxPooling2D(pool_size=(2, 2)))
enc4_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc4_model.add(Conv2D(64, (3, 3), padding='same'))
enc4_model.add(Activation('relu'))
enc4_model.add(Conv2D(64, (3, 3)))
enc4_model.add(Activation('relu'))
enc4_model.add(MaxPooling2D(pool_size=(2, 2)))
enc4_model.add(Dropout(0.25))
```

```
# FLATTERN => DENSE => RELU => DROPOUT
enc4_model.add(Flatten())
enc4_model.add(Dense(512))
enc4_model.add(Activation('relu'))
enc4_model.add(Dropout(0.5))
# a softmax classifier
enc4_model.add(Dense(num_classes))
enc4_model.add(Activation('softmax'))
```

[445]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Compile
enc4_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[446]:
```
#Train model
enc4_history = enc4_model.fit(enc4_X_train, enc4_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(enc4_X_test, enc4_y_test),
              shuffle=True)
```

```
Not using data augmentation.
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.9485 -
acc: 0.2923 - val_loss: 1.6645 - val_acc: 0.4128
Epoch 2/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.6254 -
acc: 0.4187 - val_loss: 1.5035 - val_acc: 0.4678
Epoch 3/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.5161 -
acc: 0.4590 - val_loss: 1.4234 - val_acc: 0.4957
Epoch 4/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.4418 -
acc: 0.4862 - val_loss: 1.3450 - val_acc: 0.5254
Epoch 5/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3887 -
acc: 0.5063 - val_loss: 1.3032 - val_acc: 0.5329
Epoch 6/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3444 -
acc: 0.5232 - val_loss: 1.3210 - val_acc: 0.5283
Epoch 7/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.3105 -
acc: 0.5336 - val_loss: 1.2241 - val_acc: 0.5676
```

```
Epoch 8/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.2810 -
acc: 0.5471 - val_loss: 1.2187 - val_acc: 0.5634
Epoch 9/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.2513 -
acc: 0.5571 - val_loss: 1.2042 - val_acc: 0.5654
Epoch 10/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.2283 -
acc: 0.5629 - val_loss: 1.1603 - val_acc: 0.5856
Epoch 11/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.2085 -
acc: 0.5756 - val_loss: 1.1666 - val_acc: 0.5873
Epoch 12/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.1914 -
acc: 0.5790 - val_loss: 1.1643 - val_acc: 0.5905
Epoch 13/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.1704 -
acc: 0.5868 - val_loss: 1.1508 - val_acc: 0.6000
Epoch 14/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1574 -
acc: 0.5922 - val_loss: 1.1166 - val_acc: 0.6003
Epoch 15/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.1478 -
acc: 0.5955 - val_loss: 1.0984 - val_acc: 0.6090
Epoch 16/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1321 -
acc: 0.6023 - val_loss: 1.1040 - val_acc: 0.6045
Epoch 17/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.1235 -
acc: 0.6042 - val_loss: 1.1220 - val_acc: 0.6023
Epoch 18/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.1091 -
acc: 0.6099 - val_loss: 1.0719 - val_acc: 0.6171
Epoch 19/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.1035 -
acc: 0.6115 - val_loss: 1.0735 - val_acc: 0.6142
Epoch 20/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0964 -
acc: 0.6150 - val_loss: 1.0730 - val_acc: 0.6186
Epoch 21/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0881 -
acc: 0.6187 - val_loss: 1.1016 - val_acc: 0.6153
Epoch 22/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0824 -
acc: 0.6180 - val_loss: 1.0771 - val_acc: 0.6260
Epoch 23/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0755 -
acc: 0.6230 - val_loss: 1.0696 - val_acc: 0.6262
```

```
Epoch 24/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0726 -
acc: 0.6237 - val_loss: 1.0552 - val_acc: 0.6229
Epoch 25/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0669 -
acc: 0.6254 - val_loss: 1.0578 - val_acc: 0.6225
Epoch 26/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0609 -
acc: 0.6276 - val_loss: 1.0753 - val_acc: 0.6225
Epoch 27/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0578 -
acc: 0.6298 - val_loss: 1.0367 - val_acc: 0.6283
Epoch 28/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.0560 -
acc: 0.6303 - val_loss: 1.0617 - val_acc: 0.6245
Epoch 29/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0464 -
acc: 0.6339 - val_loss: 1.0594 - val_acc: 0.6272
Epoch 30/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.0466 -
acc: 0.6322 - val_loss: 1.0815 - val_acc: 0.6324
Epoch 31/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0393 -
acc: 0.6375 - val_loss: 1.0545 - val_acc: 0.6280
Epoch 32/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0386 -
acc: 0.6368 - val_loss: 1.0519 - val_acc: 0.6334
Epoch 33/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.0379 -
acc: 0.6385 - val_loss: 1.0494 - val_acc: 0.6293
Epoch 34/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0365 -
acc: 0.6392 - val_loss: 1.0987 - val_acc: 0.6123
Epoch 35/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0335 -
acc: 0.6387 - val_loss: 1.0685 - val_acc: 0.6347
Epoch 36/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0278 -
acc: 0.6408 - val_loss: 1.0959 - val_acc: 0.6217
Epoch 37/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0293 -
acc: 0.6411 - val_loss: 1.0671 - val_acc: 0.6255
Epoch 38/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0242 -
acc: 0.6445 - val_loss: 1.0589 - val_acc: 0.6357
Epoch 39/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0259 -
acc: 0.6449 - val_loss: 1.1126 - val_acc: 0.6372
```

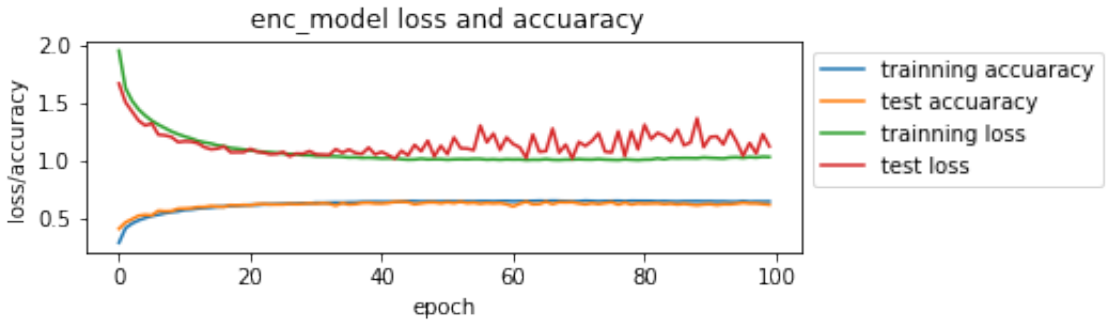
```
Epoch 40/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0231 -
acc: 0.6459 - val_loss: 1.0532 - val_acc: 0.6311
Epoch 41/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0191 -
acc: 0.6451 - val_loss: 1.0764 - val_acc: 0.6304
Epoch 42/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0201 -
acc: 0.6451 - val_loss: 1.0470 - val_acc: 0.6350
Epoch 43/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0205 -
acc: 0.6433 - val_loss: 1.0173 - val_acc: 0.6454
Epoch 44/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0132 -
acc: 0.6478 - val_loss: 1.0850 - val_acc: 0.6456
Epoch 45/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.0088 -
acc: 0.6508 - val_loss: 1.0458 - val_acc: 0.6420
Epoch 46/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.0104 -
acc: 0.6504 - val_loss: 1.1356 - val_acc: 0.6237
Epoch 47/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.0169 -
acc: 0.6478 - val_loss: 1.0811 - val_acc: 0.6325
Epoch 48/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0112 -
acc: 0.6490 - val_loss: 1.1718 - val_acc: 0.6400
Epoch 49/100
50000/50000 [==============================] - 137s 3ms/step - loss: 1.0133 -
acc: 0.6486 - val_loss: 1.0371 - val_acc: 0.6406
Epoch 50/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.0130 -
acc: 0.6484 - val_loss: 1.1288 - val_acc: 0.6328
Epoch 51/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0083 -
acc: 0.6503 - val_loss: 1.0555 - val_acc: 0.6420
Epoch 52/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.0118 -
acc: 0.6491 - val_loss: 1.2003 - val_acc: 0.6342
Epoch 53/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0134 -
acc: 0.6496 - val_loss: 1.1075 - val_acc: 0.6463
Epoch 54/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0120 -
acc: 0.6494 - val_loss: 1.1043 - val_acc: 0.6341
Epoch 55/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.0149 -
acc: 0.6490 - val_loss: 1.0911 - val_acc: 0.6382
```

```
Epoch 56/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0148 -
acc: 0.6507 - val_loss: 1.3027 - val_acc: 0.6289
Epoch 57/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0091 -
acc: 0.6508 - val_loss: 1.1574 - val_acc: 0.6355
Epoch 58/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.0114 -
acc: 0.6493 - val_loss: 1.2340 - val_acc: 0.6320
Epoch 59/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0076 -
acc: 0.6513 - val_loss: 1.0938 - val_acc: 0.6353
Epoch 60/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0118 -
acc: 0.6524 - val_loss: 1.2004 - val_acc: 0.6243
Epoch 61/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0101 -
acc: 0.6502 - val_loss: 1.1436 - val_acc: 0.6054
Epoch 62/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.0111 -
acc: 0.6514 - val_loss: 1.1153 - val_acc: 0.6286
Epoch 63/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.0058 -
acc: 0.6517 - val_loss: 1.0246 - val_acc: 0.6468
Epoch 64/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.0086 -
acc: 0.6524 - val_loss: 1.2249 - val_acc: 0.6241
Epoch 65/100
50000/50000 [==============================] - 174s 3ms/step - loss: 1.0079 -
acc: 0.6545 - val_loss: 1.0771 - val_acc: 0.6260
Epoch 66/100
50000/50000 [==============================] - 166s 3ms/step - loss: 1.0136 -
acc: 0.6531 - val_loss: 1.0827 - val_acc: 0.6462
Epoch 67/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.0076 -
acc: 0.6547 - val_loss: 1.2768 - val_acc: 0.6254
Epoch 68/100
50000/50000 [==============================] - 177s 4ms/step - loss: 1.0061 -
acc: 0.6539 - val_loss: 1.0692 - val_acc: 0.6418
Epoch 69/100
50000/50000 [==============================] - 186s 4ms/step - loss: 1.0095 -
acc: 0.6510 - val_loss: 1.1383 - val_acc: 0.6404
Epoch 70/100
50000/50000 [==============================] - 198s 4ms/step - loss: 1.0095 -
acc: 0.6491 - val_loss: 1.0192 - val_acc: 0.6419
Epoch 71/100
50000/50000 [==============================] - 184s 4ms/step - loss: 1.0130 -
acc: 0.6499 - val_loss: 1.1611 - val_acc: 0.6315
```

```
Epoch 72/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.0101 -
acc: 0.6548 - val_loss: 1.1269 - val_acc: 0.6235
Epoch 73/100
50000/50000 [==============================] - 189s 4ms/step - loss: 1.0094 -
acc: 0.6522 - val_loss: 1.1696 - val_acc: 0.6337
Epoch 74/100
50000/50000 [==============================] - 188s 4ms/step - loss: 1.0140 -
acc: 0.6493 - val_loss: 1.2375 - val_acc: 0.6256
Epoch 75/100
50000/50000 [==============================] - 173s 3ms/step - loss: 1.0096 -
acc: 0.6517 - val_loss: 1.0779 - val_acc: 0.6346
Epoch 76/100
50000/50000 [==============================] - 179s 4ms/step - loss: 1.0074 -
acc: 0.6530 - val_loss: 1.0717 - val_acc: 0.6426
Epoch 77/100
50000/50000 [==============================] - 198s 4ms/step - loss: 1.0055 -
acc: 0.6539 - val_loss: 1.2525 - val_acc: 0.6211
Epoch 78/100
50000/50000 [==============================] - 191s 4ms/step - loss: 1.0119 -
acc: 0.6504 - val_loss: 1.0399 - val_acc: 0.6387
Epoch 79/100
50000/50000 [==============================] - 172s 3ms/step - loss: 1.0081 -
acc: 0.6536 - val_loss: 1.2531 - val_acc: 0.6220
Epoch 80/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0052 -
acc: 0.6529 - val_loss: 1.0976 - val_acc: 0.6320
Epoch 81/100
50000/50000 [==============================] - 162s 3ms/step - loss: 1.0092 -
acc: 0.6529 - val_loss: 1.3094 - val_acc: 0.6289
Epoch 82/100
50000/50000 [==============================] - 179s 4ms/step - loss: 1.0100 -
acc: 0.6526 - val_loss: 1.1863 - val_acc: 0.6311
Epoch 83/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0201 -
acc: 0.6492 - val_loss: 1.2394 - val_acc: 0.6324
Epoch 84/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0122 -
acc: 0.6510 - val_loss: 1.2141 - val_acc: 0.6251
Epoch 85/100
50000/50000 [==============================] - 171s 3ms/step - loss: 1.0230 -
acc: 0.6486 - val_loss: 1.1664 - val_acc: 0.6299
Epoch 86/100
50000/50000 [==============================] - 169s 3ms/step - loss: 1.0214 -
acc: 0.6475 - val_loss: 1.2682 - val_acc: 0.6220
Epoch 87/100
50000/50000 [==============================] - 163s 3ms/step - loss: 1.0226 -
acc: 0.6496 - val_loss: 1.1323 - val_acc: 0.6278
```

```
Epoch 88/100
50000/50000 [==============================] - 161s 3ms/step - loss: 1.0221 -
acc: 0.6500 - val_loss: 1.1574 - val_acc: 0.6214
Epoch 89/100
50000/50000 [==============================] - 168s 3ms/step - loss: 1.0209 -
acc: 0.6505 - val_loss: 1.3652 - val_acc: 0.6157
Epoch 90/100
50000/50000 [==============================] - 167s 3ms/step - loss: 1.0276 -
acc: 0.6492 - val_loss: 1.1168 - val_acc: 0.6226
Epoch 91/100
50000/50000 [==============================] - 216s 4ms/step - loss: 1.0241 -
acc: 0.6497 - val_loss: 1.2044 - val_acc: 0.6257
Epoch 92/100
50000/50000 [==============================] - 227s 5ms/step - loss: 1.0216 -
acc: 0.6480 - val_loss: 1.2094 - val_acc: 0.6174
Epoch 93/100
50000/50000 [==============================] - 224s 4ms/step - loss: 1.0188 -
acc: 0.6485 - val_loss: 1.1401 - val_acc: 0.6243
Epoch 94/100
50000/50000 [==============================] - 225s 5ms/step - loss: 1.0275 -
acc: 0.6462 - val_loss: 1.2645 - val_acc: 0.6236
Epoch 95/100
50000/50000 [==============================] - 224s 4ms/step - loss: 1.0295 -
acc: 0.6459 - val_loss: 1.1635 - val_acc: 0.6293
Epoch 96/100
50000/50000 [==============================] - 229s 5ms/step - loss: 1.0240 -
acc: 0.6512 - val_loss: 1.0417 - val_acc: 0.6373
Epoch 97/100
50000/50000 [==============================] - 226s 5ms/step - loss: 1.0313 -
acc: 0.6461 - val_loss: 1.1524 - val_acc: 0.6315
Epoch 98/100
50000/50000 [==============================] - 223s 4ms/step - loss: 1.0273 -
acc: 0.6479 - val_loss: 1.0617 - val_acc: 0.6314
Epoch 99/100
50000/50000 [==============================] - 225s 5ms/step - loss: 1.0337 -
acc: 0.6472 - val_loss: 1.2277 - val_acc: 0.6260
Epoch 100/100
50000/50000 [==============================] - 216s 4ms/step - loss: 1.0314 -
acc: 0.6478 - val_loss: 1.1233 - val_acc: 0.6202
```

[447]:
```
acc_loss_graphic(enc4_history, ' enc_model loss and accuaracy')
```



[448]:
```
# Score trained enc_model.
enc4_scores = enc4_model.evaluate(enc4_X_test, enc4_y_test, verbose=1)
print('Test loss:', enc4_scores[0])
print('Test accuracy:', enc4_scores[1])

# make prediction.
enc_pred = enc4_model.predict(enc4_X_test)
```

```
10000/10000 [==============================] - 10s 974us/step
Test loss: 1.1233301809310914
Test accuracy: 0.6202
```

6 Train with encrypted CIFAR10 dataset using key 5

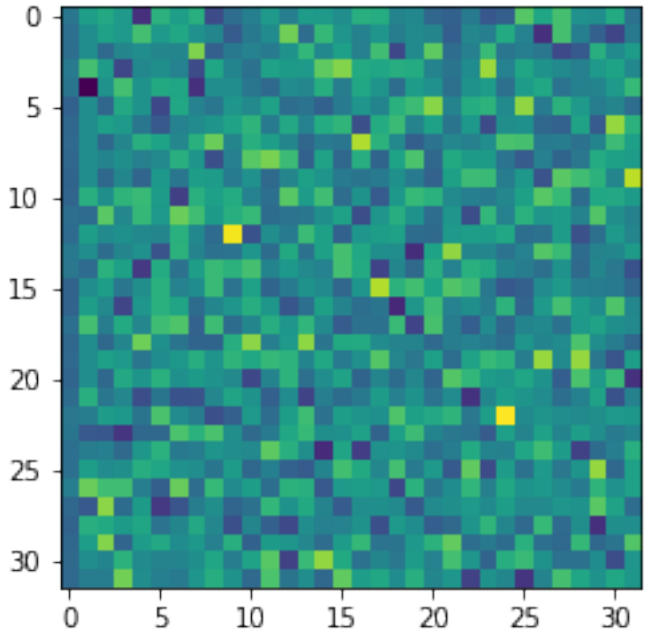
Drive orthogonal matrix to get encryption key Q5 and decryption key D5

[412]:
```
#Read key image as RGB
key5 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/
 ↪key5.png")
#Show RGB key image
plt.imshow(key5)
#gray scale key image
key5 = key5.convert('L')
#resize key image
key5 = np.resize(key5,(32,32))
#convert to 2D array
key5 = np.asarray(key5)
#QR decomposition
Q5, R5 = scipy.linalg.qr(key5)
#Drive the decryption matrix
D5 = np.linalg.inv(Q5)
```

Encryption key Q5

[413]:
```
plt.imshow(Q5)
```

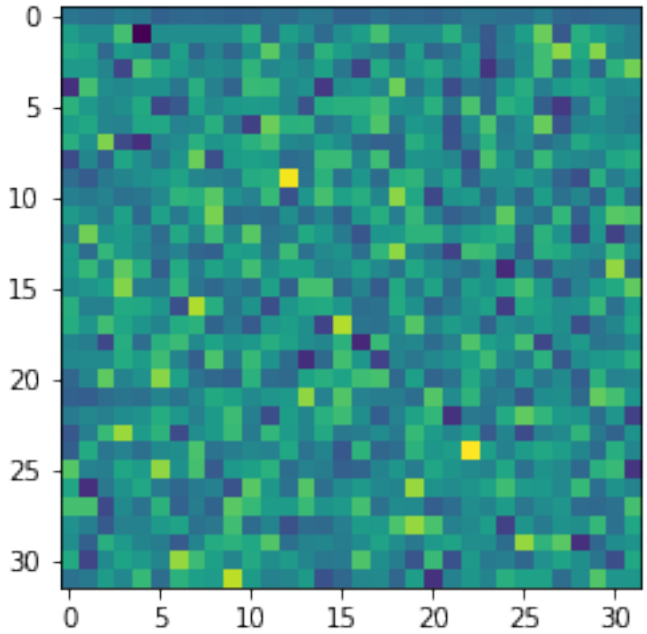
[413]:
```
<matplotlib.image.AxesImage at 0x16f7a4b70>
```



Decryption key D5

[414]:
```
plt.imshow(D5)
```

[414]:
```
<matplotlib.image.AxesImage at 0x177a72b00>
```



Encrypt the dataset

[415]:
```
enc5_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    enc5_b = np.dot(b, Q5)
    enc5_g = np.dot(g, Q5)
    enc5_r = np.dot(r, Q5)
    enc5_rgb = cv2.merge((enc5_b,enc5_g,enc5_r))
    enc5_X_train.append(enc5_rgb)
print("Done encrypting ",len(enc5_X_train),"train images!")
enc5_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    enc5_b = np.dot(b, Q5)
    enc5_g = np.dot(g, Q5)
    enc5_r = np.dot(r, Q5)
    enc5_rgb = cv2.merge((enc5_b,enc5_g,enc5_r))
    enc5_X_test.append(enc5_rgb)
```

```
print("Done encrypting ",len(enc5_X_test),"test images!")
```

```
Done encrypting  50000 train images!
Done encrypting  10000 test images!
```

[416]:
```
# Normalize the data.
enc5_X_train = np.array(enc5_X_train)
enc5_X_test = np.array(enc5_X_test)

enc5_X_train = enc5_X_train.astype('float32')
enc5_X_test = enc5_X_test.astype('float32')
enc5_X_train /= 255
enc5_X_test /= 255

# Convert class vectors to binary class matrices(one hot encoding).
enc5_y_train = keras.utils.to_categorical(y_train, num_classes)
enc5_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[417]:
```
print('enc5_x_train shape:', enc5_X_train.shape)
print('enc5_y_train shape:', enc5_y_train.shape)
print('enc5_x_test shape:', enc5_X_test.shape)
print('enc5_y_test shape:', enc5_y_test.shape)
```

```
enc5_x_train shape: (50000, 32, 32, 3)
enc5_y_train shape: (50000, 10)
enc5_x_test shape: (10000, 32, 32, 3)
enc5_y_test shape: (10000, 10)
```

[418]:
```
#define the convnet
enc5_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc5_model.add(Conv2D(32, (3, 3), padding='same',input_shape = enc5_X_train.
 ↪shape[1:]))
enc5_model.add(Activation('relu'))
enc5_model.add(Conv2D(32, (3, 3)))
enc5_model.add(Activation('relu'))
enc5_model.add(MaxPooling2D(pool_size=(2, 2)))
enc5_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
enc5_model.add(Conv2D(64, (3, 3), padding='same'))
enc5_model.add(Activation('relu'))
enc5_model.add(Conv2D(64, (3, 3)))
enc5_model.add(Activation('relu'))
enc5_model.add(MaxPooling2D(pool_size=(2, 2)))
enc5_model.add(Dropout(0.25))
```

```
# FLATTERN => DENSE => RELU => DROPOUT
enc5_model.add(Flatten())
enc5_model.add(Dense(512))
enc5_model.add(Activation('relu'))
enc5_model.add(Dropout(0.5))
# a softmax classifier
enc5_model.add(Dense(num_classes))
enc5_model.add(Activation('softmax'))
```

[419]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Compile
enc5_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[420]:
```
#Train model
enc5_history = enc5_model.fit(enc5_X_train, enc5_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(enc5_X_test, enc5_y_test),
              shuffle=True)
```

```
Not using data augmentation.
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 188s 4ms/step - loss: 1.8985 -
acc: 0.3091 - val_loss: 1.5989 - val_acc: 0.4411
Epoch 2/100
50000/50000 [==============================] - 178s 4ms/step - loss: 1.5903 -
acc: 0.4270 - val_loss: 1.4853 - val_acc: 0.4689
Epoch 3/100
50000/50000 [==============================] - 170s 3ms/step - loss: 1.4911 -
acc: 0.4663 - val_loss: 1.3955 - val_acc: 0.4993
Epoch 4/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.4328 -
acc: 0.4900 - val_loss: 1.3595 - val_acc: 0.5112
Epoch 5/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.3807 -
acc: 0.5092 - val_loss: 1.2865 - val_acc: 0.5421
Epoch 6/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.3336 -
acc: 0.5270 - val_loss: 1.2627 - val_acc: 0.5523
Epoch 7/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.3026 -
acc: 0.5379 - val_loss: 1.2330 - val_acc: 0.5626
```

```
Epoch 8/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.2707 -
acc: 0.5506 - val_loss: 1.2337 - val_acc: 0.5609
Epoch 9/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.2468 -
acc: 0.5586 - val_loss: 1.1883 - val_acc: 0.5815
Epoch 10/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.2207 -
acc: 0.5665 - val_loss: 1.1684 - val_acc: 0.5866
Epoch 11/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.1980 -
acc: 0.5786 - val_loss: 1.2015 - val_acc: 0.5729
Epoch 12/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.1822 -
acc: 0.5824 - val_loss: 1.1433 - val_acc: 0.5931
Epoch 13/100
50000/50000 [==============================] - 163s 3ms/step - loss: 1.1671 -
acc: 0.5894 - val_loss: 1.1265 - val_acc: 0.6031
Epoch 14/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.1503 -
acc: 0.5935 - val_loss: 1.1090 - val_acc: 0.6077
Epoch 15/100
50000/50000 [==============================] - 223s 4ms/step - loss: 1.1375 -
acc: 0.6003 - val_loss: 1.1125 - val_acc: 0.6076
Epoch 16/100
50000/50000 [==============================] - 219s 4ms/step - loss: 1.1291 -
acc: 0.6023 - val_loss: 1.1130 - val_acc: 0.6089
Epoch 17/100
50000/50000 [==============================] - 198s 4ms/step - loss: 1.1187 -
acc: 0.6050 - val_loss: 1.0877 - val_acc: 0.6132
Epoch 18/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.1061 -
acc: 0.6093 - val_loss: 1.1350 - val_acc: 0.6000
Epoch 19/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0929 -
acc: 0.6175 - val_loss: 1.0925 - val_acc: 0.6163
Epoch 20/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.0872 -
acc: 0.6181 - val_loss: 1.0882 - val_acc: 0.6109
Epoch 21/100
50000/50000 [==============================] - 204s 4ms/step - loss: 1.0828 -
acc: 0.6180 - val_loss: 1.0614 - val_acc: 0.6264
Epoch 22/100
50000/50000 [==============================] - 221s 4ms/step - loss: 1.0722 -
acc: 0.6244 - val_loss: 1.0613 - val_acc: 0.6222
Epoch 23/100
50000/50000 [==============================] - 253s 5ms/step - loss: 1.0688 -
acc: 0.6257 - val_loss: 1.0775 - val_acc: 0.6161
```

```
Epoch 24/100
50000/50000 [==============================] - 223s 4ms/step - loss: 1.0605 -
acc: 0.6294 - val_loss: 1.0981 - val_acc: 0.6256
Epoch 25/100
50000/50000 [==============================] - 230s 5ms/step - loss: 1.0582 -
acc: 0.6292 - val_loss: 1.0596 - val_acc: 0.6313
Epoch 26/100
50000/50000 [==============================] - 222s 4ms/step - loss: 1.0510 -
acc: 0.6328 - val_loss: 1.0509 - val_acc: 0.6311
Epoch 27/100
50000/50000 [==============================] - 230s 5ms/step - loss: 1.0450 -
acc: 0.6337 - val_loss: 1.0762 - val_acc: 0.6328
Epoch 28/100
50000/50000 [==============================] - 250s 5ms/step - loss: 1.0430 -
acc: 0.6340 - val_loss: 1.0768 - val_acc: 0.6226
Epoch 29/100
50000/50000 [==============================] - 249s 5ms/step - loss: 1.0374 -
acc: 0.6372 - val_loss: 1.0530 - val_acc: 0.6327
Epoch 30/100
50000/50000 [==============================] - 243s 5ms/step - loss: 1.0352 -
acc: 0.6383 - val_loss: 1.0934 - val_acc: 0.6142
Epoch 31/100
50000/50000 [==============================] - 231s 5ms/step - loss: 1.0332 -
acc: 0.6392 - val_loss: 1.0728 - val_acc: 0.6324
Epoch 32/100
50000/50000 [==============================] - 243s 5ms/step - loss: 1.0294 -
acc: 0.6399 - val_loss: 1.0497 - val_acc: 0.6291
Epoch 33/100
50000/50000 [==============================] - 206s 4ms/step - loss: 1.0258 -
acc: 0.6429 - val_loss: 1.0600 - val_acc: 0.6353
Epoch 34/100
50000/50000 [==============================] - 199s 4ms/step - loss: 1.0262 -
acc: 0.6412 - val_loss: 1.0387 - val_acc: 0.6358
Epoch 35/100
50000/50000 [==============================] - 199s 4ms/step - loss: 1.0209 -
acc: 0.6442 - val_loss: 1.0633 - val_acc: 0.6345
Epoch 36/100
50000/50000 [==============================] - 199s 4ms/step - loss: 1.0202 -
acc: 0.6444 - val_loss: 1.0389 - val_acc: 0.6376
Epoch 37/100
50000/50000 [==============================] - 199s 4ms/step - loss: 1.0163 -
acc: 0.6461 - val_loss: 1.0516 - val_acc: 0.6387
Epoch 38/100
50000/50000 [==============================] - 199s 4ms/step - loss: 1.0174 -
acc: 0.6473 - val_loss: 1.1125 - val_acc: 0.6367
Epoch 39/100
50000/50000 [==============================] - 201s 4ms/step - loss: 1.0116 -
acc: 0.6480 - val_loss: 1.0495 - val_acc: 0.6327
```

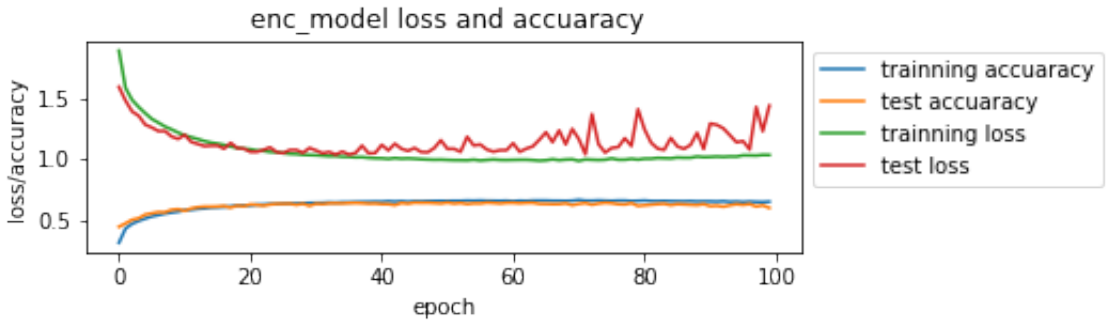
```
Epoch 40/100
50000/50000 [==============================] - 201s 4ms/step - loss: 1.0087 -
acc: 0.6494 - val_loss: 1.0513 - val_acc: 0.6338
Epoch 41/100
50000/50000 [==============================] - 214s 4ms/step - loss: 1.0076 -
acc: 0.6487 - val_loss: 1.1190 - val_acc: 0.6294
Epoch 42/100
50000/50000 [==============================] - 221s 4ms/step - loss: 1.0033 -
acc: 0.6525 - val_loss: 1.0739 - val_acc: 0.6286
Epoch 43/100
50000/50000 [==============================] - 231s 5ms/step - loss: 1.0094 -
acc: 0.6498 - val_loss: 1.1284 - val_acc: 0.6205
Epoch 44/100
50000/50000 [==============================] - 225s 5ms/step - loss: 1.0049 -
acc: 0.6525 - val_loss: 1.0867 - val_acc: 0.6385
Epoch 45/100
50000/50000 [==============================] - 245s 5ms/step - loss: 1.0069 -
acc: 0.6486 - val_loss: 1.0696 - val_acc: 0.6327
Epoch 46/100
50000/50000 [==============================] - 216s 4ms/step - loss: 1.0042 -
acc: 0.6525 - val_loss: 1.0933 - val_acc: 0.6415
Epoch 47/100
50000/50000 [==============================] - 222s 4ms/step - loss: 1.0004 -
acc: 0.6515 - val_loss: 1.0601 - val_acc: 0.6397
Epoch 48/100
50000/50000 [==============================] - 334s 7ms/step - loss: 0.9978 -
acc: 0.6539 - val_loss: 1.0580 - val_acc: 0.6403
Epoch 49/100
50000/50000 [==============================] - 348s 7ms/step - loss: 0.9959 -
acc: 0.6543 - val_loss: 1.0928 - val_acc: 0.6377
Epoch 50/100
50000/50000 [==============================] - 288s 6ms/step - loss: 0.9969 -
acc: 0.6551 - val_loss: 1.1574 - val_acc: 0.6344
Epoch 51/100
50000/50000 [==============================] - 282s 6ms/step - loss: 0.9942 -
acc: 0.6551 - val_loss: 1.0861 - val_acc: 0.6419
Epoch 52/100
50000/50000 [==============================] - 282s 6ms/step - loss: 0.9924 -
acc: 0.6557 - val_loss: 1.0865 - val_acc: 0.6334
Epoch 53/100
50000/50000 [==============================] - 294s 6ms/step - loss: 0.9921 -
acc: 0.6558 - val_loss: 1.0615 - val_acc: 0.6388
Epoch 54/100
50000/50000 [==============================] - 287s 6ms/step - loss: 0.9911 -
acc: 0.6580 - val_loss: 1.1860 - val_acc: 0.6298
Epoch 55/100
50000/50000 [==============================] - 268s 5ms/step - loss: 0.9962 -
acc: 0.6567 - val_loss: 1.1142 - val_acc: 0.6352
```

```
Epoch 56/100
50000/50000 [==============================] - 258s 5ms/step - loss: 0.9894 -
acc: 0.6587 - val_loss: 1.1220 - val_acc: 0.6346
Epoch 57/100
50000/50000 [==============================] - 252s 5ms/step - loss: 0.9938 -
acc: 0.6569 - val_loss: 1.0807 - val_acc: 0.6416
Epoch 58/100
50000/50000 [==============================] - 166s 3ms/step - loss: 0.9980 -
acc: 0.6563 - val_loss: 1.0627 - val_acc: 0.6389
Epoch 59/100
50000/50000 [==============================] - 169s 3ms/step - loss: 0.9940 -
acc: 0.6554 - val_loss: 1.0748 - val_acc: 0.6345
Epoch 60/100
50000/50000 [==============================] - 171s 3ms/step - loss: 0.9933 -
acc: 0.6569 - val_loss: 1.0750 - val_acc: 0.6395
Epoch 61/100
50000/50000 [==============================] - 166s 3ms/step - loss: 0.9945 -
acc: 0.6533 - val_loss: 1.1324 - val_acc: 0.6400
Epoch 62/100
50000/50000 [==============================] - 176s 4ms/step - loss: 0.9955 -
acc: 0.6567 - val_loss: 1.0617 - val_acc: 0.6363
Epoch 63/100
50000/50000 [==============================] - 152s 3ms/step - loss: 0.9957 -
acc: 0.6555 - val_loss: 1.0877 - val_acc: 0.6395
Epoch 64/100
50000/50000 [==============================] - 140s 3ms/step - loss: 0.9926 -
acc: 0.6584 - val_loss: 1.1067 - val_acc: 0.6405
Epoch 65/100
50000/50000 [==============================] - 138s 3ms/step - loss: 0.9881 -
acc: 0.6586 - val_loss: 1.1621 - val_acc: 0.6311
Epoch 66/100
50000/50000 [==============================] - 140s 3ms/step - loss: 0.9913 -
acc: 0.6584 - val_loss: 1.2228 - val_acc: 0.6303
Epoch 67/100
50000/50000 [==============================] - 143s 3ms/step - loss: 0.9997 -
acc: 0.6564 - val_loss: 1.1411 - val_acc: 0.6286
Epoch 68/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.9922 -
acc: 0.6562 - val_loss: 1.2385 - val_acc: 0.6292
Epoch 69/100
50000/50000 [==============================] - 141s 3ms/step - loss: 0.9975 -
acc: 0.6552 - val_loss: 1.1234 - val_acc: 0.6354
Epoch 70/100
50000/50000 [==============================] - 143s 3ms/step - loss: 0.9994 -
acc: 0.6572 - val_loss: 1.2510 - val_acc: 0.6198
Epoch 71/100
50000/50000 [==============================] - 144s 3ms/step - loss: 0.9863 -
acc: 0.6637 - val_loss: 1.1667 - val_acc: 0.6313
```

```
Epoch 72/100
50000/50000 [==============================] - 147s 3ms/step - loss: 0.9986 -
acc: 0.6574 - val_loss: 1.0418 - val_acc: 0.6381
Epoch 73/100
50000/50000 [==============================] - 147s 3ms/step - loss: 0.9960 -
acc: 0.6570 - val_loss: 1.3735 - val_acc: 0.6236
Epoch 74/100
50000/50000 [==============================] - 146s 3ms/step - loss: 0.9937 -
acc: 0.6596 - val_loss: 1.1242 - val_acc: 0.6341
Epoch 75/100
50000/50000 [==============================] - 144s 3ms/step - loss: 0.9960 -
acc: 0.6577 - val_loss: 1.0554 - val_acc: 0.6405
Epoch 76/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.0069 -
acc: 0.6557 - val_loss: 1.0932 - val_acc: 0.6370
Epoch 77/100
50000/50000 [==============================] - 142s 3ms/step - loss: 0.9961 -
acc: 0.6583 - val_loss: 1.1010 - val_acc: 0.6283
Epoch 78/100
50000/50000 [==============================] - 140s 3ms/step - loss: 0.9964 -
acc: 0.6582 - val_loss: 1.1738 - val_acc: 0.6260
Epoch 79/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.0033 -
acc: 0.6542 - val_loss: 1.1098 - val_acc: 0.6371
Epoch 80/100
50000/50000 [==============================] - 139s 3ms/step - loss: 0.9973 -
acc: 0.6560 - val_loss: 1.4138 - val_acc: 0.6139
Epoch 81/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.0045 -
acc: 0.6586 - val_loss: 1.2474 - val_acc: 0.6190
Epoch 82/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.0046 -
acc: 0.6543 - val_loss: 1.1367 - val_acc: 0.6238
Epoch 83/100
50000/50000 [==============================] - 137s 3ms/step - loss: 1.0106 -
acc: 0.6535 - val_loss: 1.0881 - val_acc: 0.6289
Epoch 84/100
50000/50000 [==============================] - 137s 3ms/step - loss: 1.0088 -
acc: 0.6550 - val_loss: 1.0784 - val_acc: 0.6318
Epoch 85/100
50000/50000 [==============================] - 136s 3ms/step - loss: 1.0071 -
acc: 0.6533 - val_loss: 1.1763 - val_acc: 0.6205
Epoch 86/100
50000/50000 [==============================] - 135s 3ms/step - loss: 1.0145 -
acc: 0.6524 - val_loss: 1.1060 - val_acc: 0.6288
Epoch 87/100
50000/50000 [==============================] - 136s 3ms/step - loss: 1.0124 -
acc: 0.6519 - val_loss: 1.0862 - val_acc: 0.6264
```

```
Epoch 88/100
50000/50000 [==============================] - 136s 3ms/step - loss: 1.0126 -
acc: 0.6534 - val_loss: 1.1180 - val_acc: 0.6234
Epoch 89/100
50000/50000 [==============================] - 135s 3ms/step - loss: 1.0222 -
acc: 0.6508 - val_loss: 1.2183 - val_acc: 0.6209
Epoch 90/100
50000/50000 [==============================] - 136s 3ms/step - loss: 1.0177 -
acc: 0.6517 - val_loss: 1.0854 - val_acc: 0.6291
Epoch 91/100
50000/50000 [==============================] - 136s 3ms/step - loss: 1.0192 -
acc: 0.6515 - val_loss: 1.2958 - val_acc: 0.6123
Epoch 92/100
50000/50000 [==============================] - 136s 3ms/step - loss: 1.0221 -
acc: 0.6499 - val_loss: 1.2847 - val_acc: 0.6156
Epoch 93/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.0198 -
acc: 0.6527 - val_loss: 1.2554 - val_acc: 0.6077
Epoch 94/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.0228 -
acc: 0.6480 - val_loss: 1.1985 - val_acc: 0.6191
Epoch 95/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.0221 -
acc: 0.6497 - val_loss: 1.1406 - val_acc: 0.6282
Epoch 96/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.0325 -
acc: 0.6463 - val_loss: 1.1502 - val_acc: 0.6227
Epoch 97/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.0320 -
acc: 0.6497 - val_loss: 1.0811 - val_acc: 0.6295
Epoch 98/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.0305 -
acc: 0.6471 - val_loss: 1.4326 - val_acc: 0.6092
Epoch 99/100
50000/50000 [==============================] - 185s 4ms/step - loss: 1.0358 -
acc: 0.6458 - val_loss: 1.2321 - val_acc: 0.6190
Epoch 100/100
50000/50000 [==============================] - 182s 4ms/step - loss: 1.0350 -
acc: 0.6489 - val_loss: 1.4436 - val_acc: 0.5972
```

[421]:
```
acc_loss_graphic(enc5_history, ' enc_model loss and accuaracy')
```



[422]:
```
# Score trained enc_model.
enc5_scores = enc5_model.evaluate(enc5_X_test, enc5_y_test, verbose=1)
print('Test loss:', enc5_scores[0])
print('Test accuracy:', enc5_scores[1])

# make prediction.
enc_pred = enc5_model.predict(enc5_X_test)
```

```
10000/10000 [==============================] - 7s 716us/step
Test loss: 1.443616304397583
Test accuracy: 0.5972
```

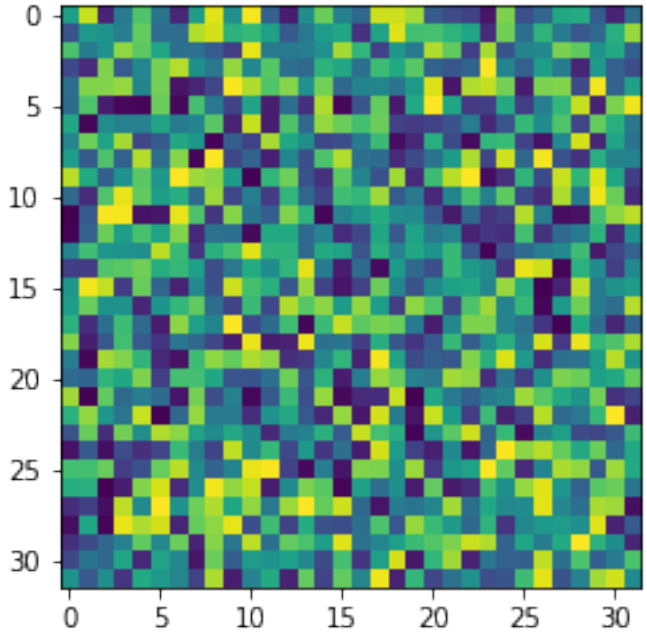
7 Train with encrypted CIFAR10 dataset using fixed key F1

[449]:
```
#load key f1
F1 = np.load("/Users/thiennngole/Desktop/MINES/3Fall2020/MS-project/keys/f1.npy")
#Drive the decryption matrix from the fixed encryption key.
DF1 = np.linalg.inv(F1)
```

Encryption key F1

[450]:
```
plt.imshow(F1)
```

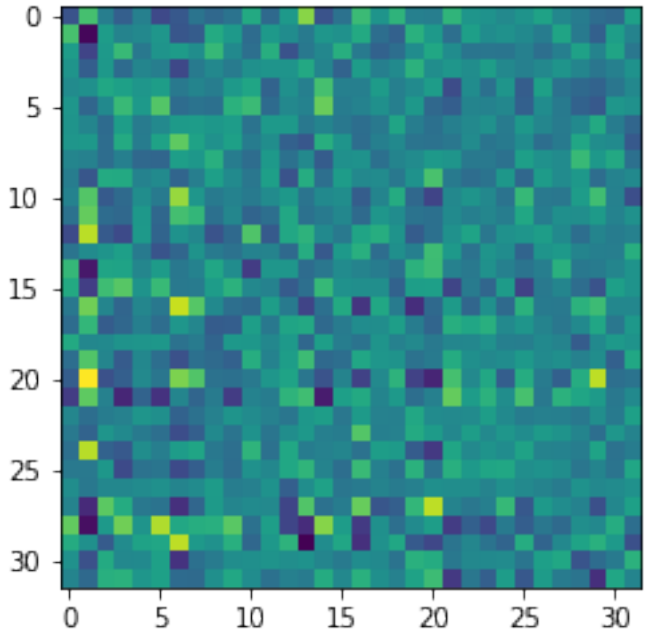
[450]:
```
<matplotlib.image.AxesImage at 0x176ff9b00>
```



Decryption key DF1

[451]:
```
plt.imshow(DF1)
```

[451]:
```
<matplotlib.image.AxesImage at 0x177190a90>
```



[462]:
```
encf1_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    encf1_b = np.dot(b, F1)
    encf1_g = np.dot(g, F1)
    encf1_r = np.dot(r, F1)
    encf1_rgb = cv2.merge((encf1_b,encf1_g,encf1_r))
    encf1_X_train.append(encf1_rgb)
print("Done encrypting ",len(encf1_X_train),"train images")
encf1_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    encf1_b = np.dot(b, F1)
    encf1_g = np.dot(g, F1)
    encf1_r = np.dot(r, F1)
    encf1_rgb = cv2.merge((encf1_b,encf1_g,encf1_r))
    encf1_X_test.append(encf1_rgb)
print("Done encrypting ",len(encf1_X_test),"test images")
```

```
Done encrypting  50000 train images
Done encrypting  10000 test images
```

[463]:
```
# Normalize the data.
encf1_X_train = np.array(encf1_X_train)
encf1_X_test = np.array(encf1_X_test)

encf1_X_train = encf1_X_train.astype('float32')
encf1_X_test = encf1_X_test.astype('float32')
encf1_X_train /= 255
encf1_X_test /= 255

# Convert class vectors to binary class matrices. This is called one hot␣
 ↪encoding.
encf1_y_train = keras.utils.to_categorical(y_train, num_classes)
encf1_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[464]:
```
print('encf1_X_train shape:', encf1_X_train.shape)
print('encf1_y_train shape:', encf1_y_train.shape)
print('encf1_X_test shape:', encf1_X_test.shape)
print('encf1_y_test shape:', encf1_y_test.shape)
```

```
encf1_X_train shape: (50000, 32, 32, 3)
encf1_y_train shape: (50000, 10)
encf1_X_test shape: (10000, 32, 32, 3)
encf1_y_test shape: (10000, 10)
```

[465]:
```
#define the convnet
encf1_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf1_model.add(Conv2D(32, (3, 3), padding='same',input_shape = encf1_X_train.
 ↪shape[1:]))
encf1_model.add(Activation('relu'))
encf1_model.add(Conv2D(32, (3, 3)))
encf1_model.add(Activation('relu'))
encf1_model.add(MaxPooling2D(pool_size=(2, 2)))
encf1_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf1_model.add(Conv2D(64, (3, 3), padding='same'))
encf1_model.add(Activation('relu'))
encf1_model.add(Conv2D(64, (3, 3)))
encf1_model.add(Activation('relu'))
encf1_model.add(MaxPooling2D(pool_size=(2, 2)))
encf1_model.add(Dropout(0.25))

# FLATTERN => DENSE => RELU => DROPOUT
encf1_model.add(Flatten())
encf1_model.add(Dense(512))
encf1_model.add(Activation('relu'))
encf1_model.add(Dropout(0.5))
# a softmax classifier
encf1_model.add(Dense(num_classes))
encf1_model.add(Activation('softmax'))
```

[466]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
encf1_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[506]:
```
#Train model
encf1_history = encf1_model.fit(encf1_X_train, encf1_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(encf1_X_test, encf1_y_test),
              shuffle=True)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 175s 3ms/step - loss: 1.8818 -
acc: 0.3079 - val_loss: 1.7309 - val_acc: 0.3783
```

```
Epoch 2/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.6685 -
acc: 0.3900 - val_loss: 1.6109 - val_acc: 0.4158
Epoch 3/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.5974 -
acc: 0.4186 - val_loss: 1.6725 - val_acc: 0.4027
Epoch 4/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.5574 -
acc: 0.4357 - val_loss: 1.5186 - val_acc: 0.4658
Epoch 5/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.5233 -
acc: 0.4485 - val_loss: 1.5175 - val_acc: 0.4474
Epoch 6/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4978 -
acc: 0.4612 - val_loss: 1.4676 - val_acc: 0.4719
Epoch 7/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.4709 -
acc: 0.4696 - val_loss: 1.4338 - val_acc: 0.4860
Epoch 8/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.4535 -
acc: 0.4794 - val_loss: 1.4348 - val_acc: 0.4812
Epoch 9/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.4317 -
acc: 0.4859 - val_loss: 1.4215 - val_acc: 0.4858
Epoch 10/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.4183 -
acc: 0.4908 - val_loss: 1.4228 - val_acc: 0.4976
Epoch 11/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.4056 -
acc: 0.4971 - val_loss: 1.3802 - val_acc: 0.5042
Epoch 12/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3968 -
acc: 0.5036 - val_loss: 1.4580 - val_acc: 0.4856
Epoch 13/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3823 -
acc: 0.5056 - val_loss: 1.3827 - val_acc: 0.5026
Epoch 14/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.3723 -
acc: 0.5105 - val_loss: 1.4399 - val_acc: 0.4800
Epoch 15/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.3625 -
acc: 0.5129 - val_loss: 1.3862 - val_acc: 0.5180
Epoch 16/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.3597 -
acc: 0.5175 - val_loss: 1.3541 - val_acc: 0.5273
Epoch 17/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3562 -
acc: 0.5169 - val_loss: 1.3707 - val_acc: 0.5300
```

```
Epoch 18/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3460 -
acc: 0.5223 - val_loss: 1.4333 - val_acc: 0.5253
Epoch 19/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3505 -
acc: 0.5230 - val_loss: 1.3276 - val_acc: 0.5219
Epoch 20/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3399 -
acc: 0.5269 - val_loss: 1.3761 - val_acc: 0.5370
Epoch 21/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.3383 -
acc: 0.5259 - val_loss: 1.3872 - val_acc: 0.5305
Epoch 22/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3336 -
acc: 0.5293 - val_loss: 1.3261 - val_acc: 0.5335
Epoch 23/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3341 -
acc: 0.5265 - val_loss: 1.3479 - val_acc: 0.5321
Epoch 24/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3337 -
acc: 0.5318 - val_loss: 1.3195 - val_acc: 0.5311
Epoch 25/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3335 -
acc: 0.5286 - val_loss: 1.4058 - val_acc: 0.5227
Epoch 26/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3307 -
acc: 0.5312 - val_loss: 1.3063 - val_acc: 0.5486
Epoch 27/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3308 -
acc: 0.5327 - val_loss: 1.3824 - val_acc: 0.5284
Epoch 28/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3287 -
acc: 0.5330 - val_loss: 1.3520 - val_acc: 0.5425
Epoch 29/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3253 -
acc: 0.5354 - val_loss: 1.4823 - val_acc: 0.5140
Epoch 30/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3285 -
acc: 0.5341 - val_loss: 1.6398 - val_acc: 0.4446
Epoch 31/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3288 -
acc: 0.5339 - val_loss: 1.3084 - val_acc: 0.5390
Epoch 32/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3263 -
acc: 0.5367 - val_loss: 1.4049 - val_acc: 0.5355
Epoch 33/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3311 -
acc: 0.5337 - val_loss: 1.4243 - val_acc: 0.5370
```

```
Epoch 34/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3271 -
acc: 0.5345 - val_loss: 1.3028 - val_acc: 0.5507
Epoch 35/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3241 -
acc: 0.5376 - val_loss: 1.4466 - val_acc: 0.5229
Epoch 36/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.3360 -
acc: 0.5339 - val_loss: 1.2905 - val_acc: 0.5472
Epoch 37/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3301 -
acc: 0.5367 - val_loss: 1.3576 - val_acc: 0.5375
Epoch 38/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3342 -
acc: 0.5340 - val_loss: 1.3481 - val_acc: 0.5375
Epoch 39/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3356 -
acc: 0.5345 - val_loss: 1.4589 - val_acc: 0.5163
Epoch 40/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3428 -
acc: 0.5316 - val_loss: 1.5697 - val_acc: 0.4982
Epoch 41/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3338 -
acc: 0.5347 - val_loss: 1.2984 - val_acc: 0.5539
Epoch 42/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.3324 -
acc: 0.5359 - val_loss: 1.4260 - val_acc: 0.5016
Epoch 43/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3425 -
acc: 0.5316 - val_loss: 1.4522 - val_acc: 0.5016
Epoch 44/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.3396 -
acc: 0.5320 - val_loss: 1.3462 - val_acc: 0.5269
Epoch 45/100
50000/50000 [==============================] - 138s 3ms/step - loss: 1.3439 -
acc: 0.5304 - val_loss: 1.2834 - val_acc: 0.5464
Epoch 46/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3361 -
acc: 0.5329 - val_loss: 1.3608 - val_acc: 0.5384
Epoch 47/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3424 -
acc: 0.5331 - val_loss: 1.2755 - val_acc: 0.5561
Epoch 48/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3333 -
acc: 0.5354 - val_loss: 1.2916 - val_acc: 0.5426
Epoch 49/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3404 -
acc: 0.5313 - val_loss: 1.4288 - val_acc: 0.5260
```

```
Epoch 50/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3401 -
acc: 0.5330 - val_loss: 1.3592 - val_acc: 0.5146
Epoch 51/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3426 -
acc: 0.5346 - val_loss: 1.2761 - val_acc: 0.5538
Epoch 52/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3427 -
acc: 0.5319 - val_loss: 1.3448 - val_acc: 0.5451
Epoch 53/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3397 -
acc: 0.5350 - val_loss: 1.4046 - val_acc: 0.5259
Epoch 54/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3442 -
acc: 0.5336 - val_loss: 1.3334 - val_acc: 0.5290
Epoch 55/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3444 -
acc: 0.5333 - val_loss: 1.3981 - val_acc: 0.5179
Epoch 56/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3503 -
acc: 0.5328 - val_loss: 1.3273 - val_acc: 0.5443
Epoch 57/100
50000/50000 [==============================] - 139s 3ms/step - loss: 1.3476 -
acc: 0.5314 - val_loss: 1.3926 - val_acc: 0.5151
Epoch 58/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3465 -
acc: 0.5315 - val_loss: 1.2845 - val_acc: 0.5482
Epoch 59/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3446 -
acc: 0.5335 - val_loss: 1.3618 - val_acc: 0.5313
Epoch 60/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3465 -
acc: 0.5332 - val_loss: 1.3246 - val_acc: 0.5500
Epoch 61/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3448 -
acc: 0.5320 - val_loss: 1.4287 - val_acc: 0.5037
Epoch 62/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3533 -
acc: 0.5306 - val_loss: 1.3811 - val_acc: 0.5353
Epoch 63/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3483 -
acc: 0.5297 - val_loss: 1.2607 - val_acc: 0.5621
Epoch 64/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3517 -
acc: 0.5301 - val_loss: 1.3927 - val_acc: 0.5221
Epoch 65/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3452 -
acc: 0.5317 - val_loss: 1.2980 - val_acc: 0.5515
```

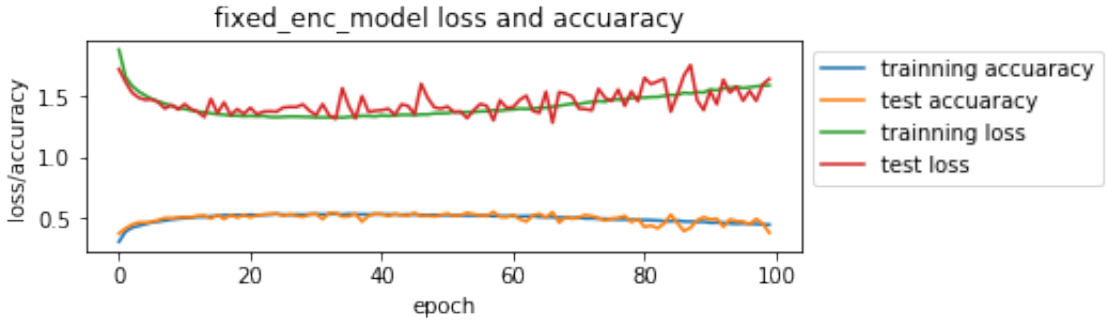
```
Epoch 66/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3540 -
acc: 0.5284 - val_loss: 1.4085 - val_acc: 0.5144
Epoch 67/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3524 -
acc: 0.5295 - val_loss: 1.4018 - val_acc: 0.5458
Epoch 68/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3523 -
acc: 0.5282 - val_loss: 1.3591 - val_acc: 0.5423
Epoch 69/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3562 -
acc: 0.5291 - val_loss: 1.3015 - val_acc: 0.5360
Epoch 70/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3577 -
acc: 0.5312 - val_loss: 1.4740 - val_acc: 0.5173
Epoch 71/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3616 -
acc: 0.5285 - val_loss: 1.4650 - val_acc: 0.5291
Epoch 72/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3577 -
acc: 0.5303 - val_loss: 1.4159 - val_acc: 0.5056
Epoch 73/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3654 -
acc: 0.5263 - val_loss: 1.3666 - val_acc: 0.5241
Epoch 74/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3677 -
acc: 0.5265 - val_loss: 1.5343 - val_acc: 0.4859
Epoch 75/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3683 -
acc: 0.5257 - val_loss: 1.2461 - val_acc: 0.5623
Epoch 76/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3610 -
acc: 0.5279 - val_loss: 1.3595 - val_acc: 0.5372
Epoch 77/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3752 -
acc: 0.5243 - val_loss: 1.3890 - val_acc: 0.5287
Epoch 78/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3738 -
acc: 0.5263 - val_loss: 1.3033 - val_acc: 0.5412
Epoch 79/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3812 -
acc: 0.5210 - val_loss: 1.3820 - val_acc: 0.5288
Epoch 80/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3775 -
acc: 0.5220 - val_loss: 1.2583 - val_acc: 0.5587
Epoch 81/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3734 -
acc: 0.5248 - val_loss: 1.4281 - val_acc: 0.5010
```

```
Epoch 82/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3827 -
acc: 0.5235 - val_loss: 1.4944 - val_acc: 0.4734
Epoch 83/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3870 -
acc: 0.5212 - val_loss: 1.3930 - val_acc: 0.5243
Epoch 84/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3842 -
acc: 0.5212 - val_loss: 1.3893 - val_acc: 0.5150
Epoch 85/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3840 -
acc: 0.5164 - val_loss: 1.3675 - val_acc: 0.5287
Epoch 86/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3917 -
acc: 0.5190 - val_loss: 1.3461 - val_acc: 0.5336
Epoch 87/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3945 -
acc: 0.5184 - val_loss: 1.3467 - val_acc: 0.5308
Epoch 88/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3954 -
acc: 0.5167 - val_loss: 1.3614 - val_acc: 0.5204
Epoch 89/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3967 -
acc: 0.5155 - val_loss: 1.3302 - val_acc: 0.5416
Epoch 90/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3952 -
acc: 0.5144 - val_loss: 1.3814 - val_acc: 0.5060
Epoch 91/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3981 -
acc: 0.5135 - val_loss: 1.3861 - val_acc: 0.5475
Epoch 92/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.4086 -
acc: 0.5153 - val_loss: 1.3626 - val_acc: 0.5304
Epoch 93/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.4074 -
acc: 0.5120 - val_loss: 1.3540 - val_acc: 0.5138
Epoch 94/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.4062 -
acc: 0.5149 - val_loss: 1.5253 - val_acc: 0.4521
Epoch 95/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.4118 -
acc: 0.5111 - val_loss: 1.3829 - val_acc: 0.5228
Epoch 96/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.4159 -
acc: 0.5067 - val_loss: 1.4106 - val_acc: 0.5176
Epoch 97/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.4148 -
acc: 0.5104 - val_loss: 1.6720 - val_acc: 0.4505
```

```
Epoch 98/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.4112 -
acc: 0.5117 - val_loss: 1.3230 - val_acc: 0.5428
Epoch 99/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.4109 -
acc: 0.5084 - val_loss: 1.3004 - val_acc: 0.5373
Epoch 100/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4190 -
acc: 0.5073 - val_loss: 1.3581 - val_acc: 0.5341
```

[435]:
```
def acc_loss_graphic(model_history, title):
    fig = plt.figure()
    plt.subplot(2,1,2)
    plt.plot(model_history.history['acc'], label = 'trainning accuaracy')
    plt.plot(model_history.history['val_acc'], label = 'test accuaracy')
    plt.plot(model_history.history['loss'], label = 'trainning loss')
    plt.plot(model_history.history['val_loss'], label = 'test loss')
    plt.title(title)
    plt.ylabel('loss/accuracy')
    plt.xlabel('epoch')
    plt.legend(bbox_to_anchor=(1,1), loc='upper left', ncol =1)
```

[436]:
```
acc_loss_graphic(encf1_history, ' encf1_model loss and accuaracy')
```



[519]:
```
# Score trained enc_model.
encf1_scores = encf1_model.evaluate(encf1_X_test, encf1_y_test, verbose=1)
print('Test loss:', encf1_scores[0])
print('Test accuracy:', encf1_scores[1])

# make prediction.
encf1_pred = encf1_model.predict(encf1_X_test)
```

```
10000/10000 [==============================] - 7s 705us/step
Test loss: 1.3580538692474364
Test accuracy: 0.5341
```

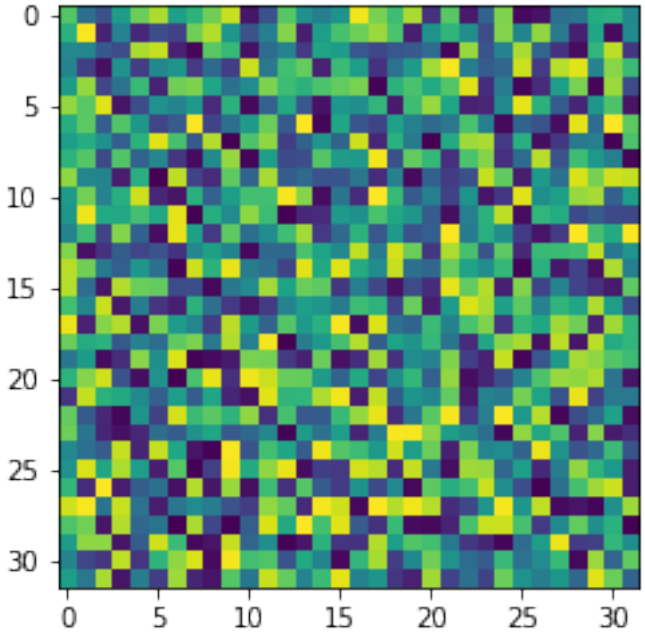
8 Train with encrypted CIFAR10 dataset using fixed key F2

[468]:
```
#load key f2
F2 = np.load("/Users/thiennngole/Desktop/MINES/3Fall2020/MS-project/keys/f2.npy")
#Drive the decryption matrix from the fixed encryption key.
DF2 = np.linalg.inv(F2)
```

Encryption key F2

[469]:
```
plt.imshow(F2)
```

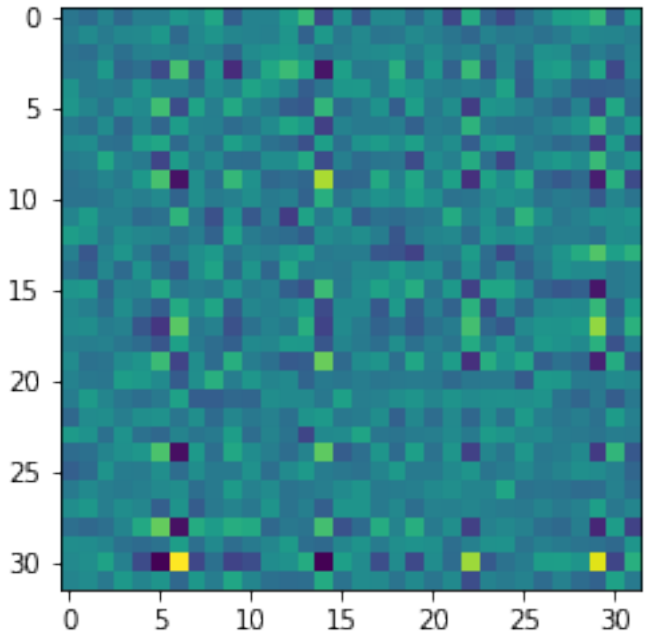
[469]:
```
<matplotlib.image.AxesImage at 0x18bf18a20>
```



Decryption key DF2

[470]:
```
plt.imshow(DF2)
```

[470]:
```
<matplotlib.image.AxesImage at 0x18bf48128>
```



Encrypt the dataset

[472]:
```
encf2_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    encf2_b = np.dot(b, F2)
    encf2_g = np.dot(g, F2)
    encf2_r = np.dot(r, F2)
    encf2_rgb = cv2.merge((encf2_b,encf2_g,encf2_r))
    encf2_X_train.append(encf2_rgb)
print("Done encrypting ",len(encf1_X_train),"train images")
encf2_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    encf2_b = np.dot(b, F2)
    encf2_g = np.dot(g, F2)
    encf2_r = np.dot(r, F2)
    encf2_rgb = cv2.merge((encf2_b,encf2_g,encf2_r))
    encf2_X_test.append(encf2_rgb)
print("Done encrypting ",len(encf2_X_test),"test images")
```

```
Done encrypting  50000 train images
Done encrypting  10000 test images
```

[473]:
```
# Normalize the data.
encf2_X_train = np.array(encf2_X_train)
```

```
encf2_X_test = np.array(encf2_X_test)

encf2_X_train = encf2_X_train.astype('float32')
encf2_X_test = encf2_X_test.astype('float32')
encf2_X_train /= 255
encf2_X_test /= 255

# Convert class vectors to binary class matrices. This is called one hot␣
 ↪encoding.
encf2_y_train = keras.utils.to_categorical(y_train, num_classes)
encf2_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[474]:
```
print('encf2_X_train shape:', encf2_X_train.shape)
print('encf2_y_train shape:', encf2_y_train.shape)
print('encf2_X_test shape:', encf2_X_test.shape)
print('encf2_y_test shape:', encf2_y_test.shape)
```

```
encf2_X_train shape: (50000, 32, 32, 3)
encf2_y_train shape: (50000, 10)
encf2_X_test shape: (10000, 32, 32, 3)
encf2_y_test shape: (10000, 10)
```

[475]:
```
#define the convnet
encf2_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf2_model.add(Conv2D(32, (3, 3), padding='same',input_shape = encf2_X_train.
 ↪shape[1:]))
encf2_model.add(Activation('relu'))
encf2_model.add(Conv2D(32, (3, 3)))
encf2_model.add(Activation('relu'))
encf2_model.add(MaxPooling2D(pool_size=(2, 2)))
encf2_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf2_model.add(Conv2D(64, (3, 3), padding='same'))
encf2_model.add(Activation('relu'))
encf2_model.add(Conv2D(64, (3, 3)))
encf2_model.add(Activation('relu'))
encf2_model.add(MaxPooling2D(pool_size=(2, 2)))
encf2_model.add(Dropout(0.25))

# FLATTERN => DENSE => RELU => DROPOUT
encf2_model.add(Flatten())
encf2_model.add(Dense(512))
encf2_model.add(Activation('relu'))
encf2_model.add(Dropout(0.5))
# a softmax classifier
```

```
encf2_model.add(Dense(num_classes))
encf2_model.add(Activation('softmax'))
```

[476]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
encf2_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[507]:
```
#Train model
encf2_history = encf2_model.fit(encf2_X_train, encf2_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(encf2_X_test, encf2_y_test),
              shuffle=True)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.8369 -
acc: 0.3210 - val_loss: 1.6495 - val_acc: 0.3999
Epoch 2/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.6396 -
acc: 0.4026 - val_loss: 1.5607 - val_acc: 0.4283
Epoch 3/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.5705 -
acc: 0.4301 - val_loss: 1.5235 - val_acc: 0.4484
Epoch 4/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.5293 -
acc: 0.4419 - val_loss: 1.5208 - val_acc: 0.4503
Epoch 5/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.4918 -
acc: 0.4587 - val_loss: 1.5715 - val_acc: 0.4391
Epoch 6/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4683 -
acc: 0.4710 - val_loss: 1.4330 - val_acc: 0.4847
Epoch 7/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.4442 -
acc: 0.4797 - val_loss: 1.4545 - val_acc: 0.4871
Epoch 8/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4197 -
acc: 0.4910 - val_loss: 1.4054 - val_acc: 0.4944
Epoch 9/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4068 -
acc: 0.4958 - val_loss: 1.4141 - val_acc: 0.4972
Epoch 10/100
```

```
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3836 -
acc: 0.5068 - val_loss: 1.4442 - val_acc: 0.4818
Epoch 11/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3738 -
acc: 0.5085 - val_loss: 1.3804 - val_acc: 0.5145
Epoch 12/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3649 -
acc: 0.5146 - val_loss: 1.3482 - val_acc: 0.5147
Epoch 13/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3546 -
acc: 0.5166 - val_loss: 1.3592 - val_acc: 0.5175
Epoch 14/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3459 -
acc: 0.5223 - val_loss: 1.3541 - val_acc: 0.5151
Epoch 15/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3387 -
acc: 0.5235 - val_loss: 1.3121 - val_acc: 0.5338
Epoch 16/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.3326 -
acc: 0.5272 - val_loss: 1.3350 - val_acc: 0.5221
Epoch 17/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3224 -
acc: 0.5309 - val_loss: 1.3941 - val_acc: 0.5227
Epoch 18/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3179 -
acc: 0.5332 - val_loss: 1.3520 - val_acc: 0.5273
Epoch 19/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3173 -
acc: 0.5337 - val_loss: 1.3106 - val_acc: 0.5375
Epoch 20/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3156 -
acc: 0.5376 - val_loss: 1.3405 - val_acc: 0.5360
Epoch 21/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.3104 -
acc: 0.5374 - val_loss: 1.2813 - val_acc: 0.5379
Epoch 22/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3063 -
acc: 0.5371 - val_loss: 1.3185 - val_acc: 0.5290
Epoch 23/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3013 -
acc: 0.5392 - val_loss: 1.3372 - val_acc: 0.5251
Epoch 24/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.3038 -
acc: 0.5404 - val_loss: 1.3286 - val_acc: 0.5409
Epoch 25/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3023 -
acc: 0.5441 - val_loss: 1.3640 - val_acc: 0.5304
Epoch 26/100
```

```
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3030 -
acc: 0.5423 - val_loss: 1.3087 - val_acc: 0.5355
Epoch 27/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2965 -
acc: 0.5418 - val_loss: 1.3142 - val_acc: 0.5473
Epoch 28/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3040 -
acc: 0.5441 - val_loss: 1.3739 - val_acc: 0.5365
Epoch 29/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3057 -
acc: 0.5440 - val_loss: 1.3656 - val_acc: 0.5486
Epoch 30/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3045 -
acc: 0.5408 - val_loss: 1.2976 - val_acc: 0.5481
Epoch 31/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3073 -
acc: 0.5413 - val_loss: 1.3593 - val_acc: 0.5371
Epoch 32/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3065 -
acc: 0.5439 - val_loss: 1.4126 - val_acc: 0.5220
Epoch 33/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3054 -
acc: 0.5433 - val_loss: 1.2678 - val_acc: 0.5501
Epoch 34/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3057 -
acc: 0.5431 - val_loss: 1.3437 - val_acc: 0.5421
Epoch 35/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3080 -
acc: 0.5428 - val_loss: 1.2963 - val_acc: 0.5518
Epoch 36/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3057 -
acc: 0.5453 - val_loss: 1.3700 - val_acc: 0.5456
Epoch 37/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3022 -
acc: 0.5443 - val_loss: 1.3528 - val_acc: 0.5309
Epoch 38/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3031 -
acc: 0.5446 - val_loss: 1.4139 - val_acc: 0.5111
Epoch 39/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3055 -
acc: 0.5438 - val_loss: 1.3151 - val_acc: 0.5515
Epoch 40/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3055 -
acc: 0.5453 - val_loss: 1.4054 - val_acc: 0.5334
Epoch 41/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3078 -
acc: 0.5457 - val_loss: 1.2732 - val_acc: 0.5589
Epoch 42/100
```

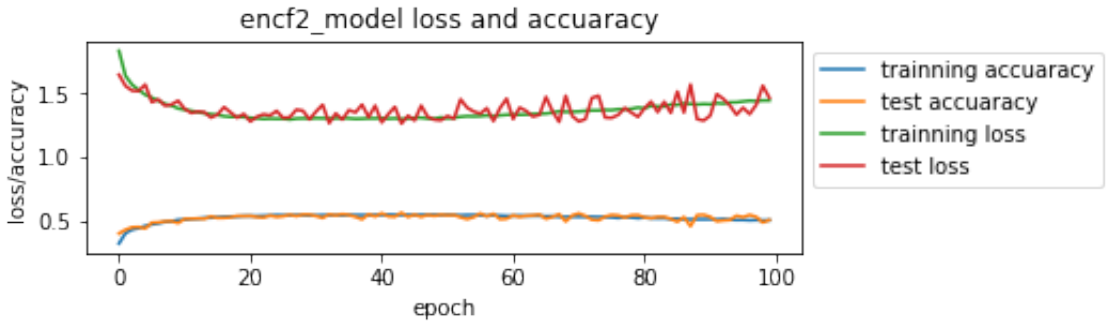
```
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3033 -
acc: 0.5476 - val_loss: 1.3392 - val_acc: 0.5345
Epoch 43/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3057 -
acc: 0.5462 - val_loss: 1.3963 - val_acc: 0.5317
Epoch 44/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.3000 -
acc: 0.5455 - val_loss: 1.2643 - val_acc: 0.5632
Epoch 45/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.3045 -
acc: 0.5463 - val_loss: 1.3266 - val_acc: 0.5297
Epoch 46/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.3105 -
acc: 0.5436 - val_loss: 1.2888 - val_acc: 0.5483
Epoch 47/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.3105 -
acc: 0.5463 - val_loss: 1.4058 - val_acc: 0.5305
Epoch 48/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.3046 -
acc: 0.5460 - val_loss: 1.3177 - val_acc: 0.5460
Epoch 49/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.3026 -
acc: 0.5471 - val_loss: 1.2954 - val_acc: 0.5381
Epoch 50/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.3135 -
acc: 0.5443 - val_loss: 1.2907 - val_acc: 0.5475
Epoch 51/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3056 -
acc: 0.5473 - val_loss: 1.3245 - val_acc: 0.5427
Epoch 52/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3109 -
acc: 0.5449 - val_loss: 1.3030 - val_acc: 0.5453
Epoch 53/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3183 -
acc: 0.5426 - val_loss: 1.4559 - val_acc: 0.5270
Epoch 54/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3228 -
acc: 0.5450 - val_loss: 1.3898 - val_acc: 0.5132
Epoch 55/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3216 -
acc: 0.5450 - val_loss: 1.3594 - val_acc: 0.5285
Epoch 56/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3233 -
acc: 0.5431 - val_loss: 1.3423 - val_acc: 0.5577
Epoch 57/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3284 -
acc: 0.5385 - val_loss: 1.3854 - val_acc: 0.5343
Epoch 58/100
```

```
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3227 -
acc: 0.5429 - val_loss: 1.2845 - val_acc: 0.5548
Epoch 59/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3294 -
acc: 0.5401 - val_loss: 1.3864 - val_acc: 0.5125
Epoch 60/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3336 -
acc: 0.5380 - val_loss: 1.4605 - val_acc: 0.5161
Epoch 61/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3366 -
acc: 0.5377 - val_loss: 1.3005 - val_acc: 0.5420
Epoch 62/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3342 -
acc: 0.5405 - val_loss: 1.3904 - val_acc: 0.5298
Epoch 63/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3360 -
acc: 0.5373 - val_loss: 1.3635 - val_acc: 0.5392
Epoch 64/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3400 -
acc: 0.5383 - val_loss: 1.3198 - val_acc: 0.5423
Epoch 65/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3412 -
acc: 0.5386 - val_loss: 1.3327 - val_acc: 0.5487
Epoch 66/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3436 -
acc: 0.5349 - val_loss: 1.4740 - val_acc: 0.5155
Epoch 67/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3504 -
acc: 0.5364 - val_loss: 1.3606 - val_acc: 0.5302
Epoch 68/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3618 -
acc: 0.5315 - val_loss: 1.2785 - val_acc: 0.5519
Epoch 69/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3565 -
acc: 0.5297 - val_loss: 1.4829 - val_acc: 0.4942
Epoch 70/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3574 -
acc: 0.5323 - val_loss: 1.3243 - val_acc: 0.5416
Epoch 71/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.3599 -
acc: 0.5312 - val_loss: 1.2809 - val_acc: 0.5577
Epoch 72/100
50000/50000 [==============================] - 160s 3ms/step - loss: 1.3647 -
acc: 0.5310 - val_loss: 1.2997 - val_acc: 0.5375
Epoch 73/100
50000/50000 [==============================] - 167s 3ms/step - loss: 1.3679 -
acc: 0.5303 - val_loss: 1.4665 - val_acc: 0.5080
Epoch 74/100
```

```
50000/50000 [==============================] - 157s 3ms/step - loss: 1.3678 -
acc: 0.5292 - val_loss: 1.4837 - val_acc: 0.5036
Epoch 75/100
50000/50000 [==============================] - 166s 3ms/step - loss: 1.3719 -
acc: 0.5293 - val_loss: 1.3122 - val_acc: 0.5321
Epoch 76/100
50000/50000 [==============================] - 179s 4ms/step - loss: 1.3706 -
acc: 0.5269 - val_loss: 1.3094 - val_acc: 0.5454
Epoch 77/100
50000/50000 [==============================] - 179s 4ms/step - loss: 1.3775 -
acc: 0.5220 - val_loss: 1.3335 - val_acc: 0.5415
Epoch 78/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.3769 -
acc: 0.5276 - val_loss: 1.3876 - val_acc: 0.5251
Epoch 79/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.3807 -
acc: 0.5257 - val_loss: 1.3509 - val_acc: 0.5428
Epoch 80/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.3909 -
acc: 0.5204 - val_loss: 1.3174 - val_acc: 0.5442
Epoch 81/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.3844 -
acc: 0.5247 - val_loss: 1.3851 - val_acc: 0.5266
Epoch 82/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.4040 -
acc: 0.5199 - val_loss: 1.4374 - val_acc: 0.5209
Epoch 83/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.3995 -
acc: 0.5188 - val_loss: 1.3547 - val_acc: 0.5301
Epoch 84/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.3992 -
acc: 0.5196 - val_loss: 1.4362 - val_acc: 0.5301
Epoch 85/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.4156 -
acc: 0.5143 - val_loss: 1.3491 - val_acc: 0.5144
Epoch 86/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.4110 -
acc: 0.5150 - val_loss: 1.5166 - val_acc: 0.4916
Epoch 87/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.4235 -
acc: 0.5118 - val_loss: 1.3517 - val_acc: 0.5289
Epoch 88/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.4190 -
acc: 0.5111 - val_loss: 1.5721 - val_acc: 0.4569
Epoch 89/100
50000/50000 [==============================] - 169s 3ms/step - loss: 1.4149 -
acc: 0.5114 - val_loss: 1.3003 - val_acc: 0.5441
Epoch 90/100
```

```
50000/50000 [==============================] - 181s 4ms/step - loss: 1.4188 -
acc: 0.5114 - val_loss: 1.2882 - val_acc: 0.5461
Epoch 91/100
50000/50000 [==============================] - 171s 3ms/step - loss: 1.4166 -
acc: 0.5121 - val_loss: 1.3271 - val_acc: 0.5304
Epoch 92/100
50000/50000 [==============================] - 180s 4ms/step - loss: 1.4253 -
acc: 0.5096 - val_loss: 1.4962 - val_acc: 0.4962
Epoch 93/100
50000/50000 [==============================] - 182s 4ms/step - loss: 1.4214 -
acc: 0.5131 - val_loss: 1.4585 - val_acc: 0.5024
Epoch 94/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.4316 -
acc: 0.5075 - val_loss: 1.4065 - val_acc: 0.5058
Epoch 95/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.4324 -
acc: 0.5061 - val_loss: 1.3347 - val_acc: 0.5390
Epoch 96/100
50000/50000 [==============================] - 161s 3ms/step - loss: 1.4389 -
acc: 0.5065 - val_loss: 1.3894 - val_acc: 0.5278
Epoch 97/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.4445 -
acc: 0.5037 - val_loss: 1.3397 - val_acc: 0.5454
Epoch 98/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.4413 -
acc: 0.5057 - val_loss: 1.4118 - val_acc: 0.5301
Epoch 99/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.4464 -
acc: 0.5030 - val_loss: 1.5623 - val_acc: 0.4863
Epoch 100/100
50000/50000 [==============================] - 163s 3ms/step - loss: 1.4461 -
acc: 0.5043 - val_loss: 1.4675 - val_acc: 0.5061
```

[511]:
```
acc_loss_graphic(encf2_history, ' encf2_model loss and accuaracy')
```



[518]:
```
# Score trained enc_model.
encf2_scores = encf2_model.evaluate(encf2_X_test, encf2_y_test, verbose=1)
print('Test loss:', encf2_scores[0])
print('Test accuracy:', encf2_scores[1])

# make prediction.
encf2_pred = encf2_model.predict(encf2_X_test)
```

```
10000/10000 [==============================] - 7s 709us/step
Test loss: 1.4674673274993897
Test accuracy: 0.5061
```

[ ]:
```

```

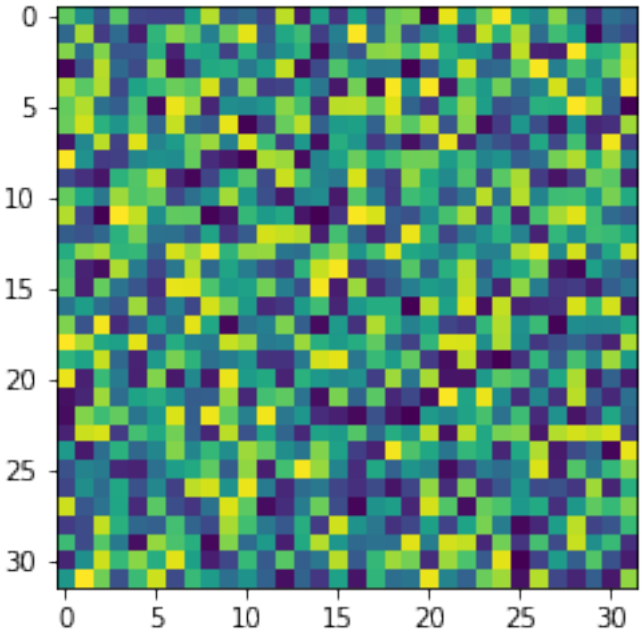
9 Train with encrypted CIFAR10 dataset using fixed key F3

[477]:
```
#load key f3
F3 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f3.npy")
#Drive the decryption matrix from the fixed encryption key.
DF3 = np.linalg.inv(F3)
```

Encryption key F3

[479]:
```
plt.imshow(F3)
```

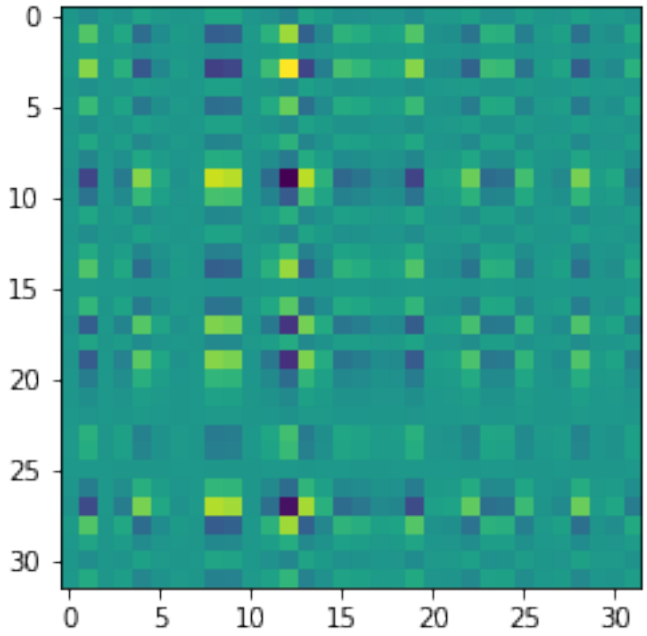
[479]:
```
<matplotlib.image.AxesImage at 0x198ca38d0>
```



Decryption key DF3

[482]:
```
plt.imshow(DF3)
```

[482]:
```
<matplotlib.image.AxesImage at 0x198e045c0>
```



Encrypt the dataset

[483]:
```
encf3_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    encf3_b = np.dot(b, F3)
    encf3_g = np.dot(g, F3)
    encf3_r = np.dot(r, F3)
    encf3_rgb = cv2.merge((encf3_b,encf3_g,encf3_r))
    encf3_X_train.append(encf3_rgb)
print("Done encrypting ",len(encf3_X_train),"train images")
encf3_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    encf3_b = np.dot(b, F3)
    encf3_g = np.dot(g, F3)
    encf3_r = np.dot(r, F3)
    encf3_rgb = cv2.merge((encf3_b,encf3_g,encf3_r))
```

```
    encf3_X_test.append(encf3_rgb)
print("Done encrypting ",len(encf3_X_test),"test images")
```

```
Done encrypting  50000 train images
Done encrypting  10000 test images
```

[484]:
```
# Normalize the data.
encf3_X_train = np.array(encf3_X_train)
encf3_X_test = np.array(encf3_X_test)

encf3_X_train = encf3_X_train.astype('float32')
encf3_X_test = encf3_X_test.astype('float32')
encf3_X_train /= 255
encf3_X_test /= 255

# Convert class vectors to binary class matrices. This is called one hot␣
 ↪encoding.
encf3_y_train = keras.utils.to_categorical(y_train, num_classes)
encf3_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[485]:
```
print('encf3_X_train shape:', encf3_X_train.shape)
print('encf3_y_train shape:', encf3_y_train.shape)
print('encf3_X_test shape:', encf3_X_test.shape)
print('encf3_y_test shape:', encf3_y_test.shape)
```

```
encf3_X_train shape: (50000, 32, 32, 3)
encf3_y_train shape: (50000, 10)
encf3_X_test shape: (10000, 32, 32, 3)
encf3_y_test shape: (10000, 10)
```

[486]:
```
#define the convnet
encf3_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf3_model.add(Conv2D(32, (3, 3), padding='same',input_shape = encf3_X_train.
 ↪shape[1:]))
encf3_model.add(Activation('relu'))
encf3_model.add(Conv2D(32, (3, 3)))
encf3_model.add(Activation('relu'))
encf3_model.add(MaxPooling2D(pool_size=(2, 2)))
encf3_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf3_model.add(Conv2D(64, (3, 3), padding='same'))
encf3_model.add(Activation('relu'))
encf3_model.add(Conv2D(64, (3, 3)))
encf3_model.add(Activation('relu'))
encf3_model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
encf3_model.add(Dropout(0.25))

# FLATTERN => DENSE => RELU => DROPOUT
encf3_model.add(Flatten())
encf3_model.add(Dense(512))
encf3_model.add(Activation('relu'))
encf3_model.add(Dropout(0.5))
# a softmax classifier
encf3_model.add(Dense(num_classes))
encf3_model.add(Activation('softmax'))
```

[487]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
encf3_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[508]:
```
#Train model
encf3_history = encf3_model.fit(encf3_X_train, encf3_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(encf3_X_test, encf3_y_test),
              shuffle=True)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 167s 3ms/step - loss: 1.8769 -
acc: 0.3086 - val_loss: 1.6745 - val_acc: 0.4069
Epoch 2/100
50000/50000 [==============================] - 173s 3ms/step - loss: 1.6610 -
acc: 0.3927 - val_loss: 1.6491 - val_acc: 0.4061
Epoch 3/100
50000/50000 [==============================] - 178s 4ms/step - loss: 1.5797 -
acc: 0.4264 - val_loss: 1.5048 - val_acc: 0.4524
Epoch 4/100
50000/50000 [==============================] - 175s 4ms/step - loss: 1.5250 -
acc: 0.4481 - val_loss: 1.4907 - val_acc: 0.4692
Epoch 5/100
50000/50000 [==============================] - 175s 3ms/step - loss: 1.4889 -
acc: 0.4624 - val_loss: 1.4804 - val_acc: 0.4680
Epoch 6/100
50000/50000 [==============================] - 167s 3ms/step - loss: 1.4574 -
acc: 0.4740 - val_loss: 1.4294 - val_acc: 0.4884
Epoch 7/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4307 -
```

```
acc: 0.4857 - val_loss: 1.3953 - val_acc: 0.5040
Epoch 8/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.4085 -
acc: 0.4945 - val_loss: 1.3780 - val_acc: 0.5066
Epoch 9/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.3910 -
acc: 0.5026 - val_loss: 1.4196 - val_acc: 0.4820
Epoch 10/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.3719 -
acc: 0.5067 - val_loss: 1.3566 - val_acc: 0.5210
Epoch 11/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3595 -
acc: 0.5139 - val_loss: 1.3997 - val_acc: 0.4916
Epoch 12/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.3497 -
acc: 0.5202 - val_loss: 1.3434 - val_acc: 0.5189
Epoch 13/100
50000/50000 [==============================] - 178s 4ms/step - loss: 1.3337 -
acc: 0.5254 - val_loss: 1.3594 - val_acc: 0.5171
Epoch 14/100
50000/50000 [==============================] - 191s 4ms/step - loss: 1.3291 -
acc: 0.5267 - val_loss: 1.3522 - val_acc: 0.5131
Epoch 15/100
50000/50000 [==============================] - 185s 4ms/step - loss: 1.3193 -
acc: 0.5316 - val_loss: 1.3225 - val_acc: 0.5313
Epoch 16/100
50000/50000 [==============================] - 212s 4ms/step - loss: 1.3165 -
acc: 0.5337 - val_loss: 1.3472 - val_acc: 0.5212
Epoch 17/100
50000/50000 [==============================] - 197s 4ms/step - loss: 1.3063 -
acc: 0.5365 - val_loss: 1.3232 - val_acc: 0.5310
Epoch 18/100
50000/50000 [==============================] - 176s 4ms/step - loss: 1.3019 -
acc: 0.5363 - val_loss: 1.3874 - val_acc: 0.5300
Epoch 19/100
50000/50000 [==============================] - 173s 3ms/step - loss: 1.2982 -
acc: 0.5411 - val_loss: 1.2915 - val_acc: 0.5452
Epoch 20/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.2990 -
acc: 0.5428 - val_loss: 1.3809 - val_acc: 0.5188
Epoch 21/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.2942 -
acc: 0.5438 - val_loss: 1.3397 - val_acc: 0.5435
Epoch 22/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.2932 -
acc: 0.5448 - val_loss: 1.3472 - val_acc: 0.5376
Epoch 23/100
50000/50000 [==============================] - 166s 3ms/step - loss: 1.2873 -
```

```
acc: 0.5449 - val_loss: 1.2953 - val_acc: 0.5490
Epoch 24/100
50000/50000 [==============================] - 167s 3ms/step - loss: 1.2851 -
acc: 0.5497 - val_loss: 1.3371 - val_acc: 0.5269
Epoch 25/100
50000/50000 [==============================] - 173s 3ms/step - loss: 1.2830 -
acc: 0.5456 - val_loss: 1.2984 - val_acc: 0.5385
Epoch 26/100
50000/50000 [==============================] - 163s 3ms/step - loss: 1.2803 -
acc: 0.5503 - val_loss: 1.2742 - val_acc: 0.5571
Epoch 27/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.2818 -
acc: 0.5466 - val_loss: 1.3193 - val_acc: 0.5381
Epoch 28/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.2747 -
acc: 0.5501 - val_loss: 1.3077 - val_acc: 0.5487
Epoch 29/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.2778 -
acc: 0.5512 - val_loss: 1.3312 - val_acc: 0.5491
Epoch 30/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.2821 -
acc: 0.5508 - val_loss: 1.5342 - val_acc: 0.4829
Epoch 31/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.2827 -
acc: 0.5525 - val_loss: 1.3693 - val_acc: 0.5235
Epoch 32/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.2803 -
acc: 0.5527 - val_loss: 1.3306 - val_acc: 0.5531
Epoch 33/100
50000/50000 [==============================] - 176s 4ms/step - loss: 1.2807 -
acc: 0.5525 - val_loss: 1.3628 - val_acc: 0.5203
Epoch 34/100
50000/50000 [==============================] - 158s 3ms/step - loss: 1.2765 -
acc: 0.5538 - val_loss: 1.2583 - val_acc: 0.5586
Epoch 35/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.2842 -
acc: 0.5515 - val_loss: 1.3057 - val_acc: 0.5523
Epoch 36/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.2802 -
acc: 0.5530 - val_loss: 1.3894 - val_acc: 0.5487
Epoch 37/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.2782 -
acc: 0.5555 - val_loss: 1.2878 - val_acc: 0.5581
Epoch 38/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.2804 -
acc: 0.5521 - val_loss: 1.2460 - val_acc: 0.5730
Epoch 39/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.2805 -
```

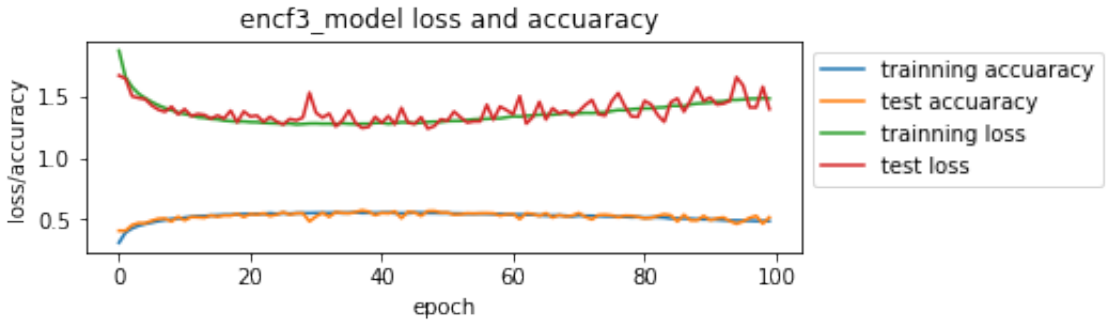
```
acc: 0.5526 - val_loss: 1.2543 - val_acc: 0.5624
Epoch 40/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2868 -
acc: 0.5521 - val_loss: 1.3366 - val_acc: 0.5333
Epoch 41/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2879 -
acc: 0.5535 - val_loss: 1.2877 - val_acc: 0.5523
Epoch 42/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.2798 -
acc: 0.5552 - val_loss: 1.3427 - val_acc: 0.5483
Epoch 43/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.2854 -
acc: 0.5504 - val_loss: 1.2753 - val_acc: 0.5642
Epoch 44/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.2926 -
acc: 0.5496 - val_loss: 1.4095 - val_acc: 0.5093
Epoch 45/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.2913 -
acc: 0.5499 - val_loss: 1.2888 - val_acc: 0.5620
Epoch 46/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.2826 -
acc: 0.5545 - val_loss: 1.2793 - val_acc: 0.5551
Epoch 47/100
50000/50000 [==============================] - 173s 3ms/step - loss: 1.2896 -
acc: 0.5542 - val_loss: 1.3337 - val_acc: 0.5280
Epoch 48/100
50000/50000 [==============================] - 160s 3ms/step - loss: 1.2944 -
acc: 0.5513 - val_loss: 1.2404 - val_acc: 0.5668
Epoch 49/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.2973 -
acc: 0.5512 - val_loss: 1.2609 - val_acc: 0.5676
Epoch 50/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.2954 -
acc: 0.5499 - val_loss: 1.3165 - val_acc: 0.5552
Epoch 51/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.3021 -
acc: 0.5505 - val_loss: 1.3041 - val_acc: 0.5546
Epoch 52/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.3035 -
acc: 0.5480 - val_loss: 1.3795 - val_acc: 0.5380
Epoch 53/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.3036 -
acc: 0.5476 - val_loss: 1.3324 - val_acc: 0.5407
Epoch 54/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.3105 -
acc: 0.5456 - val_loss: 1.2896 - val_acc: 0.5486
Epoch 55/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.3160 -
```

```
acc: 0.5457 - val_loss: 1.2995 - val_acc: 0.5469
Epoch 56/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3182 -
acc: 0.5467 - val_loss: 1.2998 - val_acc: 0.5486
Epoch 57/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3203 -
acc: 0.5454 - val_loss: 1.4356 - val_acc: 0.5500
Epoch 58/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3246 -
acc: 0.5438 - val_loss: 1.3128 - val_acc: 0.5484
Epoch 59/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3238 -
acc: 0.5435 - val_loss: 1.4204 - val_acc: 0.5298
Epoch 60/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3329 -
acc: 0.5397 - val_loss: 1.3916 - val_acc: 0.5411
Epoch 61/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3417 -
acc: 0.5384 - val_loss: 1.3581 - val_acc: 0.5441
Epoch 62/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3387 -
acc: 0.5389 - val_loss: 1.5060 - val_acc: 0.5019
Epoch 63/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3444 -
acc: 0.5369 - val_loss: 1.2807 - val_acc: 0.5500
Epoch 64/100
50000/50000 [==============================] - 141s 3ms/step - loss: 1.3485 -
acc: 0.5365 - val_loss: 1.3362 - val_acc: 0.5433
Epoch 65/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.3539 -
acc: 0.5335 - val_loss: 1.4591 - val_acc: 0.5277
Epoch 66/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3550 -
acc: 0.5298 - val_loss: 1.3153 - val_acc: 0.5502
Epoch 67/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.3538 -
acc: 0.5319 - val_loss: 1.4068 - val_acc: 0.5141
Epoch 68/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.3609 -
acc: 0.5325 - val_loss: 1.3644 - val_acc: 0.5350
Epoch 69/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.3665 -
acc: 0.5297 - val_loss: 1.3841 - val_acc: 0.5227
Epoch 70/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.3684 -
acc: 0.5244 - val_loss: 1.3431 - val_acc: 0.5303
Epoch 71/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.3654 -
```

```
acc: 0.5301 - val_loss: 1.3818 - val_acc: 0.5513
Epoch 72/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.3687 -
acc: 0.5290 - val_loss: 1.4338 - val_acc: 0.5198
Epoch 73/100
50000/50000 [==============================] - 166s 3ms/step - loss: 1.3676 -
acc: 0.5281 - val_loss: 1.4746 - val_acc: 0.4989
Epoch 74/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.3707 -
acc: 0.5276 - val_loss: 1.3585 - val_acc: 0.5349
Epoch 75/100
50000/50000 [==============================] - 162s 3ms/step - loss: 1.3734 -
acc: 0.5247 - val_loss: 1.3472 - val_acc: 0.5370
Epoch 76/100
50000/50000 [==============================] - 164s 3ms/step - loss: 1.3838 -
acc: 0.5242 - val_loss: 1.4640 - val_acc: 0.5319
Epoch 77/100
50000/50000 [==============================] - 168s 3ms/step - loss: 1.3925 -
acc: 0.5196 - val_loss: 1.5311 - val_acc: 0.5164
Epoch 78/100
50000/50000 [==============================] - 163s 3ms/step - loss: 1.3903 -
acc: 0.5203 - val_loss: 1.4242 - val_acc: 0.5298
Epoch 79/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.3986 -
acc: 0.5169 - val_loss: 1.3404 - val_acc: 0.5284
Epoch 80/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.4031 -
acc: 0.5187 - val_loss: 1.3368 - val_acc: 0.5192
Epoch 81/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.4053 -
acc: 0.5138 - val_loss: 1.4670 - val_acc: 0.5086
Epoch 82/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.4056 -
acc: 0.5160 - val_loss: 1.4384 - val_acc: 0.5117
Epoch 83/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.4131 -
acc: 0.5159 - val_loss: 1.3513 - val_acc: 0.5276
Epoch 84/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.4147 -
acc: 0.5137 - val_loss: 1.2961 - val_acc: 0.5408
Epoch 85/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.4274 -
acc: 0.5064 - val_loss: 1.4569 - val_acc: 0.5323
Epoch 86/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.4266 -
acc: 0.5071 - val_loss: 1.4867 - val_acc: 0.4826
Epoch 87/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4238 -
```

```
acc: 0.5069 - val_loss: 1.3788 - val_acc: 0.5366
Epoch 88/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4377 -
acc: 0.5039 - val_loss: 1.4712 - val_acc: 0.4913
Epoch 89/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4454 -
acc: 0.5006 - val_loss: 1.5751 - val_acc: 0.4911
Epoch 90/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4482 -
acc: 0.4996 - val_loss: 1.4655 - val_acc: 0.5275
Epoch 91/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4525 -
acc: 0.4996 - val_loss: 1.4950 - val_acc: 0.4927
Epoch 92/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4607 -
acc: 0.4962 - val_loss: 1.4377 - val_acc: 0.5069
Epoch 93/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4634 -
acc: 0.4928 - val_loss: 1.4450 - val_acc: 0.5150
Epoch 94/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4746 -
acc: 0.4903 - val_loss: 1.4666 - val_acc: 0.4858
Epoch 95/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4748 -
acc: 0.4902 - val_loss: 1.6605 - val_acc: 0.4632
Epoch 96/100
50000/50000 [==============================] - 140s 3ms/step - loss: 1.4783 -
acc: 0.4887 - val_loss: 1.5936 - val_acc: 0.4833
Epoch 97/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4848 -
acc: 0.4887 - val_loss: 1.4145 - val_acc: 0.5114
Epoch 98/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.4853 -
acc: 0.4862 - val_loss: 1.4132 - val_acc: 0.5277
Epoch 99/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.4887 -
acc: 0.4861 - val_loss: 1.5818 - val_acc: 0.4647
Epoch 100/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.4867 -
acc: 0.4862 - val_loss: 1.4002 - val_acc: 0.5160
```

[516]:
```
acc_loss_graphic(encf3_history, ' encf3_model loss and accuaracy')
```



[517]:
```
# Score trained enc_model.
encf3_scores = encf3_model.evaluate(encf3_X_test, encf3_y_test, verbose=1)
print('Test loss:', encf3_scores[0])
print('Test accuracy:', encf3_scores[1])

# make prediction.
encf3_pred = encf3_model.predict(encf3_X_test)
```

```
10000/10000 [==============================] - 7s 671us/step
Test loss: 1.4001767147064208
Test accuracy: 0.516
```

[ ]:
```
```

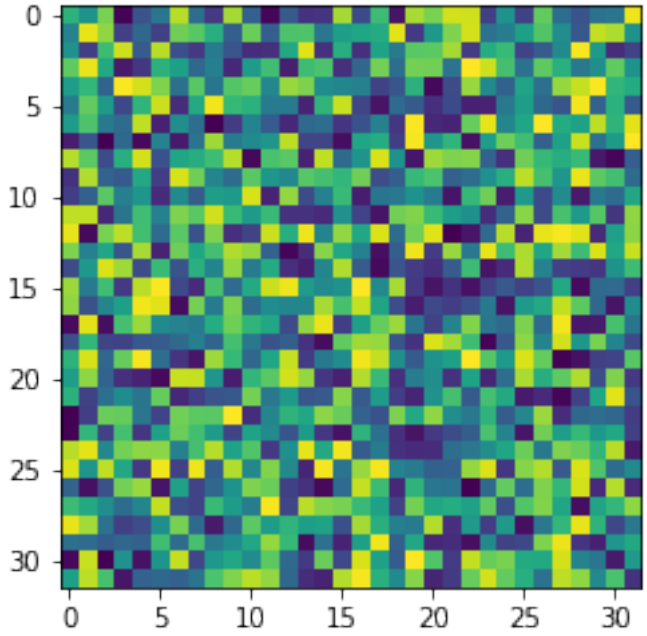
10 Train with encrypted CIFAR10 dataset using fixed key F4

[488]:
```
#load key f4
F4 = np.load("/Users/thiennngole/Desktop/MINES/3Fall2020/MS-project/keys/f4.npy")
#Drive the decryption matrix from the fixed encryption key.
DF4 = np.linalg.inv(F4)
```

Encryption key F4

[490]:
```
plt.imshow(F4)
```

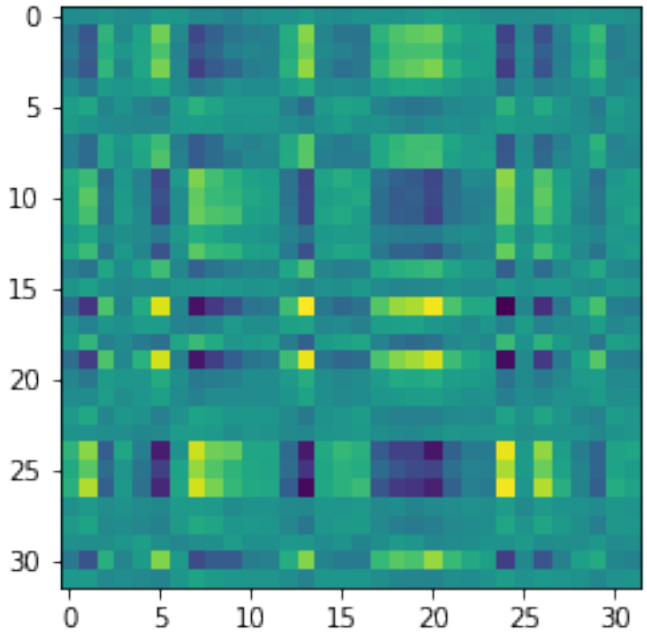
[490]:
```
<matplotlib.image.AxesImage at 0x199231e48>
```



Decryption key DF4

[491]:
```
plt.imshow(DF4)
```

[491]:
```
<matplotlib.image.AxesImage at 0x1992ee4a8>
```



Encrypt the dataset

[492]:
```
encf4_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    encf4_b = np.dot(b, F4)
    encf4_g = np.dot(g, F4)
    encf4_r = np.dot(r, F4)
    encf4_rgb = cv2.merge((encf4_b,encf4_g,encf4_r))
    encf4_X_train.append(encf4_rgb)
print("Done encrypting ",len(encf4_X_train),"train images")
encf4_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    encf4_b = np.dot(b, F4)
    encf4_g = np.dot(g, F4)
    encf4_r = np.dot(r, F4)
    encf4_rgb = cv2.merge((encf4_b,encf4_g,encf4_r))
    encf4_X_test.append(encf4_rgb)
print("Done encrypting ",len(encf4_X_test),"test images")
```

```
Done encrypting  50000 train images
Done encrypting  10000 test images
```

[493]:
```
# Normalize the data.
encf4_X_train = np.array(encf4_X_train)
encf4_X_test = np.array(encf4_X_test)

encf4_X_train = encf4_X_train.astype('float32')
encf4_X_test = encf4_X_test.astype('float32')
encf4_X_train /= 255
encf4_X_test /= 255

# Convert class vectors to binary class matrices. This is called one hot␣
 ↪encoding.
encf4_y_train = keras.utils.to_categorical(y_train, num_classes)
encf4_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[494]:
```
print('encf4_X_train shape:', encf4_X_train.shape)
print('encf4_y_train shape:', encf4_y_train.shape)
print('encf4_X_test shape:', encf4_X_test.shape)
print('encf4_y_test shape:', encf4_y_test.shape)
```

```
encf4_X_train shape: (50000, 32, 32, 3)
encf4_y_train shape: (50000, 10)
encf4_X_test shape: (10000, 32, 32, 3)
```

```
encf4_y_test shape: (10000, 10)
```

[495]:
```
#define the convnet
encf4_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf4_model.add(Conv2D(32, (3, 3), padding='same',input_shape = encf4_X_train.
 ↪shape[1:]))
encf4_model.add(Activation('relu'))
encf4_model.add(Conv2D(32, (3, 3)))
encf4_model.add(Activation('relu'))
encf4_model.add(MaxPooling2D(pool_size=(2, 2)))
encf4_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf4_model.add(Conv2D(64, (3, 3), padding='same'))
encf4_model.add(Activation('relu'))
encf4_model.add(Conv2D(64, (3, 3)))
encf4_model.add(Activation('relu'))
encf4_model.add(MaxPooling2D(pool_size=(2, 2)))
encf4_model.add(Dropout(0.25))

# FLATTERN => DENSE => RELU => DROPOUT
encf4_model.add(Flatten())
encf4_model.add(Dense(512))
encf4_model.add(Activation('relu'))
encf4_model.add(Dropout(0.5))
# a softmax classifier
encf4_model.add(Dense(num_classes))
encf4_model.add(Activation('softmax'))
```

[496]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
encf4_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[509]:
```
#Train model
encf4_history = encf4_model.fit(encf4_X_train, encf4_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(encf4_X_test, encf4_y_test),
              shuffle=True)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.9068 -
```

```
acc: 0.2980 - val_loss: 1.7556 - val_acc: 0.3749
Epoch 2/100
50000/50000 [==============================] - 167s 3ms/step - loss: 1.6672 -
acc: 0.3911 - val_loss: 1.6272 - val_acc: 0.4090
Epoch 3/100
50000/50000 [==============================] - 169s 3ms/step - loss: 1.5851 -
acc: 0.4245 - val_loss: 1.5644 - val_acc: 0.4313
Epoch 4/100
50000/50000 [==============================] - 175s 4ms/step - loss: 1.5312 -
acc: 0.4452 - val_loss: 1.4637 - val_acc: 0.4765
Epoch 5/100
50000/50000 [==============================] - 176s 4ms/step - loss: 1.4951 -
acc: 0.4601 - val_loss: 1.4567 - val_acc: 0.4770
Epoch 6/100
50000/50000 [==============================] - 174s 3ms/step - loss: 1.4641 -
acc: 0.4720 - val_loss: 1.4083 - val_acc: 0.4946
Epoch 7/100
50000/50000 [==============================] - 176s 4ms/step - loss: 1.4360 -
acc: 0.4839 - val_loss: 1.3934 - val_acc: 0.4961
Epoch 8/100
50000/50000 [==============================] - 181s 4ms/step - loss: 1.4180 -
acc: 0.4905 - val_loss: 1.4231 - val_acc: 0.4913
Epoch 9/100
50000/50000 [==============================] - 188s 4ms/step - loss: 1.3994 -
acc: 0.5008 - val_loss: 1.3730 - val_acc: 0.5116
Epoch 10/100
50000/50000 [==============================] - 198s 4ms/step - loss: 1.3820 -
acc: 0.5070 - val_loss: 1.3543 - val_acc: 0.5157
Epoch 11/100
50000/50000 [==============================] - 197s 4ms/step - loss: 1.3700 -
acc: 0.5101 - val_loss: 1.3495 - val_acc: 0.5141
Epoch 12/100
50000/50000 [==============================] - 192s 4ms/step - loss: 1.3584 -
acc: 0.5127 - val_loss: 1.3751 - val_acc: 0.5112
Epoch 13/100
50000/50000 [==============================] - 184s 4ms/step - loss: 1.3501 -
acc: 0.5194 - val_loss: 1.3701 - val_acc: 0.5169
Epoch 14/100
50000/50000 [==============================] - 185s 4ms/step - loss: 1.3447 -
acc: 0.5231 - val_loss: 1.3349 - val_acc: 0.5210
Epoch 15/100
50000/50000 [==============================] - 179s 4ms/step - loss: 1.3312 -
acc: 0.5271 - val_loss: 1.3163 - val_acc: 0.5340
Epoch 16/100
50000/50000 [==============================] - 176s 4ms/step - loss: 1.3257 -
acc: 0.5292 - val_loss: 1.3184 - val_acc: 0.5363
Epoch 17/100
50000/50000 [==============================] - 175s 4ms/step - loss: 1.3201 -
```

```
acc: 0.5316 - val_loss: 1.3564 - val_acc: 0.5264
Epoch 18/100
50000/50000 [==============================] - 161s 3ms/step - loss: 1.3153 -
acc: 0.5349 - val_loss: 1.3285 - val_acc: 0.5343
Epoch 19/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.3157 -
acc: 0.5349 - val_loss: 1.3539 - val_acc: 0.5281
Epoch 20/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.3114 -
acc: 0.5363 - val_loss: 1.4652 - val_acc: 0.4932
Epoch 21/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.3061 -
acc: 0.5411 - val_loss: 1.3501 - val_acc: 0.5361
Epoch 22/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.3033 -
acc: 0.5411 - val_loss: 1.3333 - val_acc: 0.5339
Epoch 23/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.3044 -
acc: 0.5395 - val_loss: 1.3243 - val_acc: 0.5419
Epoch 24/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.2979 -
acc: 0.5411 - val_loss: 1.3525 - val_acc: 0.5379
Epoch 25/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.3004 -
acc: 0.5402 - val_loss: 1.2821 - val_acc: 0.5517
Epoch 26/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.3007 -
acc: 0.5435 - val_loss: 1.3782 - val_acc: 0.5408
Epoch 27/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.2982 -
acc: 0.5433 - val_loss: 1.3600 - val_acc: 0.5254
Epoch 28/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.2930 -
acc: 0.5455 - val_loss: 1.4493 - val_acc: 0.5308
Epoch 29/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2946 -
acc: 0.5444 - val_loss: 1.2963 - val_acc: 0.5478
Epoch 30/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2939 -
acc: 0.5476 - val_loss: 1.3425 - val_acc: 0.5347
Epoch 31/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2969 -
acc: 0.5456 - val_loss: 1.3208 - val_acc: 0.5392
Epoch 32/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2932 -
acc: 0.5483 - val_loss: 1.3290 - val_acc: 0.5419
Epoch 33/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2932 -
```

```
acc: 0.5496 - val_loss: 1.3402 - val_acc: 0.5391
Epoch 34/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2974 -
acc: 0.5435 - val_loss: 1.3404 - val_acc: 0.5262
Epoch 35/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.2966 -
acc: 0.5481 - val_loss: 1.3491 - val_acc: 0.5305
Epoch 36/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3038 -
acc: 0.5460 - val_loss: 1.3148 - val_acc: 0.5439
Epoch 37/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3027 -
acc: 0.5445 - val_loss: 1.4298 - val_acc: 0.5278
Epoch 38/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3007 -
acc: 0.5462 - val_loss: 1.3427 - val_acc: 0.5356
Epoch 39/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3087 -
acc: 0.5446 - val_loss: 1.4597 - val_acc: 0.5211
Epoch 40/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.3017 -
acc: 0.5467 - val_loss: 1.3771 - val_acc: 0.5384
Epoch 41/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3111 -
acc: 0.5404 - val_loss: 1.4455 - val_acc: 0.5448
Epoch 42/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3117 -
acc: 0.5429 - val_loss: 1.2903 - val_acc: 0.5463
Epoch 43/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3102 -
acc: 0.5457 - val_loss: 1.3095 - val_acc: 0.5472
Epoch 44/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3139 -
acc: 0.5442 - val_loss: 1.2904 - val_acc: 0.5567
Epoch 45/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3155 -
acc: 0.5413 - val_loss: 1.3432 - val_acc: 0.5517
Epoch 46/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3146 -
acc: 0.5424 - val_loss: 1.3549 - val_acc: 0.5493
Epoch 47/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3251 -
acc: 0.5408 - val_loss: 1.3760 - val_acc: 0.5110
Epoch 48/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.3240 -
acc: 0.5416 - val_loss: 1.3873 - val_acc: 0.5248
Epoch 49/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3296 -
```

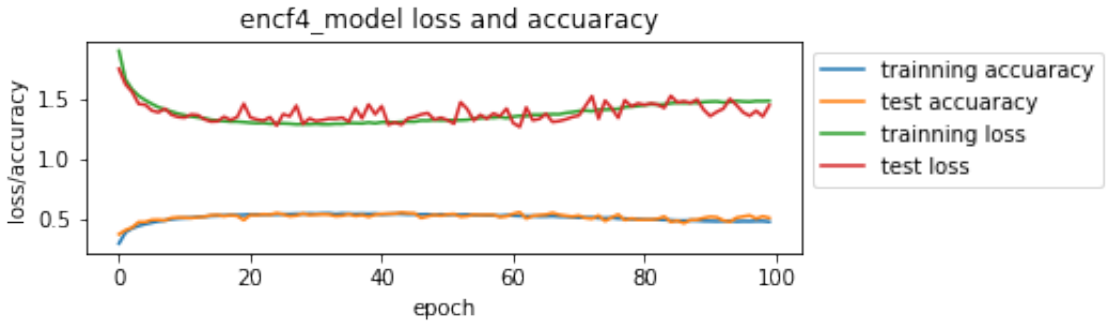
```
acc: 0.5370 - val_loss: 1.3321 - val_acc: 0.5371
Epoch 50/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3259 -
acc: 0.5418 - val_loss: 1.3488 - val_acc: 0.5300
Epoch 51/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3286 -
acc: 0.5380 - val_loss: 1.3286 - val_acc: 0.5407
Epoch 52/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3275 -
acc: 0.5414 - val_loss: 1.2987 - val_acc: 0.5436
Epoch 53/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3277 -
acc: 0.5378 - val_loss: 1.4784 - val_acc: 0.5220
Epoch 54/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3306 -
acc: 0.5403 - val_loss: 1.4218 - val_acc: 0.5264
Epoch 55/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3419 -
acc: 0.5354 - val_loss: 1.3241 - val_acc: 0.5400
Epoch 56/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3437 -
acc: 0.5338 - val_loss: 1.3720 - val_acc: 0.5360
Epoch 57/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3507 -
acc: 0.5323 - val_loss: 1.3399 - val_acc: 0.5379
Epoch 58/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3483 -
acc: 0.5318 - val_loss: 1.3837 - val_acc: 0.5372
Epoch 59/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.3536 -
acc: 0.5289 - val_loss: 1.3610 - val_acc: 0.5192
Epoch 60/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.3458 -
acc: 0.5333 - val_loss: 1.4250 - val_acc: 0.5248
Epoch 61/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.3570 -
acc: 0.5305 - val_loss: 1.3043 - val_acc: 0.5454
Epoch 62/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.3642 -
acc: 0.5285 - val_loss: 1.2713 - val_acc: 0.5588
Epoch 63/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.3615 -
acc: 0.5270 - val_loss: 1.4342 - val_acc: 0.5061
Epoch 64/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.3729 -
acc: 0.5272 - val_loss: 1.3270 - val_acc: 0.5282
Epoch 65/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.3730 -
```

```
acc: 0.5238 - val_loss: 1.3384 - val_acc: 0.5353
Epoch 66/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.3744 -
acc: 0.5244 - val_loss: 1.3805 - val_acc: 0.5392
Epoch 67/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.3779 -
acc: 0.5265 - val_loss: 1.3131 - val_acc: 0.5534
Epoch 68/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.3768 -
acc: 0.5225 - val_loss: 1.3204 - val_acc: 0.5377
Epoch 69/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.3882 -
acc: 0.5219 - val_loss: 1.3345 - val_acc: 0.5294
Epoch 70/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.3950 -
acc: 0.5178 - val_loss: 1.3541 - val_acc: 0.5223
Epoch 71/100
50000/50000 [==============================] - 159s 3ms/step - loss: 1.4040 -
acc: 0.5137 - val_loss: 1.3664 - val_acc: 0.5309
Epoch 72/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.4082 -
acc: 0.5147 - val_loss: 1.4457 - val_acc: 0.5105
Epoch 73/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.4031 -
acc: 0.5155 - val_loss: 1.5283 - val_acc: 0.5043
Epoch 74/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.4162 -
acc: 0.5092 - val_loss: 1.3401 - val_acc: 0.5316
Epoch 75/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.4156 -
acc: 0.5108 - val_loss: 1.4919 - val_acc: 0.4854
Epoch 76/100
50000/50000 [==============================] - 148s 3ms/step - loss: 1.4182 -
acc: 0.5118 - val_loss: 1.4281 - val_acc: 0.5152
Epoch 77/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.4225 -
acc: 0.5065 - val_loss: 1.3473 - val_acc: 0.5399
Epoch 78/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.4366 -
acc: 0.5068 - val_loss: 1.4949 - val_acc: 0.4930
Epoch 79/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4405 -
acc: 0.5009 - val_loss: 1.4430 - val_acc: 0.5073
Epoch 80/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.4462 -
acc: 0.5010 - val_loss: 1.4679 - val_acc: 0.4960
Epoch 81/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4543 -
```

```
acc: 0.4969 - val_loss: 1.4558 - val_acc: 0.4976
Epoch 82/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4601 -
acc: 0.4962 - val_loss: 1.4678 - val_acc: 0.4974
Epoch 83/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4570 -
acc: 0.4959 - val_loss: 1.4584 - val_acc: 0.5031
Epoch 84/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4606 -
acc: 0.4953 - val_loss: 1.4299 - val_acc: 0.5234
Epoch 85/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4687 -
acc: 0.4923 - val_loss: 1.5305 - val_acc: 0.4776
Epoch 86/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4734 -
acc: 0.4895 - val_loss: 1.4759 - val_acc: 0.4882
Epoch 87/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4709 -
acc: 0.4893 - val_loss: 1.4865 - val_acc: 0.4670
Epoch 88/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4736 -
acc: 0.4914 - val_loss: 1.4716 - val_acc: 0.4944
Epoch 89/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4726 -
acc: 0.4895 - val_loss: 1.5006 - val_acc: 0.4963
Epoch 90/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4798 -
acc: 0.4855 - val_loss: 1.4154 - val_acc: 0.5109
Epoch 91/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4830 -
acc: 0.4846 - val_loss: 1.3621 - val_acc: 0.5207
Epoch 92/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4845 -
acc: 0.4831 - val_loss: 1.3972 - val_acc: 0.5169
Epoch 93/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4759 -
acc: 0.4853 - val_loss: 1.4239 - val_acc: 0.4905
Epoch 94/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4796 -
acc: 0.4885 - val_loss: 1.5058 - val_acc: 0.4782
Epoch 95/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4884 -
acc: 0.4817 - val_loss: 1.4541 - val_acc: 0.5130
Epoch 96/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4835 -
acc: 0.4836 - val_loss: 1.3935 - val_acc: 0.5229
Epoch 97/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4816 -
```

```
acc: 0.4830 - val_loss: 1.3644 - val_acc: 0.5318
Epoch 98/100
50000/50000 [==============================] - 142s 3ms/step - loss: 1.4876 -
acc: 0.4849 - val_loss: 1.4060 - val_acc: 0.5061
Epoch 99/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4867 -
acc: 0.4848 - val_loss: 1.3590 - val_acc: 0.5240
Epoch 100/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4887 -
acc: 0.4797 - val_loss: 1.4524 - val_acc: 0.5112
```

[514]:
```
acc_loss_graphic(encf4_history, ' encf4_model loss and accuaracy')
```



[515]:
```
# Score trained enc_model.
encf4_scores = encf4_model.evaluate(encf4_X_test, encf4_y_test, verbose=1)
print('Test loss:', encf4_scores[0])
print('Test accuracy:', encf4_scores[1])

# make prediction.
encf4_pred = encf4_model.predict(encf4_X_test)
```

```
10000/10000 [==============================] - 7s 663us/step
Test loss: 1.452398067855835
Test accuracy: 0.5112
```

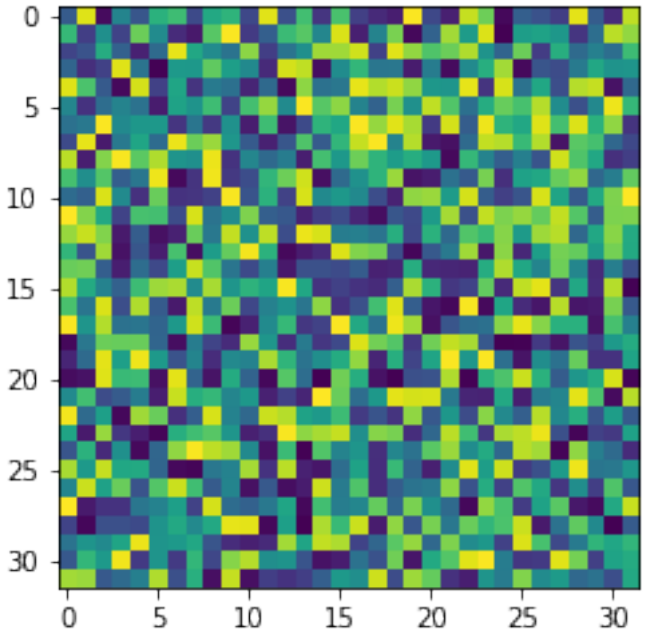
11 Train with encrypted CIFAR10 dataset using fixed key F5

[497]:
```
#load key f5
F5 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f5.npy")
#Drive the decryption matrix from the fixed encryption key.
DF5 = np.linalg.inv(F5)
```

Encryption key F5

[499]:
```
plt.imshow(F5)
```

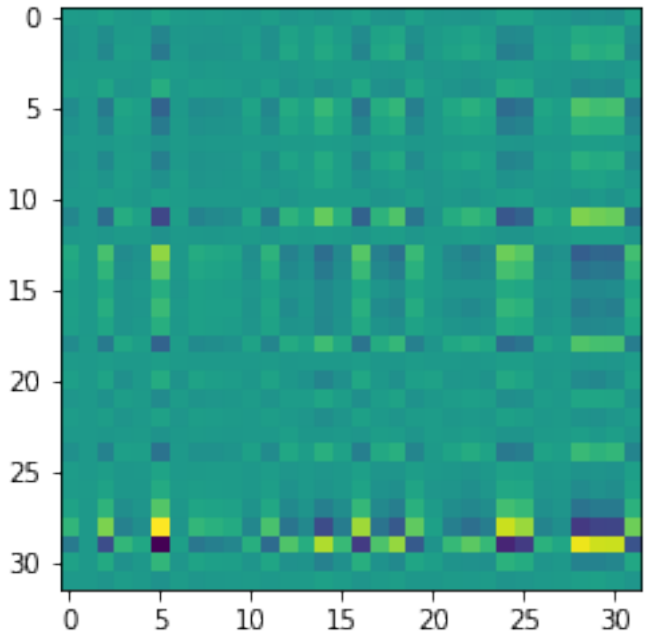
[499]:
```
<matplotlib.image.AxesImage at 0x199831da0>
```



Decryption key DF5

[500]:
```
plt.imshow(DF5)
```

[500]:
```
<matplotlib.image.AxesImage at 0x19980f3c8>
```



Encrypt the dataset

[501]:
```
encf5_X_train = []
for an_image in x_train:
    b, g, r = cv2.split(an_image)
    encf5_b = np.dot(b, F5)
    encf5_g = np.dot(g, F5)
    encf5_r = np.dot(r, F5)
    encf5_rgb = cv2.merge((encf5_b,encf5_g,encf5_r))
    encf5_X_train.append(encf5_rgb)
print("Done encrypting ",len(encf5_X_train),"train images")
encf5_X_test = []
for an_image in x_test:
    b, g, r = cv2.split(an_image)
    encf5_b = np.dot(b, F5)
    encf5_g = np.dot(g, F5)
    encf5_r = np.dot(r, F5)
    encf5_rgb = cv2.merge((encf5_b,encf5_g,encf5_r))
    encf5_X_test.append(encf5_rgb)
print("Done encrypting ",len(encf5_X_test),"test images")
```

```
Done encrypting  50000 train images
Done encrypting  10000 test images
```

[502]:
```
# Normalize the data.
encf5_X_train = np.array(encf5_X_train)
```

```
encf5_X_test = np.array(encf5_X_test)

encf5_X_train = encf5_X_train.astype('float32')
encf5_X_test = encf5_X_test.astype('float32')
encf5_X_train /= 255
encf5_X_test /= 255

# Convert class vectors to binary class matrices. This is called one hot␣
 ↪encoding.
encf5_y_train = keras.utils.to_categorical(y_train, num_classes)
encf5_y_test = keras.utils.to_categorical(y_test, num_classes)
```

[503]:
```
print('encf5_X_train shape:', encf5_X_train.shape)
print('encf5_y_train shape:', encf5_y_train.shape)
print('encf5_X_test shape:', encf5_X_test.shape)
print('encf5_y_test shape:', encf5_y_test.shape)
```

```
encf5_X_train shape: (50000, 32, 32, 3)
encf5_y_train shape: (50000, 10)
encf5_X_test shape: (10000, 32, 32, 3)
encf5_y_test shape: (10000, 10)
```

[504]:
```
#define the convnet
encf5_model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf5_model.add(Conv2D(32, (3, 3), padding='same',input_shape = encf5_X_train.
 ↪shape[1:]))
encf5_model.add(Activation('relu'))
encf5_model.add(Conv2D(32, (3, 3)))
encf5_model.add(Activation('relu'))
encf5_model.add(MaxPooling2D(pool_size=(2, 2)))
encf5_model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
encf5_model.add(Conv2D(64, (3, 3), padding='same'))
encf5_model.add(Activation('relu'))
encf5_model.add(Conv2D(64, (3, 3)))
encf5_model.add(Activation('relu'))
encf5_model.add(MaxPooling2D(pool_size=(2, 2)))
encf5_model.add(Dropout(0.25))

# FLATTERN => DENSE => RELU => DROPOUT
encf5_model.add(Flatten())
encf5_model.add(Dense(512))
encf5_model.add(Activation('relu'))
encf5_model.add(Dropout(0.5))
# a softmax classifier
```

```
encf5_model.add(Dense(num_classes))
encf5_model.add(Activation('softmax'))
```

[505]:
```
# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
encf5_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[510]:
```
#Train model
encf5_history = encf5_model.fit(encf5_X_train, encf5_y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(encf5_X_test, encf5_y_test),
              shuffle=True)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.8501 -
acc: 0.3153 - val_loss: 1.7918 - val_acc: 0.3474
Epoch 2/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.6563 -
acc: 0.3953 - val_loss: 1.5852 - val_acc: 0.4284
Epoch 3/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.5782 -
acc: 0.4258 - val_loss: 1.4937 - val_acc: 0.4564
Epoch 4/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.5276 -
acc: 0.4474 - val_loss: 1.4548 - val_acc: 0.4765
Epoch 5/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4901 -
acc: 0.4617 - val_loss: 1.4378 - val_acc: 0.4832
Epoch 6/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4584 -
acc: 0.4755 - val_loss: 1.4620 - val_acc: 0.4803
Epoch 7/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4334 -
acc: 0.4850 - val_loss: 1.3943 - val_acc: 0.4940
Epoch 8/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4132 -
acc: 0.4922 - val_loss: 1.3956 - val_acc: 0.4937
Epoch 9/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3956 -
acc: 0.4980 - val_loss: 1.3482 - val_acc: 0.5198
Epoch 10/100
```

```
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3818 -
acc: 0.5047 - val_loss: 1.3645 - val_acc: 0.5218
Epoch 11/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3664 -
acc: 0.5113 - val_loss: 1.3673 - val_acc: 0.5156
Epoch 12/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3556 -
acc: 0.5176 - val_loss: 1.3843 - val_acc: 0.5148
Epoch 13/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.3460 -
acc: 0.5183 - val_loss: 1.3742 - val_acc: 0.5201
Epoch 14/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.3351 -
acc: 0.5262 - val_loss: 1.3350 - val_acc: 0.5342
Epoch 15/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.3261 -
acc: 0.5304 - val_loss: 1.3473 - val_acc: 0.5310
Epoch 16/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.3181 -
acc: 0.5322 - val_loss: 1.3038 - val_acc: 0.5432
Epoch 17/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.3142 -
acc: 0.5352 - val_loss: 1.3059 - val_acc: 0.5386
Epoch 18/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.3060 -
acc: 0.5396 - val_loss: 1.3238 - val_acc: 0.5242
Epoch 19/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.3085 -
acc: 0.5393 - val_loss: 1.3227 - val_acc: 0.5352
Epoch 20/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.2983 -
acc: 0.5421 - val_loss: 1.3381 - val_acc: 0.5240
Epoch 21/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.2922 -
acc: 0.5419 - val_loss: 1.3342 - val_acc: 0.5326
Epoch 22/100
50000/50000 [==============================] - 154s 3ms/step - loss: 1.2919 -
acc: 0.5441 - val_loss: 1.2813 - val_acc: 0.5499
Epoch 23/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.2881 -
acc: 0.5453 - val_loss: 1.3552 - val_acc: 0.5522
Epoch 24/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.2845 -
acc: 0.5487 - val_loss: 1.3778 - val_acc: 0.5394
Epoch 25/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.2826 -
acc: 0.5497 - val_loss: 1.4686 - val_acc: 0.5356
Epoch 26/100
```

```
50000/50000 [==============================] - 157s 3ms/step - loss: 1.2812 -
acc: 0.5482 - val_loss: 1.2843 - val_acc: 0.5527
Epoch 27/100
50000/50000 [==============================] - 157s 3ms/step - loss: 1.2777 -
acc: 0.5523 - val_loss: 1.2582 - val_acc: 0.5610
Epoch 28/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.2760 -
acc: 0.5546 - val_loss: 1.2900 - val_acc: 0.5515
Epoch 29/100
50000/50000 [==============================] - 151s 3ms/step - loss: 1.2764 -
acc: 0.5541 - val_loss: 1.3355 - val_acc: 0.5397
Epoch 30/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.2744 -
acc: 0.5538 - val_loss: 1.2527 - val_acc: 0.5629
Epoch 31/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.2746 -
acc: 0.5547 - val_loss: 1.3671 - val_acc: 0.5489
Epoch 32/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.2699 -
acc: 0.5564 - val_loss: 1.2570 - val_acc: 0.5703
Epoch 33/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.2728 -
acc: 0.5541 - val_loss: 1.3407 - val_acc: 0.5575
Epoch 34/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.2792 -
acc: 0.5555 - val_loss: 1.2911 - val_acc: 0.5422
Epoch 35/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2739 -
acc: 0.5555 - val_loss: 1.3214 - val_acc: 0.5569
Epoch 36/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2736 -
acc: 0.5549 - val_loss: 1.2701 - val_acc: 0.5595
Epoch 37/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2770 -
acc: 0.5545 - val_loss: 1.3308 - val_acc: 0.5577
Epoch 38/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2745 -
acc: 0.5558 - val_loss: 1.3663 - val_acc: 0.5297
Epoch 39/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2800 -
acc: 0.5565 - val_loss: 1.3763 - val_acc: 0.5427
Epoch 40/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2765 -
acc: 0.5564 - val_loss: 1.3479 - val_acc: 0.5292
Epoch 41/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2812 -
acc: 0.5544 - val_loss: 1.4226 - val_acc: 0.5278
Epoch 42/100
```

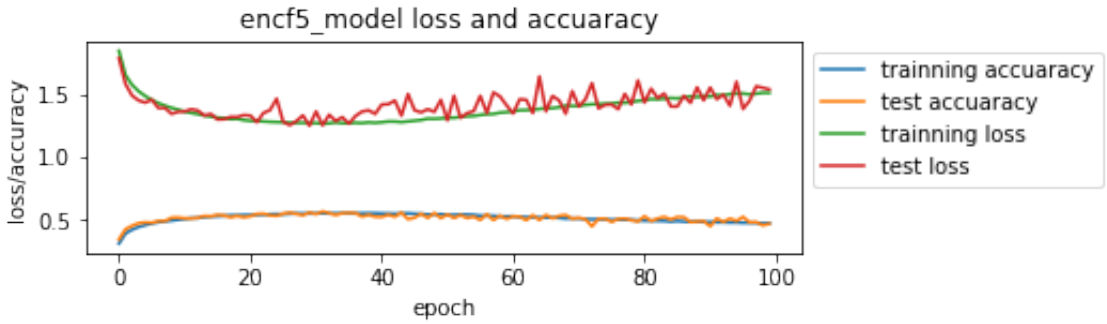
```
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2872 -
acc: 0.5540 - val_loss: 1.4252 - val_acc: 0.5438
Epoch 43/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2869 -
acc: 0.5519 - val_loss: 1.4542 - val_acc: 0.5287
Epoch 44/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.2822 -
acc: 0.5538 - val_loss: 1.3336 - val_acc: 0.5622
Epoch 45/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2884 -
acc: 0.5556 - val_loss: 1.5059 - val_acc: 0.5093
Epoch 46/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.2931 -
acc: 0.5519 - val_loss: 1.3766 - val_acc: 0.5212
Epoch 47/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.2989 -
acc: 0.5509 - val_loss: 1.3603 - val_acc: 0.5524
Epoch 48/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3098 -
acc: 0.5460 - val_loss: 1.3847 - val_acc: 0.5192
Epoch 49/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3090 -
acc: 0.5476 - val_loss: 1.3642 - val_acc: 0.5578
Epoch 50/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3115 -
acc: 0.5484 - val_loss: 1.4593 - val_acc: 0.5170
Epoch 51/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.3172 -
acc: 0.5433 - val_loss: 1.2965 - val_acc: 0.5471
Epoch 52/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3168 -
acc: 0.5430 - val_loss: 1.4921 - val_acc: 0.5181
Epoch 53/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3240 -
acc: 0.5455 - val_loss: 1.3162 - val_acc: 0.5419
Epoch 54/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3288 -
acc: 0.5453 - val_loss: 1.3536 - val_acc: 0.5185
Epoch 55/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3312 -
acc: 0.5428 - val_loss: 1.3693 - val_acc: 0.5288
Epoch 56/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3395 -
acc: 0.5365 - val_loss: 1.4912 - val_acc: 0.5032
Epoch 57/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3453 -
acc: 0.5359 - val_loss: 1.3543 - val_acc: 0.5487
Epoch 58/100
```

```
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3470 -
acc: 0.5350 - val_loss: 1.5184 - val_acc: 0.5018
Epoch 59/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3578 -
acc: 0.5312 - val_loss: 1.4872 - val_acc: 0.5370
Epoch 60/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3614 -
acc: 0.5309 - val_loss: 1.4164 - val_acc: 0.5114
Epoch 61/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3682 -
acc: 0.5318 - val_loss: 1.3512 - val_acc: 0.5310
Epoch 62/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3768 -
acc: 0.5266 - val_loss: 1.4645 - val_acc: 0.5290
Epoch 63/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3778 -
acc: 0.5251 - val_loss: 1.4597 - val_acc: 0.5008
Epoch 64/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3808 -
acc: 0.5243 - val_loss: 1.3546 - val_acc: 0.5403
Epoch 65/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3869 -
acc: 0.5224 - val_loss: 1.6465 - val_acc: 0.5022
Epoch 66/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.3944 -
acc: 0.5208 - val_loss: 1.3681 - val_acc: 0.5422
Epoch 67/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.3934 -
acc: 0.5216 - val_loss: 1.4904 - val_acc: 0.5153
Epoch 68/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.4007 -
acc: 0.5189 - val_loss: 1.4552 - val_acc: 0.5348
Epoch 69/100
50000/50000 [==============================] - 152s 3ms/step - loss: 1.4140 -
acc: 0.5140 - val_loss: 1.3494 - val_acc: 0.5416
Epoch 70/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.4111 -
acc: 0.5158 - val_loss: 1.5290 - val_acc: 0.5126
Epoch 71/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.4121 -
acc: 0.5145 - val_loss: 1.4065 - val_acc: 0.5196
Epoch 72/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.4200 -
acc: 0.5100 - val_loss: 1.4632 - val_acc: 0.5094
Epoch 73/100
50000/50000 [==============================] - 156s 3ms/step - loss: 1.4248 -
acc: 0.5081 - val_loss: 1.5928 - val_acc: 0.4516
Epoch 74/100
```

```
50000/50000 [==============================] - 163s 3ms/step - loss: 1.4258 -
acc: 0.5090 - val_loss: 1.3916 - val_acc: 0.5119
Epoch 75/100
50000/50000 [==============================] - 160s 3ms/step - loss: 1.4290 -
acc: 0.5081 - val_loss: 1.4204 - val_acc: 0.5132
Epoch 76/100
50000/50000 [==============================] - 161s 3ms/step - loss: 1.4301 -
acc: 0.5043 - val_loss: 1.4229 - val_acc: 0.4872
Epoch 77/100
50000/50000 [==============================] - 161s 3ms/step - loss: 1.4374 -
acc: 0.5053 - val_loss: 1.3856 - val_acc: 0.5107
Epoch 78/100
50000/50000 [==============================] - 161s 3ms/step - loss: 1.4361 -
acc: 0.5037 - val_loss: 1.5066 - val_acc: 0.5137
Epoch 79/100
50000/50000 [==============================] - 155s 3ms/step - loss: 1.4432 -
acc: 0.5035 - val_loss: 1.4231 - val_acc: 0.5013
Epoch 80/100
50000/50000 [==============================] - 153s 3ms/step - loss: 1.4512 -
acc: 0.4996 - val_loss: 1.6143 - val_acc: 0.4904
Epoch 81/100
50000/50000 [==============================] - 150s 3ms/step - loss: 1.4570 -
acc: 0.4968 - val_loss: 1.4380 - val_acc: 0.5320
Epoch 82/100
50000/50000 [==============================] - 149s 3ms/step - loss: 1.4657 -
acc: 0.4976 - val_loss: 1.5487 - val_acc: 0.4942
Epoch 83/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.4644 -
acc: 0.4958 - val_loss: 1.4737 - val_acc: 0.5136
Epoch 84/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.4687 -
acc: 0.4950 - val_loss: 1.5099 - val_acc: 0.5267
Epoch 85/100
50000/50000 [==============================] - 147s 3ms/step - loss: 1.4744 -
acc: 0.4902 - val_loss: 1.4077 - val_acc: 0.5066
Epoch 86/100
50000/50000 [==============================] - 146s 3ms/step - loss: 1.4716 -
acc: 0.4942 - val_loss: 1.4070 - val_acc: 0.5256
Epoch 87/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.4772 -
acc: 0.4904 - val_loss: 1.4914 - val_acc: 0.5240
Epoch 88/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.4878 -
acc: 0.4878 - val_loss: 1.4360 - val_acc: 0.4856
Epoch 89/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4814 -
acc: 0.4891 - val_loss: 1.5590 - val_acc: 0.4903
Epoch 90/100
```

```
50000/50000 [==============================] - 145s 3ms/step - loss: 1.4913 -
acc: 0.4866 - val_loss: 1.4558 - val_acc: 0.4937
Epoch 91/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4856 -
acc: 0.4873 - val_loss: 1.5576 - val_acc: 0.4531
Epoch 92/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.4991 -
acc: 0.4843 - val_loss: 1.4650 - val_acc: 0.5146
Epoch 93/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.4950 -
acc: 0.4840 - val_loss: 1.5016 - val_acc: 0.4865
Epoch 94/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.5058 -
acc: 0.4824 - val_loss: 1.4127 - val_acc: 0.5133
Epoch 95/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.5096 -
acc: 0.4826 - val_loss: 1.6042 - val_acc: 0.4943
Epoch 96/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.5102 -
acc: 0.4802 - val_loss: 1.3874 - val_acc: 0.5272
Epoch 97/100
50000/50000 [==============================] - 145s 3ms/step - loss: 1.5036 -
acc: 0.4800 - val_loss: 1.4542 - val_acc: 0.4834
Epoch 98/100
50000/50000 [==============================] - 143s 3ms/step - loss: 1.5089 -
acc: 0.4796 - val_loss: 1.5677 - val_acc: 0.4861
Epoch 99/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.5158 -
acc: 0.4752 - val_loss: 1.5557 - val_acc: 0.4598
Epoch 100/100
50000/50000 [==============================] - 144s 3ms/step - loss: 1.5154 -
acc: 0.4779 - val_loss: 1.5431 - val_acc: 0.4731
```

[512]:
```
acc_loss_graphic(encf5_history, ' encf5_model loss and accuaracy')
```



[513]:
```
# Score trained enc_model.
encf5_scores = encf5_model.evaluate(encf5_X_test, encf5_y_test, verbose=1)
print('Test loss:', encf5_scores[0])
print('Test accuracy:', encf5_scores[1])

# make prediction.
encf5_pred = encf5_model.predict(encf5_X_test)
```

```
10000/10000 [==============================] - 7s 659us/step
Test loss: 1.5430591384887695
Test accuracy: 0.4731
```

[ ]:
```

```

https://www.kaggle.com/roblexnana/cifar10-with-cnn-for-beginer/comments

Appendix_C

December 7, 2020

1 Appendix C

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import scipy.linalg
        import cv2
        from PIL import Image
        from keras.datasets import mnist
        from keras.datasets import cifar10
```

```
Using TensorFlow backend.
```

2 Generate keys

```
In [10]: #Read key image as RGB
         key1 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key1.png"
         key2 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key2.png"
         key3 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key3.png"
         key4 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key4.png"
         key5 = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/key5.png"

         #gray scale key image
         key1 = key1.convert('L')
         key2 = key2.convert('L')
         key3 = key3.convert('L')
         key4 = key4.convert('L')
         key5 = key5.convert('L')
```

```
In [11]: #Drive orthogonal matrix
         #resize key image
         mnist_key1 = np.resize(key1,(28,28))
         mnist_key2 = np.resize(key2,(28,28))
         mnist_key3 = np.resize(key3,(28,28))
         mnist_key4 = np.resize(key3,(28,28))
         mnist_key5 = np.resize(key3,(28,28))
```

```
#convert to 2D array
mnist_key1 = np.asarray(mnist_key1)
mnist_key2 = np.asarray(mnist_key2)
mnist_key3 = np.asarray(mnist_key3)
mnist_key4 = np.asarray(mnist_key4)
mnist_key5 = np.asarray(mnist_key5)

#QR decomposition
mn_key1, mn_R1 = scipy.linalg.qr(mnist_key1)
mn_key2, mn_R2 = scipy.linalg.qr(mnist_key2)
mn_key3, mn_R3 = scipy.linalg.qr(mnist_key3)
mn_key4, mn_R4 = scipy.linalg.qr(mnist_key4)
mn_key5, mn_R5 = scipy.linalg.qr(mnist_key5)

#Drive the decryption matrix
mn_D1 = np.linalg.inv(mn_key1)
mn_D2 = np.linalg.inv(mn_key2)
mn_D3 = np.linalg.inv(mn_key3)
mn_D4 = np.linalg.inv(mn_key4)
mn_D5 = np.linalg.inv(mn_key5)
```

3 Keys use on MNIST

In [12]:
```
#load key 6
mn_key6 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f1.npy")
mn_key6 = np.resize(mn_key6, (28,28))
#Drive the decryption matrix from the fixed encryption key.
mn_D6 = np.linalg.inv(mn_key6)

#load key 7
mn_key7 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f2.npy")
mn_key7 = np.resize(mn_key7, (28,28))
#Drive the decryption matrix from the fixed encryption key.
mn_D7 = np.linalg.inv(mn_key7)

#load key 8
mn_key8 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f3.npy")
mn_key8 = np.resize(mn_key8, (28,28))
#Drive the decryption matrix from the fixed encryption key.
mn_D8 = np.linalg.inv(mn_key8)

#load key 9
mn_key9 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f4.npy")
mn_key9 = np.resize(mn_key9, (28,28))
#Drive the decryption matrix from the fixed encryption key.
mn_D9 = np.linalg.inv(mn_key9)
```

```
#load key 10
mn_key10 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f5.npy")
mn_key10 = np.resize(mn_key10, (28,28))
#Drive the decryption matrix from the fixed encryption key.
mn_D10 = np.linalg.inv(mn_key10)
```

4 Keys use on CIFAR10

In [13]:
```
#Drive orthogonal matrix
#resize key image
cifar10_key1 = np.resize(key1,(32,32))
cifar10_key2 = np.resize(key2,(32,32))
cifar10_key3 = np.resize(key3,(32,32))
cifar10_key4 = np.resize(key3,(32,32))
cifar10_key5 = np.resize(key3,(32,32))

#convert to 2D array
cifar10_key1 = np.asarray(cifar10_key1)
cifar10_key2 = np.asarray(cifar10_key2)
cifar10_key3 = np.asarray(cifar10_key3)
cifar10_key4 = np.asarray(cifar10_key4)
cifar10_key5 = np.asarray(cifar10_key5)

#QR decomposition
cf_key1, cf_R1 = scipy.linalg.qr(cifar10_key1)
cf_key2, cf_R2 = scipy.linalg.qr(cifar10_key2)
cf_key3, cf_R3 = scipy.linalg.qr(cifar10_key3)
cf_key4, cf_R4 = scipy.linalg.qr(cifar10_key4)
cf_key5, cf_R5 = scipy.linalg.qr(cifar10_key5)

#Drive the decryption matrix
cf_D1 = np.linalg.inv(cf_key1)
cf_D2 = np.linalg.inv(cf_key2)
cf_D3 = np.linalg.inv(cf_key3)
cf_D4 = np.linalg.inv(cf_key4)
cf_D5 = np.linalg.inv(cf_key5)
```

In [14]:
```
#load key 6
cf_key6 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f1.npy")
#Drive the decryption matrix from the fixed encryption key.
cf0_D6 = np.linalg.inv(cf_key6)

#load key 7
cf_key7 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f2.npy")
#Drive the decryption matrix from the fixed encryption key.
cf_D7 = np.linalg.inv(cf_key7)
```

```
#load key 8
cf_key8 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f3.npy")
#Drive the decryption matrix from the fixed encryption key.
cf_D8 = np.linalg.inv(cf_key8)

#load key 9
cf_key9 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f4.npy")
#Drive the decryption matrix from the fixed encryption key.
cf_D9 = np.linalg.inv(cf_key9)

#load key 10
cf_key10 = np.load("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/keys/f5.npy")
#Drive the decryption matrix from the fixed encryption key.
cf_D10 = np.linalg.inv(cf_key10)
```

5 Load MNIST data

In [15]:
```
#Load MNIST data
(mnist_X_train, mnist_y_train), (mnist_X_test, mnist_Y_test) = mnist.load_data()
print("Done loading MNIST data!")
```

```
Done loading MNIST data!
```

6 Encrypt MNIST data

In [17]:
```
mn_enc_X_train1 = []
mn_enc_X_train2 = []
mn_enc_X_train3 = []
mn_enc_X_train4 = []
mn_enc_X_train5 = []
mn_enc_X_train6 = []
mn_enc_X_train7 = []
mn_enc_X_train8 = []
mn_enc_X_train9 = []
mn_enc_X_train10 = []
for an_image in mnist_X_train:
    an_image1 = np.dot(an_image, mn_key1)
    mn_enc_X_train1.append(an_image1)

    an_image2 = np.dot(an_image, mn_key2)
    mn_enc_X_train2.append(an_image2)

    an_image3 = np.dot(an_image, mn_key3)
    mn_enc_X_train3.append(an_image3)

    an_image4 = np.dot(an_image, mn_key4)
```

```
            mn_enc_X_train4.append(an_image4)

            an_image5 = np.dot(an_image, mn_key5)
            mn_enc_X_train5.append(an_image5)

            an_image6 = np.dot(an_image, mn_key6)
            mn_enc_X_train6.append(an_image6)

            an_image7 = np.dot(an_image, mn_key7)
            mn_enc_X_train7.append(an_image7)

            an_image8 = np.dot(an_image, mn_key8)
            mn_enc_X_train8.append(an_image8)

            an_image9 = np.dot(an_image, mn_key9)
            mn_enc_X_train9.append(an_image9)

            an_image10 = np.dot(an_image, mn_key10)
            mn_enc_X_train10.append(an_image10)

        print("Done encrypting!")
```

```
Done encrypting!
```

Get list of object indexes

```
In [18]: list_0 = []
         list_1 = []
         list_2 = []
         list_3 = []
         list_4 = []
         list_5 = []
         list_6 = []
         list_7 = []
         list_8 = []
         list_9 = []

         count0 = 0
         count1 = 0
         count2 = 0
         count3 = 0
         count4 = 0
         count5 = 0
         count6 = 0
         count7 = 0
         count8 = 0
         count9 = 0
```

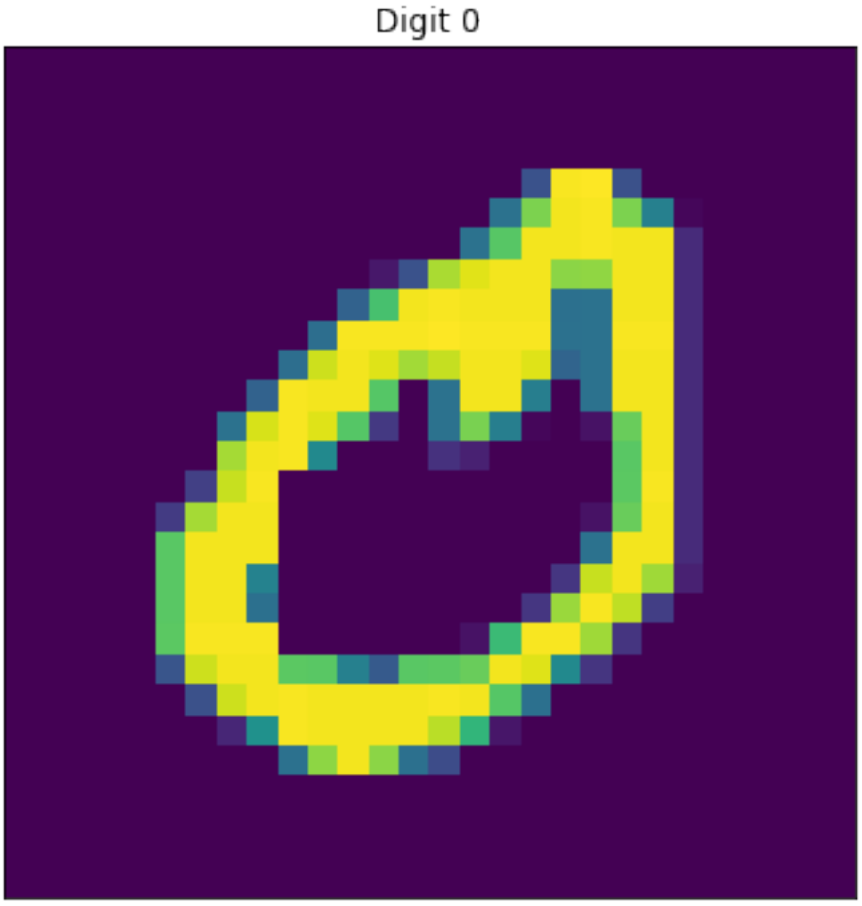
```
for i in range(len(mnist_y_train)):
    if mnist_y_train[i] == 0 and count0 < 10:
        list_0.append(i)
        count0 +=1
    elif mnist_y_train[i] == 1 and count1 < 10:
        list_1.append(i)
        count1 +=1
    elif mnist_y_train[i] == 2 and count2 < 10:
        list_2.append(i)
        count2 +=1
    elif mnist_y_train[i] == 3 and count3 < 10:
        list_3.append(i)
        count3 +=1
    elif mnist_y_train[i] == 4 and count4 < 10:
        list_4.append(i)
        count4 +=1
    elif mnist_y_train[i] == 5 and count5 < 10:
        list_5.append(i)
        count5 +=1
    elif mnist_y_train[i] == 6 and count6 < 10:
        list_6.append(i)
        count6 +=1
    elif mnist_y_train[i] == 7 and count7 < 10:
        list_7.append(i)
        count7 +=1
    elif mnist_y_train[i] == 8 and count8 < 10:
        list_8.append(i)
        count8 +=1
    elif mnist_y_train[i] == 9 and count9 < 10:
        list_9.append(i)
        count9 +=1
```

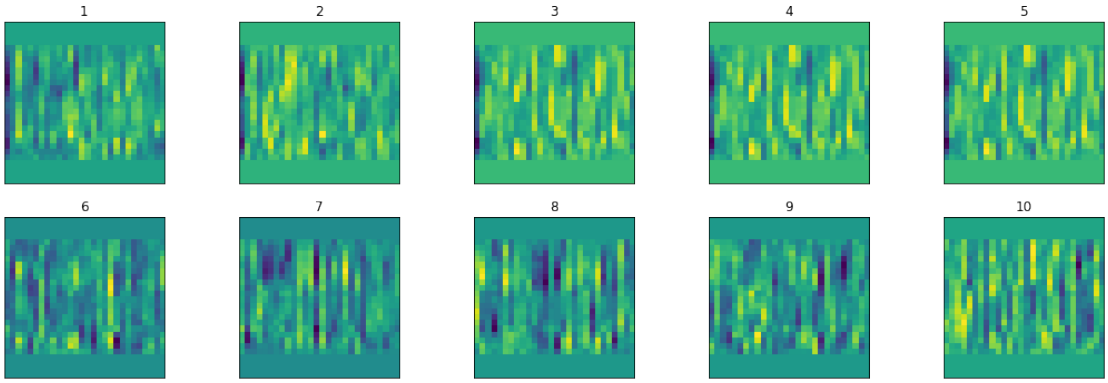
7 Same object with different keys

Digit 0

```
In [20]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_0 = [mn_enc_X_train1[21],
                       mn_enc_X_train2[21],
                       mn_enc_X_train3[21],
                       mn_enc_X_train4[21],
                       mn_enc_X_train5[21],
                       mn_enc_X_train6[21],
                       mn_enc_X_train7[21],
                       mn_enc_X_train8[21],
```

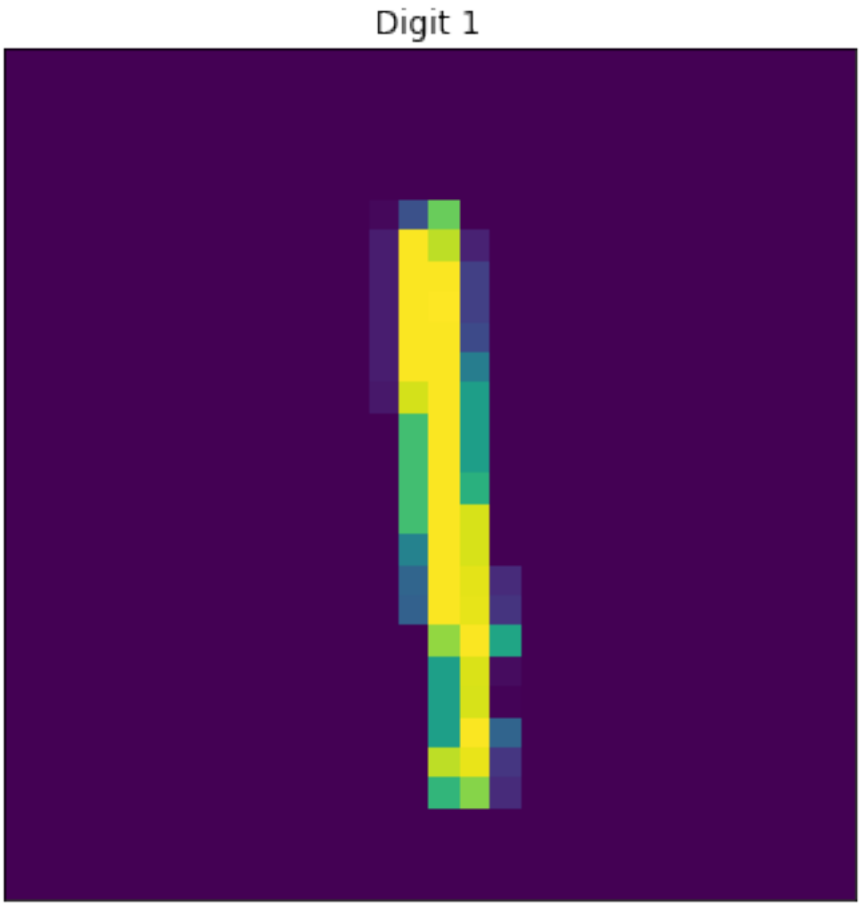
```
                mn_enc_X_train9[21],
                mn_enc_X_train10[21]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[21], cmap='viridis')
plt.title("Digit 0")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_0[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```

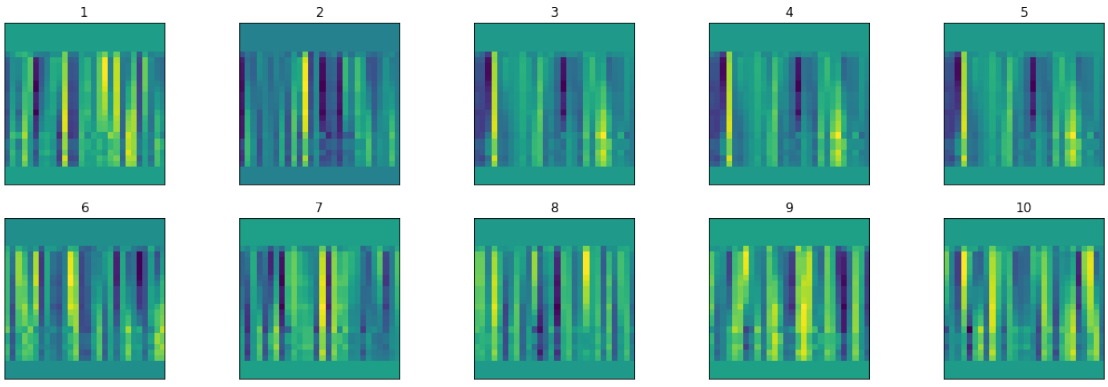




Digit 1

```
In [21]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_1 = [mn_enc_X_train1[8],
                       mn_enc_X_train2[8],
                       mn_enc_X_train3[8],
                       mn_enc_X_train4[8],
                       mn_enc_X_train5[8],
                       mn_enc_X_train6[8],
                       mn_enc_X_train7[8],
                       mn_enc_X_train8[8],
                       mn_enc_X_train9[8],
                       mn_enc_X_train10[8]]

         fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         plt.imshow(mnist_X_train[8], cmap='viridis')
         plt.title("Digit 1")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(diff_key_1[i], cmap='viridis')
             ax.set_title(i+1)
         plt.tight_layout()
         plt.show()
```

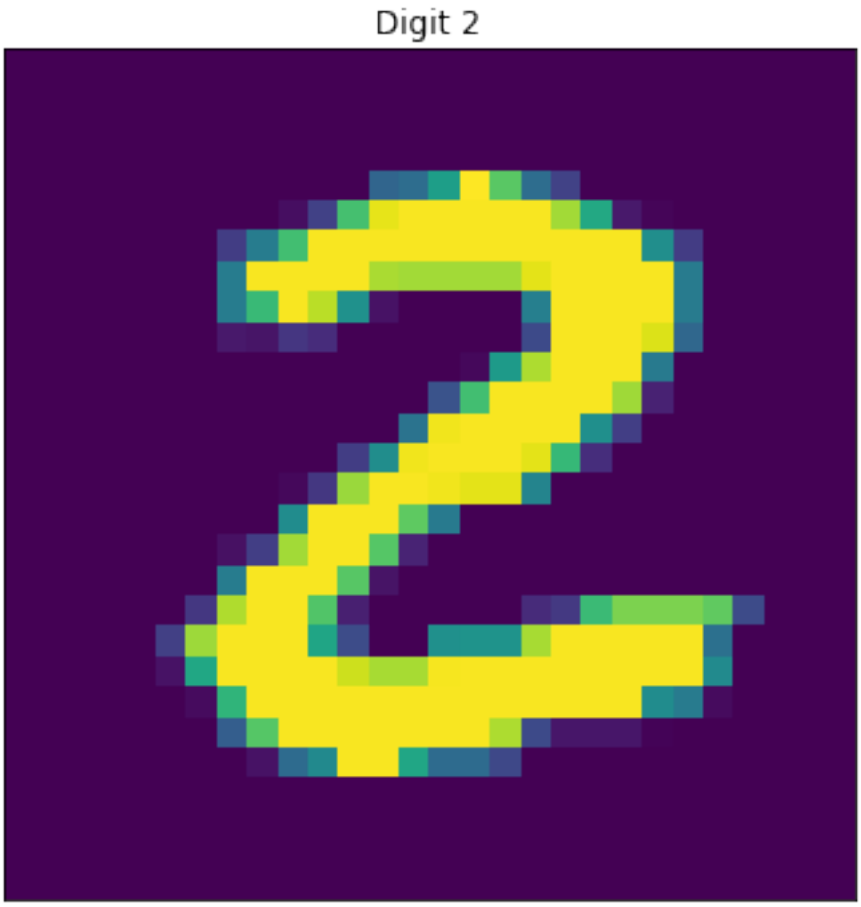


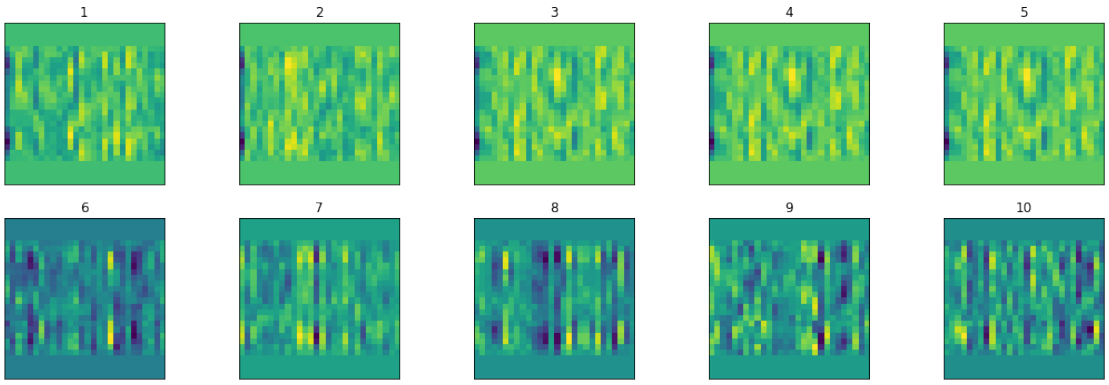


Digit 2

```
In [22]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_2 = [mn_enc_X_train1[25],
```

```
                mn_enc_X_train2[25],
                mn_enc_X_train3[25],
                mn_enc_X_train4[25],
                mn_enc_X_train5[25],
                mn_enc_X_train6[25],
                mn_enc_X_train7[25],
                mn_enc_X_train8[25],
                mn_enc_X_train9[25],
                mn_enc_X_train10[25]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[25], cmap='viridis')
plt.title("Digit 2")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_2[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```

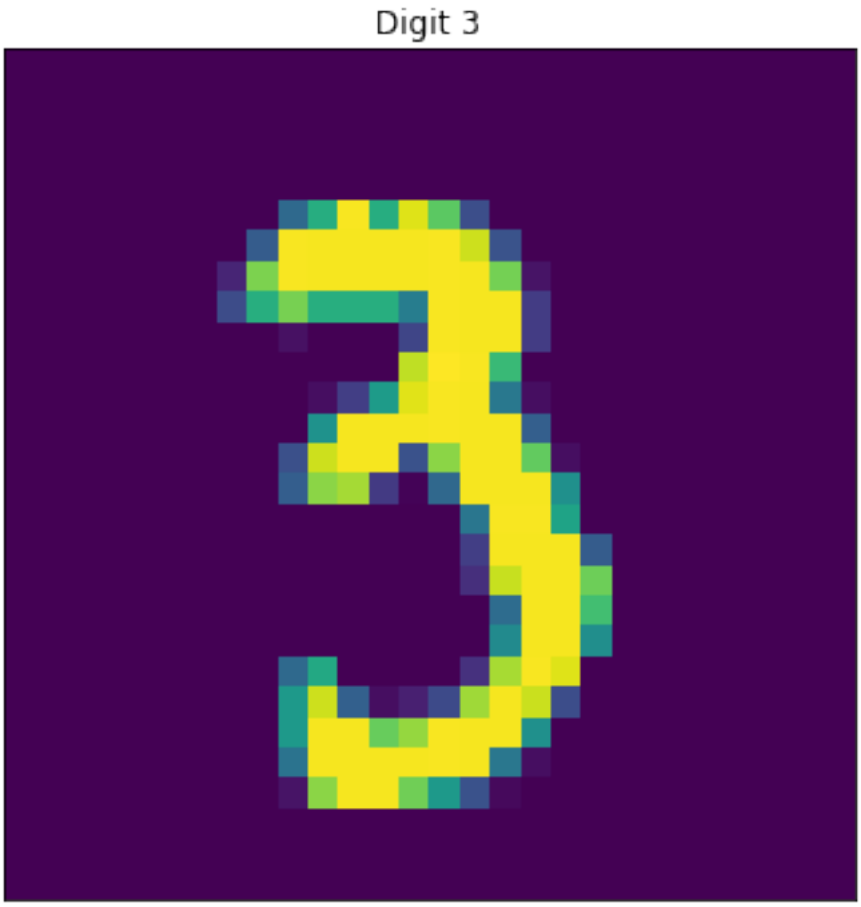


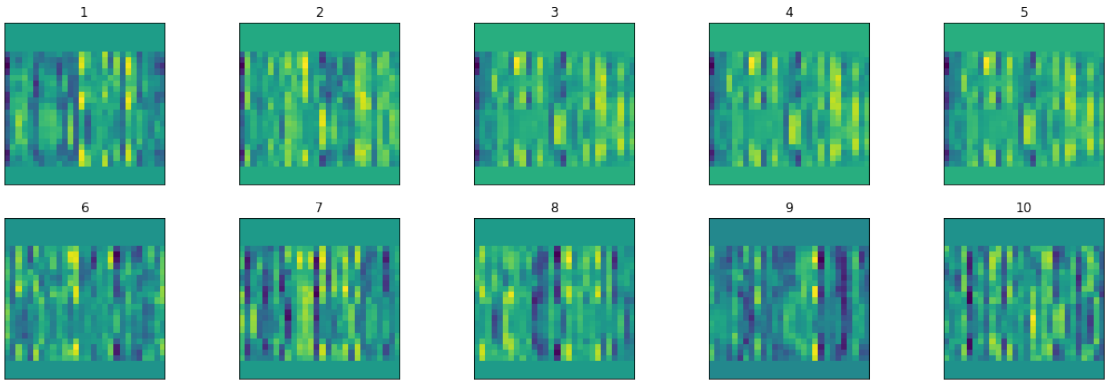


Digit 3

```
In [24]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_3 = [mn_enc_X_train1[74],
```

```
                mn_enc_X_train2[74],
                mn_enc_X_train3[74],
                mn_enc_X_train4[74],
                mn_enc_X_train5[74],
                mn_enc_X_train6[74],
                mn_enc_X_train7[74],
                mn_enc_X_train8[74],
                mn_enc_X_train9[74],
                mn_enc_X_train10[74]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[74], cmap='viridis')
plt.title("Digit 3")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_3[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```

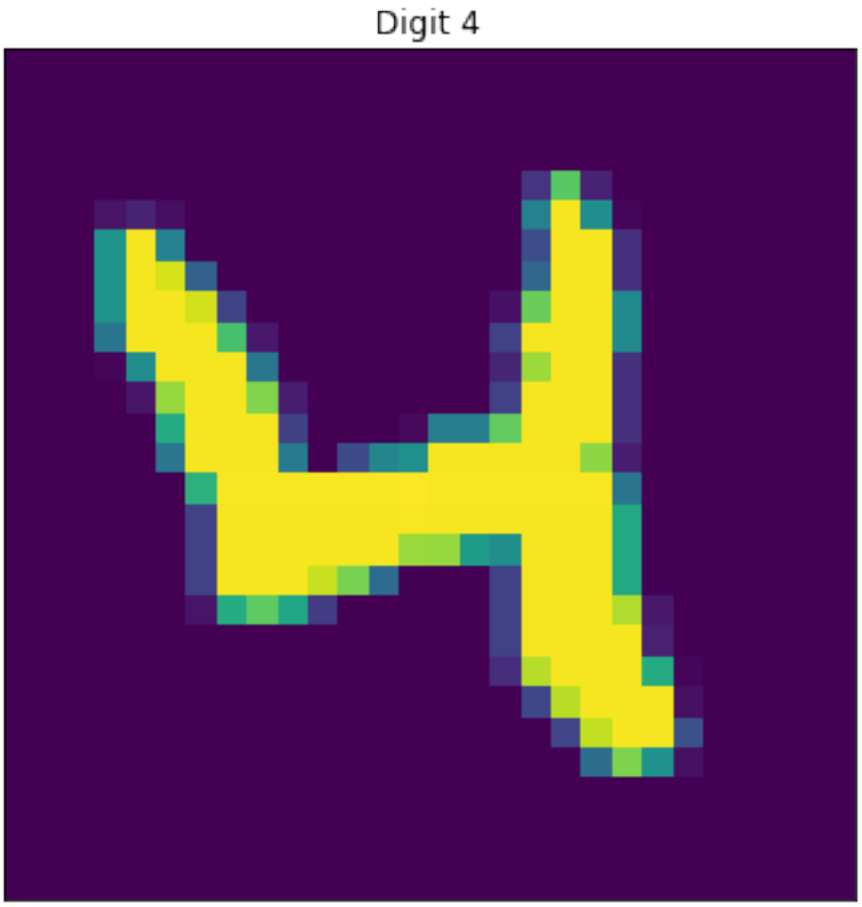


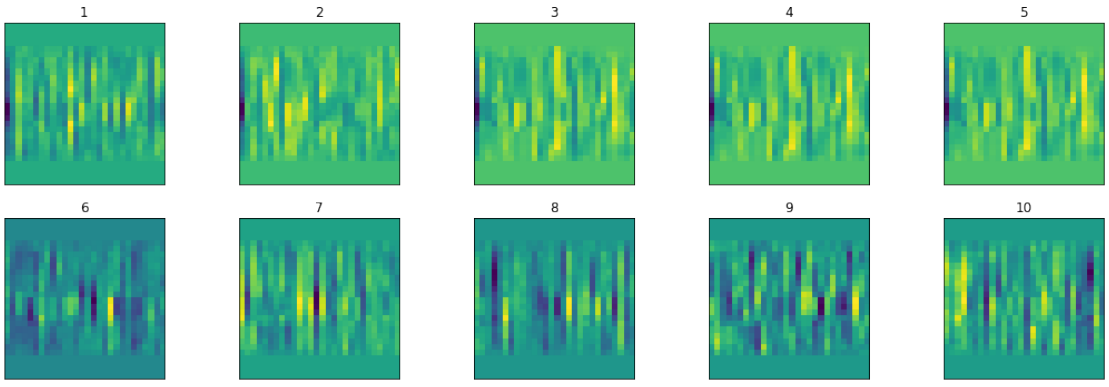


Digit 4

```
In [25]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_4 = [mn_enc_X_train1[20],
```

```
                mn_enc_X_train2[20],
                mn_enc_X_train3[20],
                mn_enc_X_train4[20],
                mn_enc_X_train5[20],
                mn_enc_X_train6[20],
                mn_enc_X_train7[20],
                mn_enc_X_train8[20],
                mn_enc_X_train9[20],
                mn_enc_X_train10[20]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[20], cmap='viridis')
plt.title("Digit 4")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_4[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```

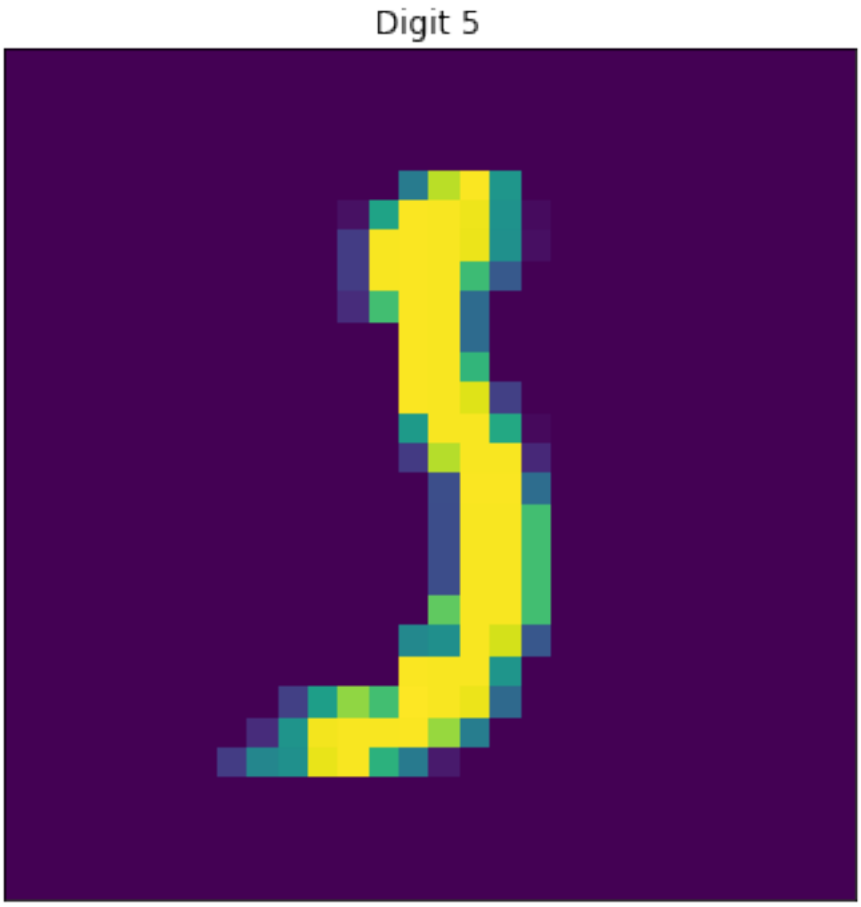


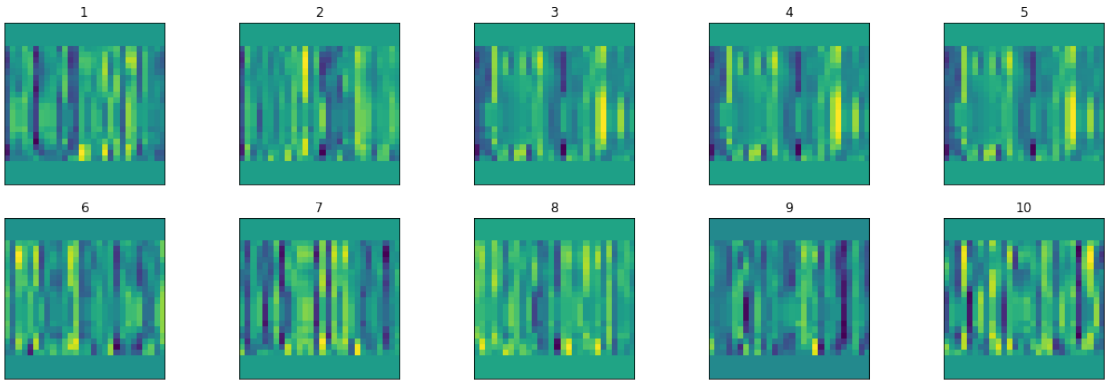


Digit 5

```
In [26]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_5 = [mn_enc_X_train1[132],
```

```
                mn_enc_X_train2[132],
                mn_enc_X_train3[132],
                mn_enc_X_train4[132],
                mn_enc_X_train5[132],
                mn_enc_X_train6[132],
                mn_enc_X_train7[132],
                mn_enc_X_train8[132],
                mn_enc_X_train9[132],
                mn_enc_X_train10[132]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[132], cmap='viridis')
plt.title("Digit 5")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_5[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```

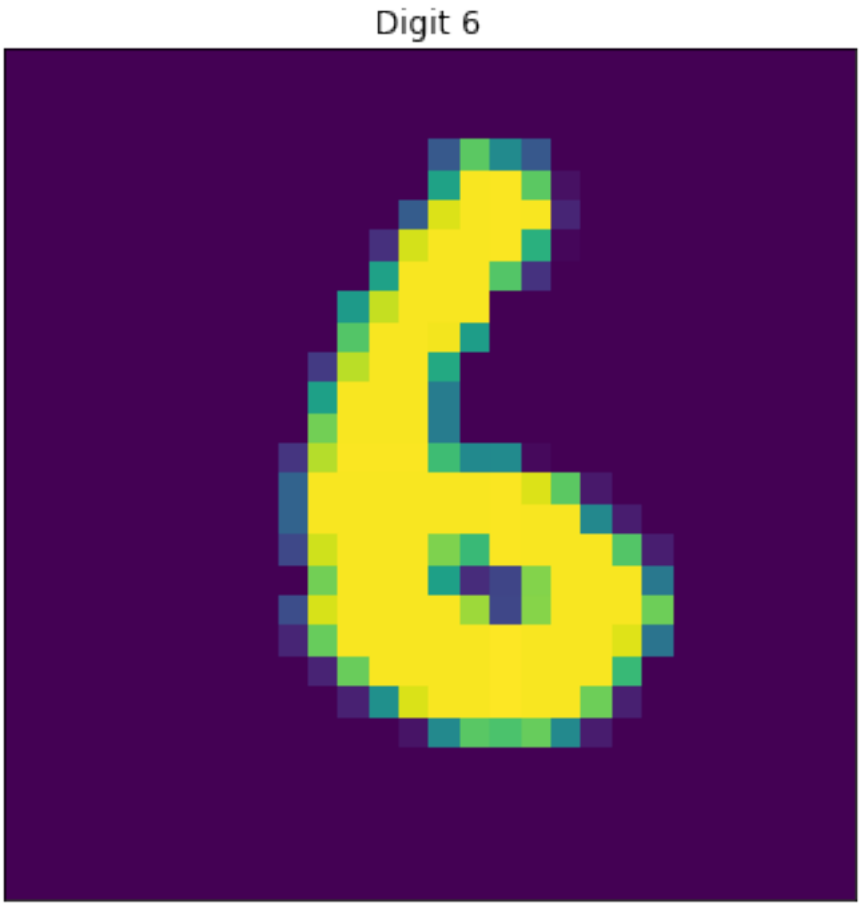


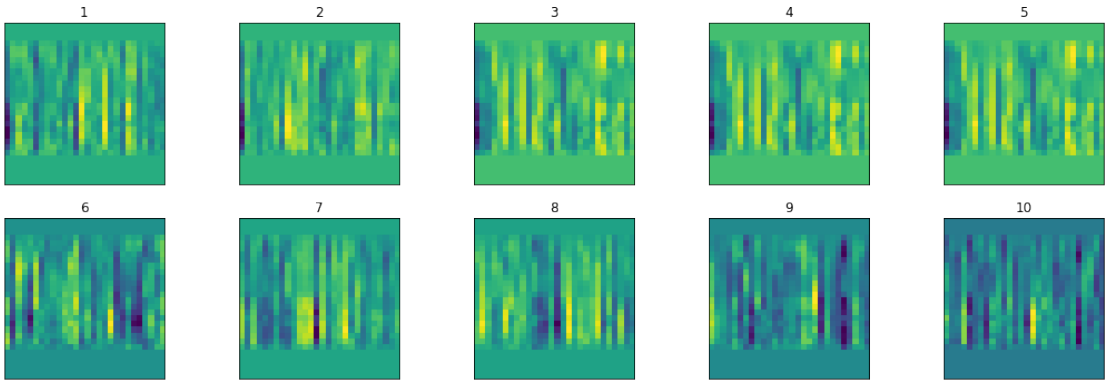


Digit 6

```
In [27]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_6 = [mn_enc_X_train1[36],
```

```
                mn_enc_X_train2[36],
                mn_enc_X_train3[36],
                mn_enc_X_train4[36],
                mn_enc_X_train5[36],
                mn_enc_X_train6[36],
                mn_enc_X_train7[36],
                mn_enc_X_train8[36],
                mn_enc_X_train9[36],
                mn_enc_X_train10[36]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[36], cmap='viridis')
plt.title("Digit 6")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_6[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```

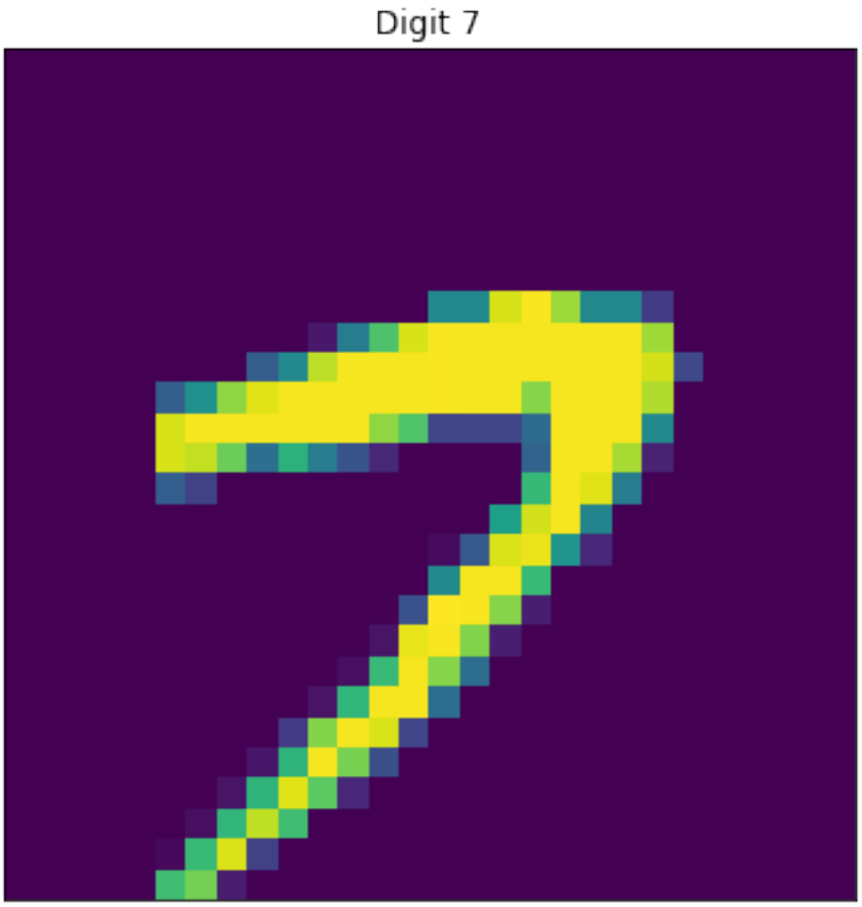


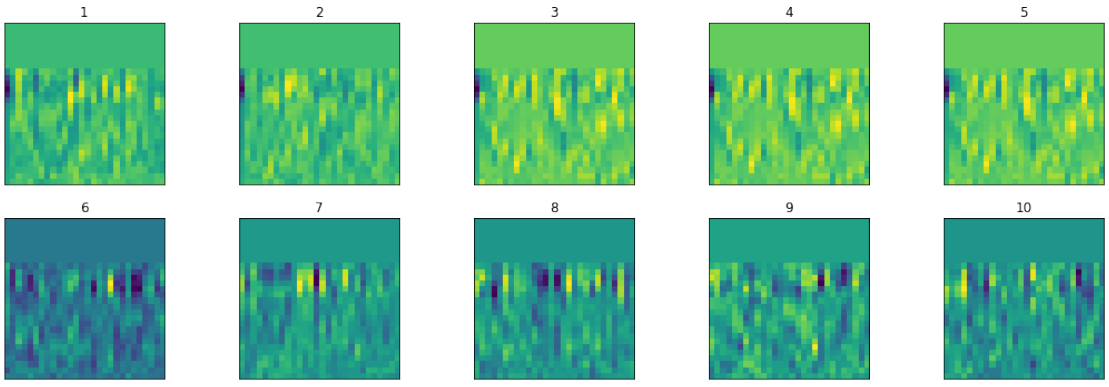


Digit 7

```
In [28]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_7 = [mn_enc_X_train1[91],
```

```
                mn_enc_X_train2[91],
                mn_enc_X_train3[91],
                mn_enc_X_train4[91],
                mn_enc_X_train5[91],
                mn_enc_X_train6[91],
                mn_enc_X_train7[91],
                mn_enc_X_train8[91],
                mn_enc_X_train9[91],
                mn_enc_X_train10[91]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[91], cmap='viridis')
plt.title("Digit 7")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_7[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```

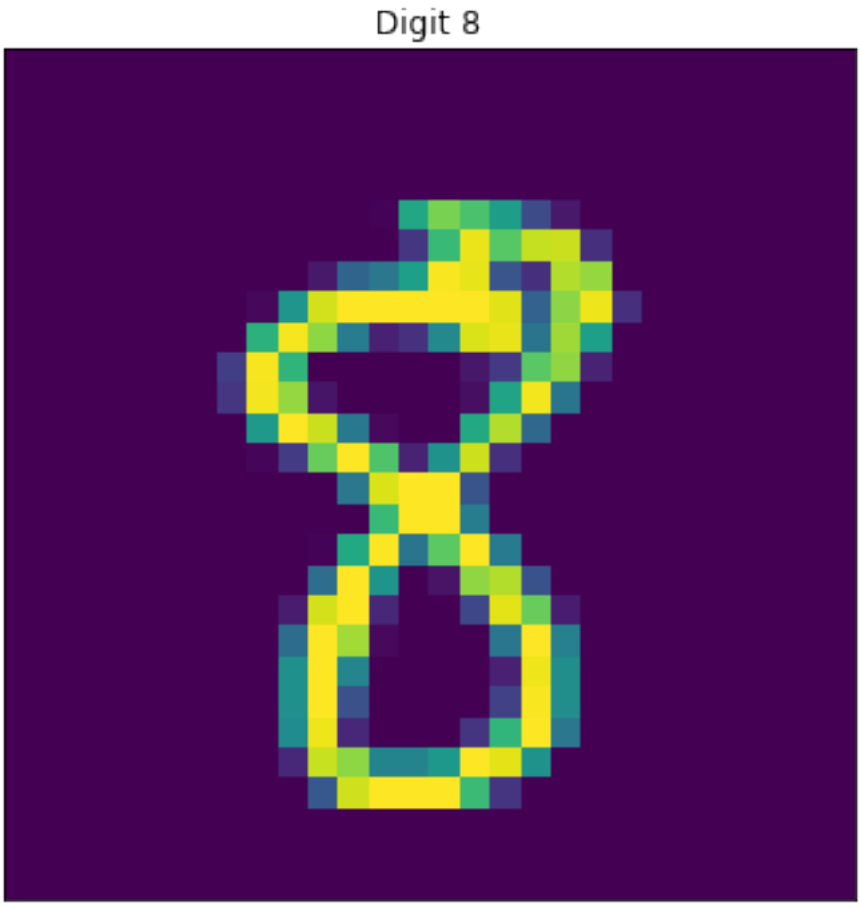


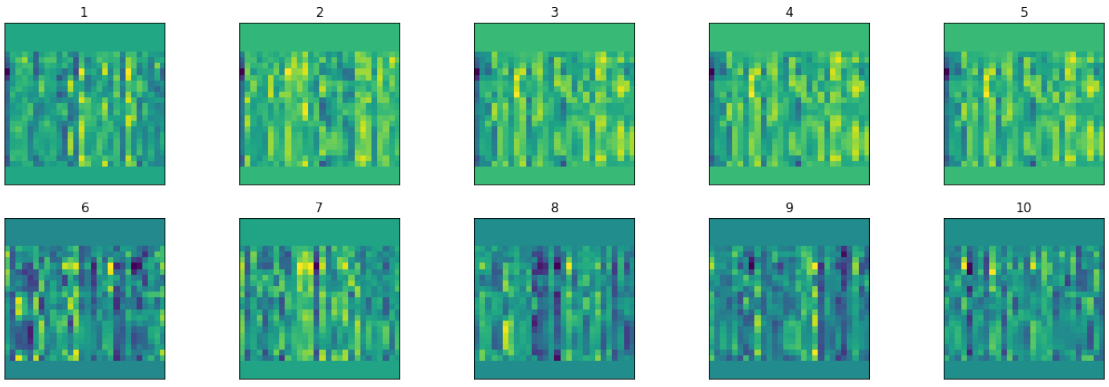


Digit 8

```
In [29]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_8 = [mn_enc_X_train1[46],
```

```
                mn_enc_X_train2[46],
                mn_enc_X_train3[46],
                mn_enc_X_train4[46],
                mn_enc_X_train5[46],
                mn_enc_X_train6[46],
                mn_enc_X_train7[46],
                mn_enc_X_train8[46],
                mn_enc_X_train9[46],
                mn_enc_X_train10[46]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[46], cmap='viridis')
plt.title("Digit 8")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_8[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```

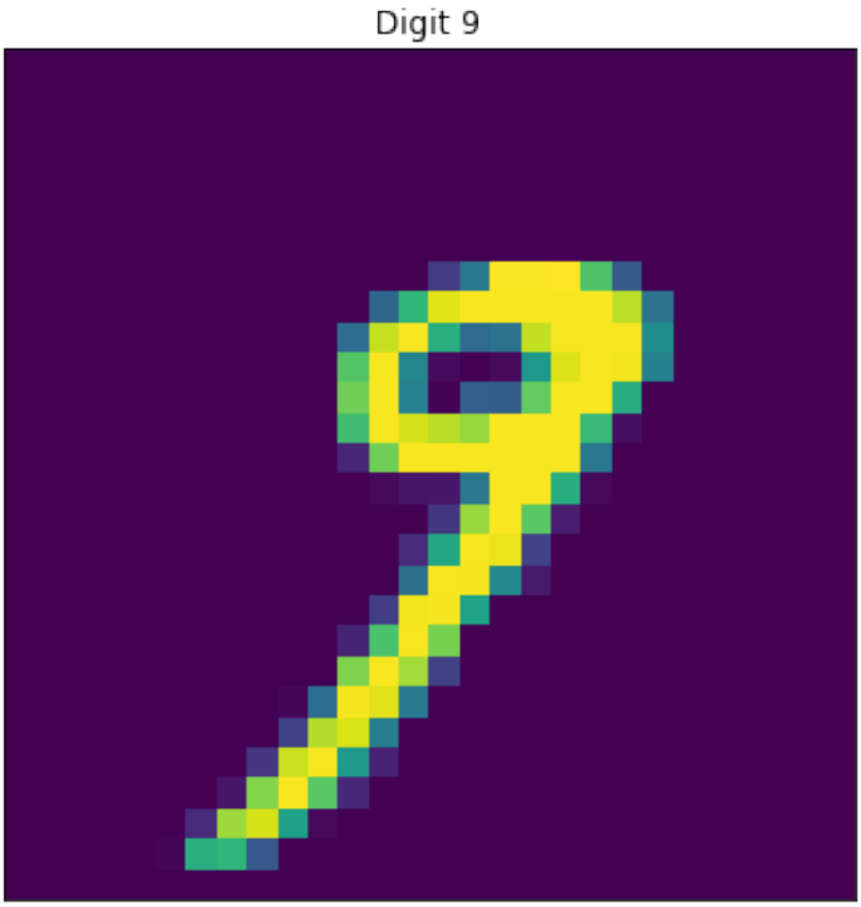


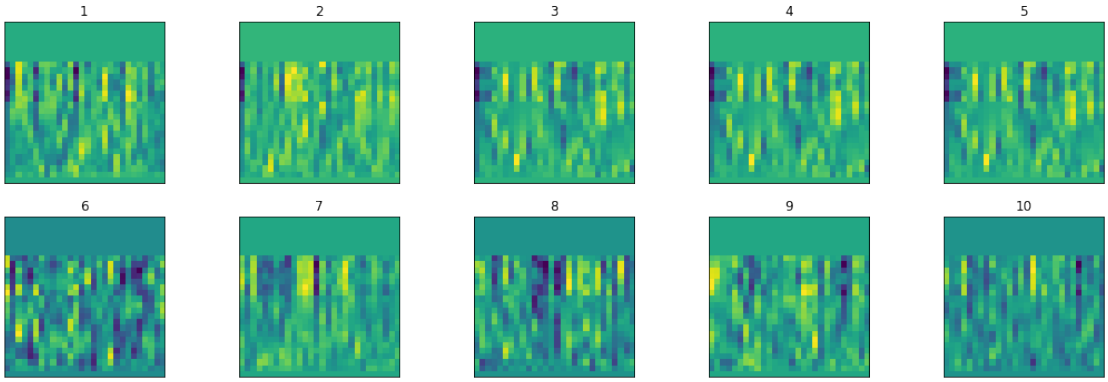


Digit 9

```
In [30]: index = [0,1,2,3,4,5,6,7,8,9]
         diff_key_9 = [mn_enc_X_train1[33],
```

```
                mn_enc_X_train2[33],
                mn_enc_X_train3[33],
                mn_enc_X_train4[33],
                mn_enc_X_train5[33],
                mn_enc_X_train6[33],
                mn_enc_X_train7[33],
                mn_enc_X_train8[33],
                mn_enc_X_train9[33],
                mn_enc_X_train10[33]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(mnist_X_train[33], cmap='viridis')
plt.title("Digit 9")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(diff_key_9[i], cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```



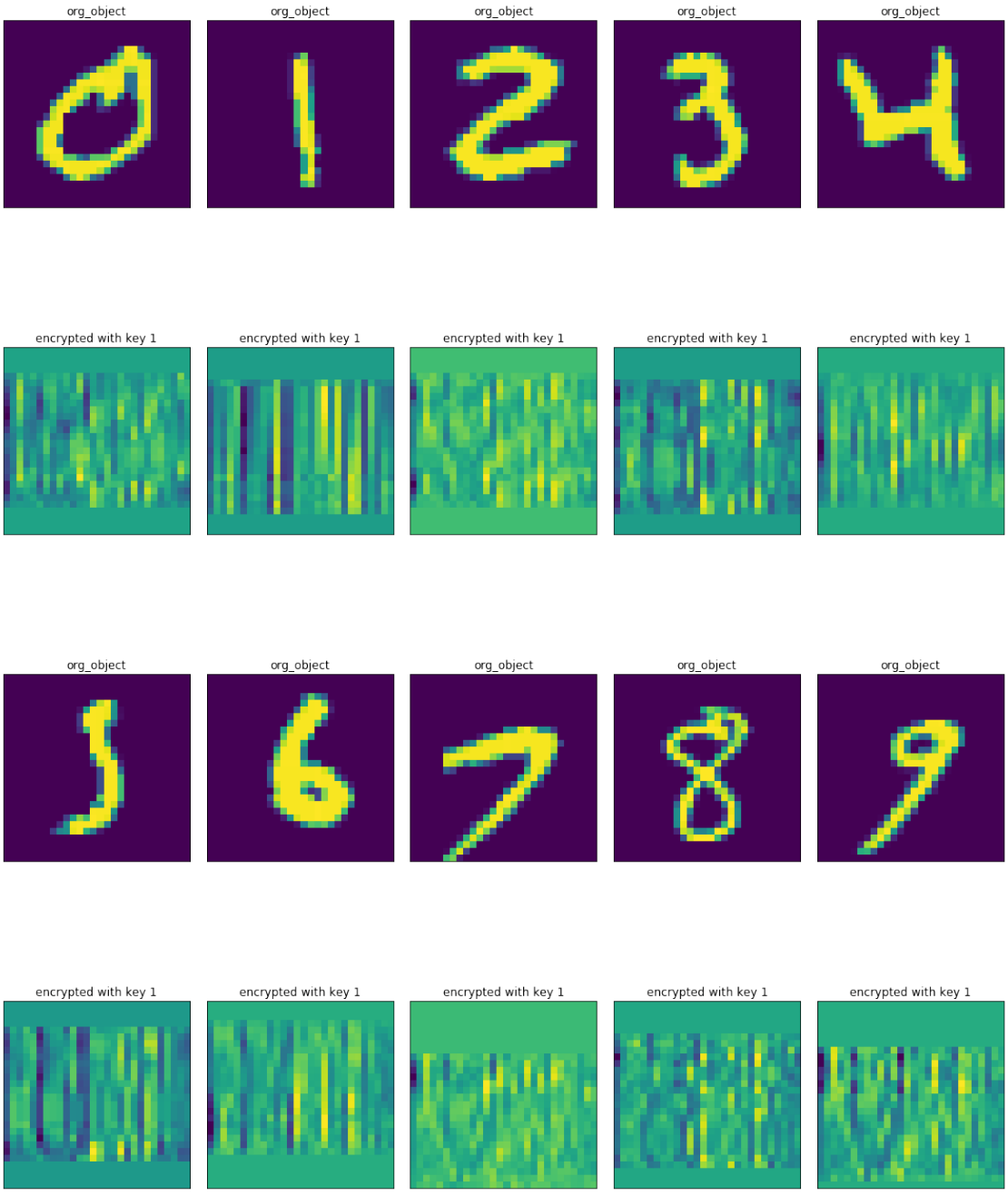


8 Different objects with same key

Key 1

```
In [43]: index = [0,1,2,3,4]
         list_org1_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org1_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key1_1 = [mn_enc_X_train1[21],
                        mn_enc_X_train1[8],
                        mn_enc_X_train1[25],
                        mn_enc_X_train1[74],
                        mn_enc_X_train1[20]]
         list_key1_2 = [mn_enc_X_train1[132],
                        mn_enc_X_train1[36],
                        mn_enc_X_train1[91],
                        mn_enc_X_train1[46],
                        mn_enc_X_train1[33]]


         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org1_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key1_1[i], cmap='viridis')
             ax.set_title("encrypted with key 1")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org1_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()
```

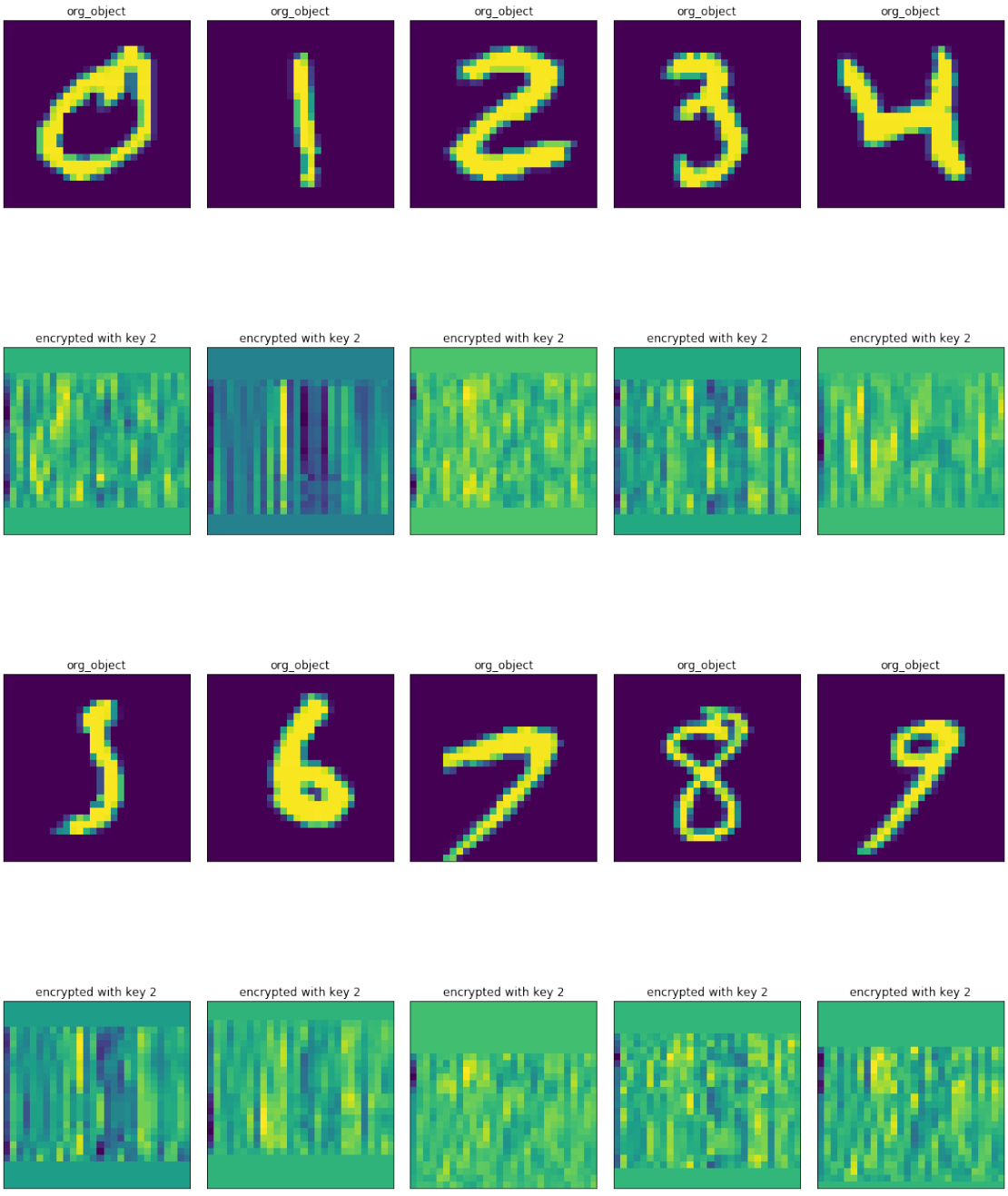
```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_key1_2[i], cmap='viridis')
    ax.set_title("encrypted with key 1")
plt.tight_layout()
plt.show()
```



Key 2

```
In [44]: index = [0,1,2,3,4]
         list_org2_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org2_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key2_1 = [mn_enc_X_train2[21],
                        mn_enc_X_train2[8],
                        mn_enc_X_train2[25],
                        mn_enc_X_train2[74],
                        mn_enc_X_train2[20]]
         list_key2_2 = [mn_enc_X_train2[132],
                        mn_enc_X_train2[36],
                        mn_enc_X_train2[91],
                        mn_enc_X_train2[46],
                        mn_enc_X_train2[33]]


         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org2_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key2_1[i], cmap='viridis')
             ax.set_title("encrypted with key 2")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org2_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()
```

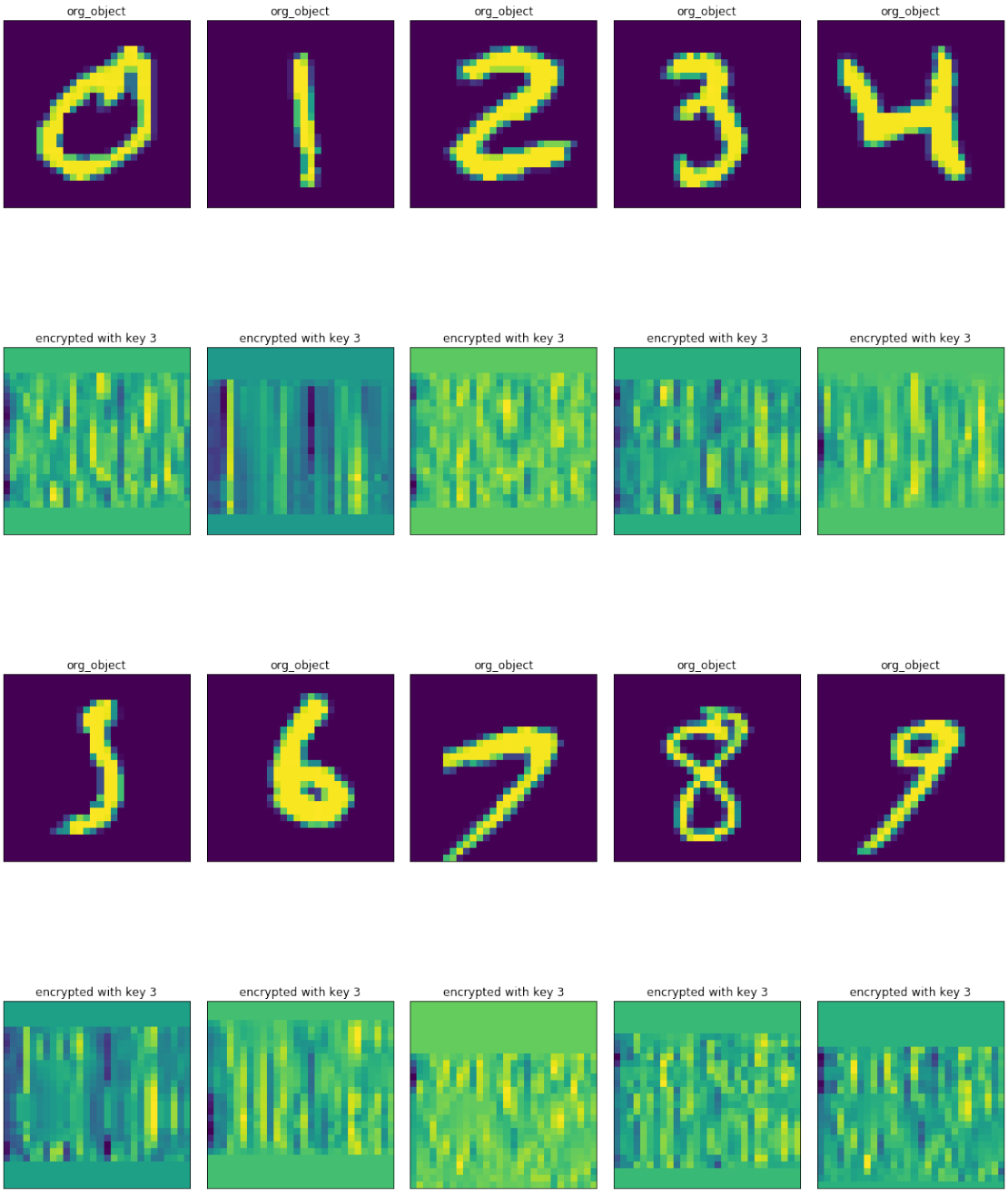
```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_key2_2[i], cmap='viridis')
    ax.set_title("encrypted with key 2")
plt.tight_layout()
plt.show()
```



Key 3

```
In [45]: index = [0,1,2,3,4]
         list_org3_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org3_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key3_1 = [mn_enc_X_train3[21],
                        mn_enc_X_train3[8],
                        mn_enc_X_train3[25],
                        mn_enc_X_train3[74],
                        mn_enc_X_train3[20]]
         list_key3_2 = [mn_enc_X_train3[132],
                        mn_enc_X_train3[36],
                        mn_enc_X_train3[91],
                        mn_enc_X_train3[46],
                        mn_enc_X_train3[33]]


         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org3_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key3_1[i], cmap='viridis')
             ax.set_title("encrypted with key 3")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org3_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()
```

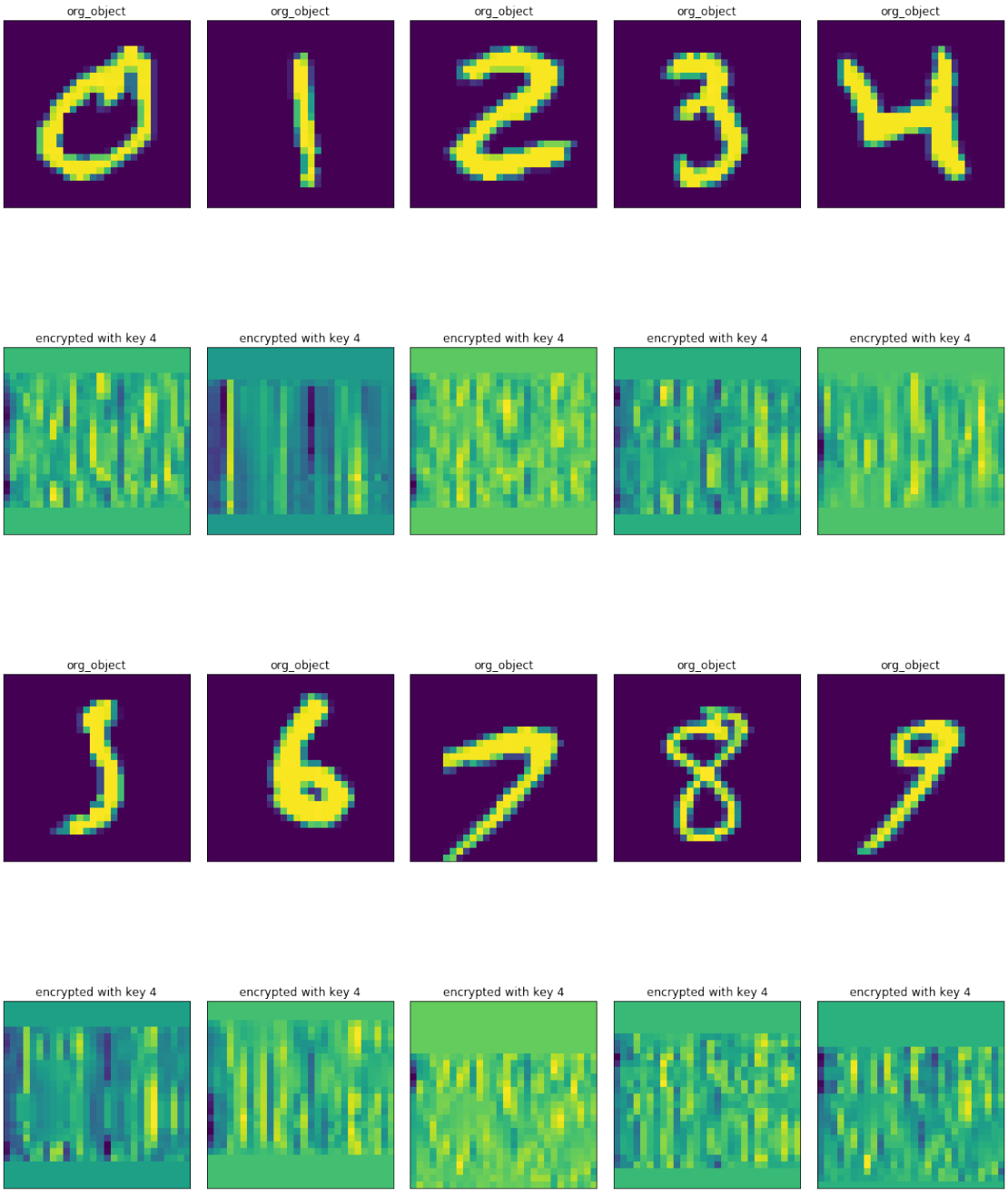
```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_key3_2[i], cmap='viridis')
    ax.set_title("encrypted with key 3")
plt.tight_layout()
plt.show()
```



Key 4

```
In [46]: index = [0,1,2,3,4]
         list_org4_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org4_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key4_1 = [mn_enc_X_train4[21],
                        mn_enc_X_train4[8],
                        mn_enc_X_train4[25],
                        mn_enc_X_train4[74],
                        mn_enc_X_train4[20]]
         list_key4_2 = [mn_enc_X_train4[132],
                        mn_enc_X_train4[36],
                        mn_enc_X_train4[91],
                        mn_enc_X_train4[46],
                        mn_enc_X_train4[33]]


         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org4_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key4_1[i], cmap='viridis')
             ax.set_title("encrypted with key 4")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org4_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()
```

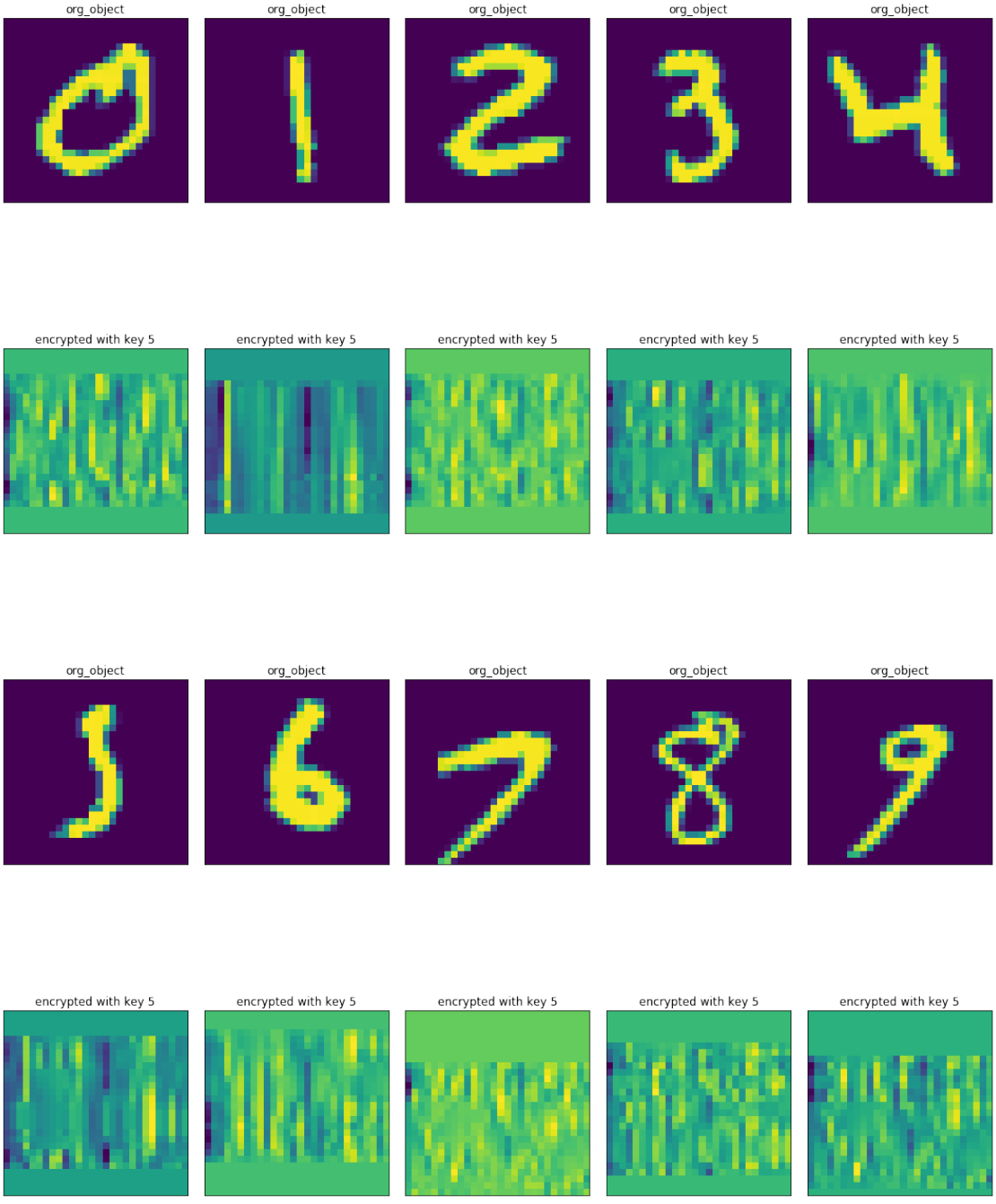
```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_key4_2[i], cmap='viridis')
    ax.set_title("encrypted with key 4")
plt.tight_layout()
plt.show()
```



Key 5

```
In [47]: index = [0,1,2,3,4]
         list_org5_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org5_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key5_1 = [mn_enc_X_train5[21],
                        mn_enc_X_train5[8],
                        mn_enc_X_train5[25],
                        mn_enc_X_train5[74],
                        mn_enc_X_train5[20]]
         list_key5_2 = [mn_enc_X_train5[132],
                        mn_enc_X_train5[36],
                        mn_enc_X_train5[91],
                        mn_enc_X_train5[46],
                        mn_enc_X_train5[33]]

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org5_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key5_1[i], cmap='viridis')
             ax.set_title("encrypted with key 5")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org5_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
```

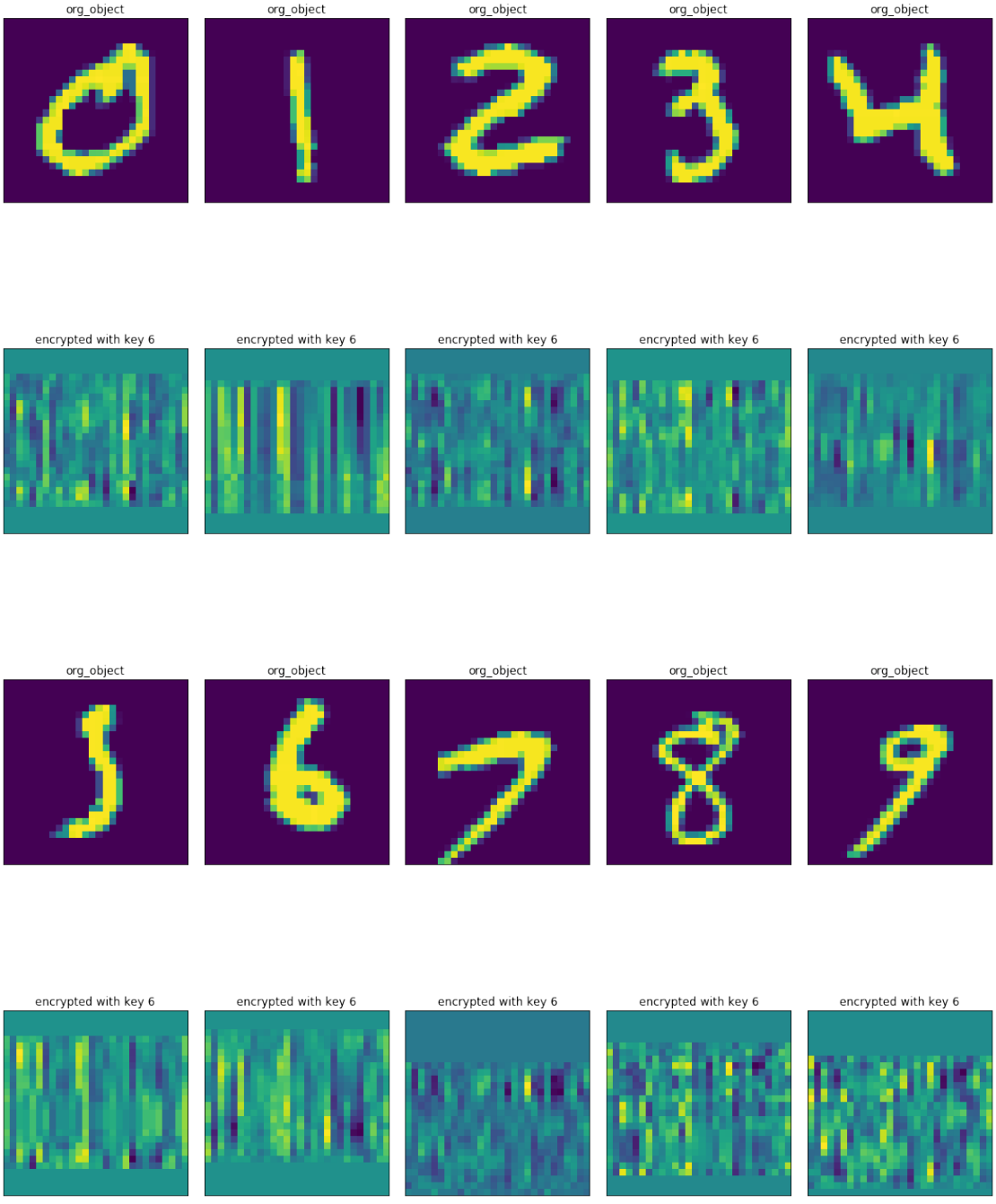
```
                                subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_key5_2[i], cmap='viridis')
    ax.set_title("encrypted with key 5")
plt.tight_layout()
plt.show()
```



Key 6

```
In [48]: index = [0,1,2,3,4]
         list_org6_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org6_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key6_1 = [mn_enc_X_train6[21],
                        mn_enc_X_train6[8],
                        mn_enc_X_train6[25],
                        mn_enc_X_train6[74],
                        mn_enc_X_train6[20]]
         list_key6_2 = [mn_enc_X_train6[132],
                        mn_enc_X_train6[36],
                        mn_enc_X_train6[91],
                        mn_enc_X_train6[46],
                        mn_enc_X_train6[33]]

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org6_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key6_1[i], cmap='viridis')
             ax.set_title("encrypted with key 6")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org6_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
```

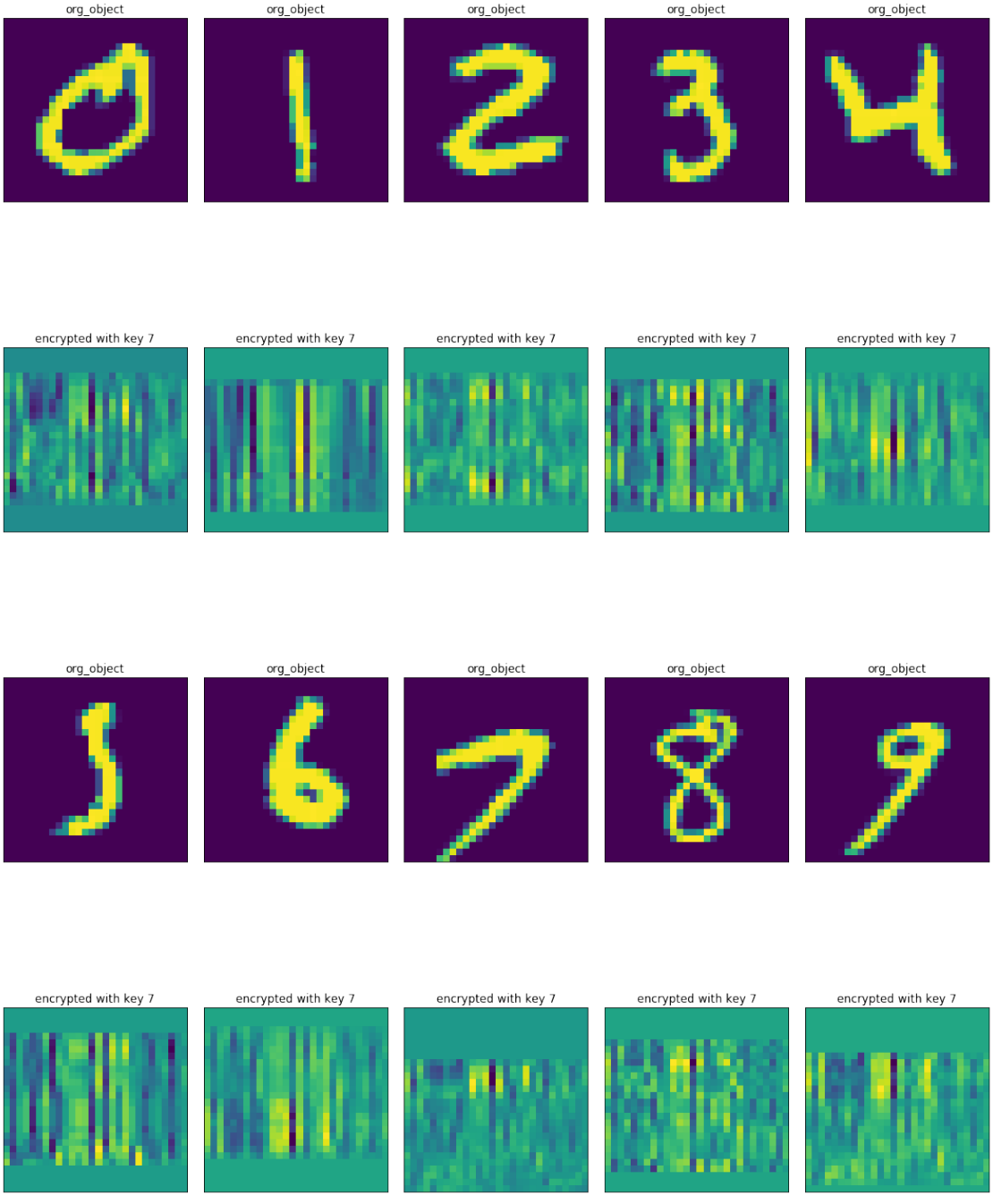
```
                            subplot_kw={'xticks': [], 'yticks': []})
    for ax, i in zip(axs.flat, index):
        ax.imshow(list_key6_2[i], cmap='viridis')
        ax.set_title("encrypted with key 6")
    plt.tight_layout()
    plt.show()
```



Key 7

```
In [49]: index = [0,1,2,3,4]
         list_org7_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org7_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key7_1 = [mn_enc_X_train7[21],
                        mn_enc_X_train7[8],
                        mn_enc_X_train7[25],
                        mn_enc_X_train7[74],
                        mn_enc_X_train7[20]]
         list_key7_2 = [mn_enc_X_train7[132],
                        mn_enc_X_train7[36],
                        mn_enc_X_train7[91],
                        mn_enc_X_train7[46],
                        mn_enc_X_train7[33]]

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org7_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key7_1[i], cmap='viridis')
             ax.set_title("encrypted with key 7")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org7_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
```

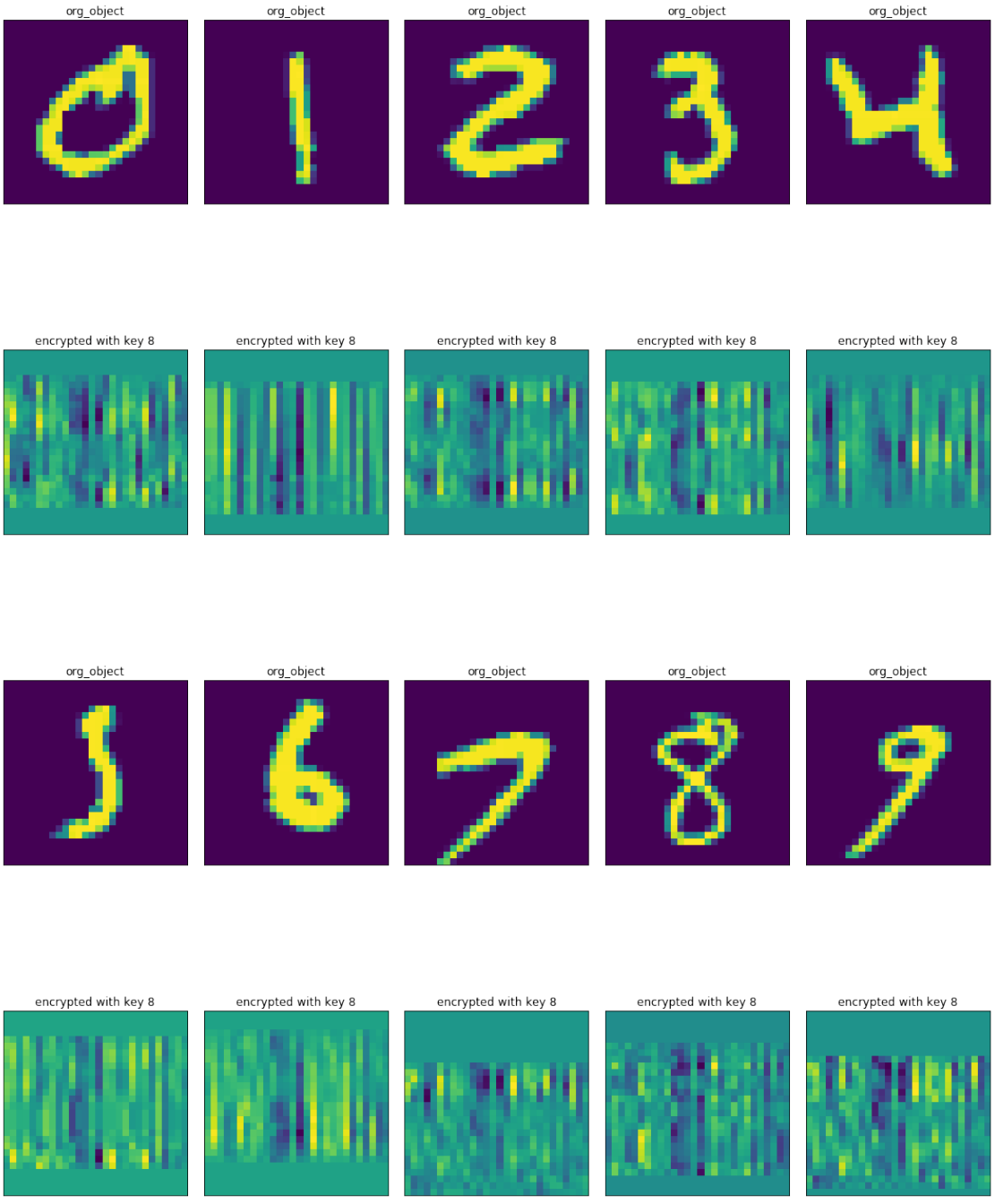
```
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_key7_2[i], cmap='viridis')
    ax.set_title("encrypted with key 7")
plt.tight_layout()
plt.show()
```



Key 8

```
In [50]: index = [0,1,2,3,4]
         list_org8_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org8_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key8_1 = [mn_enc_X_train8[21],
                        mn_enc_X_train8[8],
                        mn_enc_X_train8[25],
                        mn_enc_X_train8[74],
                        mn_enc_X_train8[20]]
         list_key8_2 = [mn_enc_X_train8[132],
                        mn_enc_X_train8[36],
                        mn_enc_X_train8[91],
                        mn_enc_X_train8[46],
                        mn_enc_X_train8[33]]

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org8_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key8_1[i], cmap='viridis')
             ax.set_title("encrypted with key 8")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org8_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
```

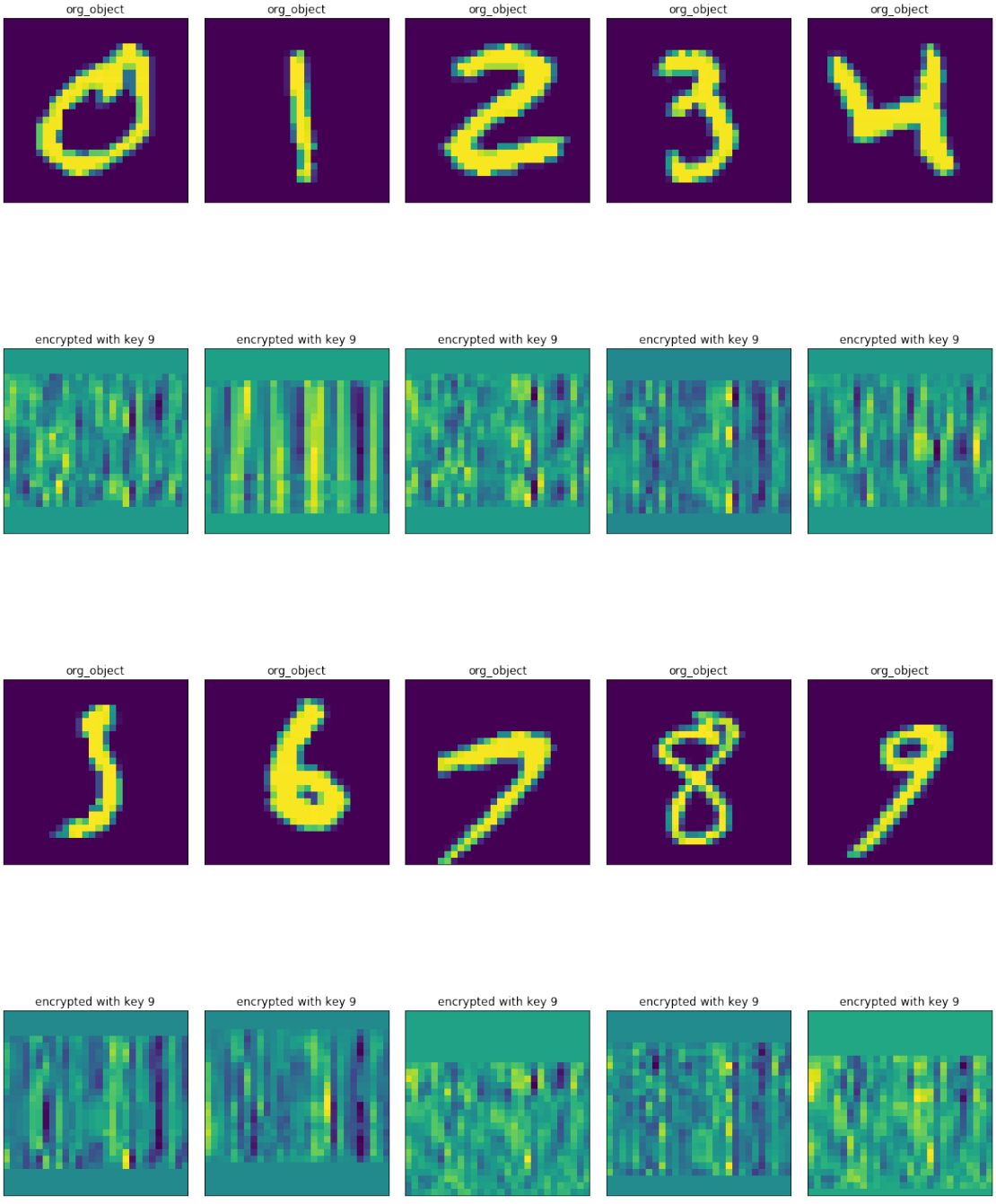
```
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_key8_2[i], cmap='viridis')
    ax.set_title("encrypted with key 8")
plt.tight_layout()
plt.show()
```



Key 9

```
In [51]: index = [0,1,2,3,4]
         list_org9_1 = [mnist_X_train[21],
                        mnist_X_train[8],
                        mnist_X_train[25],
                        mnist_X_train[74],
                        mnist_X_train[20]]
         list_org9_2 = [mnist_X_train[132],
                        mnist_X_train[36],
                        mnist_X_train[91],
                        mnist_X_train[46],
                        mnist_X_train[33]]

         list_key9_1 = [mn_enc_X_train9[21],
                        mn_enc_X_train9[8],
                        mn_enc_X_train9[25],
                        mn_enc_X_train9[74],
                        mn_enc_X_train9[20]]
         list_key9_2 = [mn_enc_X_train9[132],
                        mn_enc_X_train9[36],
                        mn_enc_X_train9[91],
                        mn_enc_X_train9[46],
                        mn_enc_X_train9[33]]

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org9_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key9_1[i], cmap='viridis')
             ax.set_title("encrypted with key 9")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org9_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
```

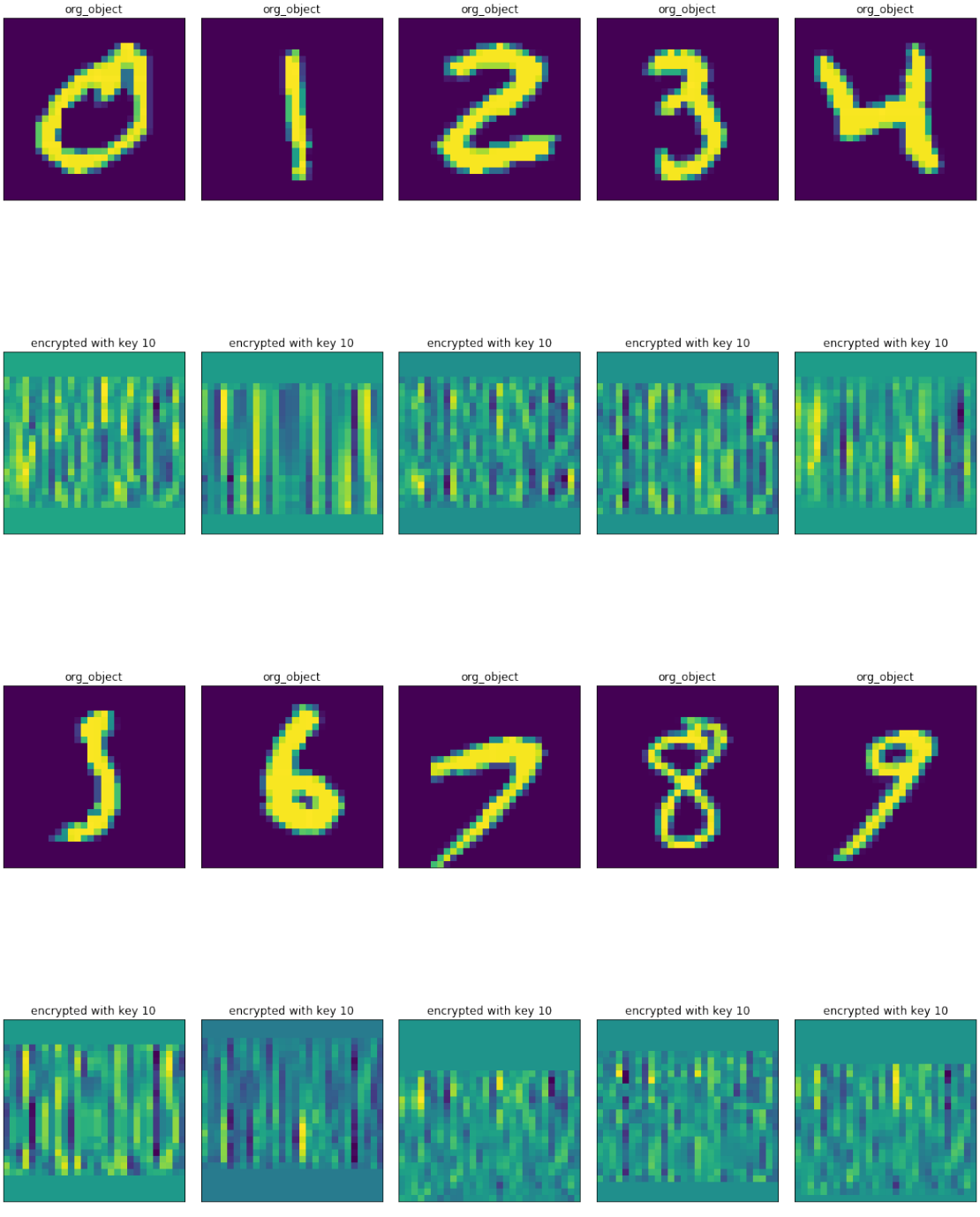
```
                                subplot_kw={'xticks': [], 'yticks': []})
    for ax, i in zip(axs.flat, index):
        ax.imshow(list_key9_2[i], cmap='viridis')
        ax.set_title("encrypted with key 9")
    plt.tight_layout()
    plt.show()
```



Key 10

```
In [53]: index = [0,1,2,3,4]
         list_org10_1 = [mnist_X_train[21],
                         mnist_X_train[8],
                         mnist_X_train[25],
                         mnist_X_train[74],
                         mnist_X_train[20]]
         list_org10_2 = [mnist_X_train[132],
                         mnist_X_train[36],
                         mnist_X_train[91],
                         mnist_X_train[46],
                         mnist_X_train[33]]

         list_key10_1 = [mn_enc_X_train10[21],
                         mn_enc_X_train10[8],
                         mn_enc_X_train10[25],
                         mn_enc_X_train10[74],
                         mn_enc_X_train10[20]]
         list_key10_2 = [mn_enc_X_train10[132],
                         mn_enc_X_train10[36],
                         mn_enc_X_train10[91],
                         mn_enc_X_train10[46],
                         mn_enc_X_train10[33]]

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org10_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_key10_1[i], cmap='viridis')
             ax.set_title("encrypted with key 10")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org10_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
```

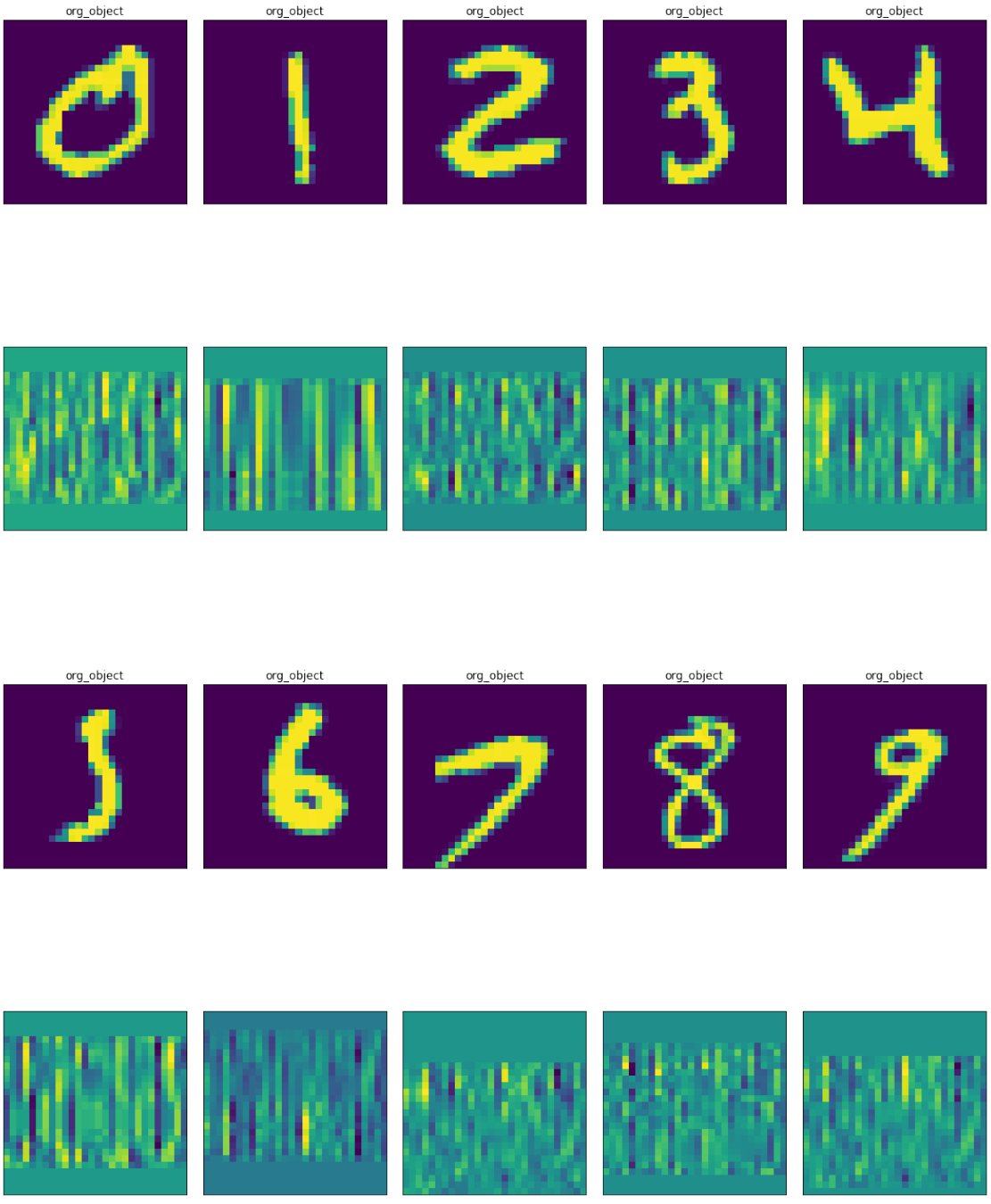
```
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_key10_2[i], cmap='viridis')
    ax.set_title("encrypted with key 10")
plt.tight_layout()
plt.show()
```



9 Different objects with different keys

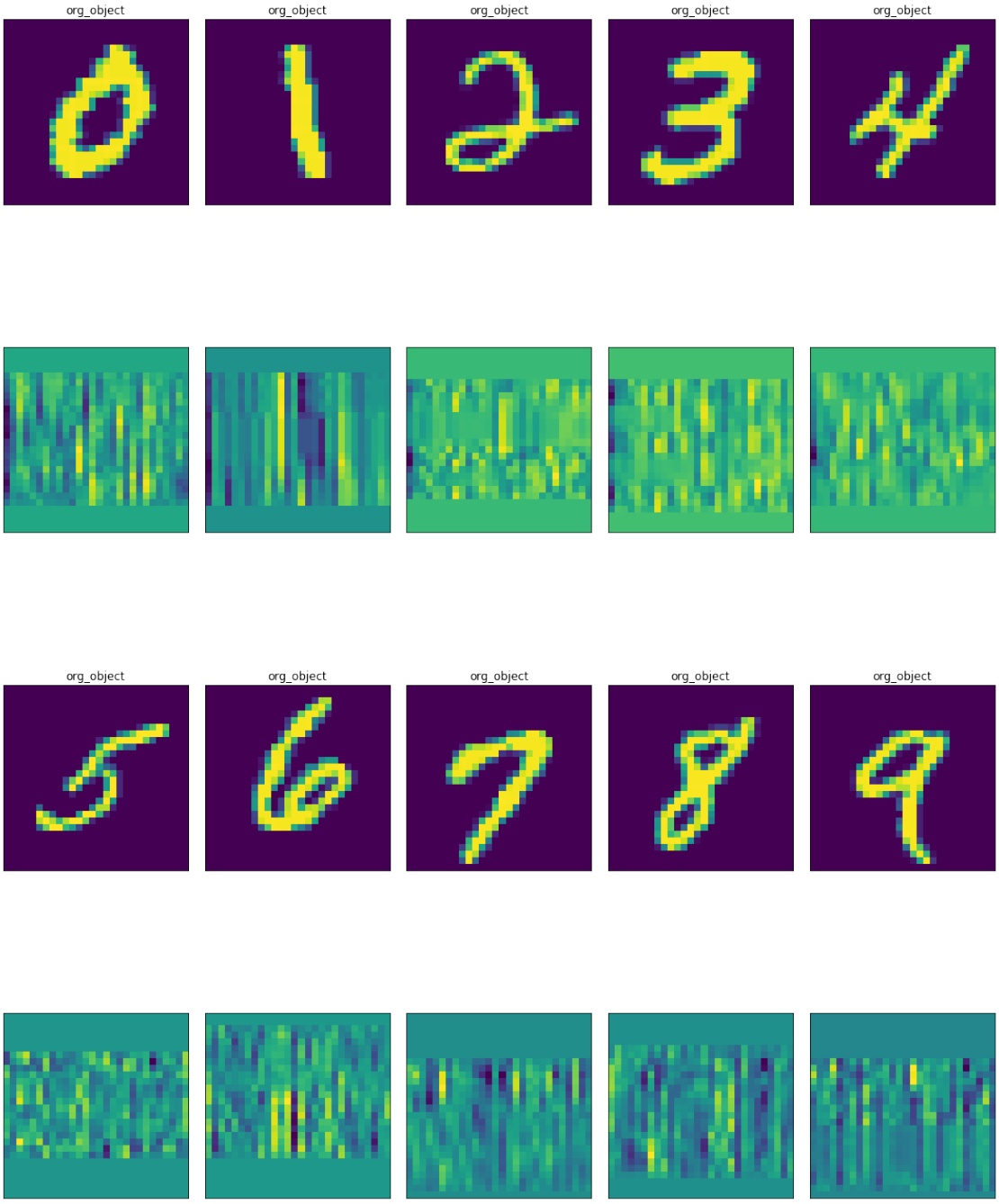
In [54]:
```
index = [0,1,2,3,4]
list_org1_1 = [mnist_X_train[21],
               mnist_X_train[8],
               mnist_X_train[25],
               mnist_X_train[74],
               mnist_X_train[20]]
list_org1_2 = [mnist_X_train[132],
               mnist_X_train[36],
               mnist_X_train[91],
               mnist_X_train[46],
               mnist_X_train[33]]

list_1_1 = [mn_enc_X_train1[21],
               mn_enc_X_train2[8],
               mn_enc_X_train3[25],
               mn_enc_X_train4[74],
               mn_enc_X_train5[20]]
list_1_2 = [mn_enc_X_train6[132],
               mn_enc_X_train7[36],
               mn_enc_X_train8[91],
               mn_enc_X_train9[46],
               mn_enc_X_train10[33]]

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_org1_1[i], cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_1_1[i], cmap='viridis')
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_org1_2[i], cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()
```

```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_1_2[i], cmap='viridis')
plt.tight_layout()
plt.show()
```



```
In [57]: index = [0,1,2,3,4]
         list_org2_1 = [mnist_X_train[37],
                        mnist_X_train[6],
                        mnist_X_train[16],
                        mnist_X_train[7],
                        mnist_X_train[9]]
         list_org2_2 = [mnist_X_train[35],
                        mnist_X_train[13],
                        mnist_X_train[15],
                        mnist_X_train[17],
                        mnist_X_train[4]]

         list_2_1 = [mn_enc_X_train1[37],
                        mn_enc_X_train2[6],
                        mn_enc_X_train3[16],
                        mn_enc_X_train4[7],
                        mn_enc_X_train5[9]]
         list_2_2 = [mn_enc_X_train6[35],
                        mn_enc_X_train7[13],
                        mn_enc_X_train8[15],
                        mn_enc_X_train9[17],
                        mn_enc_X_train10[4]]

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org2_1[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_2_1[i], cmap='viridis')
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(list_org2_2[i], cmap='viridis')
             ax.set_title("org_object")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
```

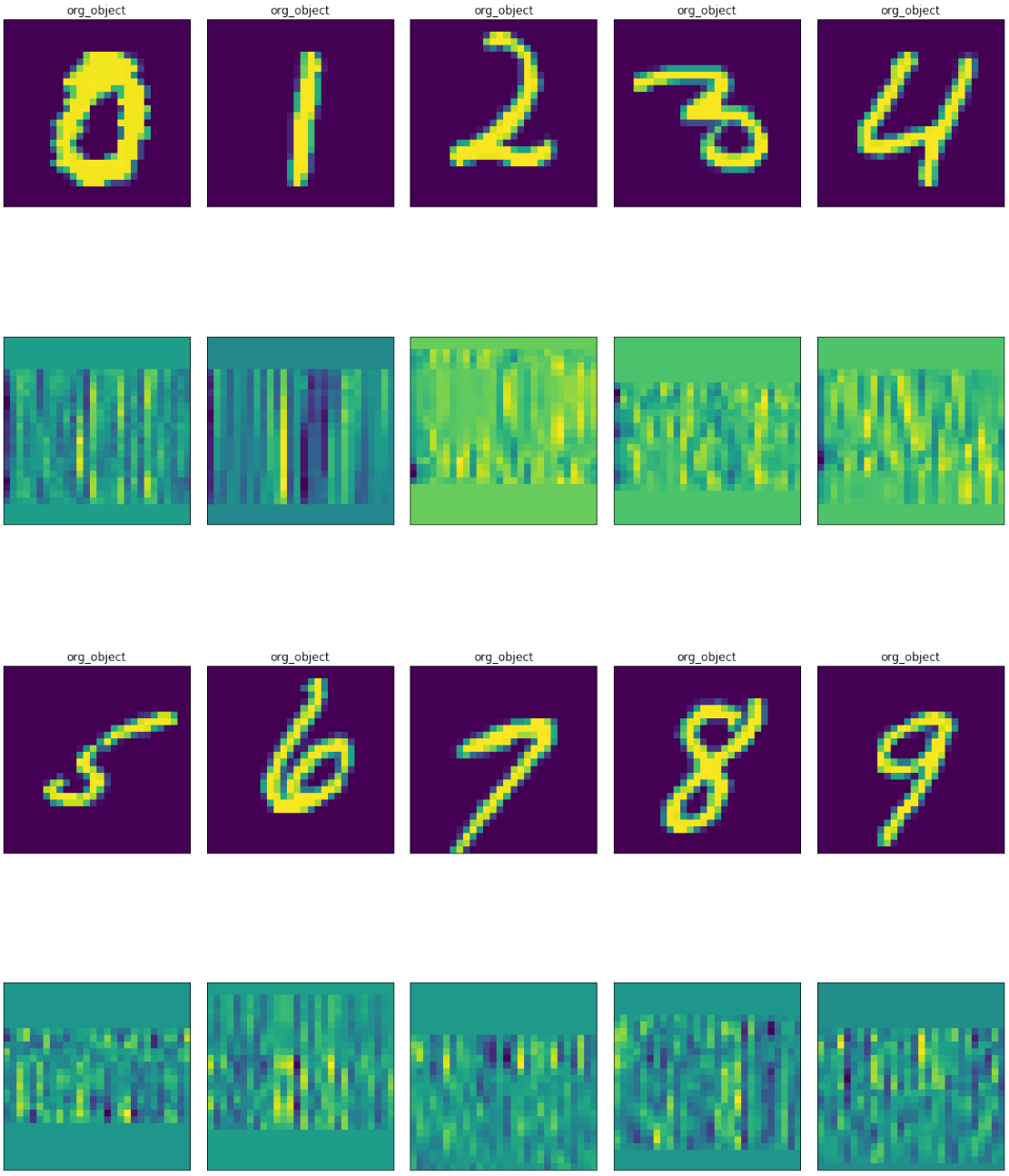
```
    for ax, i in zip(axs.flat, index):
        ax.imshow(list_2_2[i], cmap='viridis')
    plt.tight_layout()
    plt.show()
```




```
In [58]: index = [0,1,2,3,4]
         list_org3_1 = [mnist_X_train[69],
```

```
            mnist_X_train[40],
            mnist_X_train[122],
            mnist_X_train[30],
            mnist_X_train[64]]
list_org3_2 = [mnist_X_train[11],
            mnist_X_train[73],
            mnist_X_train[71],
            mnist_X_train[97],
            mnist_X_train[57]]

list_3_1 = [mn_enc_X_train1[69],
            mn_enc_X_train2[40],
            mn_enc_X_train3[122],
            mn_enc_X_train4[30],
            mn_enc_X_train5[64]]
list_3_2 = [mn_enc_X_train6[11],
            mn_enc_X_train7[73],
            mn_enc_X_train8[71],
            mn_enc_X_train9[97],
            mn_enc_X_train10[57]]

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_org3_1[i], cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_3_1[i], cmap='viridis')
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_org3_2[i], cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(list_3_2[i], cmap='viridis')
```

```
plt.tight_layout()
plt.show()
```



10 Load CIFAR10 data

```
In [60]: # The data, split between train and test sets:
         (cifar10_X_train, cifar10_Y_train), (cifar10_X_test, cifar10_Y_test) = cifar10.load_da
         print("Done loading CIFAR10 data!")
```

```
Done loading CIFAR10 data!
```

```
In [63]: cf_enc_X_train1 = []
         cf_enc_X_train2 = []
         cf_enc_X_train3 = []
         cf_enc_X_train4 = []
         cf_enc_X_train5 = []
         cf_enc_X_train6 = []
         cf_enc_X_train7 = []
         cf_enc_X_train8 = []
         cf_enc_X_train9 = []
         cf_enc_X_train10 = []

         for an_image in cifar10_X_train:
             b, g, r = cv2.split(an_image)

             enc_b1 = np.dot(b, cf_key1)
             enc_g1 = np.dot(g, cf_key1)
             enc_r1 = np.dot(r, cf_key1)
             enc_rgb1 = cv2.merge((enc_b1,enc_g1,enc_r1))
             cf_enc_X_train1.append(enc_rgb1)

             enc_b2 = np.dot(b, cf_key2)
             enc_g2 = np.dot(g, cf_key2)
             enc_r2 = np.dot(r, cf_key2)
             enc_rgb2 = cv2.merge((enc_b2,enc_g2,enc_r2))
             cf_enc_X_train2.append(enc_rgb2)

             enc_b3 = np.dot(b, cf_key3)
             enc_g3 = np.dot(g, cf_key3)
             enc_r3 = np.dot(r, cf_key3)
             enc_rgb3 = cv2.merge((enc_b3,enc_g3,enc_r3))
             cf_enc_X_train3.append(enc_rgb3)

             enc_b4 = np.dot(b, cf_key4)
             enc_g4 = np.dot(g, cf_key4)
             enc_r4 = np.dot(r, cf_key4)
             enc_rgb4 = cv2.merge((enc_b4,enc_g4,enc_r4))
             cf_enc_X_train4.append(enc_rgb4)

             enc_b5 = np.dot(b, cf_key5)
             enc_g5 = np.dot(g, cf_key5)
             enc_r5 = np.dot(r, cf_key5)
             enc_rgb5 = cv2.merge((enc_b5,enc_g5,enc_r5))
             cf_enc_X_train5.append(enc_rgb5)

             enc_b6 = np.dot(b, cf_key6)
```

```
            enc_g6 = np.dot(g, cf_key6)
            enc_r6 = np.dot(r, cf_key6)
            enc_rgb6 = cv2.merge((enc_b6,enc_g6,enc_r6))
            cf_enc_X_train6.append(enc_rgb6)

            enc_b7 = np.dot(b, cf_key7)
            enc_g7 = np.dot(g, cf_key7)
            enc_r7 = np.dot(r, cf_key7)
            enc_rgb7 = cv2.merge((enc_b7,enc_g7,enc_r7))
            cf_enc_X_train7.append(enc_rgb7)

            enc_b8 = np.dot(b, cf_key8)
            enc_g8 = np.dot(g, cf_key8)
            enc_r8 = np.dot(r, cf_key8)
            enc_rgb8 = cv2.merge((enc_b8,enc_g8,enc_r8))
            cf_enc_X_train8.append(enc_rgb8)

            enc_b9 = np.dot(b, cf_key9)
            enc_g9 = np.dot(g, cf_key9)
            enc_r9 = np.dot(r, cf_key9)
            enc_rgb9 = cv2.merge((enc_b9,enc_g9,enc_r9))
            cf_enc_X_train9.append(enc_rgb9)

            enc_b10 = np.dot(b, cf_key10)
            enc_g10 = np.dot(g, cf_key10)
            enc_r10 = np.dot(r, cf_key10)
            enc_rgb10 = cv2.merge((enc_b10,enc_g10,enc_r10))
            cf_enc_X_train10.append(enc_rgb10)
        print("Done encrypting!")
```

```
Done encrypting!
```

Get list of object indexes

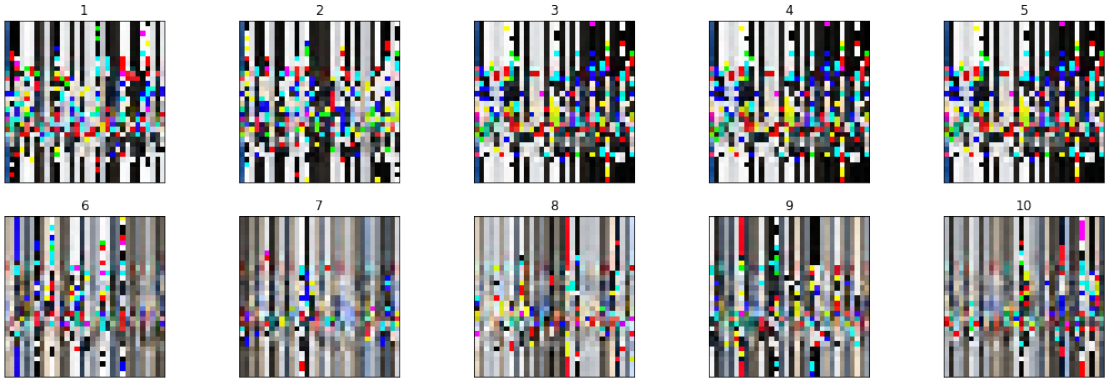
```
In [66]: list_airplane = []
         list_automobile = []
         list_bird = []
         list_cat = []
         list_deer = []
         list_dog = []
         list_frog = []
         list_horse = []
         list_ship = []
         list_truck = []

         count_airplane = 0
         count_automobile = 0
```

```
count_bird = 0
count_cat = 0
count_deer = 0
count_dog = 0
count_frog = 0
count_horse = 0
count_ship = 0
count_truck = 0

for i in range(len(cifar10_Y_train)):
    if cifar10_Y_train[i] == 0 and count_airplane < 10:
        list_airplane.append(i)
        count_airplane +=1

    elif cifar10_Y_train[i] == 1 and count_automobile < 10:
        list_automobile.append(i)
        count_automobile +=1

    elif cifar10_Y_train[i] == 2 and count_bird < 10:
        list_bird.append(i)
        count_bird +=1

    elif cifar10_Y_train[i] == 3 and count_cat < 10:
        list_cat.append(i)
        count_cat +=1

    elif cifar10_Y_train[i] == 4 and count_deer < 10:
        list_deer.append(i)
        count_deer +=1

    elif cifar10_Y_train[i] == 5 and count_dog < 10:
        list_dog.append(i)
        count_dog +=1

    elif cifar10_Y_train[i] == 6 and count_frog < 10:
        list_frog.append(i)
        count_frog +=1

    elif cifar10_Y_train[i] == 7 and count_horse < 10:
        list_horse.append(i)
        count_horse +=1

    elif cifar10_Y_train[i] == 8 and count_ship < 10:
        list_ship.append(i)
        count_ship +=1

    elif cifar10_Y_train[i] == 9 and count_truck < 10:
        list_truck.append(i)
```

```
                count_truck +=1
```

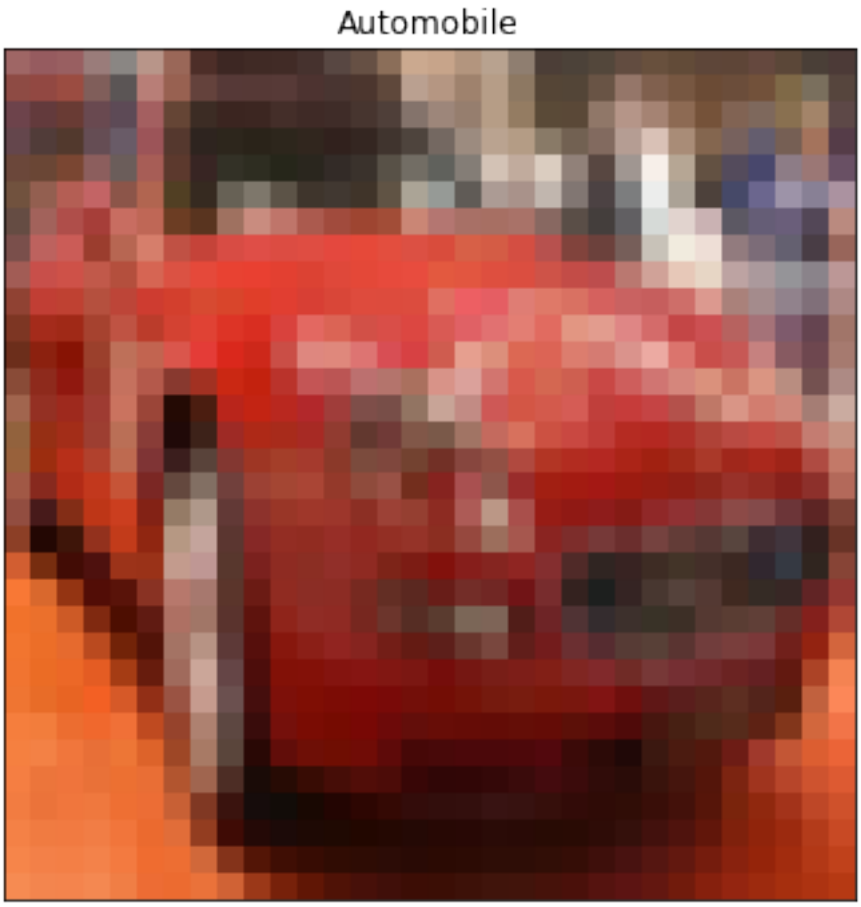
11 Same object with different keys

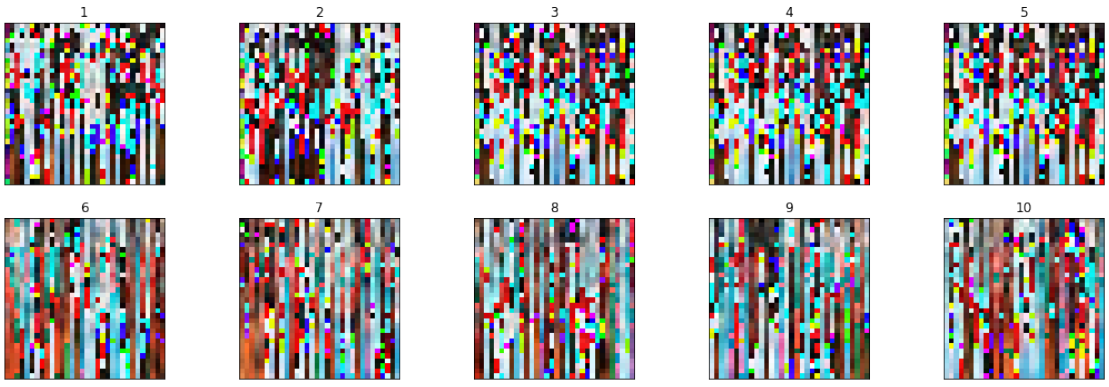
```
In [93]: index = [0,1,2,3,4,5,6,7,8,9]
         cf_obj1 = [cf_enc_X_train1[30],
                    cf_enc_X_train2[30],
                    cf_enc_X_train3[30],
                    cf_enc_X_train4[30],
                    cf_enc_X_train5[30],
                    cf_enc_X_train6[30],
                    cf_enc_X_train7[30],
                    cf_enc_X_train8[30],
                    cf_enc_X_train9[30],
                    cf_enc_X_train10[30]]

         fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         plt.imshow(cifar10_X_train[30].astype('uint8'), cmap='viridis')
         plt.title("Airplane")
         plt.tight_layout()
         plt.show()

         fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                                 subplot_kw={'xticks': [], 'yticks': []})
         for ax, i in zip(axs.flat, index):
             ax.imshow(cf_obj1[i].astype('uint8'), cmap='viridis')
             ax.set_title(i+1)
         plt.tight_layout()
         plt.show()
```




```
In [94]: index = [0,1,2,3,4,5,6,7,8,9]
         cf_obj2 = [cf_enc_X_train1[5],
                    cf_enc_X_train2[5],
```

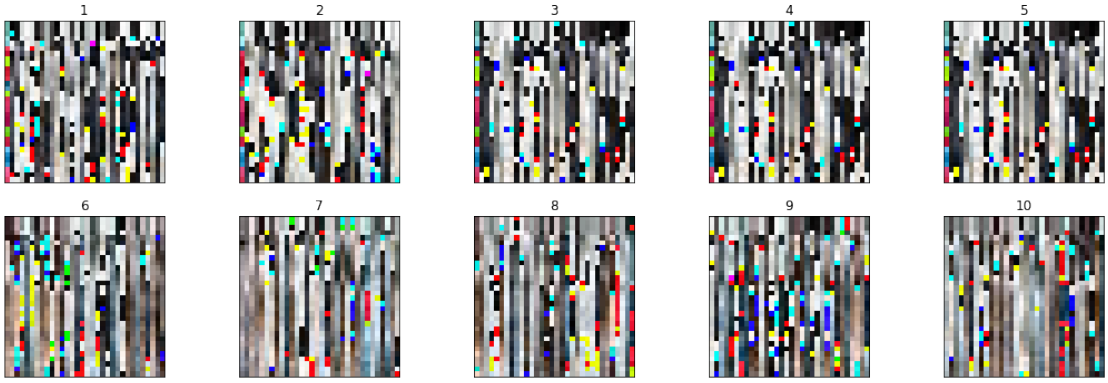
```
                cf_enc_X_train3[5],
                cf_enc_X_train4[5],
                cf_enc_X_train5[5],
                cf_enc_X_train6[5],
                cf_enc_X_train7[5],
                cf_enc_X_train8[5],
                cf_enc_X_train9[5],
                cf_enc_X_train10[5]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[5].astype('uint8'), cmap='viridis')
plt.title("Automobile")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj2[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```





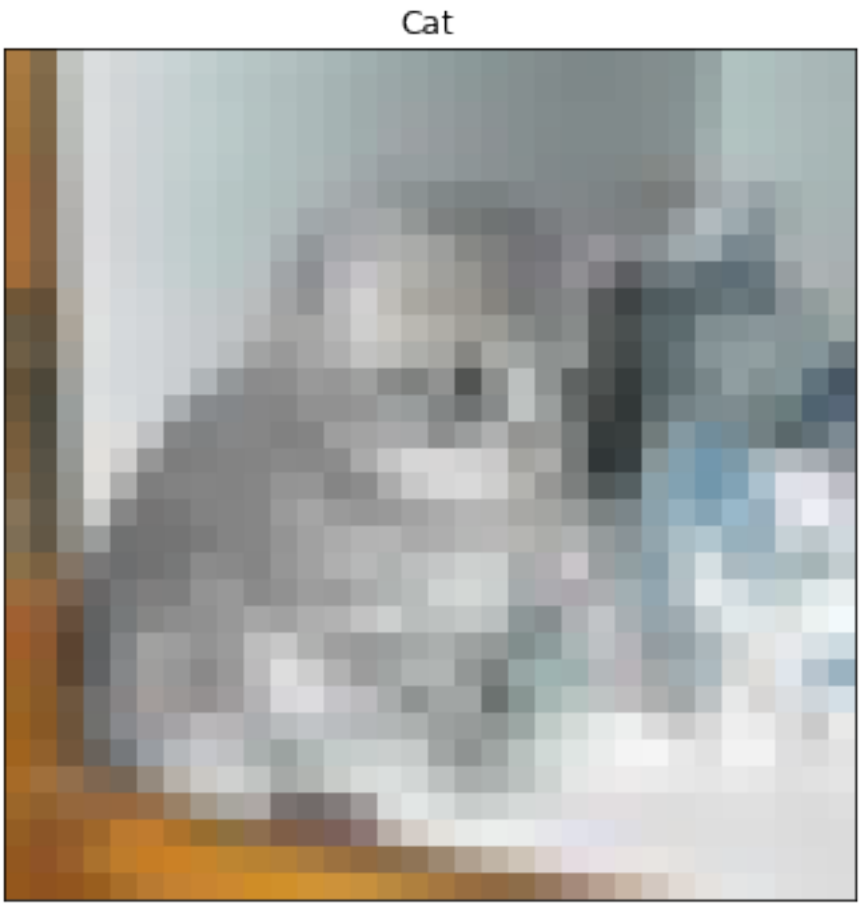

```
In [95]: index = [0,1,2,3,4,5,6,7,8,9]
         cf_obj3 = [cf_enc_X_train1[18],
                    cf_enc_X_train2[18],
```

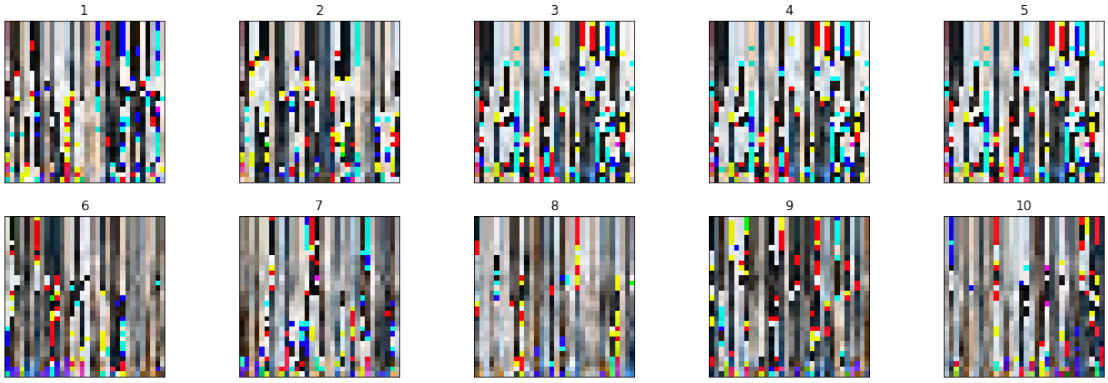
```
            cf_enc_X_train3[18],
            cf_enc_X_train4[18],
            cf_enc_X_train5[18],
            cf_enc_X_train6[18],
            cf_enc_X_train7[18],
            cf_enc_X_train8[18],
            cf_enc_X_train9[18],
            cf_enc_X_train10[18]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[18].astype('uint8'), cmap='viridis')
plt.title("Bird")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj3[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```




```
In [116]: index = [0,1,2,3,4,5,6,7,8,9]
          cf_obj4 = [cf_enc_X_train1[36],
                     cf_enc_X_train2[36],
```

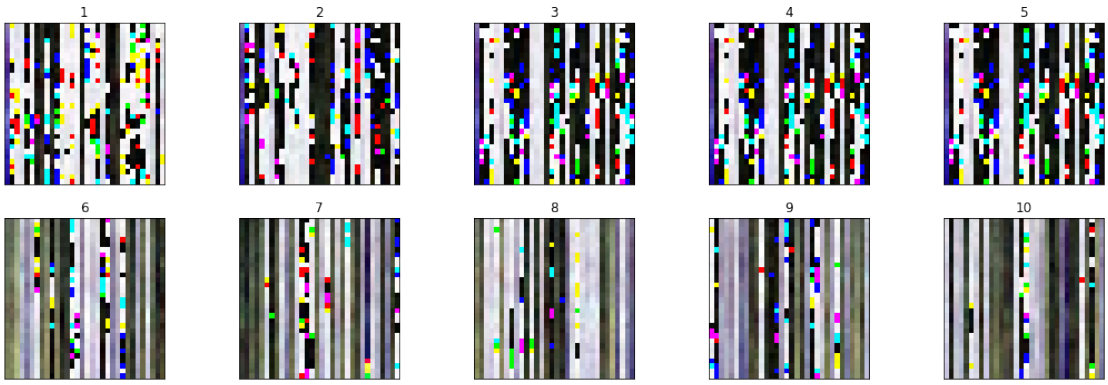
```
            cf_enc_X_train3[36],
            cf_enc_X_train4[36],
            cf_enc_X_train5[36],
            cf_enc_X_train6[36],
            cf_enc_X_train7[36],
            cf_enc_X_train8[36],
            cf_enc_X_train9[36],
            cf_enc_X_train10[36]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[36].astype('uint8'), cmap='viridis')
plt.title("Cat")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj4[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```





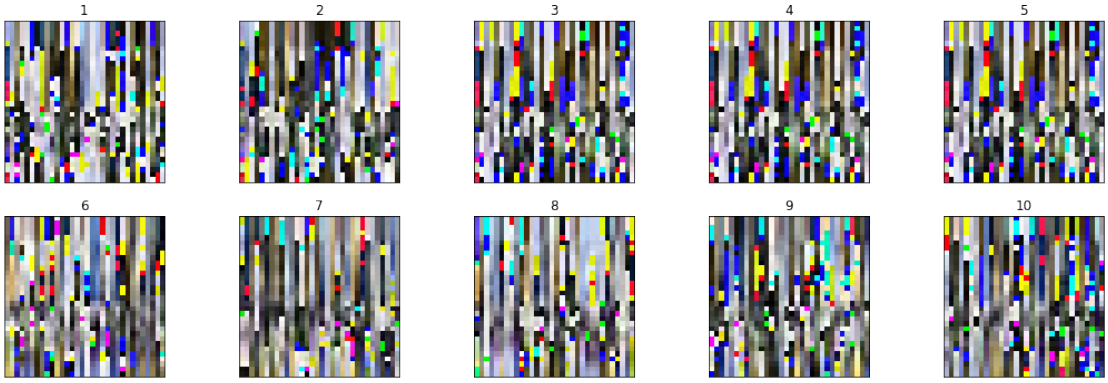

```
In [97]: index = [0,1,2,3,4,5,6,7,8,9]
         cf_obj5 = [cf_enc_X_train1[10],
                    cf_enc_X_train2[10],
```

```
                cf_enc_X_train3[10],
                cf_enc_X_train4[10],
                cf_enc_X_train5[10],
                cf_enc_X_train6[10],
                cf_enc_X_train7[10],
                cf_enc_X_train8[10],
                cf_enc_X_train9[10],
                cf_enc_X_train10[10]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[10].astype('uint8'), cmap='viridis')
plt.title("Deer")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj5[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```



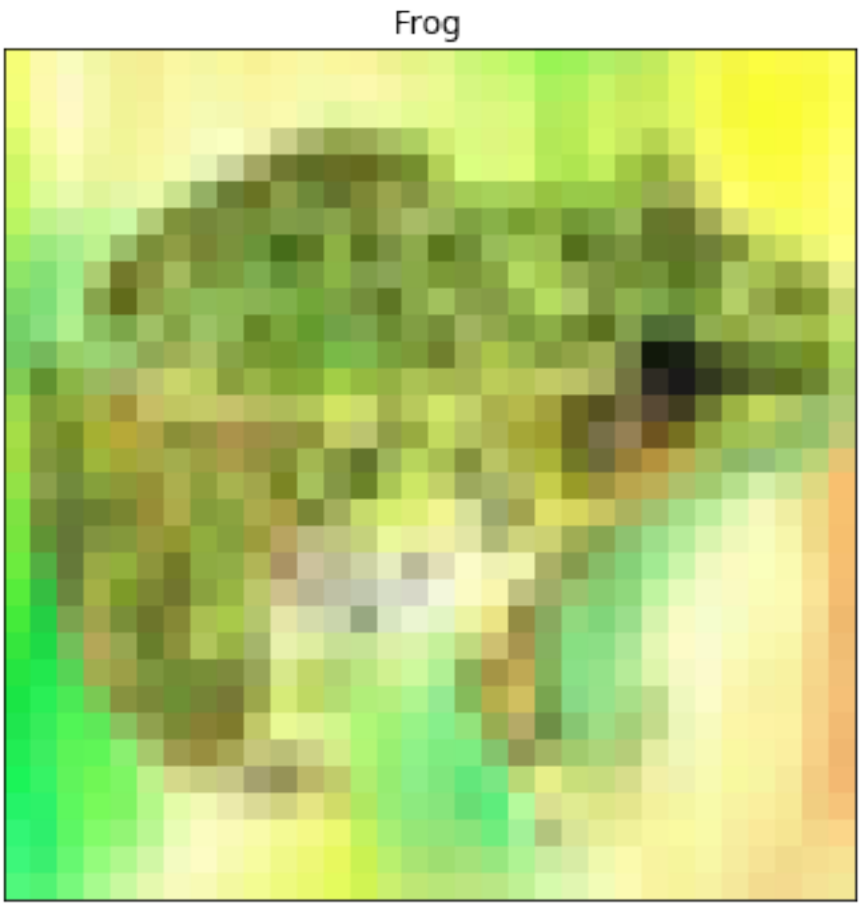

```
In [98]: index = [0,1,2,3,4,5,6,7,8,9]
         cf_obj6 = [cf_enc_X_train1[128],
                    cf_enc_X_train2[128],
```

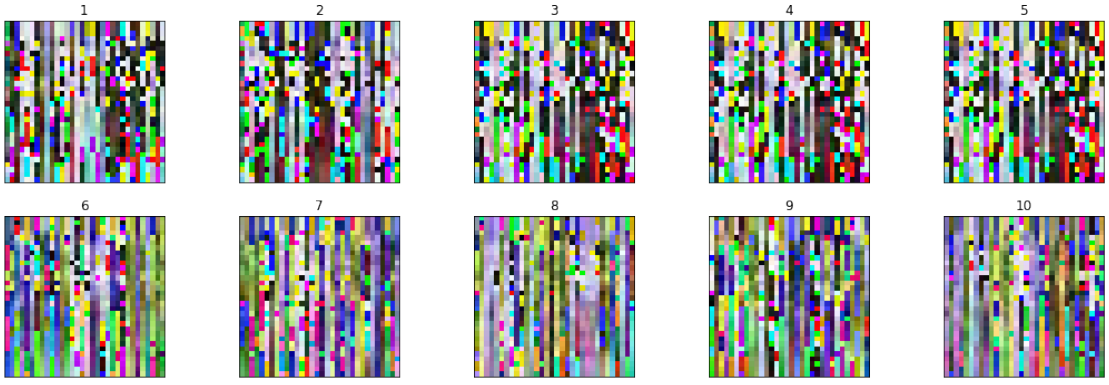
```
                cf_enc_X_train3[128],
                cf_enc_X_train4[128],
                cf_enc_X_train5[128],
                cf_enc_X_train6[128],
                cf_enc_X_train7[128],
                cf_enc_X_train8[128],
                cf_enc_X_train9[128],
                cf_enc_X_train10[128]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[128].astype('uint8'), cmap='viridis')
plt.title("Dog")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj6[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```




```
In [99]: index = [0,1,2,3,4,5,6,7,8,9]
         cf_obj7 = [cf_enc_X_train1[95],
                    cf_enc_X_train2[95],
```

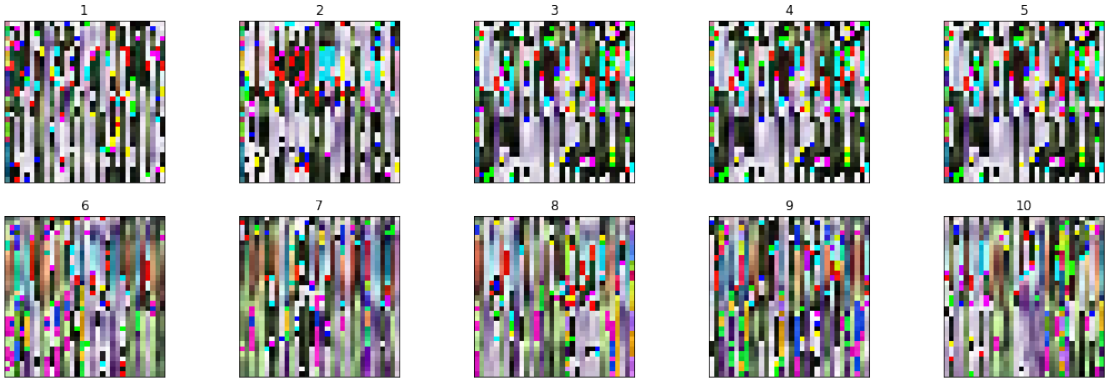
```
                cf_enc_X_train3[95],
                cf_enc_X_train4[95],
                cf_enc_X_train5[95],
                cf_enc_X_train6[95],
                cf_enc_X_train7[95],
                cf_enc_X_train8[95],
                cf_enc_X_train9[95],
                cf_enc_X_train10[95]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[95].astype('uint8'), cmap='viridis')
plt.title("Frog")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj7[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```





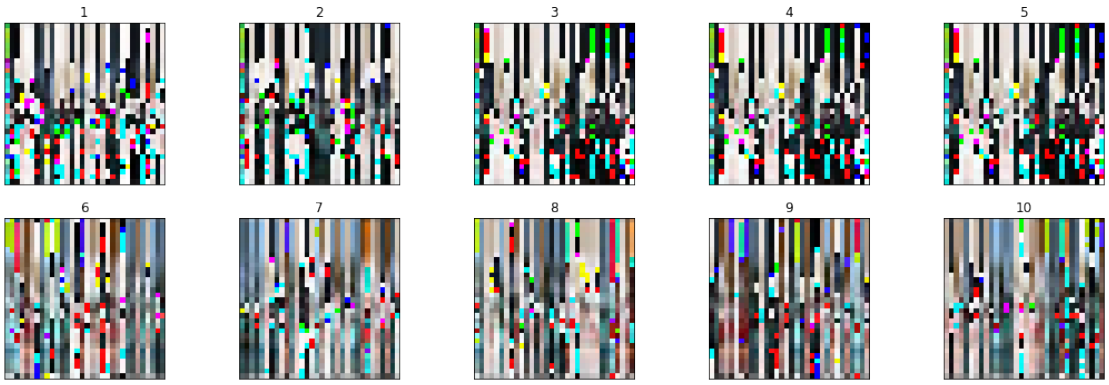

```
In [100]: index = [0,1,2,3,4,5,6,7,8,9]
          cf_obj8 = [cf_enc_X_train1[7],
                     cf_enc_X_train2[7],
```

```
            cf_enc_X_train3[7],
            cf_enc_X_train4[7],
            cf_enc_X_train5[7],
            cf_enc_X_train6[7],
            cf_enc_X_train7[7],
            cf_enc_X_train8[7],
            cf_enc_X_train9[7],
            cf_enc_X_train10[7]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[7].astype('uint8'), cmap='viridis')
plt.title("Horse")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj8[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```



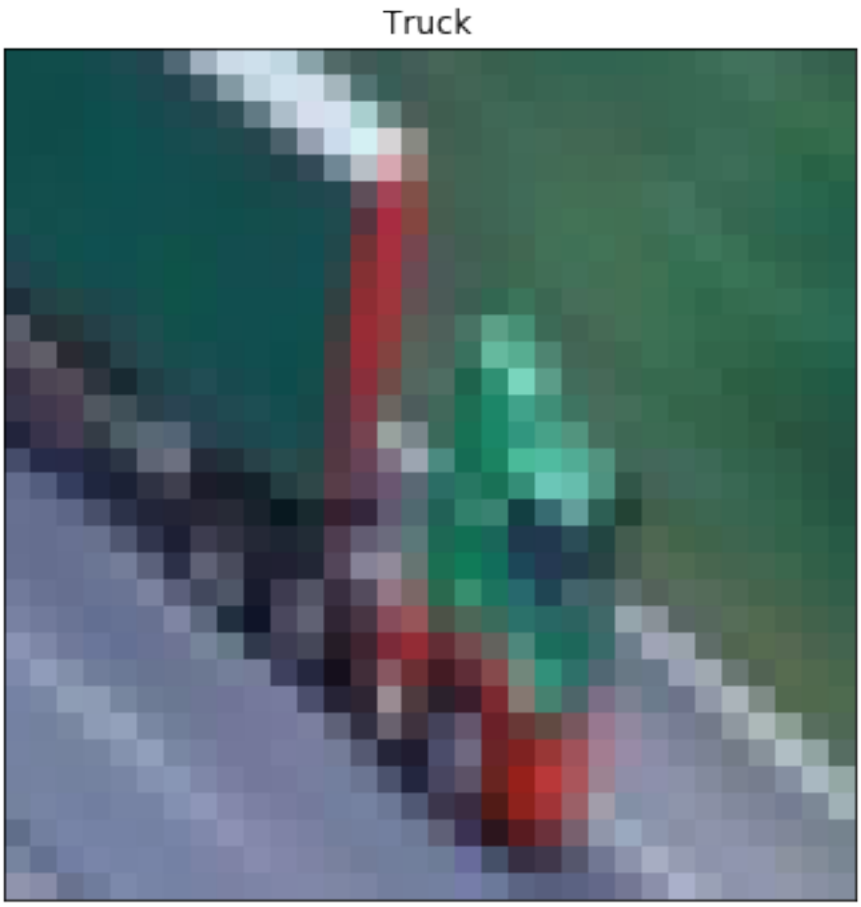

```
In [101]: index = [0,1,2,3,4,5,6,7,8,9]
          cf_obj9 = [cf_enc_X_train1[111],
                     cf_enc_X_train2[111],
```

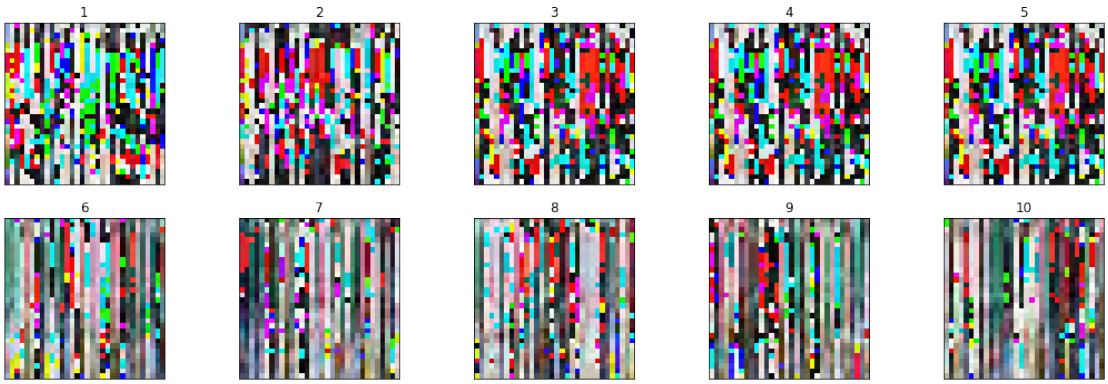
```
            cf_enc_X_train3[111],
            cf_enc_X_train4[111],
            cf_enc_X_train5[111],
            cf_enc_X_train6[111],
            cf_enc_X_train7[111],
            cf_enc_X_train8[111],
            cf_enc_X_train9[111],
            cf_enc_X_train10[111]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[111].astype('uint8'), cmap='viridis')
plt.title("Ship")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj9[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```




```
In [102]: index = [0,1,2,3,4,5,6,7,8,9]
          cf_obj10 = [cf_enc_X_train1[50],
                      cf_enc_X_train2[50],
```

```
            cf_enc_X_train3[50],
            cf_enc_X_train4[50],
            cf_enc_X_train5[50],
            cf_enc_X_train6[50],
            cf_enc_X_train7[50],
            cf_enc_X_train8[50],
            cf_enc_X_train9[50],
            cf_enc_X_train10[50]]

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
plt.imshow(cifar10_X_train[50].astype('uint8'), cmap='viridis')
plt.title("Truck")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_obj10[i].astype('uint8'), cmap='viridis')
    ax.set_title(i+1)
plt.tight_layout()
plt.show()
```



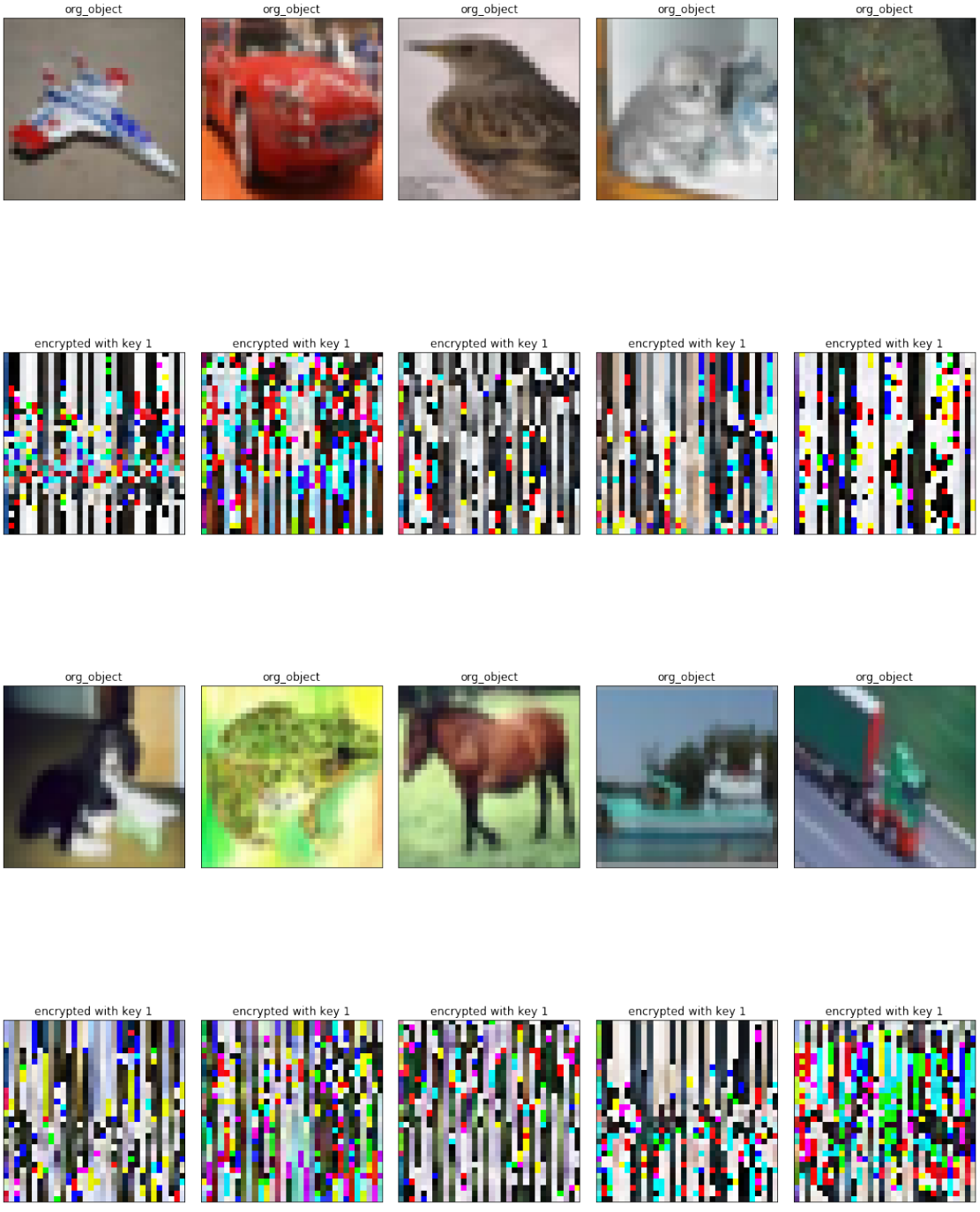


12 Different objects with same key

```
In [92]: index = [0,1,2,3,4]
         cf_org1_1 = [cifar10_X_train[30],
```

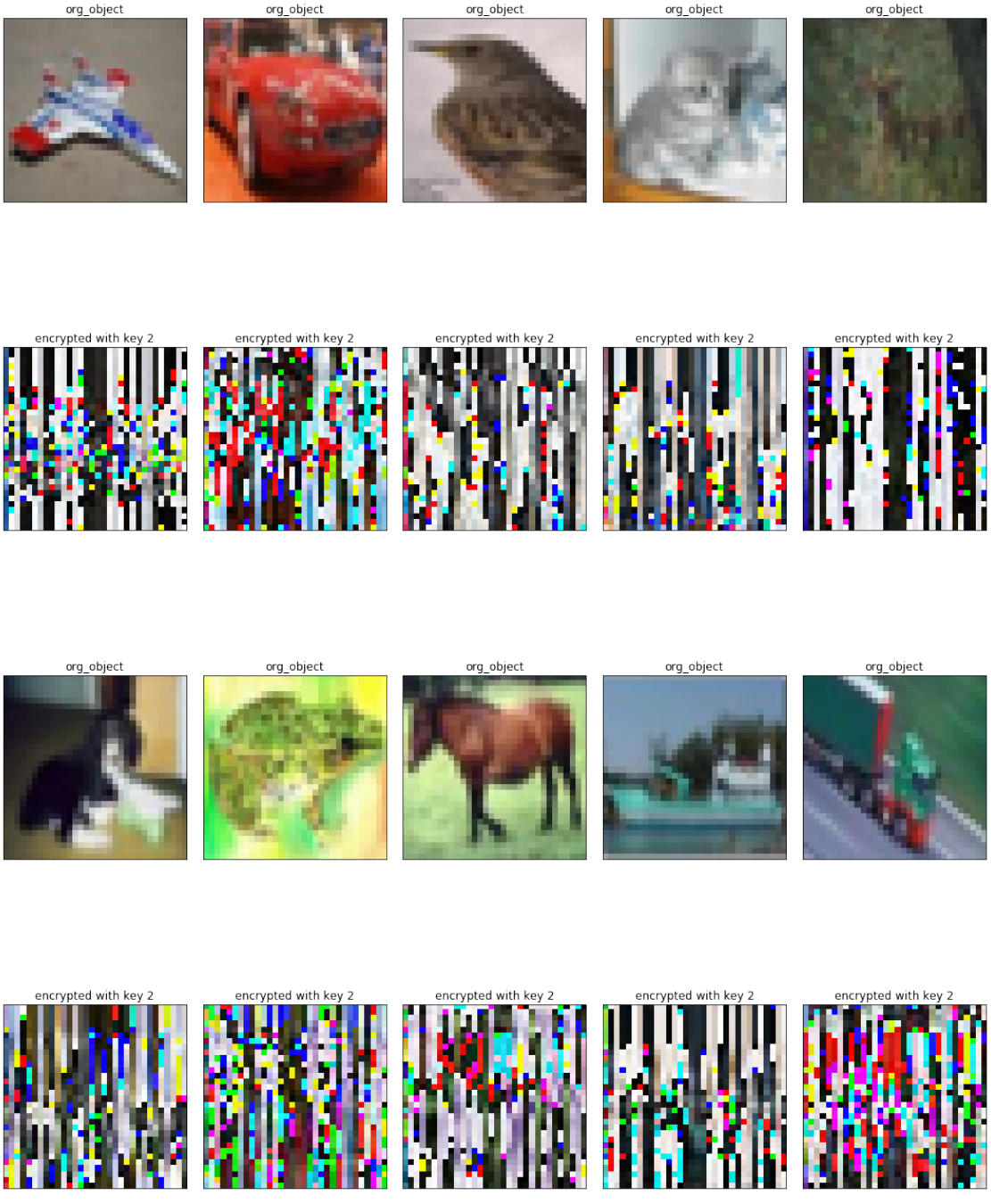
```
            cifar10_X_train[5],
            cifar10_X_train[18],
            cifar10_X_train[36],
            cifar10_X_train[10]]
cf_org1_2 = [cifar10_X_train[128],
            cifar10_X_train[95],
            cifar10_X_train[7],
            cifar10_X_train[111],
            cifar10_X_train[50]]

cf_key1_1 = [cf_enc_X_train1[30],
            cf_enc_X_train1[5],
            cf_enc_X_train1[18],
            cf_enc_X_train1[36],
            cf_enc_X_train1[10]]
cf_key1_2 = [cf_enc_X_train1[128],
            cf_enc_X_train1[95],
            cf_enc_X_train1[7],
            cf_enc_X_train1[111],
            cf_enc_X_train1[50]]


fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org1_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key1_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 1")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org1_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
```

```
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key1_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 1")
plt.tight_layout()
plt.show()
```



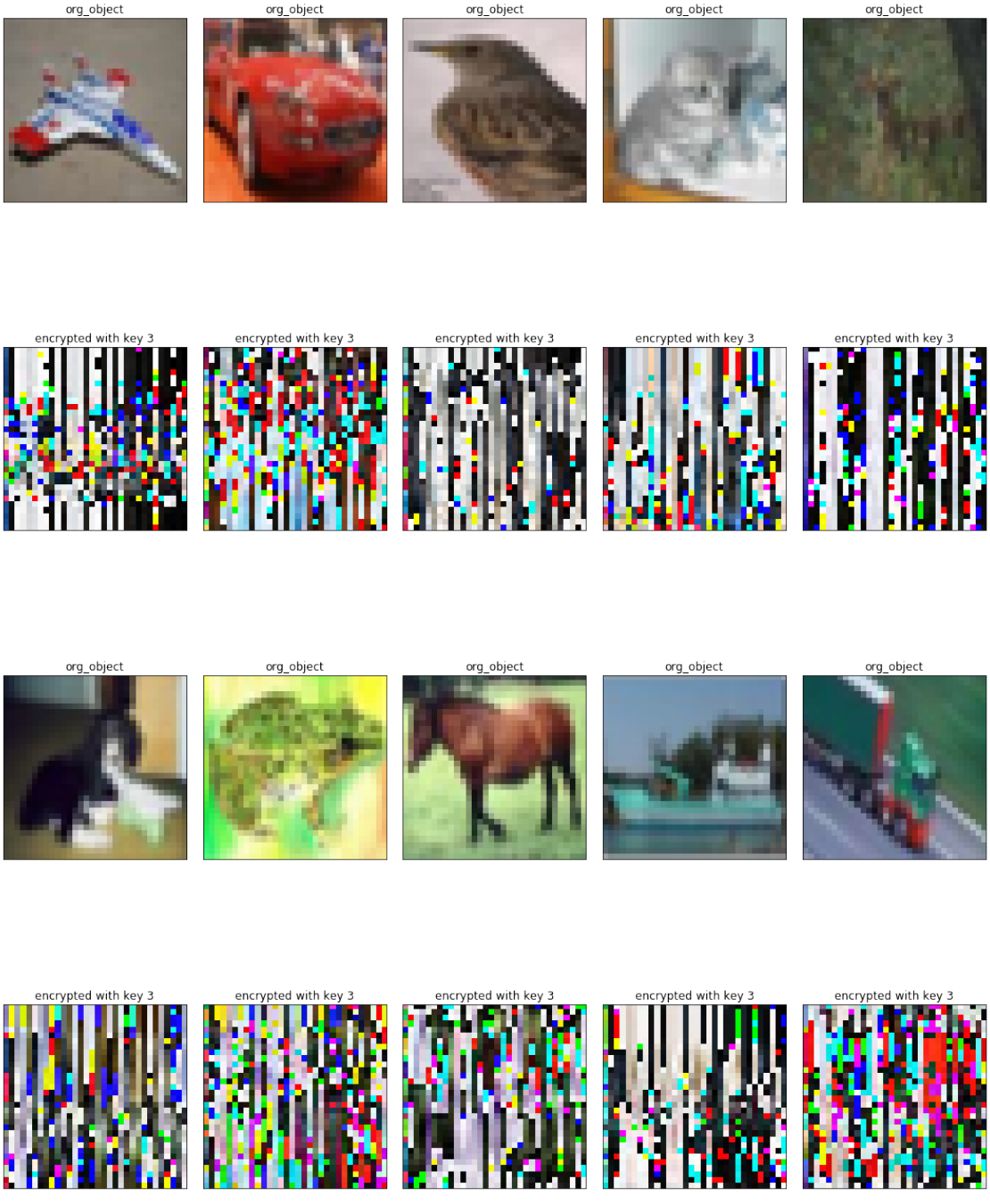
In [104]:
```
index = [0,1,2,3,4]
cf_org2_1 = [cifar10_X_train[30],
             cifar10_X_train[5],
             cifar10_X_train[18],
             cifar10_X_train[36],
             cifar10_X_train[10]]
cf_org2_2 = [cifar10_X_train[128],
             cifar10_X_train[95],
             cifar10_X_train[7],
             cifar10_X_train[111],
             cifar10_X_train[50]]

cf_key2_1 = [cf_enc_X_train2[30],
             cf_enc_X_train2[5],
             cf_enc_X_train2[18],
             cf_enc_X_train2[36],
             cf_enc_X_train2[10]]
cf_key2_2 = [cf_enc_X_train2[128],
             cf_enc_X_train2[95],
             cf_enc_X_train2[7],
             cf_enc_X_train2[111],
             cf_enc_X_train2[50]]


fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org2_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key2_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 2")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org2_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()
```

```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key2_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 2")
plt.tight_layout()
plt.show()
```



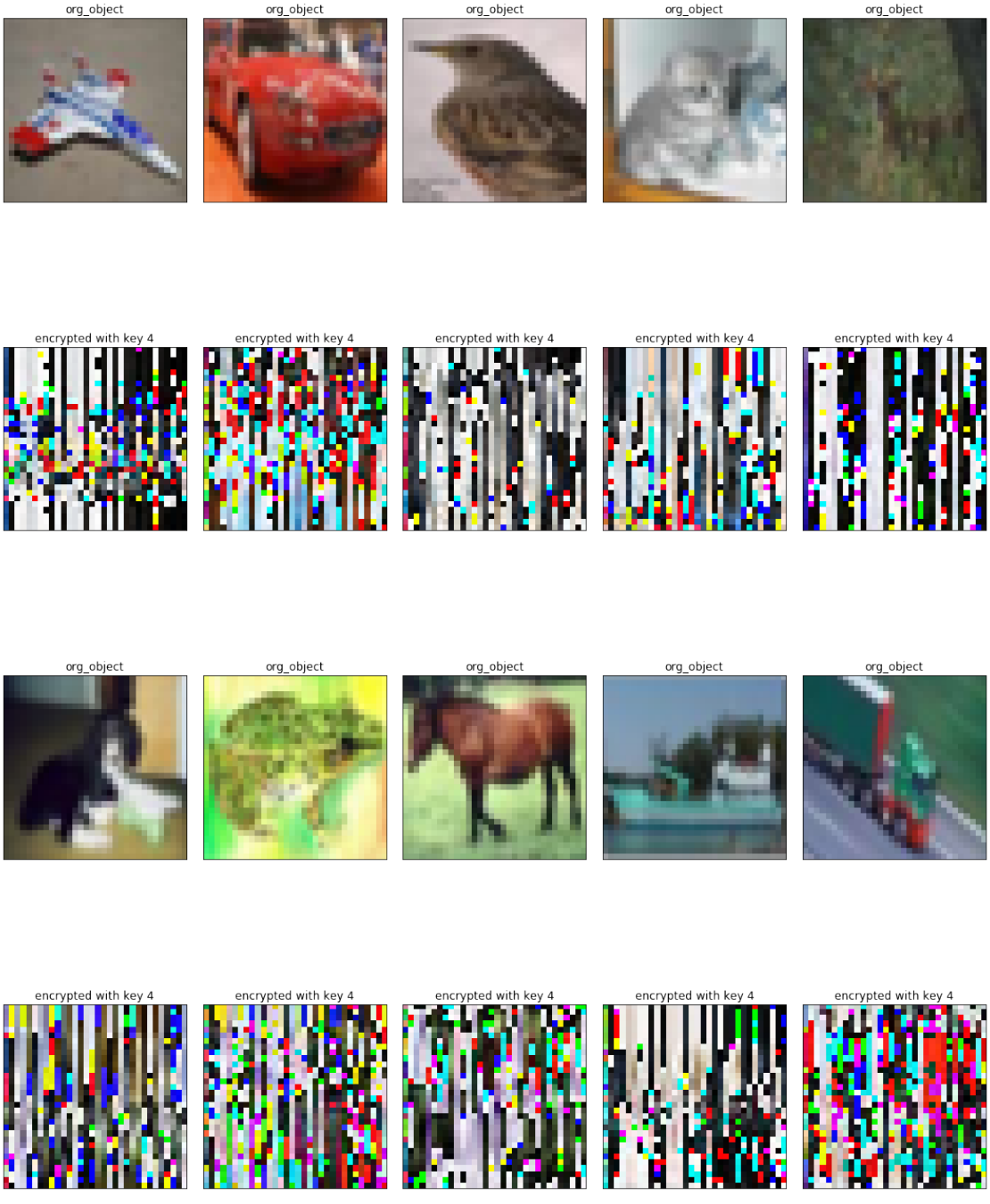
In [106]:
```
index = [0,1,2,3,4]
cf_org3_1 = [cifar10_X_train[30],
             cifar10_X_train[5],
             cifar10_X_train[18],
             cifar10_X_train[36],
             cifar10_X_train[10]]
cf_org3_2 = [cifar10_X_train[128],
             cifar10_X_train[95],
             cifar10_X_train[7],
             cifar10_X_train[111],
             cifar10_X_train[50]]

cf_key3_1 = [cf_enc_X_train3[30],
             cf_enc_X_train3[5],
             cf_enc_X_train3[18],
             cf_enc_X_train3[36],
             cf_enc_X_train3[10]]
cf_key3_2 = [cf_enc_X_train3[128],
             cf_enc_X_train3[95],
             cf_enc_X_train3[7],
             cf_enc_X_train3[111],
             cf_enc_X_train3[50]]


fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org3_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key3_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 3")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org3_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()
```

```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key3_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 3")
plt.tight_layout()
plt.show()
```



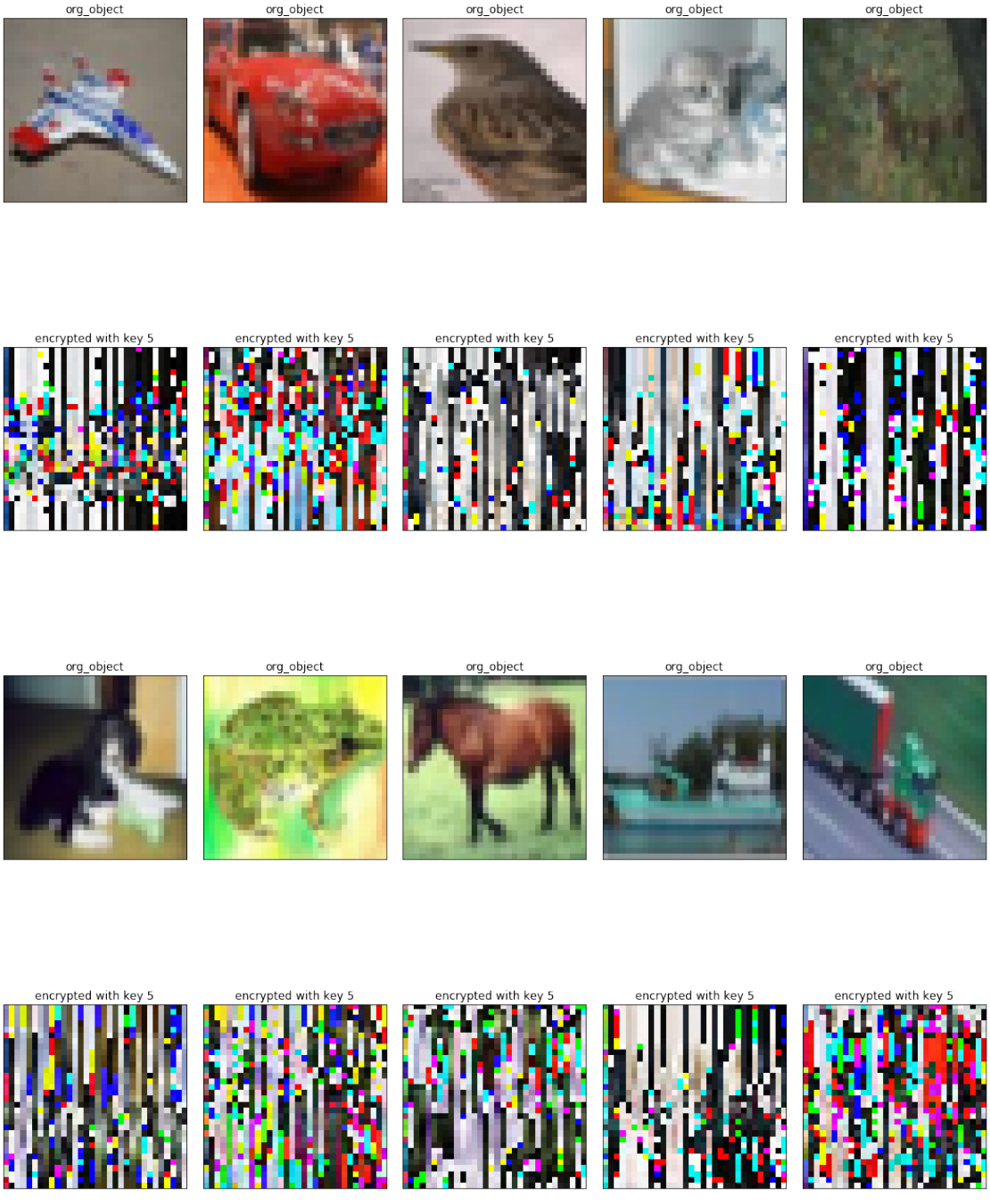
In [107]:
```
index = [0,1,2,3,4]
cf_org4_1 = [cifar10_X_train[30],
             cifar10_X_train[5],
             cifar10_X_train[18],
             cifar10_X_train[36],
             cifar10_X_train[10]]
cf_org4_2 = [cifar10_X_train[128],
             cifar10_X_train[95],
             cifar10_X_train[7],
             cifar10_X_train[111],
             cifar10_X_train[50]]

cf_key4_1 = [cf_enc_X_train4[30],
             cf_enc_X_train4[5],
             cf_enc_X_train4[18],
             cf_enc_X_train4[36],
             cf_enc_X_train4[10]]
cf_key4_2 = [cf_enc_X_train4[128],
             cf_enc_X_train4[95],
             cf_enc_X_train4[7],
             cf_enc_X_train4[111],
             cf_enc_X_train4[50]]


fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org4_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key4_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 4")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org4_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()
```

```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key4_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 4")
plt.tight_layout()
plt.show()
```



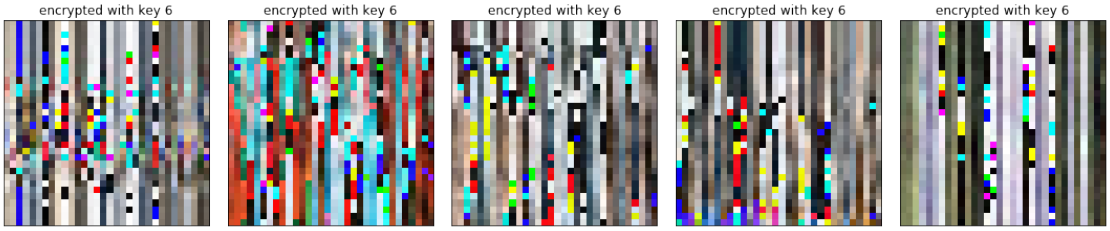
```
In [108]: index = [0,1,2,3,4]
          cf_org5_1 = [cifar10_X_train[30],
                       cifar10_X_train[5],
                       cifar10_X_train[18],
                       cifar10_X_train[36],
                       cifar10_X_train[10]]
          cf_org5_2 = [cifar10_X_train[128],
                       cifar10_X_train[95],
                       cifar10_X_train[7],
                       cifar10_X_train[111],
                       cifar10_X_train[50]]

          cf_key5_1 = [cf_enc_X_train5[30],
                       cf_enc_X_train5[5],
                       cf_enc_X_train5[18],
                       cf_enc_X_train5[36],
                       cf_enc_X_train5[10]]
          cf_key5_2 = [cf_enc_X_train5[128],
                       cf_enc_X_train5[95],
                       cf_enc_X_train5[7],
                       cf_enc_X_train5[111],
                       cf_enc_X_train5[50]]


          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org5_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_key5_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("encrypted with key 5")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org5_2[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()
```

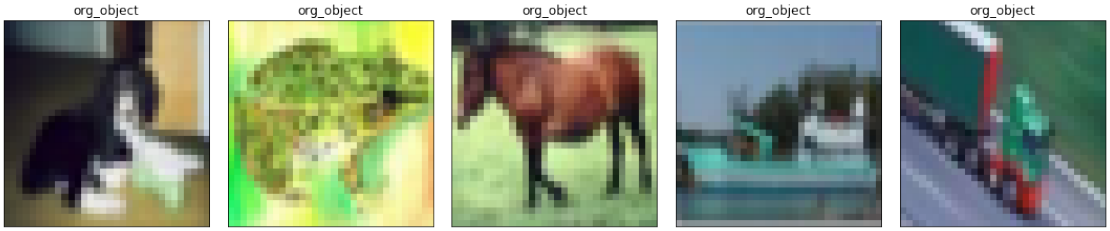
```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key5_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 5")
plt.tight_layout()
plt.show()
```



In [109]:
```
index = [0,1,2,3,4]
cf_org6_1 = [cifar10_X_train[30],
             cifar10_X_train[5],
             cifar10_X_train[18],
             cifar10_X_train[36],
             cifar10_X_train[10]]
cf_org6_2 = [cifar10_X_train[128],
             cifar10_X_train[95],
             cifar10_X_train[7],
             cifar10_X_train[111],
             cifar10_X_train[50]]

cf_key6_1 = [cf_enc_X_train6[30],
             cf_enc_X_train6[5],
             cf_enc_X_train6[18],
             cf_enc_X_train6[36],
             cf_enc_X_train6[10]]
cf_key6_2 = [cf_enc_X_train6[128],
             cf_enc_X_train6[95],
             cf_enc_X_train6[7],
             cf_enc_X_train6[111],
             cf_enc_X_train6[50]]


fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org6_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key6_1[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 6")
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_org6_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("org_object")
plt.tight_layout()
plt.show()
```

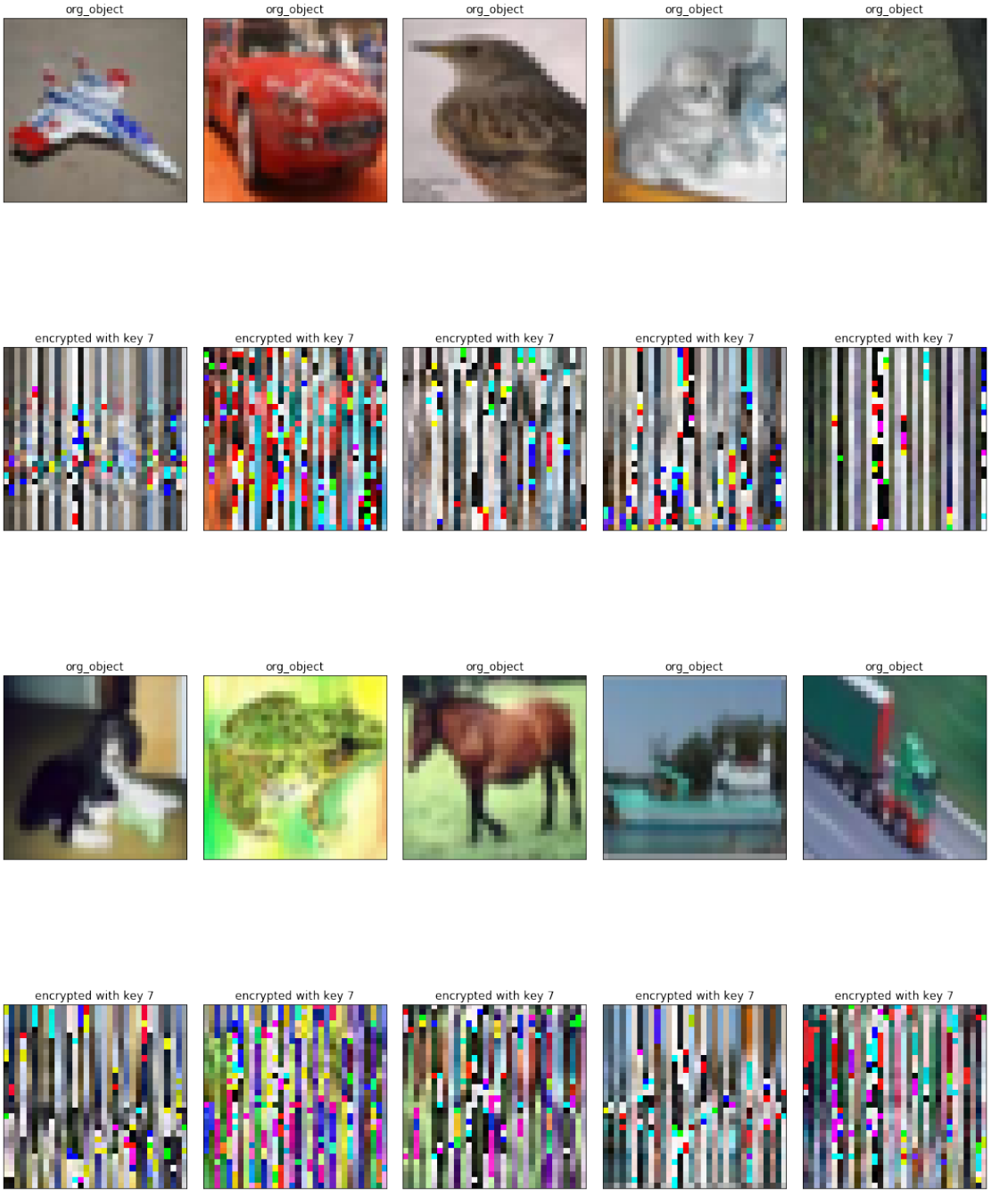
```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key6_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 6")
plt.tight_layout()
plt.show()
```





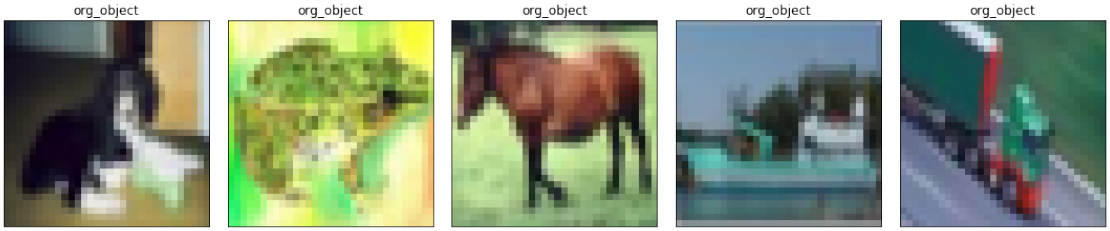
```
In [110]: index = [0,1,2,3,4]
          cf_org7_1 = [cifar10_X_train[30],
                       cifar10_X_train[5],
                       cifar10_X_train[18],
                       cifar10_X_train[36],
                       cifar10_X_train[10]]
          cf_org7_2 = [cifar10_X_train[128],
                       cifar10_X_train[95],
                       cifar10_X_train[7],
                       cifar10_X_train[111],
                       cifar10_X_train[50]]

          cf_key7_1 = [cf_enc_X_train7[30],
                       cf_enc_X_train7[5],
                       cf_enc_X_train7[18],
                       cf_enc_X_train7[36],
                       cf_enc_X_train7[10]]
          cf_key7_2 = [cf_enc_X_train7[128],
                       cf_enc_X_train7[95],
                       cf_enc_X_train7[7],
                       cf_enc_X_train7[111],
                       cf_enc_X_train7[50]]


          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org7_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_key7_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("encrypted with key 7")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org7_2[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()
```

```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key7_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 7")
plt.tight_layout()
plt.show()
```



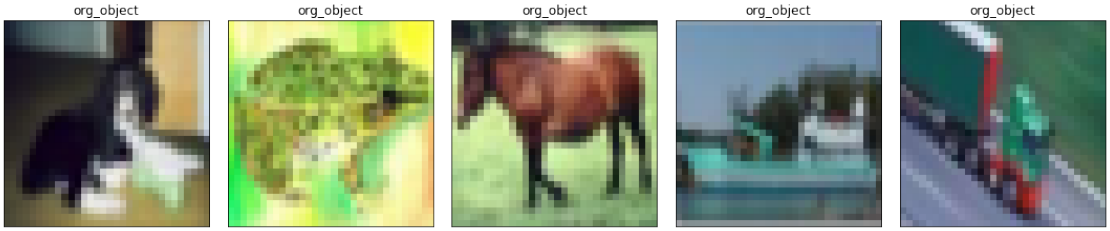
```
In [111]: index = [0,1,2,3,4]
          cf_org8_1 = [cifar10_X_train[30],
                       cifar10_X_train[5],
                       cifar10_X_train[18],
                       cifar10_X_train[36],
                       cifar10_X_train[10]]
          cf_org8_2 = [cifar10_X_train[128],
                       cifar10_X_train[95],
                       cifar10_X_train[7],
                       cifar10_X_train[111],
                       cifar10_X_train[50]]

          cf_key8_1 = [cf_enc_X_train8[30],
                       cf_enc_X_train8[5],
                       cf_enc_X_train8[18],
                       cf_enc_X_train8[36],
                       cf_enc_X_train8[10]]
          cf_key8_2 = [cf_enc_X_train8[128],
                       cf_enc_X_train8[95],
                       cf_enc_X_train8[7],
                       cf_enc_X_train8[111],
                       cf_enc_X_train8[50]]


          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org8_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_key8_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("encrypted with key 8")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org8_2[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()
```

```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key8_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 8")
plt.tight_layout()
plt.show()
```



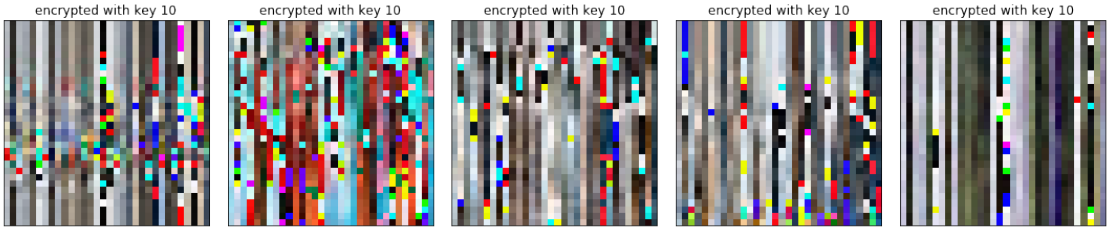
```
In [112]: index = [0,1,2,3,4]
          cf_org9_1 = [cifar10_X_train[30],
                       cifar10_X_train[5],
                       cifar10_X_train[18],
                       cifar10_X_train[36],
                       cifar10_X_train[10]]
          cf_org9_2 = [cifar10_X_train[128],
                       cifar10_X_train[95],
                       cifar10_X_train[7],
                       cifar10_X_train[111],
                       cifar10_X_train[50]]

          cf_key9_1 = [cf_enc_X_train9[30],
                       cf_enc_X_train9[5],
                       cf_enc_X_train9[18],
                       cf_enc_X_train9[36],
                       cf_enc_X_train9[10]]
          cf_key9_2 = [cf_enc_X_train9[128],
                       cf_enc_X_train9[95],
                       cf_enc_X_train9[7],
                       cf_enc_X_train9[111],
                       cf_enc_X_train9[50]]


          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org9_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_key9_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("encrypted with key 9")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org9_2[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()
```

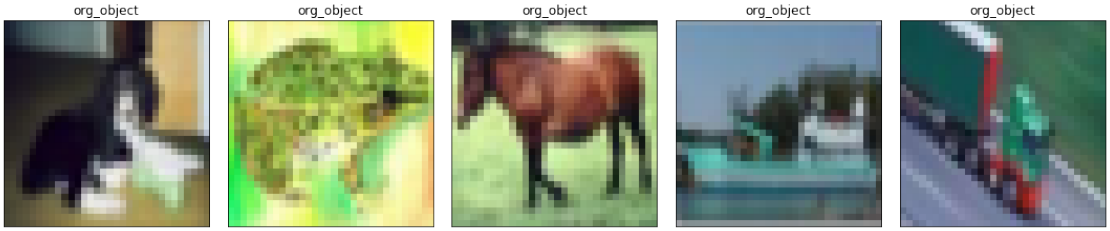
```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key9_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 9")
plt.tight_layout()
plt.show()
```

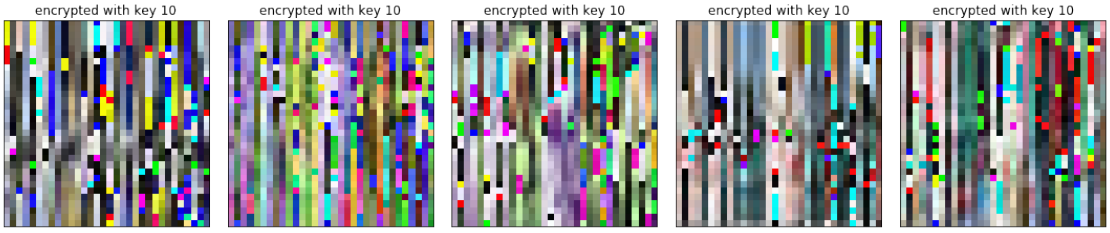


```
In [113]: index = [0,1,2,3,4]
          cf_org10_1 = [cifar10_X_train[30],
                        cifar10_X_train[5],
                        cifar10_X_train[18],
                        cifar10_X_train[36],
                        cifar10_X_train[10]]
          cf_org10_2 = [cifar10_X_train[128],
                        cifar10_X_train[95],
                        cifar10_X_train[7],
                        cifar10_X_train[111],
                        cifar10_X_train[50]]

          cf_key10_1 = [cf_enc_X_train10[30],
                        cf_enc_X_train10[5],
                        cf_enc_X_train10[18],
                        cf_enc_X_train10[36],
                        cf_enc_X_train10[10]]
          cf_key10_2 = [cf_enc_X_train10[128],
                        cf_enc_X_train10[95],
                        cf_enc_X_train10[7],
                        cf_enc_X_train10[111],
                        cf_enc_X_train10[50]]


          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org10_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_key10_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("encrypted with key 10")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org10_2[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()
```

```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_key10_2[i].astype('uint8'), cmap='viridis')
    ax.set_title("encrypted with key 10")
plt.tight_layout()
plt.show()
```



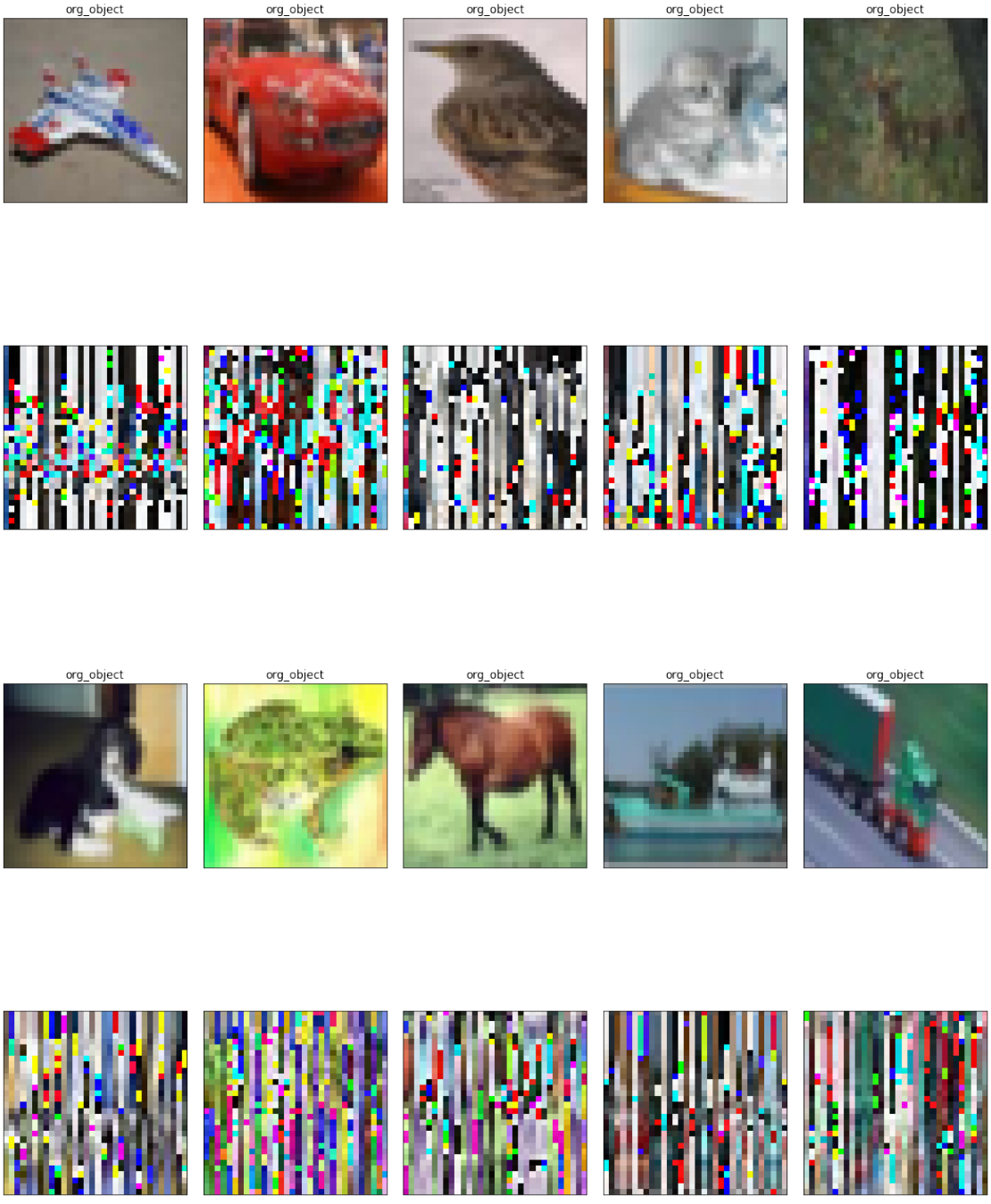




13 Different objects with different keys

```
In [118]: index = [0,1,2,3,4]
          cf_org1_1 = [cifar10_X_train[30],
                       cifar10_X_train[5],
                       cifar10_X_train[18],
                       cifar10_X_train[36],
                       cifar10_X_train[10]]
          cf_org1_2 = [cifar10_X_train[128],
                       cifar10_X_train[95],
                       cifar10_X_train[7],
                       cifar10_X_train[111],
                       cifar10_X_train[50]]

          cf_1_1 = [cf_enc_X_train1[30],
                       cf_enc_X_train2[5],
                       cf_enc_X_train3[18],
                       cf_enc_X_train4[36],
                       cf_enc_X_train5[10]]
          cf_1_2 = [cf_enc_X_train6[128],
                       cf_enc_X_train7[95],
                       cf_enc_X_train8[7],
                       cf_enc_X_train9[111],
                       cf_enc_X_train10[50]]

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org1_1[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_1_1[i].astype('uint8'), cmap='viridis')
          plt.tight_layout()
          plt.show()

          fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                                  subplot_kw={'xticks': [], 'yticks': []})
          for ax, i in zip(axs.flat, index):
              ax.imshow(cf_org1_2[i].astype('uint8'), cmap='viridis')
              ax.set_title("org_object")
          plt.tight_layout()
          plt.show()
```

```
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(15, 5),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, i in zip(axs.flat, index):
    ax.imshow(cf_1_2[i].astype('uint8'), cmap='viridis')
plt.tight_layout()
plt.show()
```



Appendix_D

December 8, 2020

1 Known plain text weakness!

Cipher = plain * key => need plain_reverse

```
In [6]: from sklearn.metrics import classification_report
        from keras.models import Sequential
        from keras.models import load_model
        from keras.layers import Dense, Conv2D, Flatten
        from keras.layers.core import Dropout
        from keras.layers.core import Activation
        from keras.optimizers import SGD
        from keras.datasets import mnist
        from keras.utils import to_categorical
        from keras.utils import np_utils
        from PIL import Image
        import matplotlib.pyplot as plt
        import numpy as np
        import pprint
        import scipy
        import scipy.linalg   # SciPy Linear Algebra Library
        import os
```

```
Using TensorFlow backend.
```

2 Find number of invertible images in MNIST data set

```
In [136]: #Load MNIST data
          (X_train, y_train), (X_test, y_test) = mnist.load_data()
          print("Done loading data!")
```

```
Done loading data!
```

```
In [ ]:
```

```
In [64]: #The rank of a matrix is the number of independent rows or columns of a matrix.
         def is_invertible(a):
             return a.shape[0] == a.shape[1] and np.linalg.matrix_rank(a) == a.shape[0]

In [65]: train_inv_img = []
         for an_image in X_train:
             if is_invertible(an_image):
                 train_inv_img.append(an_image)
         print("Done checking. Got",len(train_inv_img), "invertible images in train data set")
```

```
Done checking. Got 0 invertible images in train data set
```

```
In [66]: test_inv_img = []
         for an_image in X_test:
             if is_invertible(an_image):
                 test_inv_img.append(an_image)
         print("Done checking. Got",len(test_inv_img), "invertible images in test data set")
```

```
Done checking. Got 0 invertible images in test data set
```

3 Find number of invertible images in CIFAR10 data set

```
In [149]: # load CIFAR10 data
          from keras.datasets import cifar10
          (C10_X_train, C10_Y_train), (C10_X_test, C10_Y_test) = cifar10.load_data()
          print("Done loading CIFAR10 data!")
```

```
Done loading CIFAR10 data!
```

```
In [150]: # Check on train data set
          import cv2
          train_black_inv_index = []
          train_green_inv_index = []
          train_red_inv_index = []

          train_b = []
          train_g = []
          train_r = []

          total_img = len(C10_X_train)
          i = 0
          while i < len(C10_X_train):
              b, g, r = cv2.split(C10_X_train[i])
              #save all three unencrypted channels
              train_b.append(b)
```

```
            train_g.append(g)
            train_r.append(r)

            if is_invertible(b):
                train_black_inv_index.append(i)
            elif is_invertible(g):
                train_green_inv_index.append(i)
            elif is_invertible(r):
                train_red_inv_index.append(i)
            i +=1

        black_percent = len(train_black_inv_index) / total_img * 100
        green_percent = len(train_green_inv_index) / total_img * 100
        red_percent = len(train_red_inv_index) / total_img * 100

        print("Done checking ", total_img, "images in CIFAR10 train data set")

        print("Got ", len(train_black_inv_index), "invertible black channel matrixes.")
        print("This is", black_percent," percent")

        print("Got ",len(train_green_inv_index) , "invertible green channel matrixes.")
        print("This is", green_percent," percent")

        print("Got ",len(train_red_inv_index) , "invertible red channel matrixes.")
        print("This is", red_percent," percent")
```

```
Done checking  50000 images in CIFAR10 train data set
Got  49187 invertible black channel matrixes.
This is 98.374  percent
Got  74 invertible green channel matrixes.
This is 0.148  percent
Got  63 invertible red channel matrixes.
This is 0.126  percent
```

In [151]:
```
# Check on test data set
import cv2
test_black_inv_index = []
test_green_inv_index = []
test_red_inv_index = []

test_b = []
test_g = []
test_r = []

total_img = len(C10_X_test)
i = 0
while i < len(C10_X_test):
```

```
            b, g, r = cv2.split(C10_X_test[i])

            #save all three unencrypted channels
            test_b.append(b)
            test_g.append(g)
            test_r.append(r)

            if is_invertible(b):
                test_black_inv_index.append(i)
            elif is_invertible(g):
                test_green_inv_index.append(i)
            elif is_invertible(r):
                test_red_inv_index.append(i)
            i +=1

        black_percent = len(test_black_inv_index) / total_img * 100
        green_percent = len(test_green_inv_index) / total_img * 100
        red_percent = len(test_red_inv_index) / total_img * 100

        print("Done checking ", total_img, "images in CIFAR10 test data set")

        print("Got ", len(test_black_inv_index), "invertible black channel matrixes.")
        print("This is", black_percent," percent")

        print("Got ",len(test_green_inv_index) , "invertible green channel matrixes.")
        print("This is", green_percent," percent")

        print("Got ",len(test_red_inv_index) , "invertible red channel matrixes.")
        print("This is", red_percent," percent")
```

```
Done checking  10000 images in CIFAR10 test data set
Got  9830 invertible black channel matrixes.
This is 98.3  percent
Got  18 invertible green channel matrixes.
This is 0.18  percent
Got  13 invertible red channel matrixes.
This is 0.13  percent
```

In [175]:
```
#Drive orthogonal matrix
#Read key image as RGB
key = Image.open("/Users/thienngole/Desktop/MINES/3Fall2020/MS-project/key3.png")
#gray scale key image
key = key.convert('L')
#resize key image
key = np.resize(key,(32,32))
#convert to 2D array
key = np.asarray(key)
```

```
            #QR decomposition
            A = key
            Q, R = scipy.linalg.qr(A)
            dec_matrix = np.linalg.inv(Q)
```

In [176]:
```
            enc_X_train = []
            enc_train_b = []
            enc_train_g = []
            enc_train_r = []
            for an_image in C10_X_train:
                #slipt the three channels
                b, g, r = cv2.split(an_image)
                # encrypt back channel
                enc_b = np.dot(b, Q)
                enc_train_b.append(enc_b)
                # encrypt green channel
                enc_g = np.dot(g, Q)
                enc_train_g.append(enc_g)
                # encrypt red channel
                enc_r = np.dot(r, Q)
                enc_train_r.append(enc_r)
                #merge al three channels back together
                enc_rgb = cv2.merge((enc_b,enc_g,enc_r))
                enc_X_train.append(enc_rgb)
            print("Done encrypting ",len(C10_X_train),"train images!")
```

```
Done encrypting  50000 train images!
```

In [177]:
```
train_green_inv_index[60]
```

Out[177]:
```
42374
```

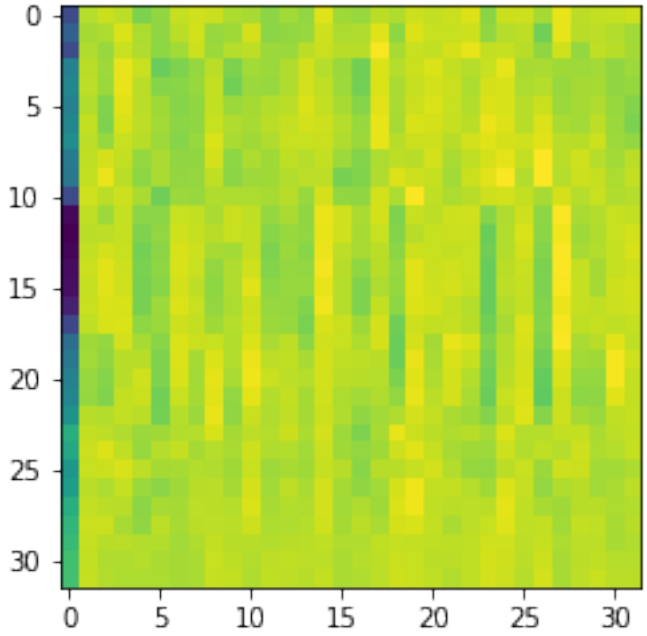
In [193]:
```
train_black_inv_index[60]
```

Out[193]:
```
60
```

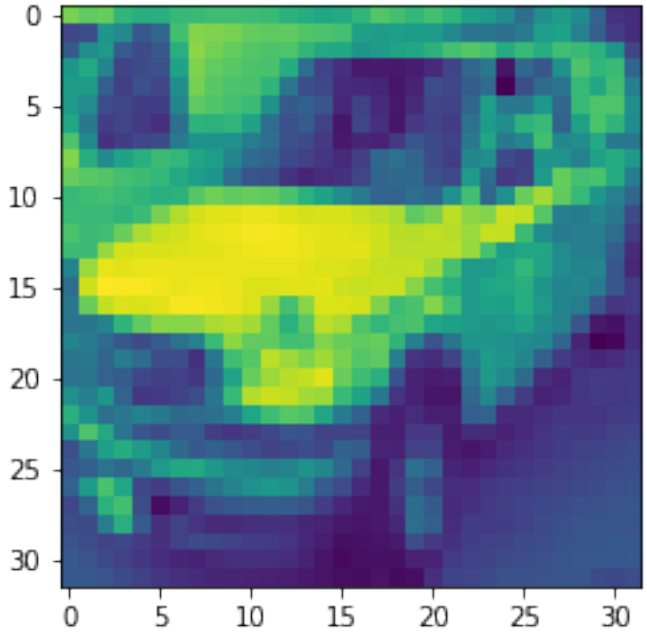
In [194]:
```
enc = enc_train_b[60]
```

In [195]:
```
plt.imshow(enc)
```

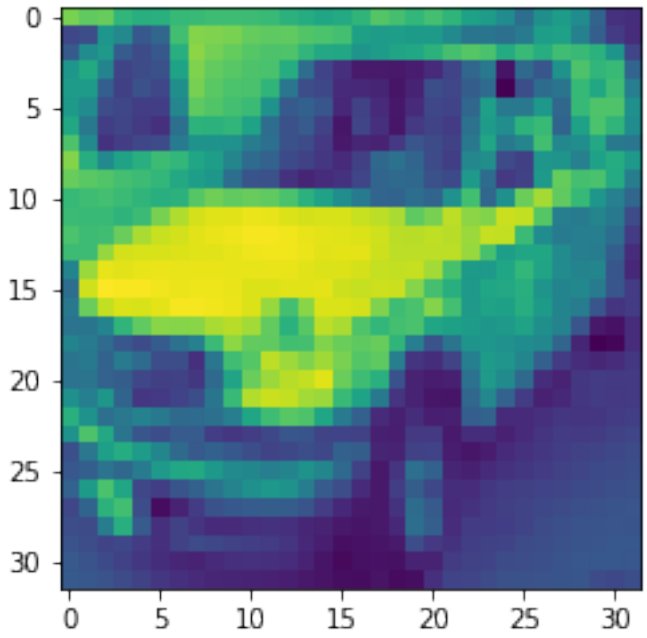
Out[195]:
```
<matplotlib.image.AxesImage at 0x15dc57898>
```



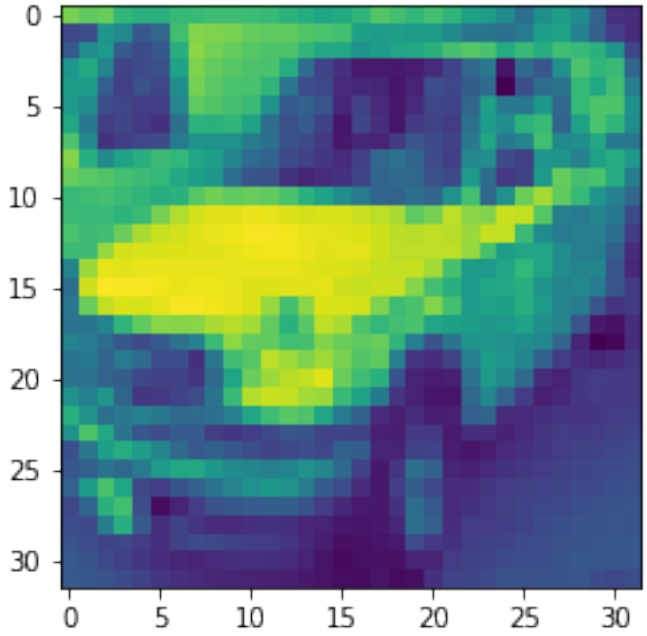

```
In [196]: org = train_b[60]

In [197]: plt.imshow(org)

Out[197]: <matplotlib.image.AxesImage at 0x15daa7e80>
```



```
In [198]: inv = np.linalg.inv(org)

In [199]: enc_key_found = np.dot(enc, inv)

In [200]: dec_key_found = np.linalg.inv(enc_key_found)

In [201]: dec = np.dot(enc, dec_key_found)

In [202]: org_dec = np.dot(enc, dec_matrix)

In [203]: plt.imshow(org)

Out[203]: <matplotlib.image.AxesImage at 0x15dd6c5f8>
```

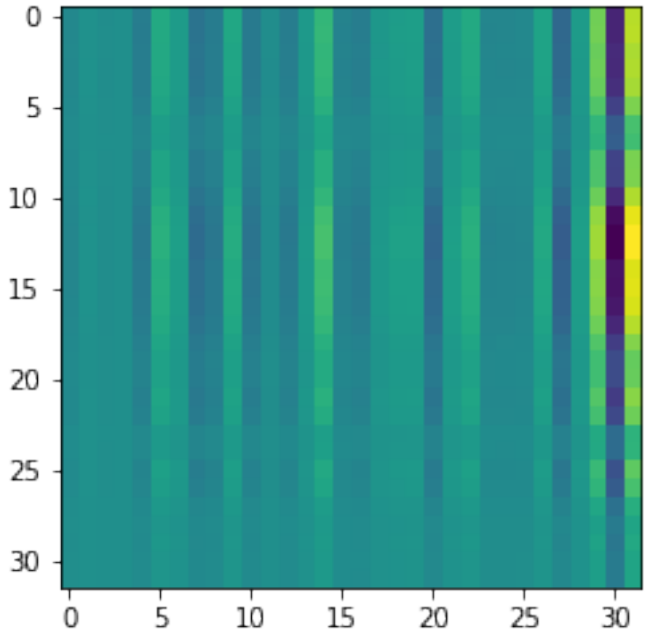



```
In [204]: plt.imshow(org_dec)

Out[204]: <matplotlib.image.AxesImage at 0x15defeb00>
```



In [205]: plt.imshow(dec)

Out[205]: <matplotlib.image.AxesImage at 0x15e05b0f0>



In [ ]: