# Math 110B Final Project Report

Yaoqi Li, Yuxuan Wang, Gufeng Yang, Zarin Tasnim Ohiba

## Introduction

Steganography is the practice of concealing images. In this project, we implement LSB, CNN and CNN with DCT to compare the performances of these three algorithms. In every image, pixels are the smallest individual element with each image contains 256x256 pixels. Basically, in color imaging systems, a color is typically represented by three or four component intensities such as red, green and blue. For jpg file type, each pixel from the image is composed of 3 values (red, green, blue) which are 8-bit values in three dimensions. For PNG file type, each pixel from the image is composed of 4 values (R, G, B, A) which in four dimensions. We will be based on the loss value and intuitively feelings to compare the 24 test images. After using those three methods to unmerge the images, we want to see how much the information for both images has left and our goal is to test which method will as much as possible to recover both images.

## Data set

**Training data:** Our training data for the CNN model is by randomly choosing 404 images out of more than 2000 images from the set on Kaggle, with size 256x256. To train the model, we used those 404 images to run 100 times.

**Test data:** the test data is from the website http://r0k.us/graphics/kodak/ which contains 24 test images with any size.

## Methods

- **LSB**

In LSB algorithm, since the left most bit have higher values then we only need to consider the most important bits which are represented in the first 4th place (leftmost) and the rightmost bits are less significant. Thus, we will delete them when merging two images. After extracting leftmost bits of secrete image and cover, we combine them into one 8bits value. First four bits

are from cover image and last four are from secrete images. By combining those four digits, we merge the secrete images into the cover image. Then, when we unmerge the image, we also extract the leftmost bits and then add “0000” to create both a new cover image and a secrete image. The performance is not bad since we indeed successfully hide the secrete image under the cover image. Also, the unmerged images still preserve the most important features. However, if you look at those images more carefully, you still can find some important features of the secret image in the cover image. After we unmerge them, some color of the secret image changed. Therefore, we will apply another method, convolutional neural network with multiple convolution layers, to let the machine ‘learn’ how to hide the secret under the cover and unmerge them with more important features preserved.

- **Convolutional Neural Network**

In CNN algorithm, there are three networks, prepare network, hiding network and reveal network combined by forward function. In our project, we only consider secrete image and cover image in same size, either 256x256 or 64x64. Prep-Network has kernel size equal to 3, 4 and 5 respectively. The input image size is a three dimensional and the output is a five dimension each represented a channel. The final layer has input of 150 channels and output 50 channels. The prepare network prepare the secret image to be hidden and we want the secrete image becomes as compressed as possible.

The second and also the main network, the Hiding Network, takes as input the output of the preparation-network and the cover image, and creates the Container image. The input to this network is a N × N pixel field, with depth concatenated RGB channels of the cover image and the transformed channels of the secret image. According to the paper, the best hidden layers and convolution sizes consisted of 5 convolution layers that had 50 filters each of {3 × 3, 4 × 4, 5 × 5} patches.

The Reveal Network, also the unmerge part, receives only the Container image (not the cover nor secret image). The decoder network removes the cover image to reveal the secret image.

This is our loss function. We want to reduce the loss

$$\mathcal{L}(c, c', s, s') = ||\mathbf{c} - \mathbf{c}'|| + \beta||\mathbf{s} - \mathbf{s}'||$$

C: cover image c’:merge image

S: secrete image s':unmerge image

a. Error term:||c – c'|| does not apply to the weights of the reveal-network that receives the container image and extracts the secret image. It only affects the first two networks.
b. all of the networks receive the error signal β*||s – s'|| for reconstructing the hidden image.

When programming a CNN, each convolutional layer within a neural network should have the following attributes:

- Input is a tensor with shape (number of images) x (image width) x (image height) x (image depth).
- Convolutional kernels whose width and height are hyper-parameters, and whose depth must be equal to that of the image. Convolutional layers convolve the input and pass its result to the next layer. This is like the response of a neuron in the visual cortex to a specific stimulus

In this project: there are 5 layers for the convolutional network.

Here is our example of one network, the prepare network. We can treat the tensor object as a nd-array with three 2-d array. We separate them into initialP3, initialP4 and initialP5. We used 3x3, 4x4 and 5x5 as our filter size for each of the first four layers. Finally, we concatenated them and passed them to the final layer.

```
class PrepNetwork(nn.Module):
    def __init__(self):
        super(PrepNetwork, self).__init__()
        self.initialP3 = nn.Sequential(
            nn.Conv2d(3, 50, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=3, padding=1),
            nn.ReLU())
        self.initialP4 = nn.Sequential(
            nn.Conv2d(3, 50, kernel_size=4, padding=1),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=4, padding=2),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=4, padding=1),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=4, padding=2),
            nn.ReLU())
        self.initialP5 = nn.Sequential(
            nn.Conv2d(3, 50, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=5, padding=2),
            nn.ReLU())
        self.finalP3 = nn.Sequential(
            nn.Conv2d(150, 50, kernel_size=3, padding=1),
            nn.ReLU())
        self.finalP4 = nn.Sequential(
            nn.Conv2d(150, 50, kernel_size=4, padding=1),
            nn.ReLU(),
            nn.Conv2d(50, 50, kernel_size=4, padding=2),
            nn.ReLU())
        self.finalP5 = nn.Sequential(
            nn.Conv2d(150, 50, kernel_size=5, padding=2),
            nn.ReLU())
```

The hiding network hides the secret image in the PrepNetwork's output and returns the hidden image. This is fed into the Reveal network to output the message, that should be close to the secret image.

Here is the code that we build three networks together as the full model.

```
Join three networks in one module   we can get a net object and train our model and use this model to solve other questions.
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.m1 = PrepNetwork()
        self.m2 = HidingNetwork()
        self.m3 = RevealNetwork()

    def forward(self, secret, cover):
        x_1 = self.m1(secret)
        mid = torch.cat((x_1, cover), 1)
        x_2, x_2_noise = self.m2(mid)
        x_3 = self.m3(x_2_noise)
        return x_2, x_3
```

- **Discrete Cosine Transform**

Our DCT algorithm is to reduce the information first on secret images. It is similar to the CNN but it only has two networks: we bypass the preparing network and direct use the hiding and reveal network as the preparation process. Our goal is to use DCT to compress the secrete

images. The formula of DCT is as following:

**DCT**

DCT uses only cosines

$$X_k = \sum_{n=0}^{N-1} x_n cos[\frac{\pi}{N}(n + \frac{1}{2})k]$$

where

$$k = 0, \dots, N-1$$

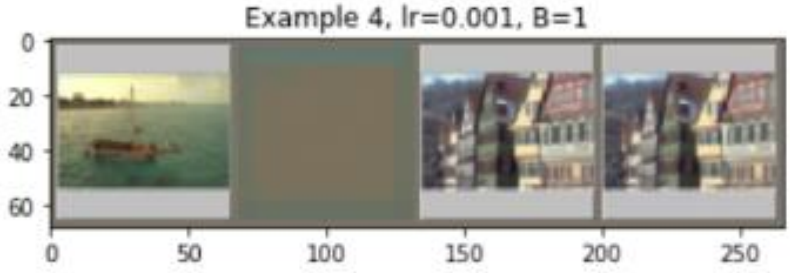
The reason that DCT is so works better is that the first 40-50 coefficients out of 256 could capture most of the image (there are some noticeable artifacts around edges). If we want to use these coefficients to reconstruct the image, we could get high compression where we do not lose much image quality and image information in the process. Most of the compression method generally use this type of schema to get a compression rate. They discard some of the high frequency information and extract only most important ones as representatives (edges and fast transition images where they preserve the slow changes and constant colors). Since most of the signals are low frequency by nature, this approach works quite good in practice. We remove the high frequency coefficients and transmit the "important" coefficients (the coefficients that have most information about the signal).

## Results and conclusion

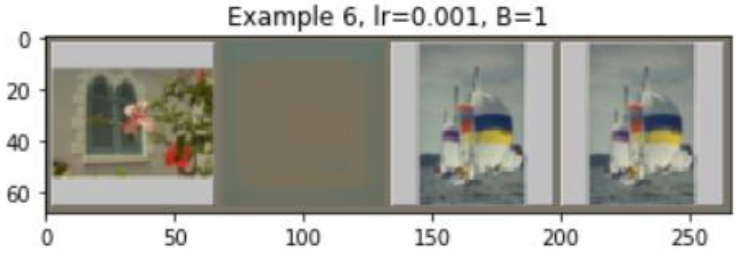
- **Results**

1. Three examples of the result of the CNN on the 24 test images:
   In an order from left to right: the secret image, the secret image after compressed, the cover image and the hidden image (merged)
   The average loss is 1.12.

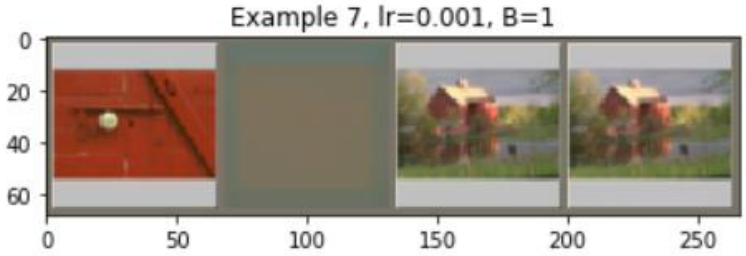
```
Total loss: 1.35
Loss on secret: 1.35
Loss on cover: 0.00
```




```
Total loss: 0.93
Loss on secret: 0.93
Loss on cover: 0.00
```




```
Total loss: 1.28
Loss on secret: 1.27
Loss on cover: 0.00
```

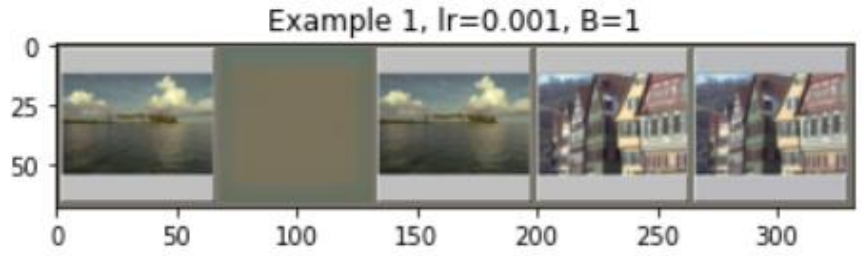



2. Three examples of the result of the DCT on the 24 test images.

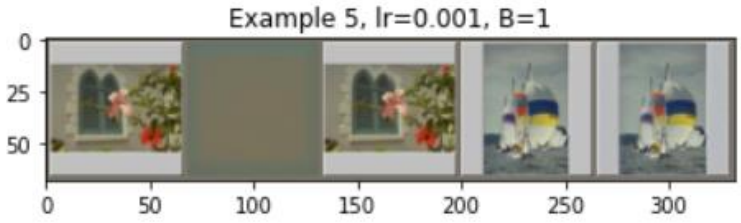
   In an order from left to right: the secret image, the secret image after compressed, the secret image by inversing the DCT to observe DCT works well on compressing the image, the cover image and the hidden image (merged)

   The average loss is 1.27.

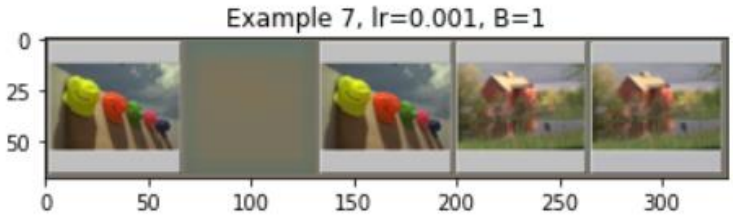
```
Total loss: 1.05
Loss on secret: 1.05
Loss on cover: 0.01
```




```
Total loss: 0.95
Loss on secret: 0.95
Loss on cover: 0.01
```




```
Total loss: 1.09
Loss on secret: 1.09
Loss on cover: 0.00
```




3. Three examples of the result of the LSB on the 24 test images.

   In an order from left to right: the secret image, the cover image and the hidden image (merged)

   a.

b.

c.

- **Conclusion**

By comparing the images, quantitively and qualitatively, we can observe that the CNN and DCT is better than the LSB and. Since the image pixel is low, we can only tell that the CNN and the DCT is similar on the performance of the merged image. The LSB's merged image can tell there is something wrong or different. Since for example 2 and 3, we can easily tell the sky is different and the sky color is been twisted. We conject that if the cover image has a large area that has the similar color. It will be affected by the secret image by changing some color of that area, which is the reason will use CNN and DCT.

The performance of the CNN and DCT is similar on the merged image since the image is not HD thus, we can only tell by intuitively feelings. However, based on the average loss we can see that the CNN is performing a little bit better than the DCT. The merged image by using two methods are don't have much different. However, the running time of DCT is faster the CNN since it is by compressing the secret images. Our testing image is small which we can't observe that the DCT is much faster than CNN. Thus, in the future, we would better test on large pixel images to observe the difference of the running time.