# Final Project Report

## How to run the code:

Please follow the structure as mentioned below. Put the Final-Project.ipynb and picture under the same directory. I did not submit pictures to GitHub. Please Put the pictures inside present/1, test/1, and training/1 directories. My code will be trained on these datasets in training/1 and show the result in the directory of present/1. If you do not want to run the training part, you can load Epoch N3.pkl.

The correct structure:

- Picture
  - coast
    - model
      - Epoch N1.pkl
      - Epoch N2.pkl
      - Epoch N3.pkl
    - present
      - 1
        - insert the pictures you want to present the final result
    - test
      - 1
        - insert the pictures you want to test, remember to change the test route
    - training
      - 1
        - insert the pictures you want to train

## First Task

### Code reference:

https://towardsdatascience.com/steganography-hiding-an-image-inside-another-77ca66b2acb1

## Result

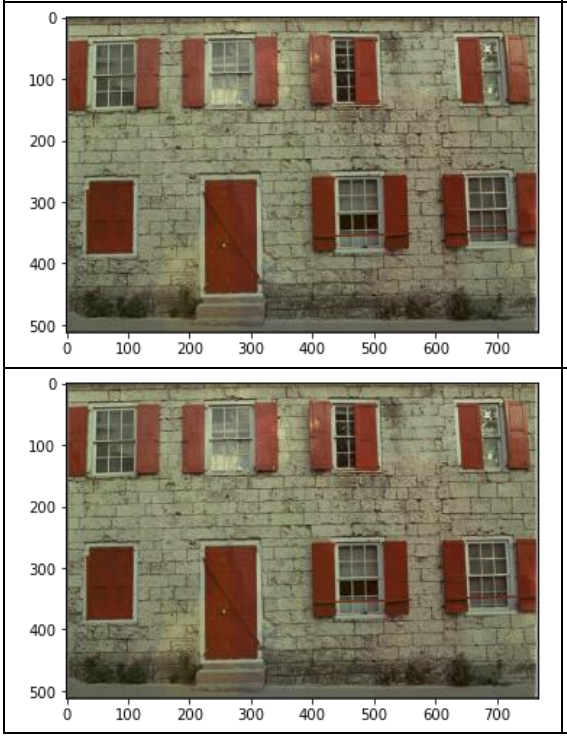
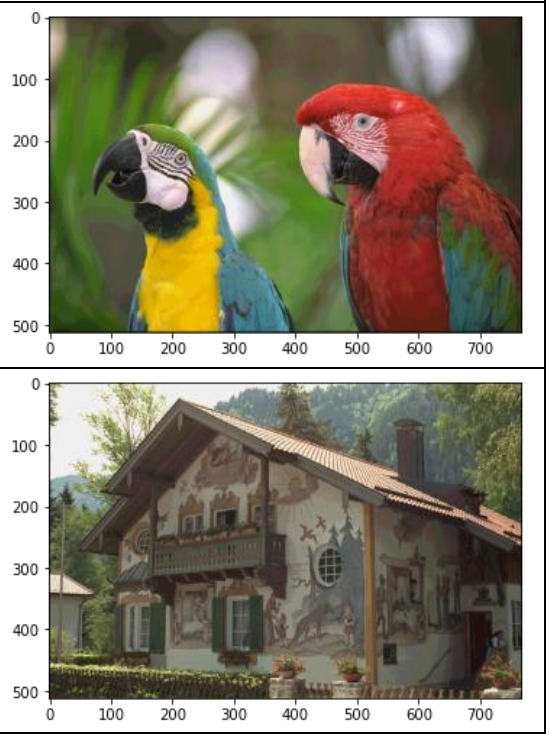
### (1) Without using tensor, calculate with int8

```
RMSE for encodeImage: 10.234880398004146
RMSE for decodeImage: 15.573037422636501
```

### (2) Using tensor

```
RMSE for encodeImage: 0.023172983161979164
RMSE for decodeImage: 0.035259203533044826
```

## Example

| Encoded Picture | Decoded Picture |
| --- | --- |
|  |  |
|  |  |

## Problem Solved

(1) I found that if we directly use the tuple of int8 to calculate the RMSE, The RMSE will become pretty big. So I need to change PIL.image to tensor using the following code

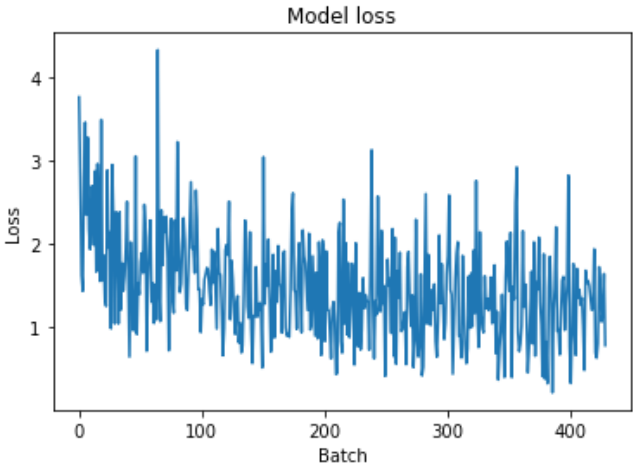
```
img1 = transforms.ToTensor()(img1).unsqueeze_(0)
```

Second Task

## Code reference:

https://github.com/fpingham/DeepSteg/blob/master/DeepSteganography.ipynb

## Result



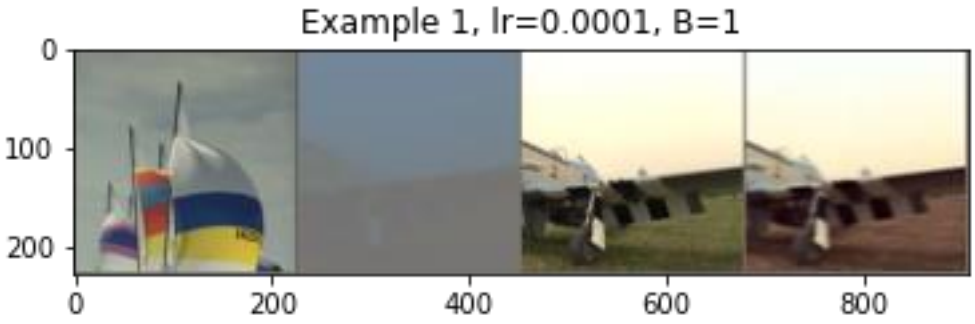

(1) Training loss

Figure 1 Training loss

(2) RMSE for test data

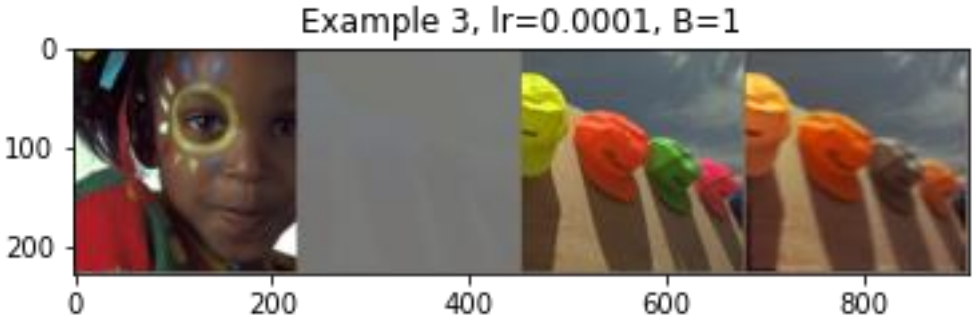
```
Test size: 20
Average loss on test set: 1.17
Average loss on secret: 0.91
Average loss on cover:0.26
```

## Examples




```
Total loss: 0.83
Loss on secret: 0.62
Loss on cover: 0.22
```


```
Total loss: 1.13
Loss on secret: 0.89
Loss on cover: 0.24
```




```
Total loss: 1.27
Loss on secret: 1.09
Loss on cover: 0.18
```

## Problem Solved

(1) The kernel in the anaconda kept dying when I wanted to train the CNN or get the test result. I add the following line of the code at the beginning to prevent such a situation:

```
os.environ['KMP_DUPLICATE_LIB_OK'] = 'TRUE'
```

(2) If I only create the directory of training, present, validation, test, the code reference will become an error. I must include a directory named one inside these directories. On stack overflow, it says this error happens because there are too many files in one directory. I put the path as follows:

```
cwd = r"picture/coast"
VALID_PATH = cwd+ r"/validation/"
TRAIN_PATH = cwd+ r"/training/"
TEST_PATH = cwd+ r"/present/"
```

(3) The code I reference is calculating MSE rather than RMSE, so I fix the problem by changing the code to:

```
loss_cover = math.sqrt(torch.nn.functional.mse_loss(C_prime, C))
loss_secret = math.sqrt(torch.nn.functional.mse_loss(S_prime, S))
loss_all = loss_cover + B * loss_secret
return loss_all, loss_cover, loss_secret
```

## Problem NOT Solved

(1) Warning of the following appears in my notebook:

```
<ipython-input-122-91837e67e0c5>:28: UserWarning: nn.init.normal is now deprecated in favor
 of nn.init.normal_.
  noise = torch.nn.init.normal(torch.Tensor(tensor.size()), 0, 0.1)
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
```

And I do not know how to solve it.

# Optional Task

## LSB

It could work, but the performance is not that good. To encrypt two pictures using LSB, the best option is to use an int-16 image to encrypt two int-8 images. 8 (cover image) 4 (Secret Image 1) 4 (Secret Image 2). If we use an int-8 image to encrypt two int-8 images. I would recommend 2(cover image), 3(Secret Image 1) 3(Secret Image 2)

## CNN

To Encrypt two pictures inside one picture using CNN, we first apply the code we wrote to encode Secret Image 1, used as a cover picture, and Secret Image 2, used as the secret picture. This will generate the first CNN, and then we try to reverse the process. This will cause the second CNN after we have got the encoded image covered in Secret Image 1. We use this image as the Secret Image and the current cover image as Cover Image to generate the final image. This will produce two CNN like before. Since we have got 4 CNN models, we can use these 4 CNN models to encode two pictures inside one picture

## Conclusion:

LSB's secret or cover loss is much lower than CNNs. However, they are pretty easy to decode. CNN is much harder to decipher unless users have the .pkl file the CNNs have produced.