Final project: Steganography
Group member: Qingcheng Zhang
Task1: In this task, we use the code which given on github:
https://github.com/kelvins/steganography

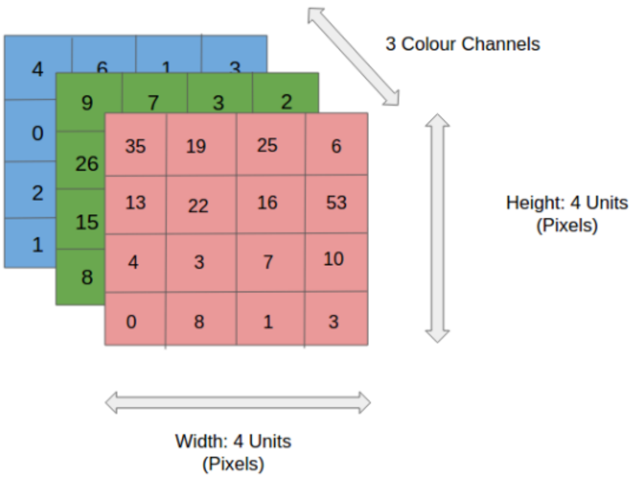
In this task, we understand how to use python to implement LSB(LSB is called least significant bits, which means you will replace the least significant bits of the original image with the secret image's most significant bits), and make the images (original and secret) are of the same sizes. First of all, we understand what the pixel(the smallest element is an image) is and We split the image into three channels: red, green, and blue. So, each pixel has three channels, and each channel is 8-bits sequence. For 8-bits calculation, the last 4 digits do not influence too much.

	128	64	32	16	8	4	2	1
8 bit binary digit	1	0	1	1	0	0	0	1
	128 + 32 + 16 + 1 = 177							

So, we change the original image’s last four digits into the hidden image’s four digits to complete the requirements of steganography.

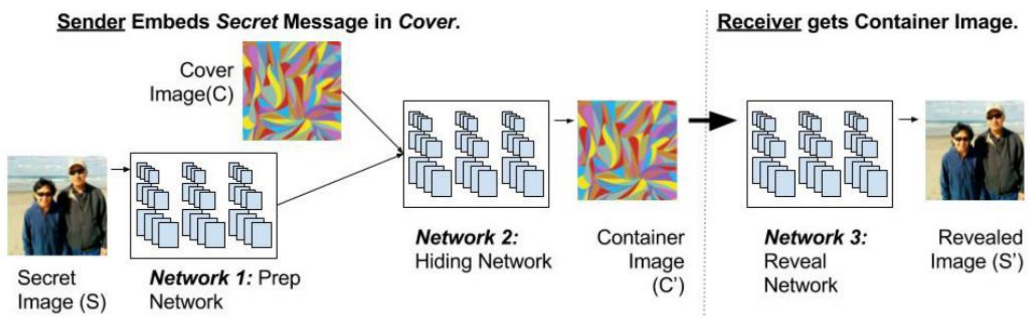
Task2: In this task, we use the code which given on github:
https://github.com/fpingham/DeepSteg/blob/master/DeepSteganography.ipynb

There are 5 layers in the method of CNN, the Convolutional Layer, Pooling Layer, Normalization Layer, Fully-Connected Layer and Converting Fully-Connected Layers to Convolutional Layer.
In this case l, we take an RGB image which has been separated by its three color planes—Red, Green, and Blue, as an input.



The goal of the CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.
The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.
Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. This is useful for extracting dominant features which are rotational and positional invariant, while maintaining the process of effectively training of the model.
The Fully Connected layer is learning a possibly non-linear function in that space.

The CNN on our final project process is in the figure below



For this code, we use GPU computing. It is significantly faster than CPU. If we use CPU, 5 data set only have 870 batch, it will use 18 hours to finish running (only for training part). But if we use GPU, it will shorten time 30 times, it will only take 30 mins for 5 data sets.

Because of the big data set, we want to minimize the data set, we want to minimize the data set. But the influence is very small. Total number of iterations of larger dataset (batch size is 2) does not have big difference with smaller dataset. This is because the smaller dataset provides enough iteration for training.

The following sub section is talking about the performance of different types of parameter and models during this final project.

Number of epochs:

If only have one epoch, the loss will be big. But if we have 3 or 5 or bigger epochs, the performances are relatively same. Even though we run 50 epochs, loss is the relatively same as 3 epochs' one.

Epoch = 1, testing lass 1.4, training loss is 1.4399(graph below)

```
Training: Batch 860/870. Loss of 1.0608, cover loss of 0.0711, secret loss of 0.9898
Training: Batch 861/870. Loss of 0.7820, cover loss of 0.0724, secret loss of 0.7096
Training: Batch 862/870. Loss of 0.8232, cover loss of 0.0204, secret loss of 0.8028
Training: Batch 863/870. Loss of 1.0961, cover loss of 0.0161, secret loss of 1.0801
Training: Batch 864/870. Loss of 1.0948, cover loss of 0.0380, secret loss of 1.0568
Training: Batch 865/870. Loss of 1.3199, cover loss of 0.0458, secret loss of 1.2742
Training: Batch 866/870. Loss of 1.3823, cover loss of 0.0247, secret loss of 1.3576
Training: Batch 867/870. Loss of 2.3242, cover loss of 0.0226, secret loss of 2.3016
Training: Batch 868/870. Loss of 0.7793, cover loss of 0.0290, secret loss of 0.7503
Training: Batch 869/870. Loss of 1.4547, cover loss of 0.0289, secret loss of 1.4257
Training: Batch 870/870. Loss of 1.0139, cover loss of 0.0132, secret loss of 1.0007
Epoch [1/1], Average_loss: 1.4399
f:\MATH 110A\FinalProject.py:392: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_secret = Variable(test_secret, volatile=True)
f:\MATH 110A\FinalProject.py:394: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_cover = Variable(test_cover, volatile=True)
Total loss: 1.29
Loss on secret: 1.13
Loss on cover: 0.16
Total loss: 1.67
Loss on secret: 1.59
Loss on cover: 0.08
Total loss: 2.36
Loss on secret: 2.33
Loss on cover: 0.03
Total loss: 1.38
Loss on secret: 1.35
Loss on cover: 0.03
Average loss on test set: 1.40
```

Epoch =3, testing loss 1.29, training loss is 1.31(graph below)

```
Training: Batch 850/870. Loss of 0.6433, cover loss of 0.0043, secret loss of 0.6390
Training: Batch 851/870. Loss of 1.4826, cover loss of 0.0031, secret loss of 1.4795
Training: Batch 852/870. Loss of 1.1072, cover loss of 0.0042, secret loss of 1.1030
Training: Batch 853/870. Loss of 0.6779, cover loss of 0.0040, secret loss of 0.6739
Training: Batch 854/870. Loss of 2.2714, cover loss of 0.0042, secret loss of 2.2671
Training: Batch 855/870. Loss of 1.2985, cover loss of 0.0084, secret loss of 1.2901
Training: Batch 856/870. Loss of 1.1800, cover loss of 0.0127, secret loss of 1.1672
Training: Batch 857/870. Loss of 1.3025, cover loss of 0.0039, secret loss of 1.2986
Training: Batch 858/870. Loss of 1.6549, cover loss of 0.0039, secret loss of 1.6510
Training: Batch 859/870. Loss of 1.0353, cover loss of 0.0041, secret loss of 1.0312
Training: Batch 860/870. Loss of 1.0060, cover loss of 0.0023, secret loss of 1.0037
Training: Batch 861/870. Loss of 1.0824, cover loss of 0.0017, secret loss of 1.0808
Training: Batch 862/870. Loss of 0.6250, cover loss of 0.0026, secret loss of 0.6223
Training: Batch 863/870. Loss of 1.0465, cover loss of 0.0020, secret loss of 1.0445
Training: Batch 864/870. Loss of 1.4767, cover loss of 0.0172, secret loss of 1.4595
Training: Batch 865/870. Loss of 1.2822, cover loss of 0.0045, secret loss of 1.2778
Training: Batch 866/870. Loss of 2.0761, cover loss of 0.0026, secret loss of 2.0734
Training: Batch 867/870. Loss of 1.3348, cover loss of 0.0035, secret loss of 1.3313
Training: Batch 868/870. Loss of 0.7506, cover loss of 0.0079, secret loss of 0.7426
Training: Batch 869/870. Loss of 1.5056, cover loss of 0.0041, secret loss of 1.5015
Training: Batch 870/870. Loss of 0.8331, cover loss of 0.0031, secret loss of 0.8301
Epoch [3/3], Average_loss: 1.3146
f:\MATH 110A\FinalProject.py:392: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_secret = Variable(test_secret, volatile=True)
f:\MATH 110A\FinalProject.py:394: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_cover = Variable(test_cover, volatile=True)
Total loss: 1.69
Loss on secret: 1.67
Loss on cover: 0.01
Total loss: 1.68
Loss on secret: 1.68
Loss on cover: 0.00
Total loss: 1.28
Loss on secret: 1.26
Loss on cover: 0.02
Total loss: 1.79
Loss on secret: 1.78
Loss on cover: 0.00
Average loss on test set: 1.29
```

Even after epoch = 10, the training loss is the same, and the testing loss is slightly better.

Learning rate:

Learning rate with 0.0001 and 0.000146 have the same result. 0.0005's loss is slightly worse than the previous two choices. These 3 learning rates are the popular choices of CNN.

Beta:

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

For the beta parameter, the smaller the beta, the loss is smaller by the loss function above. For example, when beat = 0.75, the loss on training is relatively 1, the testing loss is 0.92.

Activation Function:

Since sigmoid and tanh are not popular choice currently, so we tried activated function related with ReLU. But dying ReLU problem is a problem of ReLU.

"Unfortunately, ReLU units can be fragile during training and can 'die'. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as 40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset)

So we also choose leaky ReLU and Randomized ReLU to fix the "dying ReLU" problem. Instead of the function being zero when $x < 0$, a leaky ReLU will instead have a small negative slope (of 0.01, or so). Randomize ReLU (Randomized ReLU (RReLU in pytorch) is a randomized version of Leaky ReLU, where the α is a random number. In Randomized ReLU, the slopes of negative parts are randomized in a given range in the training, and then fixed in the testing. It is reported that Randomized ReLU could reduce overfitting due to its randomized nature

The following is my experiment:
For the author's code(https://github.com/fpingham/DeepSteg/blob/master/DeepSteganography.ipynb), the test loss is 1.37, and the training loss is 1.33.

For the Leaky ReLU,
Testing loss is 1.29, training loss is 1.31

```
Training: Batch 850/870. Loss of 0.6433, cover loss of 0.0043, secret loss of 0.6390
Training: Batch 851/870. Loss of 1.4826, cover loss of 0.0031, secret loss of 1.4795
Training: Batch 852/870. Loss of 1.1072, cover loss of 0.0042, secret loss of 1.1030
Training: Batch 853/870. Loss of 0.6779, cover loss of 0.0040, secret loss of 0.6739
Training: Batch 854/870. Loss of 2.2714, cover loss of 0.0042, secret loss of 2.2671
Training: Batch 855/870. Loss of 1.2985, cover loss of 0.0084, secret loss of 1.2901
Training: Batch 856/870. Loss of 1.1800, cover loss of 0.0127, secret loss of 1.1672
Training: Batch 857/870. Loss of 1.3025, cover loss of 0.0039, secret loss of 1.2986
Training: Batch 858/870. Loss of 1.6549, cover loss of 0.0039, secret loss of 1.6510
Training: Batch 859/870. Loss of 1.0353, cover loss of 0.0041, secret loss of 1.0312
Training: Batch 860/870. Loss of 1.0060, cover loss of 0.0023, secret loss of 1.0037
Training: Batch 861/870. Loss of 1.0824, cover loss of 0.0017, secret loss of 1.0808
Training: Batch 862/870. Loss of 0.6250, cover loss of 0.0026, secret loss of 0.6223
Training: Batch 863/870. Loss of 1.0465, cover loss of 0.0020, secret loss of 1.0445
Training: Batch 864/870. Loss of 1.4767, cover loss of 0.0172, secret loss of 1.4595
Training: Batch 865/870. Loss of 1.2822, cover loss of 0.0045, secret loss of 1.2778
Training: Batch 866/870. Loss of 2.0761, cover loss of 0.0026, secret loss of 2.0734
Training: Batch 867/870. Loss of 1.3348, cover loss of 0.0035, secret loss of 1.3313
Training: Batch 868/870. Loss of 0.7506, cover loss of 0.0079, secret loss of 0.7426
Training: Batch 869/870. Loss of 1.5056, cover loss of 0.0041, secret loss of 1.5015
Training: Batch 870/870. Loss of 0.8331, cover loss of 0.0031, secret loss of 0.8301
Epoch [3/3], Average_loss: 1.3146
f:\MATH 110A\FinalProject.py:392: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_secret = Variable(test_secret, volatile=True)
f:\MATH 110A\FinalProject.py:394: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_cover = Variable(test_cover, volatile=True)
Total loss: 1.69
Loss on secret: 1.67
Loss on cover: 0.01
Total loss: 1.68
Loss on secret: 1.68
Loss on cover: 0.00
Total loss: 1.28
Loss on secret: 1.26
Loss on cover: 0.02
Total loss: 1.79
Loss on secret: 1.78
Loss on cover: 0.00
Average loss on test set: 1.29
```

In our code, I also combine the use of ReLU and Leaky ReLU(The preparation layer use ReLU, and the rest of them use LeakyReLU)
Testing loss is 1.22, training loss is 1.31

```
Training: Batch 331/341. Loss of 2.1895, cover loss of 0.0113, secret loss of 2.1782
Training: Batch 332/341. Loss of 1.2069, cover loss of 0.0111, secret loss of 1.1958
Training: Batch 333/341. Loss of 1.0616, cover loss of 0.0127, secret loss of 1.0489
Training: Batch 334/341. Loss of 1.9334, cover loss of 0.0167, secret loss of 1.9168
Training: Batch 335/341. Loss of 1.4376, cover loss of 0.0048, secret loss of 1.4327
Training: Batch 336/341. Loss of 1.3655, cover loss of 0.0169, secret loss of 1.3487
Training: Batch 337/341. Loss of 1.3151, cover loss of 0.0121, secret loss of 1.3030
Training: Batch 338/341. Loss of 1.1835, cover loss of 0.0038, secret loss of 1.1797
Training: Batch 339/341. Loss of 0.8800, cover loss of 0.0038, secret loss of 0.8762
Training: Batch 340/341. Loss of 0.7741, cover loss of 0.0125, secret loss of 0.7617
Training: Batch 341/341. Loss of 1.1066, cover loss of 0.0081, secret loss of 1.0985
Epoch [5/5], Average_loss: 1.3158
f:\MATH 110A\FinalProject.py:392: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_secret = Variable(test_secret, volatile=True)
f:\MATH 110A\FinalProject.py:394: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_cover = Variable(test_cover, volatile=True)
Total loss: 0.98
Loss on secret: 0.97
Loss on cover: 0.01
Total loss: 1.58
Loss on secret: 1.57
Loss on cover: 0.01
Total loss: 1.57
Loss on secret: 1.57
Loss on cover: 0.00
Total loss: 1.63
Loss on secret: 1.63
Loss on cover: 0.00
Average loss on test set: 1.22
```

Training loss is the same, test loss is better than relu. The loss is reduced by 6 to 8 percent.

Optimizer:

SGD: By using it, the loss is large

```
Training: Batch 850/870. Loss of 2.5365, cover loss of 0.8002, secret loss of 1.7363
Training: Batch 851/870. Loss of 2.5384, cover loss of 1.2915, secret loss of 1.2470
Training: Batch 852/870. Loss of 3.1915, cover loss of 1.3301, secret loss of 1.8614
Training: Batch 853/870. Loss of 3.6905, cover loss of 1.8904, secret loss of 1.8001
Training: Batch 854/870. Loss of 1.6070, cover loss of 0.6327, secret loss of 0.9744
Training: Batch 855/870. Loss of 2.9013, cover loss of 1.1292, secret loss of 1.7721
Training: Batch 856/870. Loss of 2.7744, cover loss of 1.0359, secret loss of 1.7385
Training: Batch 857/870. Loss of 1.8877, cover loss of 0.7872, secret loss of 1.1005
Training: Batch 858/870. Loss of 1.9358, cover loss of 0.7140, secret loss of 1.2217
Training: Batch 859/870. Loss of 2.8321, cover loss of 1.0453, secret loss of 1.7869
Training: Batch 860/870. Loss of 2.1424, cover loss of 1.0371, secret loss of 1.1052
Training: Batch 861/870. Loss of 2.8083, cover loss of 1.2503, secret loss of 1.5580
Training: Batch 862/870. Loss of 2.8234, cover loss of 1.4916, secret loss of 1.3319
Training: Batch 863/870. Loss of 2.2356, cover loss of 1.1892, secret loss of 1.0463
Training: Batch 864/870. Loss of 1.2731, cover loss of 0.4540, secret loss of 0.8191
Training: Batch 865/870. Loss of 4.0607, cover loss of 2.2431, secret loss of 1.8177
Training: Batch 866/870. Loss of 3.1734, cover loss of 1.2897, secret loss of 1.8837
Training: Batch 867/870. Loss of 2.7140, cover loss of 1.1731, secret loss of 1.5408
Training: Batch 868/870. Loss of 4.1402, cover loss of 1.8778, secret loss of 2.2624
Training: Batch 869/870. Loss of 3.3671, cover loss of 2.0808, secret loss of 1.2863
Training: Batch 870/870. Loss of 2.7217, cover loss of 0.6595, secret loss of 2.0622
Epoch [3/3], Average_loss: 2.7090
f:\MATH 110A\FinalProject.py:392: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_secret = Variable(test_secret, volatile=True)
f:\MATH 110A\FinalProject.py:394: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_cover = Variable(test_cover, volatile=True)
Total loss: 2.26
Loss on secret: 1.55
Loss on cover: 0.71
Total loss: 3.79
Loss on secret: 0.82
Loss on cover: 2.97
Total loss: 1.97
Loss on secret: 1.16
Loss on cover: 0.81
Total loss: 2.03
Loss on secret: 0.86
Loss on cover: 1.17
Average loss on test set: 2.61
```

This is because the SGD need to use more epochs to finish training.

Adam: (which the author uses) testing loss is 1.29, training loss is 1.31

```
Training: Batch 850/870. Loss of 0.6433, cover loss of 0.0043, secret loss of 0.6390
Training: Batch 851/870. Loss of 1.4826, cover loss of 0.0031, secret loss of 1.4795
Training: Batch 852/870. Loss of 1.1072, cover loss of 0.0042, secret loss of 1.1030
Training: Batch 853/870. Loss of 0.6779, cover loss of 0.0040, secret loss of 0.6739
Training: Batch 854/870. Loss of 2.2714, cover loss of 0.0042, secret loss of 2.2671
Training: Batch 855/870. Loss of 1.2985, cover loss of 0.0084, secret loss of 1.2901
Training: Batch 856/870. Loss of 1.1800, cover loss of 0.0127, secret loss of 1.1672
Training: Batch 857/870. Loss of 1.3025, cover loss of 0.0039, secret loss of 1.2986
Training: Batch 858/870. Loss of 1.6549, cover loss of 0.0039, secret loss of 1.6510
Training: Batch 859/870. Loss of 1.0353, cover loss of 0.0041, secret loss of 1.0312
Training: Batch 860/870. Loss of 1.0060, cover loss of 0.0023, secret loss of 1.0037
Training: Batch 861/870. Loss of 1.0824, cover loss of 0.0017, secret loss of 1.0808
Training: Batch 862/870. Loss of 0.6250, cover loss of 0.0026, secret loss of 0.6223
Training: Batch 863/870. Loss of 1.0465, cover loss of 0.0020, secret loss of 1.0445
Training: Batch 864/870. Loss of 1.4767, cover loss of 0.0172, secret loss of 1.4595
Training: Batch 865/870. Loss of 1.2822, cover loss of 0.0045, secret loss of 1.2778
Training: Batch 866/870. Loss of 2.0761, cover loss of 0.0026, secret loss of 2.0734
Training: Batch 867/870. Loss of 1.3348, cover loss of 0.0035, secret loss of 1.3313
Training: Batch 868/870. Loss of 0.7506, cover loss of 0.0079, secret loss of 0.7426
Training: Batch 869/870. Loss of 1.5056, cover loss of 0.0041, secret loss of 1.5015
Training: Batch 870/870. Loss of 0.8331, cover loss of 0.0031, secret loss of 0.8301
Epoch [3/3], Average_loss: 1.3146
f:\MATH 110A\FinalProject.py:392: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_secret = Variable(test_secret, volatile=True)
f:\MATH 110A\FinalProject.py:394: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_cover = Variable(test_cover, volatile=True)
Total loss: 1.69
Loss on secret: 1.67
Loss on cover: 0.01
Total loss: 1.68
Loss on secret: 1.68
Loss on cover: 0.00
Total loss: 1.28
Loss on secret: 1.26
Loss on cover: 0.02
Total loss: 1.79
Loss on secret: 1.78
Loss on cover: 0.00
Average loss on test set: 1.29
```

RMSPROP(See the figure below):
Testing loss is 1.27, training loss is 1.35

```
Training: Batch 326/341. Loss of 1.7779, cover loss of 0.0656, secret loss of 1.7123
Training: Batch 327/341. Loss of 0.9205, cover loss of 0.1915, secret loss of 0.7290
Training: Batch 328/341. Loss of 1.6046, cover loss of 0.0729, secret loss of 1.5317
Training: Batch 329/341. Loss of 0.9872, cover loss of 0.0265, secret loss of 0.9608
Training: Batch 330/341. Loss of 0.9590, cover loss of 0.0178, secret loss of 0.9412
Training: Batch 331/341. Loss of 1.3388, cover loss of 0.0966, secret loss of 1.2422
Training: Batch 332/341. Loss of 1.3994, cover loss of 0.0362, secret loss of 1.3632
Training: Batch 333/341. Loss of 0.9453, cover loss of 0.0396, secret loss of 0.9058
Training: Batch 334/341. Loss of 1.1558, cover loss of 0.0154, secret loss of 1.1404
Training: Batch 335/341. Loss of 0.9366, cover loss of 0.0147, secret loss of 0.9219
Training: Batch 336/341. Loss of 2.3314, cover loss of 0.0325, secret loss of 2.2989
Training: Batch 337/341. Loss of 3.0707, cover loss of 0.0144, secret loss of 3.0563
Training: Batch 338/341. Loss of 1.3052, cover loss of 0.0251, secret loss of 1.2800
Training: Batch 339/341. Loss of 0.8446, cover loss of 0.0587, secret loss of 0.7858
Training: Batch 340/341. Loss of 1.3741, cover loss of 0.0271, secret loss of 1.3470
Training: Batch 341/341. Loss of 1.7827, cover loss of 0.0290, secret loss of 1.7537
Epoch [3/3], Average_loss: 1.3536
f:\MATH 110A\FinalProject.py:392: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_secret = Variable(test_secret, volatile=True)
f:\MATH 110A\FinalProject.py:394: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_cover = Variable(test_cover, volatile=True)
Total loss: 1.15
Loss on secret: 1.14
Loss on cover: 0.01
Total loss: 0.71
Loss on secret: 0.68
Loss on cover: 0.02
Total loss: 1.28
Loss on secret: 1.27
Loss on cover: 0.02
Total loss: 0.73
Loss on secret: 0.70
Loss on cover: 0.02
Average loss on test set: 1.27
```

In the final version, I use Adam because its training speed and accuracy.

Increase module and decrease module:

Increasing or decreasing modules in **torch.nn.sequential** does not have influence on the performance too much. Increasing may cause the loss fluctuation.

Final Model:

The final model provides the testing loss is 1.28, and training loss is 1.29

```
Training: Batch 294/300. Loss of 1.0892, cover loss of 0.0882, secret loss of 1.0010
Training: Batch 295/300. Loss of 1.3108, cover loss of 0.0360, secret loss of 1.2748
Training: Batch 296/300. Loss of 1.4874, cover loss of 0.0273, secret loss of 1.4601
Training: Batch 297/300. Loss of 0.6985, cover loss of 0.0218, secret loss of 0.6767
Training: Batch 298/300. Loss of 1.2844, cover loss of 0.0373, secret loss of 1.2472
Training: Batch 299/300. Loss of 1.1444, cover loss of 0.0505, secret loss of 1.0939
Training: Batch 300/300. Loss of 0.7013, cover loss of 0.0463, secret loss of 0.6550
Epoch [3/3], Average_loss: 1.2957
f:\MATH 110A\FinalProject.py:392: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_secret = Variable(test_secret, volatile=True)
f:\MATH 110A\FinalProject.py:394: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  test_cover = Variable(test_cover, volatile=True)
Total loss: 1.23
Loss on secret: 1.21
Loss on cover: 0.02
Total loss: 1.20
Loss on secret: 1.15
Loss on cover: 0.05
Total loss: 0.83
Loss on secret: 0.82
Loss on cover: 0.02
Total loss: 1.99
Loss on secret: 1.98
Loss on cover: 0.01
Average loss on test set: 1.28
```

This is the best attempt I had, both the training and testing loss is reduced by 3 percent than the original author's loss.

On test dataset:

For the test dataset, the test loss is the figure below:

```
Average loss on test set: 0.99
```

Summary:

I modify some activation function on the code and running numerous experiments to see how the parameter affects the model accuracy, and how to solve some specific problems. In the future, I will try to apply CNN on more datasets.