Haley Feng
June 15, 2019

Project 3: Steganography

Method	Data Set	Average RMSE	Final Loss
LSB	Kaggle	116.35	N/A
	Kodak	117.27	N/A
NN	Tiny ImageNet	17.63	484.34 (from original post)
	Kodak	126.51	6849.42 (w/ DCT)

➢ **Implementation with LSB**

To implement the least significant bits (LSB), I used the steganography code provided by Github user kelvins. Most of the code I wrote consist of loading datasets from Kaggle and Kodak. Since the samples from Kodak have different sizes, I used the crop function from Pillow Image module to crop out a 256x256 image from each and created a new data set. Afterward, I randomly picked two numbers from the range of 20 and use them as the indices to pick out the secret and cover images. Steganography is ran for 20 times to get a total of 20 merge images. RMSE is calculated using *mse* from sklearn.metrics and the results for the loss are shown above. Since the RMSE is not too high, the implementation of LSB works well by not losing too much information.

➢ **Implementation with NN**

The codes for the Neutral Network (NN) I implemented is by the Github user alexandremuzio. Before training the NN model with the 64x64 images from Tiny ImageNet, I reduce the information on the secret images *input_S* using Discrete Cosine Transform (DCT) with the norm parameter set to 'ortho'. I trained the model with 30 epoch as opposed to 1000 since every epoch take around 4000 seconds to run. The reduced Kodak dataset is test on the trained model and RMSE is calculated using the function *pixel_errors*. While RMSE on the Kodak set is higher than that of LSB, it still remains at a fairly good average (not above 180), meaning the model works well.

➢ **Optional**

In an attempt to see if LSB works with more secret images , I randomly selected three images from the Kaggle dataset and chose two images as secrets, *image2* and *image3*. Using steganography, I encode *image3* with *image2* to create *init_merge.* The initial merge is then merge with *image1* to create *final_merge*. When unmerging the *final_merge*, an image identical to *init_merge* is returned (labeled *unmerge_image1)*. However, when unmerging the *init_merge* to get the second secret image *image3*, the result is a black image. Encoding two secret images with LSB did not work most likely due to when *image1* encode the merge of *image2* and *image3*, the *image3's* pixels did not get carry on to the new merge image; *image1's* pixel took the first bits of secrets' merge which is *image2*. On the other hand, NN works splendidly in unmerging the two secret images from merge image by simply using the decoder and encoder functions. Therefore, I believe it would work as well if there are more images to encode.