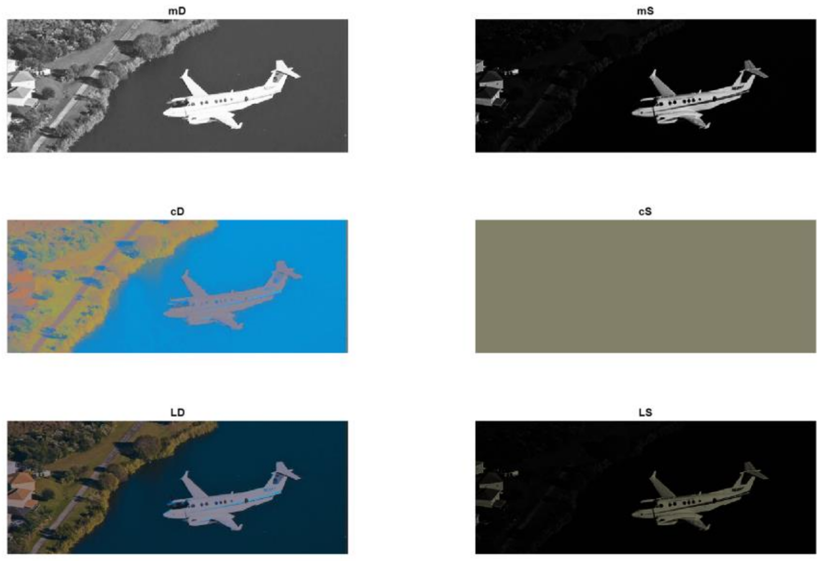
CIVIL 6415 Midterm Project

Elliot Lee

Submitted to Dr. Yilmaz



1. Program Environment

a) Windows

Using Matlab, execute "main_light.m". In order to run the full pipeline, Ubuntu 16.04 is required.

b) Ubuntu 16

Required Softwares/packages:

- git (assumed default)
- Matlab (assumed default)
- python2 (assumed default)
- python2 packages

```
$ chmod +x pythonReady.sh
$ sudo sh pythonReady.sh
```

- Tensorflow 1.3.1 or later (skip if already installed)

```
$ sudo apt-get install python-pip python-dev
$ sudo pip install tensorflow       (this assumes pip2)
```

- Keras (skip if already installed)

```
$ sudo pip install keras            (this assumes pip2)
or
$ sudo pip --no-cache-dir  install keras   (if memory error)
```

2. Running the Full-Pipe-Line Program on Ubuntu 16.

Using Matlab, execute "main_full.m"

If there is any hardware incompatibility or any circumstance that the program does not run, please let me know. I will bring a hardware to run.

3. Program Structure

A. Pipe Line

Since Matlab super pixel function is convenient, Matlab is applied to extract the relevant data regarding an image and its super pixel. The data is then saved as text files so that python application can use it for optimization using Keras with Tensorflow backend. The optimization result is then saved as text file so that Matlab can parse and plot the result images. This pipe line is shown in Figure 1.

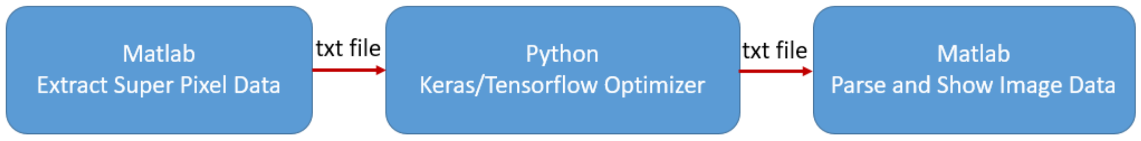


Figure 1: Program Pipe Line

As it can be understood from the Figure 1, in this assignment, Matlab is used for data package and parsing. In other words, the deep learning library is the back bone for the optimization sub routine. Each values of mS, cS, mD, and cD are viewed as function outputs where the function is specifically designed neural network. Then, the outputs are used to calculate the loss so that it can be used for neural network optimization. The optimized neural network at each epoch will output better mS, cS, mD, and cD values for each pixel in the image.

B. Optimizer Structure

There are 4 different neural networks which outputs mD, cD, and mS based on the observed pixel values while cS output is based on initial random RGB value.

$$mD_{ij} = NN_1\left(R_{ij}, G_{ij}, B_{ij}\right)$$
$$mC_i = NN_2(R_{ij}, G_{ij}, B_{ij})$$
$$mS_{ij} = NN_3\left(R_{ij}, G_{ij}, B_{ij}\right)$$
$$mS = NN_3(R, G, B\ )$$

where i is ith super pixel, j is jth original pixel in the super pixel, NN1 and NN3 are fully connected layer with 2 hidden layers, NN2 is long short term memory (LSTM) with 100 hidden steps, and NN3 is fully connected layer with 1 hidden layer.

- Pre-Training

Especially, each neural network block is pre-trained for the fast convergence as shown in Figure 2. NN1 is function that outputs mD values given the observed pixel. For the faster convergence, it is pre-tained with the corresponding gray scale image. This pre-trained network is copied and used for mS as well. NN2 is the LSTM that outputs the mean RGB value of a given set of RGB pixels in each super pixel. Lastly, the NN4 outputs the same value as the same input RGB (1X3) vector.

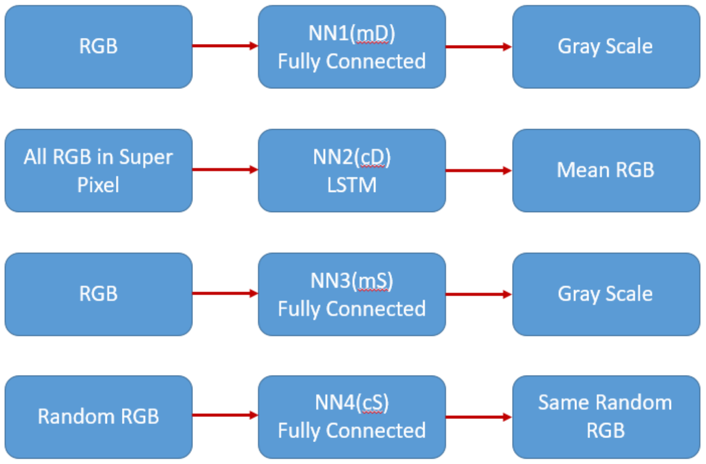


Figure 2: Pre-training Schedule for each neural network estimator.

Once all these 4 networks are trained, they are ready for the further optimization.

- Optimization

Next, each output is concatenated into one matrix as shown in Figure 3.

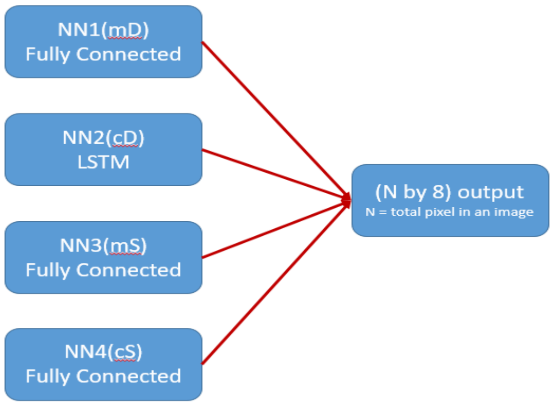


Figure 3: Concatenated Outputs

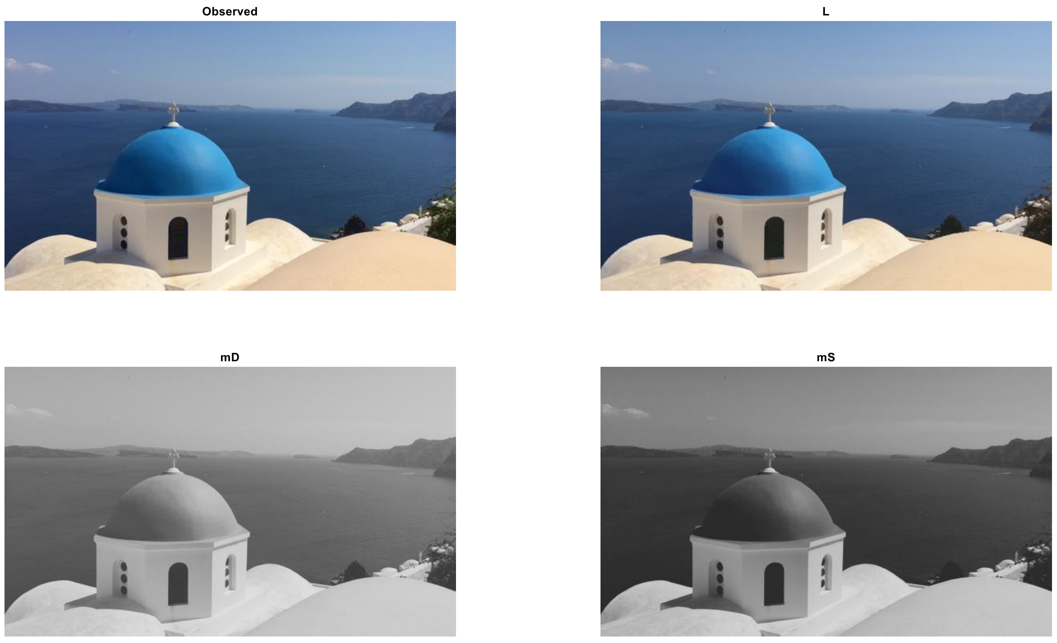
First column of the concatenated output is mD, 2nd, 3rd, 4th are cD, and mS, and cD correspondingly. Then, simply the loss is calculated using the equation provided from the instruction handout; it is repeated as below.

$$r_{ij} = m_D(x_{ij})\mathrm{C_D}(\mathrm{X_k}) + \mathrm{m_S}(x_{ij})\mathrm{C_S} - \mathrm{C_L}(\mathrm{x_{ij}})$$

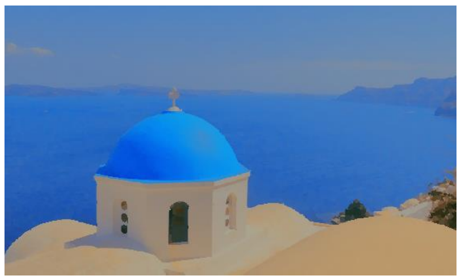
The optimizer used here is Adam, adaptive momentum estimation, which is built in the deep learning libraries for stochastic gradient descent. The loss will update the functions so that they can output better mD, cD, mS, and cS. Then the loss is calculated again and feed back to the optimizer. The optimizer is stopped when the sum squared of error (SSE) of the 0~1 scale RGB values are about 10~15. Yet this value can be different by the size of the batch. For instance, SSE of about 10 is when the number of pixels per batch is 600K while SSE is about 0.0003 when batch size is 0.1K.

4. Results

Image1

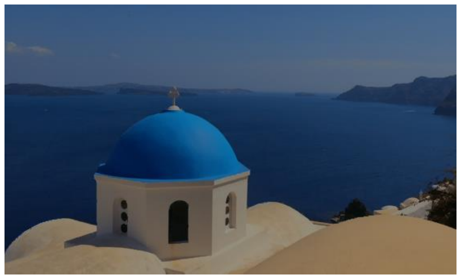


cD



cS

LD



LS

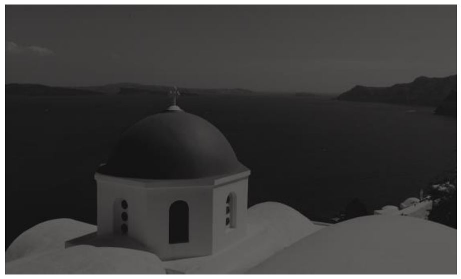
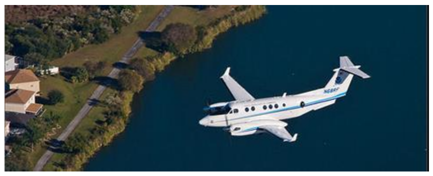
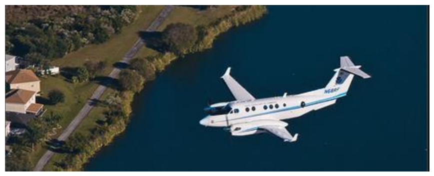


Image2

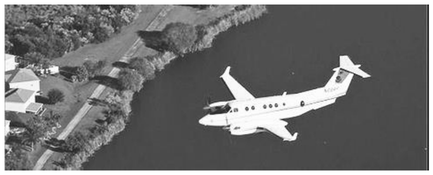
Observed



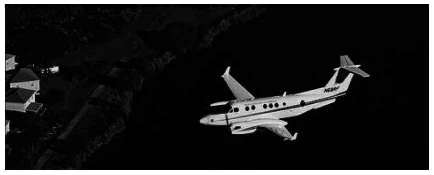
L



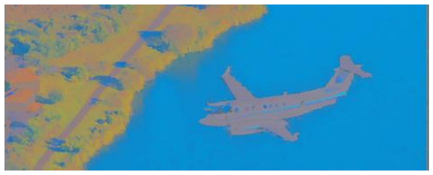
mD



mS

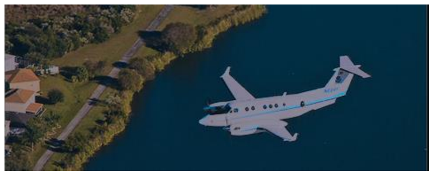


cD



cS

LD



LS

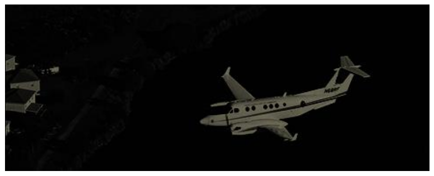
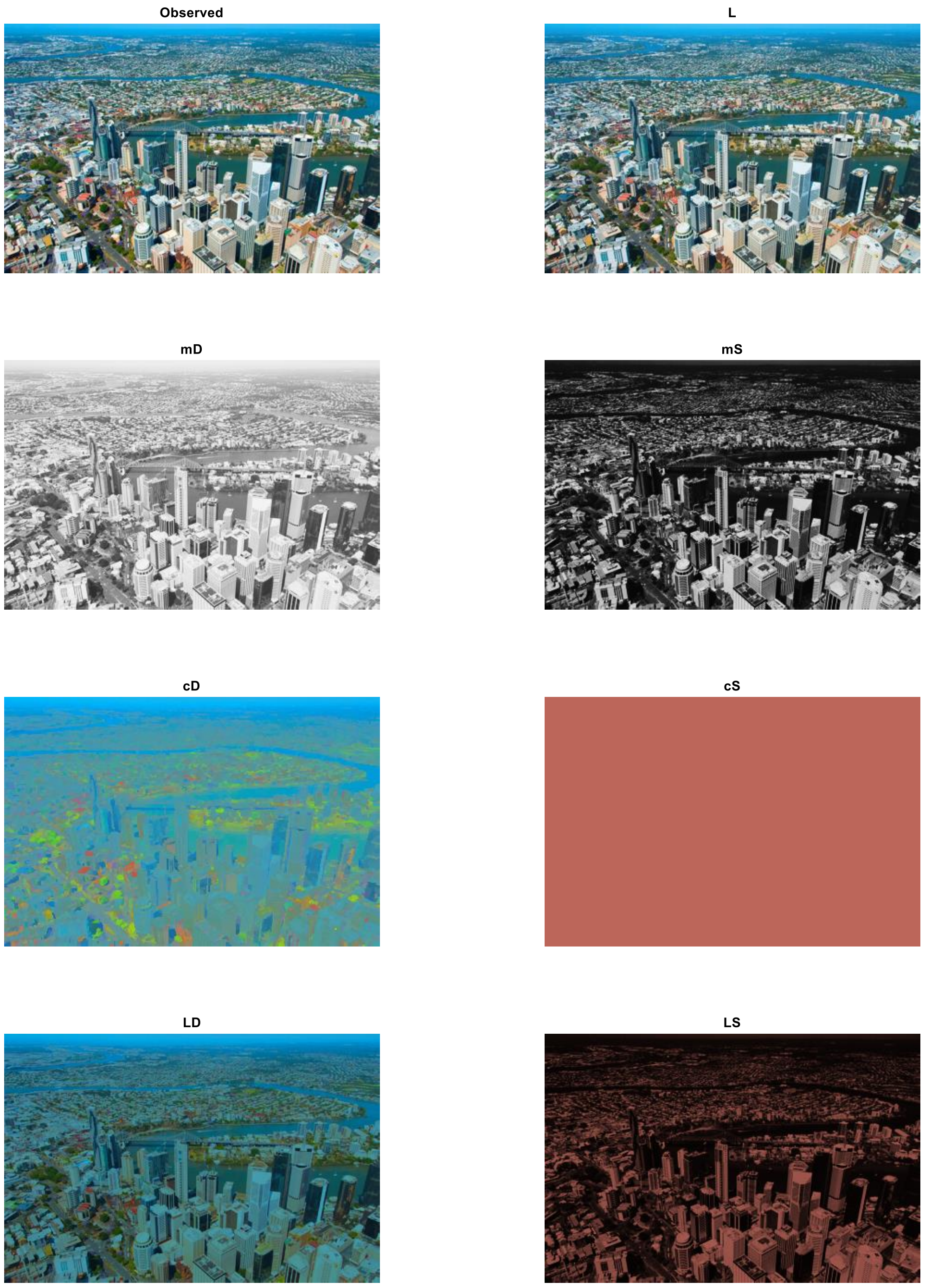
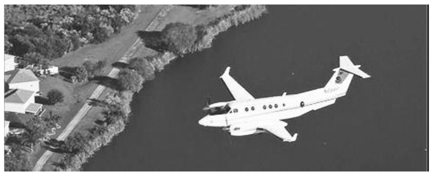


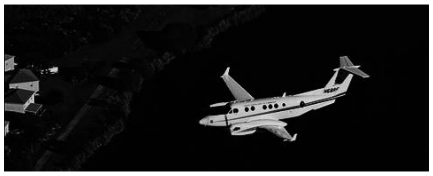
Image3



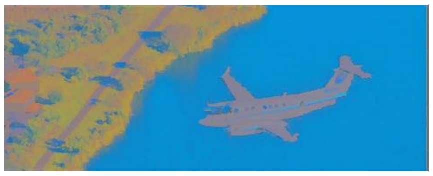
5. Discussion

First, I was worried if each of the pre-trained neural nets can be further optimized for our goal especially, mS and mD values because their pre-trained model is RGB to Gray scale function. Surprisingly, the optimization can further found local minimal point for our objective. The figures below show that the same RGB2Gray functions can separately evolve into mS and mD functions.





Also, for cD and cS functions, they evolved into the corresponding functions as well as shown below.



This shows that the LSTM that pre-trained to output mean RGB values of the pixels in a super pixel can be further optimized to only output the true colors assuming that the global objective's assumption is correct. Also, cS's initial value is random RGB values and it can find a local its own local minimum. Yet, there is no possible way to recover true color of an pixel because some portion of cS can be moved to cD. It depends on the local minimum and luck to recover true cD since there is no constraints for it. As lots of photographers do, however, if we can add color chart in the image, the known true color value can be extra constraints for optimization goal. For example, Loss = [estimated – Original] + [cD – color chart]. Then the network can be trained for verity of training set (images with the color chart), then the end-to-end estimator might be possible.