Jose Caballero  
Dr. Fuentes  
Computer Vision

Lab 5

**Problem Description**

The problem for this lab was to familiarize ourselves with machine learning using Keras, segmentation, and tracking objects for each problem separately.

**Algorithms implemented**

For this lab, no algorithms were implemented on my own, but rather I made use of some built-in algorithms such as the relu, adam, sigmoid, etc. These algorithms were used in the machine learning part of the lab.

**Experimental Results**

**Problem 1**

For classifying the cats and dogs, I did multiple runs but will only show two of the runs. Both runs were validated using 80% of the 12,500 pictures, which was approximately 10,000 pictures for each both dogs and cats. The training data were 2496 pictures of both dogs and cats that did not have any labels. The structure of the images were set as follow:

* Data
  + Train
    - Train
      * 1.jpg
      * 2.jpg
      * …
      * 2496.jpg
  + Validation
    - Cats
      * Cat.0.jpg
      * …
      * Cat.10000.jpg
    - Dogs
      * Dog.0.jpg
      * …
      * Dog.10000.jpg

It took me a while to find out that the train data had to have a subfolder names the same for keras to actually run. Anyways, the first run I did was using 20 epochs. The output log is the following:

Found 2496 images belonging to 1 classes.  
Found 1998 images belonging to 2 classes.

Epoch 1/20

125/125 [==============================] - 103s 823ms/step - loss: 0.0075 - acc: 0.9925 - val\_loss: 7.7233 - val\_acc: 0.5208

Epoch 2/20

125/125 [==============================] - 99s 791ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.3389 - val\_acc: 0.4826

Epoch 3/20

125/125 [==============================] - 99s 793ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.2269 - val\_acc: 0.4896

Epoch 4/20

125/125 [==============================] - 99s 794ms/step - loss: 3.9319e-06 - acc: 1.0000 - val\_loss: 7.8352 - val\_acc: 0.5139

Epoch 5/20

125/125 [==============================] - 99s 793ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.3948 - val\_acc: 0.4792

Epoch 6/20

125/125 [==============================] - 99s 793ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.3948 - val\_acc: 0.4792

Epoch 7/20

125/125 [==============================] - 99s 793ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 7.4955 - val\_acc: 0.5350

Epoch 8/20

125/125 [==============================] - 99s 793ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.6187 - val\_acc: 0.4653

Epoch 9/20

125/125 [==============================] - 99s 791ms/step - loss: 1.0315e-07 - acc: 1.0000 - val\_loss: 8.5627 - val\_acc: 0.4688

Epoch 10/20

125/125 [==============================] - 99s 793ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 7.7233 - val\_acc: 0.5208

Epoch 11/20

125/125 [==============================] - 99s 792ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 7.4434 - val\_acc: 0.5382

Epoch 12/20

125/125 [==============================] - 99s 793ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.5627 - val\_acc: 0.4688

Epoch 13/20

125/125 [==============================] - 99s 791ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 7.6113 - val\_acc: 0.5278

Epoch 14/20

125/125 [==============================] - 99s 788ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 7.9463 - val\_acc: 0.5070

Epoch 15/20

125/125 [==============================] - 99s 791ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 7.3315 - val\_acc: 0.5451

Epoch 16/20

125/125 [==============================] - 99s 792ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 7.7233 - val\_acc: 0.5208

Epoch 17/20

125/125 [==============================] - 99s 792ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 7.7792 - val\_acc: 0.5174

Epoch 18/20

125/125 [==============================] - 99s 791ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.8426 - val\_acc: 0.4514

Epoch 19/20

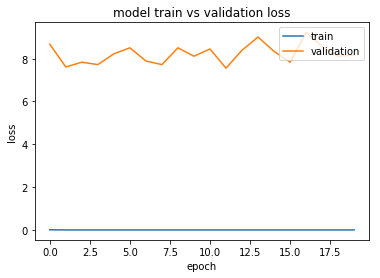
125/125 [==============================] - 99s 792ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.2829 - val\_acc: 0.4861

Epoch 20/20

125/125 [==============================] - 99s 792ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.6187 - val\_acc: 0.4653

I roughly got an accuracy of 43-51% with the following parameters

Config { Activation = relu, Dense(64), Dropout(0.5), Activation = Sigmoid, optimizer = Adam. }



Using pyplot I was able to plot the model train vs the validation loss of this run. I noticed that my train was linear, it did not increase or decrease, while my validation decreased and increased constantly. I thought this was weird so I ran only 5 epochs, which I will describe next.

The output for running 5 epochs is as follows

Found 2496 images belonging to 1 classes.  
Found 20974 images belonging to 2 classes.

Epoch 1/5

125/125 [==============================] - 95s 757ms/step - loss: 0.0061 - acc: 0.9955 - val\_loss: 7.2755 - val\_acc: 0.5486

Epoch 2/5

125/125 [==============================] - 90s 723ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.0031 - val\_acc: 0.5035

Epoch 3/5

125/125 [==============================] - 89s 714ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.1150 - val\_acc: 0.4965

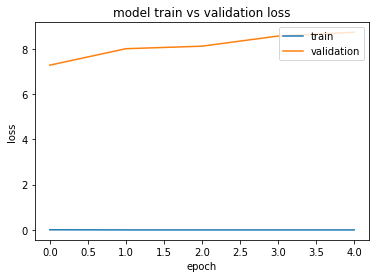
Epoch 4/5

125/125 [==============================] - 90s 722ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.5627 - val\_acc: 0.4688

Epoch 5/5

125/125 [==============================] - 90s 720ms/step - loss: 1.0000e-07 - acc: 1.0000 - val\_loss: 8.7306 - val\_acc: 0.4583

As you can see, I got pretty much around the same accuracy, which was around 49%. I also plotted the model train vs the validation loss and I got the following



The validation for this case slowly increased, while my train data remained linear as in the other run. I tried looking into why was this happening, and found out about underfitting and overfitting, but it does not seem to be the case with me. I am not sure if I could not get it running correctly as I am suspicious of both linear lines in train for both runs. I did follow two examples that I found in the Keras documentation, and the other for the actual cat and dog image classifier contest that Kaggle hosts.

The running time for both were the following:

Run 1 (20 epochs):  
Run 2 (5 epochs): 8:45 mins

**Problem 2:**

I ran many different pictures, mostly bright and colorful images, but I will only discuss 5 of the pictures, as the pictures take a lot of space.



Figure 1 Original img

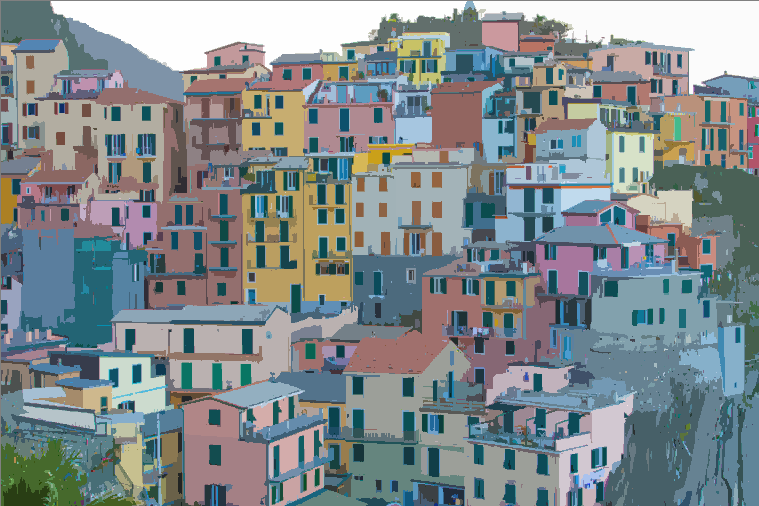


Figure 2 Segmented with original code



Figure 3 Segmented after modifying code



Figure 4 Original img



Figure 5 Segmented with original code – Ran in 13.93 seconds



Figure 6 Segmented after modifying code – Ran in 17.84 seconds



Figure 7 Original img



Figure 8 Segmented with original code - Ran in 13.86 seconds



Figure 9 Segmented after modifying code - Ran in 16.42 seconds



Figure 10 Original img



Figure 11 Segmented with original code- Ran in 15.42 seconds



Figure 12 Segmented after modifying code - Ran in 19.56 seconds



Figure 13 Original img

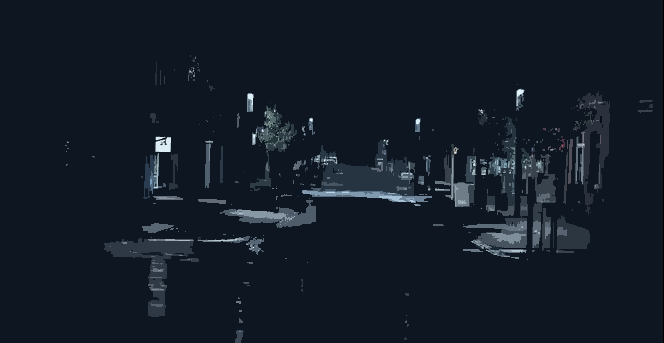


Figure 14 Segmented with original code - Ran in 4.63 seconds



Figure 15 Segmented after modifying code - Ran in 5.64 seconds



Figure 16 Segmented after modifying code - Ran in 5.94 seconds (Threshold of 0.5)

Changes done to the code were joining diagonal connections, and changing the threshold from 2.0 to 1.0. I attempted to make the code more efficient by changing the for loops, but was unable to do so using the pythonic ways.

Figures 1 – 3 “Town”   
The original pictures is of a town I could not find the name of but, after segmenting the image with the original code, some of the mountain-ish parts get lost, especially in the lower right hand corner and at the top towards the middle, it looks like just one big area. After modifying the code with the parameters discussed at the beginning, the areas become more clearer and you can actually differentiate between them now.

Figures 4 - 6 “Rainbow”

The original picture is of a woman’s face from a very close point of view. After segmenting the code with the original code, most of the woman’s skin color gets lost and is just assigned one big area, you can see this around the nose and the eyes, specially in the eyebrow section, it looks like one big area. After running my code, you can now appreciate the different skin tones and how they change from the nose towards the eye. The rainbow colors of her eye shadow is now more visible and her eyebrow Is way clearer. Her right eye drastically improves a lot as well.

Figures 7 – 9 “Tucan”

The original picture is of a toucan with very bright colors. The important details in this picture are the hairs arounds its chest area. After running the original code, most of the yellow hair gets lost, and you can see like only 3 changes in hair color in its chest. After running mine, you are able to see the different shadows and how they change around the chest area.

Figures 10 – 12 “Shadow man”

The original picture is of a mans shadow from what looks like a cave. The outside forest is visible as well. After running the original code, most, if not all of the forest gets lost in grayish and black colors. Most of the detail is lost. After running my version of the code, you are able to see the forest again. It is a massive improvement from the original segmented one. Even though the photo is mostly dark/greenish, you can appreciate it all the details after segmentation.

Figures 13 – 16 “Street”

The original picture is of a picture taken late at night and you can barely see the buildings at the end of the street. After segmenting with the original code, most of the street becomes black, and most details are lost. You can more or less still look at the sidewalk and the lights, but not really clearly. After segmenting with my code, you are able to look at most of the picture clearer than before. Most of the street at the beginning of the picture is still kind of lost, and the shadows cast by the trees are also joint together, even though in the original picture you can clearly tell which shadow belongs to what. Also, most of the top of the buildings, and the building at the end blended together with the night sky. After lowering the threshold for this picture from 1.0 to 0.5, the picture becomes clearer, but that was to be expected.

**Problem 3**

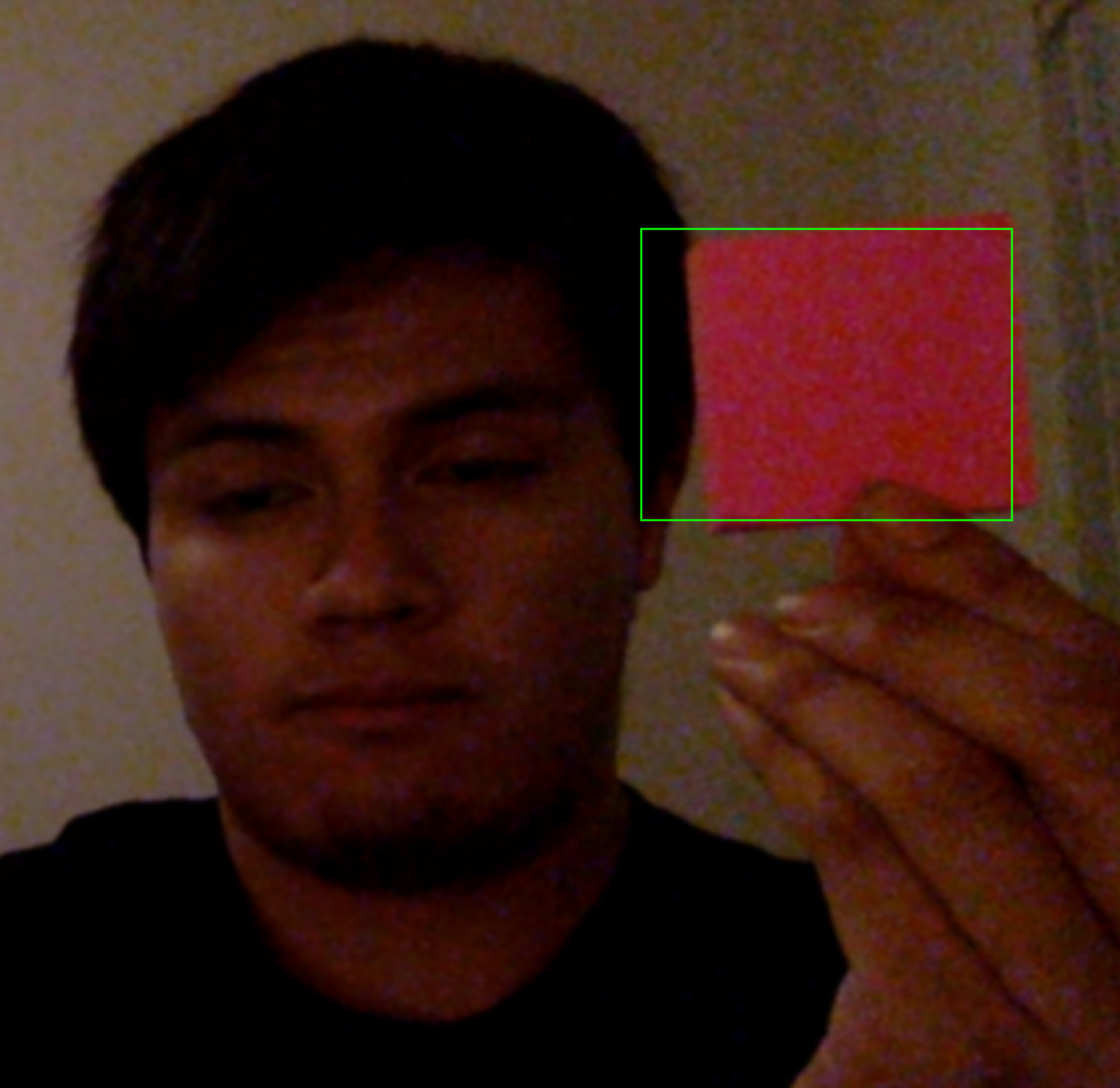


Figure 16 Bad picture of me holding a post-it after detection

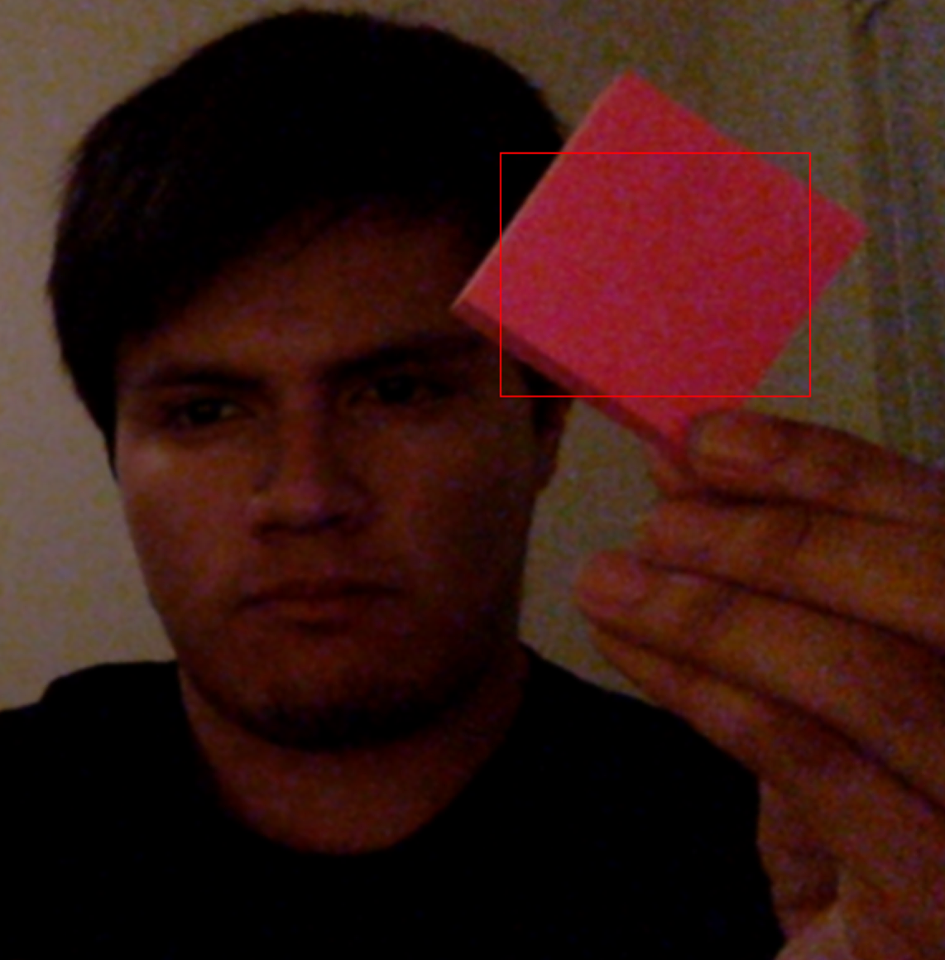


Figure 17 Region not detected

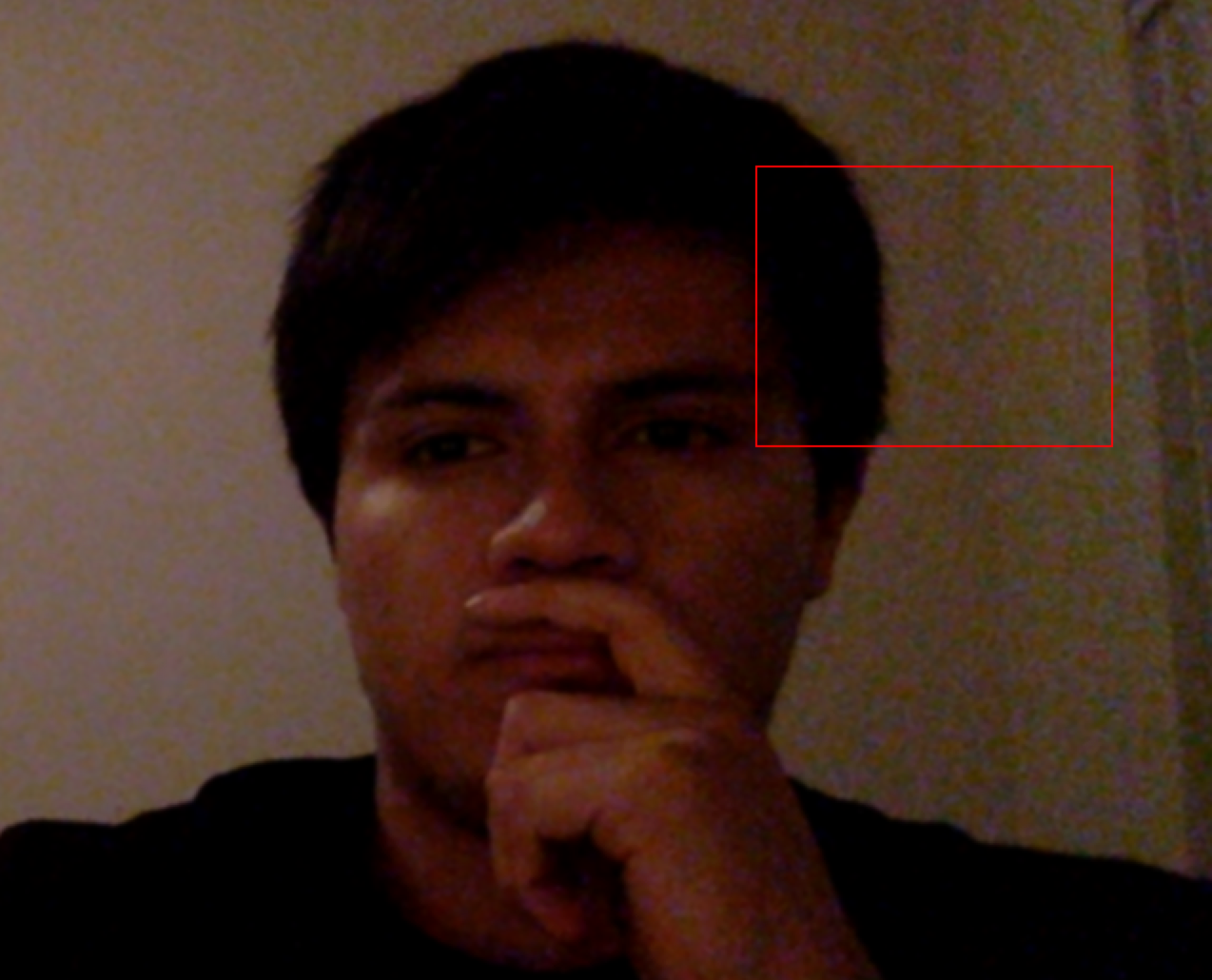


Figure 18 Region not detected

For this problem, what I did was just changing the behavior of the rectangle on the detected pattern. For this, I used a post-it note to track. I used a threshold and anytime the match object was higher than my threshold, I would change the color of the rectangle from green to red to demonstrate the pattern was no longer detected. If the threshold would then come back to an “acceptable” value, the rectangle would change back to green to demonstrate the pattern was detected once again. Something that could be done instead of having a red color rectangle, would be to simply make the rectangle disappear, but I thought it was nice to let the rectangle stay to see how big the original pattern was. I tried using ginput but was not successful in adding it to the existing code. I wanted to make it so whenever a certain button was detected, for the program to basically ask for a new pattern, you would supply the two clicks again, and then it would use that new pattern.

**Discussion of results**

The results for problem 2 and 3 were kind of straightforward, I knew what I had to do right away, but I am still having some trouble for the machine learning part. Since the results for part 2 and 3 is something you can actually look at like a picture or a video, it is infinitely easier to understand what is happening. For problem 1, I had a little bit of trouble setting up the training, and even understanding what I was getting as a result.

Conclusions

Appendix