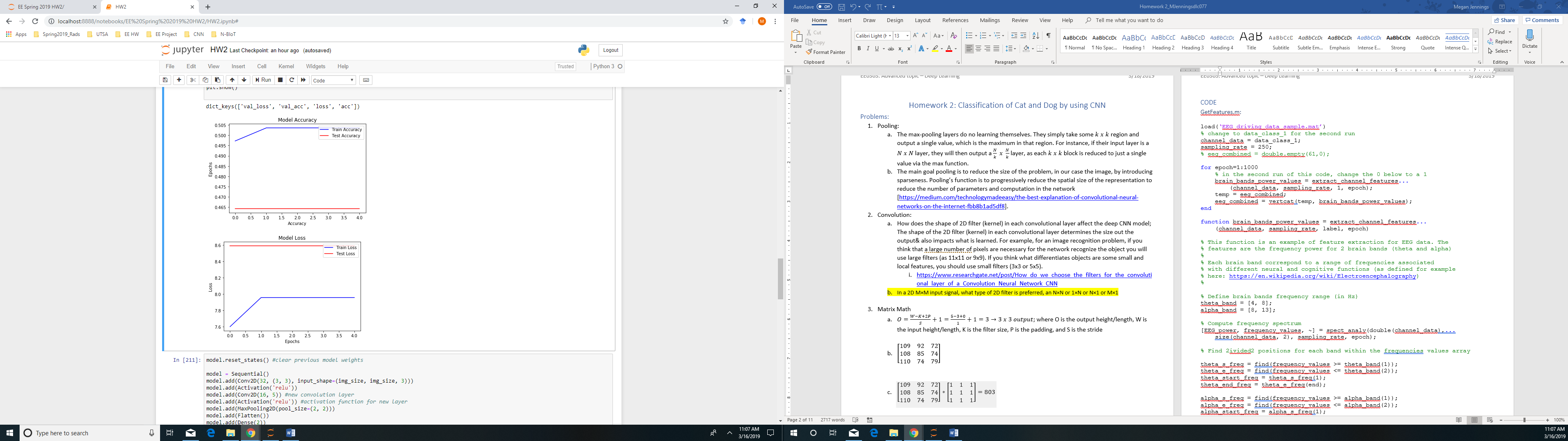
# Homework 2: Classification of Cat and Dog by using CNN

## Problems:

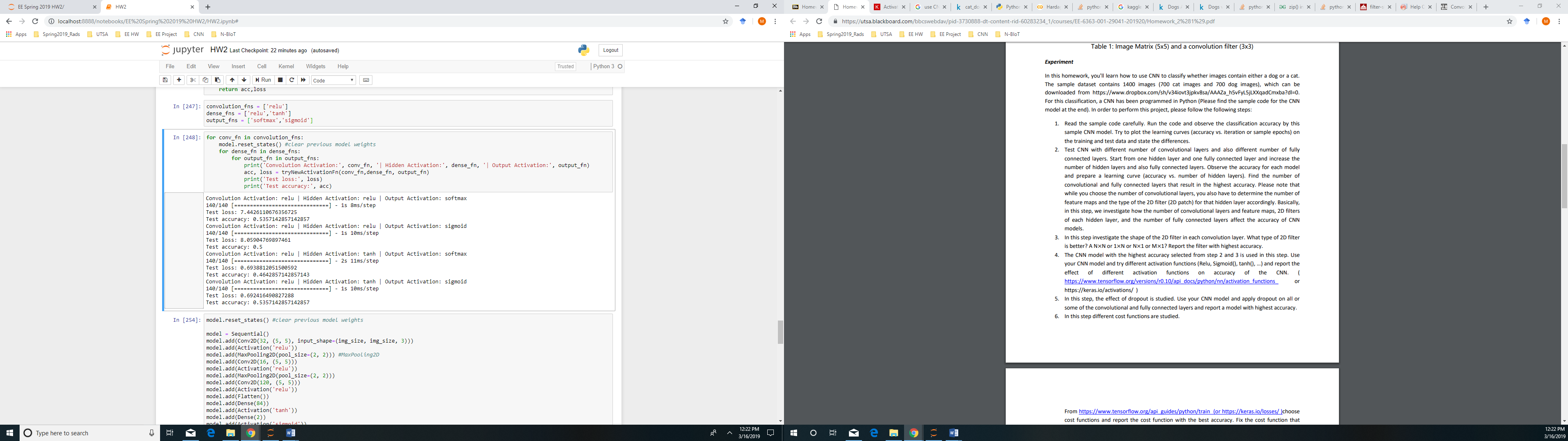
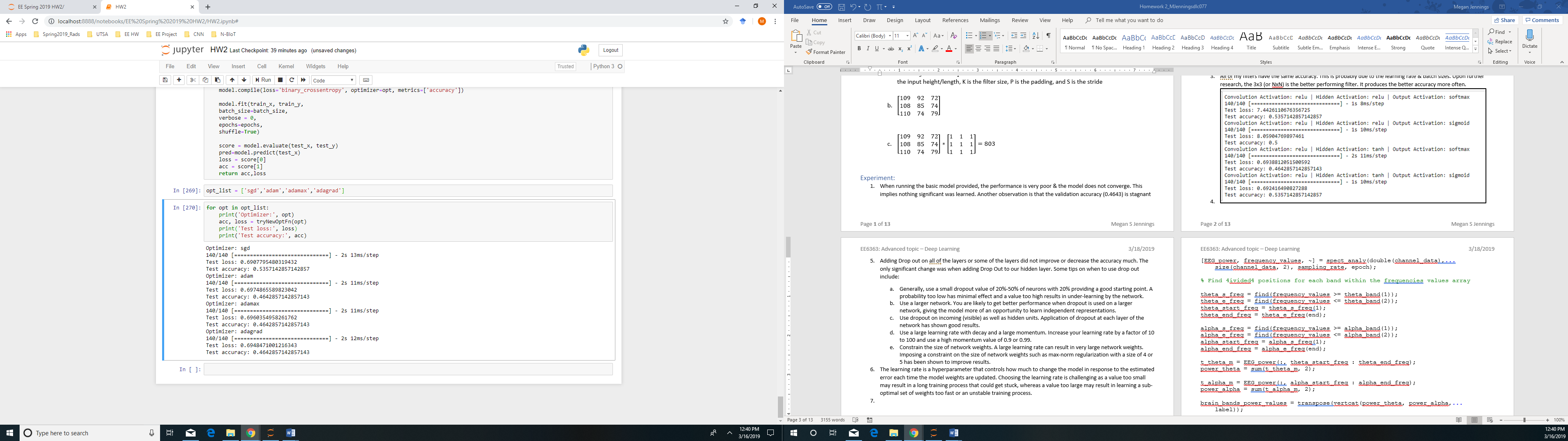
1. Pooling:
   1. The max-pooling layers do no learning themselves. They simply take some region and output a single value, which is the maximum in that region. For instance, if their input layer is a layer, they will then output a layer, as each block is reduced to just a single value via the max function.
   2. The main goal pooling is to reduce the size of the problem, in our case the image, by introducing sparseness. Pooling’s function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network [<https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8>].
2. Convolution:
   1. How does the shape of 2D filter (kernel) in each convolutional layer affect the deep CNN model; The shape of the 2D filter (kernel) in each convolutional layer determines the size out the output & also impacts what is learned. For example, for an image recognition problem, if you think that a large number of pixels are necessary for the network recognize the object you will use large filters (as 11x11 or 9x9). If you think what differentiates objects are some small and local features, you should use small filters (3x3 or 5x5).
      1. <https://www.researchgate.net/post/How_do_we_choose_the_filters_for_the_convolutional_layer_of_a_Convolution_Neural_Network_CNN>
   2. In a 2D M×M input signal, what type of 2D filter is preferred, an N×N or 1×N or N×1 or M×1; NxN performs better as it will go through more rows & columns to try to identify the various features of the image (edges, borders, etc.)
3. Matrix Math
   1. ; where O is the output height/length, W is the input height/length, K is the filter size, P is the padding, and S is the stride

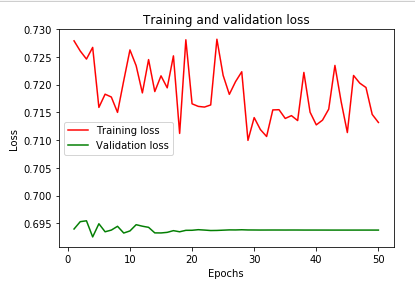
## Experiment:

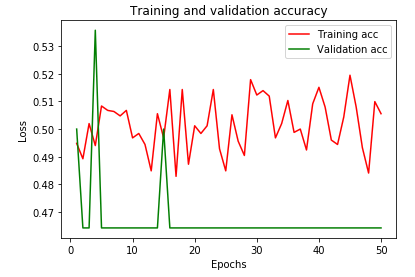
1. When running the basic model provided, the performance is very poor & the model does not converge. This implies nothing significant was learned. Another observation is that the validation accuracy (0.4643) is stagnant & not changing as expected. This typically means the learning rate is too high. This is further demonstrated via the Model Accuracy plot.

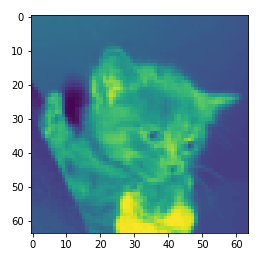
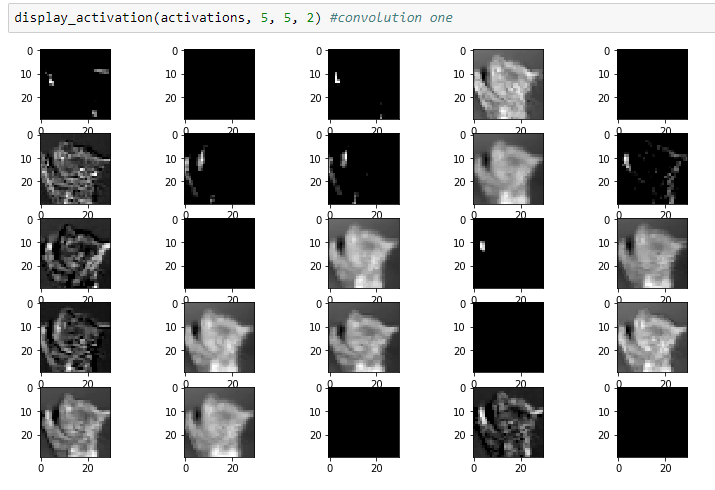
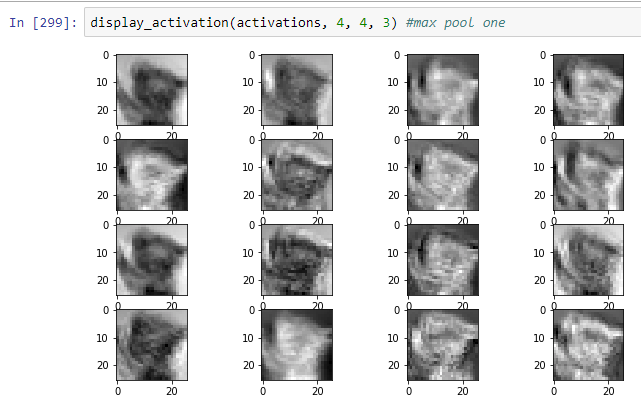
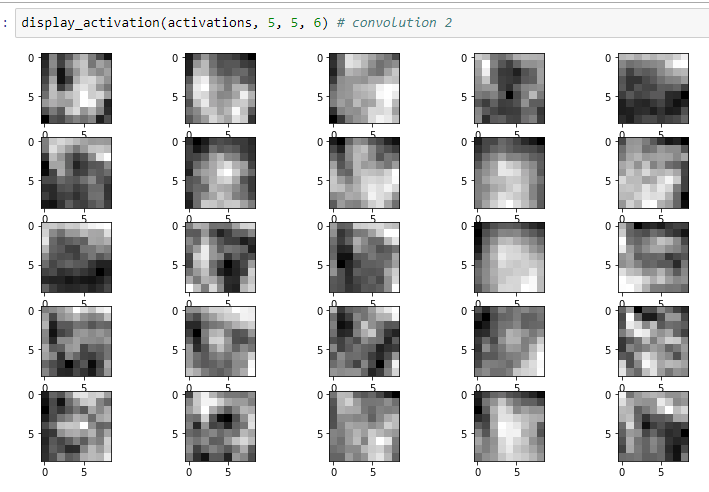


Using a sigmoid function for the Conv layer is effectively killing our leaning by causing a vanishing gradient. Additionally, the small batch size helps identify the fact that the photos are not standardized, which is causing some of the stagnant validation accuracy. This can be addressed by replacing the sigmoid to a relu.

1. Testing the CNN by adding and removing various hidden & fully connected layers was done in the code section of this paper. Adding additional Conv layers was more beneficial to adding additional dense layers.
2. All of my filters have the same accuracy. This is probably due to the learning rate & batch sizes. Upon further research, the 3x3 (or NxN) is the better performing filter. It produces the better accuracy more often.
3. 
4. Adding Drop out on all of the layers or some of the layers did not improve or decrease the accuracy much. The only significant change was when adding Drop Out to our hidden layer. Some tips on when to use drop out include:
   1. Generally, use a small dropout value of 20%-50% of neurons with 20% providing a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network.
   2. Use a larger network. You are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations.
   3. Use dropout on incoming (visible) as well as hidden units. Application of dropout at each layer of the network has shown good results.
   4. Use a large learning rate with decay and a large momentum. Increase your learning rate by a factor of 10 to 100 and use a high momentum value of 0.9 or 0.99.
   5. Constrain the size of network weights. A large learning rate can result in very large network weights. Imposing a constraint on the size of network weights such as max-norm regularization with a size of 4 or 5 has been shown to improve results.
5. Optimizer & Learning Rate
   1. The best optimizer for this problem is the sgd optimizer as can be seen in the image below. 
   2. The learning Rate of 0.1 that was provided in the original code looks to be the best Learning Rate. I tried 0.001, 0.01, 1.0 and none performed better.





1. I selected image 35 in my dataset. It happens to be a cat.
   1. Actual Image 
   2. Results after Convolution 1 
   3. Results after Max Pool 1 
   4. Results after Convolution 2 

You can see in the images a progression of what is being learned at each layer. First it is finding foreground & background. Following that the edges are detected. After that more detailed depth is being analyzed. They show what each neuron “wants to see”, and thus what each neuron has learned to look for.

## CODE

<https://github.com/msjennings/EE-6363-Homework-2/blob/master/HW2.ipynb>

**import** **os**

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D

**from** **sklearn.model\_selection** **import** train\_test\_split

**import** **numpy** **as** **np**

**from** **os** **import** listdir

**from** **skimage** **import** io

**from** **scipy.misc** **import** imresize

**import** **keras**

**from** **keras.optimizers** **import** Adam

**import** **pandas** **as** **pd**

**import** **pylab** **as** **plt**

img\_size = 64

batch\_size = 128

epochs = 5

*# load image*

data\_path='C:/Users/pli6894/Desktop/Train\_Data/'

labels = listdir(data\_path)

x\_cat=[];

x\_dog=[];

cat\_imgpath = listdir(data\_path+'/'+labels[0])

dog\_imgpath = listdir(data\_path+'/'+labels[1])

**for** img **in** cat\_imgpath:

cat\_img = io.imread(data\_path+'/'+labels[0]+'/'+img)

x\_cat.append(imresize(cat\_img, (img\_size, img\_size, 3)))

y\_cat=np.ones(len(cat\_imgpath))

**for** img **in** dog\_imgpath:

dog\_img = io.imread(data\_path + '/' + labels[1] + '/' + img)

x\_dog.append(imresize(dog\_img, (img\_size, img\_size, 3)))

y\_cat = np.zeros(len(cat\_imgpath))

y\_dog = np.ones(len(dog\_imgpath))

x=np.asarray(x\_cat+x\_dog)

y=np.append(y\_cat,y\_dog)

y = keras.utils.to\_categorical(y)

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x, y, test\_size=0.1, random\_state=300)

Basic Model:

model = Sequential() model.add(Conv2D(32, (3, 3), input\_shape=(img\_size, img\_size, 3))) model.add(Activation('sigmoid')) model.add(MaxPooling2D(pool\_size=(2, 2))) model.add(Flatten()) model.add(Dense(64)) model.add(Dropout(0.1)) model.add(Activation('sigmoid')) model.add(Dense(2)) model.add(Activation('softmax')) adamop=Adam(lr=0.1) model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy']) h = model.fit(train\_x, train\_y, validation\_data=(test\_x, test\_y), batch\_size=batch\_size, epochs=epochs, shuffle=**True**) score = model.evaluate(test\_x, test\_y) pred=model.predict(test\_x) print('Test loss:', score[0]) print('Test accuracy:', score[1])

*#Plot Accuracy & Loss for basic model*

print(h.history.keys())

fig = plt.figure()

plt.plot(h.history['acc'], color='blue')

plt.plot(h.history['val\_acc'], color='red')

plt.legend(['Train Accuracy', 'Test Accuracy'], loc='upper right')

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epochs')

fig

fig2 = plt.figure()

plt.plot(h.history['loss'], color ='blue')

plt.plot(h.history['val\_loss'],color ='red')

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epochs')

plt.legend(['Train Loss', 'Test Loss'], loc='upper right')

plt.show()

Investigate Kernel Size

**def** tryNewFilterSizeFn(h,w):

model = Sequential()

model.add(Conv2D(32, (h,w), input\_shape=(img\_size, img\_size, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) *#MaxPooling2D*

model.add(Conv2D(16, (h,w)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(120, (h,w)))

model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(84))

model.add(Activation('relu'))

model.add(Dense(2))

model.add(Activation('softmax'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(train\_x, train\_y,

batch\_size=batch\_size,

verbose = 0,

epochs=5,

shuffle=**True**)

score = model.evaluate(test\_x, test\_y)

pred=model.predict(test\_x)

loss = score[0]

acc = score[1]

**return** acc,loss

h\_list = [3,3,1,9]

w\_list = [3,1,3,1]

**for** h,w **in** map(**lambda** x,y:(x,y),h\_list,w\_list):

print('Filter Size : '+str(h)+','+str(w))

acc, loss = tryNewFilterSizeFn(h,w)

print('Test loss:', loss)

print('Test accuracy:', acc)

Investigate Activation Functions

**def** tryNewActivationFn(conv\_act, dense\_act, output\_act):

model = Sequential()

model.add(Conv2D(32, (5, 5), input\_shape=(img\_size, img\_size, 3)))

model.add(Activation(conv\_act))

model.add(MaxPooling2D(pool\_size=(2, 2))) *#MaxPooling2D*

model.add(Conv2D(16, (5, 5)))

model.add(Activation(conv\_act))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(120, (5, 5)))

model.add(Activation(conv\_act))

model.add(Flatten())

model.add(Dense(84))

model.add(Activation(dense\_act))

model.add(Dense(2))

model.add(Activation(output\_act))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(train\_x, train\_y,

batch\_size=batch\_size,

verbose = 0,

epochs=5,

shuffle=**True**)

score = model.evaluate(test\_x, test\_y)

pred=model.predict(test\_x)

loss = score[0]

acc = score[1]

**return** acc,loss

convolution\_fns = ['relu']

dense\_fns = ['relu','tanh']

output\_fns = ['softmax','sigmoid']

**for** conv\_fn **in** convolution\_fns:

model.reset\_states() *#clear previous model weights*

**for** dense\_fn **in** dense\_fns:

**for** output\_fn **in** output\_fns:

print('Convolution Activation:', conv\_fn, '| Hidden Activation:', dense\_fn, '| Output Activation:', output\_fn)

acc, loss = tryNewActivationFn(conv\_fn,dense\_fn, output\_fn)

print('Test loss:', loss)

print('Test accuracy:', acc)

Investigate Cost Function & Optimizers

**def** tryNewOptFn(opt):

model = Sequential()

model.add(Conv2D(32, (5, 5), input\_shape=(img\_size, img\_size, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) *#MaxPooling2D*

model.add(Conv2D(16, (5, 5)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(120, (5, 5)))

model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(84))

model.add(Activation('tanh'))

model.add(Dropout(rate = 0.2)) *#adding dropout to our hidden layer*

model.add(Dense(2))

model.add(Activation('softmax'))

model.compile(loss='binary\_crossentropy', optimizer=opt, metrics=['accuracy'])

model.fit(train\_x, train\_y,

batch\_size=batch\_size,

verbose = 0,

epochs=epochs,

shuffle=**True**)

score = model.evaluate(test\_x, test\_y)

pred=model.predict(test\_x)

loss = score[0]

acc = score[1]

**return** acc,loss

opt\_list = ['sgd','adam','adamax','adagrad']

**for** opt **in** opt\_list:

print('Optimizer:', opt)

acc, loss = tryNewOptFn(opt)

print('Test loss:', loss)

print('Test accuracy:', acc)

Visualize the CNN

**from** **keras.callbacks** **import** ModelCheckpoint,LearningRateScheduler

**import** **math**

**def** step\_decay(epoch):

initial\_lrate=0.01

drop=0.6

epochs\_drop = 3.0

lrate= initial\_lrate \* math.pow(drop,

math.floor((1+epoch)/epochs\_drop))

**return** lrate

lrate = LearningRateScheduler(step\_decay)

callbacks\_list = [ lrate]

model = Sequential()

model.add(Conv2D(32, (5, 5), input\_shape=(img\_size, img\_size, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) *#AveragePooling2D*

model.add(Conv2D(16, (5, 5)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(120, (5, 5)))

model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(84))

model.add(Activation('tanh'))

model.add(Dropout(rate = 0.2)) *#adding dropout to our hidden layer*

model.add(Dense(2))

model.add(Activation('sigmoid'))

model.summary()

model.compile(loss='binary\_crossentropy', optimizer='sgd', metrics=['accuracy'])

history=model.fit(train\_x, train\_y, validation\_data=(test\_x, test\_y),

epochs=50,callbacks=callbacks\_list,verbose=0)

score = model.evaluate(test\_x, test\_y)

pred=model.predict(test\_x)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

*#Training & Validation Loss*

**import** **matplotlib.pyplot** **as** **plt**

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, color='red', label='Training loss')

plt.plot(epochs, val\_loss, color='green', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

*# Training & Validation Accuracy*

acc = history.history['acc']

val\_acc = history.history['val\_acc']

plt.plot(epochs, acc, color='red', label='Training acc')

plt.plot(epochs, val\_acc, color='green', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

**from** **keras.models** **import** Model

layer\_outputs = [layer.output **for** layer **in** model.layers]

activation\_model = Model(inputs=model.input, outputs=layer\_outputs)

activations = activation\_model.predict(train\_x[35].reshape(1,64,64,3))

**def** display\_activation(activations, col\_size, row\_size, act\_index):

activation = activations[act\_index]

activation\_index=0

fig, ax = plt.subplots(row\_size, col\_size, figsize=(row\_size\*2.5,col\_size\*1.5))

**for** row **in** range(0,row\_size):

**for** col **in** range(0,col\_size):

ax[row][col].imshow(activation[0, :, :, activation\_index], cmap='gray')

activation\_index += 1

plt.imshow(train\_x[35][:,:,0]);

display\_activation(activations, 5, 5, 2) *#convolution one*

display\_activation(activations, 4, 4, 3) *#max pool one*

display\_activation(activations, 5, 5, 6) *# convolution 2*