Raymond Yung
Report 3

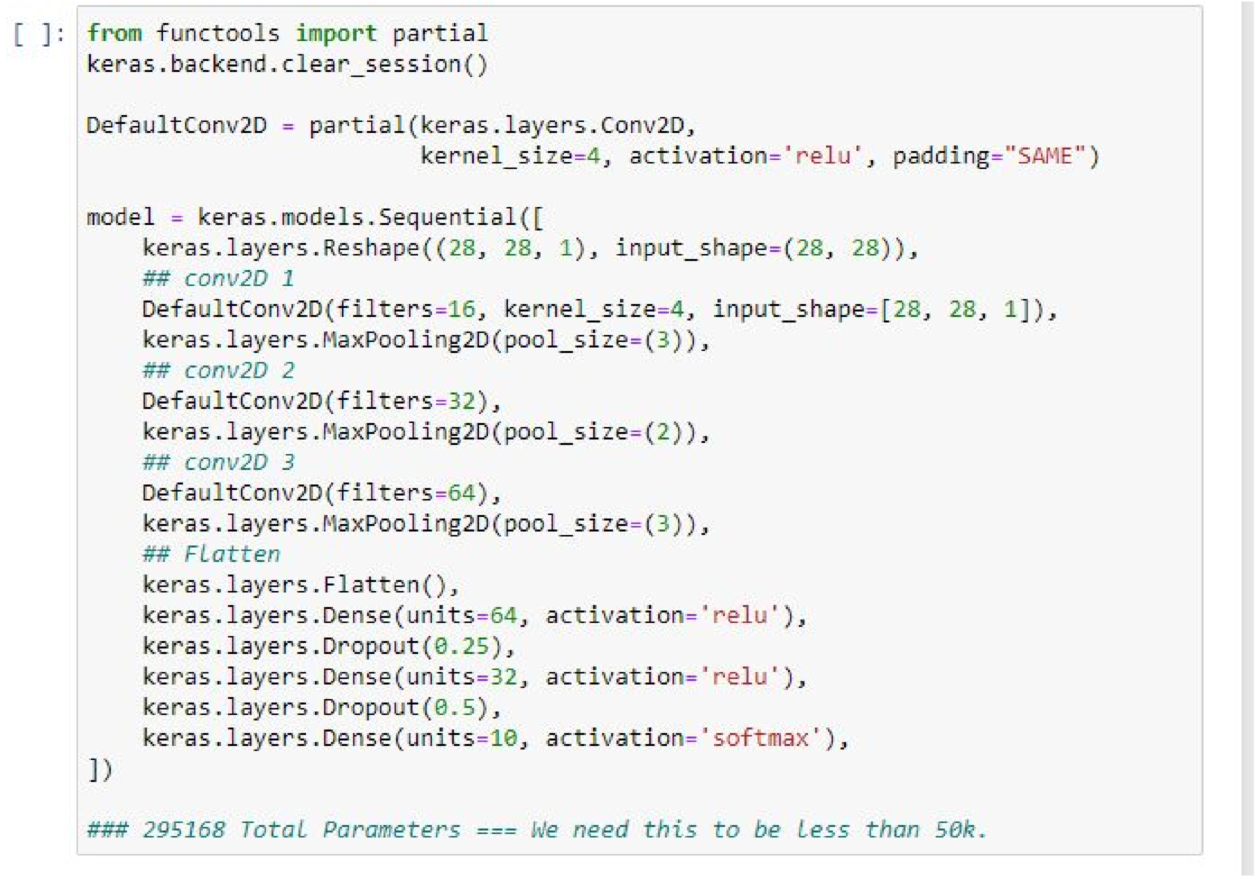
I created my model based on the default CNN model provided in chapter 14 in the textbook. I also added in a reshape layer, that reshaped the data into an acceptable parameter. I then reduced the amount of filters used along with increasing the pool size. Upon doing so, I reduced my parameters from 1.4 million to 70k.

By reducing the number of filters and kernels, I was then able to reduce the amount of dense layers in my model without losing out on that much accuracy. To further reduce my parameters, I began to experiment with my Max Pool values, which succeeded. This led to my final parameters being 47k.

I noticed a pattern while creating this model. I had began getting better results when my filter size increased incrementally by factors of 16* 2^x. This led me to developing my gridsearch method which we will discuss later.

I also used a checkpoint to save to roll back to my best method. I actually changed my model to rely on sparse_categorical_accuracy, as there is a bug that results in my saved model being .01% accurate after it is reloaded when it uses the [“accuracy”] metric.

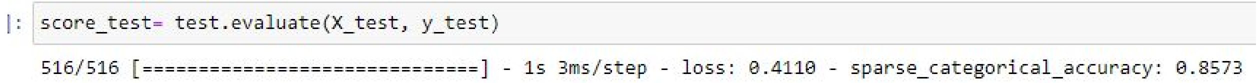
Figure 1: Model (no Gridsearch)



```
[ ]: from functools import partial
     keras.backend.clear_session()

     DefaultConv2D = partial(keras.layers.Conv2D,
                             kernel_size=4, activation='relu', padding="SAME")

     model = keras.models.Sequential([
         keras.layers.Reshape((28, 28, 1), input_shape=(28, 28)),
         ## conv2D 1
         DefaultConv2D(filters=16, kernel_size=4, input_shape=[28, 28, 1]),
         keras.layers.MaxPooling2D(pool_size=(3)),
         ## conv2D 2
         DefaultConv2D(filters=32),
         keras.layers.MaxPooling2D(pool_size=(2)),
         ## conv2D 3
         DefaultConv2D(filters=64),
         keras.layers.MaxPooling2D(pool_size=(3)),
         ## Flatten
         keras.layers.Flatten(),
         keras.layers.Dense(units=64, activation='relu'),
         keras.layers.Dropout(0.25),
         keras.layers.Dense(units=32, activation='relu'),
         keras.layers.Dropout(0.5),
         keras.layers.Dense(units=10, activation='softmax'),
     ])

     ### 295168 Total Parameters === We need this to be less than 50k.
```

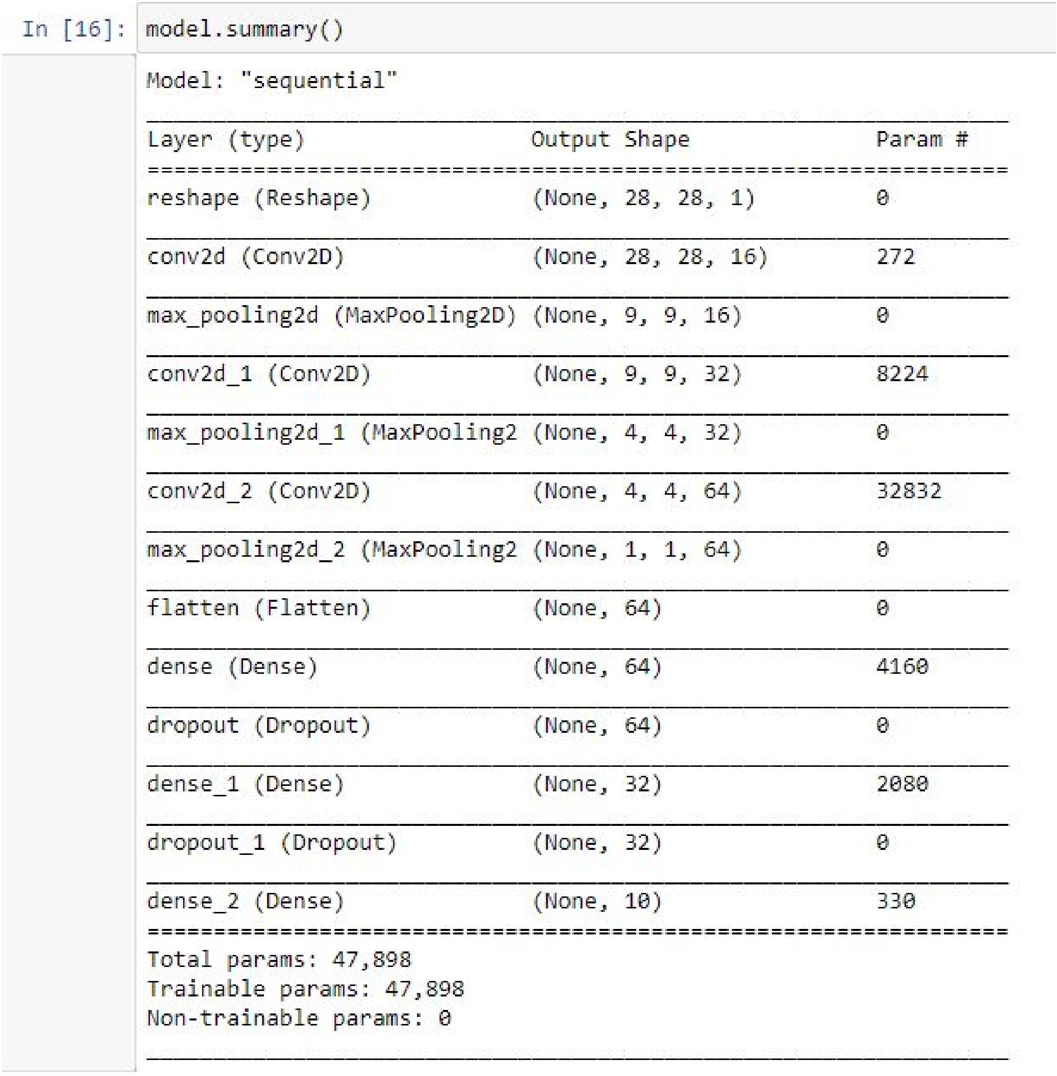
My final model here had an accuracy between 85 - 86%.

Figure 2: Model Final Test Accuracy after 20 Epochs (no overfitting)



```
]: score_test= test.evaluate(X_test, y_test)

   516/516 [==============================] - 1s 3ms/step - loss: 0.4110 - sparse_categorical_accuracy: 0.8573
```

Figure 3: Total Parameters of Model

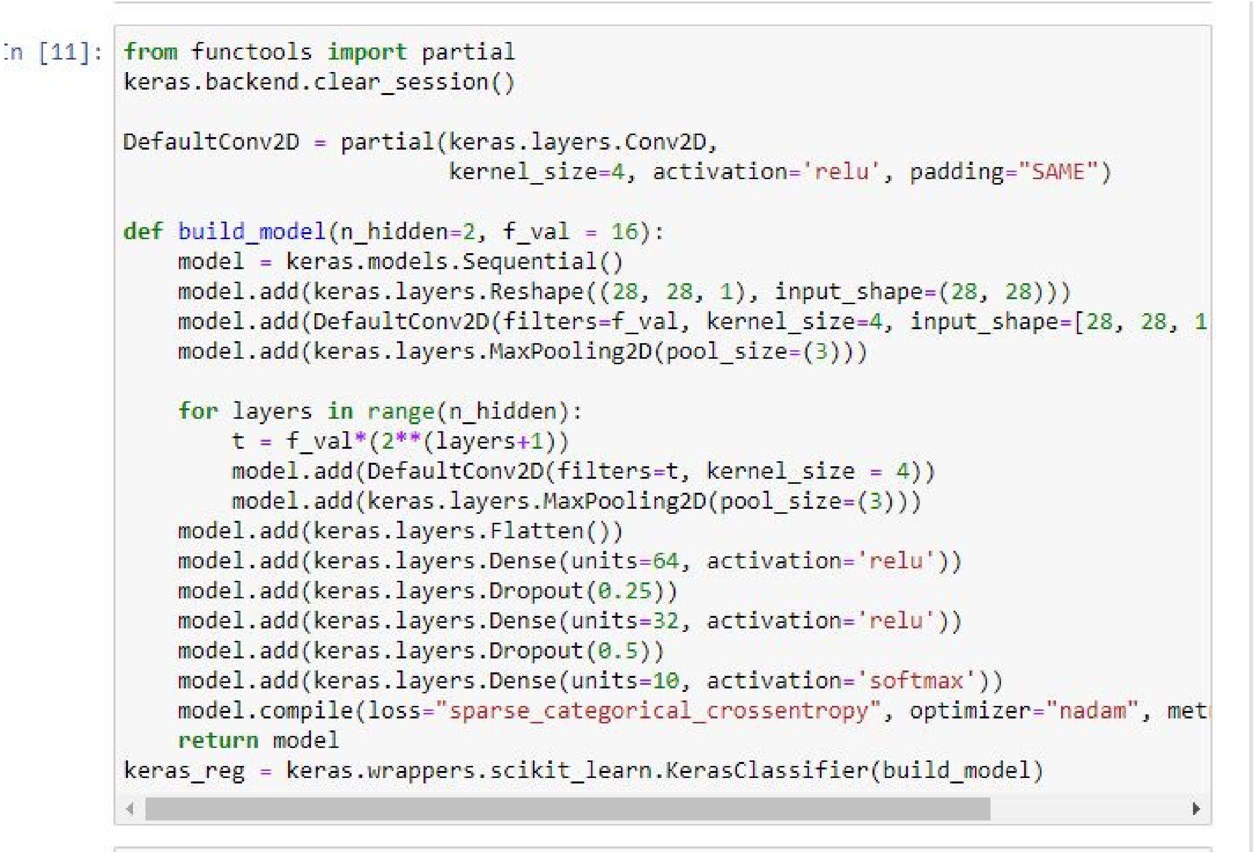


```
In [16]: model.summary()

         Model: "sequential"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #   
         =================================================================
         reshape (Reshape)            (None, 28, 28, 1)         0         
         _________________________________________________________________
         conv2d (Conv2D)              (None, 28, 28, 16)        272       
         _________________________________________________________________
         max_pooling2d (MaxPooling2D) (None, 9, 9, 16)          0         
         _________________________________________________________________
         conv2d_1 (Conv2D)            (None, 9, 9, 32)          8224      
         _________________________________________________________________
         max_pooling2d_1 (MaxPooling2 (None, 4, 4, 32)          0         
         _________________________________________________________________
         conv2d_2 (Conv2D)            (None, 4, 4, 64)          32832     
         _________________________________________________________________
         max_pooling2d_2 (MaxPooling2 (None, 1, 1, 64)          0         
         _________________________________________________________________
         flatten (Flatten)            (None, 64)                0         
         _________________________________________________________________
         dense (Dense)                (None, 64)                4160      
         _________________________________________________________________
         dropout (Dropout)            (None, 64)                0         
         _________________________________________________________________
         dense_1 (Dense)              (None, 32)                2080      
         _________________________________________________________________
         dropout_1 (Dropout)          (None, 32)                0         
         _________________________________________________________________
         dense_2 (Dense)              (None, 10)                330       
         =================================================================
         Total params: 47,898
         Trainable params: 47,898
         Non-trainable params: 0
         _________________________________________________________________
```

Gridsearch Model:

My gridsearch model was not successful. As I stated previously, I noticed a pattern in my convolution layers. I created a build_model. This model was based off of the model I had just created. I wanted to vary the amount of layers I had along with the filter values. You can see my code below.

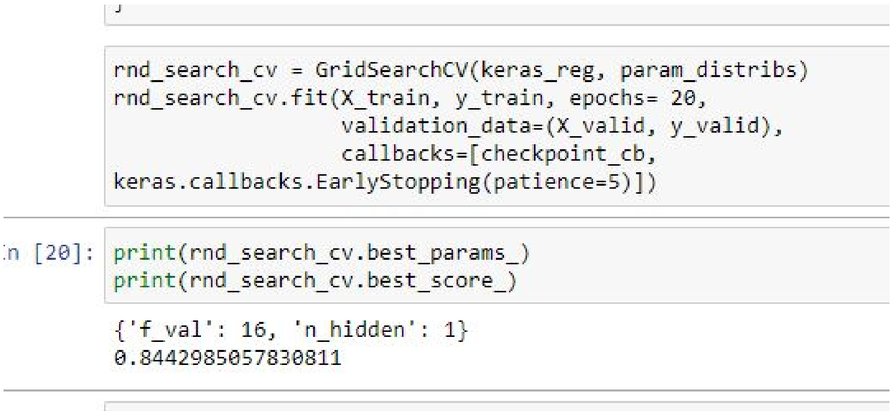
Figure 4: Gridsearch Model



```
In [11]: from functools import partial
         keras.backend.clear_session()

         DefaultConv2D = partial(keras.layers.Conv2D,
                                 kernel_size=4, activation='relu', padding="SAME")

         def build_model(n_hidden=2, f_val = 16):
             model = keras.models.Sequential()
             model.add(keras.layers.Reshape((28, 28, 1), input_shape=(28, 28)))
             model.add(DefaultConv2D(filters=f_val, kernel_size=4, input_shape=[28, 28, 1
             model.add(keras.layers.MaxPooling2D(pool_size=(3)))

             for layers in range(n_hidden):
                 t = f_val*(2**(layers+1))
                 model.add(DefaultConv2D(filters=t, kernel_size = 4))
                 model.add(keras.layers.MaxPooling2D(pool_size=(3)))
             model.add(keras.layers.Flatten())
             model.add(keras.layers.Dense(units=64, activation='relu'))
             model.add(keras.layers.Dropout(0.25))
             model.add(keras.layers.Dense(units=32, activation='relu'))
             model.add(keras.layers.Dropout(0.5))
             model.add(keras.layers.Dense(units=10, activation='softmax'))
             model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", met
             return model
         keras_reg = keras.wrappers.scikit_learn.KerasClassifier(build_model)
```

Upon running my validation data, I had only achieved an accuracy score of 84.44%, which you can see here:

Figure 5: Gridsearch Training Values



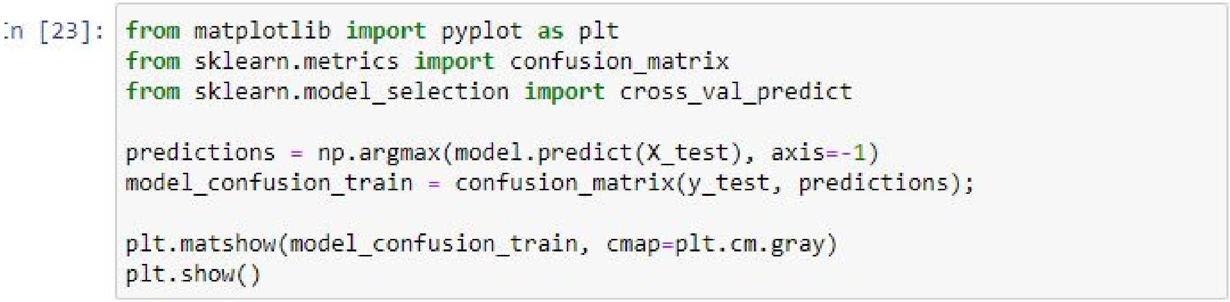
```
rnd_search_cv = GridSearchCV(keras_reg, param_distribs)
rnd_search_cv.fit(X_train, y_train, epochs= 20,
                  validation_data=(X_valid, y_valid),
                  callbacks=[checkpoint_cb,
keras.callbacks.EarlyStopping(patience=5)])
```

```
In [20]: print(rnd_search_cv.best_params_)
         print(rnd_search_cv.best_score_)

         {'f_val': 16, 'n_hidden': 1}
         0.8442985057830811
```

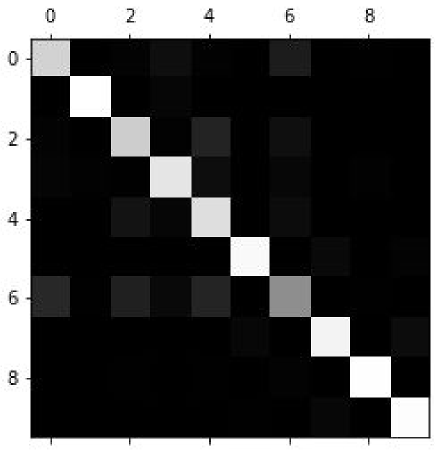
These parameters and validation score is not the same score that I have managed before, leading me to believe that I had made a mistake in my gridsearch. However, since I cannot achieve a better accuracy than my original I have submitted my original model as my final model.

Conclusion:

Figure: 6 : Confusion Matrix of Final Model



```
In [23]: from matplotlib import pyplot as plt
         from sklearn.metrics import confusion_matrix
         from sklearn.model_selection import cross_val_predict

         predictions = np.argmax(model.predict(X_test), axis=-1)
         model_confusion_train = confusion_matrix(y_test, predictions);

         plt.matshow(model_confusion_train, cmap=plt.cm.gray)
         plt.show()
```



I looked at my confusion matrix of my final model (the first one) and honestly it doesn't look too bad. For the most part, we can see the values are consolidated near the diagonal, which shows that my model is pretty accurate. There are some false positives, but again, my model achieves an okay accuracy shown above.