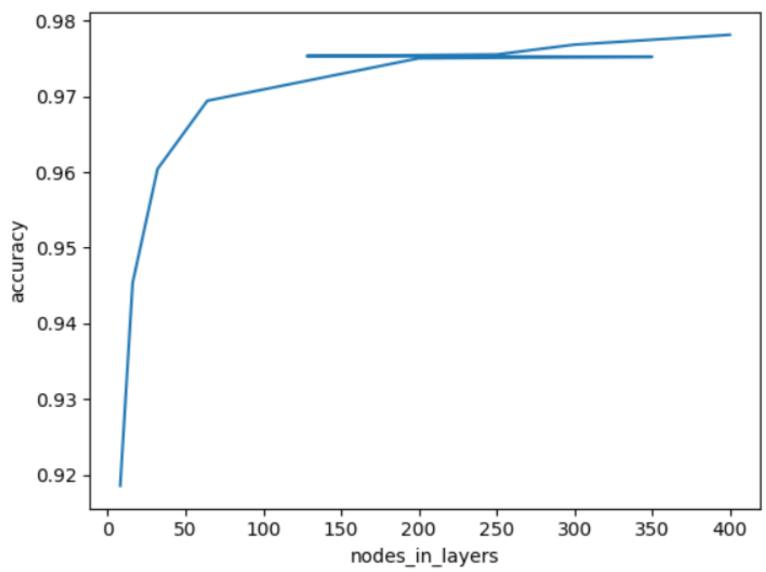
Inf 264
project 2
Andreas Valen

Accuracy and validation
When testing the impact of the different types of hyperparameters, I decided to use regular hold-out validation. I decided this as when testing a range of values for a certain parameter, the graph usually shows a trend that would be true even if the accuracy would increase or decrease with small amounts.

Accuracy is measured in percentages of correct predictions when predicting labels for unseen data. I also used confusion matrices throughout the project to get an overview of the most common errors the classifiers made. As a side note I made a function to manually test the accuracy as the models were suspiciously good at predicting their own accuracy.

Comparing hyper parameters
I initially began by creating similar models with small differences to roughly test the performance of the layers. The first models I created only looked at the keras library's "dense" layer, and the difference in accuracy between number of nodes and number of layers. This became tedious and not very scalable, so I opted to make a function that created models based on given parameters and compared them.

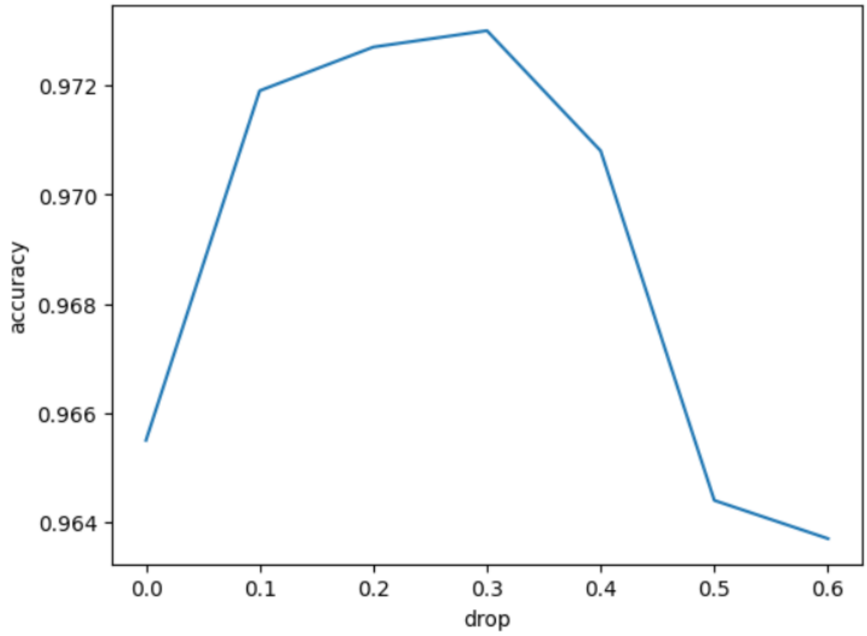
Neurons
I started by comparing the accuracy of similar models where the only difference was the number of neurons in the dense layer. The models had two hidden layers with 8, 16, 32, 64, 128, 200, 250, 300, 350 and 400 neurons each.



It seems accuracy increased alongside the number of neurons in each layer, however, there is a sharp decline in improvement after the neurons in each layer exceeds 128. Factoring in training time I decided to go with 128 neurons per layer.

Dropout

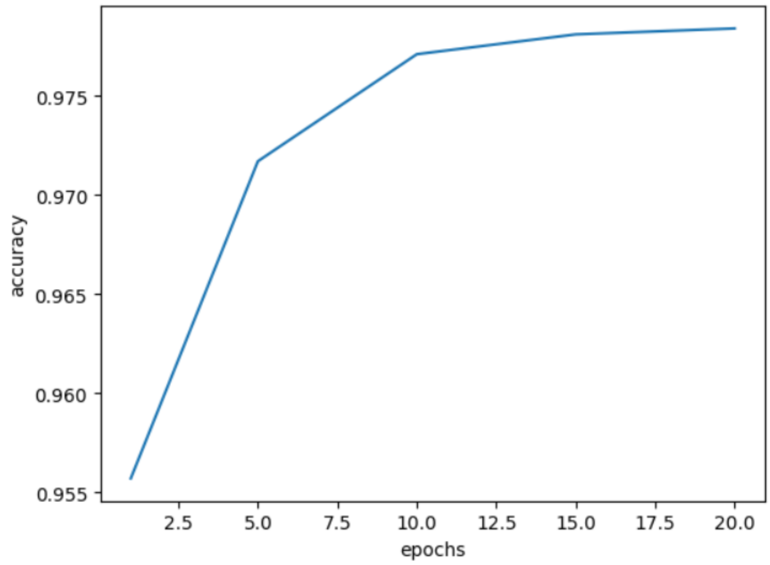
I then tested the best performing model with dropout layers. I tested models with an increasing drop ratio from 0 to 0.6. I only used 5 epochs in the test, so the start accuracy doesn't match the previous model.



Accuracy peaked around the 0.3 mark so I used 0.3 as dropout rate going forward.

Epochs

The epochs variable denotes how many times the training algorithm performs a forward pass and backpropagation on each training data point.



It again seems the accuracy increases with each epoch. Looking at the graph though, it looks like using 10 epochs is a good middle ground for testing. When training the final model, I will use more.

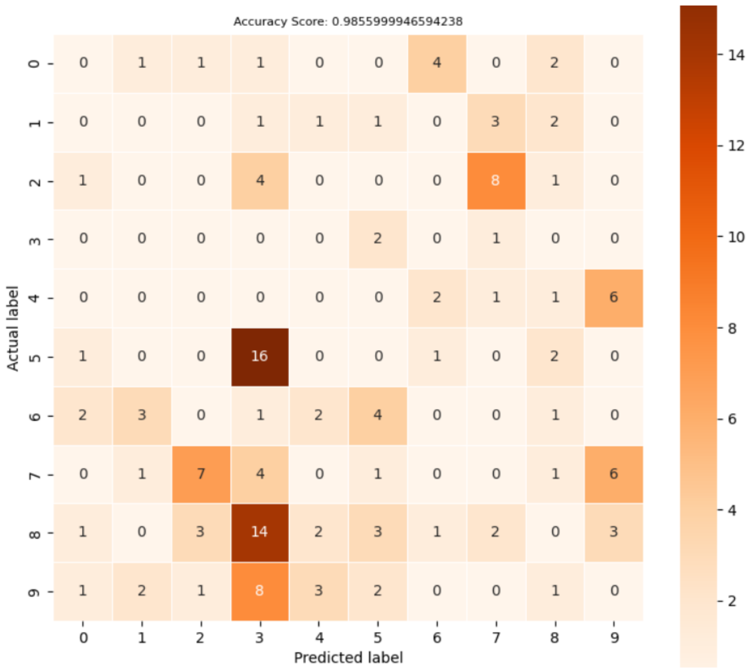
Adding a convolutional layer

I then tested if adding a convolutional layer at the beginning. When playing around it seemed like it improved certain prediction errors. I will test this theory using a modified confusion matrix. If correct, the errors will be more random and less grouped.

```
acc: 0.986, loss: 0.085, time: 1247.96, npl: 128, drop: 0.3, extra_layers: 1.0, convolutional layer: yes,
acc: 0.978, loss: 0.075, time: 54.69, npl: 128, drop: 0.3, extra_layers: 0.0, convolutional layer: no,
```

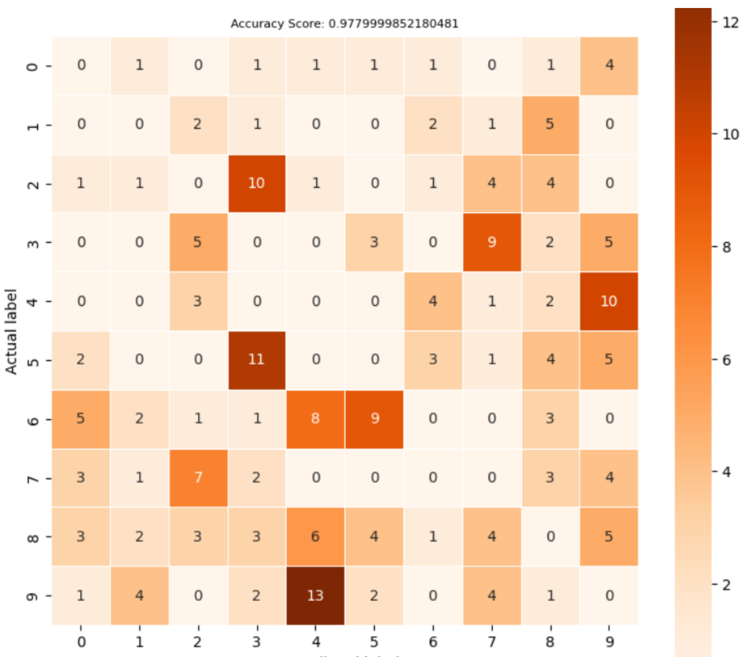
Looking at the data we see accuracy improvement is absolutely fantastic, plus 0.8 percent, with the convolutional layer addition, but it comes at a price. Training time is now a nightmare. Almost 21 minutes for a single classifier.

Let’s see if I guessed correctly…
Here is the confusion matrix for the model using convolutional layers:



This is a modified confusion matrix because I have removed the correct predictions to better highlight the mistakes. We clearly see the model favors the number 3 a bit too much.

Now looking at the cm for the model that doesn’t use those layers we clearly see there are more errors, however one can argue they seem more random and spread out. Looks like my hypothesis was not only wrong, but the opposite of what was true.



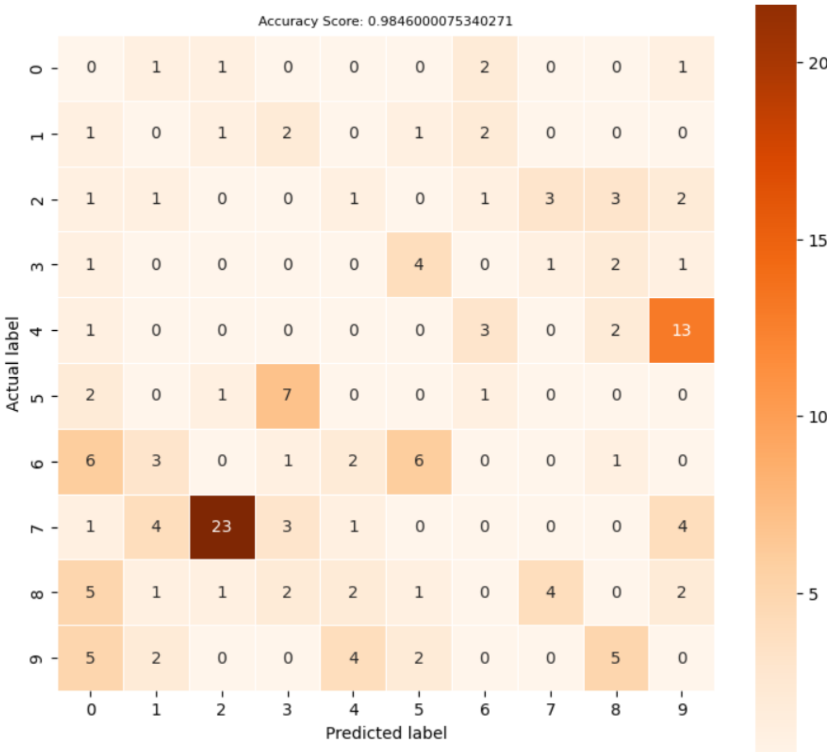
Testing a high node and epochs values classifier
Using the values I had discovered I made trained a new classifier with:

- nodes=250
- drop=0.3
- epochs=50
- extra convolutional layer

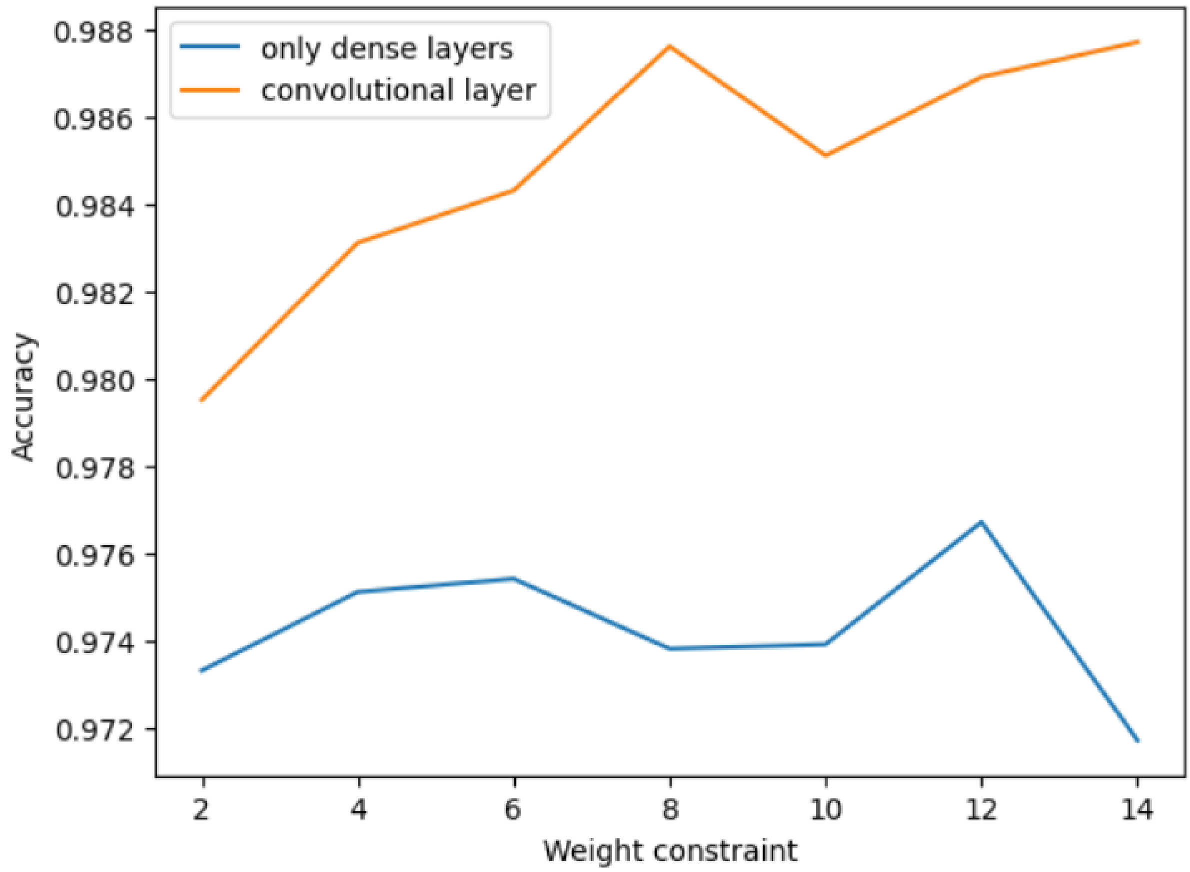
and trained it overnight.

```
acc: 0.985, loss: 0.335, time: 29169.26, npl: 250, drop: 0.3, extra_layers: 1.0, convolutional layer: yes,
```

Needless to say I was disappointed with the result. It trained for 8.1 hours and had the same accuracy as the previous model. The high loss suggests overfitting, which might be because of the number of epochs. It seems the golden number might lie somewhere between 5 and 10.



Weight constraints
I went back a few steps and tried adding weight constraints. It took over an hour to test and it looks like the less constraints, the more accurate my models become. This might be due to the placement of the constraints and the number of layers with constraints, but testing all possible combinations is not feasible due to time constraints.
Graph of accuracy of the models with respect to weight_constraints:



Additional parameters
There are a number of other hyperparameters that might improve accuracy. Notably batch size, activation function, loss function and optimizing function. I have decided to not test them as each new model now takes 20 minutes to train. The increase in accuracy would also be so incremental that cross validation would be in order. With k=4 that gives approximately 1 hour of train time per model. Testing between 'relu' and 'leaky_relu' would then be at least a multiple hour ordeal.

Training the final model
Based on what I found I tested these values:

- nodes=128
- drop=0.25
- epochs=7
- extra convolutional layer

The resulting classifier had an accuracy of 98.89% when predicting the labels of the unseen test data.
I really wanted to get 99% accuracy, so i tried some minor tweaks an ran it again with these values:

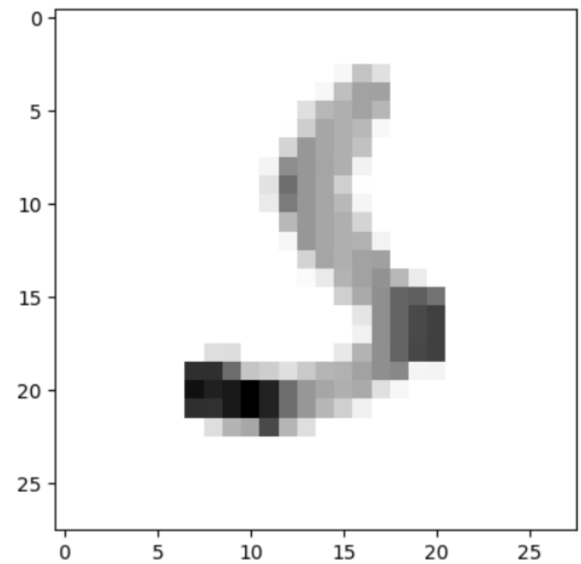
- nodes=128
- drop=0.27
- epochs=6
- extra convolutional layer

The new classifier had a prediction accuracy of 98.90%. This is the classifier I am turning in for this project. To reproduce the result run the first block in the ‘final_v2’ file.

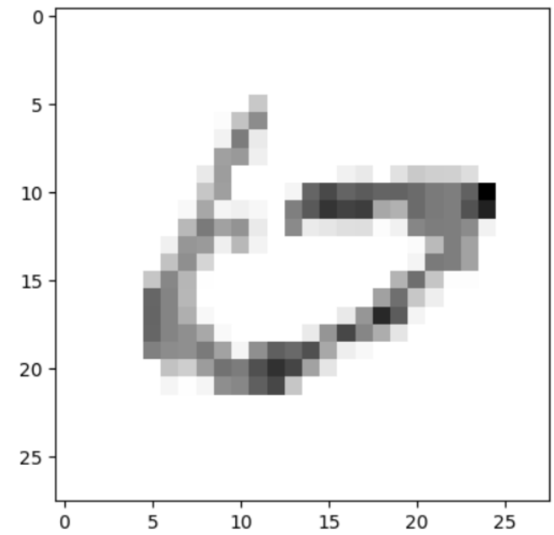
Evaluation

98.90% is good, but it doesn’t tell us where it goes wrong. I wrote a function that finds the errors and prints them out. Let’s look at a few examples of where the classifier fails:

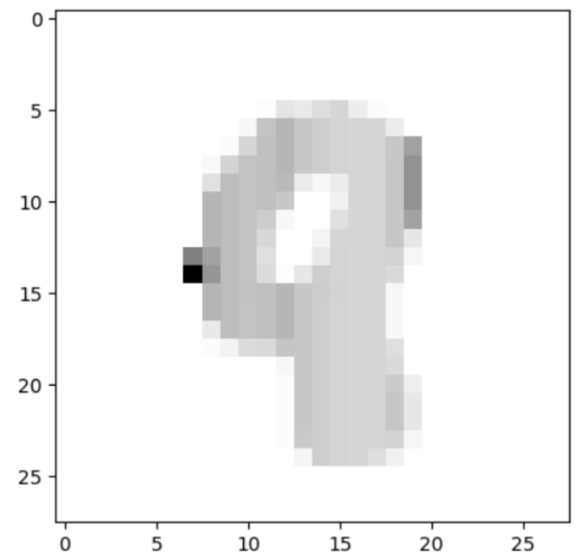
Predicted: 3 - Actual: 5



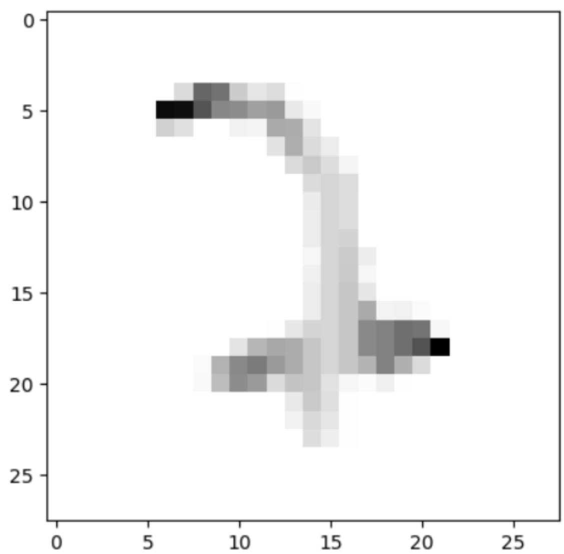
Predicted: 0 - Actual: 6



Predicted: 9 - Actual: 8



Predicted: 2 - Actual: 7



Run block 2 in the ‘final_v2’ file and it prints all of the numbers it got wrong. After looking at the images printed out it’s clear that the performance of the classifier is sufficient. Appart from a couple few examples, it only messes up on numbers that even humans would struggle with. I for one would, along with the classifier, say the bottom left picture above is a “nine” and not an “eight”.

Documentation / Code showcase

plotCorrelationGraph

```
def plotCorrelationGraph(modelDataList, x="nodes_in_layers", y="accuracy"):
    x_arr = []
    y_arr = []
    for modelData in modelDataList:
        x_arr.append(modelData[x])
        y_arr.append(modelData[y])
    plt.ylabel(y)
    plt.xlabel(x)
    plt.plot(x_arr, y_arr)
    plt.show()
```

The parameter value had to be extracted from the resulting list of dictionaries. I made this function to quickly plot the correlation between two parameters. Note that the modelDataList has to be sorted based on x value before being passed to the plotting function.

cleanPredictions

```
def cleanPredictions(predictions):
    _predictions = []
    for p in predictions:
        _predictions.append(np.argmax(p))
    return _predictions
```

The model.predict function returns a list of arrays of the last layer. Argmax finds the label that the classifier deemed most likely to be correct. This function turns the 2d array into a list of the actual predicted labels.

```python
def trainAndEvaluateModel(x_train, y_train, x_test, y_test, use_cnn=0,
                          nodes_in_layers=32,
                          epochs=10,
                          drop=0,
                          add_dense_layer=0,
                          weight_constraint=0):
    model = tf.keras.models.Sequential()
    if use_cnn==1:
        model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
                                         input_shape=(28, 28, 1)))
        model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
        model.add(tf.keras.layers.Dropout(drop, seed=3))
        model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
    else:
        model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
        model.add(tf.keras.layers.Dense(units=nodes_in_layers, activation=tf.nn.relu,))
        model.add(tf.keras.layers.Dropout(drop, seed=1))
    model.add(tf.keras.layers.Dense(units=nodes_in_layers, activation=tf.nn.relu,
                                    kernel_constraint= tf.keras.constraints.max_norm(weight_constraint)))
    if add_dense_layer==1:
        model.add(tf.keras.layers.Dense(units=nodes_in_layers, activation=tf.nn.relu))
    model.add(tf.keras.layers.Dropout(drop, seed=2))
    model.add(tf.keras.layers.Dense(units=10, activation=tf.nn.softmax))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    start = time.time()

    model.fit(x_train, y_train, epochs=epochs, verbose=1)
    end = time.time()
    elapsed_time = end - start
```

This is the function I made to test the different hyper parameters. I made a mapping function and fed it an array of different combinations to test which it then trained and tested. This made it easy to compare different results.

Example of testing array and printing the wanted values:

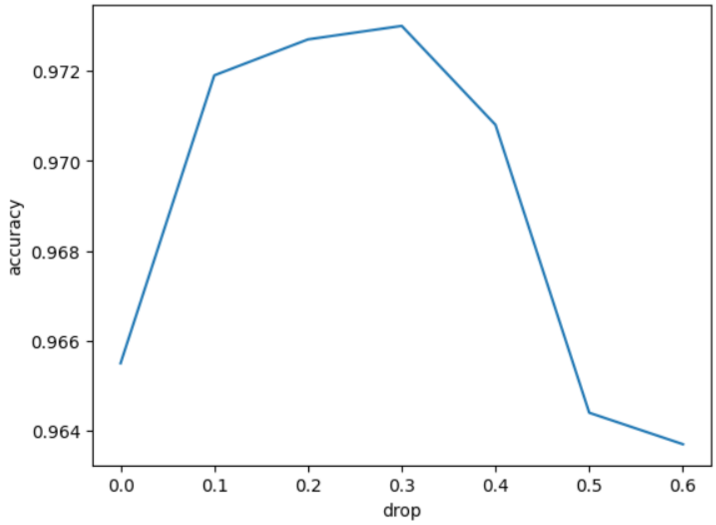
```python
####################################
## This block is testing dropout ##
####################################
nodes=128
epochs=5

#[use_cnn = 0 or 1, nodes_in_layers, epochs, drop = float between 0 and 1, add_dense_layer = 0 or 1]
d_parameters_to_test = np.array([[0, nodes, epochs, 0, 0],
                                 [0, nodes, epochs, 0.1, 0],
                                 [0, nodes, epochs, 0.2, 0],
                                 [0, nodes, epochs, 0.3, 0],
                                 [0, nodes, epochs, 0.4, 0],
                                 [0, nodes, epochs, 0.5, 0],
                                 [0, nodes, epochs, 0.6, 0],
                                ])
#Testing the significance of the number of neurons and if an extra layer adds to the accuracy
d_tested_models = mapThen_trainAndEvaluateModel(x_train, y_train, x_test, y_test, d_parameters_to_test)

#Printing models from best to worst
for modelData in d_tested_models:
    printChosenParams(modelData, use_cnn=False, epochs=False, time=False)
```

Result (removed the keras printout):

```
acc: 0.973, loss: 0.091, npl: 128, drop: 0.3, extra_layers: 0.0,
acc: 0.973, loss: 0.091, npl: 128, drop: 0.2, extra_layers: 0.0,
acc: 0.972, loss: 0.092, npl: 128, drop: 0.1, extra_layers: 0.0,
acc: 0.971, loss: 0.101, npl: 128, drop: 0.4, extra_layers: 0.0,
acc: 0.965, loss: 0.115, npl: 128, drop: 0.0, extra_layers: 0.0,
acc: 0.964, loss: 0.113, npl: 128, drop: 0.5, extra_layers: 0.0,
acc: 0.964, loss: 0.125, npl: 128, drop: 0.6, extra_layers: 0.0,
```

Sorting and printing out the result:

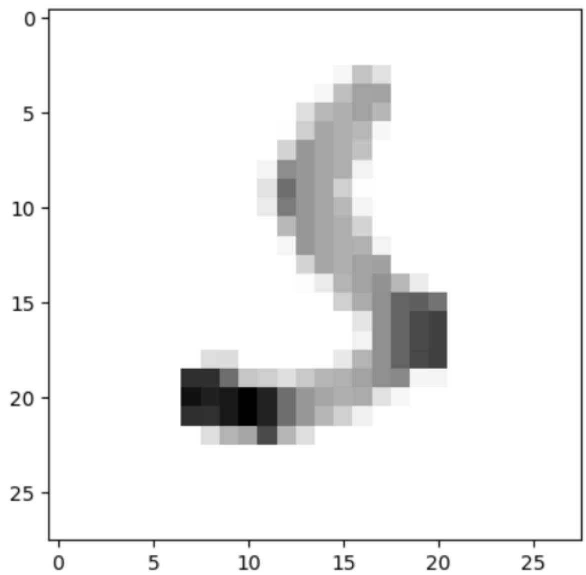
```
[106]: sortedList = sorted(d_tested_models, key=lambda d: d['drop'], reverse=False)
       plotCorrelationGraph(sortedList, x="drop", y="accuracy")
```



Locating the wrongly predicted images:

```
predictions = cleanPredictions(model.predict(x_test))

for i in range(0, len(y_test)-1):
    if predictions[i]!=y_test[i]:
        print("Predicted: {} - Actual: {}".format(predictions[i], y_test[i]))
        plt.imshow(x_test[i], cmap='Greys')
        plt.show()
```

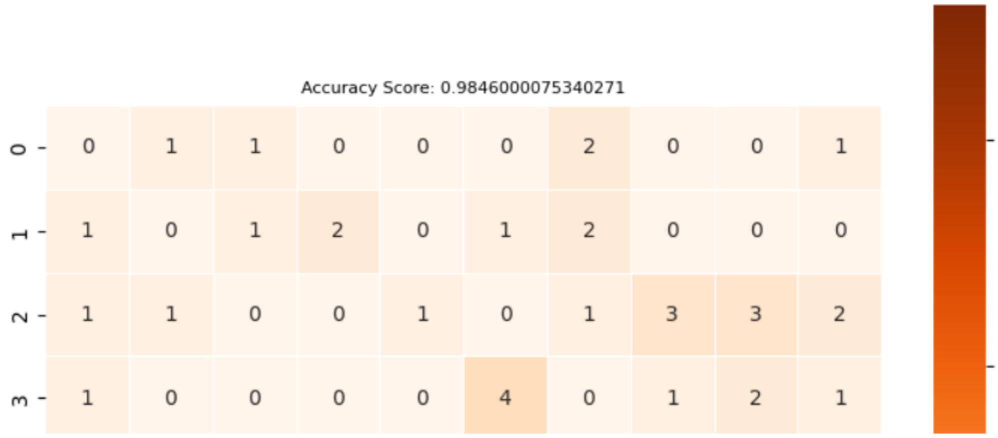
```
313/313 [==============================] - 3s 10ms/step
Predicted: 3 - Actual: 5
```



This is how I located the images of the numbers that failed.

Here is the code i used to print out confusion matrices:

```
_modelData = c_tested_models[0]

predictions = _modelData['predictions']
cm = metrics.confusion_matrix(y_test, predictions)
error_cm = onlyShowErrors(cm)

plt.figure(figsize=(9,9))
sns.heatmap(error_cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap = 'Oranges');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(_modelData['accuracy'])
plt.title(all_sample_title, size = 8);
```



Note that I made a function that removes the correct answers to easier see where the errors are.

```
def onlyShowErrors(_cm):
    cm = _cm
    for i in range(0,10):
        cm[i][i]=0
    return cm
```