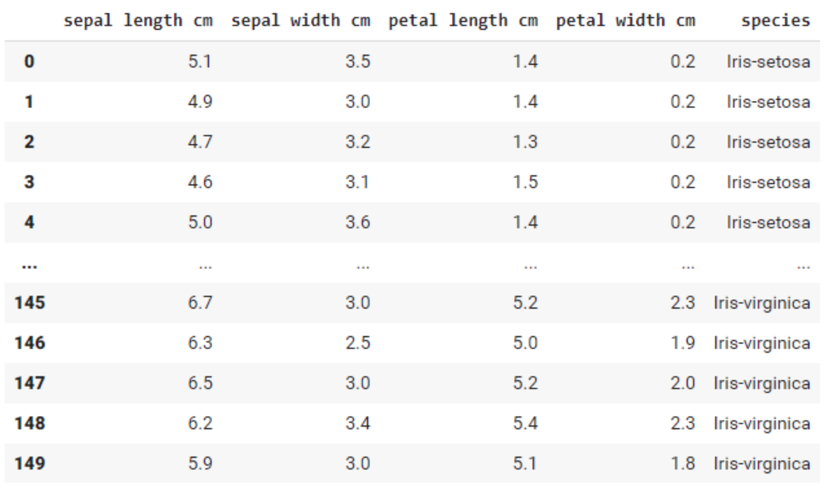
MultiLayer Perceptron

Kelly Ryan 0347345
Surya Narayanan Thottathil Ravindranathan 18104452

1. Iris Dataset

1.1 Introduction

We chose to use the Iris dataset which we obtained from Kaggle. This dataset was used by R A Fisher in his 1936 paper "*The use of multiple measurements in taxonomic problem*". It is a multivariate dataset containing 50 samples from each of 3 different species of iris (150 total). Each sample has 4 different attributes being measurements in centimetres of the width and length of the sepals and petals of each sample.

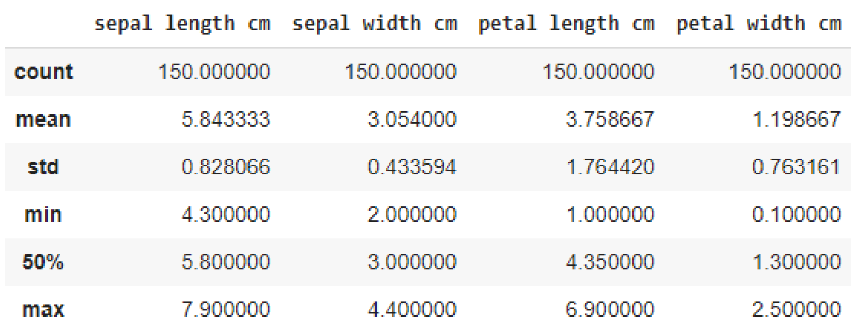


	sepal length cm	sepal width cm	petal length cm	petal width cm	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

Figure 1 - Iris Dataset

1.2 Data Exploration and Visualisation

The pandas describe() function returns statistics for the dataset including count, mean values, standard deviation, median and min and max values for each of the numerical columns. The largest max value will be important for the normalization step.



	sepal length cm	sepal width cm	petal length cm	petal width cm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
50%	5.800000	3.000000	4.350000	1.300000
max	7.900000	4.400000	6.900000	2.500000

Figure 2 - Statistical Information for Iris Dataset

Figure 3 below plots the petal length according to each species and from this we can infer that while iris setosa is linearly separable from iris versicolor and iris virginica using the petal length attribute, the latter two are not linearly separable from each other.

Figure 4 confirms the same level of linear separability as Figure 1 and although it looks as though there may be some overlap in petal widths of iris setosa and iris versicolor as the lines appear to join we have confirmed using a histogram (see Jupyter Notebook file) that there is not.

Figures 5 and 6 show overlap in both sepal width and sepal length and are therefore not linearly separable using these attributes.

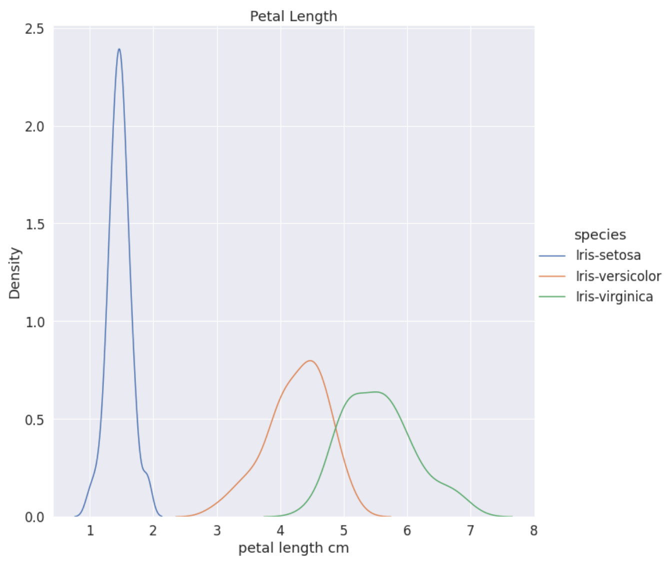


Figure 3 - Petal Length by Species

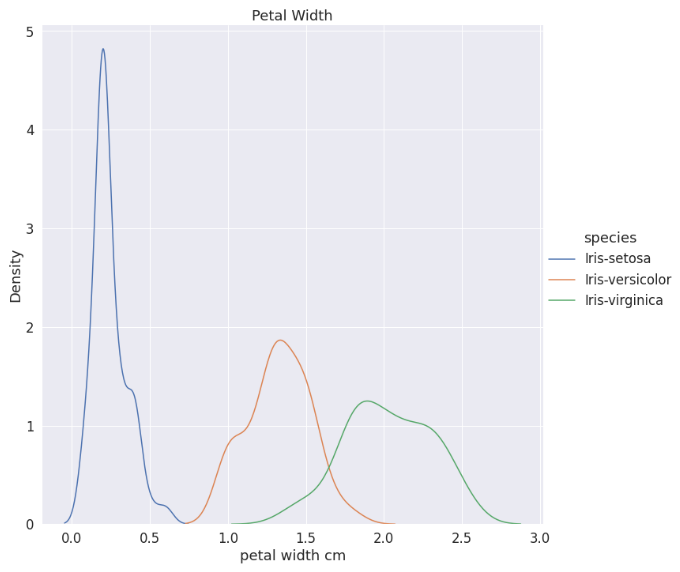


Figure 4 - Petal Width by Species

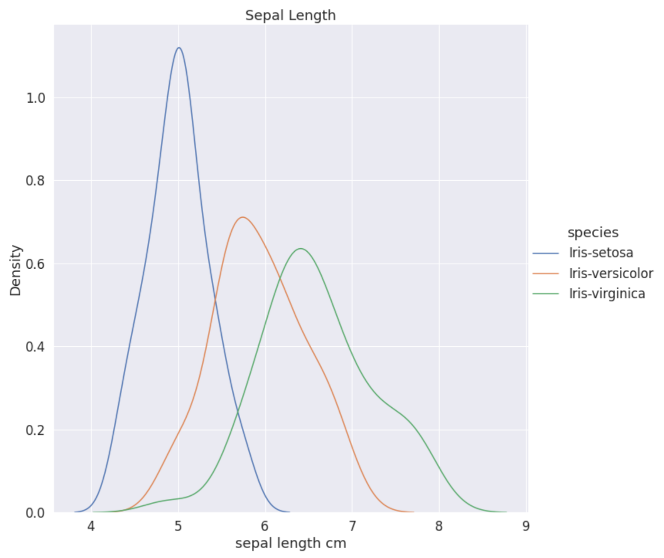


Figure 5 - Sepal Length by Species

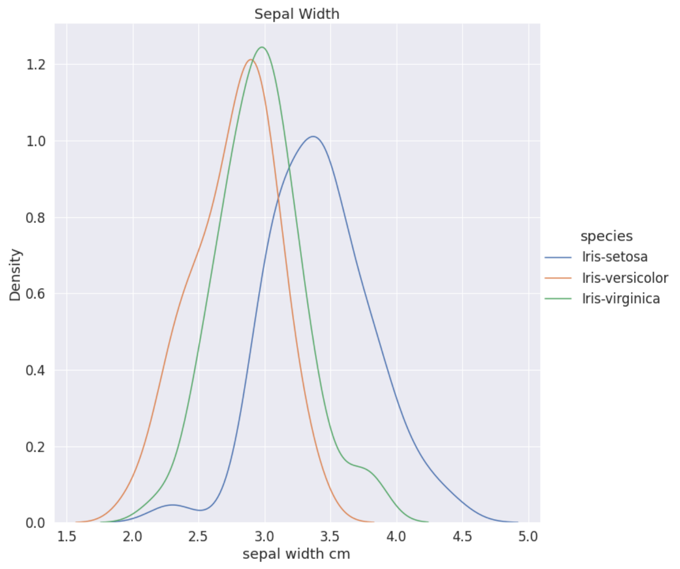


Figure 6 - Sepal Width by Species

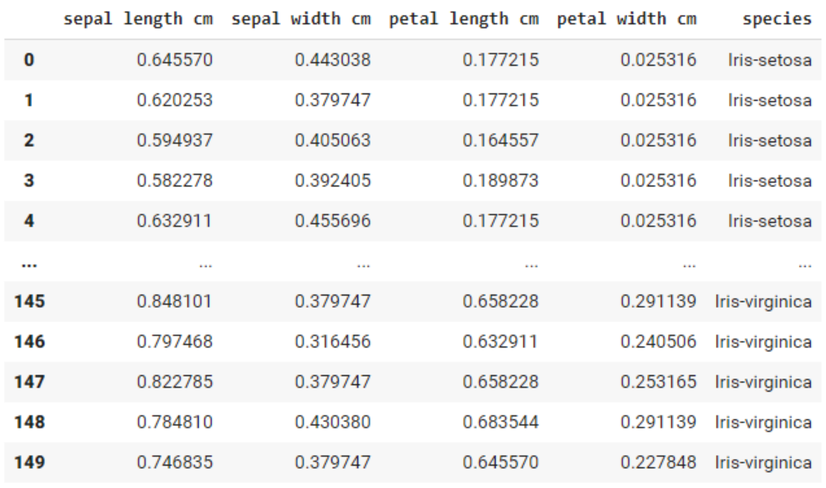
2. Pre-Processing

2.1 Normalisation

We normalised the data in the numerical columns using the following technique:

1. Output the max value in each column to a series called col_maxes using the max() function.
2. Store the overall max value from col_maxes in a variable called iris_num_max.
3. Divide the values in the numerical columns by this value so that all values are between 0 and 1.
4. Reassign the normalized values into a new dataframe called iris_normalized.

Looking at the normalized data in Figure 7 we can see that all numerical values have been normalized to float values between 0 and 1.

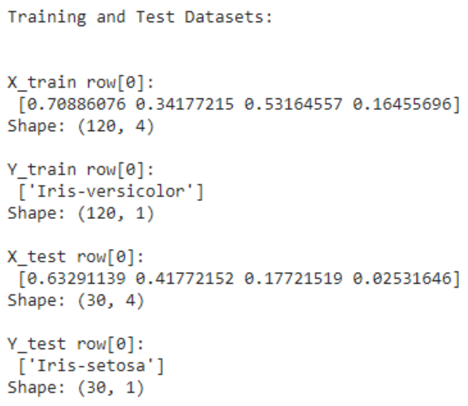


	sepal length cm	sepal width cm	petal length cm	petal width cm	species
0	0.645570	0.443038	0.177215	0.025316	Iris-setosa
1	0.620253	0.379747	0.177215	0.025316	Iris-setosa
2	0.594937	0.405063	0.164557	0.025316	Iris-setosa
3	0.582278	0.392405	0.189873	0.025316	Iris-setosa
4	0.632911	0.455696	0.177215	0.025316	Iris-setosa
...	...	...	...	...	...
145	0.848101	0.379747	0.658228	0.291139	Iris-virginica
146	0.797468	0.316456	0.632911	0.240506	Iris-virginica
147	0.822785	0.379747	0.658228	0.253165	Iris-virginica
148	0.784810	0.430380	0.683544	0.291139	Iris-virginica
149	0.746835	0.379747	0.645570	0.227848	Iris-virginica

Figure 7 - Normalized Dataset

2.2 Testing and Training Datasets

We split the dataset at a ratio of 8:2 for the training and test datasets, respectively. Additionally, 20% of the training set was set aside for validation. The training and test input/target values are stored in numpy arrays X_train and Y_train and X_test and Y_test.



```
Training and Test Datasets:


X_train row[0]:
 [0.70886076 0.34177215 0.53164557 0.16455696]
Shape: (120, 4)

Y_train row[0]:
 ['Iris-versicolor']
Shape: (120, 1)

X_test row[0]:
 [0.63291139 0.41772152 0.17721519 0.02531646]
Shape: (30, 4)

Y_test row[0]:
 ['Iris-setosa']
Shape: (30, 1)
```

Figure 8 - Training and Test Sets

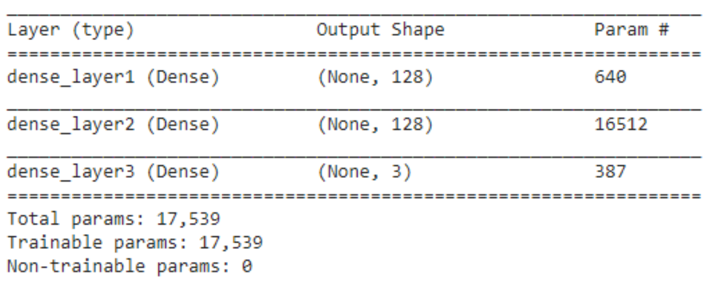
2.3 One-Hot Encoding

The species target values are of type object and therefore need to be encoded. Label encoding would not be appropriate here as numbers applied to the labels would be applied alphabetically (e.g. 0 => Iris-setosa, 1 => Iris-versicolor, 2 => Iris-virginica) and the model may infer a relationship between the species values where there is none.

Instead, we will use one-hot encoding where each species/category is represented as a one-hot vector. This removes any issue of ranking as may be seen with label encoding as each category is represented by a binary vector instead of an integer.

before encoding	**After label encoding**	**After one-hot encoding**
['Iris-setosa']	0	[1. 0. 0.]
['Iris-versicolor']	1	[0. 1. 0.]
['Iris-virginica']	2	[0. 0. 1.]

3. Network Structure



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_layer1 (Dense)         (None, 128)               640       
_________________________________________________________________
dense_layer2 (Dense)         (None, 128)               16512     
_________________________________________________________________
dense_layer3 (Dense)         (None, 3)                 387       
=================================================================
Total params: 17,539
Trainable params: 17,539
Non-trainable params: 0
```

Figure 9 - Network Structure

The network structure consists of one input layer, one hidden layer and one output layer. The input and hidden layer each have 128 neurons and the output layer has 3 neurons corresponding to the 3 species categories. Each layer has an input shape of 4 given that there are 4 attributes in each training, test and validation record. Given the small size of the Iris Dataset 3 layers seemed sufficient in this case.

The activation function used in the input and hidden layers of the network is the Rectified Linear Activation Unit (ReLU). This has become the default activation used in training most neural networks and it solves the vanishing gradient issue that can occur with the hyperbolic tangent and sigmoid functions in networks where there are many layers. We note that our network does not have many layers but agree that this is the best activation function to use in this instance.

The output layer uses the softmax activation function which converts a vector of values to a probability distribution, i.e. how likely it is for an iris record to belong to a particular classification given the inputs supplied.

The model also utilises a batch size of 2 and will make 200 epochs through the training data.

4. Loss Function

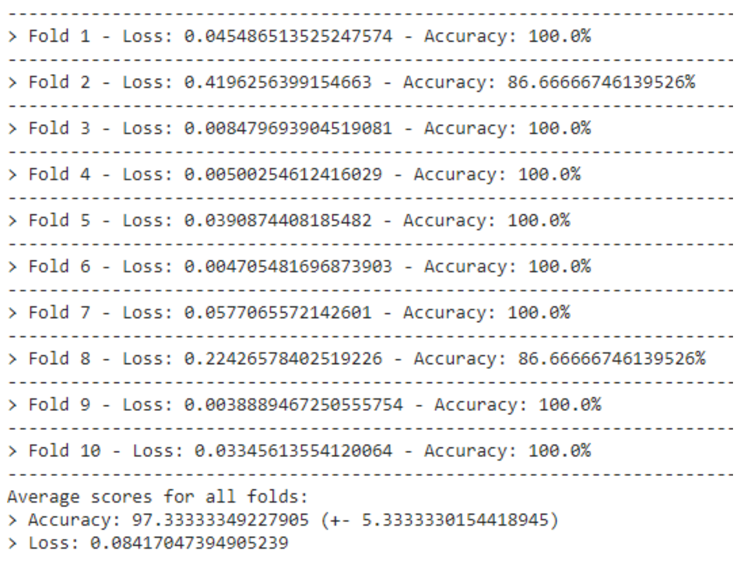
We opted to use categorical crossentropy as the loss function in our model to calculate the loss between labels and predictions. It is appropriate here as it should be used when there are 2 or more label classes (we have 3). Additionally, it expects labels to be provided in a one-hot representation which we took care of in an earlier step.

5. Optimizer

We chose Adam as the optimiser for our model as it performed best in our experiments when compared with Stochastic Gradient Descent as illustrated in Figure 18 on page 9.

6. K-Fold Cross Validation

We opted to use 10-fold cross validation as k=10 has been found to give a good balance of low computational expense and low bias in an estimate of a performance model. We recombined the training and test sets for this section and applied it to our 3-layer model.



```
------------------------------------------------------------------------
> Fold 1 - Loss: 0.045486513525247574 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 2 - Loss: 0.4196256399154663 - Accuracy: 86.66666746139526%
------------------------------------------------------------------------
> Fold 3 - Loss: 0.008479693904519081 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 4 - Loss: 0.00500254612416029 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 5 - Loss: 0.0390874408185482 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 6 - Loss: 0.004705481696873903 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 7 - Loss: 0.0577065572142601 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 8 - Loss: 0.22426578402519226 - Accuracy: 86.66666746139526%
------------------------------------------------------------------------
> Fold 9 - Loss: 0.0038889467250555754 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 10 - Loss: 0.03345613554120064 - Accuracy: 100.0%
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 97.33333349227905 (+- 5.3333330154418945)
> Loss: 0.08417047394905239
```

Figure 10 – 10-Fold Cross Validation

As can be seen from Figure 10 above, our model performed well with an average accuracy of 97.33 and it is the model we chose to use for our final build.

7. Results

For this model we chose to use the following hyperparameters:

Optimiser: Adam
Loss function: Categorical Crossentropy
Metric: Categorical Accuracy

Batch Size: 2
Epochs: 200

Using these hyperparameters we trained our network and obtained the results illustrated in the figures below:

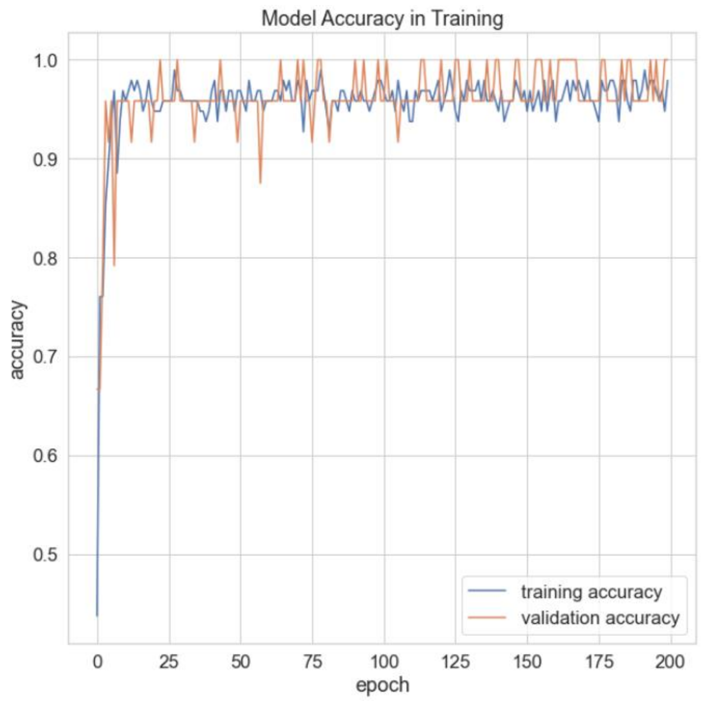


Figure 11 - Accuracy in Training and Validation

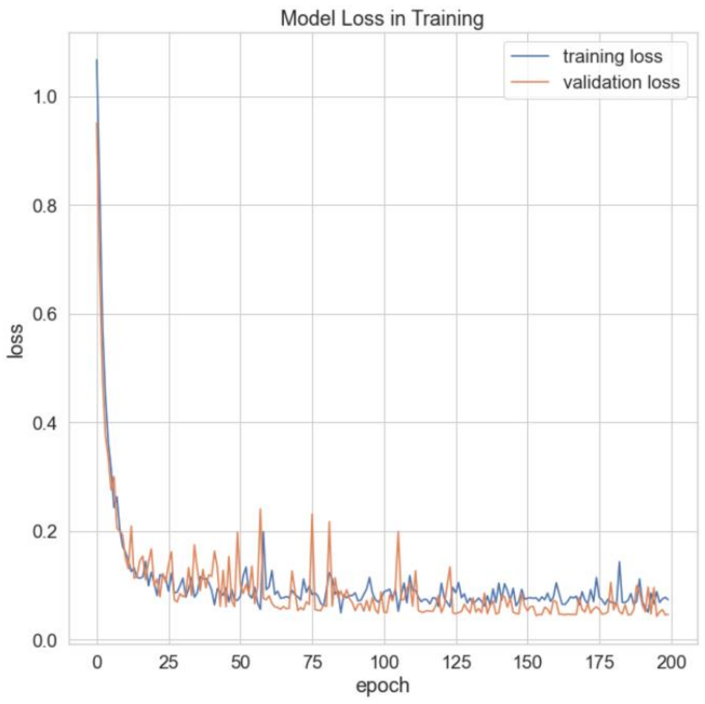
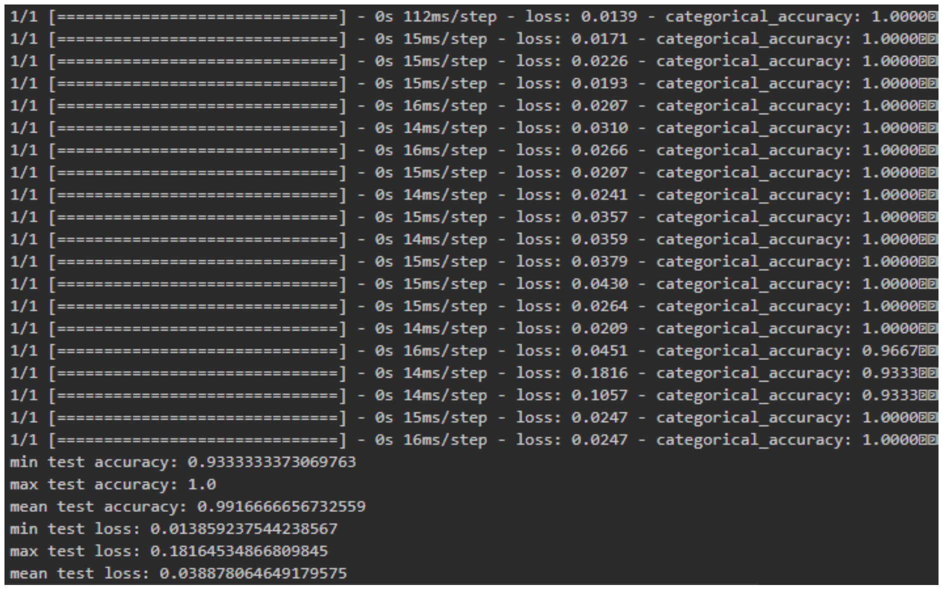


Figure 12 - Loss in Training and Validation

After each cycle of training, validation and testing our model consistently produced an accuracy value of between 0.9333 and 1. To verify this I used a loop to run the training and evaluation functions 20 times and noted the results:



```
1/1 [==============================] - 0s 112ms/step - loss: 0.0139 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 15ms/step - loss: 0.0171 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 15ms/step - loss: 0.0226 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 15ms/step - loss: 0.0193 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 16ms/step - loss: 0.0207 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 14ms/step - loss: 0.0310 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 16ms/step - loss: 0.0266 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 15ms/step - loss: 0.0207 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 14ms/step - loss: 0.0241 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 15ms/step - loss: 0.0357 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 14ms/step - loss: 0.0359 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 15ms/step - loss: 0.0379 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 15ms/step - loss: 0.0430 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 15ms/step - loss: 0.0264 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 14ms/step - loss: 0.0209 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 16ms/step - loss: 0.0451 - categorical_accuracy: 0.9667
1/1 [==============================] - 0s 14ms/step - loss: 0.1816 - categorical_accuracy: 0.9333
1/1 [==============================] - 0s 14ms/step - loss: 0.1057 - categorical_accuracy: 0.9333
1/1 [==============================] - 0s 15ms/step - loss: 0.0247 - categorical_accuracy: 1.0000
1/1 [==============================] - 0s 16ms/step - loss: 0.0247 - categorical_accuracy: 1.0000
min test accuracy: 0.9333333373069763
max test accuracy: 1.0
mean test accuracy: 0.9916666656732559
min test loss: 0.013859237544238567
max test loss: 0.18164534866809845
mean test loss: 0.038878064649179575
```

Figure 13 - Output of 20 Training and Test Cycles

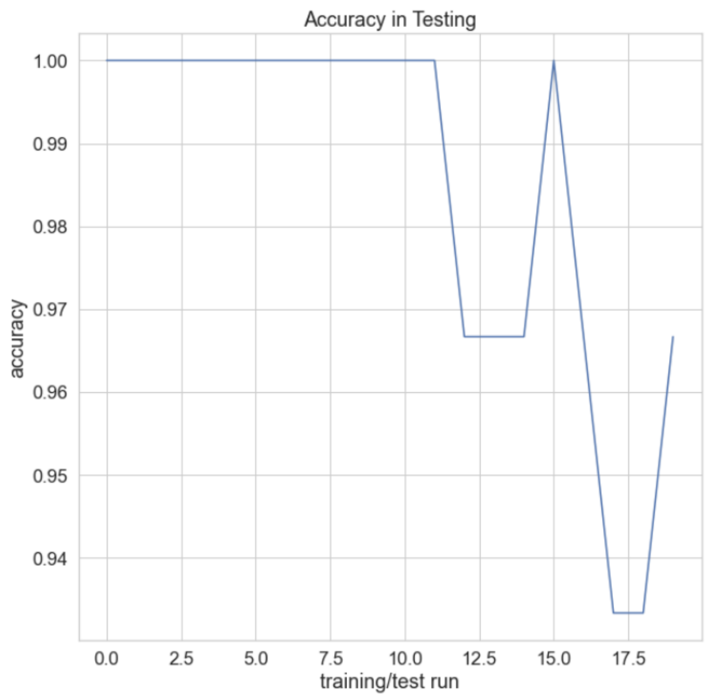


Figure 14 – Categorical Accuracy measured across 20 Training/Testing Cycles

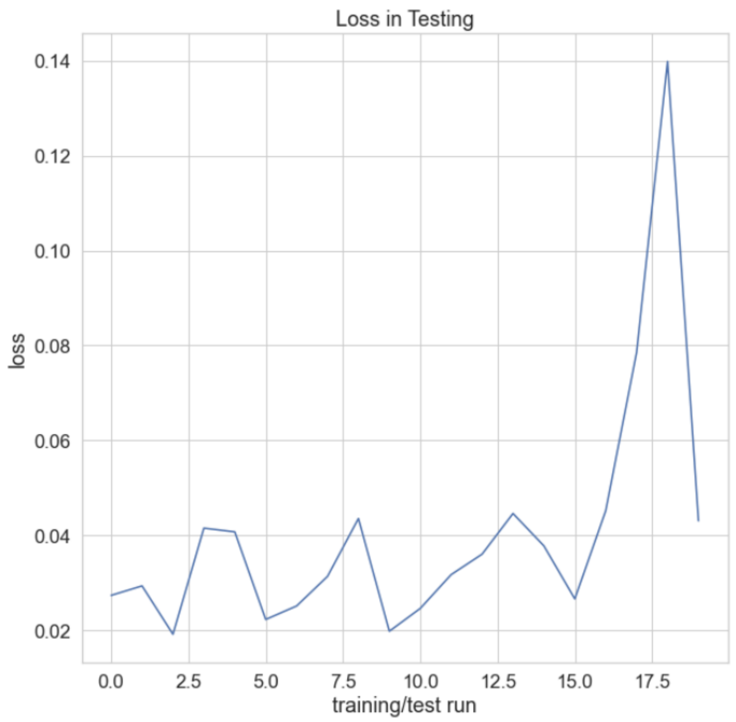
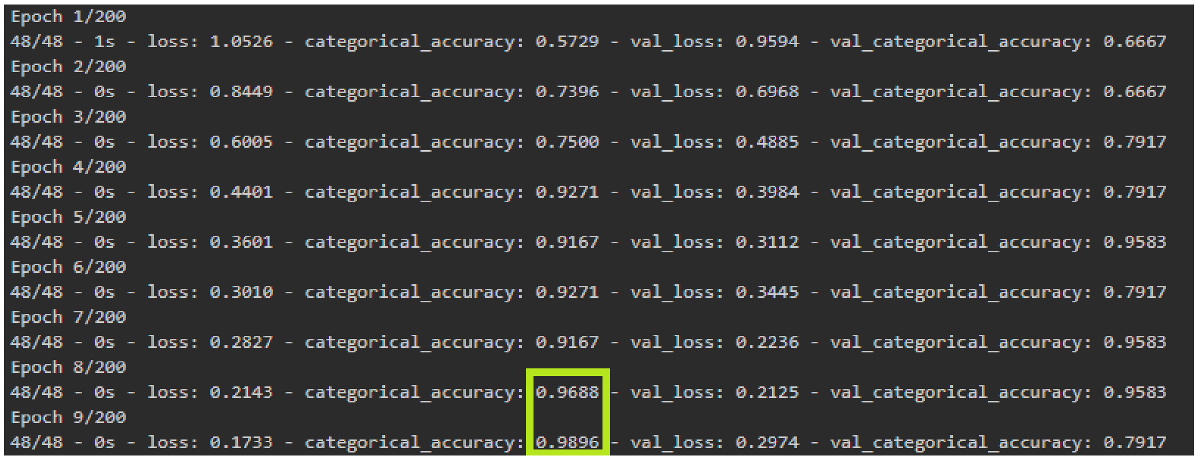


Figure 15 - Loss measured across 20 Training/Testing Cycle

8. Evaluation of the Results

When we began this project our research had indicated that the Adam optimiser, categorical cross entropy loss function and categorial accuracy would be the best options for our model given the dataset we were using and the type of classification being done. These options have given us very good results as set out above.

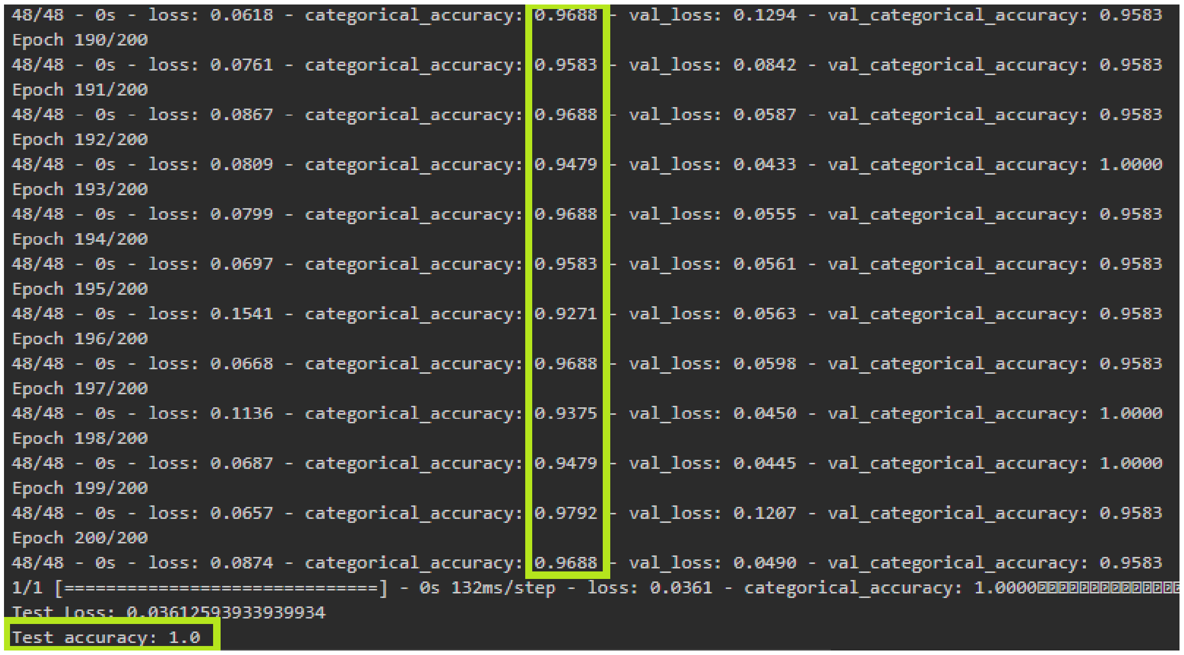
No underfitting was observed with this model. It performed well on the test data and reached a high level of accuracy as early as the eighth epoch as seen in Figure 16.



```
Epoch 1/200
48/48 - 1s - loss: 1.0526 - categorical_accuracy: 0.5729 - val_loss: 0.9594 - val_categorical_accuracy: 0.6667
Epoch 2/200
48/48 - 0s - loss: 0.8449 - categorical_accuracy: 0.7396 - val_loss: 0.6968 - val_categorical_accuracy: 0.6667
Epoch 3/200
48/48 - 0s - loss: 0.6005 - categorical_accuracy: 0.7500 - val_loss: 0.4885 - val_categorical_accuracy: 0.7917
Epoch 4/200
48/48 - 0s - loss: 0.4401 - categorical_accuracy: 0.9271 - val_loss: 0.3984 - val_categorical_accuracy: 0.7917
Epoch 5/200
48/48 - 0s - loss: 0.3601 - categorical_accuracy: 0.9167 - val_loss: 0.3112 - val_categorical_accuracy: 0.9583
Epoch 6/200
48/48 - 0s - loss: 0.3010 - categorical_accuracy: 0.9271 - val_loss: 0.3445 - val_categorical_accuracy: 0.7917
Epoch 7/200
48/48 - 0s - loss: 0.2827 - categorical_accuracy: 0.9167 - val_loss: 0.2236 - val_categorical_accuracy: 0.9583
Epoch 8/200
48/48 - 0s - loss: 0.2143 - categorical_accuracy: 0.9688 - val_loss: 0.2125 - val_categorical_accuracy: 0.9583
Epoch 9/200
48/48 - 0s - loss: 0.1733 - categorical_accuracy: 0.9896 - val_loss: 0.2974 - val_categorical_accuracy: 0.7917
```

Figure 16 - No underfitting detected

Additionally, no overfitting was detected either as is confirmed by Figure 17 below which shows an accuracy of 0.9688 on the training data after 200 epochs and an accuracy of 1 when the model is applied to previously unseen test data. Consequently, no data augmentation was necessary.



```
48/48 - 0s - loss: 0.0618 - categorical_accuracy: 0.9688 - val_loss: 0.1294 - val_categorical_accuracy: 0.9583
Epoch 190/200
48/48 - 0s - loss: 0.0761 - categorical_accuracy: 0.9583 - val_loss: 0.0842 - val_categorical_accuracy: 0.9583
Epoch 191/200
48/48 - 0s - loss: 0.0867 - categorical_accuracy: 0.9688 - val_loss: 0.0587 - val_categorical_accuracy: 0.9583
Epoch 192/200
48/48 - 0s - loss: 0.0809 - categorical_accuracy: 0.9479 - val_loss: 0.0433 - val_categorical_accuracy: 1.0000
Epoch 193/200
48/48 - 0s - loss: 0.0799 - categorical_accuracy: 0.9688 - val_loss: 0.0555 - val_categorical_accuracy: 0.9583
Epoch 194/200
48/48 - 0s - loss: 0.0697 - categorical_accuracy: 0.9583 - val_loss: 0.0561 - val_categorical_accuracy: 0.9583
Epoch 195/200
48/48 - 0s - loss: 0.1541 - categorical_accuracy: 0.9271 - val_loss: 0.0563 - val_categorical_accuracy: 0.9583
Epoch 196/200
48/48 - 0s - loss: 0.0668 - categorical_accuracy: 0.9688 - val_loss: 0.0598 - val_categorical_accuracy: 0.9583
Epoch 197/200
48/48 - 0s - loss: 0.1136 - categorical_accuracy: 0.9375 - val_loss: 0.0450 - val_categorical_accuracy: 1.0000
Epoch 198/200
48/48 - 0s - loss: 0.0687 - categorical_accuracy: 0.9479 - val_loss: 0.0445 - val_categorical_accuracy: 1.0000
Epoch 199/200
48/48 - 0s - loss: 0.0657 - categorical_accuracy: 0.9792 - val_loss: 0.1207 - val_categorical_accuracy: 0.9583
Epoch 200/200
48/48 - 0s - loss: 0.0874 - categorical_accuracy: 0.9688 - val_loss: 0.0490 - val_categorical_accuracy: 0.9583
1/1 [==============================] - 0s 132ms/step - loss: 0.0361 - categorical_accuracy: 1.0000
Test Loss: 0.03612593933939934
Test accuracy: 1.0
```

Figure 17 - Performance on Training Data against Test Data

We believe that this is a good model based on the consistently good performance results, both during training and validation and afterwards with the testing set.

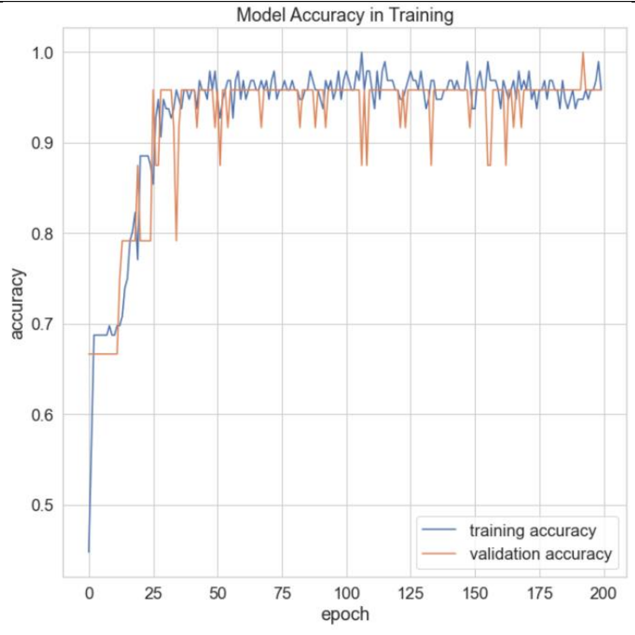
9. Varying hyperparameters

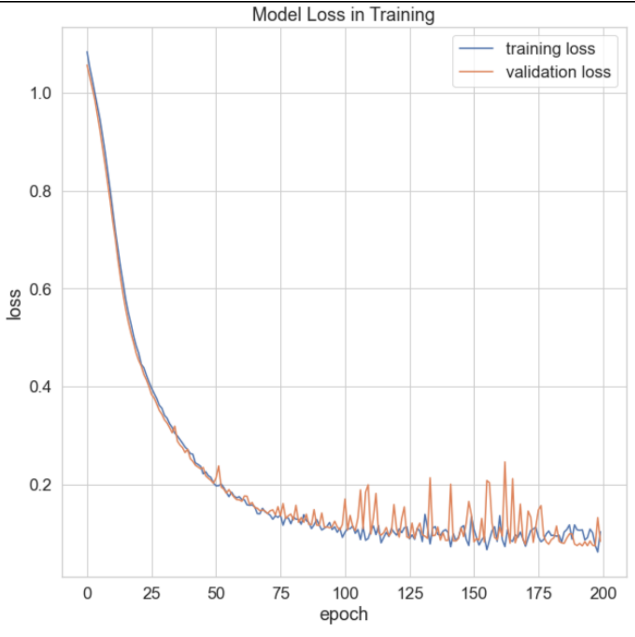
We experimented with some different hyperparameters to look for changes in learning curve or accuracy:

1. We changed the optimiser from Adam to Stochastic Gradient Descent. When compared with our original configuration we observed that it took longer to reach a higher level of accuracy during training and that accuracy during validation was overall lower. We also noted that loss decreased at a slower rate.

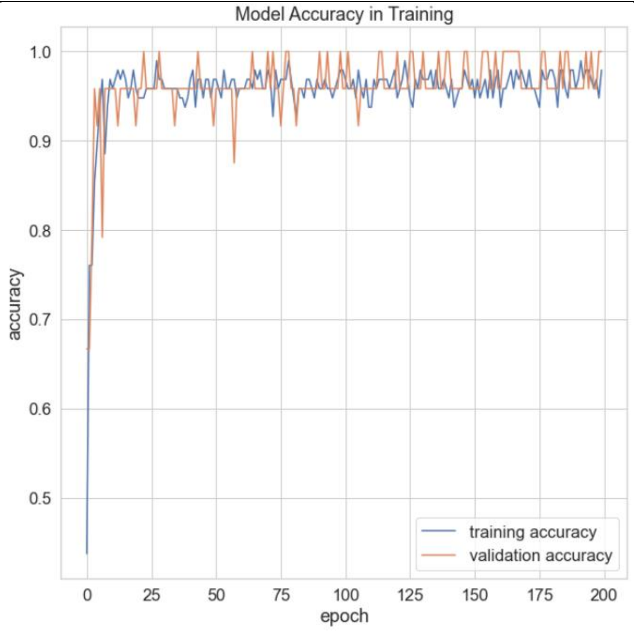
Stochastic Gradient Descent vs Adam

Stochastic Gradient Descent





Adam



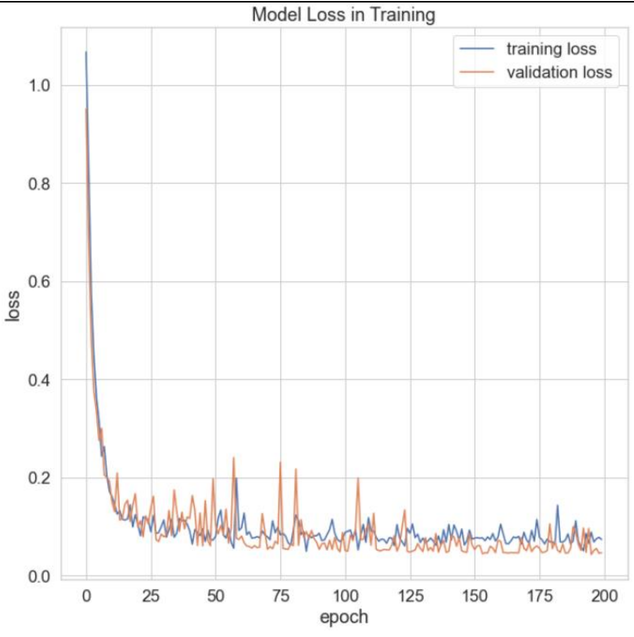


Figure 18 - Stochastic Gradient Descent vs Adam

2. We changed the loss function from Categorical Crossentropy to Mean Squared Error. Here we noted only slight differences from the original configuration with larger drops in accuracy and increases in loss during validation.

Mean Squared Error vs Categorical Crossentropy

Other hyperparameter values:

Optimiser = Adam, Metric = Categorical Accuracy

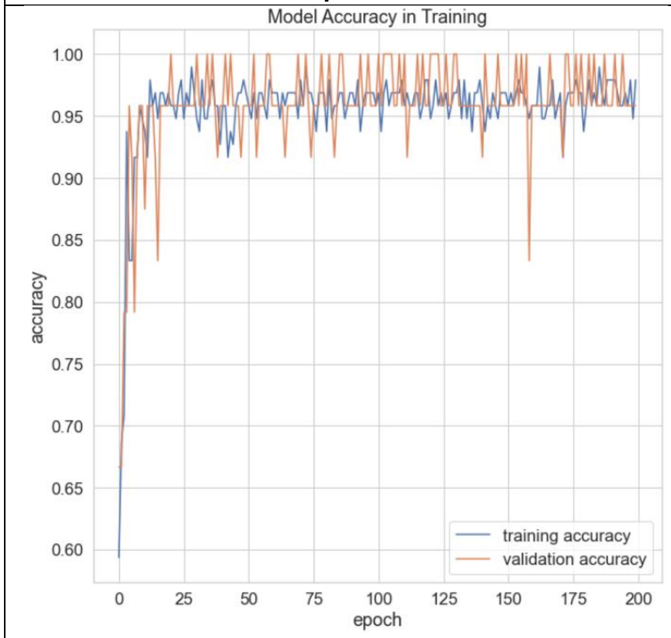
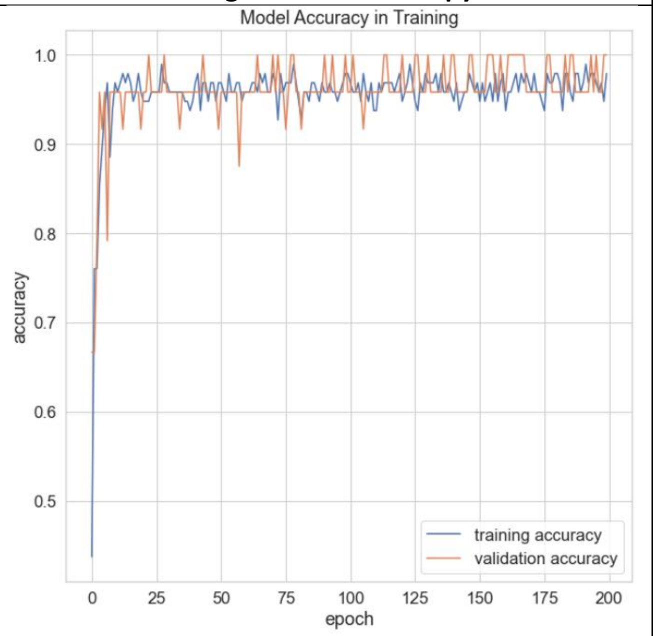
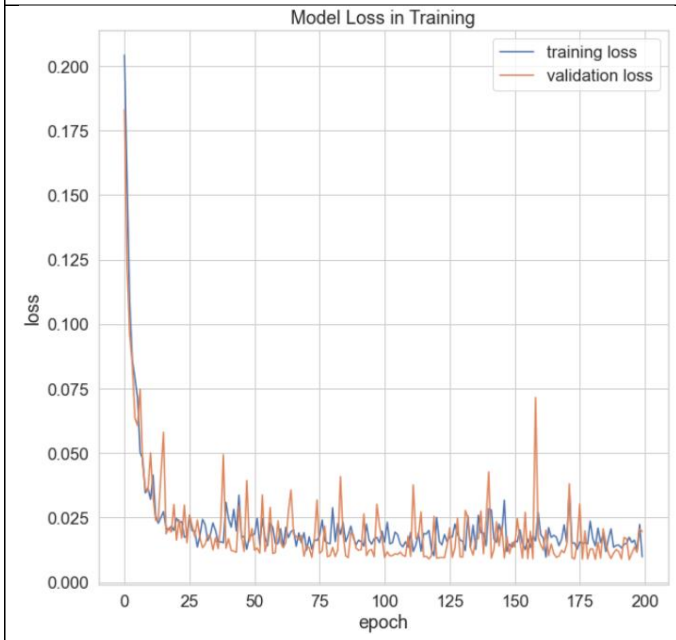
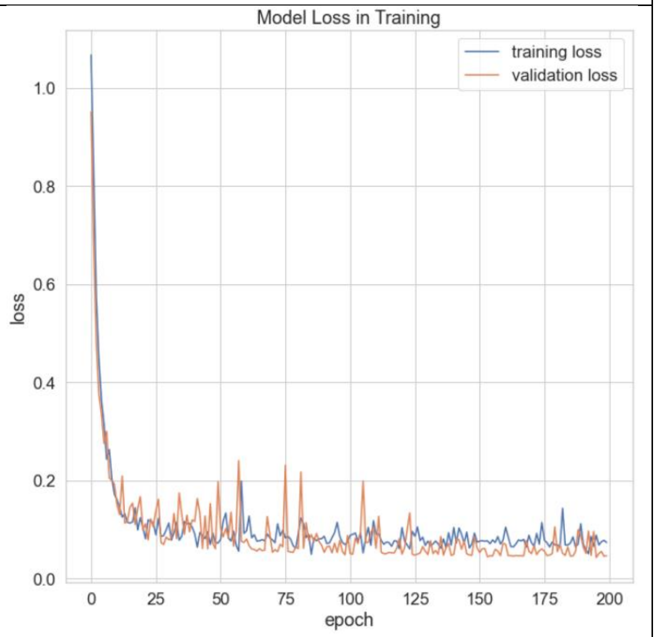
Mean Squared Error	**Categorical Crossentropy**
	
	

Figure 19 - Mean Squared Error vs Categorical Crossentropy

3. Here we replaced the original optimiser and loss function with Stochastic Gradient Descent and Mean Squared error and observed a notable difference during training. During training accuracy increases very quickly at the beginning then plateaus for approx. 65-70 epochs. It then increases in smaller steps gradually reaching about 0.96. Accuracy during validation in increase in larger steps but there are very few decreases as it reaches the end of the training period. Loss has become and almost complete smooth curve only starting to deviate after the halfway mark is passing in training.

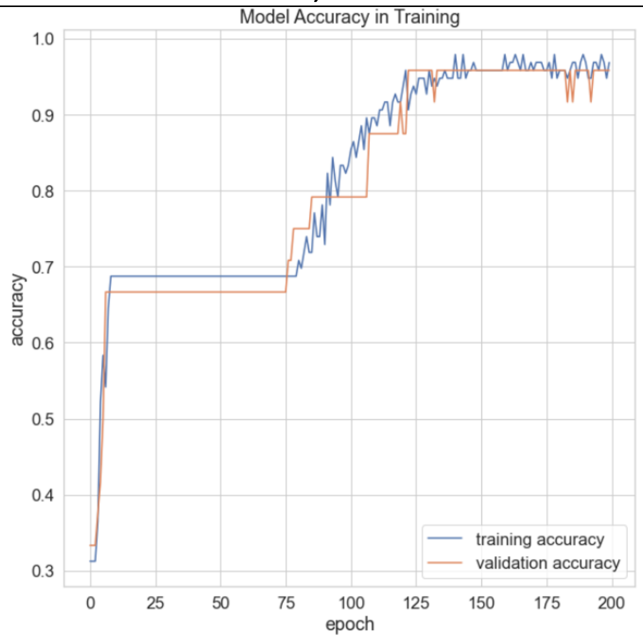
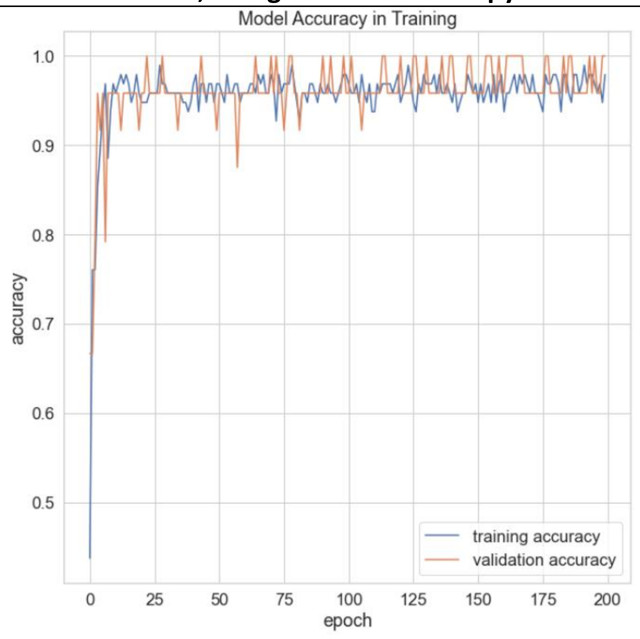
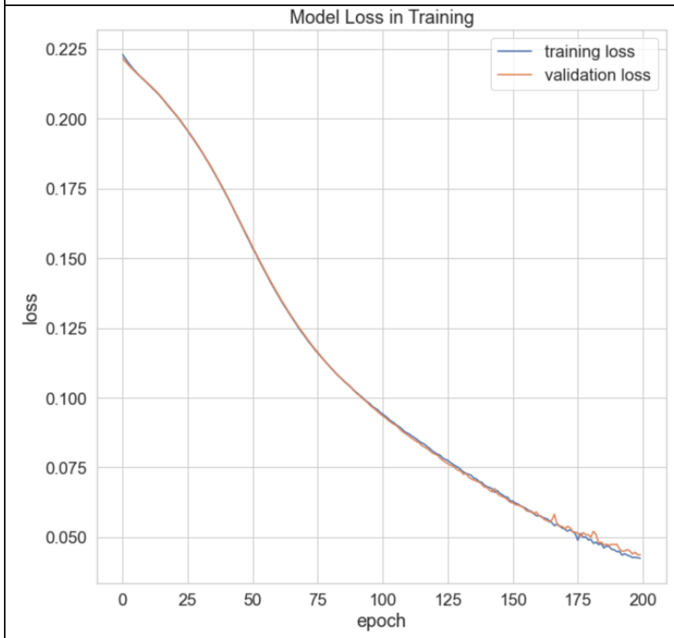
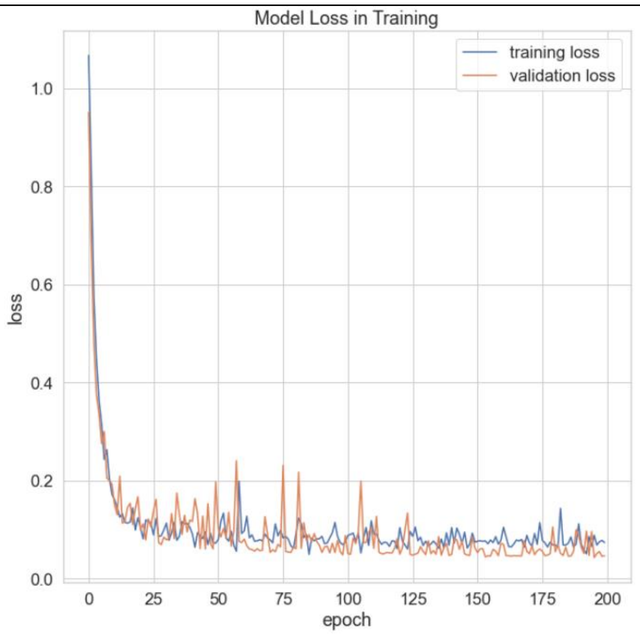
Stochastic Gradient Descent, Mean Squared Error vs Adam, Categorical Crossentropy Other hyperparameter values: Metric = Categorical Accuracy	
SGD, MSE	**Adam, Categorical Crossentropy**
	
	

Figure 20 - Stochastic Gradient Descent, Mean Squared Error vs Adam, Categorical Crossentropy

4. Here we used Recall as the metric and compared the differences between using SGD/MSE to calculate it against using Adam/Categorical Crossentropy. We found that using SGD/MSE means the model take a little longer to get a high level of recall however once it gets there it is higher than the level of recall achieved by the Adam/Categorical Crossentropy model. Additionally, the loss decreases at a steadier rate in the SGD/MSE model.

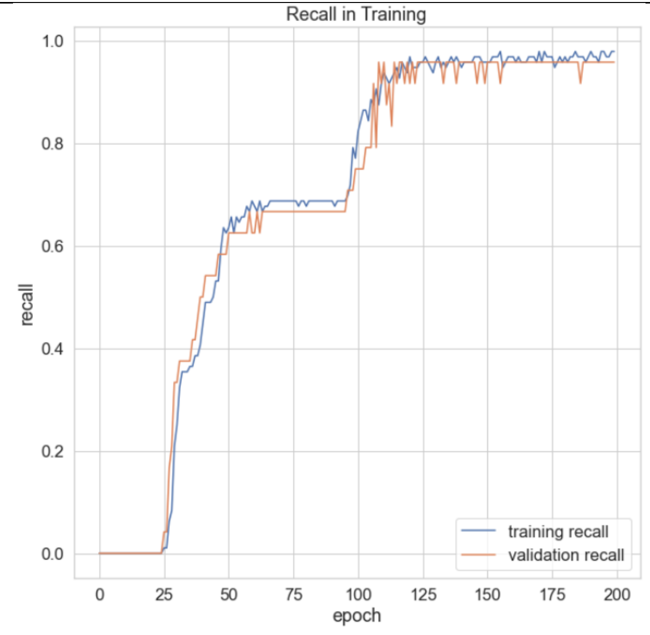
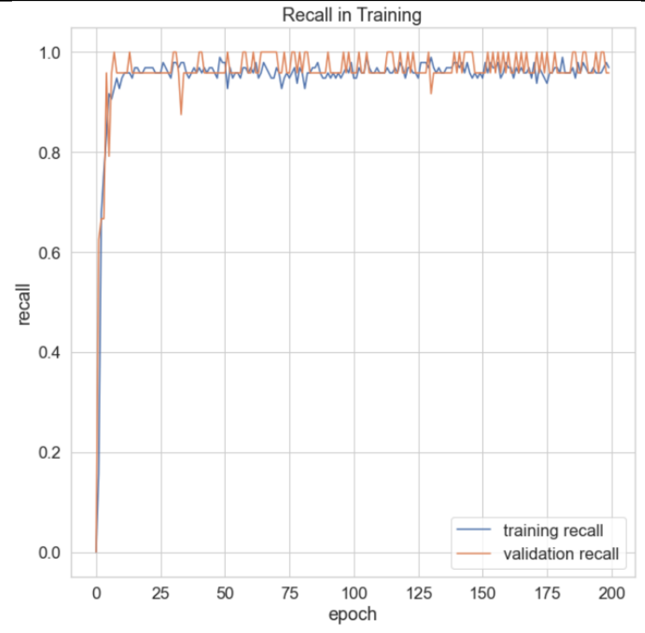
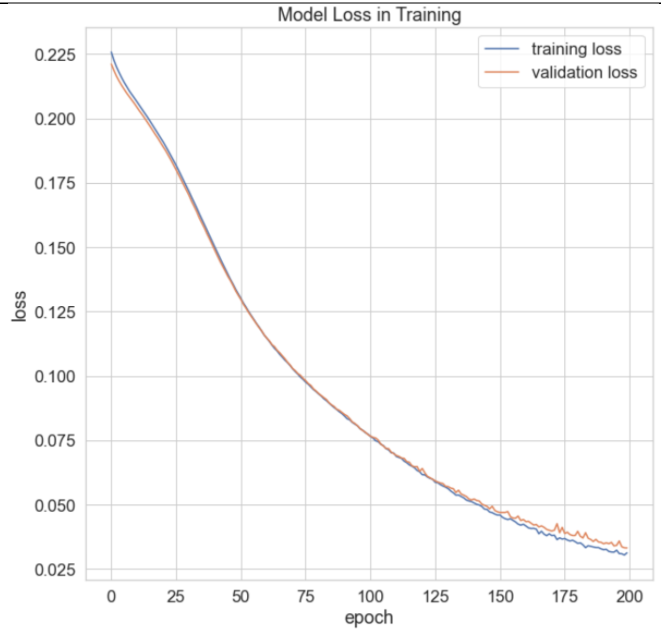
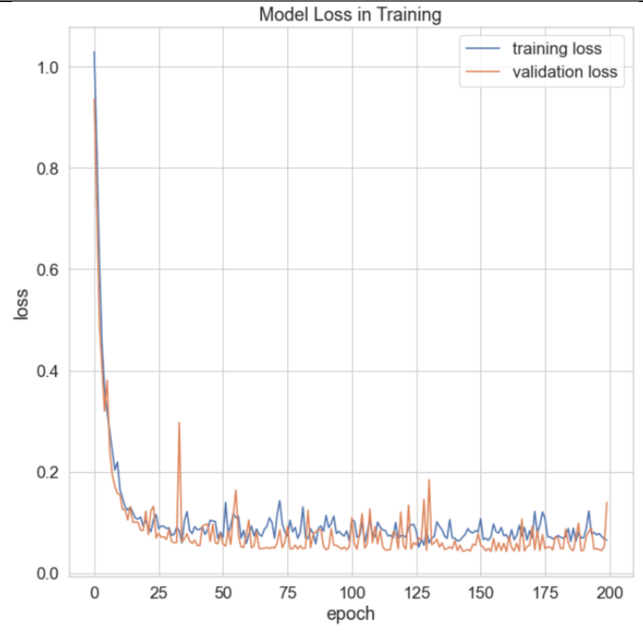
Stochastic Gradient Descent, Mean Squared Error, Recall vs Adam, Categorical Crossentropy, Recall	
SGD, MSE, Recall – Recall =0.96	**Adam, Categorical Crossentropy, Recall – Recall = 0.90**
	
	

Figure 21 - Stochastic Gradient Descent, Mean Squared Error, Recall vs Adam, Categorical Crossentropy, Recall

Sources:

https://www.kaggle.com/uciml/iris

https://www.kaggle.com/louisong97/neural-network-approach-to-iris-dataset

https://www.machinecurve.com/index.php/2020/02/18/how-to-use-k-fold-cross-validation-with-keras/

https://developers.google.com/machine-learning/data-prep/transform/normalization

https://www.datatechnotes.com/2019/05/one-hot-encoding-example-in-python.html

https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b

https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/

https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

https://keras.io/api/layers/activations/

https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e

https://stackoverflow.com/questions/41908379/keras-plot-training-validation-and-test-set-accuracy

https://www.machinecurve.com/index.php/2020/02/18/how-to-use-k-fold-cross-validation-with-keras/