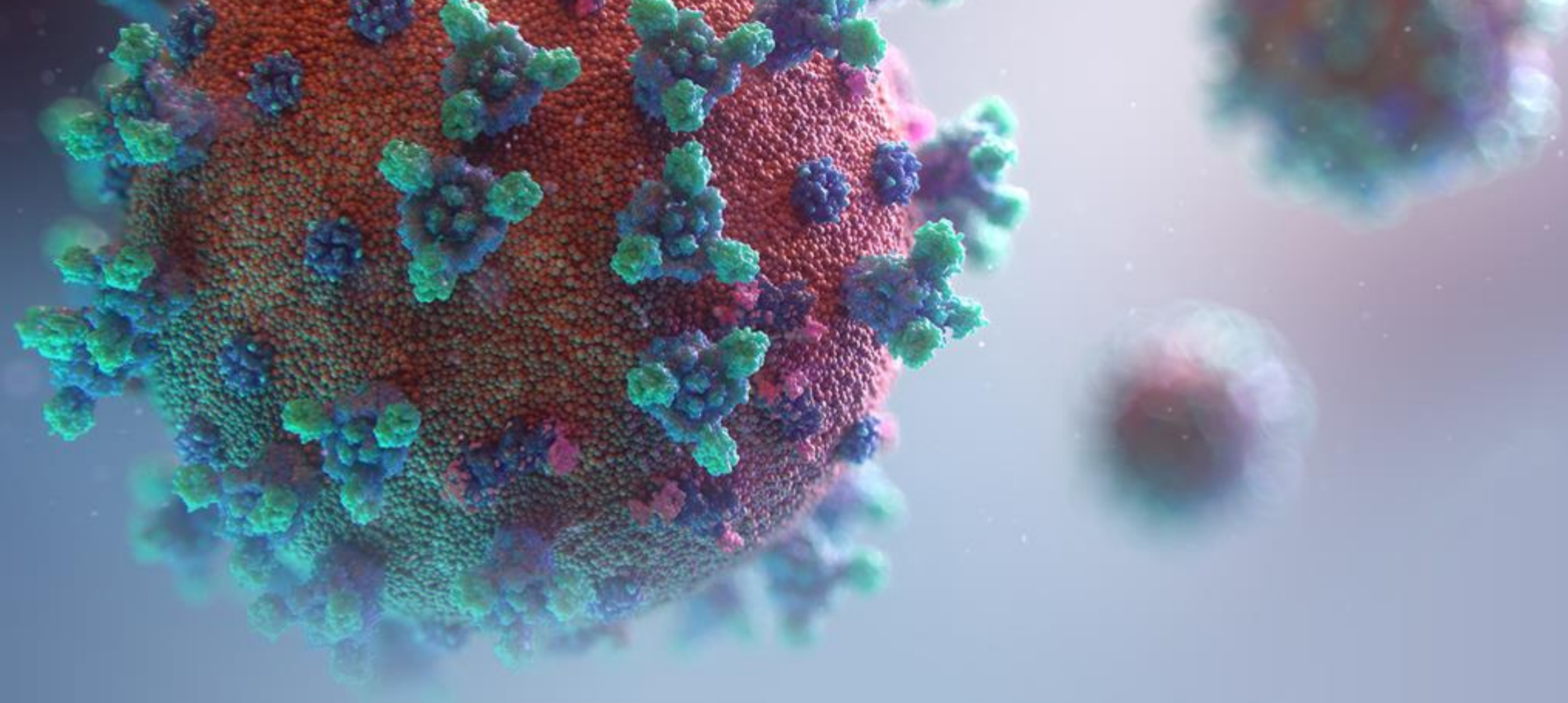


Leaf Classification

By: Yara Elzahy
ID: 20398570

Problem Statement

We want to find and predict the **Leaf Classification** given the other features Using Neural network.

- **Input**: Features collected from half a million species of plant in the world.
- **Output**: Predicted species for leaves.
- **Deep Learning Function**: Manipulating, analyzing, preprocessing the data, and training the data.
- **Problem**:

  Classification of species has been historically problematic and often results in duplicate identifications.

- **Objective**:

  The objective of this playground competition is to use binary leaf images and extracted features, including shape, margin & texture, to accurately identify 99 species of plants. Leaves, due to their volume, prevalence, and unique characteristics, are an effective means of differentiating plant species.

- **Challenges**:
  1. Nan cells.
  2. Unused and unnecessary column.
  3. convert strings by label encoding.
  4. choose the best hyper-parameters for the network.

- **Impact**: Predicting the species of the leaf that will lead to a successful match.

Data Description:

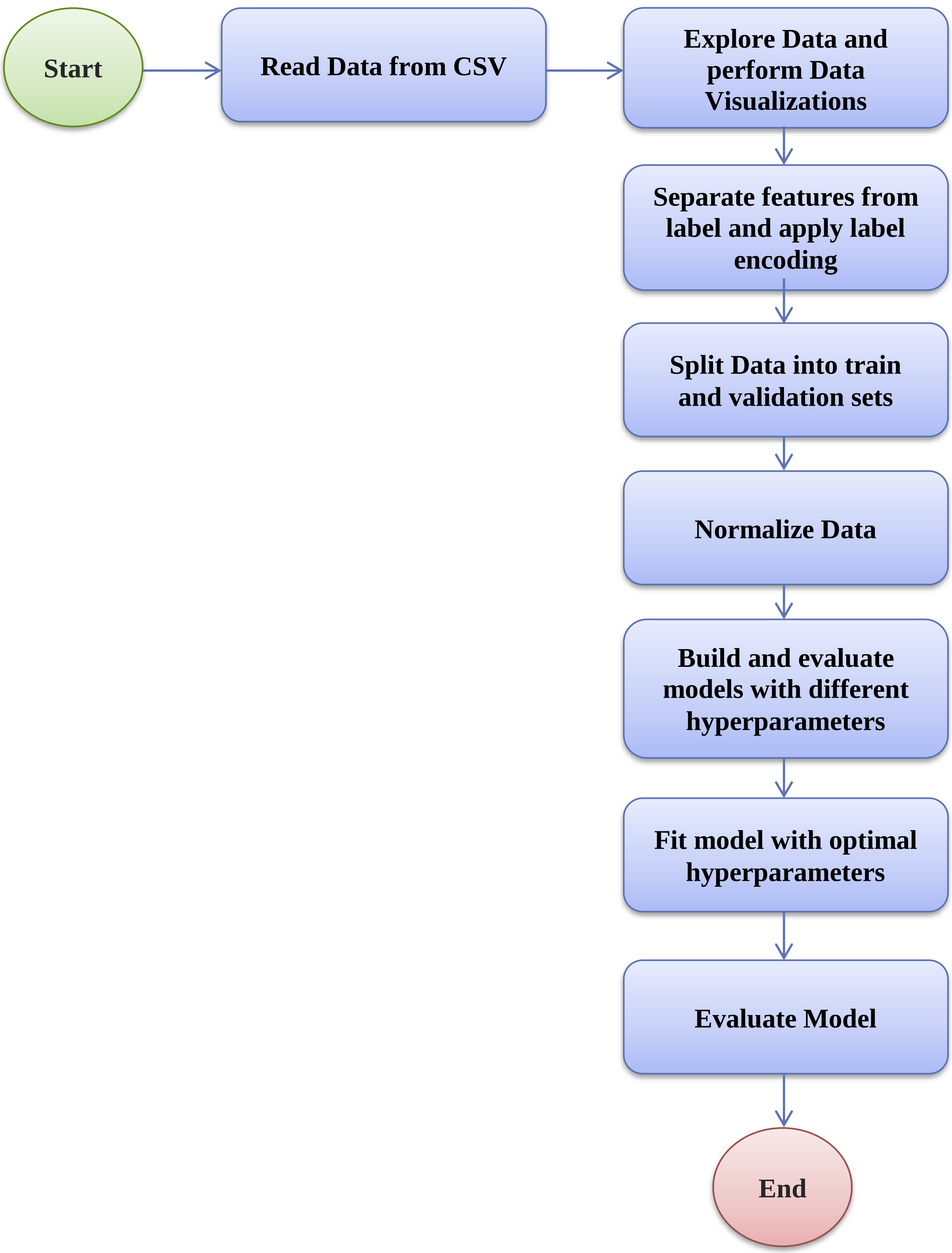
- The dataset consists approximately 1,584 images of leaf specimens (16 samples each of 99 species) which have been converted to binary black leaves against white backgrounds. Three sets of features are also provided per image: a shape contiguous descriptor, an interior texture histogram, and a fine-scale margin histogram. For each feature, a 64-attribute vector is given per leaf sample.

- Note that of the original 100 species, we have eliminated one on account of incomplete associated data in the original dataset.

Data fields:

- id - an anonymous id unique to an image
- margin_1, margin_2, margin_3, ..., margin_64 - each of the 64 attribute vectors for the margin feature
- shape_1, shape_2, shape_3, ..., shape_64 - each of the 64 attribute vectors for the shape feature
- texture_1, texture_2, texture_3, ..., texture_64 - each of the 64 attribute vectors for the texture feature

Project Pipeline



Part 1 – Data Preparation

1) Read and Display Training Data

In [2]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
```

In [3]:
```
train_data.head()
```

Out[3]:

	id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	margin11	margin12	margin13
0	1	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	0.001953	0.033203	0.013672	0.019531	0.066406
1	2	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	0.000000	0.007812	0.003906	0.027344	0.023438
2	3	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	0.000000	0.044922	0.007812	0.011719	0.021484
3	5	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	0.013672	0.017578	0.001953	0.019531	0.001953
4	6	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	0.000000	0.005859	0.001953	0.044922	0.041016

2) Describe the Data

In [5]:
```
train_data.describe()
```

Out[5]:

	id	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	margin11	ma
count	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.0
mean	799.595960	0.017412	0.028539	0.031988	0.023280	0.014264	0.038579	0.019202	0.001083	0.007167	0.018639	0.024209	0.0
std	452.477568	0.019739	0.038855	0.025847	0.028411	0.018390	0.052030	0.017511	0.002743	0.008933	0.016071	0.026086	0.0
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	415.250000	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.001953	0.005859	0.003906	0.0
50%	802.500000	0.009766	0.011719	0.025391	0.013672	0.007812	0.015625	0.015625	0.000000	0.005859	0.015625	0.013672	0.0
75%	1195.500000	0.025391	0.041016	0.044922	0.029297	0.017578	0.056153	0.029297	0.000000	0.007812	0.027344	0.041016	0.0
max	1584.000000	0.087891	0.205080	0.156250	0.169920	0.111330	0.310550	0.091797	0.031250	0.076172	0.097656	0.125000	0.0

3) Check for Null values

Data contains no null values

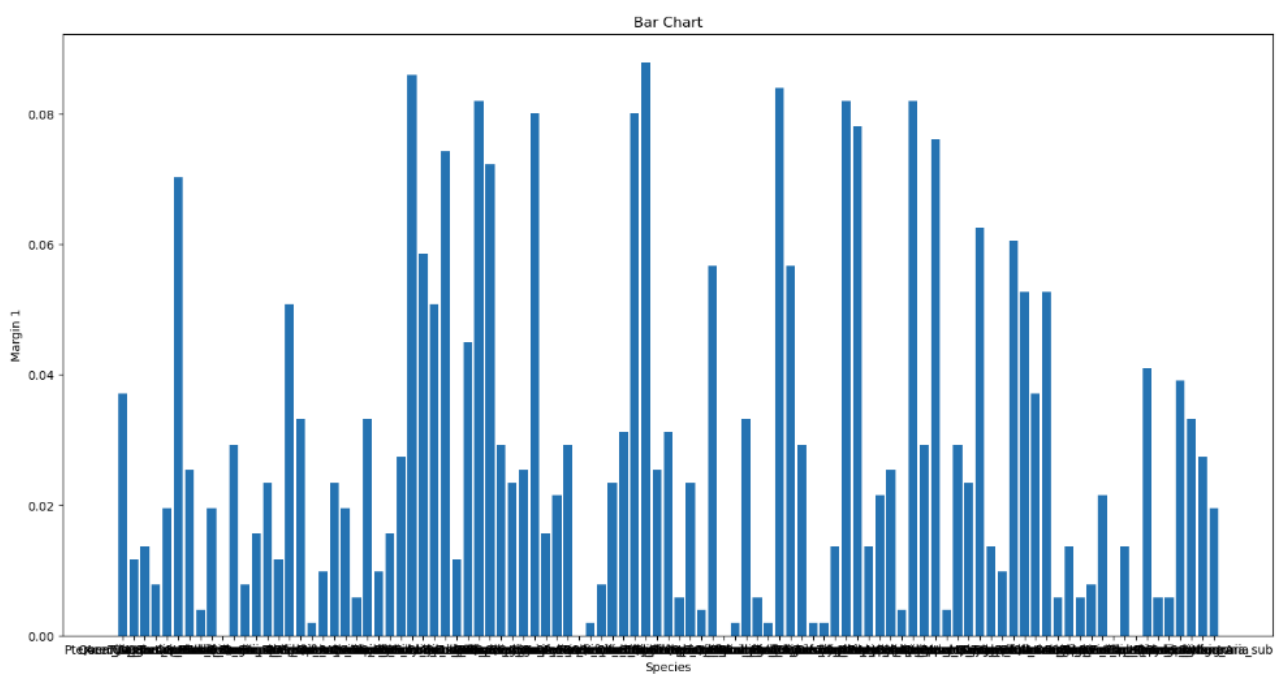
```
In [8]: train_data.isnull().sum()

Out[8]: id          0
        species     0
        margin1     0
        margin2     0
        margin3     0
        margin4     0
        margin5     0
        margin6     0
        margin7     0
        margin8     0
        margin9     0
        margin10    0
        margin11    0
        margin12    0
        margin13    0
        margin14    0
        margin15    0
        margin16    0
        margin17    0
```

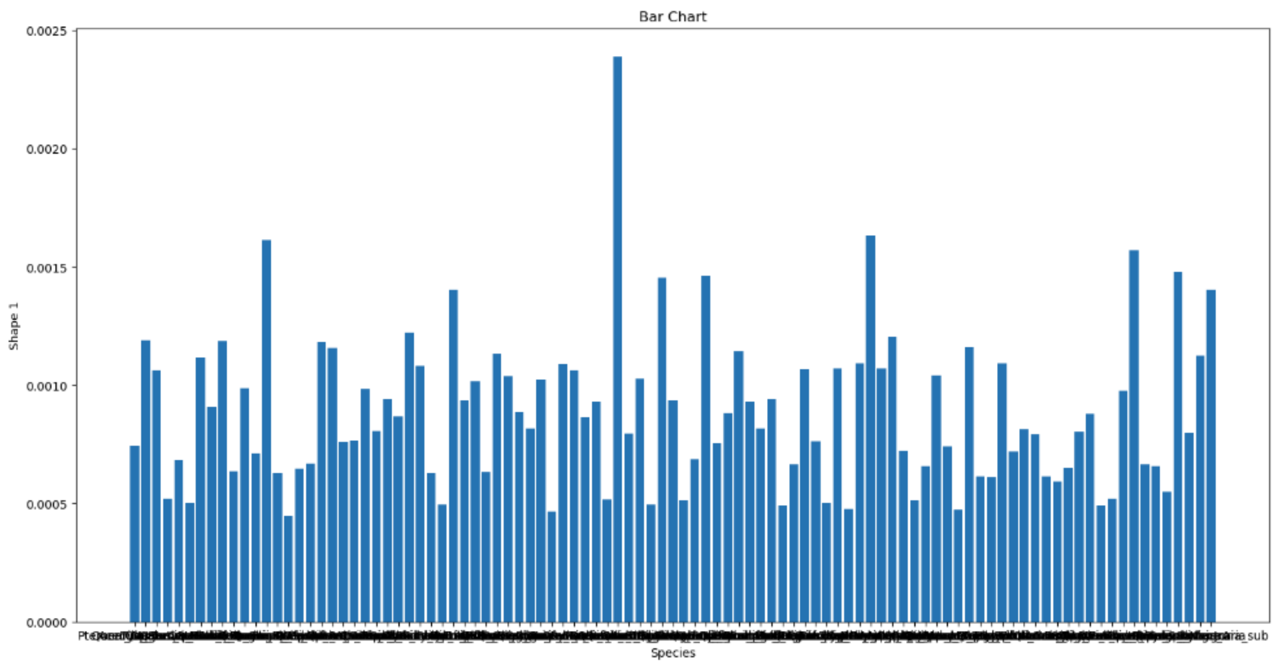
4) Check for Duplicates

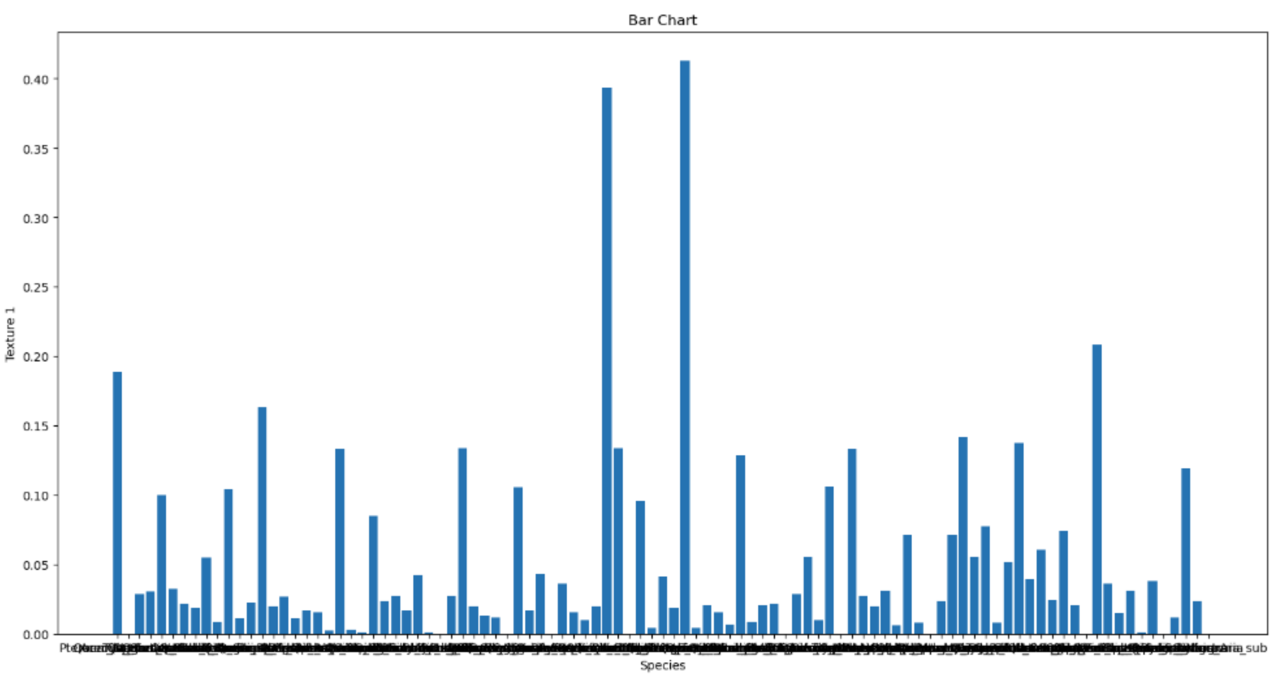
Data has no duplicate values

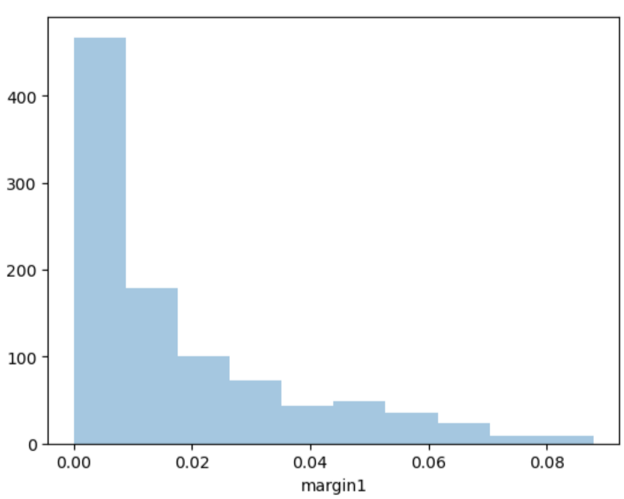
```
In [9]: train_data.duplicated().any()

Out[9]: False
```

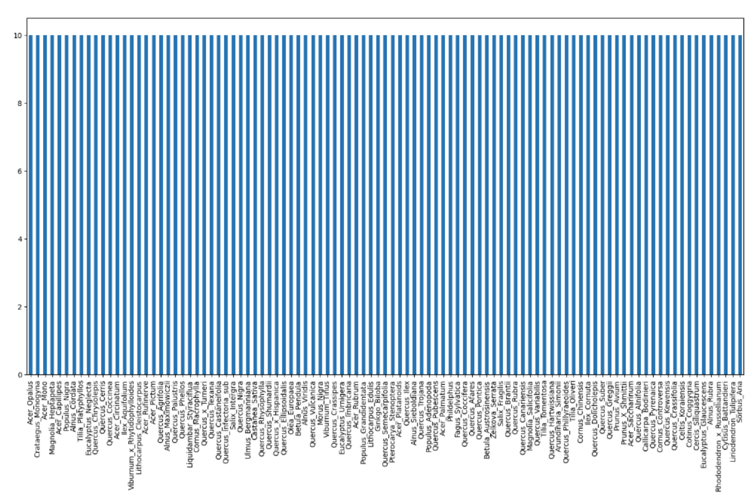
5) Perform some Data Visualizations to understand the data distribution and correlations



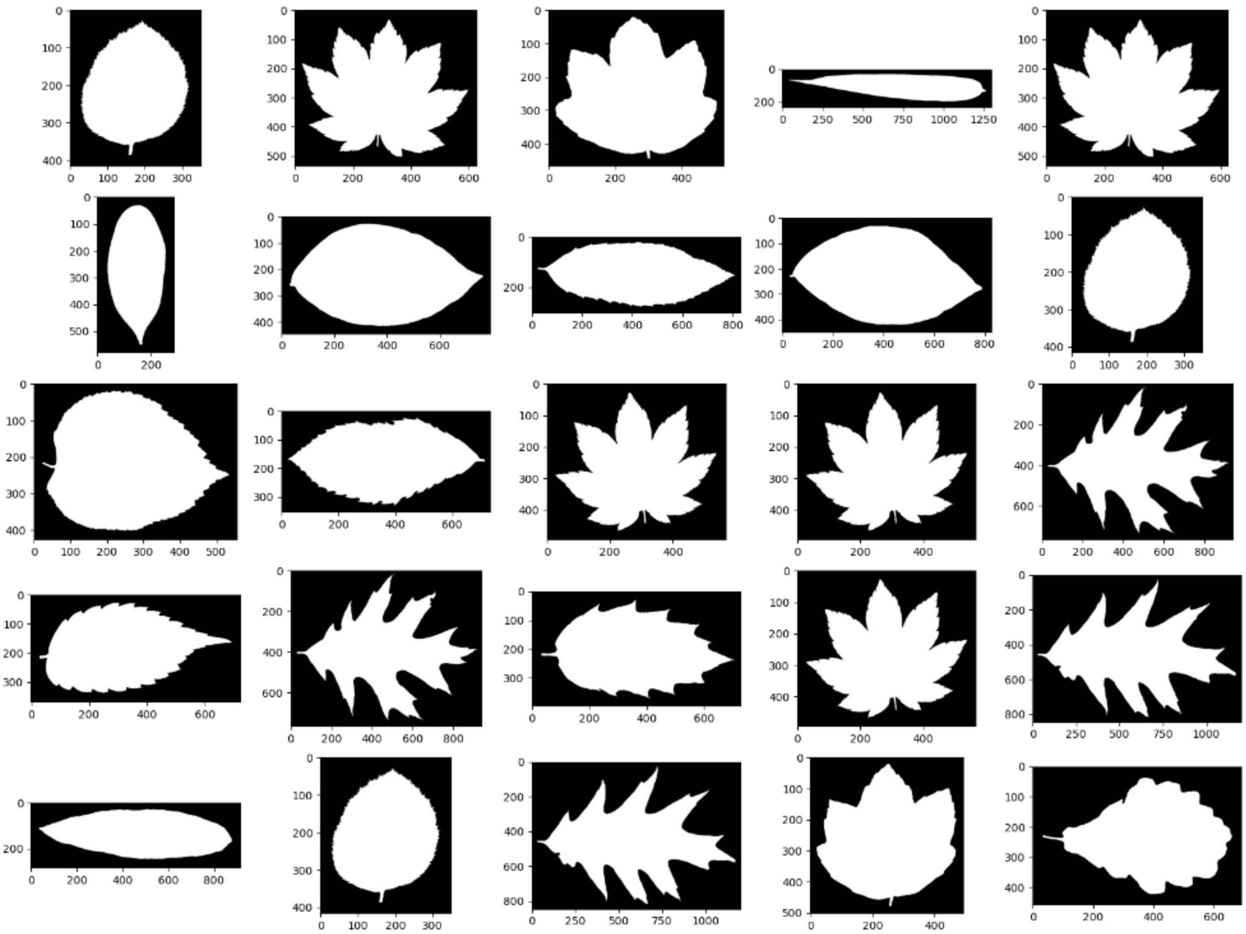

Bar Chart
Shape 1
Species
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000


Bar Chart
Texture 1
Species
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00

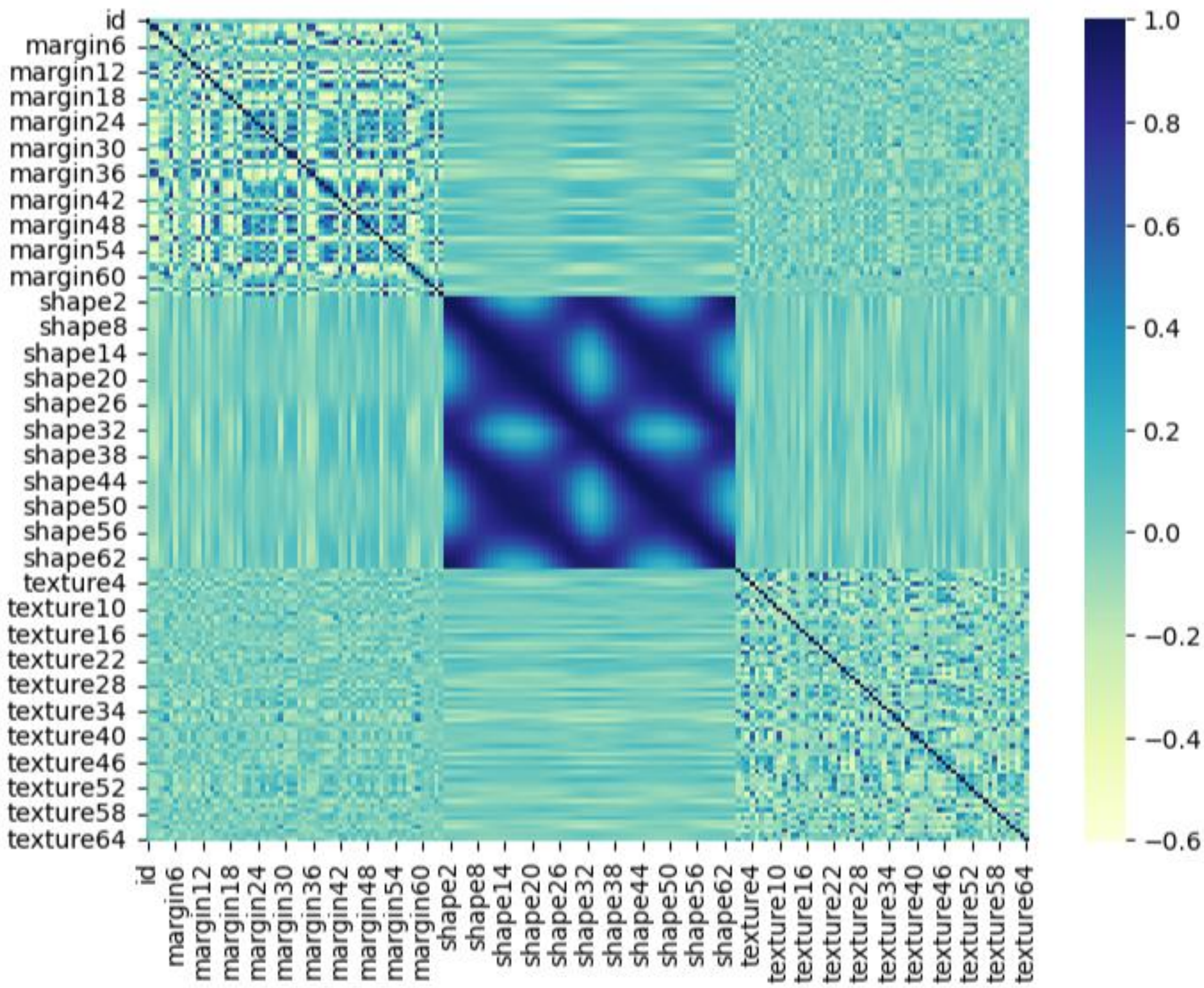

400
300
200
100
0
0.00
0.02
0.04
0.06
0.08
margin1



6) Display some images



7) Correlation Matrix



A clearer correlations figure:

	id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	margin11	margin12
id	1.000000	0.071345	-0.011673	-0.027565	-0.059533	0.001639	-0.002419	-0.051818	0.061214	-0.039509	-0.070954	0.016381	0.020884	0.019541
species	0.071345	1.000000	-0.018265	-0.089907	0.182852	0.120366	-0.119675	-0.074594	0.179173	-0.055272	-0.204726	0.123252	-0.007464	-0.002833
margin1	-0.011673	-0.018265	1.000000	0.806390	-0.182829	-0.297807	-0.475874	0.767718	0.066273	-0.094137	-0.181496	0.397138	0.737461	-0.528224
margin2	-0.027565	-0.089907	0.806390	1.000000	-0.204640	-0.315953	-0.444312	0.825762	-0.083273	-0.086428	-0.120276	0.162587	0.805064	-0.489808
margin3	-0.059533	0.182852	-0.182829	-0.204640	1.000000	0.120042	-0.185007	-0.163976	0.095449	0.024350	-0.000042	0.008772	-0.261371	-0.004085
margin4	0.001639	0.120366	-0.297807	-0.315953	0.120042	1.000000	0.029480	-0.261437	-0.268271	-0.047693	0.227543	-0.173986	-0.172503	-0.202576
margin5	-0.002419	-0.119675	-0.475874	-0.444312	-0.185007	0.029480	1.000000	-0.438587	-0.108178	0.056557	0.196745	-0.320647	-0.514981	0.373683
margin6	-0.051818	-0.074594	0.767718	0.825762	-0.163976	-0.261437	-0.438587	1.000000	-0.093780	-0.112896	-0.136961	0.215141	0.686998	-0.479464
margin7	0.061214	0.179173	0.066273	-0.083273	0.095449	-0.268271	-0.108178	-0.093780	1.000000	0.099867	-0.350804	0.649311	-0.069978	-0.144810
margin8	-0.039509	-0.055272	-0.094137	-0.086428	0.024350	-0.047693	0.056557	-0.112896	0.099867	1.000000	-0.071887	0.012918	-0.108453	0.044335
margin9	-0.070954	-0.204726	-0.181496	-0.120276	-0.000042	0.227543	0.196745	-0.136961	-0.350804	-0.071887	1.000000	-0.337466	-0.139592	-0.065846
margin10	0.016381	0.123252	0.397138	0.162587	0.008772	-0.173986	-0.320647	0.215141	0.649311	0.012918	-0.337466	1.000000	0.226220	-0.384797
margin11	0.020884	-0.007464	0.737461	0.805064	-0.261371	-0.172503	-0.514981	0.686998	-0.069978	-0.108453	-0.139592	0.226220	1.000000	-0.579610
margin12	0.019541	-0.002833	-0.528224	-0.489808	-0.004085	-0.202576	0.373683	-0.479464	-0.144810	0.044335	-0.065846	-0.384797	-0.579610	1.000000
margin13	-0.052985	-0.107527	0.489317	0.647166	-0.048698	-0.238041	-0.463328	0.539807	-0.116093	-0.049359	-0.053170	0.043592	0.665286	-0.429504
margin14	-0.044146	-0.060140	-0.370460	-0.316377	0.095701	0.338136	0.095697	-0.317465	-0.357485	0.001100	0.372013	-0.416157	-0.373834	0.207537
margin15	0.013458	0.005471	-0.540974	-0.503059	0.050113	-0.259813	0.467991	-0.489144	0.004146	0.062293	-0.117375	-0.320361	-0.603805	0.806811
margin16	0.072274	-0.075122	-0.072127	-0.068356	-0.054076	-0.021615	0.081766	-0.065768	-0.023989	0.205817	-0.026071	-0.053492	-0.081410	0.061776
margin17	0.020472	0.117539	0.316704	0.135000	-0.130220	-0.047704	-0.235063	0.120157	0.396388	0.025698	-0.236991	0.566792	0.226382	-0.340929
margin18	-0.026818	-0.018357	0.283239	0.345410	-0.092062	0.093686	-0.431084	0.256036	-0.149460	-0.065664	-0.046305	0.059246	0.616074	-0.444259
margin19	0.010789	0.032658	-0.234398	-0.226020	-0.164152	0.362009	0.358065	-0.267886	-0.153342	0.002255	0.231236	-0.202346	-0.164134	-0.091540
margin20	0.032297	0.128452	0.325947	0.062345	0.012338	0.056523	-0.326563	0.159341	0.340324	-0.043785	-0.249363	0.640144	0.252003	-0.399623
margin21	0.032834	0.044843	-0.433734	-0.421253	0.042328	-0.138539	0.066151	-0.414130	-0.008999	0.068751	0.038512	-0.252722	-0.480018	0.603524
margin22	0.058120	-0.148561	-0.404022	-0.364703	-0.282862	-0.194713	0.273729	-0.363723	-0.130686	0.041311	0.092984	-0.319991	-0.428656	0.633346
margin23	0.073094	-0.091159	-0.142871	-0.136586	-0.145334	-0.004602	0.287659	-0.126238	-0.059832	-0.034131	0.108722	-0.127245	-0.150064	0.031318
margin24	0.050021	-0.086647	-0.315616	-0.302345	-0.255676	-0.144124	0.125076	-0.312633	0.063813	0.000173	0.045558	-0.141180	-0.315524	0.348073

As we can see in the above figure, margin3, margin4, margin7, margin10, margin17 and margin20 have the highest correlation values with the species column

8) **Encode labels** (convert string to numerical values)

In [20]:
```
# Load the label encoder
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'
train_data['species']= label_encoder.fit_transform(train_data['species'])

train_data['species'].unique()
```

Out[20]:
```
array([ 3, 49, 65, 94, 84, 40, 54, 78, 53, 89, 98, 16, 74, 50, 58, 31, 43,
        4, 75, 44, 83, 13, 66, 15,  6, 73, 22, 36, 27, 88, 12, 28, 21, 25,
       20, 60, 69, 23, 76, 18, 52,  9, 48, 47, 64, 81, 62, 34, 92, 79, 82,
       32, 35, 72, 71, 11, 51,  5,  8, 37, 97, 33,  1, 59, 56, 57, 29, 93,
       10, 46,  0, 39,  2, 24, 26, 87, 55, 38, 45,  7, 67, 30, 61, 96, 41,
       85, 14, 17, 42, 63, 86, 80, 77, 19, 95, 70, 90, 68, 91])
```

9) Split Data into training and validation sets

```
In [21]: X_train, X_valid, y_train, y_valid = train_test_split( train_data.drop(columns=["species","id"]),
                                                               train_data['species'],
                                                               stratify=train_data['species'],
                                                               test_size=0.2,
                                                               random_state=42)
```

```
In [22]: X_train
```

447	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
558	0.023438	0.019531	0.015625	0.027344	0.003906	0.025391	0.007812	0.000000	0.000000	0.025391	0.033203	0.003906	0.035156	0.000000	0.00585
826	0.019531	0.015625	0.031250	0.041016	0.021484	0.041016	0.042969	0.000000	0.005859	0.029297	0.009766	0.003906	0.029297	0.000000	0.00781
84	0.000000	0.000000	0.011719	0.021484	0.046875	0.000000	0.013672	0.000000	0.017578	0.007812	0.000000	0.019531	0.000000	0.007812	0.03125
293	0.011719	0.025391	0.027344	0.035156	0.003906	0.021484	0.005859	0.000000	0.005859	0.000000	0.046875	0.005859	0.070312	0.000000	0.00781
546	0.000000	0.000000	0.019531	0.015625	0.013672	0.001953	0.013672	0.000000	0.011719	0.007812	0.001953	0.029297	0.005859	0.007812	0.04882
100	0.017578	0.023438	0.062500	0.015625	0.003906	0.037109	0.033203	0.005859	0.000000	0.021484	0.000000	0.005859	0.062500	0.000000	0.01953
92	0.000000	0.007812	0.035156	0.017578	0.005859	0.000000	0.017578	0.007812	0.000000	0.025391	0.013672	0.013672	0.015625	0.001953	0.01562
78	0.025391	0.017578	0.023438	0.007812	0.003906	0.011719	0.015625	0.000000	0.003906	0.021484	0.031250	0.011719	0.035156	0.001953	0.01562
495	0.031250	0.033203	0.015625	0.054688	0.003906	0.021484	0.013672	0.000000	0.003906	0.023438	0.042969	0.001953	0.050781	0.003906	0.00000
163	0.001953	0.005859	0.009766	0.007812	0.054688	0.001953	0.005859	0.000000	0.007812	0.003906	0.003906	0.007812	0.000000	0.007812	0.02734
946	0.025391	0.013672	0.025391	0.009766	0.003906	0.007812	0.052734	0.000000	0.000000	0.041016	0.035156	0.000000	0.015625	0.000000	0.00195
275	0.007812	0.019531	0.029297	0.019531	0.021484	0.005859	0.054688	0.000000	0.005859	0.035156	0.007812	0.000000	0.035156	0.003906	0.00585

10) Normalize Data

```
In [23]: # define standard scaler
         stdscaler = StandardScaler()

         # normlize integer columns
         X_train = stdscaler.fit_transform(X_train)
         X_valid = stdscaler.transform(X_valid)
```

Part 2 – Training a Neural Network

1) Create a function that builds the neural network

Input parameters: X_train, y_train, X_val, y_val
Output: history and model

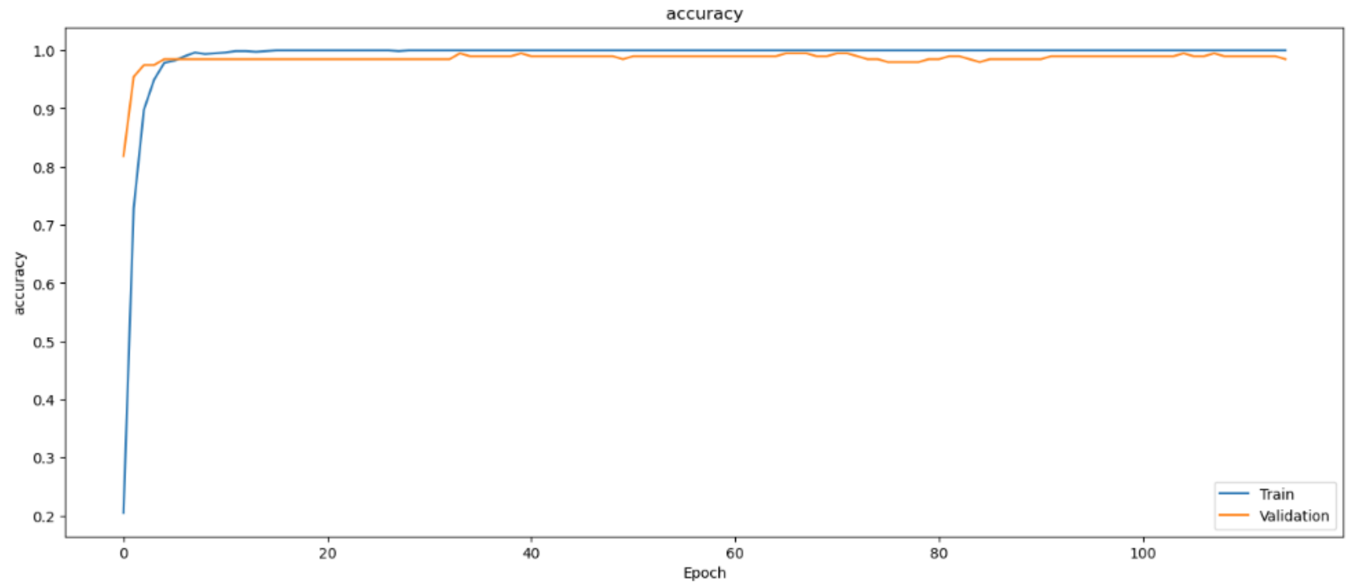
```
In [28]: def build_model(X_train, y_train, X_valid, y_valid, params):
             model = tf.keras.Sequential([
             tf.keras.Input(shape = (X_train.shape[1],) ),
             tf.keras.layers.Dense(params["hidden_layer"], activation='tanh'),
             tf.keras.layers.Dropout(params["dropout"]),
             tf.keras.layers.Dense(number_of_classes, activation='softmax')])

             model.compile(loss='SparseCategoricalCrossentropy', optimizer=params["optimizer"], metrics=['accuracy'])

             EarlyStop = tf.keras.callbacks.EarlyStopping(patience = 10)
             checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

             output = model.fit(X_train, y_train, batch_size = params["batch_size"] ,epochs = 1000, verbose = True,
                                   validation_data = (X_valid, y_valid), callbacks=[EarlyStop,checkpoint_conv])
             return output, model
```

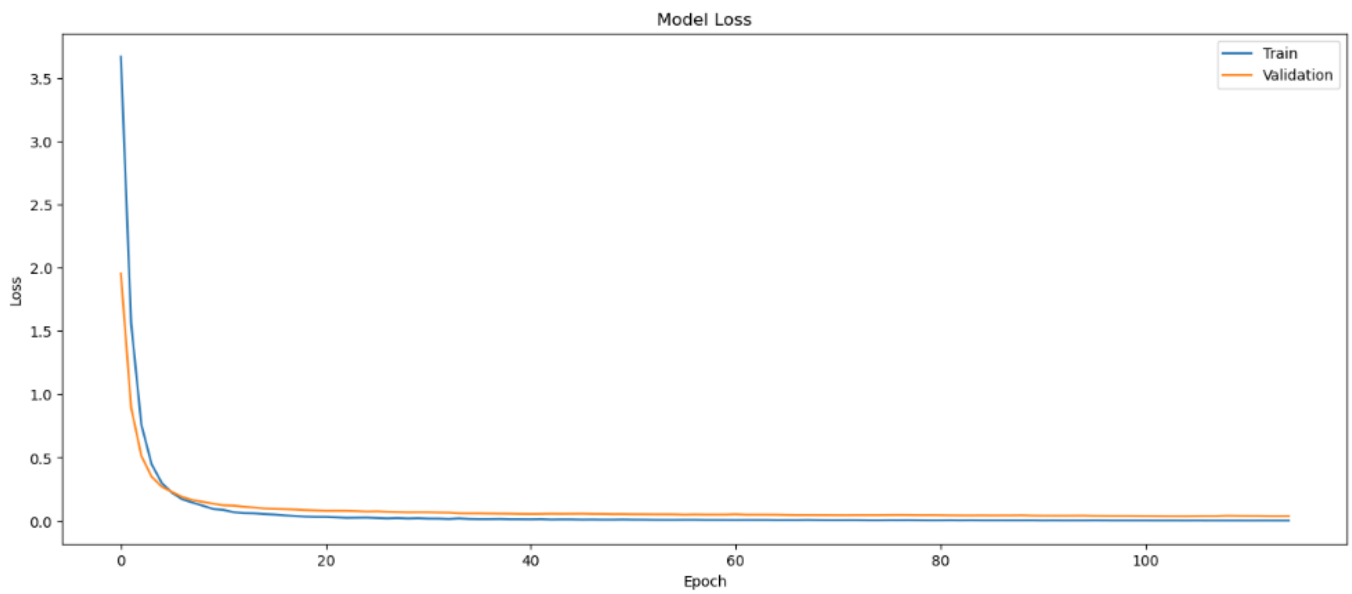
2) Create a function that evaluates the built neural network

```
In [29]: def evaluate_model_params(batch_size = 16, hidden_layer = 384, optimizer = "Adam", dropout = 0.5,
                                   learning_rate = 0.001, regularization = 0.001):

             if optimizer == "Adam":
                 opt = tf.keras.optimizers.Adam(learning_rate = learning_rate, decay = regularization)
             elif optimizer == "SGD":
                 opt = tf.keras.optimizers.SGD(learning_rate = learning_rate, decay = regularization)
             elif optimizer == "RMSprop":
                 opt = tf.keras.optimizers.RMSprop(learning_rate = learning_rate, decay = regularization)
             else:
                 print("Invalid Optimizer Name")
                 return

             params = {
               "optimizer" : opt,
               "batch_size": batch_size,
               "dropout": 0.5,
               "hidden_layer": 384
              }

             history, model = build_model(X_train, y_train, X_valid, y_valid, params)

             fig, axes = plt.subplots(2,1, figsize = [16, 16])
             axes[0].plot(history.history['accuracy'])
             try:
                 axes[0].plot(history.history['val_accuracy'])
                 axes[0].legend(['Train', 'Validation'])
             except:
                 pass
             axes[0].set_title('{:s}'.format('accuracy'))
             axes[0].set_ylabel('{:s}'.format('accuracy'))
             axes[0].set_xlabel('Epoch')
             fig.subplots_adjust(hspace=0.5)
             axes[1].plot(history.history['loss'])
             try:
                 axes[1].plot(history.history['val_loss'])
                 axes[1].legend(['Train', 'Validation'])
             except:
                 pass
             axes[1].set_title('Model Loss')
             axes[1].set_ylabel('Loss')
             axes[1].set_xlabel('Epoch')
             plt.title('Model Loss')
             plt.ylabel('Loss')
             plt.xlabel('Epoch')
```

After training the model and evaluating the model with the default hyperparameters which are:
batch_size = 16, hidden_layer = 384, optimizer = "Adam", dropout = 0.5.

Results: `loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.0373 - val_accuracy: 0.9848`



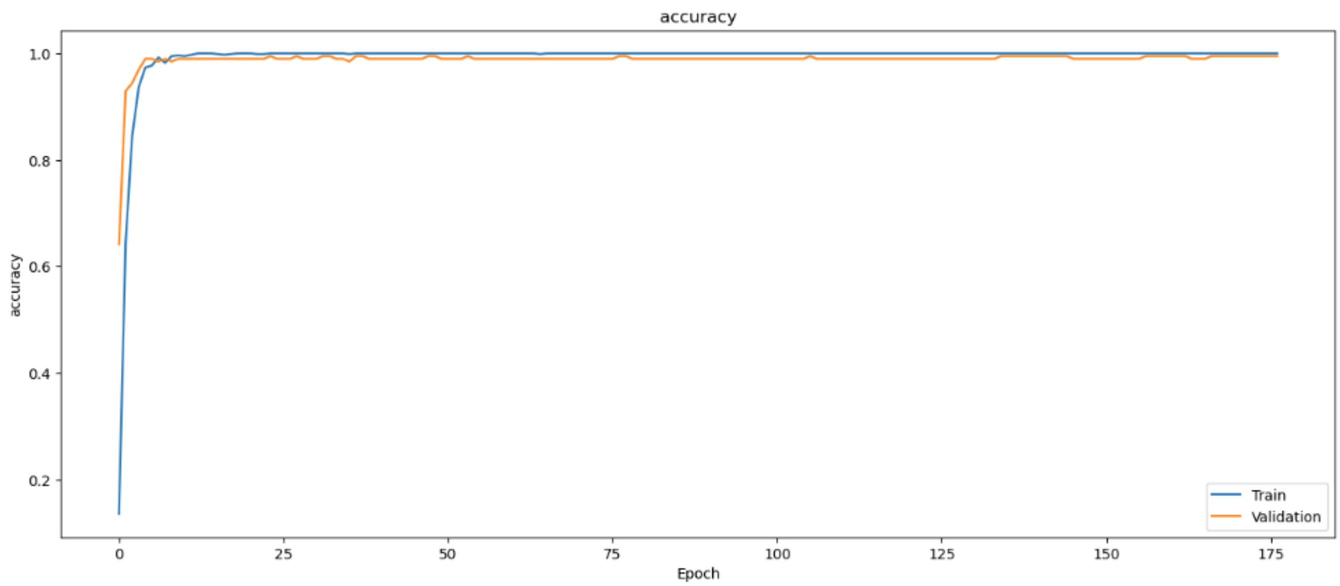


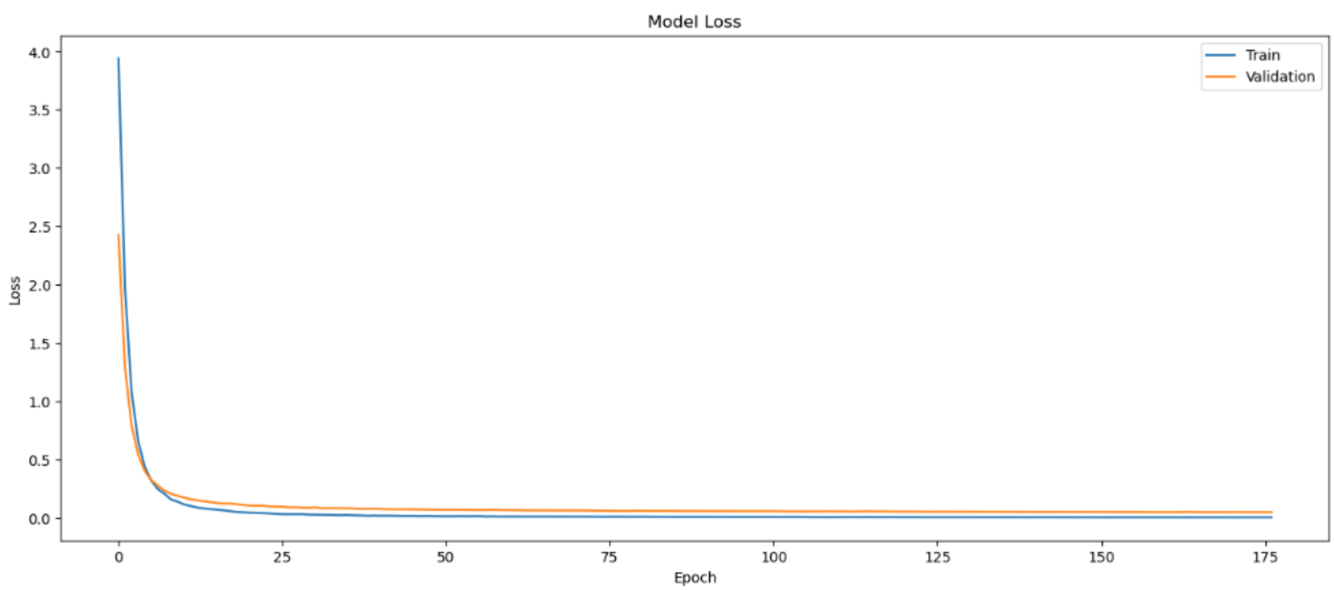
Then I experimented with different hyper-parameters as follows:

Starting with trying out different values of the batch size and the other default values

1) Batch_size = 30

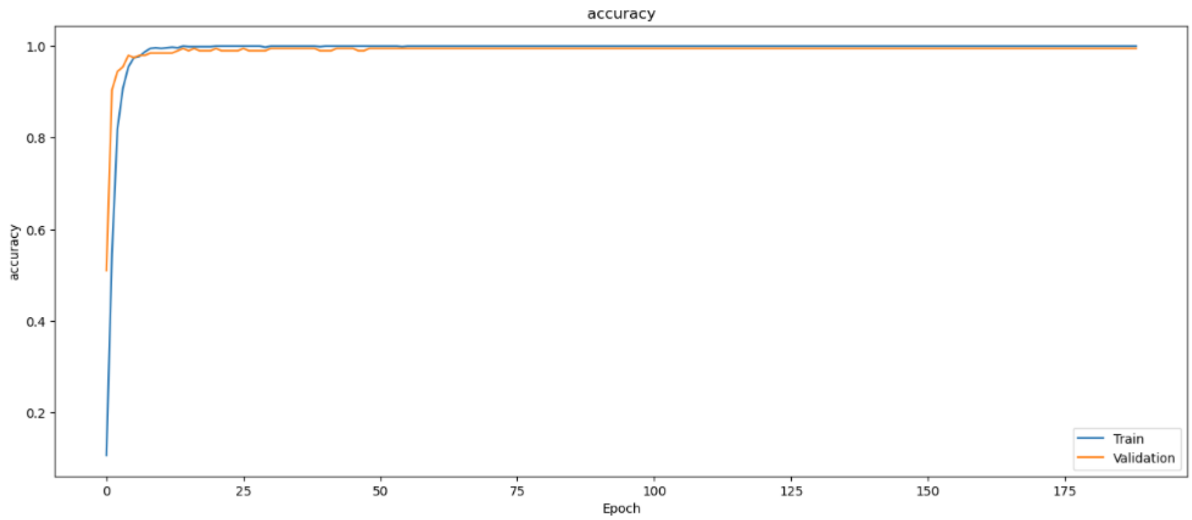
Results: loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.0462 - val_accuracy: 0.9949

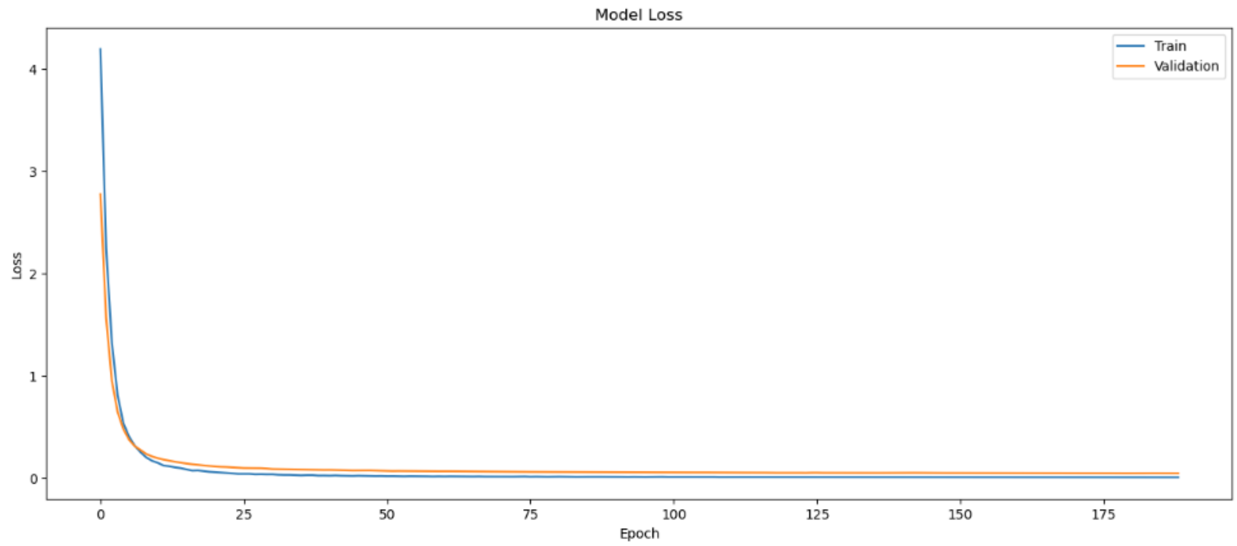




2) Batch_size = 40

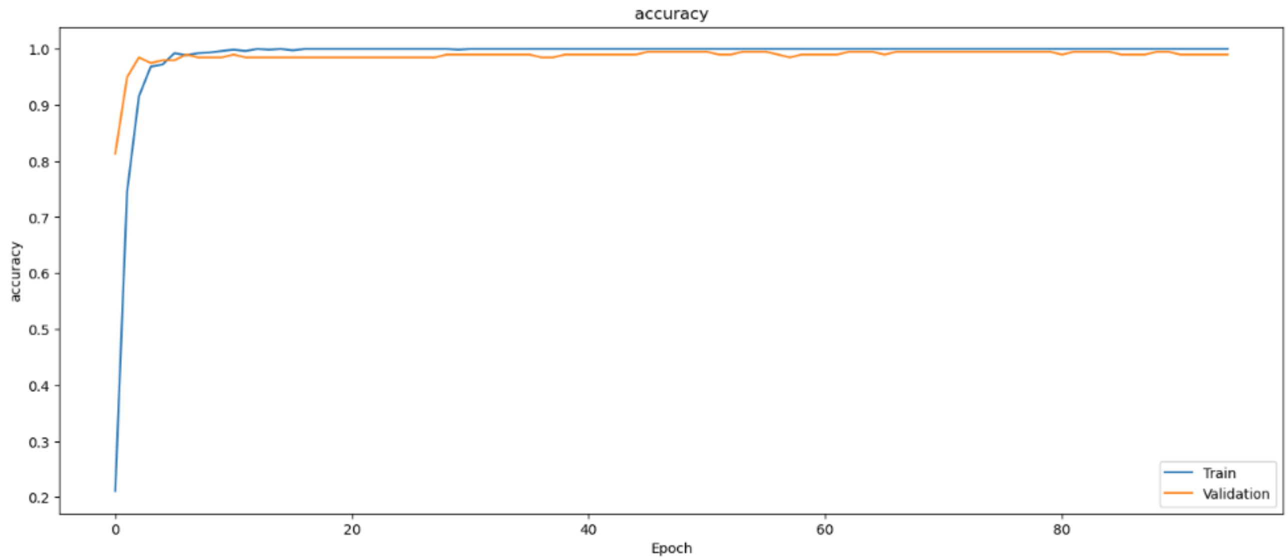
Results: loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.0419 - val_accuracy: 0.9949

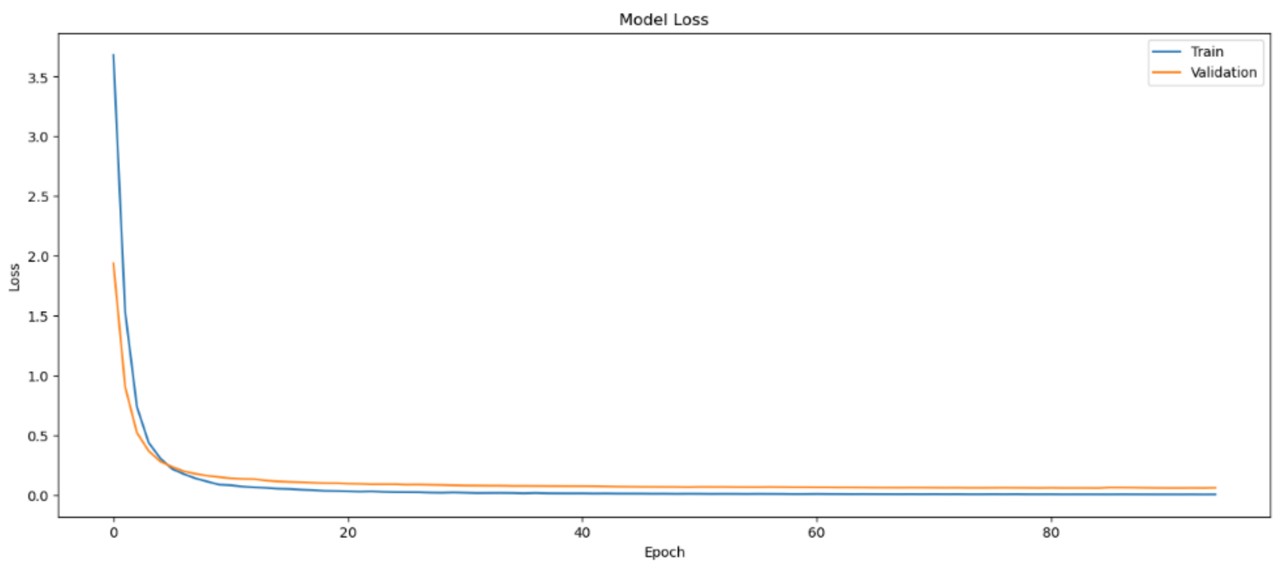




3) Dropout = 0

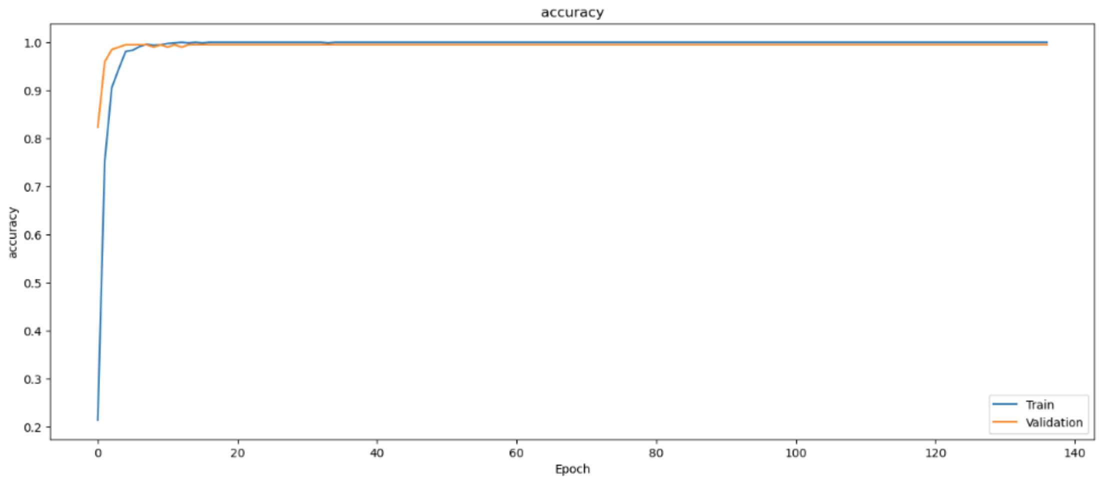
```
Results: loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.0588 -
val_accuracy: 0.9899
```

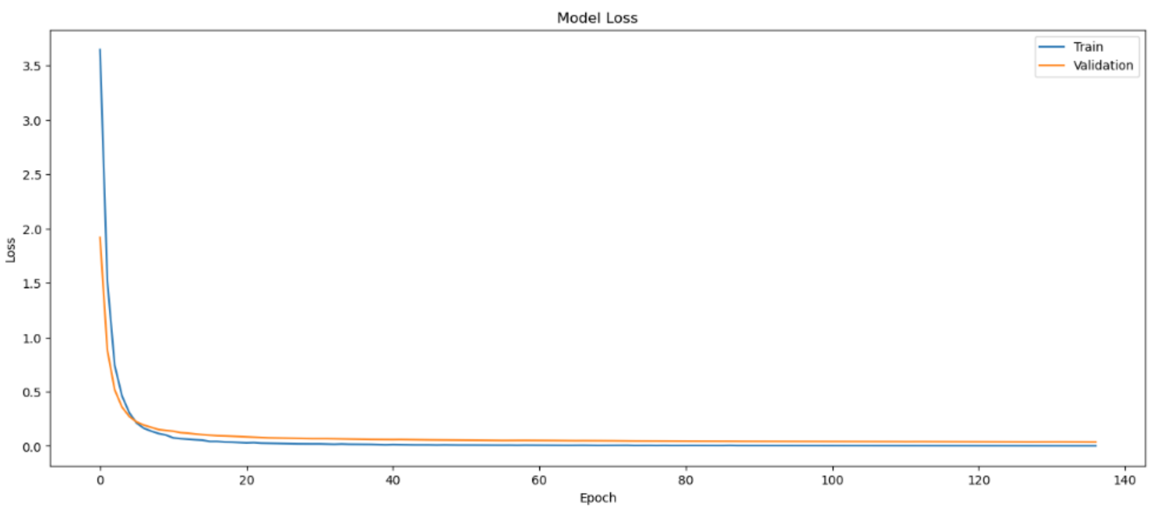

accuracy
accuracy
Epoch
Train
Validation
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80


Model Loss
Loss
Epoch
Train
Validation
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80

4) Dropout = 0.2

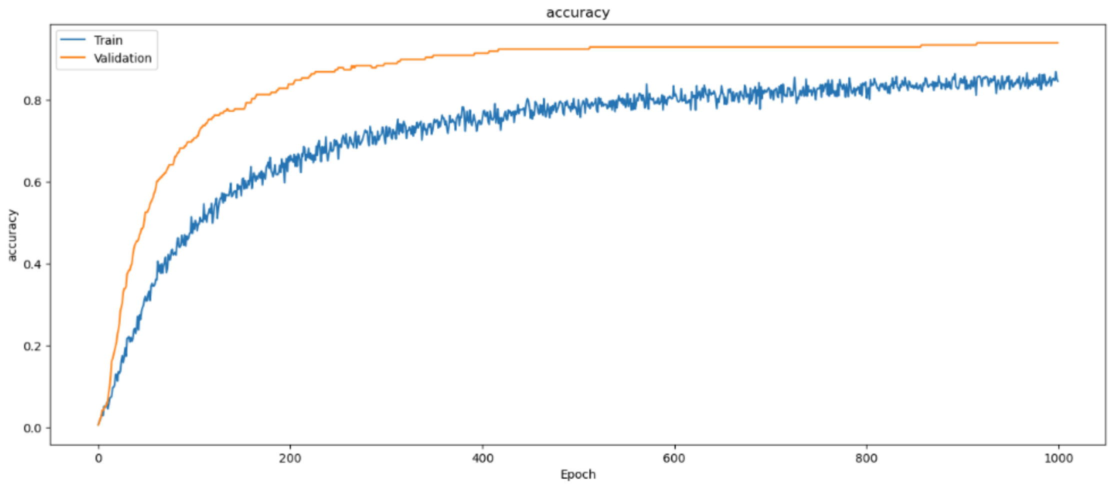
```
Results: loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0365 -
val_accuracy: 0.9949
```

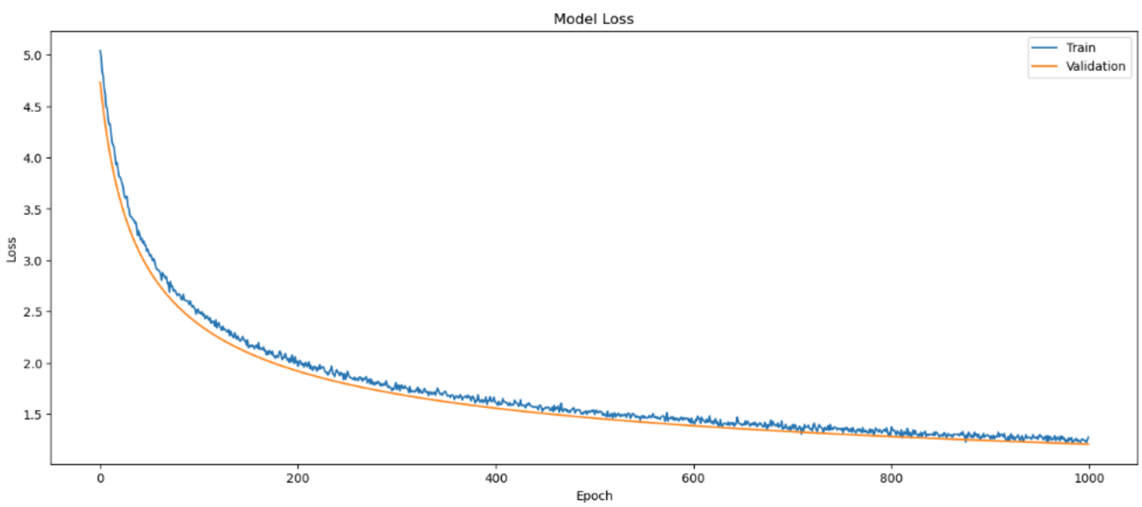




5) Optimizer = SGD

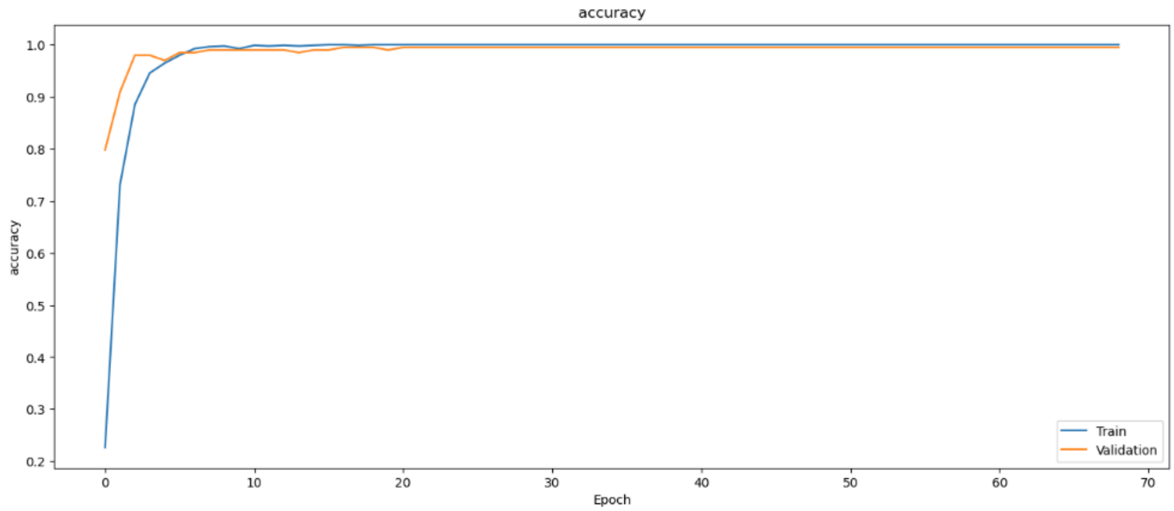
```
Results: loss: 1.2765 - accuracy: 0.8460 - val_loss: 1.2086 –
val_accuracy: 0.9394
```

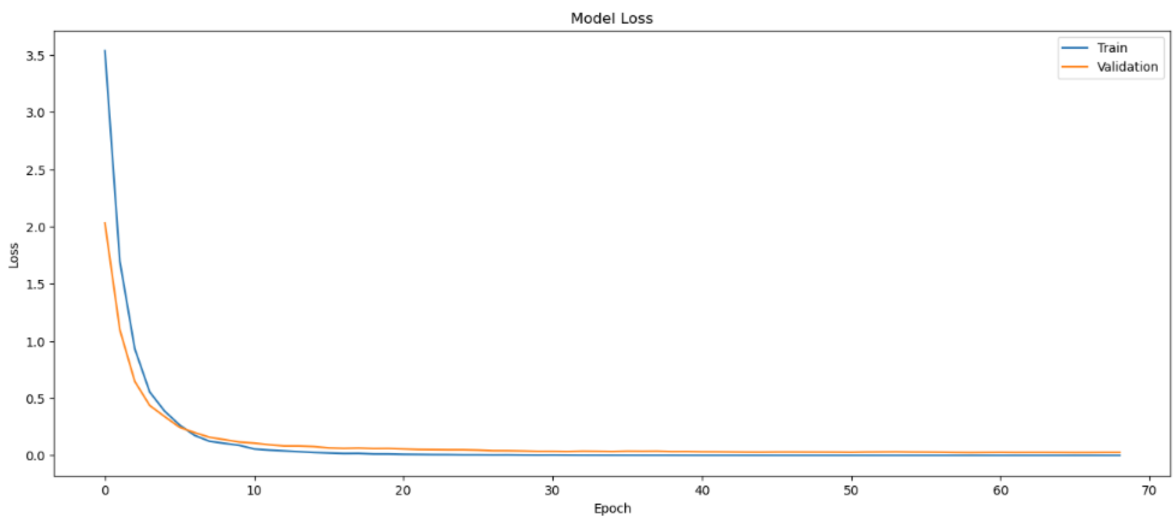




6) Optimizer = RMSprop

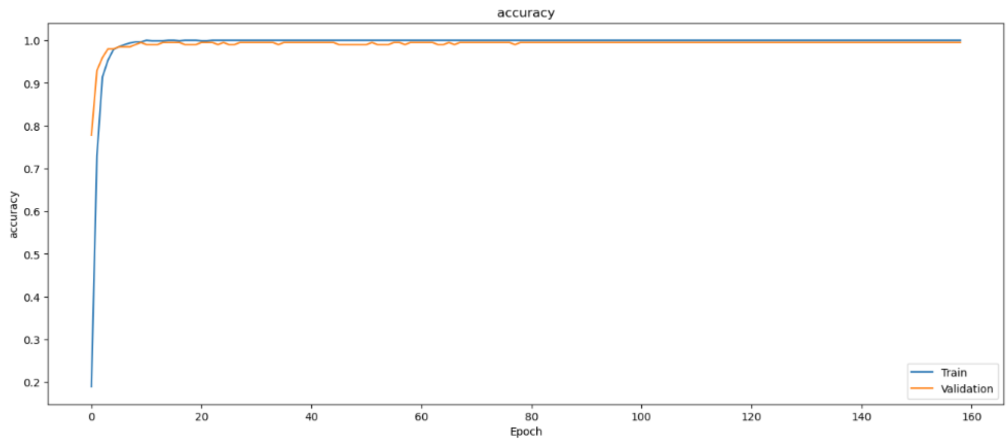
```
Results: loss: 1.0290e-04 - accuracy: 1.0000 - val_loss: 0.0260 - val_
accuracy: 0.9949
```

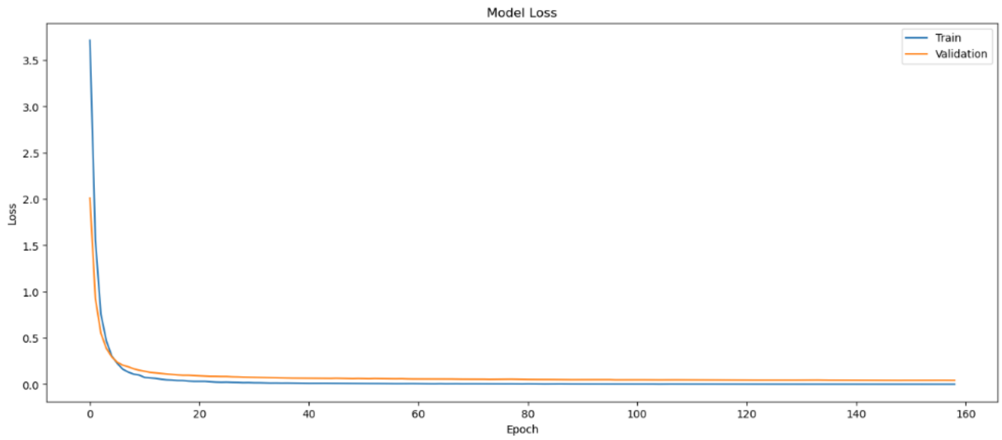




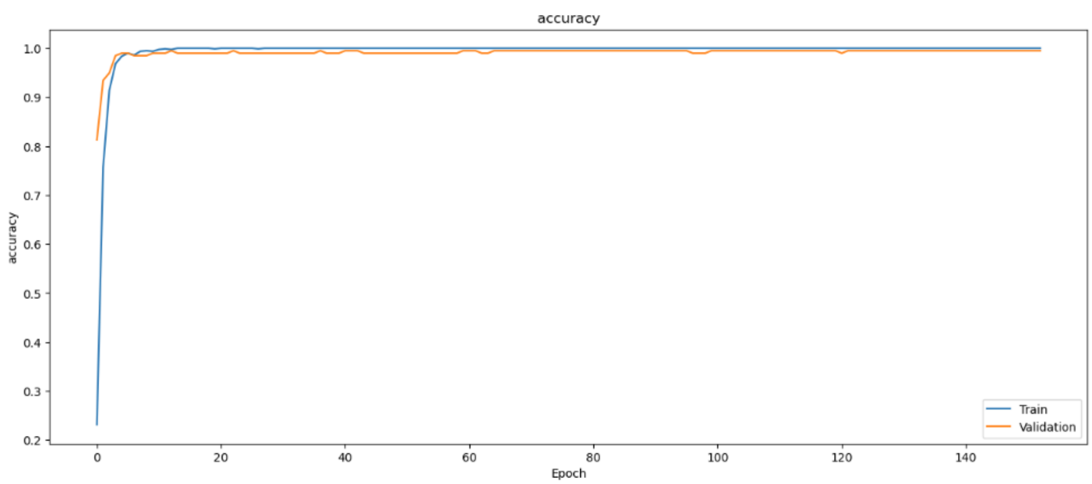
7) hidden_layer = 256

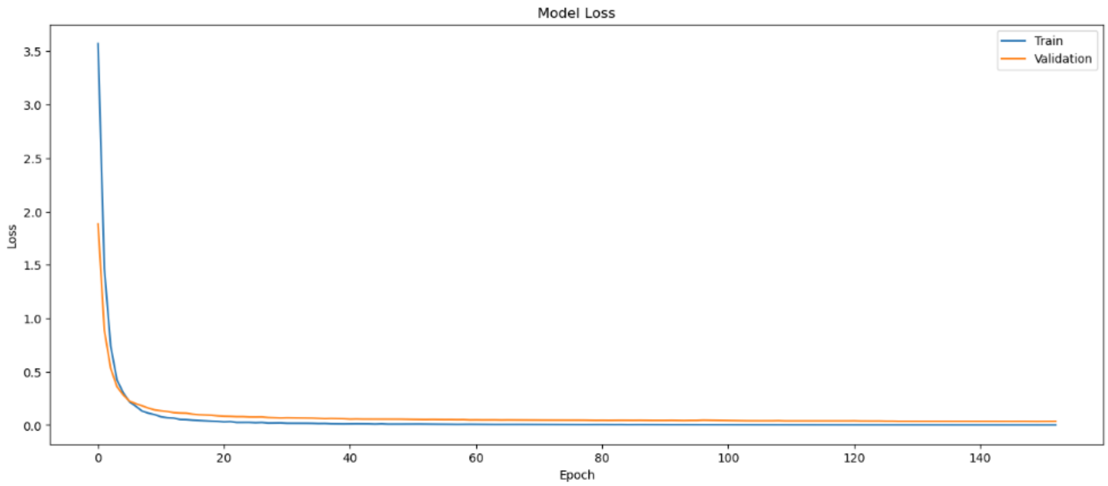
```
Results: loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.0435 –
val_accuracy: 0.9949
```






```
8) hidden_layer = 180
Results: loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.0352 –
val_accuracy: 0.9949
```

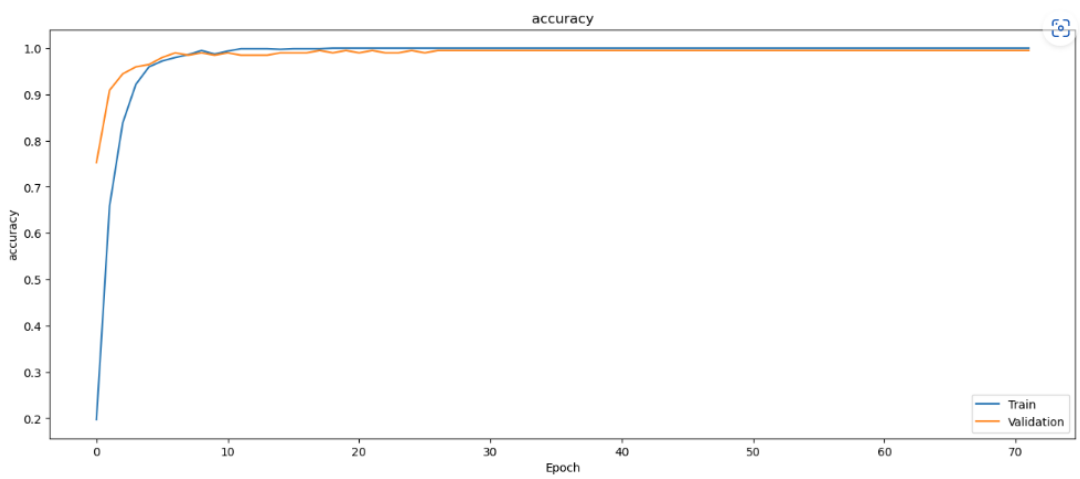


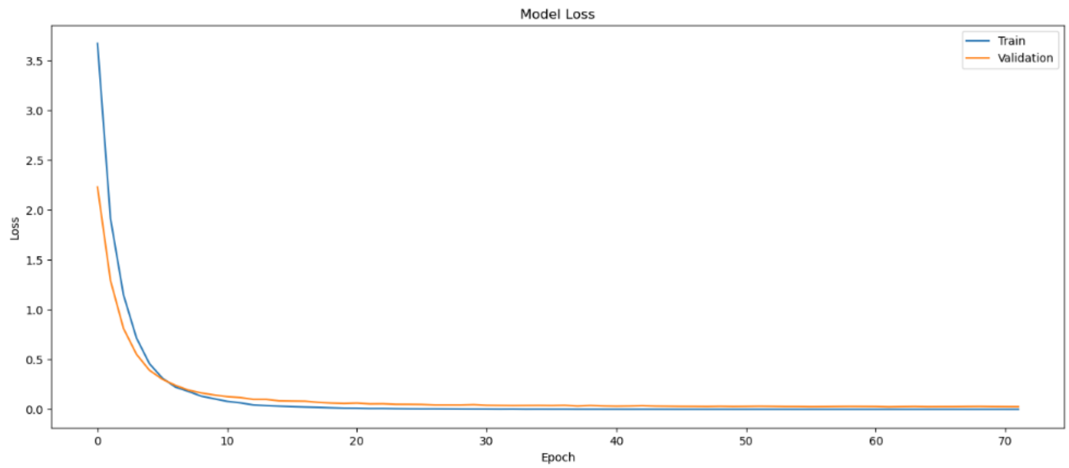


Now we'll build a model with the best hyper-parameters from the previous models which is a trivial approach.

```
hidden_layer = 180, optimizer = "RMSprop", dropout = 0.2, batch_size = 30
```

```
Results: loss: 3.4291e-05 - accuracy: 1.0000 - val_loss: 0.0266 –
val_accuracy: 0.9949
```





Finally, we'll predict the best combination of hyper-parameters using talos library which is equivalent to the grid search

```
In [40]: number_of_combinations = 20

         # Different Combinations of hyperparameters
         p = {
             "hidden_layer" : [384, 256, 180],
             "dropout" : [0, 0.2, 0.5],
             "optimizer" : ["SGD", "adam", "rmsprop"],
             "batch_size" : [16, 30, 40]
         }
         filepath = 'model8.hdf5'
         checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

         h = ta.Scan(X_train,
                     y_train,
                     params = p,
                     model = build_model,
                     experiment_name = "talos-scan",
                     x_val = X_valid,
                     y_val = y_valid,
                     round_limit = number_of_combinations,
                     print_params = True,
                     disable_progress_bar = True)
```

Results:

```
In [41]: l_model = load_model("model8.hdf5")
         l_model.evaluate(X_valid,y_valid)

         7/7 [==============================] - 0s 1ms/step - loss: 0.0624 - accuracy: 0.9949

Out[41]: [0.06244494393467903, 0.9949495196342468]
```

Conclusion: ironically, the trivial approach outperformed the fine-tuned one