CISC 867 Project 1:

Leaf Classification dataset using a Neural Network Architecture

Prepared By:

[Esraa Elsayed] **ID: 21ESMS**

[Fatma Bleity] **ID: 21FKMA**

[Gehan Essa] **ID: 21GHHE**

Submitted to:

[Dr. Hazem Abbas]

Copyright @Template.net

Table of content

Copyright @Template.net

Table of figures:

Copyright @Template.net

1. Introduction

1.1. Objective of the report

Designing different Neural Network models to classify the type of leafs based on different features by implementing a 3-layer multi-layer perceptron 'MLP' model.

- One input layer.
- One hidden layer with tanh function.
- One output layer.

1.2. The Dataset Used

The dataset contains 192 columns, major variables that will be having an impact that help to identify the 99 species of the leaf.

Copyright @Template.net

2. The libraries

We used some important libraries in python to help us for building the classifiers that shown in the figure below.

```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     import tensorflow as tf
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from keras.layers import Dropout
     import warnings
     warnings.filterwarnings('ignore')
```

Figure 1. Import Libraries

1. Numpy

The name “Numpy” stands for “Numerical Python”. It is the commonly used library. It is a popular machine learning library that supports large matrices and multi- dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library.

2. Pandas

Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.

3. Matplotlib

This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.

4. SciKit-learn

It is a famous Python library to work with complex data. Scikit-learn isan open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy.

5. Tensorflow

It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow, and used for fast numerical computing.

6. Keras

Copyright @Template.net

It is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models.

Copyright @Template.net

3. Part I

3.1 Data Preparation

3.1.1 Download the data file and load it

In [2]:
```
data_train = pd.read_csv('train.csv',index_col='id')
data_train
```

Out[2]:

id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	...	texture55	texture56	texture57	textur
1	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	0.001953	...	0.007812	0.000000	0.002930	0.002
2	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	0.000000	...	0.000977	0.000000	0.000000	0.000
3	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	0.000000	...	0.154300	0.000000	0.005859	0.000
5	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	0.013672	...	0.000000	0.000977	0.000000	0.000
6	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	0.000000	...	0.096680	0.000000	0.021484	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1575	Magnolia_Salicifolia	0.060547	0.119140	0.007812	0.003906	0.000000	0.148440	0.017578	0.0	0.001953	...	0.242190	0.000000	0.034180	0.000
1578	Acer_Pictum	0.001953	0.003906	0.021484	0.107420	0.001953	0.000000	0.000000	0.0	0.029297	...	0.170900	0.000000	0.018555	0.000
1581	Alnus_Maximowiczii	0.001953	0.003906	0.000000	0.021484	0.078125	0.003906	0.007812	0.0	0.003906	...	0.004883	0.000977	0.004883	0.027
1582	Quercus_Rubra	0.000000	0.000000	0.046875	0.056641	0.009766	0.000000	0.000000	0.0	0.037109	...	0.083008	0.030273	0.000977	0.002
1584	Quercus_Afares	0.023438	0.019531	0.031250	0.015625	0.005859	0.019531	0.035156	0.0	0.003906	...	0.000000	0.000000	0.002930	0.000

990 rows × 193 columns

Figure 2. Reading the data

We use 'read_csv' function to read the train and test data file.

As we saw after running the data it consists of **990 rows × 193 columns.**

We use the Leaf Classification dataset; we download if from Kaggle and read it to prepare data.

3.1.2 Describe the data

The data used in this project will help to accurately identify 99 species of plants based on some features, including shape, margin & texture.

Copyright @Template.net

3.1.3 Clean the data

According to **Garbage in Garbage out** phrase:

If I let the dataset full of Nulls so, our model will not train well and gives rubbish output.

So, we checked if there is any missing or duplicated values in our dataset to avoid it.

Check if there is null values or duplicates rows

```
In [31]: print('number of null values in data_train = ', data_train.isnull().any().sum())

number of null values in data_train =  0

In [33]: print('number of duplicated values in data_train = ', data_train.duplicated().any().sum())

number of duplicated values in data_train =  0
```

Figure 3. Check missing and duplicated value

As shown in the figure above:

Fortunately, our dataset doesn't contain any missing or duplicated rows.

Copyright @Template.net

3.1.4 Visualize the data using proper visualization methods

1. Plot the histogram for the features.

Data visualization

```
#plot histogram for features
data_train.hist(bins=10, figsize=(30,20))
plt.show()
```

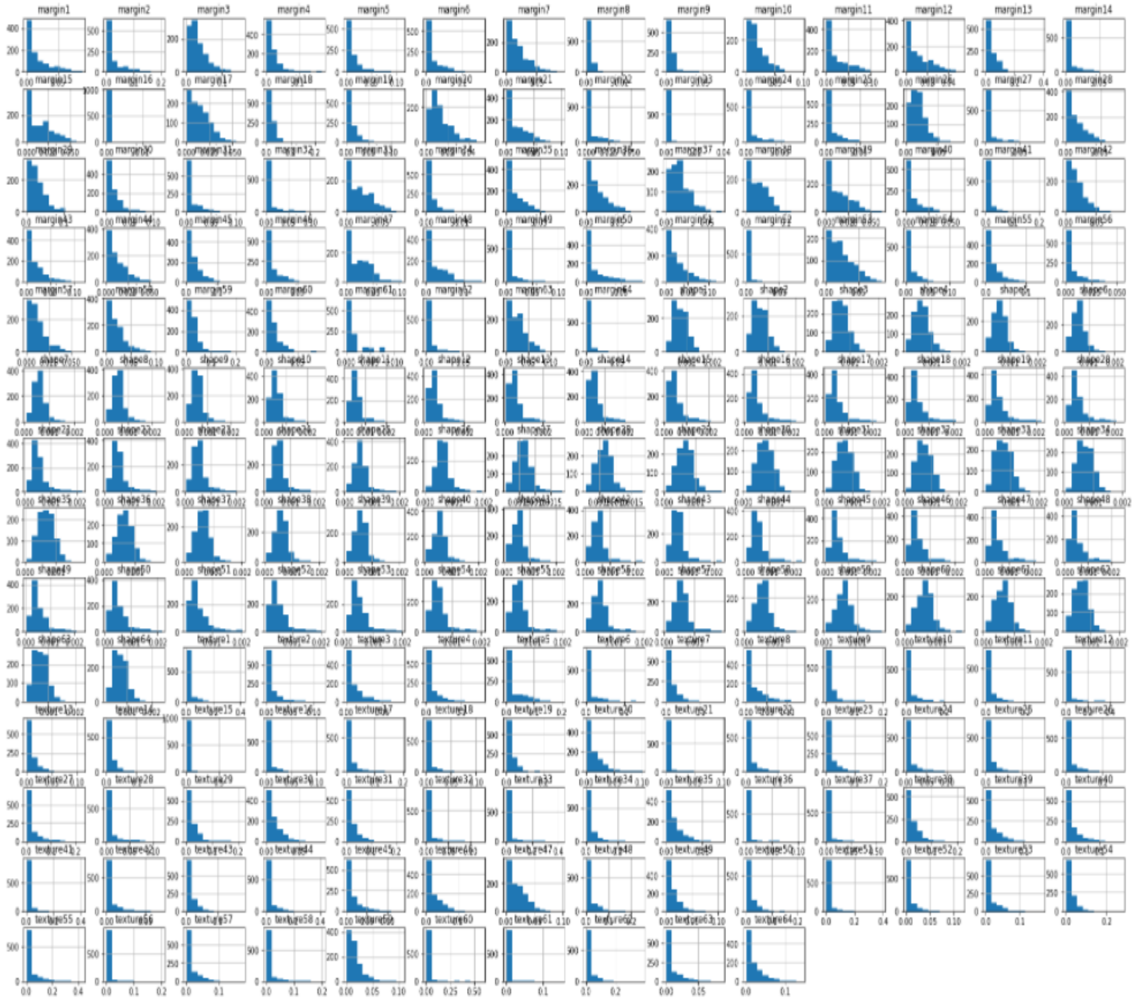


Figure 4. Visualize the data using histogram.

Copyright @Template.net

2. Using box plot to visualize features and check whether there is outliers or not.

```
# Box plot to see whether there is outliers or not
data_train.plot(kind='box')
plt.show()
```

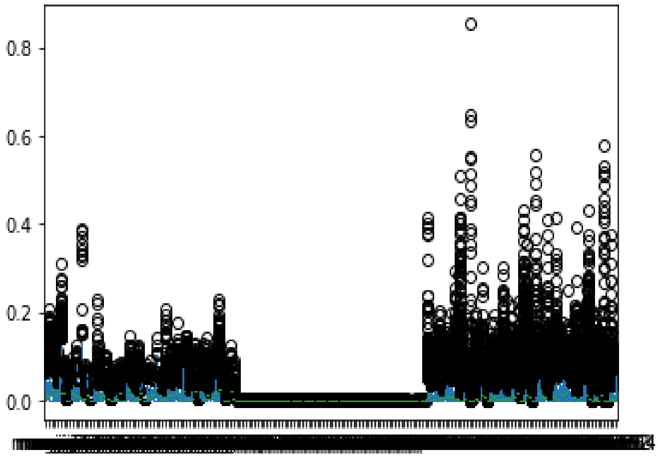


Figure 5. Box plot for all features

As we can see it is very crowd and not obvious, so we will plot the box plot for some of the features.

```
# Box plot for some features
boxplot = data_train.boxplot(column=['margin3', 'shape15', 'texture25'])
```

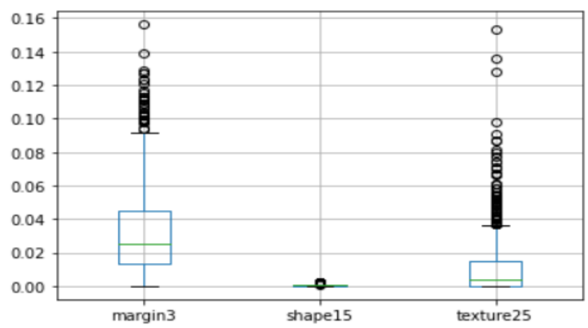


Figure 6. Box plot for some of the features

Copyright @Template.net

3.1.5 Draw random images

```
import os
os.listdir()
os.chdir('images')
from keras.preprocessing.image import load_img
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 15))

for i in range(25):
    randImg=np.random.choice(os.listdir())
    plt.subplot(5,5,i+1)
    img=load_img(os.path.join('C:\\Users\\Lab 2\\images', randImg))
    plt.imshow(img)
```

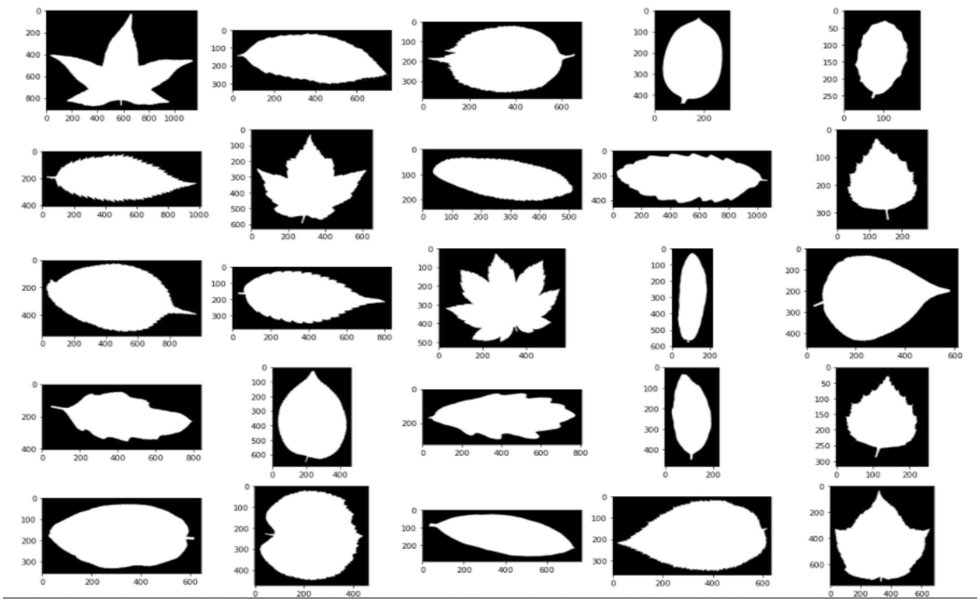


Figure 7. 25 random images

Copyright @Template.net

3.1.6 Carry out required correlation analysis.

A **correlation matrix** is a table that displays the coefficients of correlation between variables. It can show whether or not two variables are correlated and how strongly they are related. The correlation between two variables is shown in each cell of the table.

```
# Correlationn
corr = data_train.corr()
corr.style.background_gradient(cmap='coolwarm')
```

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	margin11	margin12	margin13	margin14
margin1	1.000000	0.806390	-0.182829	-0.297807	-0.475874	0.767718	0.066273	-0.094137	-0.181496	0.397138	0.737461	-0.528224	0.489317	-0.370460
margin2	0.806390	1.000000	-0.204640	-0.315953	-0.444312	0.825762	-0.083273	-0.086428	-0.120276	0.162587	0.805064	-0.489808	0.647166	-0.316377
margin3	-0.182829	-0.204640	1.000000	0.120042	-0.185007	-0.163976	0.095449	0.024350	-0.000042	0.008772	-0.261371	-0.004085	-0.048698	0.095701
margin4	-0.297807	-0.315953	0.120042	1.000000	0.029480	-0.261437	-0.268271	-0.047693	0.227543	-0.173986	-0.172503	-0.202576	-0.238041	0.338136
margin5	-0.475874	-0.444312	-0.185007	0.029480	1.000000	-0.438587	-0.108178	0.056557	0.196745	-0.320647	-0.514981	0.373683	-0.463328	0.095697
margin6	0.767718	0.825762	-0.163976	-0.261437	-0.438587	1.000000	-0.093780	-0.112896	-0.136961	0.215141	0.686998	-0.479464	0.539807	-0.317465
margin7	0.066273	-0.083273	0.095449	-0.268271	-0.108178	-0.093780	1.000000	0.099867	-0.350804	0.649311	-0.069978	-0.144810	-0.116093	-0.357485
margin8	-0.094137	-0.086428	0.024350	-0.047693	0.056557	-0.112896	0.099867	1.000000	-0.071887	0.012918	-0.108453	0.044335	-0.049359	0.001100
margin9	-0.181496	-0.120276	-0.000042	0.227543	0.196745	-0.136961	-0.350804	-0.071887	1.000000	-0.337466	-0.139592	-0.065846	-0.053170	0.372013
margin10	0.397138	0.162587	0.008772	-0.173986	-0.320647	0.215141	0.649311	0.012918	-0.337466	1.000000	0.226220	-0.384797	0.043592	-0.416157
margin11	0.737461	0.805064	-0.261371	-0.172503	-0.514981	0.686998	-0.069978	-0.108453	-0.139592	0.226220	1.000000	-0.579610	0.665286	-0.373834
margin12	-0.528224	-0.489808	-0.004085	-0.202576	0.373683	-0.479464	-0.144810	0.044335	-0.065846	-0.384797	-0.579610	1.000000	-0.429504	0.207537
margin13	0.489317	0.647166	-0.048698	-0.238041	-0.463328	0.539807	-0.116093	-0.049359	-0.053170	0.043592	0.665286	-0.429504	1.000000	-0.274913
margin14	-0.370460	-0.316377	0.095701	0.338136	0.095697	-0.317465	-0.357485	0.001100	0.372013	-0.416157	-0.373834	0.207537	-0.274913	1.000000
margin15	-0.540974	-0.503059	0.050113	-0.259813	0.467991	-0.489144	0.004146	0.062293	-0.117375	-0.320361	-0.603805	0.806811	-0.441614	0.062245
margin16	-0.072127	-0.068356	-0.054076	-0.021615	0.081766	-0.065768	-0.023989	0.205817	-0.026071	-0.053492	-0.081410	0.061776	-0.068353	0.098103
margin17	0.316704	0.135000	-0.130220	-0.047704	-0.235063	0.120157	0.396388	0.025698	-0.236991	0.566792	0.226382	-0.340929	0.006105	-0.356164
margin18	0.283239	0.345410	-0.092062	0.093686	-0.431084	0.256036	-0.149460	-0.065664	-0.046305	0.059246	0.616074	-0.444259	0.693567	-0.207573
margin19	-0.234398	-0.226020	-0.164152	0.362009	0.358065	-0.267886	-0.153342	0.002255	0.231236	-0.202346	-0.164134	-0.091540	-0.232015	0.146172
margin20	0.325947	0.062345	0.012338	0.056523	-0.326563	0.159341	0.340324	-0.043785	-0.249363	0.640144	0.252003	-0.399623	0.002403	-0.380911
margin21	-0.433734	-0.421253	0.042328	-0.138539	0.066151	-0.414130	-0.008999	0.068751	0.038512	-0.252722	-0.480018	0.603524	-0.343826	0.355155
margin22	-0.404022	-0.364703	-0.282862	-0.194713	0.273729	-0.363723	-0.130686	0.041311	0.092984	-0.319991	-0.428656	0.633346	-0.372955	0.387410
margin23	-0.142871	-0.136586	-0.145334	-0.004602	0.287659	-0.126238	-0.059832	-0.034131	0.108722	-0.127245	-0.150064	0.031318	-0.122316	0.063696
margin24	-0.315616	-0.302345	-0.255676	-0.144124	0.125076	-0.312633	0.063813	0.000173	0.045558	-0.141180	-0.315524	0.348073	-0.289536	0.189039
margin25	-0.452295	-0.397290	-0.116910	0.031567	0.331684	-0.390545	-0.309087	0.010964	0.206134	-0.436080	-0.481658	0.594966	-0.388592	0.644349
margin26	-0.190360	-0.342079	0.040840	0.076747	0.194088	-0.305049	0.488261	0.072231	-0.195966	0.408622	-0.259287	-0.049417	-0.347003	-0.301767
margin27	-0.361443	-0.316699	-0.209353	-0.120713	0.259560	-0.313253	-0.217794	0.081717	0.250379	-0.339602	-0.387730	0.448173	-0.321530	0.607554
margin28	-0.558431	-0.544266	0.150049	0.442599	0.655811	-0.528369	-0.094071	0.010025	0.217763	-0.281482	-0.574279	0.236614	-0.516038	0.234408
margin29	0.079726	0.071445	0.246209	0.467451	-0.444565	0.064964	-0.236922	-0.104108	0.154272	-0.076641	0.238431	-0.365404	0.214680	0.142027
margin30	-0.246912	-0.364163	0.054936	0.264924	0.329904	-0.251711	0.236015	0.040826	-0.066184	0.259644	-0.302272	-0.022458	-0.362220	-0.189407
margin31	-0.449652	-0.402669	-0.304381	-0.167772	0.701231	-0.400837	-0.159510	0.040940	0.122828	-0.374604	-0.489504	0.627485	-0.428016	0.190413
margin32	-0.378156	-0.330662	-0.292799	-0.105741	0.648256	-0.332077	-0.212593	0.010721	0.182795	-0.351070	-0.401301	0.445824	-0.363818	0.255992
margin33	0.597343	0.373034	-0.133807	-0.048612	-0.454494	0.375701	0.256170	-0.057275	-0.235520	0.604386	0.555093	-0.534112	0.207065	-0.417283

Figure 8.Correlation analysis

Copyright @Template.net

3.2 Divide the data

The Experimental protocol used:

Hold-out method

Our model needs to be evaluated before it has been deployed. And that evaluation needs to be done on unseen data because when it is deployed, all incoming data is unseen.

In train_test_split method, training data is divided into training and testing. The training set is used to train the model, then testing set (20%) is used to estimate the performance of data.

```
#divide data for train and test
X_train, X_test, y_train, y_test = train_test_split(X_data, y, test_size=0.2, random_state=42 , stratify=y)
```

Figure 9. Splitting Data

Splitting label from the data as variable 'y':

Split labels from the data

```
In [13]: y = data_train2['species']
         y.max()

Out[13]: 98
```

Figure 10. Split the label from the data

Copyright @Template.net

3.3. Standardize the data

We applied StandardScaler Preprocessing on the data to scale the features to contribute equally in the model. But we found that the data didn’t need to scale as all the features is already in range (0, 1). Besides, it is obvious from the histograms and box plot that all values in the dataset is already in range 0 to 1.

3.4. Encode the labels

Models require all input and output variables to be numeric, which means that in case of categorical data, we must encode it to numbers before I can fit and evaluate a model.

We found that ‘species’ column does not include numerical values so we apply ‘LabelEncoder’ on it to convert categorical data into numerical ones.

Convert species column to numerical data

In [12]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.compose import ColumnTransformer
data_train2 = data_train.copy()
el = LabelEncoder()
data_train2.loc[:, 'species'] = el.fit_transform(data_train2.loc[:, 'species'])
```

Figure 11. Encode the label

Copyright @Template.net

4. Part II

4.1 Make the functions and train the model

We implemented two functions one for train our model, and one for evaluate the model and get the results.

Our model consists from 3 Multi-Layer Perceptron. First add the first layer that takes the inputs and how many neurons in the first hidden layer with activation function ‘tanh’. Then make the output layer that works with activation function ‘softmax’ as it is multi output problem so we will use this activation function, and this layer makes outputs as the number of the unique labels we have.

Then we compile the model with loss ‘sparse categorical loss’ as it is multi categorical output problem, and metrics ‘accuracy’.

Then while fitting the model, we will take part from the training dataset as validation dataset to be used in this step so we can get our results and visualize the difference between the training loss (error) and accuracy for the training part and validation part through all epochs. So, this will be an indicator for us to know more whether the model is overfitting or under fitting.

```
def our_model(hiddenLayers, batchSize, opt, dropRrate, learningRate):
    model = Sequential()
    model.add(Dense(hiddenLayers, activation="tanh", input_shape = (n_features,), name= 'layer_1'))
    model.add(Dropout(dropRrate))
    model.add(Dense(len(np.unique(y_train)), activation="softmax", name= 'output'))
    model.summary()
    opt = opt(learning_rate = learningRate)
    model.compile(optimizer=opt, loss="sparse_categorical_crossentropy", metrics='accuracy')
    history = model.fit(X_train, y_train, validation_split = 0.2, epochs = 150, batch_size = batchSize)
    return history, model
```

Figure 12. First function – training the model

Copyright @Template.net

Then the next function which is for evaluate our model we will plot two plots one for the accuracy between the training and validation, and one for loss between training and validation.
We will evaluate our model on the unseen dataset (test dataset) and get the accuracy and loss for the model.

```
def evaluate_our_model(hiddenLayers, batchSize, opt, dropRrate, learningRate, i):
    history, model = our_model(hiddenLayers, batchSize, opt, dropRrate, learningRate)
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy for trial number: '+ str(i))
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='bottom right')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss for trial number: '+ str(i))
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
    print("Evaluate the trial number: " + str(i))
    model.evaluate(X_test, y_test)
    return model
```

Figure 13. Second function – Evaluate the model

Through both function we pass some arguments to control with them in the model.

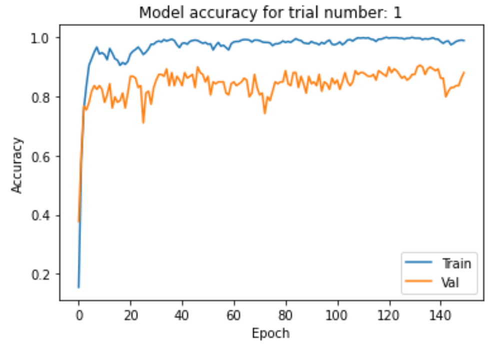
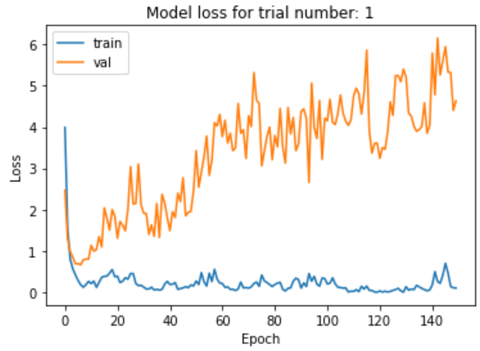
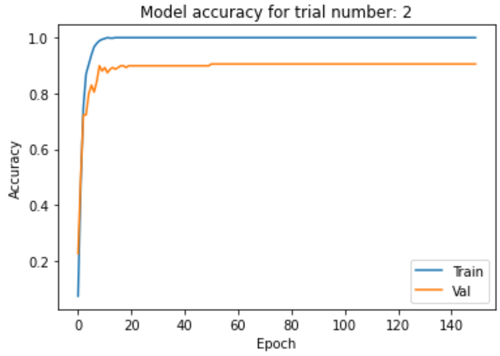
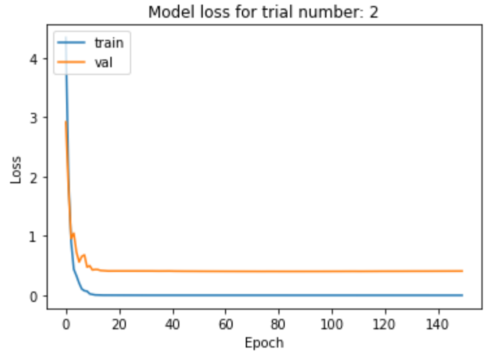
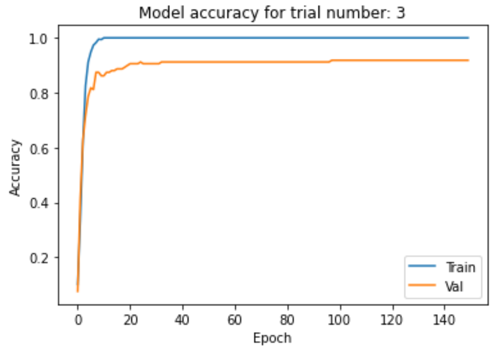
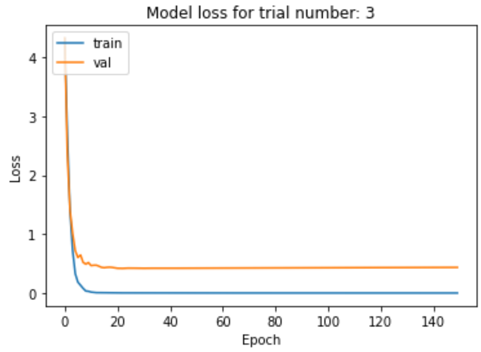
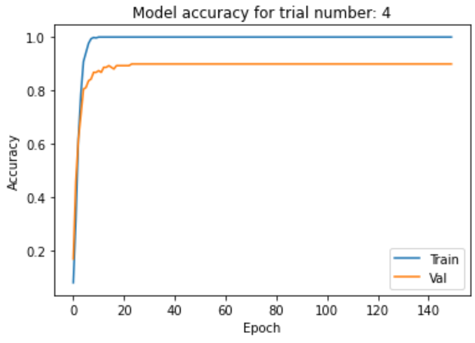
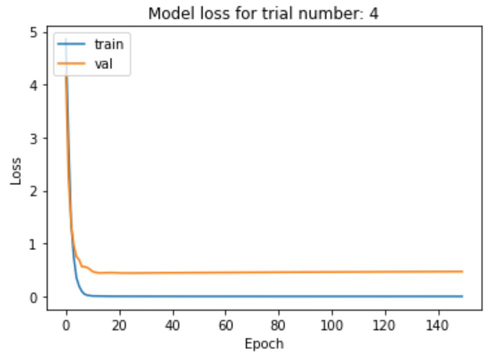
4.2 Steps

Our base model is made by arguments: hidden layers = 100, optimizer = Adam with learning rate = 0.1, dropout rate = 0.
First we will tune our batch size by starting with batch size = 32, then 64, then 128.

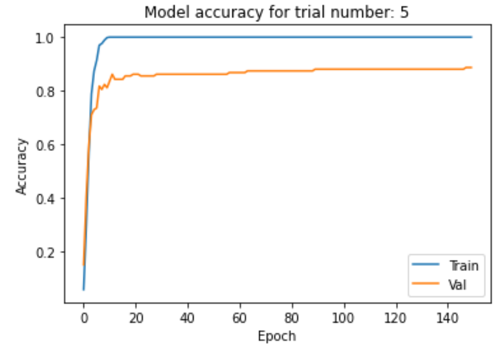
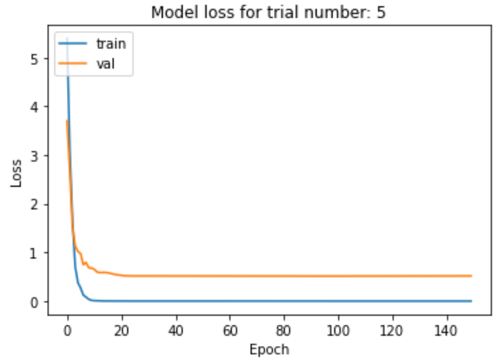
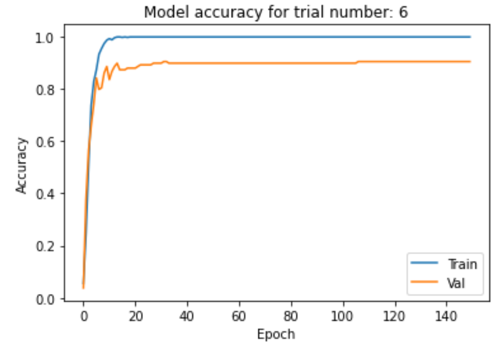
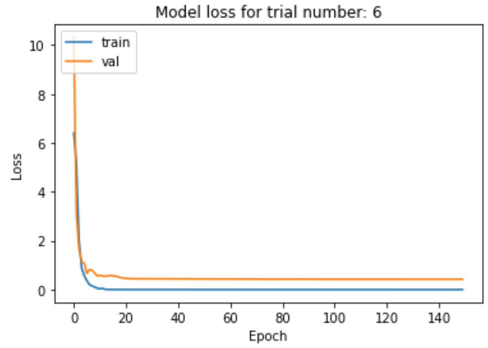
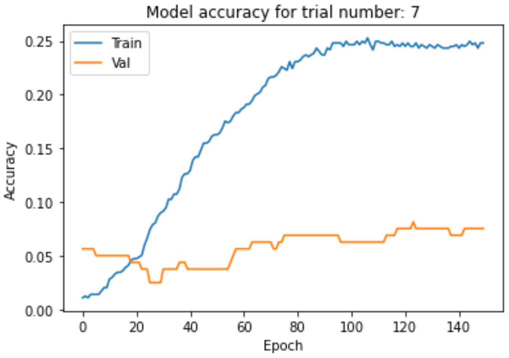
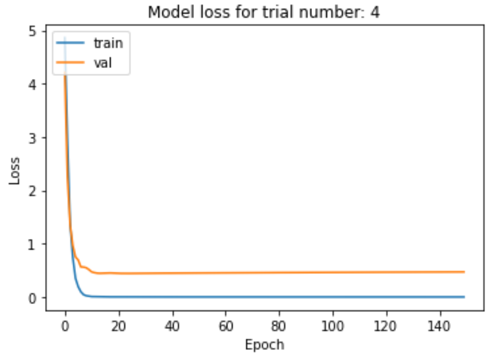
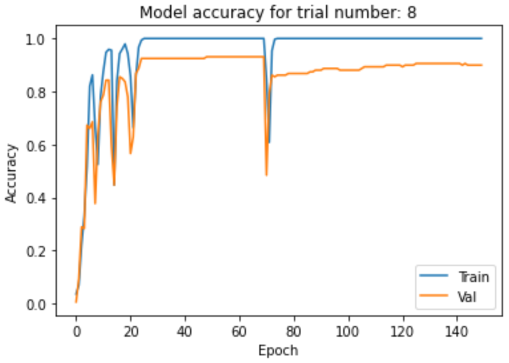
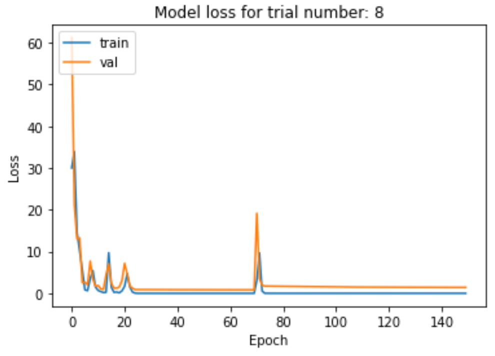
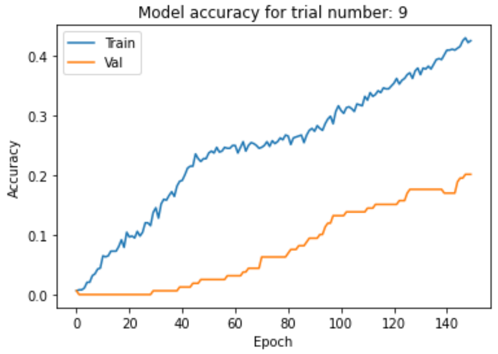
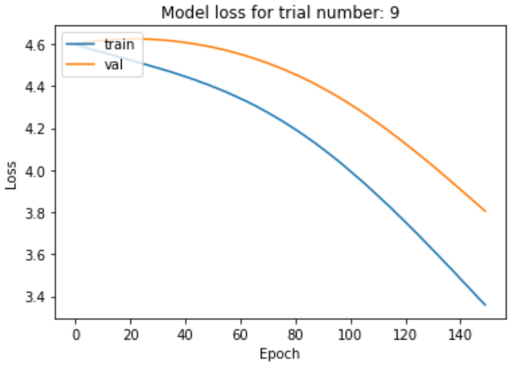
After this choose the batch size that gives the best results, and use it in tuning the next hyper parameter which is hidden layers.
We will start by hidden layers = 200, then 300, then 400, and choose the best number in the hidden layers that gives the best results in tuning the next hyper parameter which is the optimizer.
We will try RMSprop, Adadelta, and Adagrad. If any of them gives a better results than Adam which we are using in our base model, we will use it in tuning our next hyper parameter which is learning rate.
In tuning learning rate, we will try 0.01, 0.001, and 0.0001. Then we will use the one that gives best results until now to tune the dropout rate.
In the dropout rate, we will try 0.2, 0.4, and 0.5. This controls the percentage of how many neurons will turn off from the input neurons and the hidden layer neurons in each epoch.

Copyright @Template.net

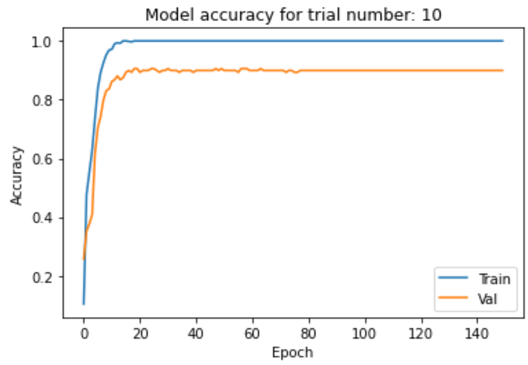
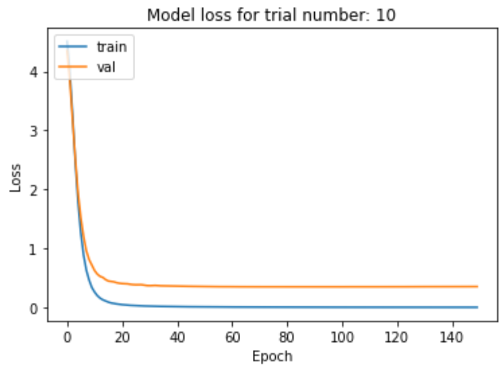
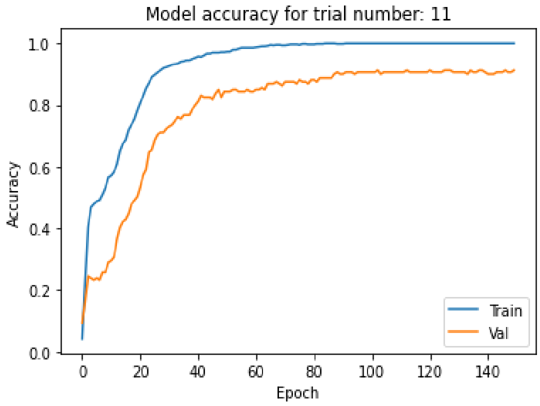
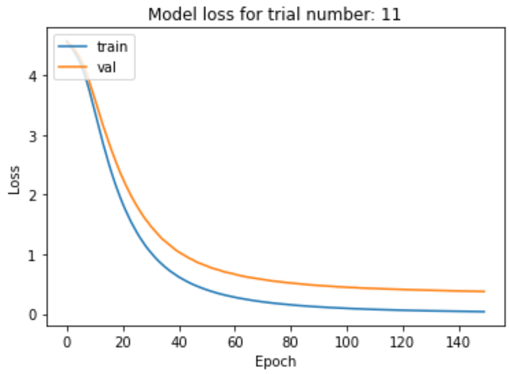
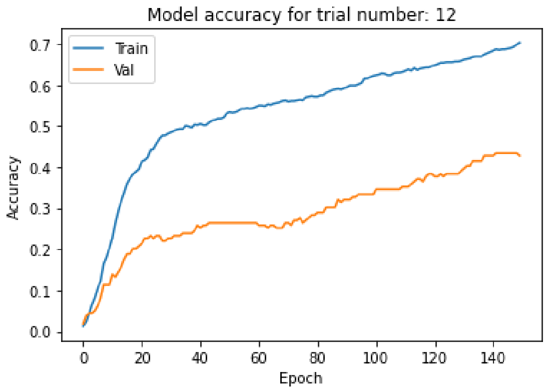
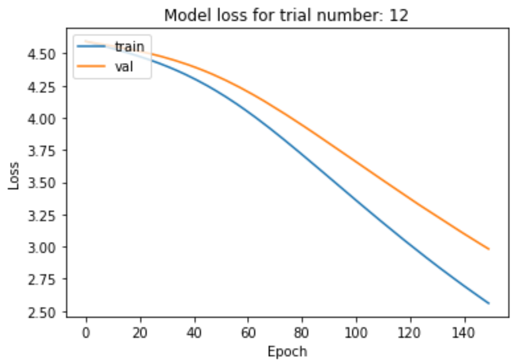
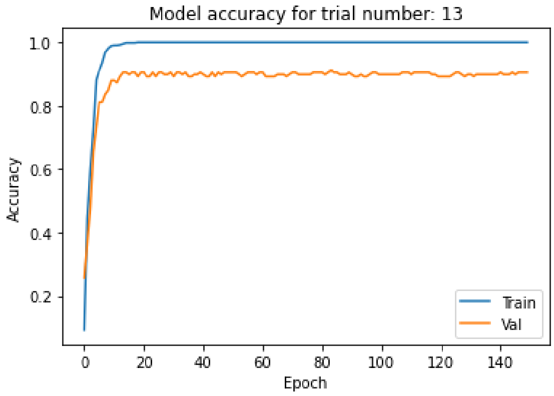
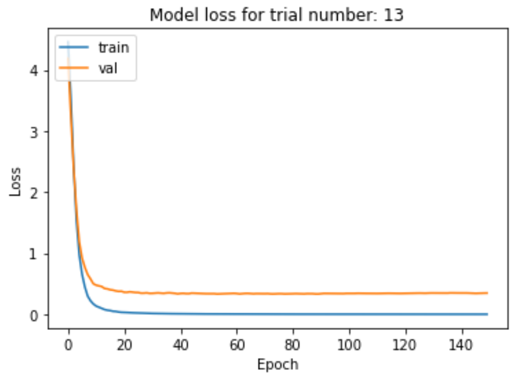
5. Results

Trial No.	Accuracy plot	Loss plot	Evaluation on test
1 Batch size = 32			loss: 4.3273 - accuracy: 0.8333
2 Batch size = 64			loss: 0.1847 - accuracy: 0.9343
3 Batch size = 128			loss: 0.1175 - accuracy: 0.9646
4 Hidden layers = 200			loss: 0.2257 - accuracy: 0.9444

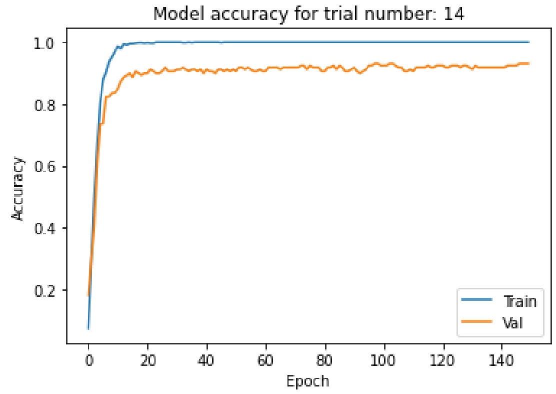
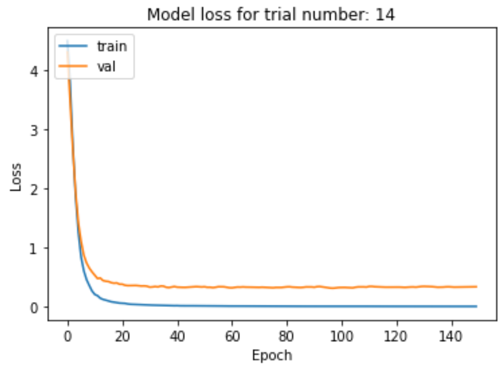
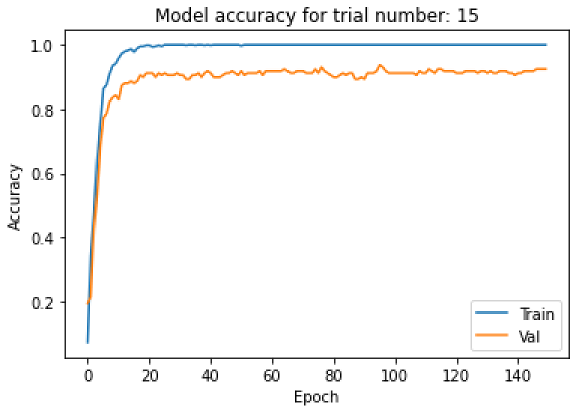
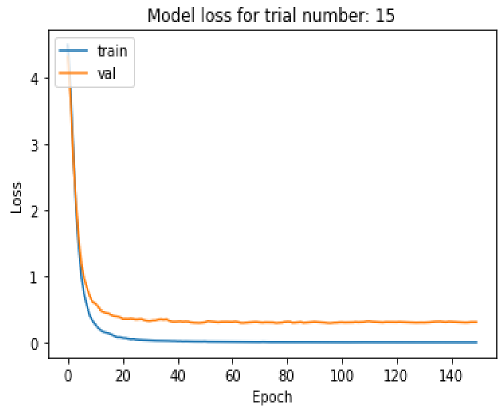
Copyright @Template.net

5 Hidden layers = 300			loss: 0.2870 - accuracy: 0.9394
6 Hidden layers = 400			loss: 0.2107 - accuracy: 0.9545
7 Adadelta oprimizer			loss: 4.4527 - accuracy: 0.1818
8 RMSprop optimizer			loss: 0.8797 - accuracy: 0.9091
9 Adagrad optimizer			loss: 3.4579 - accuracy: 0.3535

Copyright @Template.net

10 Adam Learning rate = 0.01			loss: 0.1272 - accuracy: 0.9545
11 Adam Learning rate = 0.001			loss: 0.1918 - accuracy: 0.9495
12 Adam Learning rate = 0.0001			loss: 2.6835 - accuracy: 0.5859
13 Dropout rate = 0.2			loss: 0.1294 - accuracy: 0.9596

Copyright @Template.net

14 Dropout rate = 0.4			loss: 0.1048 - accuracy: 0.9596
15 Dropout rate = 0.5	 Figures 14 for accuracy plot through the trials	 Figures 15 for loss plot through the trials	loss: 0.1129 - accuracy: 0.9545

From the shown table we can figured out that the best batch size from our trials is 128, best number of neurons in the hidden layer is 400, how bad effect of the other optimizers we tried and complete our work with Adam optimizer, the best learning rate is 0.01 and the others are very slow to reach the minima, the effect of the dropout which is turn off some of the inputs and hidden layer neurons and the best is 0.4.

So our final model will use the combination of the hyper parameters that give best results from the above which are hidden layers = 400, batch size = 128, optimizer with learning rate =

Copyright @Template.net

tf.keras.optimizers.Adam(learning_rate = 0.01), and dropout rate = 0.4.

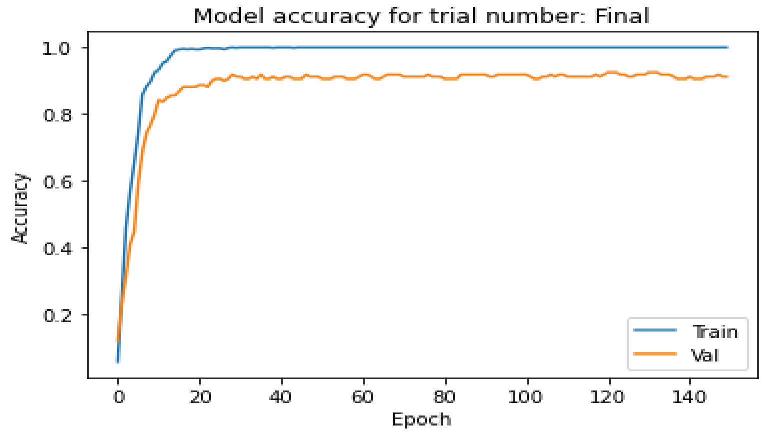


Figure 16. Accuracy plot for our final model

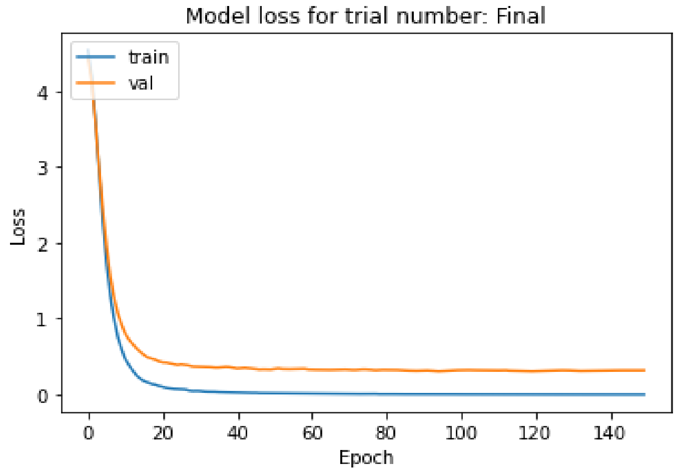


Figure 17. Loss plot for our final model

Evaluate the trial number: Final

Loss: 0.1277 - Accuracy: 0.9545

Then after this we applied this model on the dataset from the test CSV file and saved the predicted probability for each class for each test sample in a csv file to test it on Kaggle and we got score = 0.22303 loss value.

Copyright @Template.net

6.Conclusion

The neural networks is very efficient way to build strong models to be used on data to classify it. Here we used 3 MLP to classify the label for each test sample by making tuning for many hyper parameters. Hyper parameters have significant effect on the perceptron and the neural network as by small changes in them, this cause significant change in the output of the neural network through the accuracy and loss. We can see the effect of Adam optimizer in comparison to other optimizers in our problem, and how Adam is working in the best way for us. By increasing the batch size and number of neurons in the hidden layer, we got better performance as by increasing the batch size, this help the neural to take big number of samples in each epoch to update the parameters of the neural network. In addition to increasing the number of neurons in the hidden layer helps the neural network to find new features and learn more about the dataset. We can also find how the small learning rates affect the model and make it reach the minima in very big time.

7.References

https://keras.io/api/optimizers/
https://www.geeksforgeeks.org/data-visualization-with-python/

Copyright @Template.net