LEAF CLASSIFICATION PROJECT

Members:

1- Marim Ashraf Elsayed Mahmoud Amer.
2- Mawada Ashraf Elsayed Mahmoud Amer.
3- Youmna Alsayed Abdalatty Mohamed.

Leaf Classification project

Describe Data:

Plants are very important in our lives, but there are a lot of different shapes of leaves, so to determine the plant's type, we have a lot of features to know that.

1- Input: all features (margin, texture, and shape) columns.
2- Output: species column.

1- Reading data

```
[ ] #read all our traning data
df = pd.read_csv ('/content/drive/MyDrive/Queens_Practical/Deep_Learning/project1/train.csv')
df.head(5)
```

	id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	...	texture55	texture56	texture57	texture58	texture5
0	1	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	...	0.007812	0.000000	0.002930	0.002930	0.0351
1	2	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	...	0.000977	0.000000	0.000000	0.000977	0.0234
2	3	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	...	0.154300	0.000000	0.005859	0.000977	0.0078
3	5	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	...	0.000000	0.000977	0.000000	0.000000	0.0205
4	6	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	...	0.096680	0.000000	0.021484	0.000000	0.0000

5 rows × 194 columns

a.

2- Check the null values:

```
#Check if there is a null value or not
df.isnull().sum().sort_values(ascending=True)
```

```
id           0
shape58      0
shape59      0
shape60      0
shape61      0
            ..
shape3       0
shape4       0
shape5       0
margin47     0
texture64    0
Length: 194, dtype: int64
```

a.

b. There are no null values in the data.

3- Check the duplicated data:

```
#Cheek if there is a duplicate rows
print(df.duplicated(keep='first').sum())
df[df.duplicated()]
#there is no duplicate
```

```
0
```

id	species	margin1	margin2	margin3	margin4	m

a.
b. There is no duplicated data

4- There are 3 types of data in this data set: float64: 192 features, int64:1 feature, and object:1 feature.

a. int64

```
#Data has integer type (id) column
col_int = df.select_dtypes('int64')
col_int
```

i.

	id
0	1
1	2
2	3
3	5
4	6

b. float64

```
#Data has float type (margin1:margin64) and (shape1:shape64) and (texture1:texture64) columns
col_float = df.select_dtypes('float64')
col_float
```

i.

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10
0	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	0.001953	0.033203
1	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	0.000000	0.007812
2	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	0.000000	0.044922
3	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	0.013672	0.017578
4	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	0.000000	0.005859
...	...	...	...	...	...	...	...	...	...	...

c. Object

```
#Data has object type (species) column
col_object = df.select_dtypes('object')
col_object
```

i.

	species
0	Acer_Opalus
1	Pterocarya_Stenoptera
2	Quercus_Hartwissiana
3	Tilia_Tomentosa
4	Quercus_Variabilis
...	...

5- Describe data(min, max, and std)

a. To get the minimum and maximum for each column we use the describe function.

```
#To get the minimum and maximum for each column we use the describe function
df.describe()
```

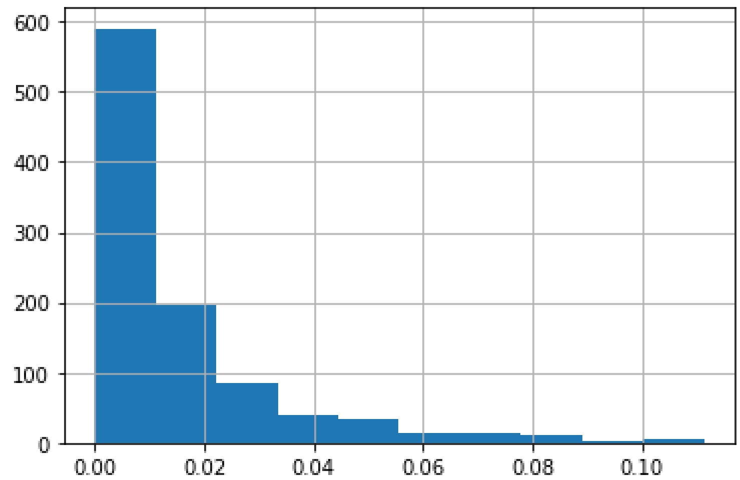
	id	margin1	margin2	margin3	margin4	margin5	margin6
count	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000
mean	799.595960	0.017412	0.028539	0.031988	0.023280	0.014264	0.038579
std	452.477568	0.019739	0.038855	0.025847	0.028411	0.018390	0.052030
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	415.250000	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000
50%	802.500000	0.009766	0.011719	0.025391	0.013672	0.007812	0.015625
75%	1195.500000	0.025391	0.041016	0.044922	0.029297	0.017578	0.056153
max	1584.000000	0.087891	0.205080	0.156250	0.169920	0.111330	0.310550

b.

6- Visualize some of the features: we display 5 plots in a notebook

```
[ ]  df['margin5'].hist()
```

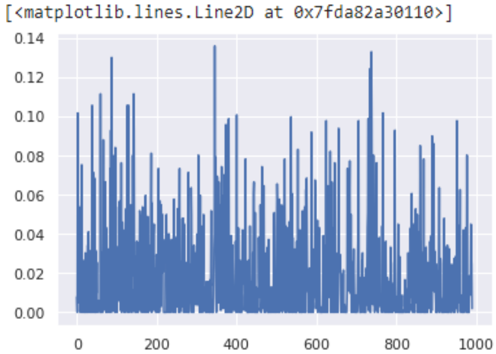
<matplotlib.axes._subplots.AxesSubplot at 0x7f6b765b6b10>

a. 

i. From the graph, margin5's values range from 0 to 580.
ii. The most value found in this column is 0.01.

```
[ ] import matplotlib.pyplot as plt # import matplotlib.pyplot to draw a plot diagram.
    plt.plot(df['texture8']) # Draw plot for texture8 features between inputs and it's number of values in each input
```

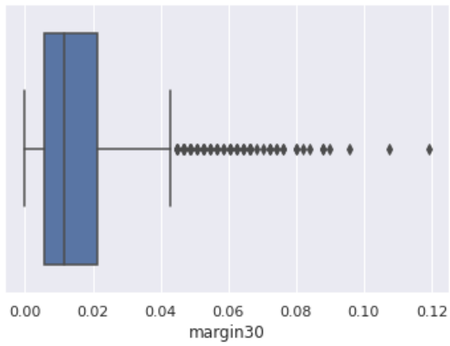
[<matplotlib.lines.Line2D at 0x7fda82a30110>]

b. 

i. The graph shows texture8's rows and rows' values range from 0 to 0.13.
ii. The maximum value found in this column is 0.13 in row 280.

```
# Draw Boxplot for margin30 features to determine outliers
sns.boxplot('margin30', data=df)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f105d2c5290>
```

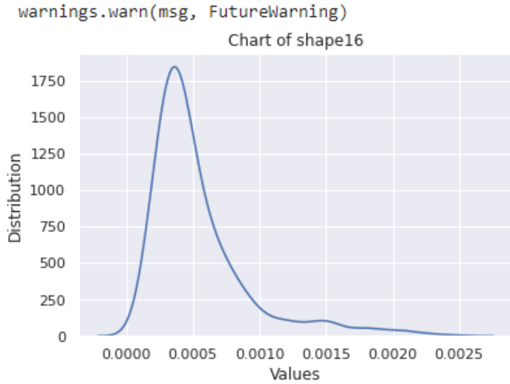


c.

i. The graph shows margin30's outliers that strat from 0.05.

```
# Draw normal distribution function for shape16 features between inputs and it's number of values in each input
sns.distplot(df['shape16'], hist = False, kde = True, label='species')
plt.title('Chart of shape16')
plt.xlabel('Values')
plt.ylabel('Distribution')
# Show the plot
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated f
  warnings.warn(msg, FutureWarning)
```



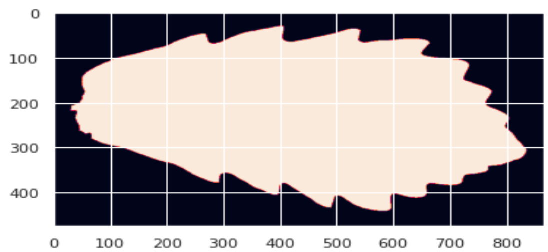
d.

i. The graph shows shape16's values and their distribution.

7- Draw some of the images: we draw 4 images

```
[ ] import matplotlib.pyplot as plt
    import matplotlib.image as mpimg

    img=mpimg.imread('/content/drive/MyDrive/Queens_Practical/Deep_Learning/project1/images/1579.jpg')
    imgplot = plt.imshow(img)
```



a.

8- Correlation analysis: to know the relation between features.

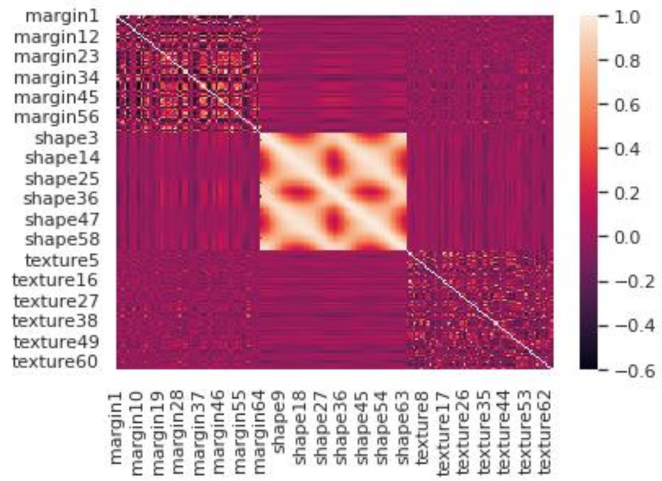
```
[ ] #Display the correlation between features by using corr function
    df.corr()
```

	id	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8
id	1.000000	-0.011673	-0.027565	-0.059533	0.001639	-0.002419	-0.051818	0.061214	-0.039509
margin1	-0.011673	1.000000	0.806390	-0.182829	-0.297807	-0.475874	0.767718	0.066273	-0.094137
margin2	-0.027565	0.806390	1.000000	-0.204640	-0.315953	-0.444312	0.825762	-0.083273	-0.086428
margin3	-0.059533	-0.182829	-0.204640	1.000000	0.120042	-0.185007	-0.163976	0.095449	0.024350
margin4	0.001639	-0.297807	-0.315953	0.120042	1.000000	0.029480	-0.261437	-0.268271	-0.047693
...	...	...	...	...	...	...	...	...	...
texture60	-0.000823	0.035072	0.081069	-0.019850	-0.052317	0.006542	0.066262	-0.034094	0.048647
texture61	0.026319	-0.007581	-0.007057	0.084957	0.320644	-0.109229	-0.050498	-0.163375	-0.079283
texture62	0.032873	-0.033159	-0.037405	-0.081999	-0.073886	0.151675	-0.031555	0.015391	-0.048843
texture63	0.024299	-0.075171	-0.098957	-0.148193	0.050970	0.022299	-0.132087	-0.001364	0.027758
texture64	0.035396	0.030414	-0.029532	0.061780	0.014343	-0.148834	-0.003164	0.068512	-0.003191

193 rows × 193 columns

a.

```
[ ]  datavis=sns.heatmap(df.corr()) # Heatmap used to visualize correlation between features
```



b.

   i. The correlation between a feature and itself equals 1.
   ii. The correlation between features and each other is a negative coefficient, which tells us that the relationship is negative, and a positive coefficient, which tells us that the relationship is positive.

9- Label Encoder: we do label encoder to categorical data and in our data species column is categorical data.

10- Split data: split data into training and testing data by using train_test_split.
   a. X will contain all features except species and the id column.
   b. y will contain species.

```
[ ]  from sklearn.model_selection import train_test_split
     y=df['species']
     X=df.drop(columns = ['species','id'])

     X_train, X_test,y_train,y_test = train_test_split(X,y, test_size=0.2)
```

   c.

11- Normalizing data: We do normalized data by using a standard scaler to be mean = 0 and std = 1.

```
from sklearn import preprocessing
scaler=preprocessing.StandardScaler()
X_train= scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)
```

   a.

12- Neural network model: In this section, we will try different hyperparameters, one hyperparameter at a time. We will choose the best value of the hyperparameter that gives us the best accuracy, then we will take it and add to it new hyperparameters, and so on until we finish all the hyperparameters that we use.
   a. Epochs in all models = 100, and for each hyperparameter, we try 3 times in each model.

13- Batch size hyperparameter: we try many values and we reached trial 2 is the best accuracy.

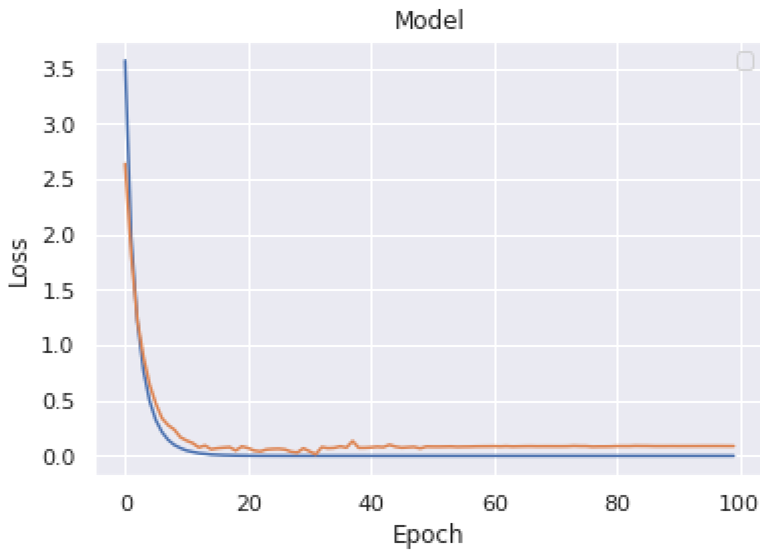
a. In this section, we will try a different number of batch sizes in each trial.

b. Trial 1: batch size = 50, test accuracy = 0.985.

c. Trial 2: batch size = 90, test accuracy = 0.980.

d. Trial 3: batch size = 6, test accuracy = 0.975.

e. We tried a lot of values, but trial 1 gave us the largest test accuracy = 0.985.



f.

14- Optimizer hyperparameters:

a. In this section, we will try different types of the optimizer in each trial.

b. Trial 1: optimizer = Adam, test accuracy = 0.965.

c. Trial 2: optimizer = RMSProp, test accuracy = 0.995.

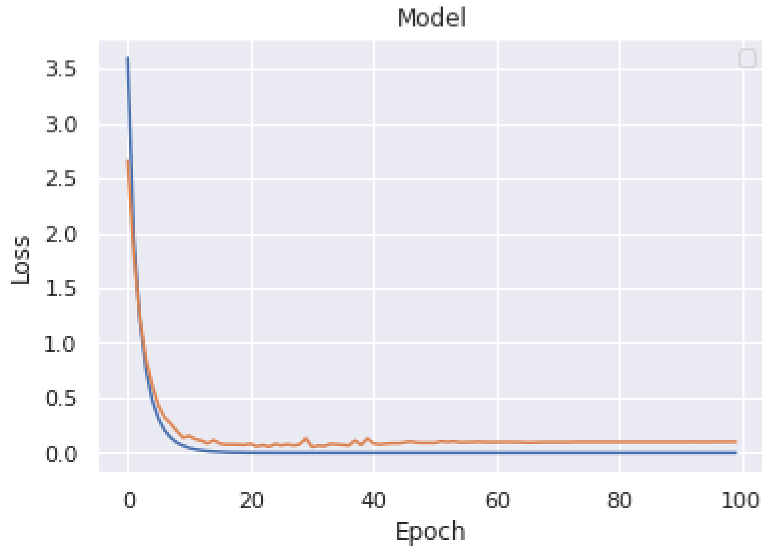
d. Trial 3: optimizer = SGD, test accuracy = 0.995.

e. We tried a lot of values, but trial 2 gave us the largest test accuracy = 0.995 “RMSProp”.

f.

15- RMSProp Optimizer hyperparameters

a. In this section, we will try a different Learning rate of the RMSprop optimizer in each trial.
b. Trial 1: LR = 0.001, test accuracy = 0.985.
c. Trial 2: LR = 0.01, test accuracy = 0.975.
d. Trial 3: LR = 0.0001, test accuracy = 0.975.
e. the largest test accuracy = 0.985 (Trial 1)



f.

16- Regularization: it is used to prevent overfitting.

a. We did not have overfitting in the data so we used very small values to decay.
b. we tried a different weight decay for the L2 regularizer in each trial.
c. Trial 1: wd = 0.12, test accuracy = 0.965.
d. Trial 2: wd = 0.0013, test accuracy = 0.990.
e. Trial 3: wd = 0.0004, test accuracy = 0.990.
f. We tried a lot of values, but trial 2 gave us the largest test accuracy = 0.990.

g.

17- Hidden size

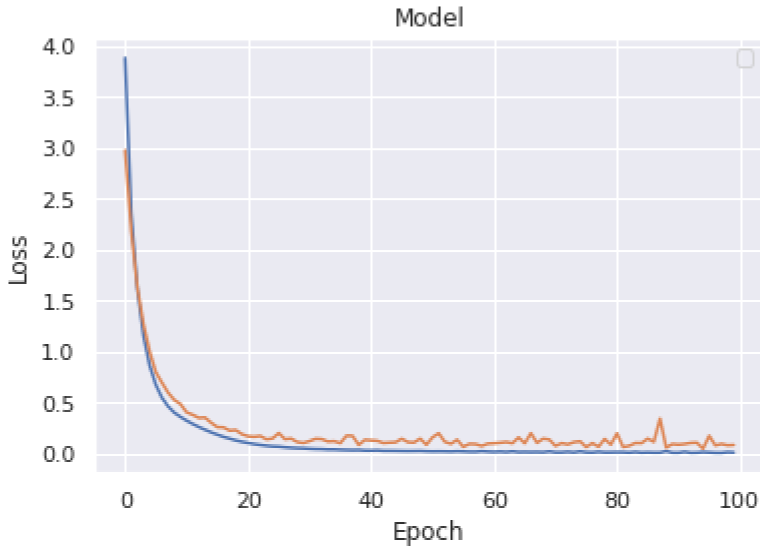
a. In this section, we will try a different hidden size in each trial.

b. Trial 1: num_node = 226, test accuracy = 0.995.

c. Trial 2: num_node = 64, test accuracy = 0.985.

d. Trial 3: num_node = 512, test accuracy = 0.990.

e. We tried a lot of values, but trial 1 gave us the largest test accuracy = 0.995.



f.

18- The final model with all hyperparameters

```
# NN_model4 Function to build the model
# It have 3 parameters
# 1- num_node parameter is number of neurons in hedden layer
# 2- wd is a weight of L2 regularizer
# 3- opLR is the learning rate of RMSprop's optimizer.
def NN_model4(num_node, wd, opLR):
  opt = tf.keras.optimizers.RMSprop(learning_rate=opLR)   # opt is a variable that has the learning rate of an RMSprop optimizer.
  model=Sequential()
  # hidden layer contains:
   # 1- 192 inputs
   # 2- 265 neurons in hidden layer
   # 3- tanh as activation function
   # kernel_regularizer is a L2 regularizer
  model.add(Dense(num_node,activation='tanh' ,input_shape=(n_features,),name='layer1', kernel_regularizer = regularizers.l2(wd)))
  # Output layer contains:
   # 1- 99 neurons in output layer
   # 2- softmax as activation function
  model.add(Dense(n_output,activation='softmax' ,name='output'))
  # loss function is a sparse_categorical_crossentropy
  # Measure the model's accuracy by accuracy
  # optimizer is RMSprop, who has a different learning rate.with learning rate = 0.01
  model.compile(loss="sparse_categorical_crossentropy",metrics='accuracy',optimizer=opt)
  return model
```

The best hyperprameters:

1- Batch size = 50.

2- Optimizer = ‘RMSProp’.

3- Optimizer ‘RMSProp’ learning rate = 0.001.

4- Regular wd= 0.00013.

5- Nodes in heddin layers = 226.