CISC 867 Project 1: Leaf Classification dataset using a neural network architecture

By:

Alhassan Ehab Ramadan Ahmed

ID:

20398553

Supervised by:

Prof. Hazem Abbas

Project objective:

- In this project, the Leaf Classification dataset was used by neural network architecture.

Problem Description:

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

Dataset Description:

- The dataset consists of 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 and finally, it contains 193 Features.

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

Part I: Data Preparation

Import libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

Read the data:

```
[20] #read data
df = pd.read_csv('/content/train.csv')
```

```
#display the first five rows from the data
df.head()
```

	id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	...	texture55	texture56	texture57	texture58	texture59	texture60	texture61	texture62	texture63	texture64
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	0.035156	0.0	0.0	0.004883	0.000000	0.025391
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	0.023438	0.0	0.0	0.000977	0.039062	0.022461
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	0.007812	0.0	0.0	0.000000	0.020508	0.002930
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	0.020508	0.0	0.0	0.017578	0.000000	0.047852
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	0.000000	0.0	0.0	0.000000	0.000000	0.031250

5 rows × 194 columns

```
[ ] #display data dimension
df.shape

(990, 194)
```

Check nulls and duplicates:

```
[ ] #check null values
print(df.isna().sum().sort_values())

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

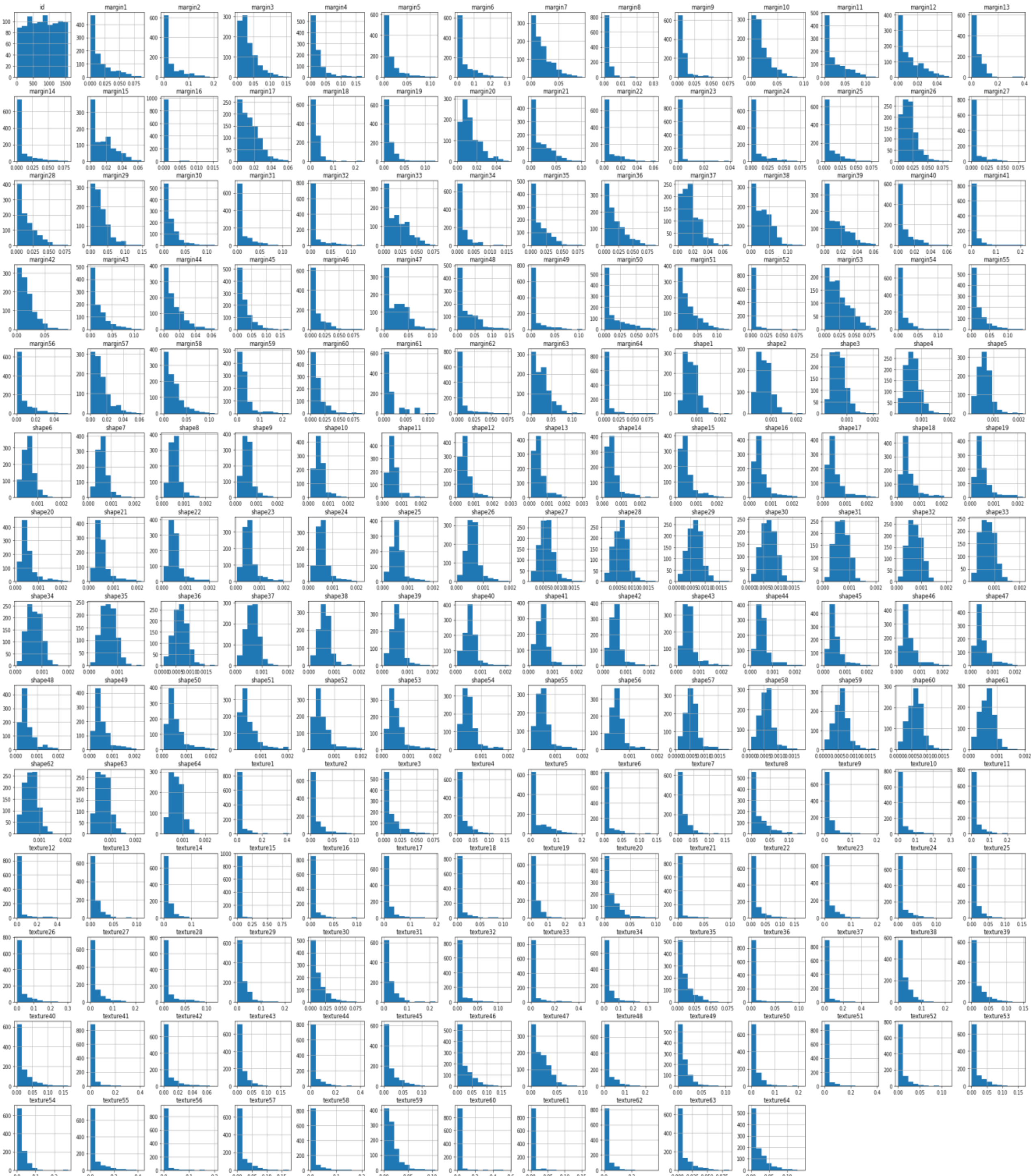
```
#check duplications and drop them if exist
df.drop_duplicates()
```

	id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	...	texture55	texture56	texture57	texture58	texture59	texture60	texture61	texture62	texture63	texture64
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	0.035156	0.000000	0.000000	0.004883	0.000000	0.025391
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	0.023438	0.000000	0.000000	0.000977	0.039062	0.022461
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	0.007812	0.000000	0.000000	0.000000	0.020508	0.002930
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	0.020508	0.000000	0.000000	0.017578	0.000000	0.047852
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	0.000000	0.000000	0.000000	0.000000	0.000000	0.031250
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
985	1575	Magnolia_Salicifolia	0.060547	0.119140	0.007812	0.003906	0.000000	0.148440	0.017578	0.0	...	0.242190	0.000000	0.034180	0.000000	0.010742	0.000000	0.000000	0.000000	0.000000	0.018555
986	1578	Acer_Pictum	0.001953	0.003906	0.021484	0.107420	0.001953	0.000000	0.000000	0.0	...	0.170900	0.000000	0.018555	0.000000	0.011719	0.000000	0.000000	0.000977	0.000000	0.021484
987	1581	Alnus_Maximowiczii	0.001953	0.003906	0.000000	0.021484	0.078125	0.003906	0.007812	0.0	...	0.004883	0.000977	0.004883	0.027344	0.016602	0.007812	0.000000	0.027344	0.000000	0.001953
988	1582	Quercus_Rubra	0.000000	0.000000	0.046875	0.056641	0.009766	0.000000	0.000000	0.0	...	0.083008	0.030273	0.000977	0.002930	0.014648	0.000000	0.041992	0.000000	0.001953	0.002930
989	1584	Quercus_Afares	0.023438	0.019531	0.031250	0.015625	0.005859	0.019531	0.035156	0.0	...	0.000000	0.000000	0.002930	0.000000	0.012695	0.000000	0.000000	0.023438	0.025391	0.022461

990 rows × 194 columns

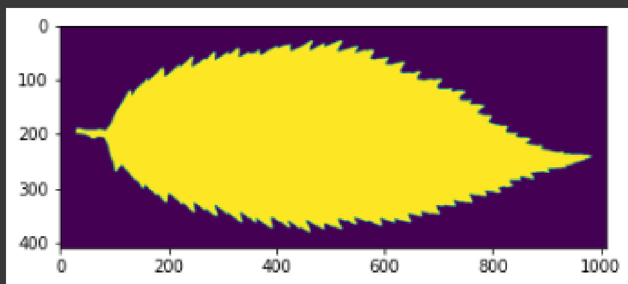
```
[ ] #no duplications beause the dimension still the same
df.shape

(990, 194)
```

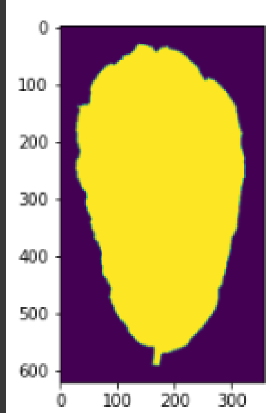
Data visualization:

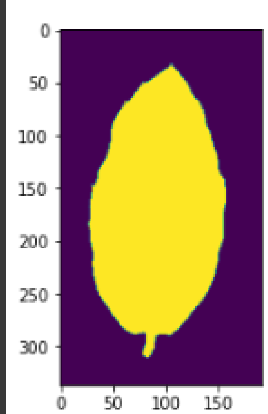


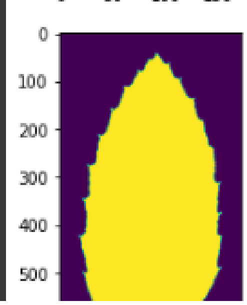
Drawing some images:

```
#read and show some images from the data
import matplotlib.image as mpimg
images = ['81','82','83','84','89','90','91','92']
for i in images:
  img = mpimg.imread('/content/drive/MyDrive/imagess/'+i+'.jpg')
  plt.imshow(img)
  plt.show()
```









Dataset:

- assigned target column to y and encoded classes
- assign features to X and drop the ID column
- split the data into train and test

```
#assign target column to y and encoded classe
#assign features to X and drop id column
y = pd.DataFrame(OneHotEncoder().fit_transform(df[['species']]).toarray())
X = df.drop(['species','id'],axis=1)
```

```
[22] #split data into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=.2,random_state=42)
```

```
[23] #print train features dimensions
X_train.shape

(792, 192)
```

```
[24] #print dimension of train label
y_train.shape

(792, 99)
```

Check if the data need standardization or not:

```
[25] #print standard deviation for the features to check if the data needs standarization or normalization
# but it was found that the data was not need all values are colse from each other
print(X_train.std())

margin1      0.019840
margin2      0.039481
margin3      0.025302
margin4      0.028378
margin5      0.018455
               ...
texture60    0.060194
texture61    0.011689
texture62    0.039412
texture63    0.013950
texture64    0.023202
Length: 192, dtype: float64
```

```
[ ] #print meann for the features to check if the data needs standarization or normalization
# but it was found that the data was not need all values are colse from each other
print(X_train.mean())

margin1      0.017460
margin2      0.028841
margin3      0.031065
margin4      0.023265
margin5      0.014256
               ...
texture60    0.013684
texture61    0.002612
texture62    0.019820
texture63    0.009286
texture64    0.019894
Length: 192, dtype: float64
```

```
[ ] #Describe the data using summary statistics
X_train.describe()
```

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	...
count	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	...
mean	0.017460	0.028841	0.031065	0.023265	0.014256	0.038229	0.019351	0.001142	0.007122	0.018466	...
std	0.019840	0.039481	0.025302	0.028378	0.018455	0.051631	0.017731	0.002869	0.008986	0.015886	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.000000	0.005859	...
50%	0.009766	0.011719	0.023438	0.013672	0.007812	0.014649	0.015625	0.000000	0.005859	0.015625	...
75%	0.025391	0.039551	0.042969	0.029297	0.019531	0.054688	0.029297	0.000000	0.007812	0.027344	...
max	0.085938	0.205080	0.138670	0.169920	0.111330	0.275390	0.091797	0.031250	0.076172	0.097656	...

	texture55	texture56	texture57	texture58	texture59	texture60	texture61	texture62	texture63	texture64
count	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000	792.000000
mean	0.034414	0.005200	0.016491	0.010726	0.016110	0.013684	0.002612	0.019820	0.009286	0.019894
std	0.059560	0.019709	0.023981	0.023750	0.015522	0.060194	0.011689	0.039412	0.013950	0.023202
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000977	0.000000	0.004883	0.000000	0.000000	0.000000	0.000000	0.000977
50%	0.003906	0.000000	0.006836	0.000977	0.012695	0.000000	0.000000	0.003906	0.002930	0.011719
75%	0.040283	0.000000	0.021484	0.008789	0.021484	0.000000	0.000000	0.022461	0.013672	0.029541
max	0.375980	0.202150	0.172850	0.200200	0.106450	0.578130	0.151370	0.375980	0.082031	0.141600

8 rows × 192 columns

As shown in the above figure the data do not need standardization or normalization because all features’ value is close.

Part II: Training a neural network

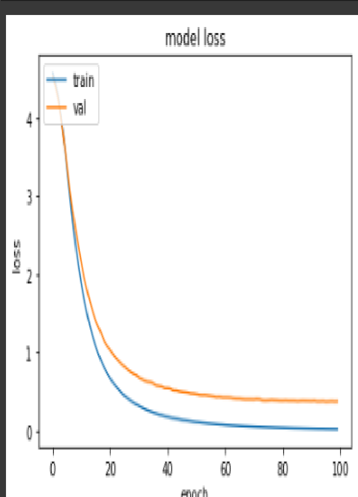
Model architecture:

```
model = Sequential()
model.add(Dense(512, activation='tanh',kernel_initializer='glorot_uniform',input_shape=(X_train.shape[1],)))
model.add(Dense(99, activation='softmax'))
model.summary()
```

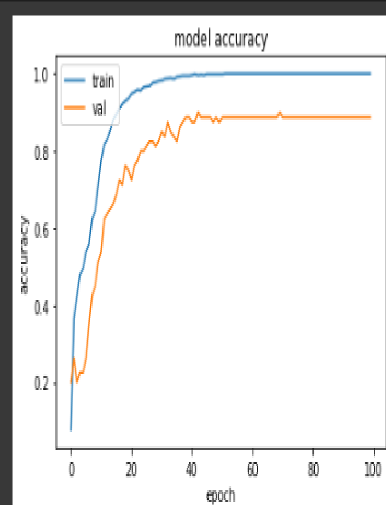
```
Model: "sequential_90"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_180 (Dense)           (None, 512)               98816

 dense_181 (Dense)           (None, 99)                50787

=================================================================
Total params: 149,603
Trainable params: 149,603
Non-trainable params: 0
_________________________________________________________________
```

Plot accuracy and loss for the model:

```
#plot loss
plt.plot(model.history['loss'])
plt.plot(model.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
[42] #plot accuracy
plt.plot(model.history['accuracy'])
plt.plot(model.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



Fine Tuning to find the best hyperparameters:

```
#try combination for 4 hyperparameters
hidden_size = [128,256,512]
drop = [.2,.4,.6]
opt = ['SGD','adam','RMSProp']
batches = [20,25,30]

results = []

for i in hidden_size:
  for j in drop:
    for k in opt:
      for l in batches:
        model = Sequential()
        model.add(Dense(i, activation='tanh',kernel_initializer='glorot_uniform',input_shape=(X_train.shape[1],)))
        model.add(Dropout(j))
        model.add(Dense(99, activation='softmax'))

        model.compile( loss='CategoricalCrossentropy',optimizer= k, metrics=['accuracy'])

        modell = model.fit(X_train, y_train, epochs = 100, validation_split = 0.1,batch_size = l,shuffle=True)
        model.summary()
        model.evaluate(X_test, y_test)
        print(f"Hidden size: {i}, dropout: {j}, Optimizer: {k}, Batch size: {l}")
        plot_acuraccy(model)
        plot_loss(model)
```

As shown in the above figure (hidden size, optimizer, dropout, number of batches) the four hyperparameters that were selected, and each one of them was given three different values:

hidden_size = [128,256,512]

drop = [.2,.4,.6]

opt = ['SGD','adam','RMSProp']

batches = [20,25,30]

3 nested for loops were implemented to find the best combination for the hyperparameters' values and it was found that the best combination according to test accuracy and loss accuracy is:

hidden_size = [512]

drop = [.2]

opt = ['adam']

batches = [25]

test accuracy and loss:

```
7/7 [==============================] - 0s 7ms/step - loss: 0.1572 - accuracy: 0.9697
Hidden size: 512, dropout: 0.2, Optimizer: adam, Batch size: 25
```

Train and validation loss and accuracy: