CISC 867: PROJECT 1

Leaf Classification

Lomai Mohamed Saadeldin
ID: 20398049

Supervised by: Prof. Hazem Abbas

✓ Problem Formulation:

▪ 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.

▪ **The Problems:**

➢ We will use this data using a neural network architecture.

▪ **Input**:

➢ Features collected from numbers of species of plant.

▪ **Output**:

➢ Predicted species for leaves.

▪ **Challenges** :

➢ Clean the data set.

➢ Try different hyperparameters for the netwoek.

▪ **Impact** :

➢ Predicting the species of the leaf that will lead to a successful match.

Part I: Data Pearation

1)import modules to dealing with data set

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np # linear algebra
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler , OneHotEncoder

from matplotlib.pyplot import figure, imshow, axis

#DEEP LEARNING
from keras.layers.core import Dense
from keras.layers import Dropout
from tensorflow import keras
from keras.models import Sequential
from keras import regularizers
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.optimizers import Optimizer
from keras.optimizers import SGD
from keras.optimizers import Adam
from keras.optimizers import RMSprop
```

2) Download the data file, load it:

```
[4] #read dataset into files
    train=pd.read_csv('/content/train.csv') #train dataset
    test=pd.read_csv('/content/test.csv') #test dataset
```

3) Describe the data

```
[9] train.describe()
```

	id	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin
count	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.00000
mean	799.595960	0.017412	0.028539	0.031988	0.023280	0.014264	0.038579	0.019202	0.00108
std	452.477568	0.019739	0.038855	0.025847	0.028411	0.018390	0.052030	0.017511	0.00274
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	415.250000	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.00000
50%	802.500000	0.009766	0.011719	0.025391	0.013672	0.007812	0.015625	0.015625	0.00000
75%	1195.500000	0.025391	0.041016	0.044922	0.029297	0.017578	0.056153	0.029297	0.00000
max	1584.000000	0.087891	0.205080	0.156250	0.169920	0.111330	0.310550	0.091797	0.03125

8 rows × 193 columns

4) Clean the data set

check the null value

```
[13] #check the null values (train set)
     train.isnull().sum()
```

```
id            0
species       0
margin1       0
margin2       0
margin3       0
             ..
texture60     0
texture61     0
texture62     0
texture63     0
texture64     0
Length: 194, dtype: int64
```

```
[14] #check the null values (test set)
     test.isnull().sum()
```

```
id            0
margin1       0
margin2       0
margin3       0
margin4       0
             ..
texture60     0
texture61     0
texture62     0
texture63     0
texture64     0
Length: 193, dtype: int64
```

Check for duplicates

```
[15]
     #check for duplicates (train set)
     train.duplicated().sum()
```

```
0
```

```
[16] #check for duplicates (test set)
     test.duplicated().sum()
```

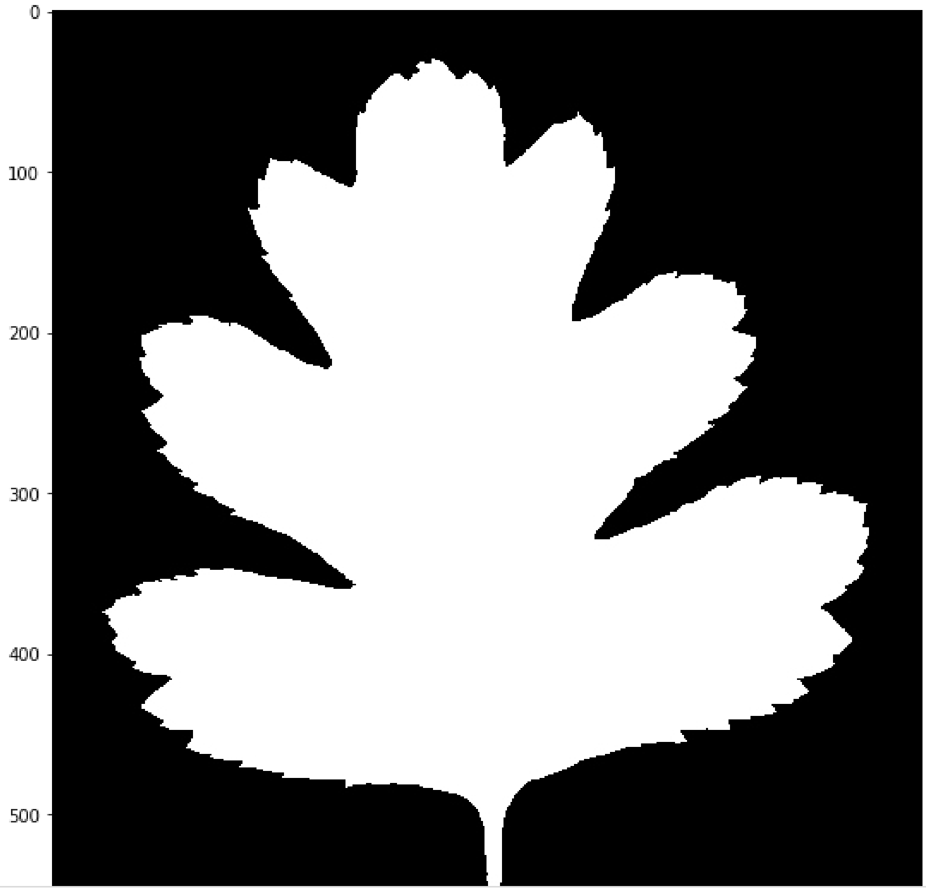
```
0
```

5) Draw some of the images

```
[17] import cv2
     import glob

     images = []
     files = glob.glob ("/content/drive/MyDrive/images/*.jpg")
     for myFile in files:
         image = cv2.imread(myFile, cv2.IMREAD_GRAYSCALE)
         images.append(image)
```

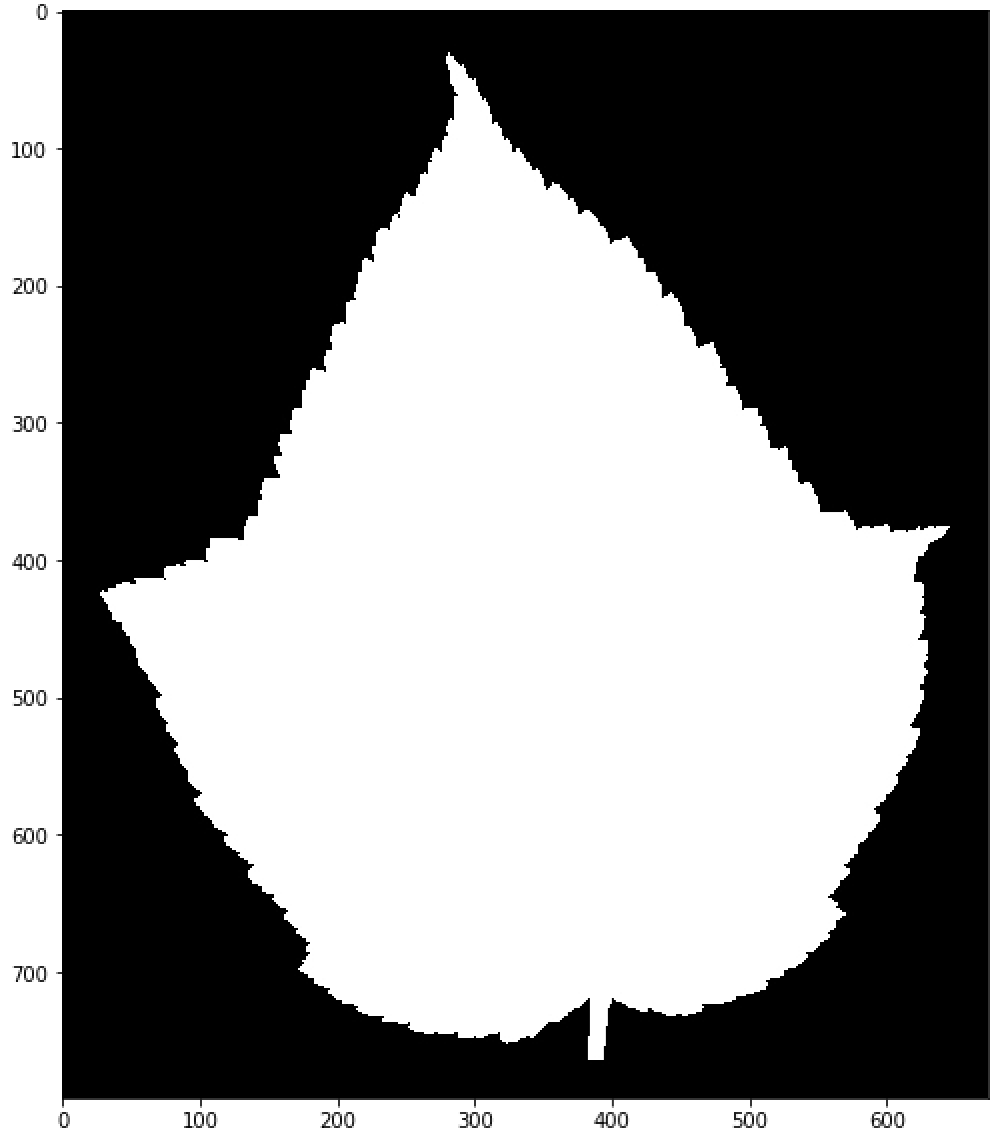
```
[31] plt.figure(figsize=(15,10))
     plt.imshow(images[0],cmap='gray', interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x7f0038cb4b20>
```



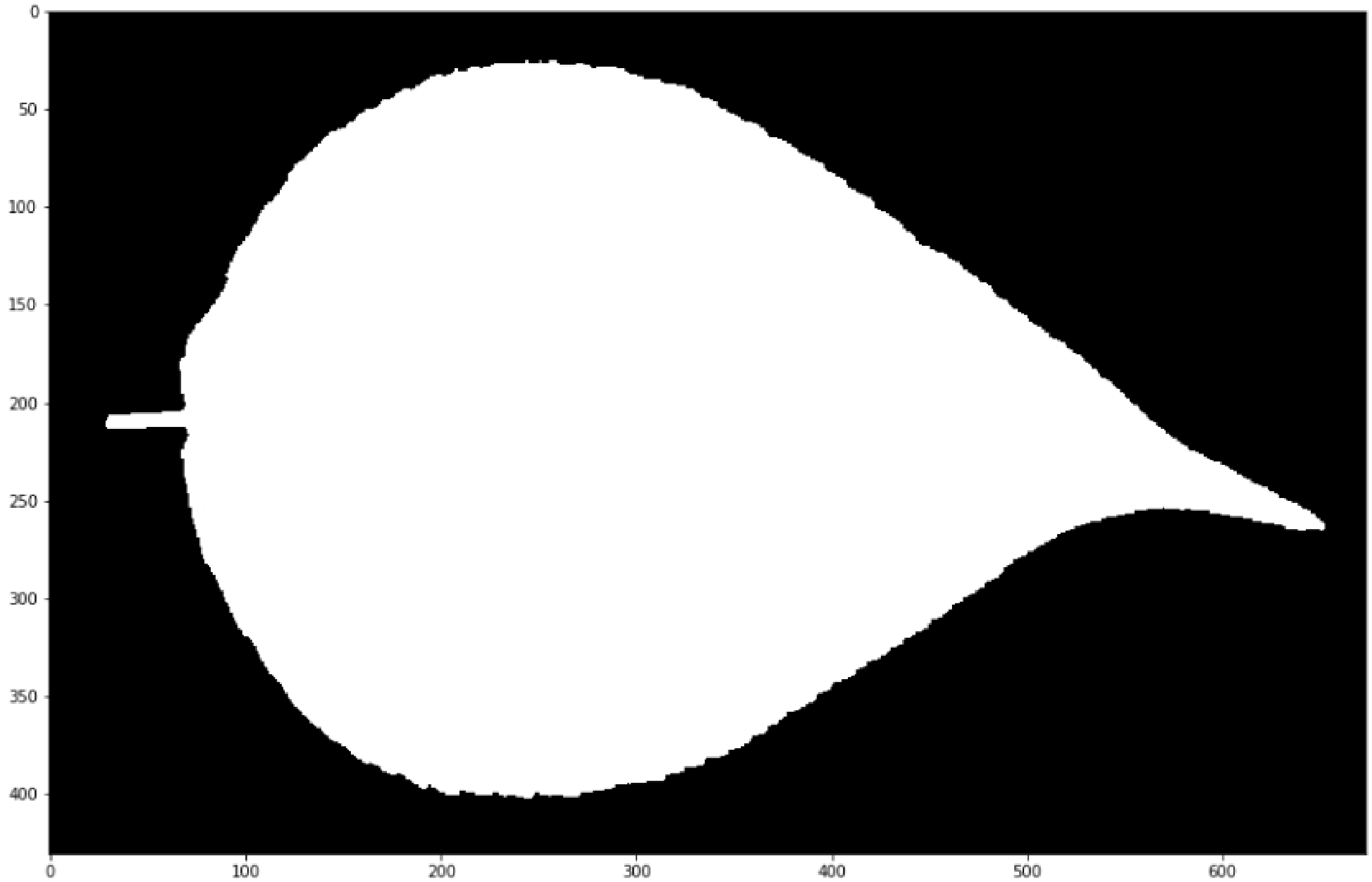
```python
plt.figure(figsize=(15,10))
plt.imshow(images[2],cmap='gray', interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x7f0034a45070>
```



```python
plt.figure(figsize=(15,10))
plt.imshow(images[5],cmap='gray', interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x7f00341bdfd0>
```



6) correlation analysis

```python
correlation_matrix = train.corr().abs()
print(correlation_matrix)
```

```
                 id   margin1   margin2   margin3   margin4   margin5  \
id         1.000000  0.011673  0.027565  0.059533  0.001639  0.002419
margin1    0.011673  1.000000  0.806390  0.182829  0.297807  0.475874
margin2    0.027565  0.806390  1.000000  0.204640  0.315953  0.444312
margin3    0.059533  0.182829  0.204640  1.000000  0.120042  0.185007
margin4    0.001639  0.297807  0.315953  0.120042  1.000000  0.029480
...             ...       ...       ...       ...       ...       ...
texture60  0.000823  0.035072  0.081069  0.019850  0.052317  0.006542
texture61  0.026319  0.007581  0.007057  0.084957  0.320644  0.109229
texture62  0.032873  0.033159  0.037405  0.081999  0.073886  0.151675
texture63  0.024299  0.075171  0.098957  0.148193  0.050970  0.022299
texture64  0.035396  0.030414  0.029532  0.061780  0.014343  0.148834

            margin6   margin7   margin8   margin9  ...  texture55  texture56  \
id         0.051818  0.061214  0.039509  0.070954  ...   0.040292   0.005132
margin1    0.767718  0.066273  0.094137  0.181496  ...   0.137158   0.047771
margin2    0.825762  0.083273  0.086428  0.120276  ...   0.154407   0.021096
margin3    0.163976  0.095449  0.024350  0.000042  ...   0.047347   0.027618
margin4    0.261437  0.268271  0.047693  0.227543  ...   0.071974   0.009537
...             ...       ...       ...       ...  ...        ...        ...
texture60  0.066262  0.034094  0.048647  0.028292  ...   0.129365   0.004412
texture61  0.050498  0.163375  0.079283  0.088517  ...   0.002235   0.053707
texture62  0.031555  0.015391  0.048843  0.031954  ...   0.217239   0.171577
texture63  0.132087  0.001364  0.027758  0.119494  ...   0.207887   0.002057
texture64  0.003164  0.068512  0.003191  0.097760  ...   0.095205   0.095913

           texture57  texture58  texture59  texture60  texture61  texture62  \
id          0.043101   0.063337   0.007915   0.000823   0.026319   0.032873
margin1     0.126227   0.024139   0.168201   0.035072   0.007581   0.033159
margin2     0.123834   0.063654   0.157842   0.081069   0.007057   0.037405
margin3     0.007261   0.021390   0.033505   0.019850   0.084957   0.081999
margin4     0.050529   0.044318   0.088857   0.052317   0.320644   0.073886
...              ...        ...        ...        ...        ...        ...
texture60   0.155187   0.240704   0.183369   1.000000   0.051838   0.265879
texture61   0.072814   0.084638   0.023539   0.051838   1.000000   0.063582
texture62   0.283316   0.563088   0.128010   0.265879   0.063582   1.000000
texture63   0.064724   0.059866   0.156568   0.089679   0.068065   0.058189
texture64   0.224686   0.269157   0.015374   0.190194   0.036374   0.245527
```

```
upper_tri = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape),k=1).ast
print(upper_tri)
```

```
            id   margin1   margin2   margin3   margin4   margin5   margin6  \
id         NaN  0.011673  0.027565  0.059533  0.001639  0.002419  0.051818
margin1    NaN       NaN  0.806390  0.182829  0.297807  0.475874  0.767718
margin2    NaN       NaN       NaN  0.204640  0.315953  0.444312  0.825762
margin3    NaN       NaN       NaN       NaN  0.120042  0.185007  0.163976
margin4    NaN       NaN       NaN       NaN       NaN  0.029480  0.261437
...         ..       ...       ...       ...       ...       ...       ...
texture60  NaN       NaN       NaN       NaN       NaN       NaN       NaN
texture61  NaN       NaN       NaN       NaN       NaN       NaN       NaN
texture62  NaN       NaN       NaN       NaN       NaN       NaN       NaN
texture63  NaN       NaN       NaN       NaN       NaN       NaN       NaN
texture64  NaN       NaN       NaN       NaN       NaN       NaN       NaN

            margin7   margin8   margin9  ...  texture55  texture56  texture57  \
id         0.061214  0.039509  0.070954  ...   0.040292   0.005132   0.043101
margin1    0.066273  0.094137  0.181496  ...   0.137158   0.047771   0.126227
margin2    0.083273  0.086428  0.120276  ...   0.154407   0.021096   0.123834
margin3    0.095449  0.024350  0.000042  ...   0.047347   0.027618   0.007261
margin4    0.268271  0.047693  0.227543  ...   0.071974   0.009537   0.050529
...             ...       ...       ...  ...        ...        ...        ...
texture60       NaN       NaN       NaN  ...        NaN        NaN        NaN
texture61       NaN       NaN       NaN  ...        NaN        NaN        NaN
texture62       NaN       NaN       NaN  ...        NaN        NaN        NaN
texture63       NaN       NaN       NaN  ...        NaN        NaN        NaN
texture64       NaN       NaN       NaN  ...        NaN        NaN        NaN

           texture58  texture59  texture60  texture61  texture62  texture63  \
id          0.063337   0.007915   0.000823   0.026319   0.032873   0.024299
margin1     0.024139   0.168201   0.035072   0.007581   0.033159   0.075171
margin2     0.063654   0.157842   0.081069   0.007057   0.037405   0.098957
margin3     0.021390   0.033505   0.019850   0.084957   0.081999   0.148193
margin4     0.044318   0.088857   0.052317   0.320644   0.073886   0.050970
...              ...        ...        ...        ...        ...        ...
texture60        NaN        NaN        NaN   0.051838   0.265879   0.089679
texture61        NaN        NaN        NaN        NaN   0.063582   0.068065
texture62        NaN        NaN        NaN        NaN        NaN   0.058189
texture63        NaN        NaN        NaN        NaN        NaN        NaN
texture64        NaN        NaN        NaN        NaN        NaN        NaN
```

```
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.95)]
print(to_drop)
```

```
['shape2', 'shape3', 'shape4', 'shape5', 'shape6', 'shape7', 'shape8', 'shape9', 'shape10', 'shape11', 'shape
```

7) spilt data into train set and test set

```
[ ]  x =  train.drop('species',axis = 1)
     Y = train.species.values
     from sklearn.preprocessing import LabelEncoder
     label_encoder = LabelEncoder()
     y = label_encoder.fit_transform(Y)
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y, random_state = 5, train_size=0.75,shuffle = True)
```

8) Normalize data

```
scaler = StandardScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

```
[ ] print('mean',scaler.mean_)
    print('scale',scaler.scale_)
```

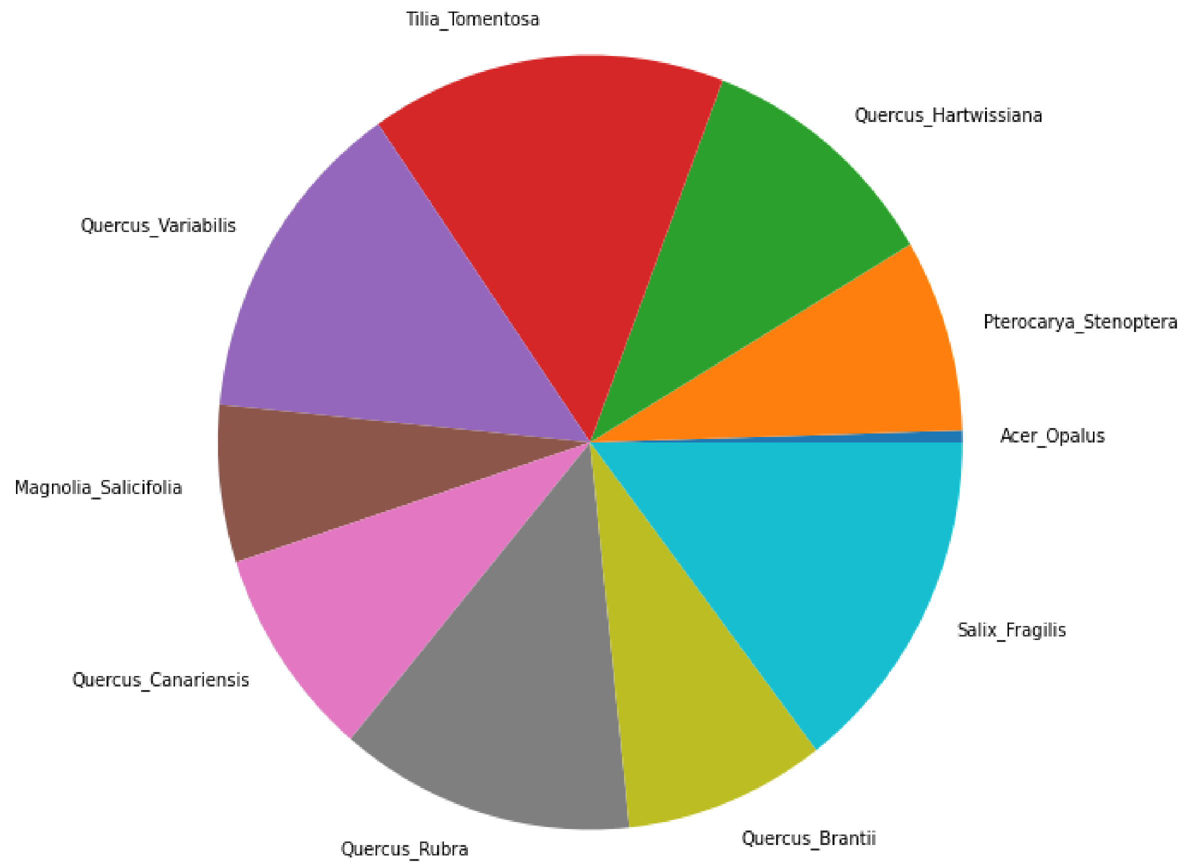
```
mean [7.83587601e+02 1.70990108e-02 2.80518248e-02 3.21133841e-02
 2.28478140e-02 1.46404784e-02 3.82439313e-02 1.92074434e-02
 1.06863208e-03 7.43858221e-03 1.85072790e-02 2.38112318e-02
 1.20872049e-02 4.08340647e-02 8.00983962e-03 1.59171348e-02
 1.18447439e-04 1.49379515e-02 1.98286550e-02 1.22582911e-02
 1.30348140e-02 1.90626523e-02 6.16465903e-03 1.14501078e-03
 7.84665499e-03 8.85481806e-03 1.87363059e-02 5.69612129e-03
 1.57223464e-02 2.80860162e-02 1.63698666e-02 1.10843154e-02
 1.00314367e-02 1.94311792e-02 1.03967790e-03 1.30716779e-02
 1.79650202e-02 1.62725121e-02 3.12131307e-02 1.51880040e-02
 8.27043396e-03 1.14159771e-02 1.70016132e-02 1.87968167e-02
 1.28952965e-02 2.42060701e-02 1.00577183e-02 2.50510040e-02
 2.74858666e-02 8.82587197e-03 1.39060970e-02 2.55643208e-02
 2.94542318e-03 2.43956199e-02 1.02472790e-02 1.86888935e-02
 5.89615364e-03 1.26136429e-02 1.93548464e-02 3.06893005e-02
 1.20240323e-02 1.39500809e-03 5.15124394e-03 2.54853598e-02
 4.59581671e-03 7.28471941e-04 7.07037129e-04 6.82184960e-04
 6.59185903e-04 6.38773410e-04 6.21596954e-04 6.07364737e-04
 5.93877190e-04 5.80945175e-04 5.68772197e-04 5.59865303e-04
 5.51597615e-04 5.44224272e-04 5.39178863e-04 5.35114623e-04
 5.33399751e-04 5.32427930e-04 5.31567396e-04 5.31953889e-04
 5.36114877e-04 5.43444691e-04 5.52944600e-04 5.64752032e-04
 5.75690770e-04 5.88560433e-04 6.02043435e-04 6.15739018e-04
 6.31904035e-04 6.51519131e-04 6.75313616e-04 6.99679654e-04
 7.21895206e-04 7.26824232e-04 7.06407965e-04 6.79579040e-04
 6.54815082e-04 6.32504625e-04 6.11320988e-04 5.92783865e-04
 5.75329730e-04 5.61806011e-04 5.51197875e-04 5.44890580e-04
 5.38795937e-04 5.35181608e-04 5.33334003e-04 5.34240786e-04
 5.35375089e-04 5.37140028e-04 5.38120687e-04 5.37302439e-04
 5.38117682e-04 5.41949146e-04 5.46436186e-04 5.52663935e-04
 5.59650171e-04 5.68595879e-04 5.81757454e-04 5.97590139e-04
 6.16994602e-04 6.38255210e-04 6.64383302e-04 6.93275229e-04
 7.22135782e-04 2.16515323e-02 1.18214259e-02 1.01104582e-02
 1.54618491e-02 2.65199286e-02 1.01499272e-02 1.66345121e-02
 1.94891361e-02 1.49972237e-02 1.96405512e-02 1.88758544e-02
```

9) Data Visualizing

Visualize the data using proper visualization methods.

Sample of the species

```
[ ] Figure = plt.figure(figsize =(15, 10))
    plt.pie(y[:10],labels=train['species'][:10]);
```



Part II: Training a neural network

1)Build ANN Model

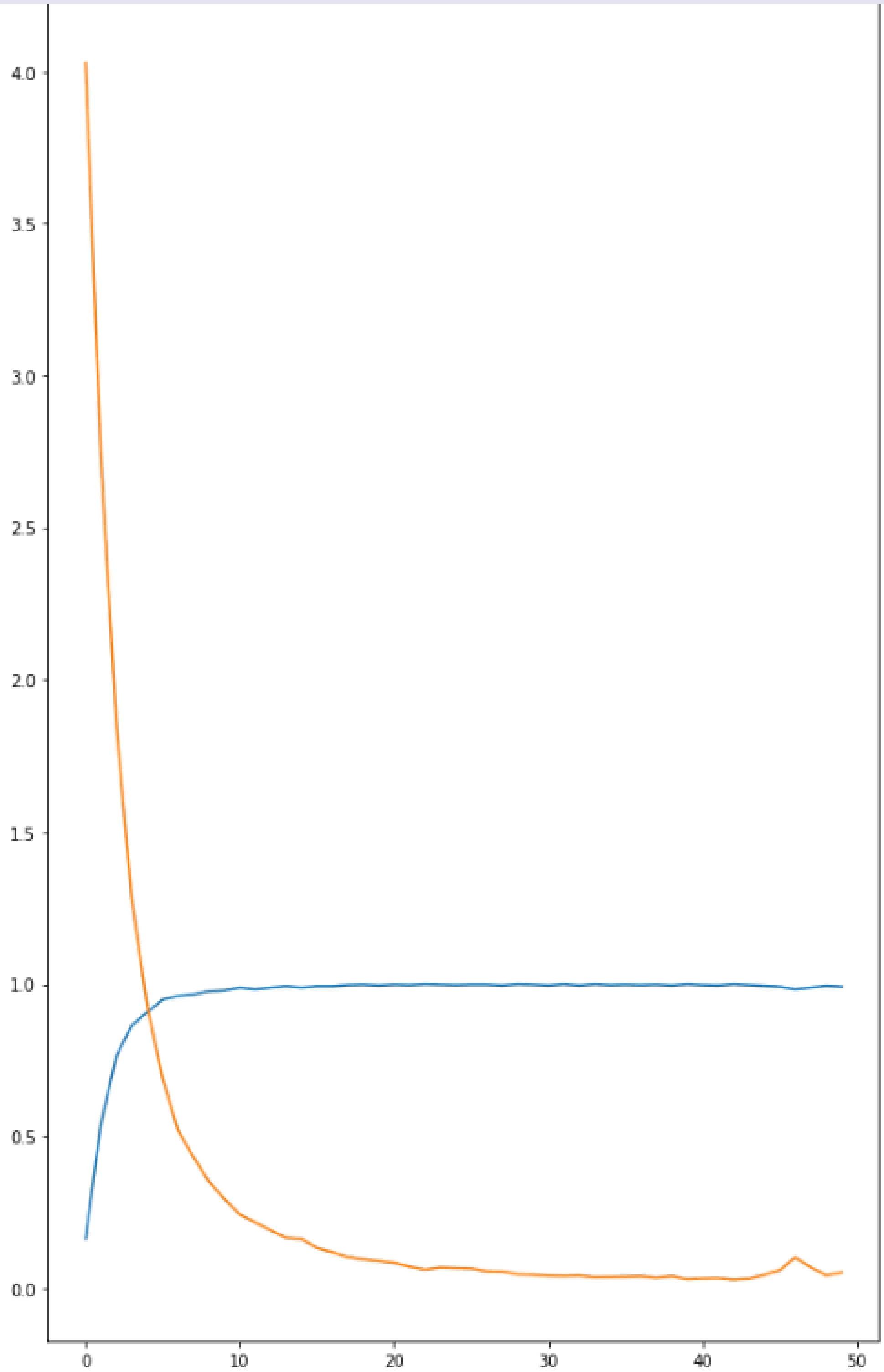
```
[50]
def ANN(batch_size, hidden_dropout, Regularization, learning_rate, hidden_Size, n_epoches):
  #Build ANN MODEL
  Model = Sequential()
  Model.add(keras.layers.Flatten())
  #hidden layers
  Model.add(Dense(hidden_Size,activation='tanh'))
  #Dropout layer
  Model.add(Dropout(hidden_dropout))
  #Output layer
  Model.add(Dense(99, activation='softmax'))

  #I am try using different optimizers such as SGD, Adam, RMSProp and I found Adam was the best
  opt1 = keras.optimizers.SGD(learning_rate=learning_rate)
  opt2 = keras.optimizers.Adam(learning_rate=learning_rate)
  opt3 = keras.optimizers.RMSprop(learning_rate=learning_rate)

  Model.compile(loss='sparse_categorical_crossentropy',optimizer=opt2, metrics=['accuracy'] )
  history = Model.fit(x_train, y_train, epochs=50,validation_data=(x_test,y_test))
  acc = history.history['accuracy']
  loss = history.history['loss']
  Model.build(x_train.shape)
  print(Model.summary())
  plt.subplot(1, 2, 1)
  plt.plot(acc)
  plt.subplot(1, 2, 1)
  plt.plot(loss)
  plt.title('model loss & accuracy')
  plt.show()
  print(Model.evaluate(x_test, y_test, verbose=1))
```

I create function that take (batch size, hidden dropout, Regularization, Learning rate, hidden Size, no. epoches)

I call the Function with different values:

```
ANN(32,0.1,0,0.01,20,100)
```

```
24/24 [==============================] - 0s 4ms/step - loss: 4.4353 - accuracy: 0.0391 - val_loss: 4.4647 - val_accuracy: 0.0323
Epoch 7/50
24/24 [==============================] - 0s 5ms/step - loss: 4.3972 - accuracy: 0.0512 - val_loss: 4.4317 - val_accuracy: 0.0363
Epoch 8/50
24/24 [==============================] - 0s 4ms/step - loss: 4.3394 - accuracy: 0.0553 - val_loss: 4.4017 - val_accuracy: 0.0444
Epoch 9/50
24/24 [==============================] - 0s 4ms/step - loss: 4.3127 - accuracy: 0.0660 - val_loss: 4.3714 - val_accuracy: 0.0444
Epoch 10/50
24/24 [==============================] - 0s 4ms/step - loss: 4.2822 - accuracy: 0.0782 - val_loss: 4.3435 - val_accuracy: 0.0605
Epoch 11/50
24/24 [==============================] - 0s 4ms/step - loss: 4.2409 - accuracy: 0.0930 - val_loss: 4.3134 - val_accuracy: 0.0645
Epoch 12/50
24/24 [==============================] - 0s 4ms/step - loss: 4.2119 - accuracy: 0.1092 - val_loss: 4.2868 - val_accuracy: 0.0766
Epoch 13/50
24/24 [==============================] - 0s 4ms/step - loss: 4.1705 - accuracy: 0.1186 - val_loss: 4.2606 - val_accuracy: 0.0847
Epoch 14/50
24/24 [==============================] - 0s 4ms/step - loss: 4.1325 - accuracy: 0.1429 - val_loss: 4.2343 - val_accuracy: 0.0968
Epoch 15/50
24/24 [==============================] - 0s 4ms/step - loss: 4.1003 - accuracy: 0.1496 - val_loss: 4.2087 - val_accuracy: 0.1048
Epoch 16/50
24/24 [==============================] - 0s 4ms/step - loss: 4.0686 - accuracy: 0.1577 - val_loss: 4.1861 - val_accuracy: 0.1048
Epoch 17/50
24/24 [==============================] - 0s 3ms/step - loss: 4.0313 - accuracy: 0.1739 - val_loss: 4.1630 - val_accuracy: 0.1169
Epoch 18/50
24/24 [==============================] - 0s 5ms/step - loss: 4.0033 - accuracy: 0.1671 - val_loss: 4.1388 - val_accuracy: 0.1210
Epoch 19/50
24/24 [==============================] - 0s 3ms/step - loss: 3.9788 - accuracy: 0.2116 - val_loss: 4.1159 - val_accuracy: 0.1411
Epoch 20/50
24/24 [==============================] - 0s 3ms/step - loss: 3.9448 - accuracy: 0.2102 - val_loss: 4.0925 - val_accuracy: 0.1492
Epoch 21/50
24/24 [==============================] - 0s 3ms/step - loss: 3.9225 - accuracy: 0.2291 - val_loss: 4.0702 - val_accuracy: 0.1694
Epoch 22/50
24/24 [==============================] - 0s 4ms/step - loss: 3.8898 - accuracy: 0.2534 - val_loss: 4.0482 - val_accuracy: 0.1815
Epoch 23/50
24/24 [==============================] - 0s 3ms/step - loss: 3.8553 - accuracy: 0.2493 - val_loss: 4.0269 - val_accuracy: 0.1895
Epoch 24/50
24/24 [==============================] - 0s 3ms/step - loss: 3.8268 - accuracy: 0.2763 - val_loss: 4.0049 - val_accuracy: 0.2056
Epoch 25/50
24/24 [==============================] - 0s 3ms/step - loss: 3.8075 - accuracy: 0.2763 - val_loss: 3.9832 - val_accuracy: 0.2097
Epoch 26/50
```

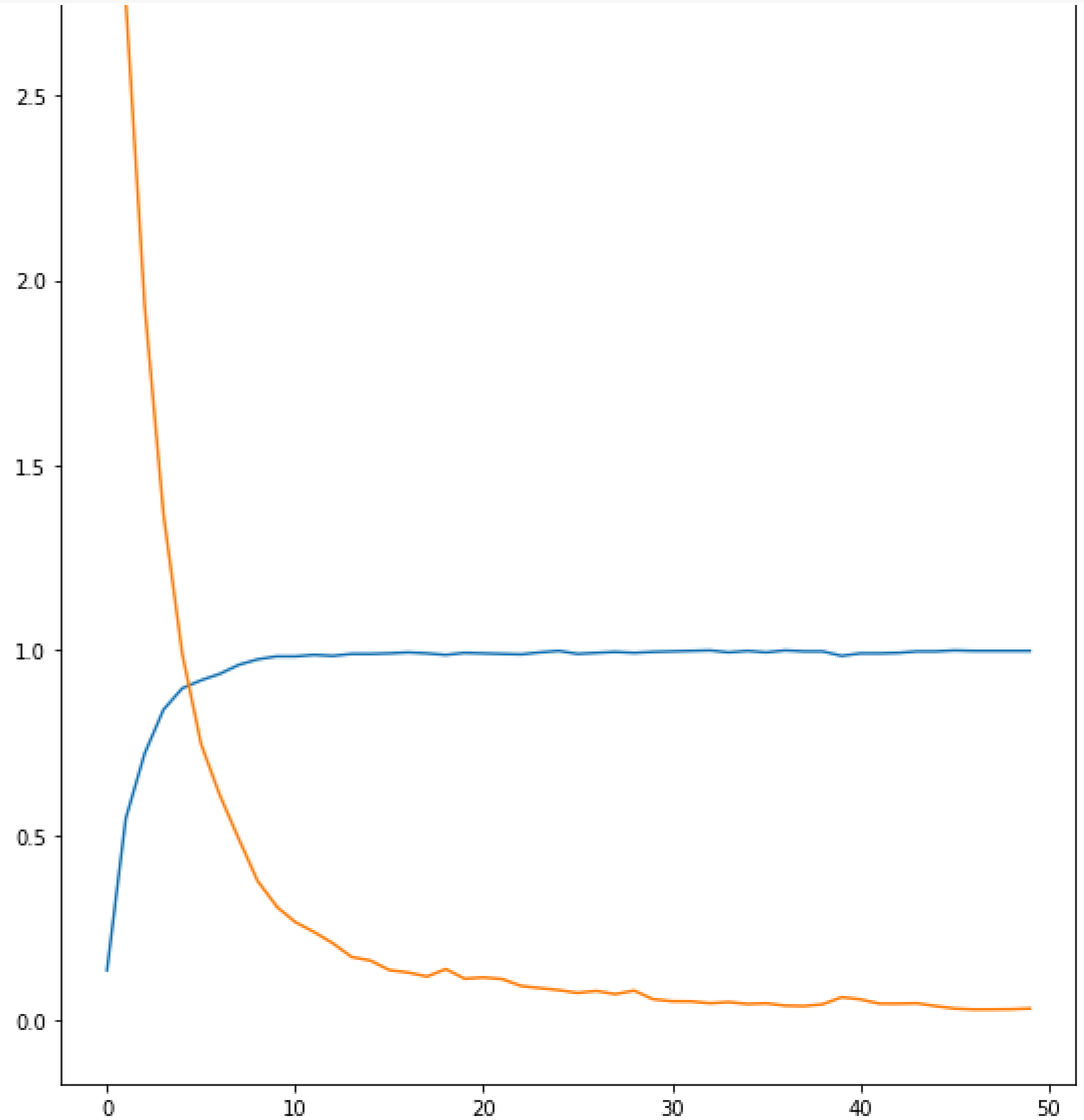



```
8/8 [==============================] - 0s 4ms/step - loss: 0.3682 - accuracy: 0.8831
[0.36819079518318176, 0.8830645084381104]
```

- I tried different values of batch sizes

```python
#try different values of batch sizes
batch_sizes=[64,128,256]
for i in batch_sizes:
  ANN(i,0.1,0,0.01,20,100)
```

I found this the best model (batch_sizes=128)

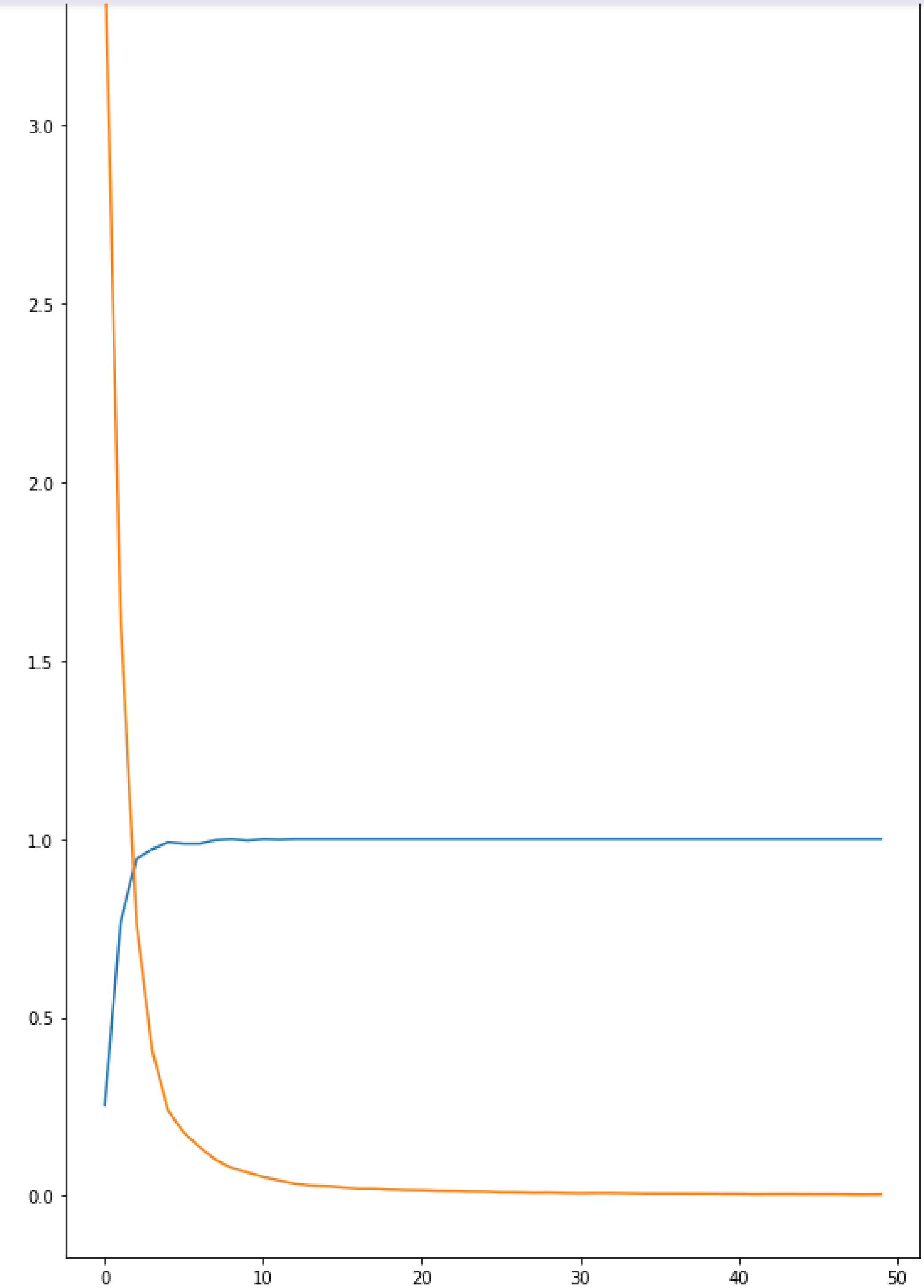



```
8/8 [==============================] - 0s 4ms/step - loss: 0.3817 - accuracy: 0.9153
[0.38174891471862793, 0.9153226017951965]
```

- I tried different values of hidden sizes

```
hidden_Size=[10,20,30,40]
for i in hidden_Size:
    ANN(64,0.1,0,0.01,i,100)
```

I found this the best model (hidden_size=40)

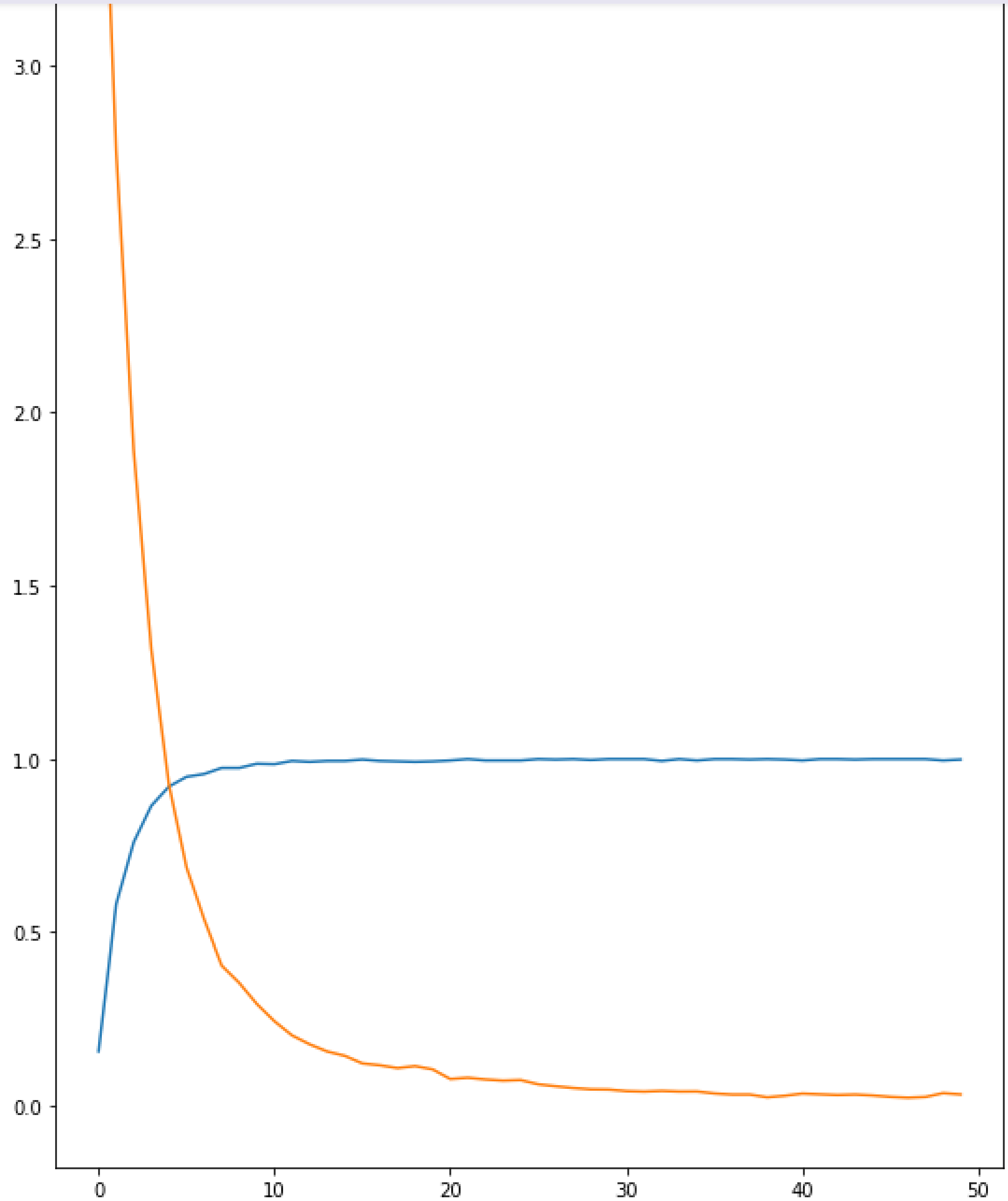



```
8/8 [==============================] - 0s 2ms/step - loss: 0.1193 - accuracy: 0.9637
[0.11927899718284607, 0.9637096524238586]
```

- I tried different values of hidden dropout

```
[55] hidden_dropout=[0,0.1,0.3,0.5,0.7]
     for i in hidden_dropout:
       ANN(64,i,0,0.01,20,100)
```

I found this the best model (hidden_dropout=0.1)

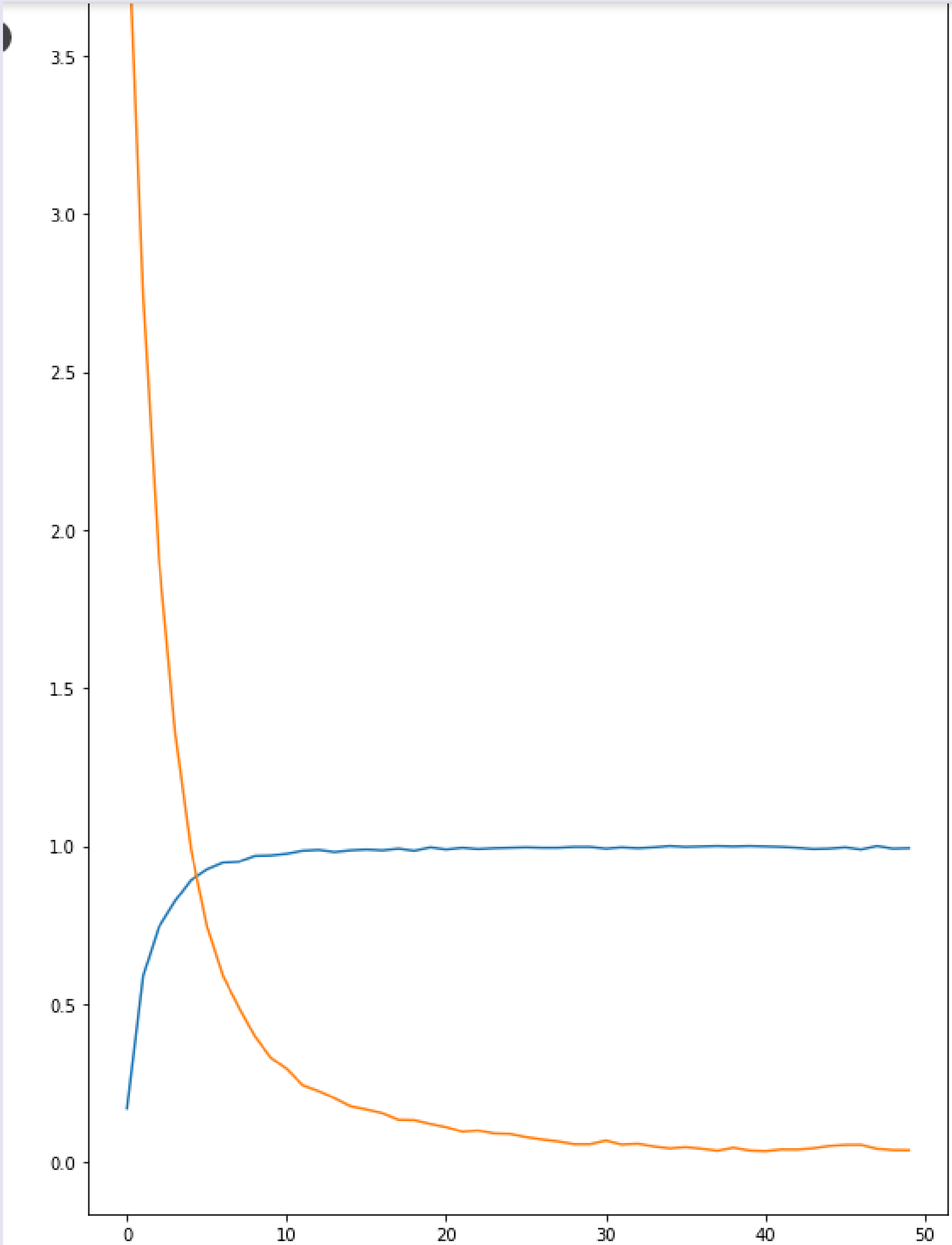



```
8/8 [==============================] - 0s 3ms/step - loss: 0.2726 - accuracy: 0.9435
```

➢ I tried different values of learning rate

```
#try different values of learning_rates
learning_rates=[0.1,0.03,0.01,0.0003]
for i in learning_rates:
  ANN(64,0.1,0,i,20,100)
```

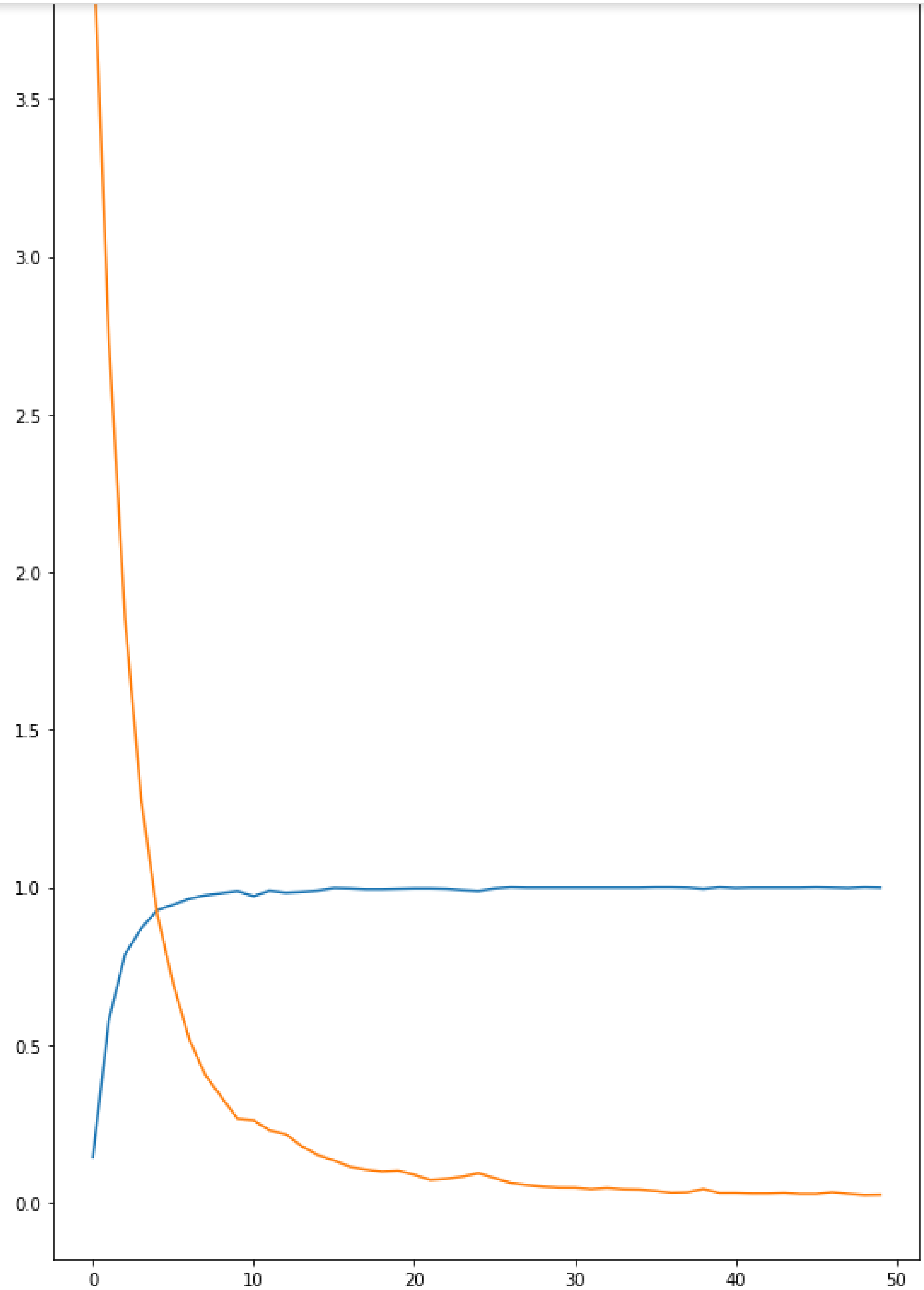
I found this the best model (learning_rates=0.01)




```
8/8 [==============================] - 0s 2ms/step - loss: 0.4101 - accuracy: 0.9073
[0.41005998849868774, 0.9072580933570862]
```

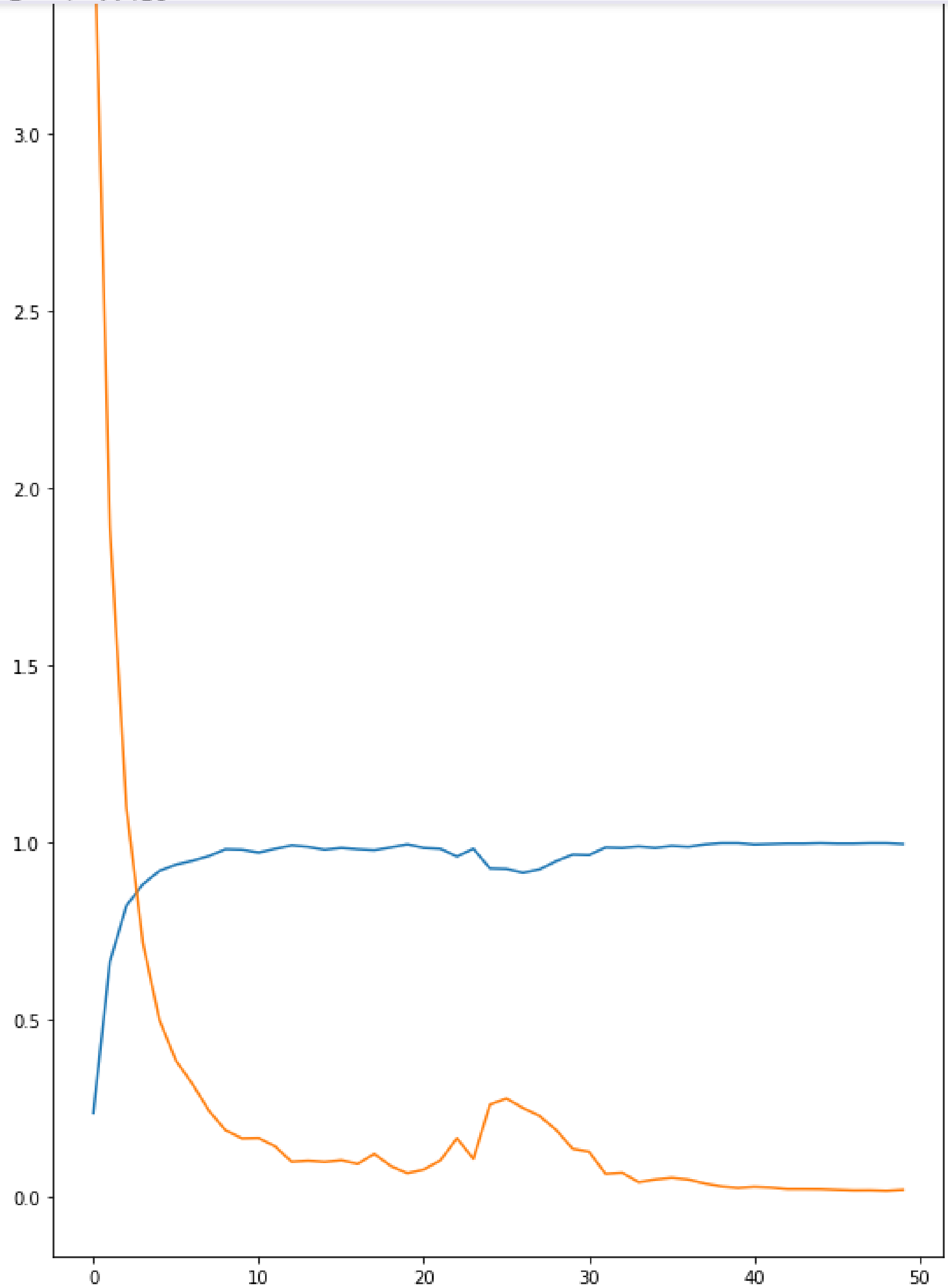
- When training a model, it is often useful to lower the learning rate as the training progresses. This schedule applies an exponential decay function to an optimizer step, given a provided initial learning rate.

```
learningRate_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=1e-2,
    decay_steps=10000,
    decay_rate=0.9)
ANN(64,0.1,0,learningRate_schedule,20,100)
```




```
8/8 [==============================] - 0s 2ms/step - loss: 0.2946 - accuracy: 0.9234
[0.29455944895744324, 0.9233871102333069]
```

```python
learningRate_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=0.02,
    decay_steps=10000,
    decay_rate=0.9)
ANN(64,0.1,0,learningRate_schedule,20,100)
```




```
8/8 [==============================] - 0s 3ms/step - loss: 0.3342 - accuracy: 0.9274
[0.3341643214225769, 0.9274193644523621]
```

Conclusions:

- **I am try using different optimizers such as SGD, Adam, RMSProp and I found Adam was the best**
- **Best Hyperparameters**
  batch_sizes=128
  hidden_size=40
  hidden_dropout=0.1
  learning_rates=0.01

Best accuracy is 96.37%

References

- https://www.kaggle.com/c/leaf-classification/data
- https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/ExponentialDecay
- https://keras.io/api/optimizers/
- https://www.tensorflow.org/tutorials/keras/keras_tuner