# Project Report

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

Project by:

Adel Abdelfatah

ID:

20398047

Supervised by:

Prof. Hazem Abbas

## Problem Statement

There are estimated to be nearly half a million species of plant in the world. Classification of species has been historically problematic and often results in duplicate identifications.

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.

## Dataset 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.

## 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 numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
pd.options.display.max_rows = None
pd.options.display.max_columns = None
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential ,layers ,optimizers ,datasets ,losses
from tensorflow.keras.layers import Dense,Dropout,Flatten , Activation , Input
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from keras.models import load_model
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, TensorBoard, LearningRateScheduler
import keras_tuner as kt
import warnings
warnings.filterwarnings("ignore")
```

## Data Processing:

```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
train.head(2)
```

| | 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 |

```
train.isnull().values.any()
# No null values
```

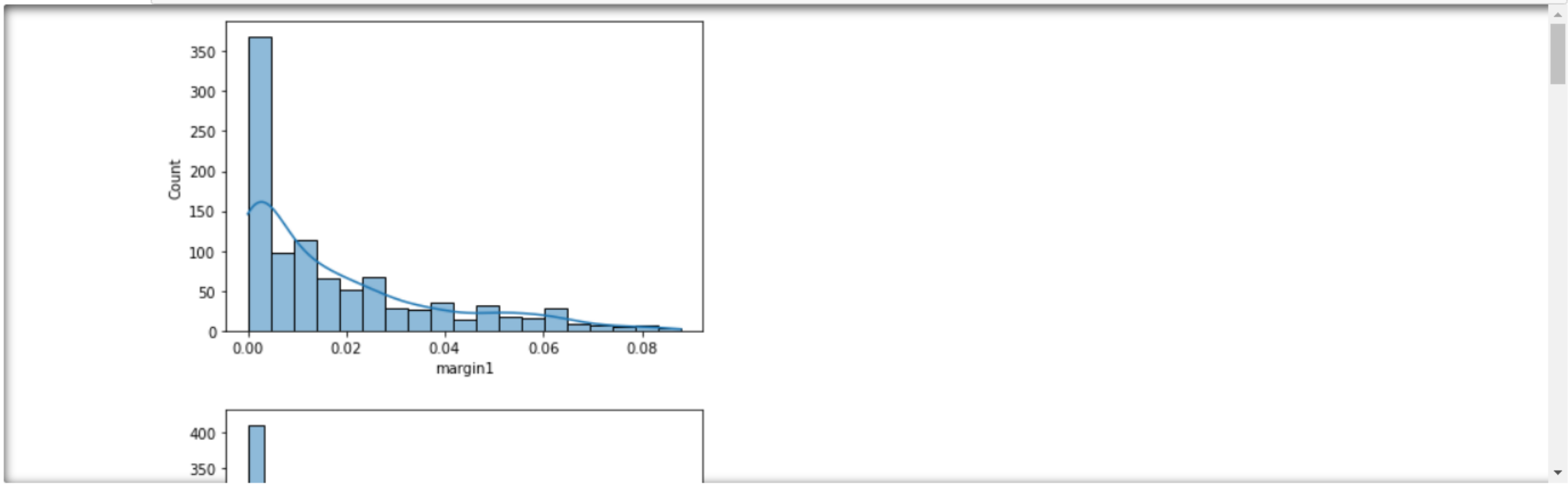
```
False
```

```
train.duplicated().sum()
# no duplicated data
```

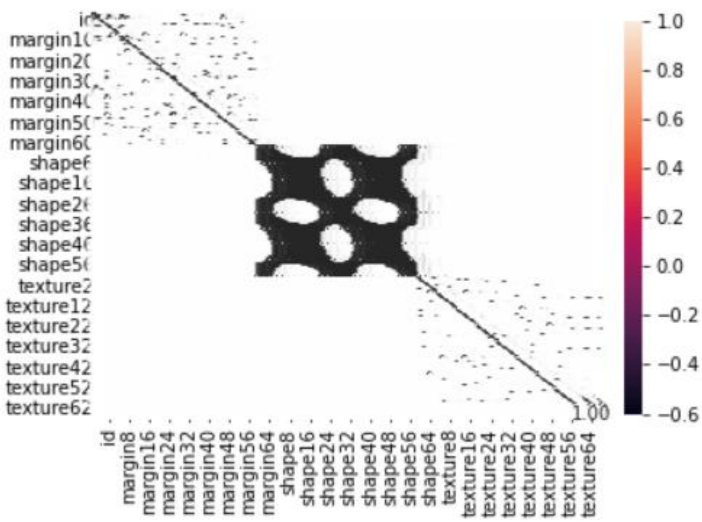
```
0
```

# Data Visualization:

In [14]:
```
# Histogram of some features
A = ['margin1','margin2','margin3','margin4','margin5','margin6','margin7','margin8','margin9']
for i in A:
    sns.histplot(train[i], kde=True)
    plt.show()
```

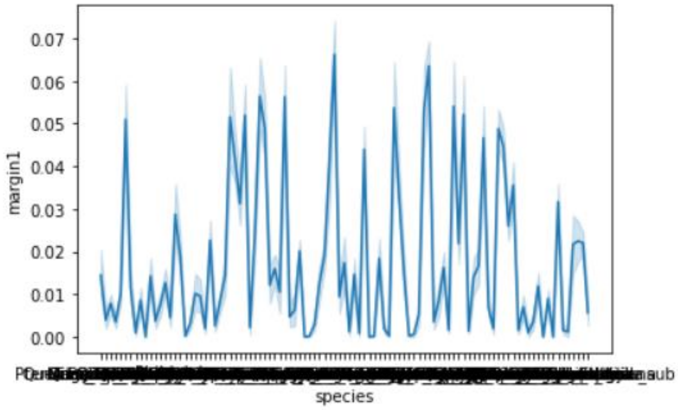



In [15]:
```
# correlation matrix of training dataset
sns.heatmap(train.corr(), annot=True, fmt=".2f")
plt.show()
```



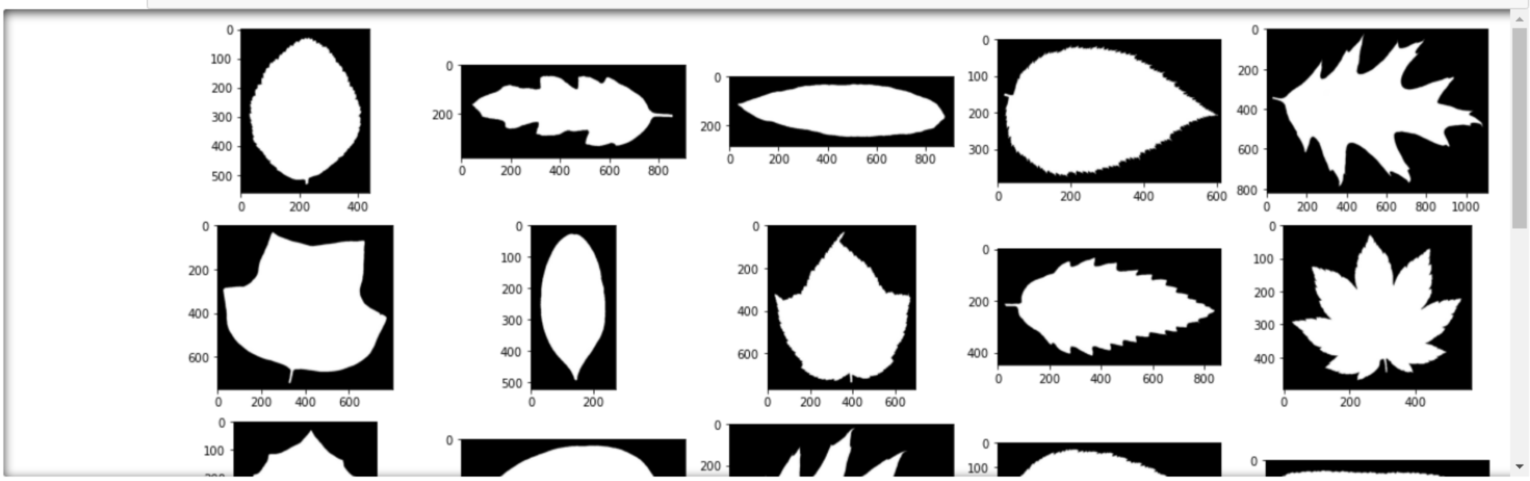
In [16]:
```
sns.lineplot(train['species'], train['margin1'])
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x286dfb46bb0>




## **Drawing Some images:**

In [17]:
```
# Draw some of the images
import os
plt.figure(figsize=(20,15))
import cv2 as cv
from tensorflow.keras.utils import load_img
for i in range(25):
    j=np.random.choice((os.listdir('images')))
    plt.subplot(5,5,i+1)
    img=load_img(os.path.join('images',j))
    plt.imshow(img)
```



# Label Encoding and Devide the dataset:

```
  # label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

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

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

```
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])
```

### Devide the dataset

```
X_train, X_test, y_train, y_test= train_test_split(train.drop(columns=['id','species']),
                                                   train['species'] ,
                                                   random_state=42,
                                                   test_size=0.2,
                                                   shuffle=True)
```

# Calculate mean and std:

```
print(X_train.mean())
```

```
margin1     0.017110
margin2     0.028111
margin3     0.031894
margin4     0.022577
margin5     0.014454
margin6     0.038700
margin7     0.019373
margin8     0.001075
margin9     0.007102
margin10    0.018668
margin11    0.023603
margin12    0.012143
margin13    0.041428
margin14    0.008034
margin15    0.015938
margin16    0.000086
margin17    0.015102
margin18    0.019844
margin19    0.012020
```

There is a difference in mean for the features, so we can do standarization to the data. For Example:

- shape35::: 0.000690
- texture1::: 0.020690

```
print(X_train.std())
```

```
margin1      0.019466
margin2      0.037968
margin3      0.025598
margin4      0.028015
margin5      0.017992
margin6      0.052312
margin7      0.017359
margin8      0.002631
margin9      0.009167
margin10     0.016071
margin11     0.025400
margin12     0.011753
margin13     0.047648
margin14     0.013299
margin15     0.014334
margin16     0.000888
margin17     0.010787
margin18     0.021390
margin19     0.013482
```

There is a difference in std for the features, so we can do standarization to the data. For Example:

- shape1 : 0.000275
- margin55 : 0.021317

## Standardization of the data:

```
# copy of datasets
X_train_stand = X_train.copy()
X_test_stand = X_test.copy()

# apply standardization on numerical features
for i in X_train.columns:

    # fit on training data column
    scale = StandardScaler().fit(X_train_stand[[i]])

    # transform the training data column
    X_train_stand[i] = scale.transform(X_train_stand[[i]])

    # transform the testing data column
    X_test_stand[i] = scale.transform(X_test_stand[[i]])
```

# Part II: Training a neural network

## My model Architecture:

First Trial: a 3-layer MLP model (one input layer, one hidden layer with tanh activation and one output layer)

```
model = Sequential(
    [
        Dense(units=192, activation="relu", input_shape=(X_train.shape[-1],) ),
        # randomly delete 30% of the input units below
        Dropout(0.2),
        Dense(units=256, activation="tanh"),
        # the output layer, with a single neuron
        Dense(units=99, activation="softmax"),
    ]
)

# save the initial weights for later
initial_weights = model.get_weights()
```

```
model.summary()
```

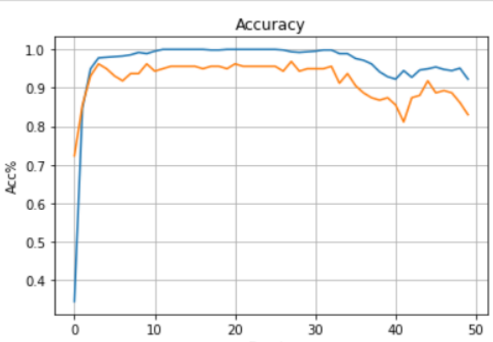
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 192)               37056

 dropout (Dropout)           (None, 192)               0

 dense_1 (Dense)             (None, 256)               49408

 dense_2 (Dense)             (None, 99)                25443

=================================================================
Total params: 111,907
Trainable params: 111,907
Non-trainable params: 0
_________________________________________________________________
```

# Training the model:

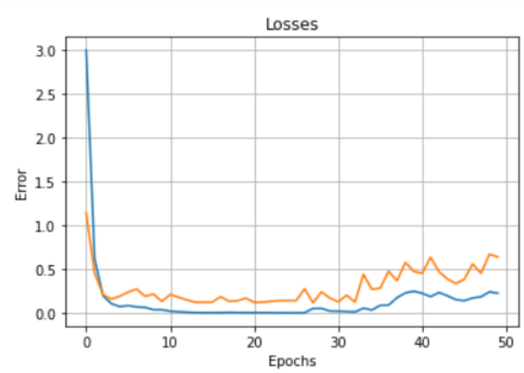
```
In [30]: tf.random.set_seed(42)
         filepath = 'model1.hdf5'

         earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
         checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
         # reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=7, verbose=1, epsilon=1e-4, mode='min')

         opt=optimizers.Adam(learning_rate=0.01)
         model.compile(optimizer=opt ,loss="sparse_categorical_crossentropy", metrics=["accuracy"] )
         #model.summary()
         #Train Model
         result=model.fit(X_train_stand , y_train ,batch_size=64 ,epochs=50 ,
                         callbacks=[earlyStopping, checkpoint_conv], validation_split=0.2)
```

```
 1/10 [==>...........................] - ETA: 0s - loss: 0.0829 - accuracy: 0.9688
Epoch 47: val_accuracy did not improve from 0.96855
10/10 [==============================] - 0s 8ms/step - loss: 0.1703 - accuracy: 0.9479 - val_loss: 0.5583 - val_accuracy: 0.89
31
Epoch 48/50
 1/10 [==>...........................] - ETA: 0s - loss: 0.0621 - accuracy: 0.9688
Epoch 48: val_accuracy did not improve from 0.96855
10/10 [==============================] - 0s 8ms/step - loss: 0.1845 - accuracy: 0.9447 - val_loss: 0.4520 - val_accuracy: 0.88
68
Epoch 49/50
 7/10 [====================>.........] - ETA: 0s - loss: 0.1991 - accuracy: 0.9509
Epoch 49: val_accuracy did not improve from 0.96855
10/10 [==============================] - 0s 12ms/step - loss: 0.2401 - accuracy: 0.9510 - val_loss: 0.6696 - val_accuracy: 0.8
616
Epoch 50/50
 1/10 [==>...........................] - ETA: 0s - loss: 0.1980 - accuracy: 0.9219
Epoch 50: val_accuracy did not improve from 0.96855
10/10 [==============================] - 0s 8ms/step - loss: 0.2247 - accuracy: 0.9226 - val_loss: 0.6378 - val_accuracy: 0.83
02
```

```
plt.plot(result.history["accuracy"])
plt.plot(result.history["val_accuracy"])
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Acc%")
plt.grid()
```




```
plt.plot(result.history["loss"])
plt.plot(result.history["val_loss"])
plt.title("Losses")
plt.xlabel("Epochs")
plt.ylabel("Error")
plt.grid()
```



# Evaluate on test data:

```
model.evaluate(X_test_stand,y_test)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.4562 - accuracy: 0.8687

[0.45618608593940735, 0.868686854839325]
```

# Fine Tuning to find the best hyperparameters using Keras Tuner:

## Fine Tuning to find the best hyperparameters

Using the same model will try to find the best possible heperparam to acheive the best performance

- Tring Different Activation fun for the first layer: ['relu', 'tanh', 'sigmoid']
- Tring Different Dropout vaues for the first layer: [ min_value=0.0,max_value=0.5,default=0.25,step=0.05]
- Hidden size: Try using different number of hidden nodes in the second layer: [min_value=32, max_value=512, step=32]
- Tune the learning rate for the optimizer, Choose an optimal value from 0.01, 0.001, or 0.0001 and Also i will use learning rate scheduler later.

In [34]:
```python
def model_builder(hp):

    model = Sequential()
    model.add(Dense(units=192, activation = hp.Choice('dense_activation',values=['relu', 'tanh', 'sigmoid'],
                                                      default='relu')))
                    #input_shape=(X_train.shape[-1],) ))

    model.add(Dropout(hp.Float('dropout', min_value=0.0,max_value=0.5,default=0.25,step=0.05)))

    # Tune the number of units in the first Dense layer
    # Choose an optimal value between 32-512
    hp_units = hp.Int('units', min_value=32, max_value=512, step=32)

    model.add(Dense(units=hp_units, activation='tanh'))
    model.add(Dense(99 , activation="softmax"))

    # Tune the learning rate for the optimizer
    # Choose an optimal value from 0.01, 0.001, or 0.0001
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    model.compile(optimizer=Adam(learning_rate=hp_learning_rate),
                  loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
#     model.fit(X_train_stand , y_train ,batch_size= hp.Choice('batch_sizee',values=[32, 64, 128], default = 32
#               ,epochs=50 ,
#                callbacks=[earlyStopping, mcp_save], validation_split=0.2)

    return model
```

In [35]:
```
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                 #    factor=3,
                     directory='models',
                     project_name='modelN')
```

In [36]:
```
stop_early = EarlyStopping(monitor='val_loss', patience=5)
tensorboard = TensorBoard("/models/tb_logs")
tuner.search(X_train_stand, y_train, epochs=50, validation_split=0.2, callbacks=[stop_early,tensorboard])

# Get the optimal hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
The hyperparameter search is complete. The optimal number of units in the second densely-connected
layer is {best_hps.get('units')} and the optimal learning rate for the optimizer
is {best_hps.get('learning_rate')} Activation function for first layer is {best_hps.get('dense_activation')} and
Dropout after first layer by: {best_hps.get('dropout')}.
""")
```

```
Trial 30 Complete [00h 00m 08s]
val_accuracy: 0.9119496941566467

Best val_accuracy So Far: 0.9748427867889404
Total elapsed time: 00h 03m 13s
INFO:tensorflow:Oracle triggered exit

The hyperparameter search is complete. The optimal number of units in the second densely-connected
layer is 192 and the optimal learning rate for the optimizer
is 0.01 Activation function for first layer is tanh and
Dropout after first layer by: 0.2.
```

In [42]:
```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 4
dense_activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh', 'sigmoid'], 'ordered': False}
dropout (Float)
{'default': 0.25, 'conditions': [], 'min_value': 0.0, 'max_value': 0.5, 'step': 0.05, 'sampling': None}
units (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
learning_rate (Choice)
{'default': 0.01, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
```

In [43]:
```
# build the best model and train it
tf.random.set_seed(42)
filepath = 'model2.hdf5'
earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

print("[INFO] training the best model...")
modell = tuner.hypermodel.build(best_hps)
H = modell.fit(X_train_stand, y_train, validation_split=0.2, batch_size=32, epochs=50,
              callbacks=[earlyStopping, checkpoint_conv], verbose=1)
```

```
15/20 [=====================>........] - ETA: 0s - loss: 2.8543e-04 - accuracy: 1.0000
Epoch 47: val_accuracy did not improve from 0.97484
20/20 [==============================] - 0s 6ms/step - loss: 2.9208e-04 - accuracy: 1.0000 - val_loss: 0.0970 - val_accuracy:
0.9686
Epoch 48/50
15/20 [=====================>........] - ETA: 0s - loss: 3.1494e-04 - accuracy: 1.0000
Epoch 48: val_accuracy did not improve from 0.97484
20/20 [==============================] - 0s 6ms/step - loss: 3.0893e-04 - accuracy: 1.0000 - val_loss: 0.0933 - val_accuracy:
0.9686
Epoch 49/50
14/20 [====================>.........] - ETA: 0s - loss: 3.1551e-04 - accuracy: 1.0000
Epoch 49: val_accuracy did not improve from 0.97484
20/20 [==============================] - 0s 6ms/step - loss: 3.1088e-04 - accuracy: 1.0000 - val_loss: 0.0937 - val_accuracy:
0.9623
Epoch 50/50
14/20 [====================>.........] - ETA: 0s - loss: 3.2013e-04 - accuracy: 1.0000
Epoch 50: val_accuracy did not improve from 0.97484
20/20 [==============================] - 0s 6ms/step - loss: 3.0799e-04 - accuracy: 1.0000 - val_loss: 0.0944 - val_accuracy:
0.9686
```

## Evaluate on test data:

In [46]:
```
modell.evaluate(X_test_stand,y_test)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.0561 - accuracy: 0.9899
```

Out[46]:
```
[0.05614694952964783, 0.9898989796638489]
```

## Trying different hyperparameters:

The optimal number of units in the second densely-connected

layer is 384 and the optimal learning rate for the optimizer

is 0.001 Activation function for first layer is tanh and

Dropout after first layer by: 0.1.

- batch_size = [16,32,64]
- Optimizer: Try using different optimizers such as Adam, SGD RMSProp
- Regularization (weight decay): L2 regularization can be specified by setting the weight_decay parameter in optimizer. [0.001, 0.01, 0.1]
- LearningRateScheduler

I will manually try 9 different combination of these hyperparameters and choose the best combination of them with the ones from the previous Tuning.

## 1

- batch_ size = 16
- Optimizer = Adam
- decay = 0.001

In [62]:
```
tf.random.set_seed(42)
filepath = '1model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=Adam(decay = 0.001 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =Model.fit(X_train_stand , y_train ,batch_size=16 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```

## 2

- batch_ size = 16
- Optimizer = Adam
- decay = 0.01

In [63]:
```
tf.random.set_seed(42)
filepath = '2model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=Adam(decay = 0.01 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =Model.fit(X_train_stand , y_train ,batch_size=16 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```

## 3

- batch_ size = 16
- Optimizer = Adam
- decay = 0.001

In [64]:
```
tf.random.set_seed(42)
filepath = '3model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=Adam(decay = 0.001 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =Model.fit(X_train_stand , y_train ,batch_size=16 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```

### 4

- batch_ size = 16
- Optimizer = SGD
- decay = 0.01

In [65]:
```
tf.random.set_seed(42)
filepath = '4model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=SGD(decay = 0.1 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =Model.fit(X_train_stand , y_train ,batch_size=16 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```

### 5

- batch_ size = 32
- Optimizer = SGD
- decay = 0.01

In [66]:
```
tf.random.set_seed(42)
filepath = '5model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=SGD(decay = 0.01 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =Model.fit(X_train_stand , y_train ,batch_size=32 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```

### 6

- batch_ size = 64
- Optimizer = SGD
- decay = 0.01

In [67]:
```
tf.random.set_seed(42)
filepath = '6model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=SGD(decay = 0.01 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =Model.fit(X_train_stand , y_train ,batch_size=64 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```

## 7

- batch_ size = 64
- Optimizer = Adam
- decay = 0.01

In [68]:
```
tf.random.set_seed(42)
filepath = '7model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=Adam(decay = 0.01 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =Model.fit(X_train_stand , y_train ,batch_size=64 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```

## 8

- batch_ size = 64
- Optimizer = RMSProp
- decay = 0.001

In [71]:
```
tf.random.set_seed(42)
filepath = '8model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=RMSprop(decay = 0.001 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =Model.fit(X_train_stand , y_train ,batch_size=64 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```

## 9

- batch_ size = 16
- Optimizer = RMSProp
- decay = 0.01

In [73]:
```
tf.random.set_seed(42)
filepath = '9model.hdf5'

earlyStopping = EarlyStopping(monitor='val_loss', patience=40, verbose=0, mode='min')
checkpoint_conv = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
LRsched = LearningRateScheduler(scheduler)

Model.compile(optimizer=RMSprop(decay = 0.01 , learning_rate=0.01),  # best LR
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])
#model.summary()
#Train Model
Hist =model.fit(X_train_stand , y_train ,batch_size=16 ,epochs=50 ,
               callbacks=[earlyStopping, checkpoint_conv, LRsched], validation_split=0.2)

Model.evaluate(X_test_stand,y_test)
```