Keras MLP

In [35]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD
```

In [36]:
```
df = pd.read_excel("Dry_Bean_Dataset.xlsx")
df = df.drop_duplicates()
X = df.iloc[:,:16]
y = df.iloc[:,16:]
y = y.reset_index().drop(columns = "index")
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns = X.columns)
pt = PowerTransformer(method = "yeo-johnson")
X = pd.DataFrame(pt.fit_transform(X), columns = X.columns)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 101)
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)
y_train = tf.keras.utils.to_categorical(y_train, num_classes=7)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=7)
```

```
C:\Users\matth\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

In [55]:
```
model = tf.keras.Sequential()
model.add(layers.Dense(16, input_dim = X_train.shape[1], activation = "relu")) # input layer requires input_dim
model.add(layers.Dense(32, activation = "tanh"))
model.add(layers.Dense(64, activation = "tanh"))
model.add(layers.Dense(32, activation = "tanh"))
model.add(layers.Dense(16, activation = "tanh"))
model.add(layers.Dense(7, activation='softmax'))
model.summary()
model.compile(loss="kullback_leibler_divergence", optimizer= "adam", metrics=['accuracy'])
es = tf.keras.callbacks.EarlyStopping(monitor='loss', min_delta=0.00005, patience=3, verbose=1, mode='auto')
with tf.device('/cpu:0'):
    model.fit(X_train, y_train, epochs = 100, shuffle = True, batch_size=32, verbose=1, callbacks=[es])
    score = model.evaluate(X_test, y_test, verbose=0)
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_82 (Dense)            (None, 16)                272

 dense_83 (Dense)            (None, 32)                544

 dense_84 (Dense)            (None, 64)                2112

 dense_85 (Dense)            (None, 32)                2080

 dense_86 (Dense)            (None, 16)                528

 dense_87 (Dense)            (None, 7)                 119

=================================================================
Total params: 5,655
Trainable params: 5,655
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
339/339 [==============================] - 1s 3ms/step - loss: 0.5353 - accuracy: 0.8454
Epoch 2/100
339/339 [==============================] - 1s 3ms/step - loss: 0.2563 - accuracy: 0.9166
Epoch 3/100
339/339 [==============================] - 1s 3ms/step - loss: 0.2258 - accuracy: 0.9229
Epoch 4/100
339/339 [==============================] - 1s 3ms/step - loss: 0.2186 - accuracy: 0.9209
Epoch 5/100
339/339 [==============================] - 1s 3ms/step - loss: 0.2117 - accuracy: 0.9245
Epoch 6/100
339/339 [==============================] - 1s 3ms/step - loss: 0.2074 - accuracy: 0.9243
Epoch 7/100
339/339 [==============================] - 1s 3ms/step - loss: 0.2000 - accuracy: 0.9290
Epoch 8/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1989 - accuracy: 0.9272
Epoch 9/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1974 - accuracy: 0.9283
Epoch 10/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1948 - accuracy: 0.9297
Epoch 11/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1930 - accuracy: 0.9311
Epoch 12/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1909 - accuracy: 0.9263
Epoch 13/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1883 - accuracy: 0.9311
Epoch 14/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1894 - accuracy: 0.9296
Epoch 15/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1877 - accuracy: 0.9301
Epoch 16/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1857 - accuracy: 0.9312
Epoch 17/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1852 - accuracy: 0.9315
Epoch 18/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1839 - accuracy: 0.9315
Epoch 19/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1816 - accuracy: 0.9327
Epoch 20/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1823 - accuracy: 0.9301
Epoch 21/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1813 - accuracy: 0.9323
Epoch 22/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1804 - accuracy: 0.9321
Epoch 23/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1803 - accuracy: 0.9331
Epoch 24/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1805 - accuracy: 0.9334
Epoch 25/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1787 - accuracy: 0.9349
Epoch 26/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1741 - accuracy: 0.9344
Epoch 27/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1740 - accuracy: 0.9343
Epoch 28/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1760 - accuracy: 0.9340
Epoch 29/100
339/339 [==============================] - 1s 4ms/step - loss: 0.1733 - accuracy: 0.9355
Epoch 30/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1738 - accuracy: 0.9335
Epoch 31/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1715 - accuracy: 0.9375
Epoch 32/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1724 - accuracy: 0.9369
Epoch 33/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1710 - accuracy: 0.9370
Epoch 34/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1702 - accuracy: 0.9368
Epoch 35/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1683 - accuracy: 0.9364
Epoch 36/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1713 - accuracy: 0.9348
Epoch 37/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1680 - accuracy: 0.9359
Epoch 38/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1650 - accuracy: 0.9363
Epoch 39/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1647 - accuracy: 0.9377
Epoch 40/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1660 - accuracy: 0.9375
Epoch 41/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1635 - accuracy: 0.9375
Epoch 42/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1633 - accuracy: 0.9379
Epoch 43/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1621 - accuracy: 0.9372
Epoch 44/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1615 - accuracy: 0.9383
Epoch 45/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1609 - accuracy: 0.9382
Epoch 46/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1595 - accuracy: 0.9382
Epoch 47/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1603 - accuracy: 0.9388
Epoch 48/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1594 - accuracy: 0.9407
Epoch 49/100
339/339 [==============================] - 1s 3ms/step - loss: 0.1597 - accuracy: 0.9396
Epoch 49: early stopping
```

In [56]:
```
predictions = model.predict(X_test)
y_pred=np.argmax(predictions, axis=1)
tests=np.argmax(y_test, axis=1)
print(confusion_matrix(tests, y_pred))
print(classification_report(tests,y_pred,target_names=y.Class.unique()))
```

```
85/85 [==============================] - 0s 2ms/step
[[246   0   8   0   0   2   4]
 [  0 115   0   0   0   0   0]
 [  9   1 324   0   1   1   3]
 [  0   0   0 649   0   8  56]
 [  0   0   5   3 332   0   8]
 [  1   0   0  11   0 388   9]
 [  1   0   3  30   5   7 479]]
              precision    recall  f1-score   support

       SEKER       0.96      0.95      0.95       260
    BARBUNYA       0.99      1.00      1.00       115
      BOMBAY       0.95      0.96      0.95       339
        CALI       0.94      0.91      0.92       713
       HOROZ       0.98      0.95      0.97       348
        SIRA       0.96      0.95      0.95       409
    DERMASON       0.86      0.91      0.88       525

    accuracy                           0.94      2709
   macro avg       0.95      0.95      0.95      2709
weighted avg       0.94      0.94      0.94      2709
```

In [57]:
```
metrics.accuracy_score(tests, y_pred)
```

Out[57]:
```
0.9350313768918422
```