Speaker Recognition

Term project for Machine Learning UoT course

The objective of this Notebook is to predict the speaker of an English digit using a Keras model.

This Notebook has three main sections.

1. Extract the features of the WAV files, save them into CSV files and store them into Pandas
2. The model will be trained based on the data set created by [https://github.com/Jakobovski/free-spoken-digit-dataset (https://github.com/Jakobovski/free-spoken-digit-dataset)](https://github.com/Jakobovski/free-spoken-digit-dataset)
3. The model will be re-trained based on the previous data plus the recordings made by Ankor (Indian accent), Caroline (Canadian female child accent) and Rodolfo (Brazilian accent)

In [ ]:

```
```

Section 1

The output of this section is the CSV files with the data to be handle by the model

```
trainData     : ../data/recordings/train
testData      : ../data/recordings/test
moreTrainData : ../data/recordings/moreSpeakersTrain
moreTestData  : ../data/recordings/moreSpeakersTest
```

In [1]:

```
# If true, the WAV files will be read and their features will be saved in the
 CSV files
# As this is the most time consuming task, only enable it if you don't have th
e CSV files yet
CREATE_CSV_FILES = True
```

In [2]:

```
# Defines the names of the CSV files
TRAIN_CSV_FILE = "train.csv"
TEST_CSV_FILE = "test.csv"
MORE_TRAIN_CSV_FILE = "more_train.csv"
MORE_TEST_CSV_FILE = "more_test.csv"
```

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
import librosa
import csv
import os

def extractWavFeatures(soundFilesFolder, csvFileName):
    print("The features of the files in the folder "+soundFilesFolder+" will b
e saved to "+csvFileName)
    header = 'filename chroma_stft rmse spectral_centroid spectral_bandwidth r
olloff zero_crossing_rate'
    for i in range(1, 21):
        header += f' mfcc{i}'
    header += ' label'
    header = header.split()
    print('CSV Header: ', header)
    file = open(csvFileName, 'w', newline='')
    #with file:
    writer = csv.writer(file)
    writer.writerow(header)
    genres = '1 2 3 4 5 6 7 8 9 0'.split()
    for filename in os.listdir(soundFilesFolder):
        number = f'{soundFilesFolder}/{filename}'
        y, sr = librosa.load(number, mono=True, duration=30)
        # remove leading and trailing silence
        y, index = librosa.effects.trim(y)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
        rmse = librosa.feature.rms(y=y)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)
        to_append = f'{filename} {np.mean(chroma_stft)} {np.mean(rmse)} {np.me
an(spec_cent)} {np.mean(spec_bw)} {np.mean(rolloff)} {np.mean(zcr)}'
        for e in mfcc:
            to_append += f' {np.mean(e)}'
        writer.writerow(to_append.split())
    file.close()
    print("End of extractWavFeatures")

if (CREATE_CSV_FILES == True):
    extractWavFeatures("../data/recordings/train", TRAIN_CSV_FILE)
    extractWavFeatures("../data/recordings/test", TEST_CSV_FILE)
    extractWavFeatures("../data/recordings/moreSpeakersTrain", MORE_TRAIN_CSV_
FILE)
    extractWavFeatures("../data/recordings/moreSpeakersTest", MORE_TEST_CSV_FI
LE)
    print("CSV files are created")
else:
    print("CSV files creation is skipped")
```

```
The features of the files in the folder ../data/recordings/train will be save
d to train.csv
CSV Header:  ['filename', 'chroma_stft', 'rmse', 'spectral_centroid', 'spectr
al_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2', 'mfcc3', 'm
fcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfcc10', 'mfcc11', 'mfcc
12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfcc17', 'mfcc18', 'mfcc19', 'm
fcc20', 'label']
End of extractWavFeatures
The features of the files in the folder ../data/recordings/test will be saved
to test.csv
CSV Header:  ['filename', 'chroma_stft', 'rmse', 'spectral_centroid', 'spectr
al_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2', 'mfcc3', 'm
fcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfcc10', 'mfcc11', 'mfcc
12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfcc17', 'mfcc18', 'mfcc19', 'm
fcc20', 'label']
End of extractWavFeatures
The features of the files in the folder ../data/recordings/moreSpeakersTrain
will be saved to more_train.csv
CSV Header:  ['filename', 'chroma_stft', 'rmse', 'spectral_centroid', 'spectr
al_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2', 'mfcc3', 'm
fcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfcc10', 'mfcc11', 'mfcc
12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfcc17', 'mfcc18', 'mfcc19', 'm
fcc20', 'label']
End of extractWavFeatures
The features of the files in the folder ../data/recordings/moreSpeakersTest w
ill be saved to more_test.csv
CSV Header:  ['filename', 'chroma_stft', 'rmse', 'spectral_centroid', 'spectr
al_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2', 'mfcc3', 'm
fcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfcc10', 'mfcc11', 'mfcc
12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfcc17', 'mfcc18', 'mfcc19', 'm
fcc20', 'label']
End of extractWavFeatures
CSV files are created
```

In [4]:
```
#Reading a dataset and convert file name to corresbonding umnber

import pandas as pd
import csv
from sklearn import preprocessing

def preProcessData(csvFileName):
    print(csvFileName+ " will be preprocessed")
    data = pd.read_csv(csvFileName)
    # we have six speakers:
    # 0: Jackson
    # 1: Nicolas
    # 2: Theo
    # 3: Ankur
    # 4: Caroline
    # 5: Rodolfo
    filenameArray = data['filename']
    speakerArray = []
    #print(filenameArray)
    for i in range(len(filenameArray)):
        speaker = filenameArray[i][2]
        #print(speaker)
        if speaker == "j":
            speaker = "0"
        elif speaker == "n":
            speaker = "1"
        elif speaker == "t":
            speaker = "2"
        elif speaker == "a":
            speaker = "3"
        elif speaker == "c":
            speaker = "4"
        elif speaker == "r":
            speaker = "5"
        else:
            speaker = "6"
        #print(speaker)
        speakerArray.append(speaker)
    data['number'] = speakerArray
    #Dropping unnecessary columns
    data = data.drop(['filename'],axis=1)
    data = data.drop(['label'],axis=1)
    data = data.drop(['chroma_stft'],axis=1)
    data.shape

    print("Preprocessing is finished")
    print(data.head())
    return data

trainData = preProcessData(TRAIN_CSV_FILE)
testData = preProcessData(TEST_CSV_FILE)
moreTrainData = preProcessData(MORE_TRAIN_CSV_FILE)
moreTestData = preProcessData(MORE_TEST_CSV_FILE)
```

```
train.csv will be preprocessed
Preprocessing is finished
       rmse  spectral_centroid  spectral_bandwidth      rolloff  \
0  0.112672         741.829081          758.492178  1438.494873
1  0.090344         635.610880          670.336296  1160.452403
2  0.091456         667.786694          732.606545  1257.180176
3  0.087751         712.304185          731.292437  1449.104818
4  0.096603         844.363886          777.868127  1569.583263

   zero_crossing_rate       mfcc1       mfcc2      mfcc3      mfcc4  \
0            0.034023 -295.578461  189.853683 -19.606564   6.078507
1            0.033458 -339.148743  204.005249  -7.485526  14.297898
2            0.033268 -327.507416  195.596924  -3.994768  21.315840
3            0.035916 -320.809937  200.023743  -8.186146  12.661074
4            0.049465 -315.801300  195.674118 -13.324564   3.544238

       mfcc5  ...     mfcc12    mfcc13     mfcc14     mfcc15    mfcc16  \
0  22.067095  ... -25.725817 -5.172223  -8.323026 -10.299589 -0.144793
1  20.885136  ... -23.196365 -1.290891  -5.515564 -15.416287  0.405876
2  18.372593  ... -18.677113 -3.098450 -10.447586 -10.053793  3.248016
3  15.654718  ... -20.832333 -1.118007  -6.681235 -11.685319  2.010999
4  12.279986  ... -18.158249  6.031695  -6.353736 -15.983871  1.465030

      mfcc17     mfcc18    mfcc19     mfcc20  number
0  -9.017329  -4.569392  2.881349 -15.627436       0
1  -3.624587 -11.204143 -0.096359  -6.751650       0
2 -11.686995 -10.726046  6.857377  -9.067446       0
3  -5.946658  -6.905020  4.136240  -9.614882       0
4  -5.109472  -8.666434  5.026890  -5.346444       0

[5 rows x 26 columns]
test.csv will be preprocessed
Preprocessing is finished
       rmse  spectral_centroid  spectral_bandwidth      rolloff  \
0  0.095394         756.450712          761.875940  1463.941148
1  0.040176         791.046914         1039.695939  2027.709961
2  0.006984         958.934867          941.639039  2084.106445
3  0.071547         759.877794          899.957003  1427.553489
4  0.030382         968.793389         1024.834851  1911.968994

   zero_crossing_rate       mfcc1       mfcc2      mfcc3      mfcc4  \
0            0.037296 -328.263885  180.479416  -0.485355  15.525293
1            0.031440 -385.602570  189.328186 -37.268154  59.937920
2            0.040039 -542.812622  217.971329 -62.197266  21.537390
3            0.030429 -355.530396  204.388977 -20.676432  26.671131
4            0.045654 -376.499390  237.137833 -59.964413  37.715607

       mfcc5  ...     mfcc12     mfcc13     mfcc14     mfcc15     mfcc16  \
0  20.992447  ... -15.426966   7.284101  -6.443027 -13.377846  -2.407696
1  45.049831  ... -31.051588   3.420474  -9.762264 -11.220519  12.306476
2  37.756233  ... -26.797358   5.341060 -12.159102 -14.180812   9.346475
3  15.797892  ... -18.198524   4.029843 -10.552087 -21.039103  -5.634320
4  21.510382  ... -21.805593  13.740063  -6.738161  -9.305484  13.205662

      mfcc17     mfcc18    mfcc19     mfcc20  number
0 -12.902534 -10.437113 -1.025342 -15.457672       0
1  -5.082399  -3.775387  9.707520  -8.757109       1
```

```
2 -10.899978  -9.715154  5.997578 -12.574761       2
3 -14.788965 -11.016036 -1.313916 -16.993853       0
4 -11.917943  -7.877903  9.777577 -10.397771       1

[5 rows x 26 columns]
more_train.csv will be preprocessed
Preprocessing is finished
       rmse  spectral_centroid  spectral_bandwidth      rolloff  \
0  0.039759        1358.208628         1890.243941  2540.917969
1  0.302424         879.994019         1137.986581  1826.501859
2  0.026959        1237.544903         1219.890113  2372.379244
3  0.027274        1523.814892         2030.693021  3053.946533
4  0.304633         625.579402          799.806332  1029.825439

   zero_crossing_rate       mfcc1       mfcc2      mfcc3      mfcc4  \
0            0.093363 -349.631744  135.204880  19.397516  12.044560
1            0.031423 -225.136642  150.581146 -11.930015   5.277394
2            0.059871 -378.096527  180.817047 -41.006123  22.502394
3            0.084253 -403.760406  127.670433  23.293980   7.495267
4            0.023584 -228.231903  191.904144   1.268919  17.871376

       mfcc5  ...     mfcc12     mfcc13     mfcc14     mfcc15     mfcc16  \
0  -8.521679  ... -12.872719  -3.118359  -4.095297  -8.339793  -4.422189
1 -16.297689  ... -13.114679 -13.413434 -17.883400 -17.694012 -18.091360
2  -8.963315  ... -32.601864  -4.715013  -8.889856 -11.225335 -10.951420
3  -6.774144  ...  -6.803737  -3.132436  -2.999972 -14.086405  -9.653265
4  -4.571987  ...  -5.878148  -9.646009 -15.565687 -17.903820 -14.293053

      mfcc17     mfcc18     mfcc19     mfcc20  number
0  -2.988979  -0.864654  -4.008632   3.243911       3
1 -14.684992 -16.672678 -10.986701 -10.445865       4
2  -8.972343  -4.716431  -5.655877  -7.133625       5
3  -7.696736  -2.747038 -11.928693  -4.345026       3
4 -14.907639  -9.752651 -10.333107  -7.938378       4

[5 rows x 26 columns]
more_test.csv will be preprocessed
Preprocessing is finished
       rmse  spectral_centroid  spectral_bandwidth      rolloff  \
0  0.070496        1736.761057         1697.429353  2782.932447
1  0.178100         986.240750         1243.094994  2207.479581
2  0.159381        1140.769641         1151.467042  2283.347731
3  0.102912        1270.788895         1624.572386  2442.362154
4  0.243922         839.268659          985.135107  1423.724724

   zero_crossing_rate       mfcc1       mfcc2      mfcc3      mfcc4  \
0            0.095979 -270.462311  120.525047 -16.563780  25.340765
1            0.045351 -230.533142  171.157135  -5.515658  20.675375
2            0.056547 -261.164734  169.949051 -12.116495   0.249355
3            0.055965 -264.190643  154.578873  -9.901300   1.276820
4            0.037626 -191.595886  189.683578 -29.157555  16.972124

       mfcc5  ...     mfcc12    mfcc13     mfcc14     mfcc15     mfcc16  \
0  -8.705835  ... -20.140141 -1.016966 -11.883263  -8.389129  -4.095136
1  -3.108402  ... -14.097899 -9.547957  -9.970568 -15.526528 -17.028362
2   8.262510  ... -26.140615 -3.177802  -5.860640 -14.931437 -12.212286
3  -4.987144  ...  -9.880266 -0.010168 -10.896443  -5.682155  -0.433506
```

```
4 -13.665516  ... -14.553379 -6.711682 -15.045198 -11.206120 -11.227142

      mfcc17     mfcc18     mfcc19     mfcc20  number
0   2.231521  -6.269879  -7.276823   0.429325       3
1 -13.014733  -9.094010   1.289093  -5.188507       4
2  -5.934423  -2.185390  -7.319934 -12.082070       5
3   6.786911 -10.947194 -11.484864  -0.297753       3
4  -9.380514 -11.611271  -7.099937  -6.555156       4

[5 rows x 26 columns]
```

In [ ]:

Section 2

There are 50 recordings for each digit for each speaker: Jackson, Nicolas and Theo (total 1500 recordings)

Training data has 49 recordings for each digit for each speaker: 1470 recordings total. Test data has 1 recordings for each digit for each speaker: 30 recordings total.

The data used here comes from the recordings stored in:

- ../data/recordings/train
- ../data/recordings/test

The model will be trained to predict the speker of a digit.

In [5]:

```python
# Splitting the dataset into training, validation and testing dataset
from sklearn.model_selection import train_test_split
X = np.array(trainData.iloc[:, :-1], dtype = float)
y = trainData.iloc[:, -1]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_
state=42)


X_test = np.array(testData.iloc[:, :-1], dtype = float)
y_test = testData.iloc[:, -1]

print("Y from training data:", y_train.shape)
print("Y from validation data:", y_val.shape)
print("Y from test data:", y_test.shape)
```

```
Y from training data: (1029,)
Y from validation data: (441,)
Y from test data: (30,)
```

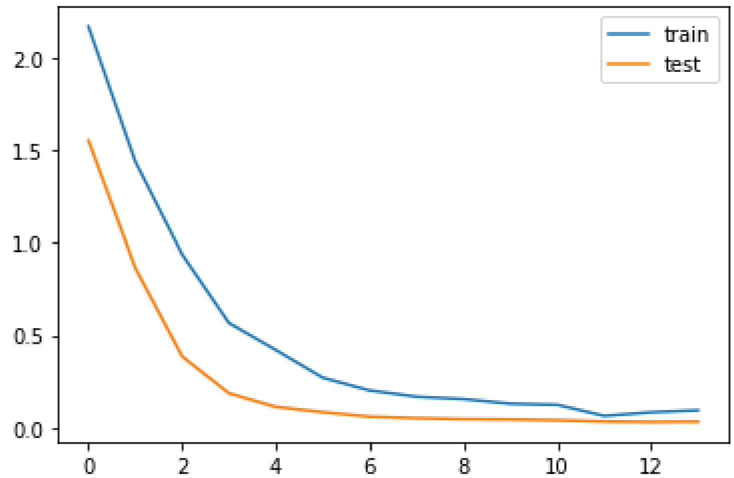
In [6]:
```
#Normalizing the dataset
from sklearn.preprocessing import StandardScaler
import numpy as np
scaler = StandardScaler()
X_train = scaler.fit_transform( X_train )
X_val = scaler.transform( X_val )
X_test = scaler.transform( X_test )

print("X from training data", X_train.shape)
print("X from validation data", X_val.shape)
print("X from test data", X_test.shape)
```

```
X from training data (1029, 25)
X from validation data (441, 25)
X from test data (30, 25)
```

In [7]:
```
#Creating a Model
from keras import models
from keras import layers
import keras

# model 1
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_shape=(X_train.shape[1
],)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

# Learning Process of a model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# simple early stopping
from keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)

#Train with early stopping to avoid overfitting
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=50,
                    batch_size=128,
                    callbacks=[es])
```

```
Using TensorFlow backend.

Train on 1029 samples, validate on 441 samples
Epoch 1/50
1029/1029 [==============================] - 0s 187us/step - loss: 2.1741 - a
ccuracy: 0.2439 - val_loss: 1.5546 - val_accuracy: 0.7029
Epoch 2/50
1029/1029 [==============================] - 0s 31us/step - loss: 1.4414 - ac
curacy: 0.5384 - val_loss: 0.8647 - val_accuracy: 0.9274
Epoch 3/50
1029/1029 [==============================] - 0s 33us/step - loss: 0.9387 - ac
curacy: 0.6774 - val_loss: 0.3857 - val_accuracy: 0.9728
Epoch 4/50
1029/1029 [==============================] - 0s 37us/step - loss: 0.5667 - ac
curacy: 0.8066 - val_loss: 0.1855 - val_accuracy: 0.9751
Epoch 5/50
1029/1029 [==============================] - 0s 48us/step - loss: 0.4205 - ac
curacy: 0.8669 - val_loss: 0.1118 - val_accuracy: 0.9751
Epoch 6/50
1029/1029 [==============================] - 0s 42us/step - loss: 0.2702 - ac
curacy: 0.9145 - val_loss: 0.0831 - val_accuracy: 0.9773
Epoch 7/50
1029/1029 [==============================] - 0s 46us/step - loss: 0.2007 - ac
curacy: 0.9475 - val_loss: 0.0588 - val_accuracy: 0.9864
Epoch 8/50
1029/1029 [==============================] - 0s 60us/step - loss: 0.1665 - ac
curacy: 0.9582 - val_loss: 0.0507 - val_accuracy: 0.9887
Epoch 9/50
1029/1029 [==============================] - 0s 52us/step - loss: 0.1537 - ac
curacy: 0.9572 - val_loss: 0.0456 - val_accuracy: 0.9909
Epoch 10/50
1029/1029 [==============================] - 0s 56us/step - loss: 0.1288 - ac
curacy: 0.9611 - val_loss: 0.0438 - val_accuracy: 0.9909
Epoch 11/50
1029/1029 [==============================] - 0s 86us/step - loss: 0.1228 - ac
curacy: 0.9640 - val_loss: 0.0393 - val_accuracy: 0.9909
Epoch 12/50
1029/1029 [==============================] - 0s 56us/step - loss: 0.0626 - ac
curacy: 0.9864 - val_loss: 0.0326 - val_accuracy: 0.9887
Epoch 13/50
1029/1029 [==============================] - 0s 53us/step - loss: 0.0822 - ac
curacy: 0.9757 - val_loss: 0.0300 - val_accuracy: 0.9887
Epoch 14/50
1029/1029 [==============================] - 0s 49us/step - loss: 0.0931 - ac
curacy: 0.9708 - val_loss: 0.0316 - val_accuracy: 0.9909
Epoch 00014: early stopping
```

In [8]:
```
# plot training history
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



Auxiliary functions to show the results

In [9]:
```
def getSpeaker(speaker):
    speaker = str(speaker)
    if speaker == "0":
        return "Jackson"
    elif speaker == "1":
        return "Nicola"
    elif speaker == "2":
        return "Theo"
    elif speaker == "3":
        return "Ankur"
    elif speaker == "4":
        return "Caroline"
    elif speaker == "5":
        return "Rodolfo"
    else:
        speaker = "Unknown"

def printPrediction(X_data, y_data, printDigit):
    print('\n# Generate predictions')
    for i in range(len(y_data)):
        prediction = getSpeaker(model.predict_classes(X_data[i:i+1])[0])
        speaker = getSpeaker(y_data[i])
        if printDigit == True:
            print("Number={0:d}, y={1:10s}- prediction={2:10s}- match={3}".for
mat(i, speaker, prediction, speaker==prediction))
        else:
            print("y={0:10s}- prediction={1:10s}- match={2}".format(speaker, p
rediction, speaker==prediction))
```

In [10]:
```
import numpy as np
from keras import backend as K
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix

def report(X_data, y_data):
    #Confution Matrix and Classification Report
    Y_pred = model.predict_classes(X_data)
    y_test_num = y_data.astype(np.int64)
    conf_mt = confusion_matrix(y_test_num, Y_pred)
    print(conf_mt)
    plt.matshow(conf_mt)
    plt.show()
    print('\nClassification Report')
    target_names = ["Jackson", "Nicola", "Theo", "Ankur", "Caroline", "Rodolf
o", "Unknown"]
    print(classification_report(y_test_num, Y_pred))
```

In [ ]:

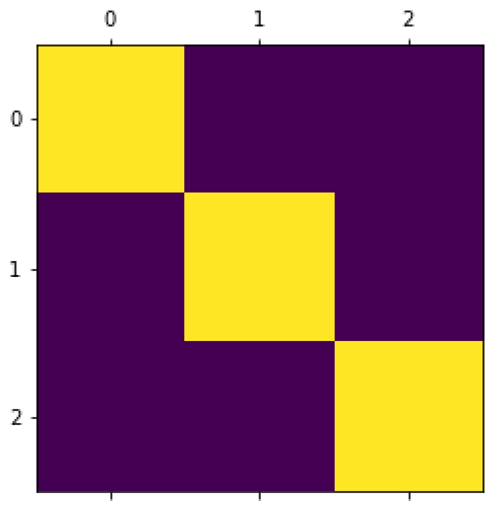
Present the model performance

In [11]:
```
print('\n# TEST DATA #\n')
score = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

# Prediction
printPrediction(X_test[0:10], y_test[0:10], False)
```

```
# TEST DATA #

30/30 [==============================] - 0s 33us/step
accuracy: 100.00%

# Generate predictions
y=Jackson    - prediction=Jackson    - match=True
y=Nicola     - prediction=Nicola     - match=True
y=Theo       - prediction=Theo       - match=True
y=Jackson    - prediction=Jackson    - match=True
y=Nicola     - prediction=Nicola     - match=True
y=Theo       - prediction=Theo       - match=True
y=Jackson    - prediction=Jackson    - match=True
y=Nicola     - prediction=Nicola     - match=True
y=Theo       - prediction=Theo       - match=True
y=Jackson    - prediction=Jackson    - match=True
```

In [12]:
```
print("Classification Report for Test Data\n")
report(X_test, y_test)
```

```
Classification Report for Test Data

[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```




```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

In [ ]:

Section 3

There are 50 recordings for each digit for each speaker: Jackson, Nicolas and Theo (total 1500 recordings) Training data has 49 recordings for each digit for each speaker: 1470 recordings total. Test data has 1 recordings for each digit for each speaker: 30 recordings total.

In addition, there are 2 recordings for each digit for each speaker: Ankur, Caroline and Rodolfo (total 60 recordings) This addition training data has 1 recordings for each digit for each speaker: 30 recordings total. This addition test data has 1 recordings for each digit for each speaker: 30 recordings total.

Therefore the full data set has:

- Training: 1500 recordings
- Training: 60 recordings

The data used here comes from the recordings stored in:

- ../data/recordings/train
- ../data/recordings/test
- ../data/recordings/moreSpeakersTrain
- ../data/recordings/moreSpeakersTest

In [ ]:

In [13]:
```
# Splitting the dataset into training, validation and testing dataset
from sklearn.model_selection import train_test_split

fullTrainData = trainData.append(moreTrainData)

X = np.array(fullTrainData.iloc[:, :-1], dtype = float)
y = fullTrainData.iloc[:, -1]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_
state=42)

X_test = np.array(testData.iloc[:, :-1], dtype = float)
y_test = testData.iloc[:, -1]

X_more_test = np.array(moreTestData.iloc[:, :-1], dtype = float)
y_more_test = moreTestData.iloc[:, -1]

print("Y from training data:", y_train.shape)
print("Y from validation data:", y_val.shape)
print("Y from test data:", y_test.shape)
print("Y from other speakers test data:", y_more_test.shape)
```

```
Y from training data: (1050,)
Y from validation data: (450,)
Y from test data: (30,)
Y from other speakers test data: (30,)
```

In [14]:
```
#Normalizing the dataset
from sklearn.preprocessing import StandardScaler
import numpy as np
scaler = StandardScaler()
X_train = scaler.fit_transform( X_train )
X_val = scaler.transform( X_val )
X_test = scaler.transform( X_test )
X_more_test = scaler.transform( X_more_test )

print("X from training data", X_train.shape)
print("X from validation data", X_val.shape)
print("X from test data", X_test.shape)
print("X from other speakers test data", X_more_test.shape)
```

```
X from training data (1050, 25)
X from validation data (450, 25)
X from test data (30, 25)
X from other speakers test data (30, 25)
```

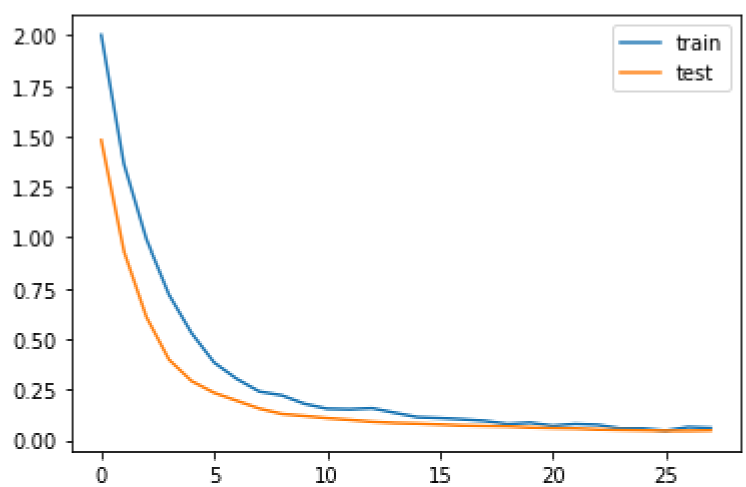
In [15]:
```
#Creating a Model
from keras import models
from keras import layers
import keras

# model 1
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_shape=(X_train.shape[1
],)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

# Learning Process of a model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# simple early stopping
from keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)

#Train with early stopping to avoid overfitting
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=50,
                    batch_size=128,
                    callbacks=[es])
```

```
Train on 1050 samples, validate on 450 samples
Epoch 1/50
1050/1050 [==============================] - 0s 176us/step - loss: 2.0021 - a
ccuracy: 0.2886 - val_loss: 1.4814 - val_accuracy: 0.8289
Epoch 2/50
1050/1050 [==============================] - 0s 30us/step - loss: 1.3657 - ac
curacy: 0.5495 - val_loss: 0.9358 - val_accuracy: 0.9111
Epoch 3/50
1050/1050 [==============================] - 0s 30us/step - loss: 0.9921 - ac
curacy: 0.6943 - val_loss: 0.6088 - val_accuracy: 0.9378
Epoch 4/50
1050/1050 [==============================] - 0s 31us/step - loss: 0.7191 - ac
curacy: 0.8000 - val_loss: 0.3982 - val_accuracy: 0.9400
Epoch 5/50
1050/1050 [==============================] - 0s 36us/step - loss: 0.5295 - ac
curacy: 0.8667 - val_loss: 0.2920 - val_accuracy: 0.9489
Epoch 6/50
1050/1050 [==============================] - 0s 32us/step - loss: 0.3837 - ac
curacy: 0.9105 - val_loss: 0.2339 - val_accuracy: 0.9533
Epoch 7/50
1050/1050 [==============================] - 0s 34us/step - loss: 0.3030 - ac
curacy: 0.9229 - val_loss: 0.1947 - val_accuracy: 0.9556
Epoch 8/50
1050/1050 [==============================] - 0s 34us/step - loss: 0.2402 - ac
curacy: 0.9495 - val_loss: 0.1559 - val_accuracy: 0.9578
Epoch 9/50
1050/1050 [==============================] - 0s 36us/step - loss: 0.2211 - ac
curacy: 0.9524 - val_loss: 0.1291 - val_accuracy: 0.9622
Epoch 10/50
1050/1050 [==============================] - 0s 37us/step - loss: 0.1796 - ac
curacy: 0.9524 - val_loss: 0.1190 - val_accuracy: 0.9644
Epoch 11/50
1050/1050 [==============================] - 0s 36us/step - loss: 0.1548 - ac
curacy: 0.9629 - val_loss: 0.1083 - val_accuracy: 0.9644
Epoch 12/50
1050/1050 [==============================] - 0s 35us/step - loss: 0.1522 - ac
curacy: 0.9667 - val_loss: 0.0997 - val_accuracy: 0.9689
Epoch 13/50
1050/1050 [==============================] - 0s 36us/step - loss: 0.1574 - ac
curacy: 0.9590 - val_loss: 0.0904 - val_accuracy: 0.9733
Epoch 14/50
1050/1050 [==============================] - 0s 35us/step - loss: 0.1350 - ac
curacy: 0.9667 - val_loss: 0.0844 - val_accuracy: 0.9733
Epoch 15/50
1050/1050 [==============================] - 0s 37us/step - loss: 0.1133 - ac
curacy: 0.9733 - val_loss: 0.0812 - val_accuracy: 0.9733
Epoch 16/50
1050/1050 [==============================] - 0s 36us/step - loss: 0.1086 - ac
curacy: 0.9714 - val_loss: 0.0767 - val_accuracy: 0.9733
Epoch 17/50
1050/1050 [==============================] - 0s 34us/step - loss: 0.1026 - ac
curacy: 0.9762 - val_loss: 0.0718 - val_accuracy: 0.9756
Epoch 18/50
1050/1050 [==============================] - 0s 35us/step - loss: 0.0946 - ac
curacy: 0.9695 - val_loss: 0.0697 - val_accuracy: 0.9756
Epoch 19/50
1050/1050 [==============================] - 0s 37us/step - loss: 0.0793 - ac
```

```
curacy: 0.9705 - val_loss: 0.0678 - val_accuracy: 0.9756
Epoch 20/50
1050/1050 [==============================] - 0s 38us/step - loss: 0.0859 - ac
curacy: 0.9743 - val_loss: 0.0621 - val_accuracy: 0.9800
Epoch 21/50
1050/1050 [==============================] - 0s 33us/step - loss: 0.0705 - ac
curacy: 0.9752 - val_loss: 0.0591 - val_accuracy: 0.9800
Epoch 22/50
1050/1050 [==============================] - 0s 37us/step - loss: 0.0796 - ac
curacy: 0.9771 - val_loss: 0.0572 - val_accuracy: 0.9800
Epoch 23/50
1050/1050 [==============================] - 0s 35us/step - loss: 0.0750 - ac
curacy: 0.9733 - val_loss: 0.0522 - val_accuracy: 0.9822
Epoch 24/50
1050/1050 [==============================] - 0s 35us/step - loss: 0.0571 - ac
curacy: 0.9838 - val_loss: 0.0496 - val_accuracy: 0.9822
Epoch 25/50
1050/1050 [==============================] - 0s 38us/step - loss: 0.0567 - ac
curacy: 0.9800 - val_loss: 0.0478 - val_accuracy: 0.9844
Epoch 26/50
1050/1050 [==============================] - 0s 40us/step - loss: 0.0463 - ac
curacy: 0.9857 - val_loss: 0.0455 - val_accuracy: 0.9867
Epoch 27/50
1050/1050 [==============================] - 0s 40us/step - loss: 0.0643 - ac
curacy: 0.9800 - val_loss: 0.0455 - val_accuracy: 0.9844
Epoch 28/50
1050/1050 [==============================] - ETA: 0s - loss: 0.0536 - accurac
y: 0.98 - 0s 40us/step - loss: 0.0599 - accuracy: 0.9819 - val_loss: 0.0475 -
val_accuracy: 0.9867
Epoch 00028: early stopping
```

In [16]:

```python
# plot training history
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



Present the model performance

In [17]:
```
print('\n# TEST DATA #\n')
score = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

# Prediction
printPrediction(X_test[0:10], y_test[0:10], False)
```

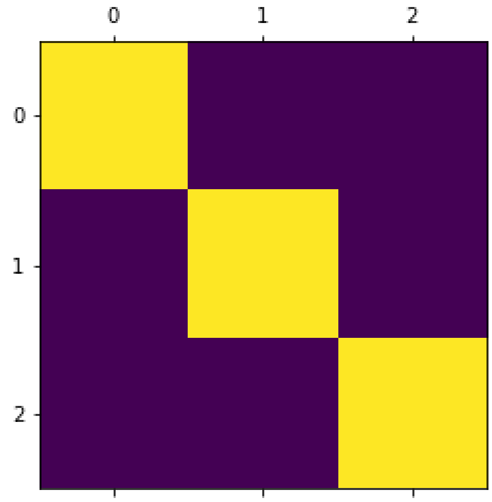
```
# TEST DATA #

30/30 [==============================] - 0s 33us/step
accuracy: 100.00%

# Generate predictions
y=Jackson   - prediction=Jackson   - match=True
y=Nicola    - prediction=Nicola    - match=True
y=Theo      - prediction=Theo      - match=True
y=Jackson   - prediction=Jackson   - match=True
y=Nicola    - prediction=Nicola    - match=True
y=Theo      - prediction=Theo      - match=True
y=Jackson   - prediction=Jackson   - match=True
y=Nicola    - prediction=Nicola    - match=True
y=Theo      - prediction=Theo      - match=True
y=Jackson   - prediction=Jackson   - match=True
```

In [18]:
```
print('\n# OTHER SPEAKERS DATA #\n')
score = model.evaluate(X_more_test, y_more_test)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

# Prediction
printPrediction(X_more_test[0:10], y_more_test[0:10], False)
```

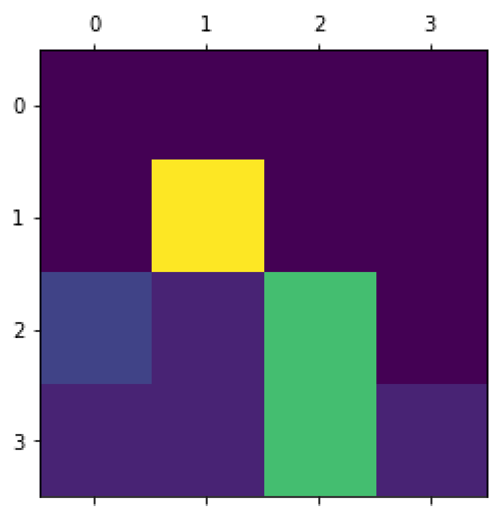
```
# OTHER SPEAKERS DATA #

30/30 [==============================] - 0s 33us/step
accuracy: 60.00%

# Generate predictions
y=Ankur     - prediction=Ankur     - match=True
y=Caroline  - prediction=Caroline  - match=True
y=Rodolfo   - prediction=Caroline  - match=False
y=Ankur     - prediction=Ankur     - match=True
y=Caroline  - prediction=Caroline  - match=True
y=Rodolfo   - prediction=Rodolfo   - match=True
y=Ankur     - prediction=Ankur     - match=True
y=Caroline  - prediction=Caroline  - match=True
y=Rodolfo   - prediction=Caroline  - match=False
y=Ankur     - prediction=Ankur     - match=True
```

In [19]:
```
print("Classification Report for Test Data\n")
report(X_test, y_test)

print("Classification Report for Other Speakers\n")
report(X_more_test, y_more_test)
```

```
Classification Report for Test Data

[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```




```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Classification Report for Other Speakers

[[ 0  0  0  0]
 [ 0 10  0  0]
 [ 2  1  7  0]
 [ 1  1  7  1]]
```



```
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           3       0.83      1.00      0.91        10
           4       0.50      0.70      0.58        10
           5       1.00      0.10      0.18        10

    accuracy                           0.60        30
   macro avg       0.58      0.45      0.42        30
weighted avg       0.78      0.60      0.56        30


c:\users\erodvas\env\lib\site-packages\sklearn\metrics\classification.py:143
9: UndefinedMetricWarning: Recall and F-score are ill-defined and being set t
o 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

In [ ]: