Name: Apoorva Asgekar , Udita Katrolia , Devika Pillai

Roll no.: C006 , C022 , C031

Program: BTech EXTC

Semester: VIII

Date of Performance: 14 / 03 / 22

Date of Submission: 21 / 02 / 22

Digit Recognition using Speech Processing

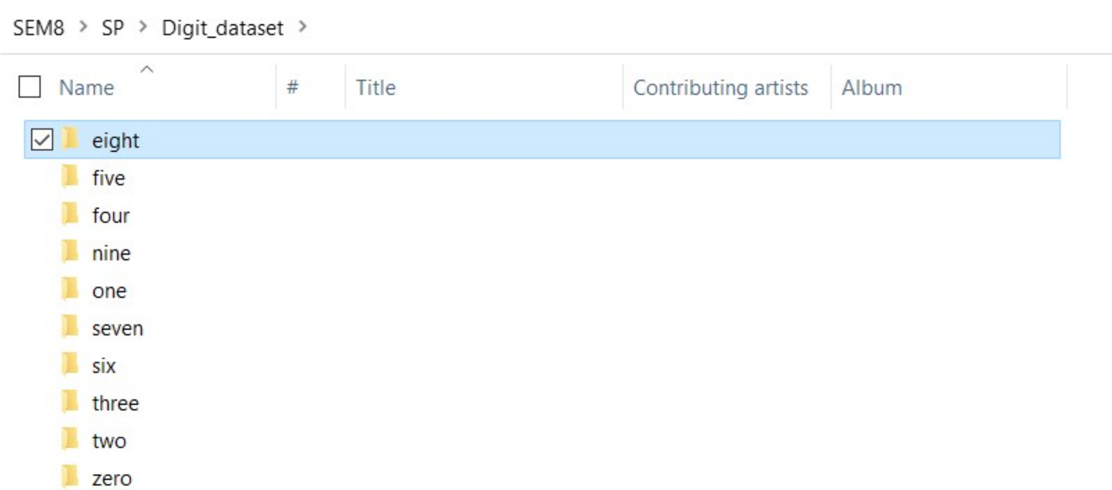
AIM: To use custom speech digit dataset to build a CNN model to recognize the digits

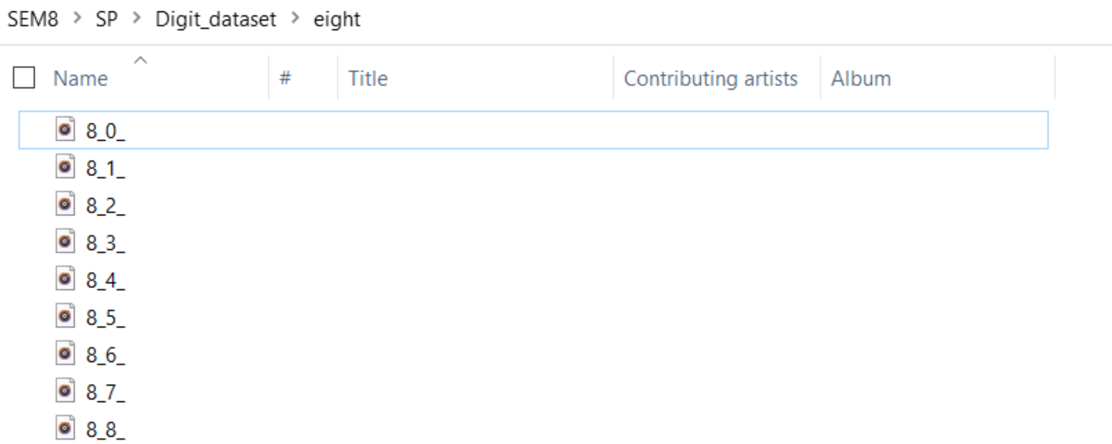
Step 1 Downdload/Extract dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

The Digit_dataset has 9 folders for digits between 0-9 each having 9 speech samples recorded by 3 individuals as shown below:





The dataset is uploaded as a zip file on google drive and is then unzipped as below:

```
#Unzipping the dataset
!unzip drive/MyDrive/Digit_dataset.zip
```

```
Archive:  drive/MyDrive/Digit_dataset.zip
   creating: Digit_dataset/
   creating: Digit_dataset/eight/
  inflating: Digit_dataset/eight/8_0_.wav
  inflating: Digit_dataset/eight/8_1_.wav
  inflating: Digit_dataset/eight/8_2_.wav
  inflating: Digit_dataset/eight/8_3_.wav
  inflating: Digit_dataset/eight/8_4_.wav
  inflating: Digit_dataset/eight/8_5_.wav
  inflating: Digit_dataset/eight/8_6_.wav
  inflating: Digit_dataset/eight/8_7_.wav
  inflating: Digit_dataset/eight/8_8_.wav
   creating: Digit_dataset/five/
  inflating: Digit_dataset/five/5_0_.wav
  inflating: Digit_dataset/five/5_1_.wav
  inflating: Digit_dataset/five/5_2_.wav
  inflating: Digit_dataset/five/5_3_.wav
  inflating: Digit_dataset/five/5_4_.wav
  inflating: Digit_dataset/five/5_5_.wav
  inflating: Digit_dataset/five/5_6_.wav
  inflating: Digit_dataset/five/5_7_.wav
  inflating: Digit_dataset/five/5_8_.wav
   creating: Digit_dataset/four/
  inflating: Digit_dataset/four/4_0_.wav
  inflating: Digit_dataset/four/4_1_.wav
  inflating: Digit_dataset/four/4_2_.wav
  inflating: Digit_dataset/four/4_3_.wav
  inflating: Digit_dataset/four/4_4_.wav
  inflating: Digit_dataset/four/4_5_.wav
  inflating: Digit_dataset/four/4_6_.wav
  inflating: Digit_dataset/four/4_7_.wav
```

```
  inflating: Digit_dataset/four/4_8_.wav
   creating: Digit_dataset/nine/
  inflating: Digit_dataset/nine/9_0_.wav
  inflating: Digit_dataset/nine/9_1_.wav
  inflating: Digit_dataset/nine/9_2_.wav
  inflating: Digit_dataset/nine/9_3_.wav
  inflating: Digit_dataset/nine/9_4_.wav
  inflating: Digit_dataset/nine/9_5_.wav
  inflating: Digit_dataset/nine/9_6_.wav
  inflating: Digit_dataset/nine/9_7_.wav
  inflating: Digit_dataset/nine/9_8_.wav
   creating: Digit_dataset/one/
  inflating: Digit_dataset/one/1_0_.wav
  inflating: Digit_dataset/one/1_1_.wav
  inflating: Digit_dataset/one/1_2_.wav
  inflating: Digit_dataset/one/1_3_.wav
  inflating: Digit_dataset/one/1_4_.wav
  inflating: Digit_dataset/one/1_5_.wav
  inflating: Digit_dataset/one/1_6_.wav
  inflating: Digit_dataset/one/1_7_.wav
  inflating: Digit_dataset/one/1_8_.wav
   creating: Digit_dataset/seven/
  inflating: Digit_dataset/seven/7_0_.wav
  inflating: Digit_dataset/seven/7_1_.wav
  inflating: Digit_dataset/seven/7_2_.wav
  inflating: Digit_dataset/seven/7_3_.wav
  inflating: Digit_dataset/seven/7_4_.wav
  inflating: Digit_dataset/seven/7_5_.wav
  inflating: Digit_dataset/seven/7_6_.wav
  inflating: Digit_dataset/seven/7_7_.wav
  inflating: Digit_dataset/seven/7_8_.wav
   creating: Digit_dataset/six/
  inflating: Digit_dataset/six/6_0_.wav
  inflating: Digit_dataset/six/6_1_.wav
  inflating: Digit_dataset/six/6_2_.wav
  inflating: Digit_dataset/six/6_3_.wav
  inflating: Digit_dataset/six/6_4_.wav
  inflating: Digit_dataset/six/6_5_.wav
  inflating: Digit_dataset/six/6_6_.wav
  inflating: Digit_dataset/six/6_7_.wav
  inflating: Digit_dataset/six/6_8_.wav
   creating: Digit_dataset/three/
  inflating: Digit_dataset/three/3_0_.wav
  inflating: Digit_dataset/three/3_1_.wav
  inflating: Digit_dataset/three/3_2_.wav
  inflating: Digit_dataset/three/3_3_.wav
  inflating: Digit_dataset/three/3_4_.wav
  inflating: Digit_dataset/three/3_5_.wav
  inflating: Digit_dataset/three/3_6_.wav
  inflating: Digit_dataset/three/3_7_.wav
```

```
  inflating: Digit_dataset/three/3_8_.wav
   creating: Digit_dataset/two/
  inflating: Digit_dataset/two/2_0_.wav
  inflating: Digit_dataset/two/2_1_.wav
  inflating: Digit_dataset/two/2_2_.wav
  inflating: Digit_dataset/two/2_3_.wav
  inflating: Digit_dataset/two/2_4_.wav
  inflating: Digit_dataset/two/2_5_.wav
  inflating: Digit_dataset/two/2_6_.wav
  inflating: Digit_dataset/two/2_7_.wav
  inflating: Digit_dataset/two/2_8_.wav
   creating: Digit_dataset/zero/
  inflating: Digit_dataset/zero/0_0_.wav
  inflating: Digit_dataset/zero/0_1_.wav
  inflating: Digit_dataset/zero/0_2_.wav
  inflating: Digit_dataset/zero/0_3_.wav
  inflating: Digit_dataset/zero/0_4_.wav
  inflating: Digit_dataset/zero/0_5_.wav
  inflating: Digit_dataset/zero/0_6_.wav
  inflating: Digit_dataset/zero/0_7_.wav
  inflating: Digit_dataset/zero/0_8_.wav
```

Step 2 Data Preprocessing

```
import librosa
import os
import json

DATASET_PATH = "/content/Digit_dataset"      #   enter the dataset
path here
#!touch data.json
JSON_PATH = "data.json"
SAMPLES_TO_CONSIDER = 22050 # 1 sec. of audio

# Defining a function to preprocess the dataset

def preprocess_dataset(dataset_path, json_path, num_mfcc=13,
n_fft=2048, hop_length=512):

    """Extracts MFCCs from music dataset and saves them into a json
file.
    :param dataset_path (str): Path to dataset
    :param json_path (str): Path to json file used to save MFCCs
    :param num_mfcc (int): Number of coefficients to extract
    :param n_fft (int): Interval we consider to apply FFT. Measured in
# of samples
    :param hop_length (int): Sliding window for FFT. Measured in # of
samples
    :return:
    """
```

```
    import numpy as np

    # dictionary where we'll store mapping, labels, MFCCs and
filenames
    data = {
        "mapping": [],
        "labels": [],
        "MFCCs": [],
        "files": []
    }

    # loop through all sub-dirs
    for i, (dirpath, dirnames, filenames) in
enumerate(os.walk(dataset_path)):

        # ensure we're at sub-folder level
        if dirpath is not dataset_path:

            # save label (i.e., sub-folder name) in the mapping
            label = dirpath.split("/")[-1]
            data["mapping"].append(label)
            print("\nProcessing: '{}'".format(label))

            # process all audio files in sub-dir and store MFCCs
            for f in filenames:
                file_path = os.path.join(dirpath, f)

                # load audio file and slice it to ensure length
consistency among different files
                signal, sample_rate = librosa.load(file_path)

                # drop audio files with less than pre-decided number
of samples
                if len(signal) >= SAMPLES_TO_CONSIDER:

                    # ensure consistency of the length of the signal
                    signal = signal[:SAMPLES_TO_CONSIDER]

                    # extract MFCCs
                    MFCCs =
librosa.feature.mfcc(signal,sample_rate,n_mfcc=num_mfcc,n_fft=n_fft,ho
p_length=hop_length)

                    # store data for analysed track

data["MFCCs"].append((np.transpose(MFCCs)).tolist())
                    data["labels"].append(i-1)
                    data["files"].append(file_path)
                    print("{}: {}".format(file_path, i-1))
```

```python
    # save data in json file
    with open(json_path, "w") as fp:
        json.dump(data, fp, indent=4)

#Mapping the wav file of one digit to a particular labbel between 0-9
if __name__ == "__main__":
    preprocess_dataset(DATASET_PATH, JSON_PATH)
```

```
Processing: 'five'
/content/Digit_dataset/five/5_5_.wav: 0
/content/Digit_dataset/five/5_6_.wav: 0
/content/Digit_dataset/five/5_4_.wav: 0
/content/Digit_dataset/five/5_8_.wav: 0
/content/Digit_dataset/five/5_0_.wav: 0
/content/Digit_dataset/five/5_2_.wav: 0
/content/Digit_dataset/five/5_1_.wav: 0
/content/Digit_dataset/five/5_3_.wav: 0
/content/Digit_dataset/five/5_7_.wav: 0

Processing: 'nine'
/content/Digit_dataset/nine/9_1_.wav: 1
/content/Digit_dataset/nine/9_3_.wav: 1
/content/Digit_dataset/nine/9_5_.wav: 1
/content/Digit_dataset/nine/9_4_.wav: 1
/content/Digit_dataset/nine/9_6_.wav: 1
/content/Digit_dataset/nine/9_8_.wav: 1
/content/Digit_dataset/nine/9_0_.wav: 1
/content/Digit_dataset/nine/9_2_.wav: 1
/content/Digit_dataset/nine/9_7_.wav: 1

Processing: 'one'
/content/Digit_dataset/one/1_4_.wav: 2
/content/Digit_dataset/one/1_0_.wav: 2
/content/Digit_dataset/one/1_7_.wav: 2
/content/Digit_dataset/one/1_3_.wav: 2
/content/Digit_dataset/one/1_2_.wav: 2
/content/Digit_dataset/one/1_6_.wav: 2
/content/Digit_dataset/one/1_5_.wav: 2
/content/Digit_dataset/one/1_1_.wav: 2
/content/Digit_dataset/one/1_8_.wav: 2

Processing: 'six'
/content/Digit_dataset/six/6_0_.wav: 3
/content/Digit_dataset/six/6_8_.wav: 3
/content/Digit_dataset/six/6_1_.wav: 3
/content/Digit_dataset/six/6_2_.wav: 3
/content/Digit_dataset/six/6_6_.wav: 3
/content/Digit_dataset/six/6_3_.wav: 3
```

```
/content/Digit_dataset/six/6_5_.wav: 3
/content/Digit_dataset/six/6_4_.wav: 3
/content/Digit_dataset/six/6_7_.wav: 3

Processing: 'two'
/content/Digit_dataset/two/2_6_.wav: 4
/content/Digit_dataset/two/2_1_.wav: 4
/content/Digit_dataset/two/2_7_.wav: 4
/content/Digit_dataset/two/2_5_.wav: 4
/content/Digit_dataset/two/2_8_.wav: 4
/content/Digit_dataset/two/2_3_.wav: 4
/content/Digit_dataset/two/2_4_.wav: 4
/content/Digit_dataset/two/2_2_.wav: 4
/content/Digit_dataset/two/2_0_.wav: 4

Processing: 'three'
/content/Digit_dataset/three/3_6_.wav: 5
/content/Digit_dataset/three/3_0_.wav: 5
/content/Digit_dataset/three/3_1_.wav: 5
/content/Digit_dataset/three/3_4_.wav: 5
/content/Digit_dataset/three/3_2_.wav: 5
/content/Digit_dataset/three/3_5_.wav: 5
/content/Digit_dataset/three/3_8_.wav: 5
/content/Digit_dataset/three/3_7_.wav: 5
/content/Digit_dataset/three/3_3_.wav: 5

Processing: 'four'
/content/Digit_dataset/four/4_0_.wav: 6
/content/Digit_dataset/four/4_1_.wav: 6
/content/Digit_dataset/four/4_6_.wav: 6
/content/Digit_dataset/four/4_5_.wav: 6
/content/Digit_dataset/four/4_3_.wav: 6
/content/Digit_dataset/four/4_7_.wav: 6
/content/Digit_dataset/four/4_8_.wav: 6
/content/Digit_dataset/four/4_4_.wav: 6
/content/Digit_dataset/four/4_2_.wav: 6

Processing: 'zero'
/content/Digit_dataset/zero/0_1_.wav: 7
/content/Digit_dataset/zero/0_3_.wav: 7
/content/Digit_dataset/zero/0_0_.wav: 7
/content/Digit_dataset/zero/0_8_.wav: 7
/content/Digit_dataset/zero/0_5_.wav: 7
/content/Digit_dataset/zero/0_6_.wav: 7
/content/Digit_dataset/zero/0_2_.wav: 7
/content/Digit_dataset/zero/0_7_.wav: 7
/content/Digit_dataset/zero/0_4_.wav: 7

Processing: 'seven'
/content/Digit_dataset/seven/7_8_.wav: 8
```

```
/content/Digit_dataset/seven/7_0_.wav: 8
/content/Digit_dataset/seven/7_1_.wav: 8
/content/Digit_dataset/seven/7_6_.wav: 8
/content/Digit_dataset/seven/7_3_.wav: 8
/content/Digit_dataset/seven/7_5_.wav: 8
/content/Digit_dataset/seven/7_2_.wav: 8
/content/Digit_dataset/seven/7_4_.wav: 8
/content/Digit_dataset/seven/7_7_.wav: 8

Processing: 'eight'
/content/Digit_dataset/eight/8_8_.wav: 9
/content/Digit_dataset/eight/8_7_.wav: 9
/content/Digit_dataset/eight/8_0_.wav: 9
/content/Digit_dataset/eight/8_1_.wav: 9
/content/Digit_dataset/eight/8_5_.wav: 9
/content/Digit_dataset/eight/8_4_.wav: 9
/content/Digit_dataset/eight/8_3_.wav: 9
/content/Digit_dataset/eight/8_6_.wav: 9
/content/Digit_dataset/eight/8_2_.wav: 9
```

```
#Displaying the labels that we have mapped

import json

with open("data.json") as jsonFile:
    jsonObject = json.load(jsonFile)
    jsonFile.close()

mapping  = jsonObject['mapping']


print(mapping)
```

```
['five', 'nine', 'one', 'six', 'two', 'three', 'four', 'zero',
'seven', 'eight']
```

Step 3 Model building

```
import json
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

DATA_PATH = "data.json"
SAVED_MODEL_PATH = "model.h5"
EPOCHS = 100 #40
BATCH_SIZE = 32
```

```
PATIENCE = 5
LEARNING_RATE = 0.001 #0.0001


# Defining a function to load the dataset with labels

def load_data(data_path):
    """Loads training dataset from json file.
    :param data_path (str): Path to json file containing data
    :return X (ndarray): Inputs
    :return y (ndarray): Targets
    """
    with open(data_path, "r") as fp:
        data = json.load(fp)

    X = np.array(data["MFCCs"])
    y = np.array(data["labels"])
    print("Training sets loaded!")
    return X, y
```

Spliting data to train and test

```
# Defining a function to prepare the dataset for training

def prepare_dataset(data_path, test_size=0.2, validation_size=0.2):
    """Creates train, validation and test sets.
    :param data_path (str): Path to json file containing data
    :param test_size (flaot): Percentage of dataset used for testing
    :param validation_size (float): Percentage of train set used for
cross-validation
    :return X_train (ndarray): Inputs for the train set
    :return y_train (ndarray): Targets for the train set
    :return X_validation (ndarray): Inputs for the validation set
    :return y_validation (ndarray): Targets for the validation set
    :return X_test (ndarray): Inputs for the test set
    :return X_test (ndarray): Targets for the test set
    """

    # load dataset
    X, y = load_data(data_path)

    # create train, validation, test split
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)
    X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=validation_size)

    # add an axis to nd array
    X_train = X_train[..., np.newaxis]
```

```
    X_test = X_test[..., np.newaxis]
    X_validation = X_validation[..., np.newaxis]

    return X_train, y_train, X_validation, y_validation, X_test,
y_test
```

Defining the 2dCNN model

```
# Defining a function to build teh CNN model using the split dataset
# The model have 3 combinations of Convolution, Normalization and
Pooling layers
# Dense and dropout layers are added
# Last layer has softmax activation

def build_model(input_shape, loss="sparse_categorical_crossentropy",
learning_rate=0.01):
    """Build neural network using keras.
    :param input_shape (tuple): Shape of array representing a sample
train. E.g.: (44, 13, 1)
    :param loss (str): Loss function to use
    :param learning_rate (float):
    :return model: TensorFlow model
    """

    # build network architecture using convolutional layers
    model = tf.keras.models.Sequential()

    # 1st conv layer
    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
input_shape=input_shape,

kernel_regularizer=tf.keras.regularizers.l2(0.001)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D((3, 3), strides=(2,2),
padding='same'))

    # 2nd conv layer
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu',

kernel_regularizer=tf.keras.regularizers.l2(0.001)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D((3, 3), strides=(2,2),
padding='same'))

    # 3rd conv layer
    model.add(tf.keras.layers.Conv2D(32, (2, 2), activation='relu',

kernel_regularizer=tf.keras.regularizers.l2(0.001)))
    model.add(tf.keras.layers.BatchNormalization())
```

```
    model.add(tf.keras.layers.MaxPooling2D((2, 2), strides=(2,2),
padding='same'))

    # flatten output and feed into dense layer
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    tf.keras.layers.Dropout(0.3)

    # softmax output layer
    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    optimiser = tf.optimizers.Adam(learning_rate=learning_rate)

    # compile model
    model.compile(optimizer=optimiser,
                  loss=loss,
                  metrics=["accuracy"])

    # print model parameters on console
    model.summary()

    return model

# Defining a function to train the model

def train(model, epochs, batch_size, patience, X_train, y_train,
X_validation, y_validation):
    """Trains model
    :param epochs (int): Num training epochs
    :param batch_size (int): Samples per batch
    :param patience (int): Num epochs to wait before early stop, if
there isn't an improvement on accuracy
    :param X_train (ndarray): Inputs for the train set
    :param y_train (ndarray): Targets for the train set
    :param X_validation (ndarray): Inputs for the validation set
    :param y_validation (ndarray): Targets for the validation set
    :return history: Training history
    """

    earlystop_callback =
tf.keras.callbacks.EarlyStopping(monitor="accuracy", min_delta=0.001,
patience=patience)

    # train model
    history = model.fit(X_train,
                        y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(X_validation, y_validation),
```

```
                        callbacks=[earlystop_callback])
    return history

# Defining a function to plot the various accuracies and losses

def plot_history(history):
    """Plots accuracy/loss for training/validation set as a function
of the epochs
    :param history: Training history of model
    :return:
    """

    fig, axs = plt.subplots(2)

    # create accuracy subplot
    axs[0].plot(history.history["accuracy"], label="accuracy")
    axs[0].plot(history.history['val_accuracy'], label="val_accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy evaluation")

    # create loss subplot
    axs[1].plot(history.history["loss"], label="loss")
    axs[1].plot(history.history['val_loss'], label="val_loss")
    axs[1].set_xlabel("Epoch")
    axs[1].set_ylabel("Loss")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Loss evaluation")

    plt.show()

def main():
    # generate train, validation and test sets
    X_train, y_train, X_validation, y_validation, X_test, y_test =
prepare_dataset(DATA_PATH)

    # create network
    input_shape = (X_train.shape[1], X_train.shape[2], 1)
    model = build_model(input_shape, learning_rate=LEARNING_RATE)

    # train network
    history = train(model, EPOCHS, BATCH_SIZE, PATIENCE, X_train,
y_train, X_validation, y_validation)

    # plot accuracy/loss for training/validation set as a function of
the epochs
    plot_history(history)

    # evaluate network on test set
    test_loss, test_acc = model.evaluate(X_test, y_test)
```

```
    print("\nTest loss: {}, test accuracy: {}".format(test_loss,
100*test_acc))

    # save model
    model.save(SAVED_MODEL_PATH)
```

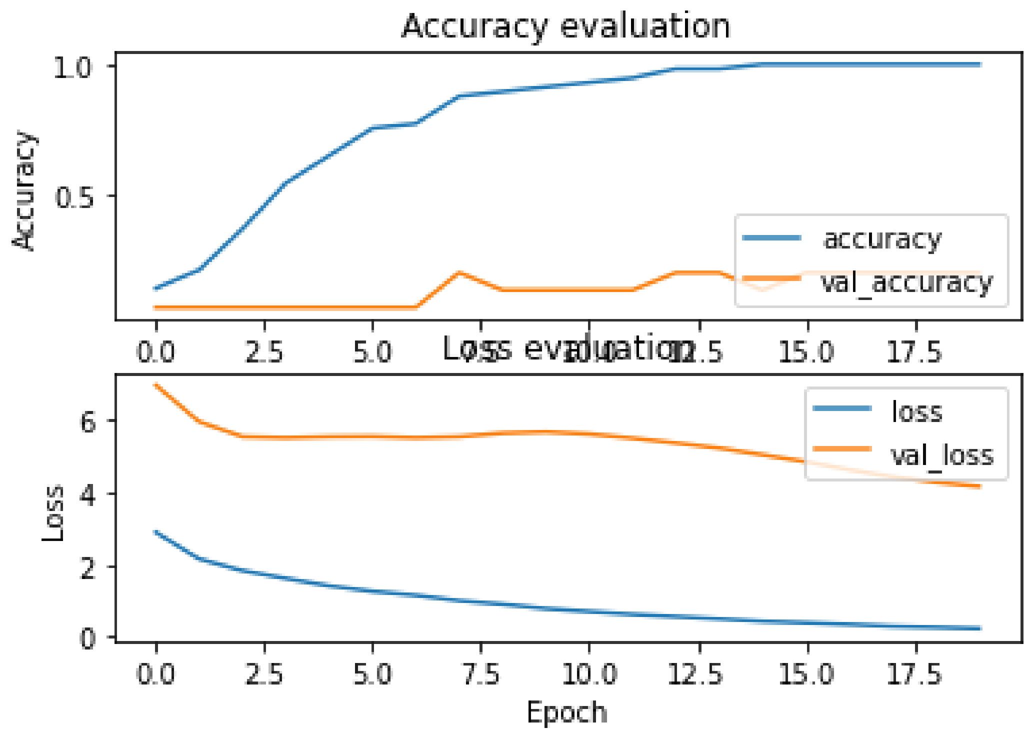
Step 4 Model training

```
if __name__ == "__main__":
  main()
```

```
Training sets loaded!
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 42, 11, 64)        640

 batch_normalization (BatchN  (None, 42, 11, 64)       256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 21, 6, 64)        0
 )

 conv2d_1 (Conv2D)           (None, 19, 4, 32)         18464

 batch_normalization_1 (Batc  (None, 19, 4, 32)        128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 10, 2, 32)        0
 2D)

 conv2d_2 (Conv2D)           (None, 9, 1, 32)          4128

 batch_normalization_2 (Batc  (None, 9, 1, 32)         128
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 5, 1, 32)         0
 2D)

 flatten (Flatten)           (None, 160)               0

 dense (Dense)               (None, 64)                10304

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 34,698
Trainable params: 34,442
Non-trainable params: 256
```

```
Epoch 1/100
2/2 [==============================] - 10s 445ms/step - loss: 2.9001 -
accuracy: 0.1404 - val_loss: 6.9733 - val_accuracy: 0.0667
Epoch 2/100
2/2 [==============================] - 0s 33ms/step - loss: 2.1594 -
accuracy: 0.2105 - val_loss: 5.9627 - val_accuracy: 0.0667
Epoch 3/100
2/2 [==============================] - 0s 32ms/step - loss: 1.8392 -
accuracy: 0.3684 - val_loss: 5.5471 - val_accuracy: 0.0667
Epoch 4/100
2/2 [==============================] - 0s 36ms/step - loss: 1.6227 -
accuracy: 0.5439 - val_loss: 5.5216 - val_accuracy: 0.0667
Epoch 5/100
2/2 [==============================] - 0s 34ms/step - loss: 1.4173 -
accuracy: 0.6491 - val_loss: 5.5466 - val_accuracy: 0.0667
Epoch 6/100
2/2 [==============================] - 0s 45ms/step - loss: 1.2637 -
accuracy: 0.7544 - val_loss: 5.5538 - val_accuracy: 0.0667
Epoch 7/100
2/2 [==============================] - 0s 38ms/step - loss: 1.1563 -
accuracy: 0.7719 - val_loss: 5.5213 - val_accuracy: 0.0667
Epoch 8/100
2/2 [==============================] - 0s 45ms/step - loss: 1.0048 -
accuracy: 0.8772 - val_loss: 5.5472 - val_accuracy: 0.2000
Epoch 9/100
2/2 [==============================] - 0s 34ms/step - loss: 0.9054 -
accuracy: 0.8947 - val_loss: 5.6462 - val_accuracy: 0.1333
Epoch 10/100
2/2 [==============================] - 0s 33ms/step - loss: 0.7865 -
accuracy: 0.9123 - val_loss: 5.6814 - val_accuracy: 0.1333
Epoch 11/100
2/2 [==============================] - 0s 41ms/step - loss: 0.7052 -
accuracy: 0.9298 - val_loss: 5.6203 - val_accuracy: 0.1333
Epoch 12/100
2/2 [==============================] - 0s 33ms/step - loss: 0.6291 -
accuracy: 0.9474 - val_loss: 5.5079 - val_accuracy: 0.1333
Epoch 13/100
2/2 [==============================] - 0s 36ms/step - loss: 0.5648 -
accuracy: 0.9825 - val_loss: 5.3729 - val_accuracy: 0.2000
Epoch 14/100
2/2 [==============================] - 0s 34ms/step - loss: 0.5041 -
accuracy: 0.9825 - val_loss: 5.2306 - val_accuracy: 0.2000
Epoch 15/100
2/2 [==============================] - 0s 35ms/step - loss: 0.4355 -
accuracy: 1.0000 - val_loss: 5.0478 - val_accuracy: 0.1333
Epoch 16/100
2/2 [==============================] - 0s 33ms/step - loss: 0.3939 -
accuracy: 1.0000 - val_loss: 4.8558 - val_accuracy: 0.2000
Epoch 17/100
```

```
2/2 [==============================] - 0s 34ms/step - loss: 0.3477 -
accuracy: 1.0000 - val_loss: 4.6325 - val_accuracy: 0.2000
Epoch 18/100
2/2 [==============================] - 0s 34ms/step - loss: 0.2960 -
accuracy: 1.0000 - val_loss: 4.4386 - val_accuracy: 0.2000
Epoch 19/100
2/2 [==============================] - 0s 32ms/step - loss: 0.2702 -
accuracy: 1.0000 - val_loss: 4.2855 - val_accuracy: 0.2000
Epoch 20/100
2/2 [==============================] - 0s 36ms/step - loss: 0.2370 -
accuracy: 1.0000 - val_loss: 4.1729 - val_accuracy: 0.2000
```




```
1/1 [==============================] - 0s 66ms/step - loss: 4.1427 -
accuracy: 0.2778

Test loss: 4.142741680145264, test accuracy: 27.77777910232544
```

Conclusion:

1. We have built a 2D-CNN model that is trained using samples of digits speech files.
2. We have got the training accuracy with 100 epochs to be ranging between 98% to a perfect 100% and the validation accuracy is 20%.
3. We have calculated the testing accuracy to be 27%.
4. We have also plotted the traing and validation accuracy as well as the training and validation loss for visual observation.