SPEECH PROCESSING MINI PROJECT

B.TECH EXTC SEM 8

NAME:

Jay Goyal C017

Abhinav Mishra C028

Mehul Lakra C051

Aditya Maheshwari C054

Step 1 Downdload/Extract dataset

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

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

Step 2 Data Preprocessing

```
import librosa
import os
import json

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

def preprocess_dataset(dataset_path, json_path, num_mfcc=13,
n_fft=2048, hop_length=512):
    import numpy as np
    """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:
    """
```

```
    # 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,hop_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))

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

```
if __name__ == "__main__":
    preprocess_dataset(DATASET_PATH, JSON_PATH)
```

```
Processing: 'T'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/T/T_0_.wav: 0
/content/drive/MyDrive/ASR/Project_Alphabet_Data/T/T_1_.wav: 0
/content/drive/MyDrive/ASR/Project_Alphabet_Data/T/T_2_.wav: 0
/content/drive/MyDrive/ASR/Project_Alphabet_Data/T/T_3_.wav: 0
/content/drive/MyDrive/ASR/Project_Alphabet_Data/T/T_4_.wav: 0

Processing: 'Q'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Q/Q_0_.wav: 1
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Q/Q_1_.wav: 1
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Q/Q_2_.wav: 1
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Q/Q_3_.wav: 1
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Q/Q_4_.wav: 1

Processing: 'S'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/S/S_0_.wav: 2
/content/drive/MyDrive/ASR/Project_Alphabet_Data/S/S_1_.wav: 2
/content/drive/MyDrive/ASR/Project_Alphabet_Data/S/S_2_.wav: 2
/content/drive/MyDrive/ASR/Project_Alphabet_Data/S/S_3_.wav: 2
/content/drive/MyDrive/ASR/Project_Alphabet_Data/S/S_4_.wav: 2

Processing: 'R'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/R/R_0_.wav: 3
/content/drive/MyDrive/ASR/Project_Alphabet_Data/R/R_1_.wav: 3
/content/drive/MyDrive/ASR/Project_Alphabet_Data/R/R_2_.wav: 3
/content/drive/MyDrive/ASR/Project_Alphabet_Data/R/R_3_.wav: 3
/content/drive/MyDrive/ASR/Project_Alphabet_Data/R/R_4_.wav: 3

Processing: 'Z'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Z/Z_0_.wav: 4
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Z/Z_1_.wav: 4
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Z/Z_2_.wav: 4
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Z/Z_3_.wav: 4
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Z/Z_4_.wav: 4

Processing: 'U'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/U/U_0_.wav: 5
/content/drive/MyDrive/ASR/Project_Alphabet_Data/U/U_1_.wav: 5
/content/drive/MyDrive/ASR/Project_Alphabet_Data/U/U_2_.wav: 5
/content/drive/MyDrive/ASR/Project_Alphabet_Data/U/U_3_ .wav: 5
/content/drive/MyDrive/ASR/Project_Alphabet_Data/U/U_4_.wav: 5

Processing: 'X'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/X/X_0_.wav: 6
/content/drive/MyDrive/ASR/Project_Alphabet_Data/X/X_1_.wav: 6
/content/drive/MyDrive/ASR/Project_Alphabet_Data/X/X_2_.wav: 6
```

```
/content/drive/MyDrive/ASR/Project_Alphabet_Data/X/X_3_.wav: 6
/content/drive/MyDrive/ASR/Project_Alphabet_Data/X/X_4_.wav: 6

Processing: 'Y'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Y/Y_0_.wav: 7
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Y/Y_1_.wav: 7
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Y/Y_2_.wav: 7
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Y/Y_3_.wav: 7
/content/drive/MyDrive/ASR/Project_Alphabet_Data/Y/Y_4_.wav: 7

Processing: 'W'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/W/W_0_.wav: 8
/content/drive/MyDrive/ASR/Project_Alphabet_Data/W/W_1_.wav: 8
/content/drive/MyDrive/ASR/Project_Alphabet_Data/W/W_2_.wav: 8
/content/drive/MyDrive/ASR/Project_Alphabet_Data/W/W_3_.wav: 8
/content/drive/MyDrive/ASR/Project_Alphabet_Data/W/W_4_.wav: 8

Processing: 'V'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/V/V_0_.wav: 9
/content/drive/MyDrive/ASR/Project_Alphabet_Data/V/V_1_.wav: 9
/content/drive/MyDrive/ASR/Project_Alphabet_Data/V/V_2_.wav: 9
/content/drive/MyDrive/ASR/Project_Alphabet_Data/V/V_3_.wav: 9
/content/drive/MyDrive/ASR/Project_Alphabet_Data/V/V_4_.wav: 9

Processing: 'J'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/J/J_0_.wav: 10
/content/drive/MyDrive/ASR/Project_Alphabet_Data/J/J_1_.wav: 10
/content/drive/MyDrive/ASR/Project_Alphabet_Data/J/J_2_.wav: 10
/content/drive/MyDrive/ASR/Project_Alphabet_Data/J/J_3_.wav: 10
/content/drive/MyDrive/ASR/Project_Alphabet_Data/J/J_4_.wav: 10

Processing: 'H'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/H/H_0_.wav: 11
/content/drive/MyDrive/ASR/Project_Alphabet_Data/H/H_1_.wav: 11
/content/drive/MyDrive/ASR/Project_Alphabet_Data/H/H_2_.wav: 11
/content/drive/MyDrive/ASR/Project_Alphabet_Data/H/H_3_.wav: 11
/content/drive/MyDrive/ASR/Project_Alphabet_Data/H/H_4_.wav: 11

Processing: 'O'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/O/O_0_.wav: 12
/content/drive/MyDrive/ASR/Project_Alphabet_Data/O/O_1_.wav: 12
/content/drive/MyDrive/ASR/Project_Alphabet_Data/O/O_2_.wav: 12
/content/drive/MyDrive/ASR/Project_Alphabet_Data/O/O_3_.wav: 12
/content/drive/MyDrive/ASR/Project_Alphabet_Data/O/O_4_.wav: 12

Processing: 'P'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/P/P_0_.wav: 13
/content/drive/MyDrive/ASR/Project_Alphabet_Data/P/P_1_.wav: 13
/content/drive/MyDrive/ASR/Project_Alphabet_Data/P/P_2_.wav: 13
```

```
/content/drive/MyDrive/ASR/Project_Alphabet_Data/P/P_3_.wav: 13
/content/drive/MyDrive/ASR/Project_Alphabet_Data/P/P_4_.wav: 13

Processing: 'K'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/K/K_0_.wav: 14
/content/drive/MyDrive/ASR/Project_Alphabet_Data/K/K_1_.wav: 14
/content/drive/MyDrive/ASR/Project_Alphabet_Data/K/K_2_.wav: 14
/content/drive/MyDrive/ASR/Project_Alphabet_Data/K/K_3_.wav: 14
/content/drive/MyDrive/ASR/Project_Alphabet_Data/K/K_4_.wav: 14

Processing: 'M'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/M/M_0_.wav: 15
/content/drive/MyDrive/ASR/Project_Alphabet_Data/M/M_1_.wav: 15
/content/drive/MyDrive/ASR/Project_Alphabet_Data/M/M_2_.wav: 15
/content/drive/MyDrive/ASR/Project_Alphabet_Data/M/M_3_.wav: 15
/content/drive/MyDrive/ASR/Project_Alphabet_Data/M/M_4_.wav: 15

Processing: 'I'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/I/I_0_.wav: 16
/content/drive/MyDrive/ASR/Project_Alphabet_Data/I/I_1_.wav: 16
/content/drive/MyDrive/ASR/Project_Alphabet_Data/I/I_2_.wav: 16
/content/drive/MyDrive/ASR/Project_Alphabet_Data/I/I_3_.wav: 16
/content/drive/MyDrive/ASR/Project_Alphabet_Data/I/I_4_.wav: 16

Processing: 'G'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/G/G_0_.wav: 17
/content/drive/MyDrive/ASR/Project_Alphabet_Data/G/G_1_.wav: 17
/content/drive/MyDrive/ASR/Project_Alphabet_Data/G/G_2_.wav: 17
/content/drive/MyDrive/ASR/Project_Alphabet_Data/G/G_3_.wav: 17
/content/drive/MyDrive/ASR/Project_Alphabet_Data/G/G_4_.wav: 17

Processing: 'L'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/L/L_0_.wav: 18
/content/drive/MyDrive/ASR/Project_Alphabet_Data/L/L_1_.wav: 18
/content/drive/MyDrive/ASR/Project_Alphabet_Data/L/L_2_.wav: 18
/content/drive/MyDrive/ASR/Project_Alphabet_Data/L/L_3_.wav: 18
/content/drive/MyDrive/ASR/Project_Alphabet_Data/L/L_4_.wav: 18

Processing: 'N'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/N/N_0_.wav: 19
/content/drive/MyDrive/ASR/Project_Alphabet_Data/N/N_1_.wav: 19
/content/drive/MyDrive/ASR/Project_Alphabet_Data/N/N_2_.wav: 19
/content/drive/MyDrive/ASR/Project_Alphabet_Data/N/N_3_.wav: 19
/content/drive/MyDrive/ASR/Project_Alphabet_Data/N/N_4_.wav: 19

Processing: 'F'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/F/F_0_.wav: 20
/content/drive/MyDrive/ASR/Project_Alphabet_Data/F/F_1_.wav: 20
/content/drive/MyDrive/ASR/Project_Alphabet_Data/F/F_2_.wav: 20
```

```
/content/drive/MyDrive/ASR/Project_Alphabet_Data/F/F_3_.wav: 20
/content/drive/MyDrive/ASR/Project_Alphabet_Data/F/F_4_.wav: 20

Processing: 'D'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/D/D_0_.wav: 21
/content/drive/MyDrive/ASR/Project_Alphabet_Data/D/D_1_.wav: 21
/content/drive/MyDrive/ASR/Project_Alphabet_Data/D/D_2_.wav: 21
/content/drive/MyDrive/ASR/Project_Alphabet_Data/D/D_3_.wav: 21
/content/drive/MyDrive/ASR/Project_Alphabet_Data/D/D_4_.wav: 21

Processing: 'A'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/A/A_0_.wav: 22
/content/drive/MyDrive/ASR/Project_Alphabet_Data/A/A_1_.wav: 22
/content/drive/MyDrive/ASR/Project_Alphabet_Data/A/A_2_.wav: 22
/content/drive/MyDrive/ASR/Project_Alphabet_Data/A/A_3_.wav: 22
/content/drive/MyDrive/ASR/Project_Alphabet_Data/A/A_4_.wav: 22

Processing: 'C'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/C/C_0_.wav: 23
/content/drive/MyDrive/ASR/Project_Alphabet_Data/C/C_1_.wav: 23
/content/drive/MyDrive/ASR/Project_Alphabet_Data/C/C_2_.wav: 23
/content/drive/MyDrive/ASR/Project_Alphabet_Data/C/C_3_.wav: 23
/content/drive/MyDrive/ASR/Project_Alphabet_Data/C/C_4_.wav: 23

Processing: 'E'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/E/E_0_.wav: 24
/content/drive/MyDrive/ASR/Project_Alphabet_Data/E/E_1_.wav: 24
/content/drive/MyDrive/ASR/Project_Alphabet_Data/E/E_2_.wav: 24
/content/drive/MyDrive/ASR/Project_Alphabet_Data/E/E_3_.wav: 24
/content/drive/MyDrive/ASR/Project_Alphabet_Data/E/E_4_.wav: 24

Processing: 'B'
/content/drive/MyDrive/ASR/Project_Alphabet_Data/B/B_0_.wav: 25
/content/drive/MyDrive/ASR/Project_Alphabet_Data/B/B_1_.wav: 25
/content/drive/MyDrive/ASR/Project_Alphabet_Data/B/B_2_.wav: 25
/content/drive/MyDrive/ASR/Project_Alphabet_Data/B/B_3_.wav: 25
/content/drive/MyDrive/ASR/Project_Alphabet_Data/B/B_4_.wav: 25
```

```
import json


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

mapping  = jsonObject['mapping']


print(mapping)
```

```
['T', 'Q', 'S', 'R', 'Z', 'U', 'X', 'Y', 'W', 'V', 'J', 'H', 'O', 'P',
'K', 'M', 'I', 'G', 'L', 'N', 'F', 'D', 'A', 'C', 'E', 'B']
```

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

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

```
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

```
def build_model(input_shape, loss="sparse_categorical_crossentropy",
learning_rate=0.001):
    """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(27, 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

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

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,
```

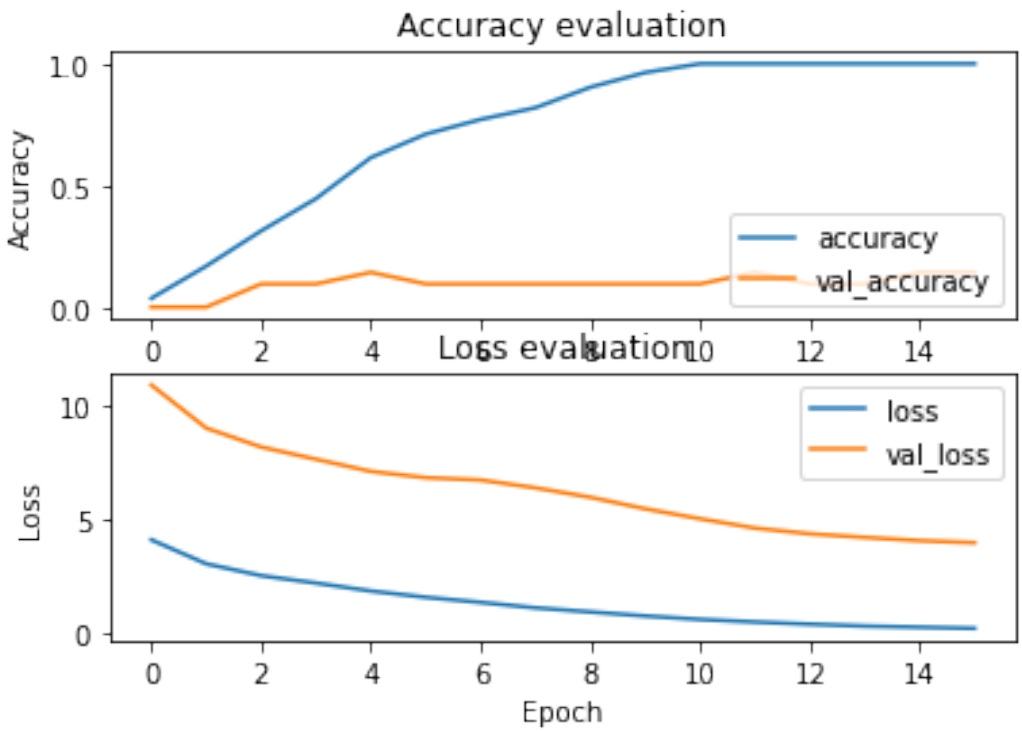
```
100*test_acc))

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

Step 4 Model training

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

```
Training sets loaded!
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)            (None, 63, 11, 64)        640

 batch_normalization_3 (Batc  (None, 63, 11, 64)       256
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 32, 6, 64)        0
 2D)

 conv2d_4 (Conv2D)            (None, 30, 4, 32)         18464

 batch_normalization_4 (Batc  (None, 30, 4, 32)        128
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 15, 2, 32)        0
 2D)

 conv2d_5 (Conv2D)            (None, 14, 1, 32)         4128

 batch_normalization_5 (Batc  (None, 14, 1, 32)        128
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 7, 1, 32)         0
 2D)

 flatten_1 (Flatten)          (None, 224)               0

 dense_2 (Dense)              (None, 64)                14400

 dense_3 (Dense)              (None, 27)                1755

=================================================================
Total params: 39,899
Trainable params: 39,643
Non-trainable params: 256
_________________________________________________________________
```

```
Epoch 1/100
3/3 [==============================] - 1s 155ms/step - loss: 4.1321 -
accuracy: 0.0361 - val_loss: 10.9184 - val_accuracy: 0.0000e+00
Epoch 2/100
3/3 [==============================] - 0s 23ms/step - loss: 3.0788 -
accuracy: 0.1687 - val_loss: 9.0294 - val_accuracy: 0.0000e+00
Epoch 3/100
3/3 [==============================] - 0s 19ms/step - loss: 2.5613 -
accuracy: 0.3133 - val_loss: 8.1999 - val_accuracy: 0.0952
Epoch 4/100
3/3 [==============================] - 0s 21ms/step - loss: 2.2359 -
accuracy: 0.4458 - val_loss: 7.6638 - val_accuracy: 0.0952
Epoch 5/100
3/3 [==============================] - 0s 20ms/step - loss: 1.8876 -
accuracy: 0.6145 - val_loss: 7.1288 - val_accuracy: 0.1429
Epoch 6/100
3/3 [==============================] - 0s 20ms/step - loss: 1.6126 -
accuracy: 0.7108 - val_loss: 6.8555 - val_accuracy: 0.0952
Epoch 7/100
3/3 [==============================] - 0s 20ms/step - loss: 1.3891 -
accuracy: 0.7711 - val_loss: 6.7516 - val_accuracy: 0.0952
Epoch 8/100
3/3 [==============================] - 0s 20ms/step - loss: 1.1498 -
accuracy: 0.8193 - val_loss: 6.4088 - val_accuracy: 0.0952
Epoch 9/100
3/3 [==============================] - 0s 24ms/step - loss: 0.9702 -
accuracy: 0.9036 - val_loss: 5.9956 - val_accuracy: 0.0952
Epoch 10/100
3/3 [==============================] - 0s 21ms/step - loss: 0.7937 -
accuracy: 0.9639 - val_loss: 5.4904 - val_accuracy: 0.0952
Epoch 11/100
3/3 [==============================] - 0s 19ms/step - loss: 0.6417 -
accuracy: 1.0000 - val_loss: 5.0537 - val_accuracy: 0.0952
Epoch 12/100
3/3 [==============================] - 0s 19ms/step - loss: 0.5304 -
accuracy: 1.0000 - val_loss: 4.6441 - val_accuracy: 0.1429
Epoch 13/100
3/3 [==============================] - 0s 20ms/step - loss: 0.4332 -
accuracy: 1.0000 - val_loss: 4.3937 - val_accuracy: 0.0952
Epoch 14/100
3/3 [==============================] - 0s 19ms/step - loss: 0.3510 -
accuracy: 1.0000 - val_loss: 4.2368 - val_accuracy: 0.0952
Epoch 15/100
3/3 [==============================] - 0s 19ms/step - loss: 0.3006 -
accuracy: 1.0000 - val_loss: 4.0947 - val_accuracy: 0.1429
Epoch 16/100
3/3 [==============================] - 0s 20ms/step - loss: 0.2508 -
accuracy: 1.0000 - val_loss: 4.0055 - val_accuracy: 0.1429
```




```
1/1 [==============================] - 0s 27ms/step - loss: 4.5115 -
accuracy: 0.0385

Test loss: 4.511535167694092, test accuracy: 3.8461539894342422
```

Step 5 Making predictions

```
import librosa
import tensorflow as tf
import numpy as np

SAVED_MODEL_PATH = "model.h5"
SAMPLES_TO_CONSIDER = 22050

class _Keyword_Spotting_Service:
    """Singleton class for keyword spotting inference with trained
models.
    :param model: Trained model
    """

    model = None
    _mapping = ['T', 'Q', 'S', 'R', 'Z', 'U', 'X', 'Y', 'W', 'V', 'J',
'H', 'O', 'P', 'K', 'M', 'I', 'G', 'L', 'N', 'F', 'D', 'A', 'C', 'E',
'B']
    _instance = None
```

```
    def predict(self, file_path):
        """
        :param file_path (str): Path to audio file to predict
        :return predicted_keyword (str): Keyword predicted by the
model
        """

        # extract MFCC
        MFCCs = self.preprocess(file_path)

        # we need a 4-dim array to feed to the model for prediction:
(# samples, # time steps, # coefficients, 1)
        MFCCs = MFCCs[np.newaxis, ..., np.newaxis]

        # get the predicted label
        predictions = self.model.predict(MFCCs)
        predicted_index = np.argmax(predictions)
        predicted_keyword = self._mapping[predicted_index]
        return predicted_keyword


    def preprocess(self, file_path, num_mfcc=13, n_fft=2048,
hop_length=512):
        """Extract MFCCs from audio file.
        :param file_path (str): Path of audio file
        :param num_mfcc (int): # of coefficients to extract
        :param n_fft (int): Interval we consider to apply STFT.
Measured in # of samples
        :param hop_length (int): Sliding window for STFT. Measured in
# of samples
        :return MFCCs (ndarray): 2-dim array with MFCC data of shape
(# time steps, # coefficients)
        """

        # load audio file
        signal, sample_rate = librosa.load(file_path)

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

def Keyword_Spotting_Service():
    """Factory function for Keyword_Spotting_Service class.
    :return _Keyword_Spotting_Service._instance
(_Keyword_Spotting_Service):
    """

    # ensure an instance is created only the first time the factory
```

```
function is called
    if _Keyword_Spotting_Service._instance is None:
        _Keyword_Spotting_Service._instance =
_Keyword_Spotting_Service()
        _Keyword_Spotting_Service.model =
tf.keras.models.load_model(SAVED_MODEL_PATH)
    return _Keyword_Spotting_Service._instance
```

PREDICTING ON A SAMPLE

```
if __name__ == "__main__":

    # create 2 instances of the keyword spotting service
    kss = Keyword_Spotting_Service()
    kss1 = Keyword_Spotting_Service()

    # check that different instances of the keyword spotting service
point back to the same object (singleton)
    assert kss is kss1

    # make a prediction
    keyword = kss.predict("/content/D_2_.wav")
    print("The predicted alphabet is :",keyword)
```

```
The predicted alphabet is : D
```