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

DATA_PATH = "/content/drive/MyDrive/Kanhaiya_Final_Project/data.json"
SAVED_MODEL_PATH = "/content/drive/MyDrive/Kanhaiya_Final_Project/model.h5"
EPOCHS = 40
BATCH_SIZE = 32
PATIENCE = 5
LEARNING_RATE = 0.0001


def load_data(data_path):

    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


def prepare_dataset(data_path, test_size=0.2, validation_size=0.2):


    # 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_tra

    # 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


def build_model(input_shape, loss="sparse_categorical_crossentropy", learning_rate=
    """Build neural network using keras.

    :param input_shape (tuple): Shape of array representing a sample train. E.g.: (
    :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=inp
                                     kernel_regularizer=tf.keras.regularizers.l2(0
    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
    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
    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(21, 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_va

    earlystop_callback = tf.keras.callbacks.EarlyStopping(monitor="accuracy", min_

    # 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):


    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(

    # 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_valida

    # 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)


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, 21)                1365

=================================================================
Total params: 35,413
Trainable params: 35,157
Non-trainable params: 256
_________________________________________________________________
Epoch 1/40
5/5 [==============================] - 2s 131ms/step - loss: 3.9020 - accurac
Epoch 2/40
5/5 [==============================] - 0s 51ms/step - loss: 3.5948 - accuracy
Epoch 3/40
5/5 [==============================] - 0s 54ms/step - loss: 3.4734 - accuracy
Epoch 4/40
5/5 [==============================] - 0s 52ms/step - loss: 3.3124 - accuracy
Epoch 5/40
5/5 [==============================] - 0s 49ms/step - loss: 3.1942 - accuracy
Epoch 6/40
5/5 [==============================] - 0s 56ms/step - loss: 3.0934 - accuracy
Epoch 7/40
5/5 [==============================] - 0s 54ms/step - loss: 2.9810 - accuracy
Epoch 8/40
5/5 [==============================] - 0s 49ms/step - loss: 2.8614 - accuracy
Epoch 9/40
5/5 [==============================] - 0s 57ms/step - loss: 2.7854 - accuracy
Epoch 10/40
5/5 [==============================] - 0s 49ms/step - loss: 2.6884 - accuracy
```