Assignment 3

Speech Emotion Recognition

Pattern Recognition

Omar Khaled - 5674
Abdelrahman Khalil - 5797
Ahmed Osama - 5935
Salah Ashraf - 5755
Noureen Mahmoud - 5810

1. Load audio :

```
def ReadAudio(path):
    i=0
    for dirname, _, filenames in os.walk(path):
        for filename in filenames:
            basename = os.path.basename(filename)
            emotion = emotion2int[basename.split("_")[2]]
            Emotions[i]=emotion
            samplerate, data = wavfile.read(os.path.join(dirname, filename))
            zero_crss_rate = ((data[:-1] * data[1:]) < 0).sum()
            energy = np.sum(data.astype(float)**2)
            Energies[i]=energy
            ZeroRates[i]=zero_crss_rate
            frequencies, times, spectrogram = signal.spectrogram(data, samplerate)
            DB = librosa.amplitude_to_db(spectrogram, ref=np.max)
            MelSpectrograms[i]=DB
            i=i+1
```

```
path="/content/drive/MyDrive/CREMA"
emotion2int = {
    "NEU": 1,
    "HAP": 2,
    "SAD": 3,
    "DIS": 4,
    "FEA": 5,
    "ANG": 6
}
Energies=np.zeros(7498,float)
ZeroRates=np.zeros(7498,int)
Emotions=np.zeros(7498,int)
MelSpectrograms = dict.fromkeys(range(7498), [])
```

2. Create Feature Space:

a. Zero Crossing Rate:

The rate of sign-changes of the signal during the duration of a particular frame.

b. Convert audio waveform to mel spectrogram

Both are created in ReadAudio function

```
# Resizing MelSpectrograms to the size of the largest
import cv2
import matplotlib.pyplot as plt

MelSpectrograms_rs= dict.fromkeys(range(7498), [])
for i in range(7498):
  MelSpectrograms_rs[i] = cv2.resize(MelSpectrograms[i],(357,129))
```

```
[ ] FeatureSpace2_Train = np.array(FeatureSpace2_Train)
    FeatureSpace2_Train = FeatureSpace2_Train[:,:,:,np.newaxis]
    print(FeatureSpace2_Train.shape)
    FeatureSpace2_Test = np.array(FeatureSpace2_Test)
    FeatureSpace2_Test = FeatureSpace2_Test[:,:,:,np.newaxis]
    print(FeatureSpace2_Test.shape)
```

```
(5248, 129, 357, 1)
(2250, 129, 357, 1)
```

```
[ ] print(type(FeatureSpace2_Train[0][0][0][0]))
    FeatureSpace2_Train = FeatureSpace2_Train.astype('float32')
    FeatureSpace2_Test = FeatureSpace2_Test.astype('float32')
    FeatureSpace2_Train = FeatureSpace2_Train / 255.
    FeatureSpace2_Test = FeatureSpace2_Test / 255.
    print(FeatureSpace2_Train[0][0][0][0])
```

```
-0.063708134
```

```
[ ] FeatureSpace2_Train = abs(FeatureSpace2_Train)
    FeatureSpace2_Test = abs(FeatureSpace2_Test)
    print(FeatureSpace2_Train[0][0][0][0])
```

```
0.063708134
```

```
[ ] !pip install keras.utils
    from tensorflow.keras.utils import to_categorical
    FeatureSpace2_TrainLabels = FeatureSpace2_TrainLabels - 1
    FeatureSpace2_TestLabels = FeatureSpace2_TestLabels - 1
    train_Y_one_hot = to_categorical(FeatureSpace2_TrainLabels)
    test_Y_one_hot = to_categorical(FeatureSpace2_TestLabels)
    print('Original label:', FeatureSpace2_TrainLabels[0])
    print('After conversion to one-hot:', train_Y_one_hot[0])
```

```
Original label: 4
After conversion to one-hot: [0. 0. 0. 0. 1. 0.]
```

3. Building the Model :

a. Split the data into 70% training and validation and 30% testing.

```
[12] def split(featureSpace,labels,testRatio):
        X_train, X_test, y_train, y_test = train_test_split(featureSpace,labels,test_size=testRatio,shuffle=False)
        return X_train, X_test, y_train, y_test
```

```
[13] FeatureSpace1_Train,FeatureSpace1_Test,FeatureSpace1_TrainLabels,FeatureSpace1_TestLabels= split(Energies,Emotions,0.3)
     FeatureSpace2_Train,FeatureSpace2_Test,FeatureSpace2_TrainLabels,FeatureSpace2_TestLabels= split(MelSpectrograms_rs,Emotions,0.3)
```

b. Use 5% of the training and validation data for validation

```
[ ] #validation split
    X_dummy, valid_X , y_dummy, valid_label = split(FeatureSpace2_Train,train_Y_one_hot,0.05)
    valid_X = np.array(valid_X)
    valid_label = np.array(valid_label)
    X_dummy = np.array(X_dummy)
    y_dummy = np.array(y_dummy)
```

c. CNN model

1D -> feature space :

```
[ ] model = tf.keras.models.Sequential()

    model.add(tf.keras.layers.Conv1D(64, 10, padding='same', input_shape=Train[0].shape))
    model.add(tf.keras.layers.Activation('relu'))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.MaxPooling1D(pool_size=(4)))

    model.add(tf.keras.layers.Conv1D(64, 10,padding='same',))
    model.add(tf.keras.layers.Activation('relu'))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.MaxPooling1D(pool_size=(4)))

    model.add(tf.keras.layers.Flatten())

    model.add(tf.keras.layers.Dense(256))
    model.add(tf.keras.layers.Dropout(0.2))

    model.add(tf.keras.layers.Dense(128))
    model.add(tf.keras.layers.Dropout(0.1))

    model.add(tf.keras.layers.Dense(6))
    model.add(tf.keras.layers.Activation('softmax'))
    model.compile(optimizer=Adam(lr=0.0006), loss='categorical_crossentropy', metrics=['accuracy'])
    model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_2 (Conv1D)            (None, 216, 64)           704
_________________________________________________________________
activation_3 (Activation)    (None, 216, 64)           0
_________________________________________________________________
dropout_4 (Dropout)          (None, 216, 64)           0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 54, 64)            0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 54, 64)            41024
_________________________________________________________________
activation_4 (Activation)    (None, 54, 64)            0
_________________________________________________________________
dropout_5 (Dropout)          (None, 54, 64)            0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 13, 64)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 832)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               213248
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 774
_________________________________________________________________
activation_5 (Activation)    (None, 6)                 0
=================================================================
Total params: 288,646
Trainable params: 288,646
Non-trainable params: 0
```

```
[ ] TensorTrain = tf.convert_to_tensor(Train)
    TensorTrainSqueezed = tf.convert_to_tensor(TrainSqueezed)
```

```
[ ] train_m, train_val , hot_m, hot_val = split(Train,TrainHot,0.05)
    train_m= np.array(train_m)
    train_val= np.array(train_val)
    hot_m= np.array(hot_m)
    hot_val= np.array(hot_val)
```

```
[ ] Energy_Model_train = model.fit(train_m, hot_m, batch_size=64,epochs=100
                                   ,verbose=1,validation_data=(train_val, hot_val))
```

```
Epoch 1/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6060 - accuracy: 0.3206 - val_loss: 1.6888 - val_accuracy: 0.2662
Epoch 2/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6049 - accuracy: 0.3147 - val_loss: 1.6830 - val_accuracy: 0.2738
Epoch 3/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6080 - accuracy: 0.3172 - val_loss: 1.6821 - val_accuracy: 0.2662
Epoch 4/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6050 - accuracy: 0.3137 - val_loss: 1.6924 - val_accuracy: 0.2700
Epoch 5/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6051 - accuracy: 0.3254 - val_loss: 1.6973 - val_accuracy: 0.2738
Epoch 6/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6032 - accuracy: 0.3194 - val_loss: 1.6870 - val_accuracy: 0.2624
Epoch 7/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6032 - accuracy: 0.3186 - val_loss: 1.7039 - val_accuracy: 0.2586
Epoch 8/100
78/78 [==============================] - 5s 58ms/step - loss: 1.6049 - accuracy: 0.3198 - val_loss: 1.6948 - val_accuracy: 0.2624
Epoch 9/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6047 - accuracy: 0.3216 - val_loss: 1.6866 - val_accuracy: 0.2814
Epoch 10/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6019 - accuracy: 0.3214 - val_loss: 1.6855 - val_accuracy: 0.2700
Epoch 11/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6018 - accuracy: 0.3184 - val_loss: 1.6812 - val_accuracy: 0.2738
Epoch 12/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6037 - accuracy: 0.3192 - val_loss: 1.6801 - val_accuracy: 0.2662
Epoch 13/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6036 - accuracy: 0.3180 - val_loss: 1.6851 - val_accuracy: 0.2776
Epoch 14/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6035 - accuracy: 0.3190 - val_loss: 1.6913 - val_accuracy: 0.2662
Epoch 15/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6040 - accuracy: 0.3224 - val_loss: 1.6858 - val_accuracy: 0.2776
Epoch 16/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6003 - accuracy: 0.3216 - val_loss: 1.6833 - val_accuracy: 0.2662
Epoch 17/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6036 - accuracy: 0.3190 - val_loss: 1.6749 - val_accuracy: 0.2776
Epoch 18/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6037 - accuracy: 0.3206 - val_loss: 1.6850 - val_accuracy: 0.2738
Epoch 19/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6021 - accuracy: 0.3210 - val_loss: 1.6950 - val_accuracy: 0.2624
Epoch 20/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6016 - accuracy: 0.3188 - val_loss: 1.6906 - val_accuracy: 0.2662
Epoch 21/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6013 - accuracy: 0.3228 - val_loss: 1.7083 - val_accuracy: 0.2548
Epoch 22/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6013 - accuracy: 0.3220 - val_loss: 1.6765 - val_accuracy: 0.2814
Epoch 23/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6039 - accuracy: 0.3218 - val_loss: 1.6927 - val_accuracy: 0.2738
Epoch 24/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6028 - accuracy: 0.3176 - val_loss: 1.6855 - val_accuracy: 0.2738
Epoch 25/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6005 - accuracy: 0.3188 - val_loss: 1.6825 - val_accuracy: 0.2738
Epoch 26/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6019 - accuracy: 0.3200 - val_loss: 1.6931 - val_accuracy: 0.2700
Epoch 27/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6010 - accuracy: 0.3238 - val_loss: 1.6994 - val_accuracy: 0.2548
Epoch 28/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6071 - accuracy: 0.3157 - val_loss: 1.6853 - val_accuracy: 0.2738
Epoch 29/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6018 - accuracy: 0.3186 - val_loss: 1.6931 - val_accuracy: 0.2624
Epoch 30/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6023 - accuracy: 0.3238 - val_loss: 1.6919 - val_accuracy: 0.2738
Epoch 31/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6008 - accuracy: 0.3264 - val_loss: 1.6938 - val_accuracy: 0.2662
Epoch 32/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6022 - accuracy: 0.3206 - val_loss: 1.6793 - val_accuracy: 0.2738
Epoch 33/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6023 - accuracy: 0.3161 - val_loss: 1.6856 - val_accuracy: 0.2776
Epoch 34/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6016 - accuracy: 0.3208 - val_loss: 1.6806 - val_accuracy: 0.2738
Epoch 35/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6012 - accuracy: 0.3190 - val_loss: 1.7042 - val_accuracy: 0.2586
Epoch 36/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6033 - accuracy: 0.3246 - val_loss: 1.6871 - val_accuracy: 0.2586
Epoch 37/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6023 - accuracy: 0.3206 - val_loss: 1.6806 - val_accuracy: 0.2776
Epoch 38/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6003 - accuracy: 0.3228 - val_loss: 1.6841 - val_accuracy: 0.2548
Epoch 39/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6022 - accuracy: 0.3200 - val_loss: 1.6902 - val_accuracy: 0.2700
Epoch 40/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6021 - accuracy: 0.3184 - val_loss: 1.6873 - val_accuracy: 0.2738
Epoch 41/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6000 - accuracy: 0.3234 - val_loss: 1.6958 - val_accuracy: 0.2586
Epoch 42/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5987 - accuracy: 0.3222 - val_loss: 1.6889 - val_accuracy: 0.2814
Epoch 43/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6039 - accuracy: 0.3254 - val_loss: 1.6802 - val_accuracy: 0.2700
Epoch 44/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6036 - accuracy: 0.3210 - val_loss: 1.6898 - val_accuracy: 0.2776
Epoch 45/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6072 - accuracy: 0.3220 - val_loss: 1.6855 - val_accuracy: 0.2738
Epoch 46/100
78/78 [==============================] - 5s 59ms/step - loss: 1.5996 - accuracy: 0.3226 - val_loss: 1.6867 - val_accuracy: 0.2510
Epoch 47/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6023 - accuracy: 0.3165 - val_loss: 1.6918 - val_accuracy: 0.2662
Epoch 48/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5990 - accuracy: 0.3214 - val_loss: 1.6883 - val_accuracy: 0.2662
Epoch 49/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6012 - accuracy: 0.3252 - val_loss: 1.6811 - val_accuracy: 0.2738
Epoch 50/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6041 - accuracy: 0.3159 - val_loss: 1.6867 - val_accuracy: 0.2700
```

```
Epoch 51/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6027 - accuracy: 0.3206 - val_loss: 1.6915 - val_accuracy: 0.2548
Epoch 52/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6003 - accuracy: 0.3216 - val_loss: 1.6779 - val_accuracy: 0.2776
Epoch 53/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6013 - accuracy: 0.3174 - val_loss: 1.6971 - val_accuracy: 0.2738
Epoch 54/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6024 - accuracy: 0.3184 - val_loss: 1.6980 - val_accuracy: 0.2700
Epoch 55/100
78/78 [==============================] - 5s 59ms/step - loss: 1.5988 - accuracy: 0.3206 - val_loss: 1.6789 - val_accuracy: 0.2700
Epoch 56/100
78/78 [==============================] - 5s 59ms/step - loss: 1.5993 - accuracy: 0.3240 - val_loss: 1.6789 - val_accuracy: 0.2624
Epoch 57/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5991 - accuracy: 0.3200 - val_loss: 1.6864 - val_accuracy: 0.2471
Epoch 58/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6013 - accuracy: 0.3210 - val_loss: 1.6958 - val_accuracy: 0.2662
Epoch 59/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6000 - accuracy: 0.3242 - val_loss: 1.6812 - val_accuracy: 0.2586
Epoch 60/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5985 - accuracy: 0.3206 - val_loss: 1.7003 - val_accuracy: 0.2738
Epoch 61/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5997 - accuracy: 0.3200 - val_loss: 1.6849 - val_accuracy: 0.2700
Epoch 62/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5997 - accuracy: 0.3202 - val_loss: 1.6846 - val_accuracy: 0.2662
Epoch 63/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5996 - accuracy: 0.3172 - val_loss: 1.6855 - val_accuracy: 0.2624
Epoch 64/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5989 - accuracy: 0.3202 - val_loss: 1.6968 - val_accuracy: 0.2738
Epoch 65/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5995 - accuracy: 0.3230 - val_loss: 1.6853 - val_accuracy: 0.2814
Epoch 66/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6024 - accuracy: 0.3202 - val_loss: 1.6815 - val_accuracy: 0.2700
Epoch 67/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6014 - accuracy: 0.3220 - val_loss: 1.6965 - val_accuracy: 0.2624
Epoch 68/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6037 - accuracy: 0.3194 - val_loss: 1.6761 - val_accuracy: 0.2776
Epoch 69/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6023 - accuracy: 0.3206 - val_loss: 1.6770 - val_accuracy: 0.2738
Epoch 70/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6015 - accuracy: 0.3178 - val_loss: 1.6900 - val_accuracy: 0.2738
Epoch 71/100
78/78 [==============================] - 5s 59ms/step - loss: 1.5997 - accuracy: 0.3216 - val_loss: 1.6862 - val_accuracy: 0.2586
Epoch 72/100
78/78 [==============================] - 5s 59ms/step - loss: 1.6007 - accuracy: 0.3234 - val_loss: 1.6858 - val_accuracy: 0.2700
Epoch 73/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5993 - accuracy: 0.3184 - val_loss: 1.6833 - val_accuracy: 0.2814
Epoch 74/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6012 - accuracy: 0.3194 - val_loss: 1.6844 - val_accuracy: 0.2624
Epoch 75/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5994 - accuracy: 0.3234 - val_loss: 1.6932 - val_accuracy: 0.2738
Epoch 76/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5993 - accuracy: 0.3240 - val_loss: 1.6974 - val_accuracy: 0.2662
Epoch 77/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6004 - accuracy: 0.3204 - val_loss: 1.6823 - val_accuracy: 0.2852
Epoch 78/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6020 - accuracy: 0.3216 - val_loss: 1.6920 - val_accuracy: 0.2586
Epoch 79/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5994 - accuracy: 0.3254 - val_loss: 1.6869 - val_accuracy: 0.2700
Epoch 80/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6001 - accuracy: 0.3200 - val_loss: 1.6857 - val_accuracy: 0.2700
Epoch 81/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5990 - accuracy: 0.3188 - val_loss: 1.6883 - val_accuracy: 0.2738
Epoch 82/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5999 - accuracy: 0.3216 - val_loss: 1.6865 - val_accuracy: 0.2624
Epoch 83/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5997 - accuracy: 0.3204 - val_loss: 1.6990 - val_accuracy: 0.2624
Epoch 84/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5994 - accuracy: 0.3222 - val_loss: 1.6833 - val_accuracy: 0.2700
Epoch 85/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5985 - accuracy: 0.3240 - val_loss: 1.6988 - val_accuracy: 0.2662
Epoch 86/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5988 - accuracy: 0.3188 - val_loss: 1.6934 - val_accuracy: 0.2662
Epoch 87/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6004 - accuracy: 0.3226 - val_loss: 1.6808 - val_accuracy: 0.2738
Epoch 88/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6026 - accuracy: 0.3224 - val_loss: 1.6894 - val_accuracy: 0.2662
Epoch 89/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5988 - accuracy: 0.3240 - val_loss: 1.6801 - val_accuracy: 0.2776
Epoch 90/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6014 - accuracy: 0.3208 - val_loss: 1.6824 - val_accuracy: 0.2814
Epoch 91/100
78/78 [==============================] - 5s 60ms/step - loss: 1.6010 - accuracy: 0.3202 - val_loss: 1.6849 - val_accuracy: 0.2700
Epoch 92/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6006 - accuracy: 0.3228 - val_loss: 1.6793 - val_accuracy: 0.2776
Epoch 93/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6014 - accuracy: 0.3232 - val_loss: 1.6816 - val_accuracy: 0.2814
Epoch 94/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5981 - accuracy: 0.3220 - val_loss: 1.6911 - val_accuracy: 0.2814
Epoch 95/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5990 - accuracy: 0.3248 - val_loss: 1.6891 - val_accuracy: 0.2776
Epoch 96/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5992 - accuracy: 0.3214 - val_loss: 1.6870 - val_accuracy: 0.2738
Epoch 97/100
78/78 [==============================] - 5s 60ms/step - loss: 1.5972 - accuracy: 0.3268 - val_loss: 1.6860 - val_accuracy: 0.2700
Epoch 98/100
78/78 [==============================] - 5s 61ms/step - loss: 1.5998 - accuracy: 0.3208 - val_loss: 1.6917 - val_accuracy: 0.2662
Epoch 99/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6004 - accuracy: 0.3202 - val_loss: 1.6877 - val_accuracy: 0.2586
Epoch 100/100
78/78 [==============================] - 5s 61ms/step - loss: 1.6011 - accuracy: 0.3226 - val_loss: 1.6782 - val_accuracy: 0.2700
```

```
[ ] model.save("/content/drive/MyDrive/1D.h5py")
```

2D -> mel spectrogram

```
[ ] spectr_model = Sequential()
    spectr_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape=(129,357,1),padding='same'))
    spectr_model.add(LeakyReLU(alpha=0.1))
    spectr_model.add(MaxPooling2D((2, 2),padding='same'))
    spectr_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
    spectr_model.add(LeakyReLU(alpha=0.1))
    spectr_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    spectr_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
    spectr_model.add(LeakyReLU(alpha=0.1))
    spectr_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    spectr_model.add(Flatten())
    spectr_model.add(Dense(128, activation='linear'))
    spectr_model.add(LeakyReLU(alpha=0.1))
    spectr_model.add(Dense(num_classes, activation='softmax'))
    spectr_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(),
                         metrics=['accuracy'])
    spectr_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 129, 357, 32)      320
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 129, 357, 32)      0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 65, 179, 32)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 65, 179, 64)       18496
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 65, 179, 64)       0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 33, 90, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 33, 90, 128)       73856
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 33, 90, 128)       0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 17, 45, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 97920)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               12533888
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 774
=================================================================
Total params: 12,627,334
Trainable params: 12,627,334
Non-trainable params: 0
```

```
[ ] spectr_train = spectr_model.fit(X_dummy, y_dummy, batch_size=64,epochs=20,
                                    verbose=1,validation_data=(valid_X, valid_label))
```

```
Epoch 1/20
78/78 [==============================] - 446s 6s/step - loss: 1.9810 - accuracy: 0.1675 - val_loss: 1.7469 - val_accuracy: 0.2586
Epoch 2/20
78/78 [==============================] - 444s 6s/step - loss: 1.5997 - accuracy: 0.3172 - val_loss: 1.5742 - val_accuracy: 0.3156
Epoch 3/20
78/78 [==============================] - 444s 6s/step - loss: 1.4012 - accuracy: 0.4261 - val_loss: 1.5227 - val_accuracy: 0.3574
Epoch 4/20
78/78 [==============================] - 439s 6s/step - loss: 1.3471 - accuracy: 0.4624 - val_loss: 1.3869 - val_accuracy: 0.4373
Epoch 5/20
78/78 [==============================] - 438s 6s/step - loss: 1.2322 - accuracy: 0.5060 - val_loss: 1.3451 - val_accuracy: 0.4525
Epoch 6/20
78/78 [==============================] - 442s 6s/step - loss: 1.1237 - accuracy: 0.5574 - val_loss: 1.3545 - val_accuracy: 0.4221
Epoch 7/20
78/78 [==============================] - 441s 6s/step - loss: 0.9670 - accuracy: 0.6428 - val_loss: 1.3155 - val_accuracy: 0.4449
Epoch 8/20
78/78 [==============================] - 437s 6s/step - loss: 0.8462 - accuracy: 0.6812 - val_loss: 1.3688 - val_accuracy: 0.4411
Epoch 9/20
78/78 [==============================] - 437s 6s/step - loss: 0.6986 - accuracy: 0.7430 - val_loss: 1.4312 - val_accuracy: 0.4677
Epoch 10/20
78/78 [==============================] - 438s 6s/step - loss: 0.5155 - accuracy: 0.8194 - val_loss: 1.6843 - val_accuracy: 0.4829
Epoch 11/20
78/78 [==============================] - 438s 6s/step - loss: 0.3203 - accuracy: 0.8951 - val_loss: 1.8952 - val_accuracy: 0.4525
Epoch 12/20
78/78 [==============================] - 438s 6s/step - loss: 0.1845 - accuracy: 0.9443 - val_loss: 2.7507 - val_accuracy: 0.4487
Epoch 13/20
78/78 [==============================] - 435s 6s/step - loss: 0.0954 - accuracy: 0.9724 - val_loss: 3.0505 - val_accuracy: 0.4943
Epoch 14/20
78/78 [==============================] - 434s 6s/step - loss: 0.0496 - accuracy: 0.9872 - val_loss: 3.3918 - val_accuracy: 0.4867
Epoch 15/20
78/78 [==============================] - 439s 6s/step - loss: 0.0248 - accuracy: 0.9960 - val_loss: 3.5317 - val_accuracy: 0.4791
Epoch 16/20
78/78 [==============================] - 437s 6s/step - loss: 0.0111 - accuracy: 0.9988 - val_loss: 3.9494 - val_accuracy: 0.4677
Epoch 17/20
78/78 [==============================] - 434s 6s/step - loss: 0.0036 - accuracy: 0.9999 - val_loss: 4.3309 - val_accuracy: 0.4829
Epoch 18/20
78/78 [==============================] - 432s 6s/step - loss: 0.0089 - accuracy: 0.9993 - val_loss: 4.3011 - val_accuracy: 0.4791
Epoch 19/20
78/78 [==============================] - 429s 5s/step - loss: 8.3158e-04 - accuracy: 1.0000 - val_loss: 4.5760 - val_accuracy: 0.4867
Epoch 20/20
78/78 [==============================] - 430s 6s/step - loss: 6.2716e-04 - accuracy: 1.0000 - val_loss: 4.5327 - val_accuracy: 0.4867
```

```
[ ] spectr_model.save("/content/drive/MyDrive/spectr_model_no_dropout.h5py")
```

4. Big Picture :

Comparing between the performance of the learned models (Different features, different learning models) by realizing the following.

a. Compute the accuracy and F-Score for each model

```
[ ] def measure_f(grndtruth, y_predict):
      fmeasure=f1_score(grndtruth,y_predict,average='micro')
      print("The F-Measure Score is:\n",fmeasure)
```

1D

```
[ ] test_eval_1D = model.evaluate(Test, TestHot, verbose=1)
    print('Test loss:', test_eval_1D[0])
    print('Test accuracy:', test_eval_1D[1])
```

```
Test loss: 1.6291776895523071
Test accuracy: 0.31244444847106934
```

```
[ ] energy_predict=model.predict(Test,verbose=1)

    energy_predict_not_hot=energy_predict.argmax(axis=1)
    TestnotHot=TestHot.argmax(axis=1)
    print(np.unique(energy_predict_not_hot))
    print(np.unique(TestnotHot))
    measure_f(TestnotHot,energy_predict_not_hot)
```

```
The F-Measure Score is:
 0.3151111111111111
```

2D

```
[ ] test_eval = spectr_model.evaluate(FeatureSpace2_Test, test_Y_one_hot, verbose=1)
    print('Test loss:', test_eval[0])
    print('Test accuracy:', test_eval[1])
```

```
Test loss: 4.837060451507568
Test accuracy: 0.46977776288986206
```

```
[ ] spectr_predict_not_hot=spectr_predict.argmax(axis=1)
    test_Y_not_hot=test_Y_one_hot.argmax(axis=1)
    print(np.unique(spectr_predict_not_hot))
    print(np.unique(test_Y_not_hot))
    measure_f(test_Y_not_hot,spectr_predict_not_hot)
```

```
The F-Measure Score is:
 0.18088888888888888
```

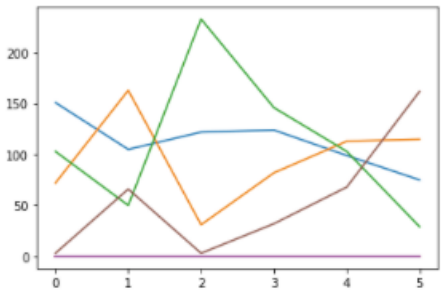
b. Plot the confusion matrices and find the most confusing classes

1D

```
[ ] confusion_energy = confusion(TestnotHot, energy_predict_not_hot)
    print(confusion_energy)
```

```
[ ] plt.plot(confusion_energy)
    plt.show()
```

```
[[151  72 103   0   0   3]
 [105 163  50   0   0  66]
 [122  31 233   0   0   3]
 [124  82 146   0   0  32]
 [ 99 113 103   0   0  68]
 [ 75 115  29   0   0 162]]
```



2D

```
[ ] confusion_spectr = confusion(test_Y_not_hot, spectr_predict_not_hot)
    print(confusion_spectr)
```

```
[ ] plt.plot(confusion_spectr)
    plt.show()
```

```
[[  0   0 329   0   0   0]
 [  0   3 377   0   2   2]
 [  0   0 389   0   0   0]
 [  0   0 384   0   0   0]
 [  0   5 377   0   0   1]
 [  1  27 336   1   1  15]]
```