Import Libraries

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

#!pip install tensorflow-io

import os  
import numpy as np  
from matplotlib import pyplot as plt  
import tensorflow as tf   
import tensorflow\_io as tfio  
from tensorflow import keras  
from keras import backend as k

# 1. Process Audio into Spectogram

Variables: batch: the process of splitting the training dataset in n batches (mini-batches), classes: number of classifications (labels) of the data, epochs: variations, one epoch is one forward pass + one backward pass on training

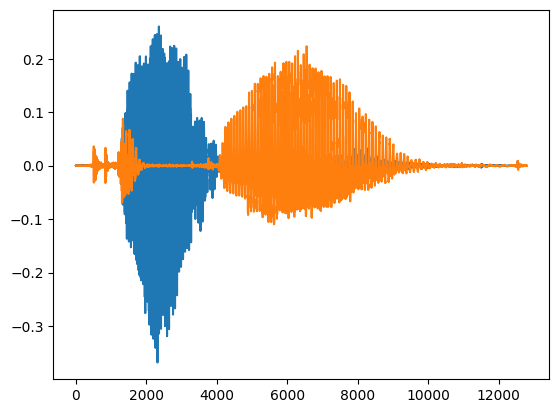
#batch\_size = 20  
num\_classes = 10  
epochs = 4

a function that returns audio in numeric representation

def load\_wav\_16k\_mono(filename):  
 # Load encoded wav file  
 file\_contents = tf.io.read\_file(filename)  
 # Decode wav (tensors by channels)   
 wav, sample\_rate = tf.audio.decode\_wav(file\_contents, desired\_channels=1)  
 # Removes trailing axis  
 wav = tf.squeeze(wav, axis=-1)  
 sample\_rate = tf.cast(sample\_rate, dtype=tf.int64)  
 # Goes from 44100Hz to 16000hz - amplitude of the audio signal  
 #wav = tfio.audio.resample(wav, rate\_in=sample\_rate, rate\_out=16000)  
 return wav

Check a sample of audio

TRAIN\_FILE = os.path.join('/content','drive','MyDrive','audio-data','Train','C03n\_0.wav')  
TEST\_FILE = os.path.join('/content','drive','MyDrive','audio-data','Test','C04n\_2.wav')  
wave = load\_wav\_16k\_mono(TRAIN\_FILE)  
nwave = load\_wav\_16k\_mono(TEST\_FILE)  
plt.plot(wave)  
plt.plot(nwave)  
plt.show()



TRAIN = os.path.join('/content','drive','MyDrive','audio-data', 'Train')  
TEST = os.path.join('/content','drive','MyDrive','audio-data', 'Test')

Read all audio files and sort

train = tf.data.Dataset.list\_files(TRAIN+'/\*.wav')  
train = sorted(list(train.as\_numpy\_iterator()))  
train = tf.data.Dataset.from\_tensor\_slices(train)  
test = tf.data.Dataset.list\_files(TEST+'/\*.wav')  
test = sorted(list(test.as\_numpy\_iterator()))  
test = tf.data.Dataset.from\_tensor\_slices(test)

Add Labels

iterations = 0  
i = 0  
train\_label = []  
while iterations!=len(train):  
 iterations +=1  
 train\_label.append(i)  
 i += 1  
 if i == 10 :  
 i = 0  
train\_label=keras.utils.to\_categorical(train\_label,num\_classes)  
trainings = tf.data.Dataset.zip((train, tf.data.Dataset.from\_tensor\_slices(train\_label)))  
#---------------------------------------------------------------#  
iterations = 0  
i = 0  
test\_label=[]  
while iterations!=len(test):  
 iterations +=1  
 test\_label.append(i)  
 i += 1  
 if i == 10 :  
 i = 0  
test\_label=keras.utils.to\_categorical(test\_label,num\_classes)  
testings = tf.data.Dataset.zip((test, tf.data.Dataset.from\_tensor\_slices(test\_label)))

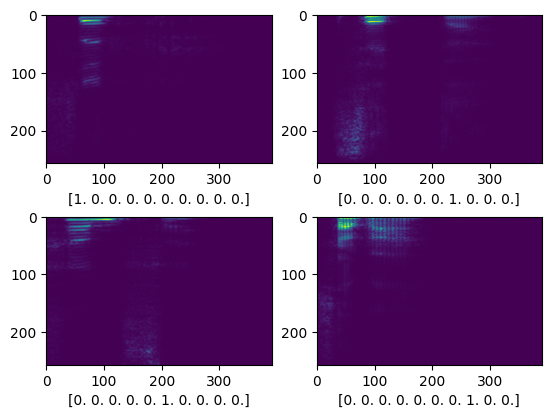
Build Preprocessing Function to get spectogram

def preprocess(file\_path, label):   
 wav = load\_wav\_16k\_mono(file\_path)  
 #wav = wav[:48000]  
 #zero\_padding = tf.zeros([48000] - tf.shape(wav), dtype=tf.float32)  
 #wav = tf.concat([zero\_padding, wav],0)  
 spectrogram = tf.signal.stft(wav, frame\_length=320, frame\_step=32)  
 spectrogram = tf.abs(spectrogram)  
 spectrogram = tf.expand\_dims(spectrogram, axis=2)  
 return spectrogram, label

Draw examples of spectogram

for i in range(4):  
 filepath, label = trainings.shuffle(buffer\_size=10000).as\_numpy\_iterator().next()  
 spectrogram, label = preprocess(filepath, label)  
 plt.subplot(2,2,i+1)  
 plt.imshow(tf.transpose(spectrogram)[0])  
 plt.xlabel(label)  
plt.show()

/usr/local/lib/python3.9/dist-packages/matplotlib/text.py:1279: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison  
 if s != self.\_text:



Convert all to Spectogram

# train data  
x\_train = trainings.map(preprocess)  
x\_train = x\_train.cache()  
x\_train = x\_train.shuffle(buffer\_size=1000)  
x\_train = x\_train.batch(16) # 16 at a time  
x\_train = x\_train.prefetch(8)  
# test data  
x\_test = testings.map(preprocess)  
x\_test = x\_test.cache()  
x\_test = x\_test.shuffle(buffer\_size=1000)  
x\_test = x\_test.batch(16) # 16 at a time  
x\_test = x\_test.prefetch(8)

# test one batch  
samples, labels = x\_train.as\_numpy\_iterator().next()  
print(samples.shape)  
print('\n',labels)

(16, 391, 257, 1)  
  
 [[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]  
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]  
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]  
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]  
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]  
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]

# 2. CNN no Attention

Design the CNN architecture

from keras.models import Sequential  
from keras.layers import Dense,Flatten,Input  
from keras.layers import Conv2D,MaxPool2D,Multiply

model=Sequential()  
input\_shape = (391, 257, 1)  
model.add( Conv2D(32,kernel\_size=(3,3),activation='relu',input\_shape=input\_shape) )  
model.add( MaxPool2D(pool\_size=(2,2)) )  
model.add( Conv2D(64,kernel\_size=(3,3),activation='relu') )  
model.add( MaxPool2D(pool\_size=(2,2)) )  
model.add( Flatten() )  
model.add( Dense(32,activation='relu') )  
model.add( Dense(num\_classes,activation='softmax') )  
model.summary()

Model: "sequential\_3"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 conv2d\_10 (Conv2D) (None, 389, 255, 32) 320   
   
 max\_pooling2d\_10 (MaxPoolin (None, 194, 127, 32) 0   
 g2D)   
   
 conv2d\_11 (Conv2D) (None, 192, 125, 64) 18496   
   
 max\_pooling2d\_11 (MaxPoolin (None, 96, 62, 64) 0   
 g2D)   
   
 flatten\_4 (Flatten) (None, 380928) 0   
   
 dense\_8 (Dense) (None, 32) 12189728   
   
 dense\_9 (Dense) (None, 10) 330   
   
=================================================================  
Total params: 12,208,874  
Trainable params: 12,208,874  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training

model.compile(optimizer=keras.optimizers.Adam(),  
 loss=keras.losses.categorical\_crossentropy,  
 metrics=['accuracy']  
 )

model.fit(x\_train,  
 epochs=epochs,  
 verbose=1,  
 validation\_data=x\_test  
 )

Epoch 1/4  
75/75 [==============================] - 13s 72ms/step - loss: 1.6801 - accuracy: 0.4983 - val\_loss: 1.0176 - val\_accuracy: 0.6967  
Epoch 2/4  
75/75 [==============================] - 3s 40ms/step - loss: 0.4932 - accuracy: 0.8600 - val\_loss: 1.0462 - val\_accuracy: 0.7667  
Epoch 3/4  
75/75 [==============================] - 3s 39ms/step - loss: 0.2516 - accuracy: 0.9408 - val\_loss: 1.0525 - val\_accuracy: 0.7800  
Epoch 4/4  
75/75 [==============================] - 3s 38ms/step - loss: 0.1798 - accuracy: 0.9625 - val\_loss: 1.2533 - val\_accuracy: 0.7633

<keras.callbacks.History at 0x7f9556ff2e20>

test\_loss, test\_acc = model.evaluate(x\_test)  
print('Test Accuracy = {:.2f} %:'.format(np.round(test\_acc, 3)\*100))

19/19 [==============================] - 0s 15ms/step - loss: 1.2533 - accuracy: 0.7633  
Test Accuracy = 76.30 %:

# 3. CNN with Attention

Design the CNN architecture

input\_shape = (391, 257, 1)  
inputs = Input(shape=input\_shape)  
  
conv1 = Conv2D(32,kernel\_size=(3,3),activation='relu')(inputs)  
pool1 = MaxPool2D(pool\_size=(2,2))(conv1)  
#Attention1  
attention\_conv1 = Conv2D(1, (1,1), padding='same', activation='sigmoid')(pool1)  
attention\_mul1 = Multiply()([pool1, attention\_conv1])  
pool2 = MaxPool2D(pool\_size=(2,2))(attention\_mul1)  
##########  
  
conv2 = Conv2D(64,kernel\_size=(3,3),activation='relu')(pool2)  
pool3 = MaxPool2D(pool\_size=(2,2))(conv2)  
#Attention2  
attention\_conv2 = Conv2D(1, (1,1), padding='same', activation='sigmoid')(pool3)  
attention\_mul2 = Multiply()([pool3, attention\_conv2])  
pool4 = MaxPool2D(pool\_size=(2,2))(attention\_mul2)  
##########  
  
flatten2 = Flatten()(pool4)  
dense2 = Dense(32,activation='relu')(flatten2)  
dense3 = Dense(num\_classes,activation='softmax')(dense2)  
  
modelAtt = keras.Model(inputs=inputs, outputs=dense3)  
  
modelAtt.summary()

Model: "model\_1"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param # Connected to   
==================================================================================================  
 input\_2 (InputLayer) [(None, 391, 257, 1 0 []   
 )]   
   
 conv2d\_12 (Conv2D) (None, 389, 255, 32 320 ['input\_2[0][0]']   
 )   
   
 max\_pooling2d\_12 (MaxPooling2D (None, 194, 127, 32 0 ['conv2d\_12[0][0]']   
 ) )   
   
 conv2d\_13 (Conv2D) (None, 194, 127, 1) 33 ['max\_pooling2d\_12[0][0]']   
   
 multiply\_2 (Multiply) (None, 194, 127, 32 0 ['max\_pooling2d\_12[0][0]',   
 ) 'conv2d\_13[0][0]']   
   
 max\_pooling2d\_13 (MaxPooling2D (None, 97, 63, 32) 0 ['multiply\_2[0][0]']   
 )   
   
 conv2d\_14 (Conv2D) (None, 95, 61, 64) 18496 ['max\_pooling2d\_13[0][0]']   
   
 max\_pooling2d\_14 (MaxPooling2D (None, 47, 30, 64) 0 ['conv2d\_14[0][0]']   
 )   
   
 conv2d\_15 (Conv2D) (None, 47, 30, 1) 65 ['max\_pooling2d\_14[0][0]']   
   
 multiply\_3 (Multiply) (None, 47, 30, 64) 0 ['max\_pooling2d\_14[0][0]',   
 'conv2d\_15[0][0]']   
   
 max\_pooling2d\_15 (MaxPooling2D (None, 23, 15, 64) 0 ['multiply\_3[0][0]']   
 )   
   
 flatten\_5 (Flatten) (None, 22080) 0 ['max\_pooling2d\_15[0][0]']   
   
 dense\_10 (Dense) (None, 32) 706592 ['flatten\_5[0][0]']   
   
 dense\_11 (Dense) (None, 10) 330 ['dense\_10[0][0]']   
   
==================================================================================================  
Total params: 725,836  
Trainable params: 725,836  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training

modelAtt.compile(optimizer=keras.optimizers.Adam(),  
 loss= keras.losses.CategoricalCrossentropy(),  
 metrics=['accuracy']  
 )

modelAtt.fit(x\_train,  
 epochs=epochs,  
 verbose=1,  
 validation\_data=x\_test  
 )

Epoch 1/4  
75/75 [==============================] - 5s 33ms/step - loss: 1.5404 - accuracy: 0.5317 - val\_loss: 1.1173 - val\_accuracy: 0.7000  
Epoch 2/4  
75/75 [==============================] - 2s 31ms/step - loss: 0.6520 - accuracy: 0.8275 - val\_loss: 1.2581 - val\_accuracy: 0.6967  
Epoch 3/4  
75/75 [==============================] - 2s 33ms/step - loss: 0.3326 - accuracy: 0.9000 - val\_loss: 1.0731 - val\_accuracy: 0.7567  
Epoch 4/4  
75/75 [==============================] - 2s 33ms/step - loss: 0.2562 - accuracy: 0.9342 - val\_loss: 1.0809 - val\_accuracy: 0.8433

<keras.callbacks.History at 0x7f95724d1970>

test\_loss, test\_acc = modelAtt.evaluate(x\_test)  
print('Test Accuracy = {:.2f} %:'.format(np.round(test\_acc, 3)\*100))

19/19 [==============================] - 0s 12ms/step - loss: 1.0809 - accuracy: 0.8433  
Test Accuracy = 84.30 %: