Import Libraries

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

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

!pip install tensorflow-io

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/  
Requirement already satisfied: tensorflow-io in /usr/local/lib/python3.9/dist-packages (0.32.0)  
Requirement already satisfied: tensorflow-io-gcs-filesystem==0.32.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow-io) (0.32.0)

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  
import time  
from tensorflow.keras.callbacks import EarlyStopping

# 1. Process Audio into Spectogram

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

Read all audio files and sort

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

num\_classes = 10  
iterations = 0  
i = 0  
train\_label = []  
while iterations!=len(train):  
 iterations +=1  
 train\_label.append(i)  
 i += 1  
 if i == num\_classes :  
 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 == num\_classes :  
 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

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[0:2],'\n...')

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

model=Sequential()  
input\_shape = (391, 257, 1)  
model.add( layers.Conv2D(32,kernel\_size=(3,3),activation='relu',input\_shape=input\_shape) )  
model.add( layers.MaxPooling2D(pool\_size=(2,2)) )  
model.add( layers.Dropout(0.2) )  
model.add( layers.Flatten() )  
model.add( layers.Dense(32,activation='relu') )  
model.add( layers.Dense(num\_classes,activation='softmax') )  
model.summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 conv2d (Conv2D) (None, 389, 255, 32) 320   
   
 max\_pooling2d (MaxPooling2D (None, 194, 127, 32) 0   
 )   
   
 dropout (Dropout) (None, 194, 127, 32) 0   
   
 flatten (Flatten) (None, 788416) 0   
   
 dense (Dense) (None, 32) 25229344   
   
 dense\_1 (Dense) (None, 10) 330   
   
=================================================================  
Total params: 25,229,994  
Trainable params: 25,229,994  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

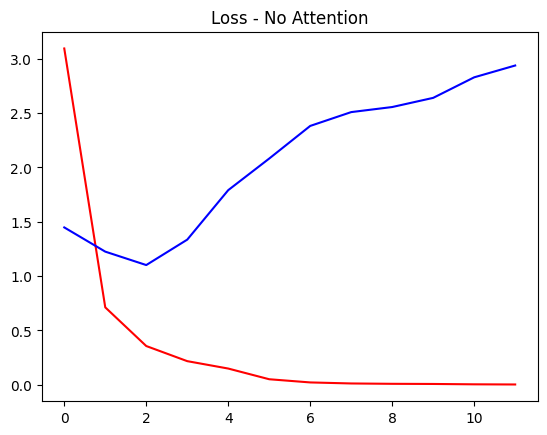
Training

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

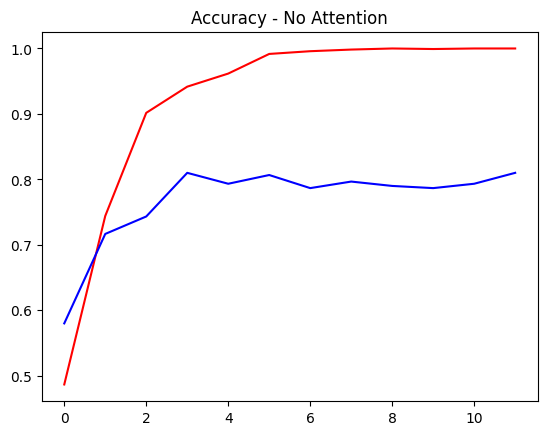
early\_stopping = EarlyStopping(monitor='accuracy', patience=3)  
tic=time.time()  
hist = model.fit(x\_train,  
 epochs=15,  
 verbose=1,  
 callbacks=[early\_stopping],  
 validation\_data=x\_test  
 )  
toc=time.time()  
training\_time=toc-tic

Epoch 1/15  
75/75 [==============================] - 21s 122ms/step - loss: 3.0945 - accuracy: 0.4867 - val\_loss: 1.4483 - val\_accuracy: 0.5800  
Epoch 2/15  
75/75 [==============================] - 5s 62ms/step - loss: 0.7128 - accuracy: 0.7442 - val\_loss: 1.2255 - val\_accuracy: 0.7167  
Epoch 3/15  
75/75 [==============================] - 5s 62ms/step - loss: 0.3571 - accuracy: 0.9017 - val\_loss: 1.1020 - val\_accuracy: 0.7433  
Epoch 4/15  
75/75 [==============================] - 5s 71ms/step - loss: 0.2179 - accuracy: 0.9417 - val\_loss: 1.3359 - val\_accuracy: 0.8100  
Epoch 5/15  
75/75 [==============================] - 4s 53ms/step - loss: 0.1498 - accuracy: 0.9617 - val\_loss: 1.7911 - val\_accuracy: 0.7933  
Epoch 6/15  
75/75 [==============================] - 4s 53ms/step - loss: 0.0515 - accuracy: 0.9917 - val\_loss: 2.0819 - val\_accuracy: 0.8067  
Epoch 7/15  
75/75 [==============================] - 4s 57ms/step - loss: 0.0221 - accuracy: 0.9958 - val\_loss: 2.3820 - val\_accuracy: 0.7867  
Epoch 8/15  
75/75 [==============================] - 4s 57ms/step - loss: 0.0126 - accuracy: 0.9983 - val\_loss: 2.5092 - val\_accuracy: 0.7967  
Epoch 9/15  
75/75 [==============================] - 4s 55ms/step - loss: 0.0094 - accuracy: 1.0000 - val\_loss: 2.5559 - val\_accuracy: 0.7900  
Epoch 10/15  
75/75 [==============================] - 4s 56ms/step - loss: 0.0080 - accuracy: 0.9992 - val\_loss: 2.6402 - val\_accuracy: 0.7867  
Epoch 11/15  
75/75 [==============================] - 5s 60ms/step - loss: 0.0048 - accuracy: 1.0000 - val\_loss: 2.8292 - val\_accuracy: 0.7933  
Epoch 12/15  
75/75 [==============================] - 4s 56ms/step - loss: 0.0033 - accuracy: 1.0000 - val\_loss: 2.9378 - val\_accuracy: 0.8100

plt.title('Loss - No Attention')  
plt.plot(hist.history['loss'], 'r')  
plt.plot(hist.history['val\_loss'], 'b')  
plt.show()



plt.title('Accuracy - No Attention')  
plt.plot(hist.history['accuracy'], 'r')  
plt.plot(hist.history['val\_accuracy'], 'b')  
plt.show()



tic=time.time()  
test\_loss, test\_acc = model.evaluate(x\_test)  
toc=time.time()  
test\_time=toc-tic  
print("Training Time = {} s".format(np.round(training\_time, 1)))  
print("Testing Time = {} ms".format(np.round(test\_time\*1000, 1)))  
print('Test Loss = {:.2f} %:'.format(np.round(test\_loss, 3)\*100))  
print('Test Accuracy = {:.2f} %:'.format(np.round(test\_acc, 3)\*100))

19/19 [==============================] - 0s 9ms/step - loss: 2.9378 - accuracy: 0.8100  
Training Time = 70.3 s  
Testing Time = 313.3 ms  
Test Loss = 293.80 %:  
Test Accuracy = 81.00 %:

# 3. CNN with Attention

Design the CNN architecture

inputs = layers.Input(shape=input\_shape)  
conv = layers.Conv2D(32,kernel\_size=(3,3),activation='relu')(inputs)  
#Attention  
attention = layers.Conv2D(1, (3,3), padding='same', activation='sigmoid')(conv)  
attention\_mul = layers.Multiply()([conv, attention])  
##########  
pool = layers.MaxPool2D(pool\_size=(2,2))(attention\_mul)  
drop = layers.Dropout(0.2)(pool)  
flatten = layers.Flatten()(drop)  
dense = layers.Dense(32,activation='relu')(flatten)  
dense2 = layers.Dense(num\_classes,activation='softmax')(dense)  
modelAtt = keras.Model(inputs=inputs, outputs=dense2)  
  
modelAtt.summary()

Model: "model"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param # Connected to   
==================================================================================================  
 input\_1 (InputLayer) [(None, 391, 257, 1 0 []   
 )]   
   
 conv2d\_1 (Conv2D) (None, 389, 255, 32 320 ['input\_1[0][0]']   
 )   
   
 conv2d\_2 (Conv2D) (None, 389, 255, 1) 289 ['conv2d\_1[0][0]']   
   
 multiply (Multiply) (None, 389, 255, 32 0 ['conv2d\_1[0][0]',   
 ) 'conv2d\_2[0][0]']   
   
 max\_pooling2d\_1 (MaxPooling2D) (None, 194, 127, 32 0 ['multiply[0][0]']   
 )   
   
 dropout\_1 (Dropout) (None, 194, 127, 32 0 ['max\_pooling2d\_1[0][0]']   
 )   
   
 flatten\_1 (Flatten) (None, 788416) 0 ['dropout\_1[0][0]']   
   
 dense\_2 (Dense) (None, 32) 25229344 ['flatten\_1[0][0]']   
   
 dense\_3 (Dense) (None, 10) 330 ['dense\_2[0][0]']   
   
==================================================================================================  
Total params: 25,230,283  
Trainable params: 25,230,283  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

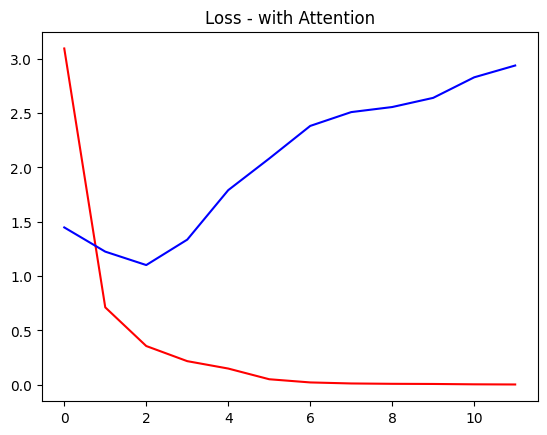
Training

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

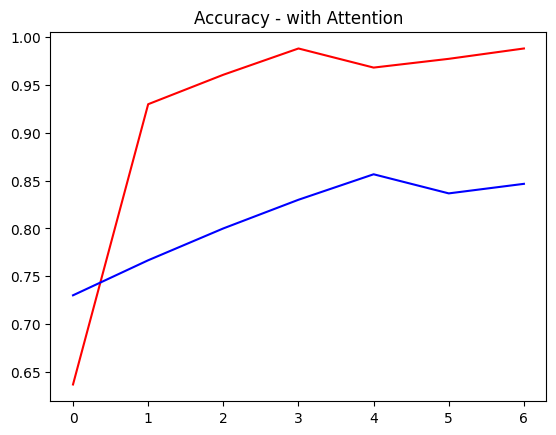
tic=time.time()  
histAtt = modelAtt.fit(x\_train,  
 epochs=15,  
 verbose=1,  
 callbacks=[early\_stopping],  
 validation\_data=x\_test  
 )  
toc=time.time()  
training\_time=toc-tic

Epoch 1/15  
75/75 [==============================] - 9s 89ms/step - loss: 1.4645 - accuracy: 0.6367 - val\_loss: 0.8348 - val\_accuracy: 0.7300  
Epoch 2/15  
75/75 [==============================] - 7s 87ms/step - loss: 0.2974 - accuracy: 0.9300 - val\_loss: 0.7497 - val\_accuracy: 0.7667  
Epoch 3/15  
75/75 [==============================] - 7s 87ms/step - loss: 0.1539 - accuracy: 0.9608 - val\_loss: 0.8659 - val\_accuracy: 0.8000  
Epoch 4/15  
75/75 [==============================] - 6s 87ms/step - loss: 0.0632 - accuracy: 0.9883 - val\_loss: 0.7461 - val\_accuracy: 0.8300  
Epoch 5/15  
75/75 [==============================] - 6s 85ms/step - loss: 0.1172 - accuracy: 0.9683 - val\_loss: 0.5103 - val\_accuracy: 0.8567  
Epoch 6/15  
75/75 [==============================] - 7s 87ms/step - loss: 0.1024 - accuracy: 0.9775 - val\_loss: 0.6533 - val\_accuracy: 0.8367  
Epoch 7/15  
75/75 [==============================] - 6s 85ms/step - loss: 0.0415 - accuracy: 0.9883 - val\_loss: 0.6919 - val\_accuracy: 0.8467

plt.title('Loss - with Attention')  
plt.plot(hist.history['loss'], 'r')  
plt.plot(hist.history['val\_loss'], 'b')  
plt.show()



plt.title('Accuracy - with Attention')  
plt.plot(histAtt.history['accuracy'], 'r')  
plt.plot(histAtt.history['val\_accuracy'], 'b')  
plt.show()



tic=time.time()  
test\_loss, test\_acc = modelAtt.evaluate(x\_test)  
toc=time.time()  
test\_time=toc-tic  
print("Training Time = {} s".format(np.round(training\_time, 1)))  
print("Testing Time = {} ms".format(np.round(test\_time\*1000, 1)))  
print('Test Loss = {:.2f} %:'.format(np.round(test\_loss, 3)\*100))  
print('Test Accuracy = {:.2f} %:'.format(np.round(test\_acc, 3)\*100))

19/19 [==============================] - 0s 19ms/step - loss: 0.6919 - accuracy: 0.8467  
Training Time = 53.6 s  
Testing Time = 637.8 ms  
Test Loss = 69.20 %:  
Test Accuracy = 84.70 %: