|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Academic Year: | 2022/2023 | Term: | Spring 2023 |  |
| Course Code: | ELC 4028 | Course Title: | Artificial Neural Networks and its Applications |

Cairo University

Faculty of Engineering

Electronics and Communications Engineering Department – 4th Year

Neural Networks Applications

- Assignment 3 -

*Submitted to: Dr. Mohsen Rashwan*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | BN | Sec | ID | رقم الجلوس |
| احمد محمود حسيني عطية | 22 | 1 | 9180178 | 34022 |
| علي ماهر عبدالسلام نبيه | 5 | 3 | 9190067 | 34117 |
| محمد احمد طه السيد | 30 | 3 | 9191043 | 34142 |
| محمد حسام عثمان يسن | 35 | 3 | 9191083 | 34147 |
| محمد عاطف ربيع | 43 | 3 | 9190924 | 34155 |

Table of Contents

[1. Problem 1 – MNIST dataset 1](#_Toc133367194)

[1.1. Code 1](#_Toc133367195)

[1.1.1 Process MNIST dataset 1](#_Toc133367196)

[1.1.2 CNN no Attention 2](#_Toc133367197)

[1.1.3 CNN with Attention 5](#_Toc133367198)

[1.2. Experiment Setup 8](#_Toc133367199)

[1.3. Comparison and Impact of Attention 9](#_Toc133367200)

[2. Problem 2 – Speech dataset 10](#_Toc133367201)

[2.1. Code 10](#_Toc133367202)

[2.1.1 Process Audio into Spectogram 10](#_Toc133367203)

[2.1.2 CNN no Attention 12](#_Toc133367204)

[2.1.3 CNN with Attention 15](#_Toc133367205)

[2.2. Experiment Setup 18](#_Toc133367206)

[2.3. Comparison and Impact of Attention 18](#_Toc133367207)

[3. Future Work 19](#_Toc133367208)

# 1. Problem 1 – MNIST dataset

## 1.1. Code

Import Libraries

!pip install tensorflow-io

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/  
Collecting tensorflow-io  
 Downloading tensorflow\_io-0.32.0-cp39-cp39-manylinux\_2\_12\_x86\_64.manylinux2010\_x86\_64.whl (28.0 MB)  
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 28.0/28.0 MB 30.1 MB/s eta 0:00:00  
ent already satisfied: tensorflow-io-gcs-filesystem==0.32.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow-io) (0.32.0)  
Installing collected packages: tensorflow-io  
Successfully installed tensorflow-io-0.32.0

import tensorflow as tf  
import numpy as np  
import matplotlib.pyplot as plt  
from tensorflow import keras  
from keras.datasets import mnist  
from keras import backend as k  
import time  
from tensorflow.keras.callbacks import EarlyStopping

### 1.1.1 Process MNIST dataset

Assign training and test data

batch\_size = 128  
num\_classes = 10  
img\_rows, img\_cols = 28,28  
(x\_train,y\_train),(x\_test,y\_test) = mnist.load\_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz  
11490434/11490434 [==============================] - 2s 0us/step

Reshape the images

if k.image\_data\_format()=='channels\_first':  
 x\_train=x\_train.reshape(x\_train.shape[0],img\_rows,img\_cols,1)  
 x\_test=x\_test.reshape(x\_test.shape[0],img\_rows,img\_cols,1)  
else:  
 x\_train=x\_train.reshape(x\_train.shape[0],img\_rows,img\_cols,1)  
 x\_test=x\_test.reshape(x\_test.shape[0],img\_rows,img\_cols,1)  
  
input\_shape=(img\_rows,img\_cols,1)  
x\_train = x\_train/255.0  
x\_test=x\_test/255.0  
print('x\_train shape:',x\_train.shape,'\nx\_test shape:',x\_test.shape)

x\_train shape: (60000, 28, 28, 1)   
x\_test shape: (10000, 28, 28, 1)

Convert class vectors to binary class matrices

y\_train=keras.utils.to\_categorical(y\_train,num\_classes)  
y\_test=keras.utils.to\_categorical(y\_test,num\_classes)

### 1.1.2 CNN no Attention

Design the CNN architecture

from keras.models import Sequential  
from keras import layers

model=Sequential()  
  
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, 26, 26, 32) 320   
   
 max\_pooling2d (MaxPooling2D (None, 13, 13, 32) 0   
 )   
   
 dropout (Dropout) (None, 13, 13, 32) 0   
   
 flatten (Flatten) (None, 5408) 0   
   
 dense (Dense) (None, 32) 173088   
   
 dense\_1 (Dense) (None, 10) 330   
   
=================================================================  
Total params: 173,738  
Trainable params: 173,738  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

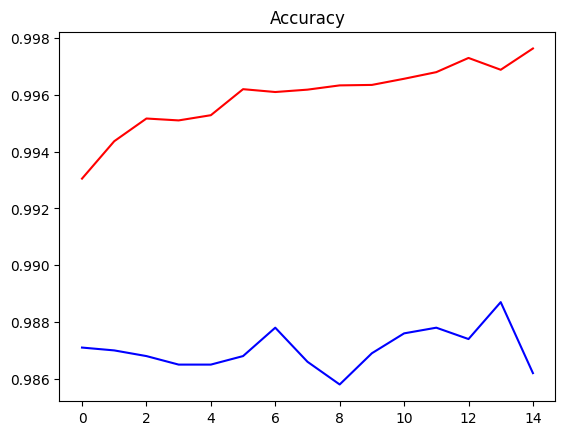
model.compile(optimizer=keras.optimizers.Adam(),  
 loss=keras.losses.categorical\_crossentropy,  
 metrics=['accuracy']  
 )  
early\_stopping = EarlyStopping(monitor='accuracy', patience=3)  
tic=time.time()  
hist = model.fit(x\_train,y\_train,  
 batch\_size=batch\_size,  
 epochs=15,  
 verbose=1,  
 callbacks=[early\_stopping],  
 validation\_data=(x\_test,y\_test)  
 )  
toc=time.time()  
training\_time=toc-tic

Epoch 1/15  
469/469 [==============================] - 4s 5ms/step - loss: 0.0208 - accuracy: 0.9930 - val\_loss: 0.0439 - val\_accuracy: 0.9871  
Epoch 2/15  
469/469 [==============================] - 2s 5ms/step - loss: 0.0180 - accuracy: 0.9944 - val\_loss: 0.0466 - val\_accuracy: 0.9870  
Epoch 3/15  
469/469 [==============================] - 2s 5ms/step - loss: 0.0157 - accuracy: 0.9952 - val\_loss: 0.0450 - val\_accuracy: 0.9868  
Epoch 4/15  
469/469 [==============================] - 2s 4ms/step - loss: 0.0143 - accuracy: 0.9951 - val\_loss: 0.0479 - val\_accuracy: 0.9865  
Epoch 5/15  
469/469 [==============================] - 3s 5ms/step - loss: 0.0137 - accuracy: 0.9953 - val\_loss: 0.0479 - val\_accuracy: 0.9865  
Epoch 6/15  
469/469 [==============================] - 2s 5ms/step - loss: 0.0119 - accuracy: 0.9962 - val\_loss: 0.0498 - val\_accuracy: 0.9868  
Epoch 7/15  
469/469 [==============================] - 3s 6ms/step - loss: 0.0114 - accuracy: 0.9961 - val\_loss: 0.0477 - val\_accuracy: 0.9878  
Epoch 8/15  
469/469 [==============================] - 3s 7ms/step - loss: 0.0114 - accuracy: 0.9962 - val\_loss: 0.0474 - val\_accuracy: 0.9866  
Epoch 9/15  
469/469 [==============================] - 4s 9ms/step - loss: 0.0109 - accuracy: 0.9963 - val\_loss: 0.0503 - val\_accuracy: 0.9858  
Epoch 10/15  
469/469 [==============================] - 3s 7ms/step - loss: 0.0097 - accuracy: 0.9963 - val\_loss: 0.0498 - val\_accuracy: 0.9869  
Epoch 11/15  
469/469 [==============================] - 3s 7ms/step - loss: 0.0098 - accuracy: 0.9966 - val\_loss: 0.0501 - val\_accuracy: 0.9876  
Epoch 12/15  
469/469 [==============================] - 2s 5ms/step - loss: 0.0090 - accuracy: 0.9968 - val\_loss: 0.0487 - val\_accuracy: 0.9878  
Epoch 13/15  
469/469 [==============================] - 2s 5ms/step - loss: 0.0080 - accuracy: 0.9973 - val\_loss: 0.0546 - val\_accuracy: 0.9874  
Epoch 14/15  
469/469 [==============================] - 2s 5ms/step - loss: 0.0083 - accuracy: 0.9969 - val\_loss: 0.0439 - val\_accuracy: 0.9887  
Epoch 15/15  
469/469 [==============================] - 2s 5ms/step - loss: 0.0076 - accuracy: 0.9976 - val\_loss: 0.0571 - val\_accuracy: 0.9862

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



plt.title('Accuracy')  
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,y\_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))

313/313 [==============================] - 1s 2ms/step - loss: 0.0571 - accuracy: 0.9862  
Training Time = 83.3 s  
Testing Time = 901.9 ms  
Test Loss = 5.70 %:  
Test Accuracy = 98.60 %:

### 1.1.3 CNN with Attention

Design the CNN architecture

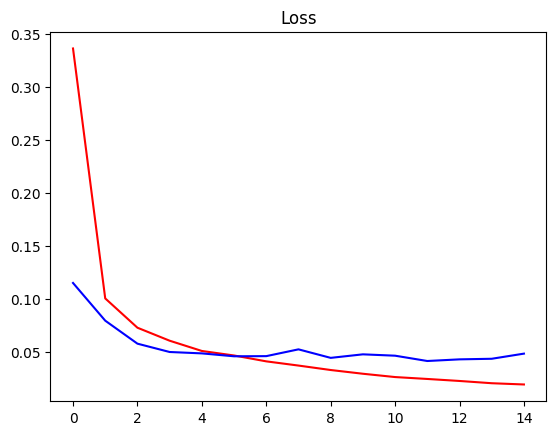
inputs = layers.Input(shape=input\_shape)  
conv = layers.Conv2D(32,kernel\_size=(3,3),activation='relu')(inputs)  
#Attention  
attention = layers.Conv2D(1, (3,3), padding='same', activation='sigmoid')(conv)  
attention\_mul = layers.Multiply()([conv, attention])  
##########  
pool = layers.MaxPool2D(pool\_size=(2,2))(attention\_mul)  
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, 28, 28, 1)] 0 []   
   
 conv2d\_1 (Conv2D) (None, 26, 26, 32) 320 ['input\_1[0][0]']   
   
 conv2d\_2 (Conv2D) (None, 26, 26, 1) 289 ['conv2d\_1[0][0]']   
   
 multiply (Multiply) (None, 26, 26, 32) 0 ['conv2d\_1[0][0]',   
 'conv2d\_2[0][0]']   
   
 max\_pooling2d\_1 (MaxPooling2D) (None, 13, 13, 32) 0 ['multiply[0][0]']   
   
 dropout\_1 (Dropout) (None, 13, 13, 32) 0 ['max\_pooling2d\_1[0][0]']   
   
 flatten\_1 (Flatten) (None, 5408) 0 ['dropout\_1[0][0]']   
   
 dense\_2 (Dense) (None, 32) 173088 ['flatten\_1[0][0]']   
   
 dense\_3 (Dense) (None, 10) 330 ['dense\_2[0][0]']   
   
=========================================================================================  
Total params: 174,027  
Trainable params: 174,027  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

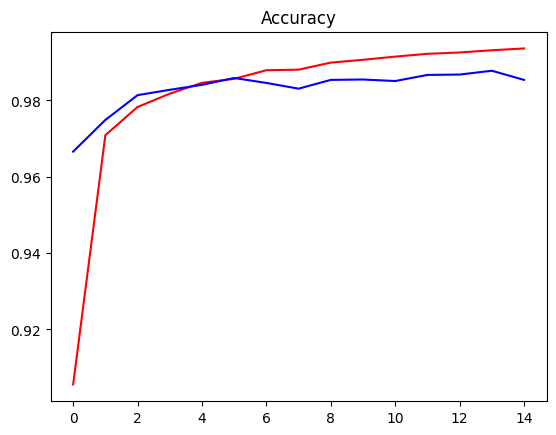
modelAtt.compile(optimizer=keras.optimizers.Adam(),  
 loss= keras.losses.CategoricalCrossentropy(),  
 metrics=['accuracy']  
 )  
tic=time.time()  
hist = modelAtt.fit(x\_train,y\_train,  
 batch\_size=batch\_size,  
 epochs=15,  
 verbose=1,  
 callbacks=[early\_stopping],  
 validation\_data=(x\_test,y\_test)  
 )  
toc=time.time()  
training\_time=toc-tic

Epoch 1/15  
469/469 [==============================] - 5s 7ms/step - loss: 0.3364 - accuracy: 0.9056 - val\_loss: 0.1149 - val\_accuracy: 0.9666  
Epoch 2/15  
469/469 [==============================] - 3s 6ms/step - loss: 0.1003 - accuracy: 0.9709 - val\_loss: 0.0792 - val\_accuracy: 0.9749  
Epoch 3/15  
469/469 [==============================] - 4s 8ms/step - loss: 0.0726 - accuracy: 0.9783 - val\_loss: 0.0575 - val\_accuracy: 0.9814  
Epoch 4/15  
469/469 [==============================] - 4s 9ms/step - loss: 0.0603 - accuracy: 0.9818 - val\_loss: 0.0496 - val\_accuracy: 0.9828  
Epoch 5/15  
469/469 [==============================] - 4s 9ms/step - loss: 0.0506 - accuracy: 0.9846 - val\_loss: 0.0483 - val\_accuracy: 0.9841  
Epoch 6/15  
469/469 [==============================] - 5s 11ms/step - loss: 0.0463 - accuracy: 0.9857 - val\_loss: 0.0456 - val\_accuracy: 0.9859  
Epoch 7/15  
469/469 [==============================] - 4s 9ms/step - loss: 0.0408 - accuracy: 0.9879 - val\_loss: 0.0457 - val\_accuracy: 0.9846  
Epoch 8/15  
469/469 [==============================] - 3s 6ms/step - loss: 0.0368 - accuracy: 0.9881 - val\_loss: 0.0521 - val\_accuracy: 0.9831  
Epoch 9/15  
469/469 [==============================] - 3s 6ms/step - loss: 0.0326 - accuracy: 0.9899 - val\_loss: 0.0441 - val\_accuracy: 0.9854  
Epoch 10/15  
469/469 [==============================] - 3s 7ms/step - loss: 0.0291 - accuracy: 0.9907 - val\_loss: 0.0474 - val\_accuracy: 0.9855  
Epoch 11/15  
469/469 [==============================] - 3s 6ms/step - loss: 0.0260 - accuracy: 0.9915 - val\_loss: 0.0461 - val\_accuracy: 0.9851  
Epoch 12/15  
469/469 [==============================] - 3s 6ms/step - loss: 0.0242 - accuracy: 0.9922 - val\_loss: 0.0411 - val\_accuracy: 0.9867  
Epoch 13/15  
469/469 [==============================] - 3s 6ms/step - loss: 0.0223 - accuracy: 0.9926 - val\_loss: 0.0427 - val\_accuracy: 0.9868  
Epoch 14/15  
469/469 [==============================] - 3s 7ms/step - loss: 0.0201 - accuracy: 0.9932 - val\_loss: 0.0432 - val\_accuracy: 0.9878  
Epoch 15/15  
469/469 [==============================] - 3s 6ms/step - loss: 0.0189 - accuracy: 0.9936 - val\_loss: 0.0481 - val\_accuracy: 0.9854

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



plt.title('Accuracy')  
plt.plot(hist.history['accuracy'], 'r')  
plt.plot(hist.history['val\_accuracy'], 'b')  
plt.show()



tic=time.time()  
test\_loss, test\_acc = modelAtt.evaluate(x\_test,y\_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))

313/313 [==============================] - 1s 3ms/step - loss: 0.0481 - accuracy: 0.9854  
Training Time = 52.4 s  
Testing Time = 921.8 ms  
Test Loss = 4.80 %:  
Test Accuracy = 98.50 %:

## 1.2. Experiment Setup

Picked a simple CNN architecture and only added the attention mechanism to the same network.



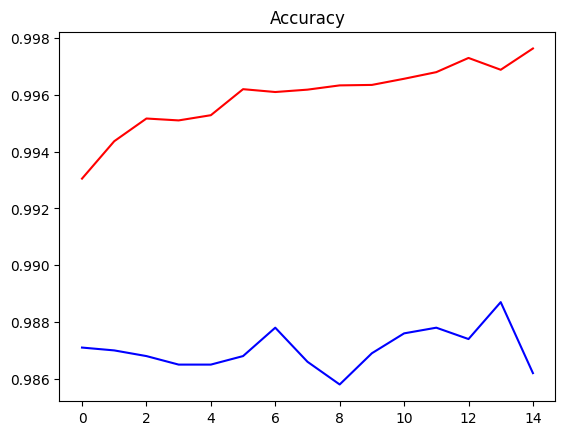
First is a convolution layer with 32 filters, kernel size 3x3 and the activation function is relu. A maxpooling layer to squeeze some information then a dropout with a parameter 20% to prevent overfitting. Then a fully connected layer.



For the attention, the mechanism chosen is adding a convolution layer with 1 filter and sigmoid activation function then multiply it by the previous conv layer. It is the simplest form of attention.

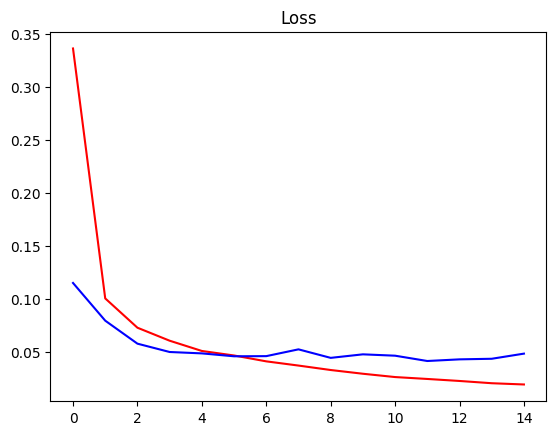
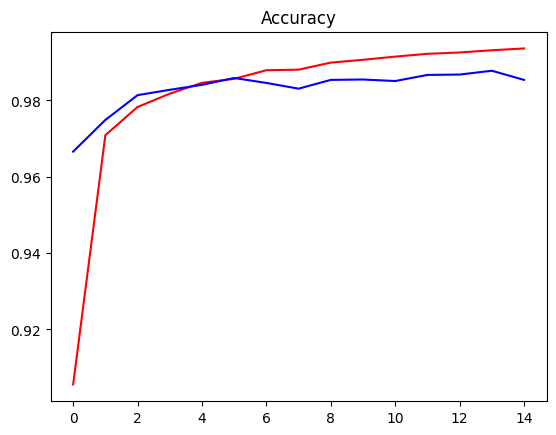
## 1.3. Comparison and Impact of Attention

Results without attention:



In the MNIST dataset it is already very simple and a simple CNN can achieve good results. The best test accuracy achieved is 98.87% after 13 epochs. And least loss is 4.39%

Results with attention:



We notice that the 2 curves become closer than without attention without really affecting the general results but it looks more accurate.

The best test accuracy achieved is 98.78% after 15 epochs. And least loss is 4.11% but it becomes nearly stable after 5 epochs.

# 2. Problem 2 – Speech dataset

## 2.1. Code

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

### 2.1.1 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.]]   
...

### 2.1.2 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  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

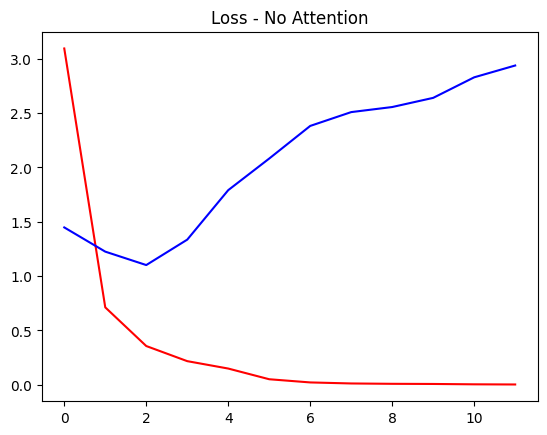
Training

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

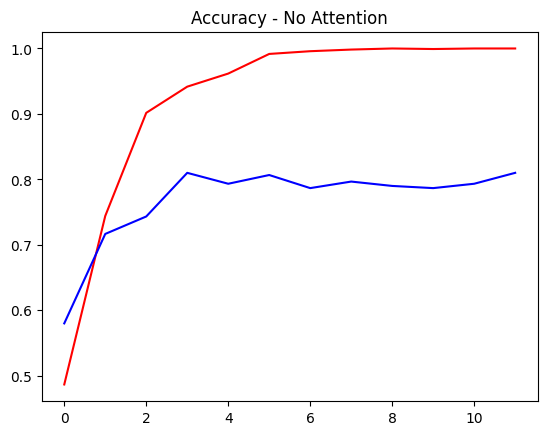
early\_stopping = EarlyStopping(monitor='accuracy', patience=3)  
tic=time.time()  
hist = model.fit(x\_train,  
 epochs=15,  
 verbose=1,  
 callbacks=[early\_stopping],  
 validation\_data=x\_test  
 )  
toc=time.time()  
training\_time=toc-tic

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 %:

### 2.1.3 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  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

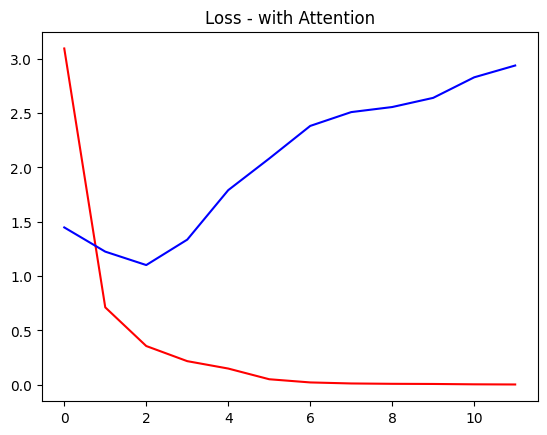
Training

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

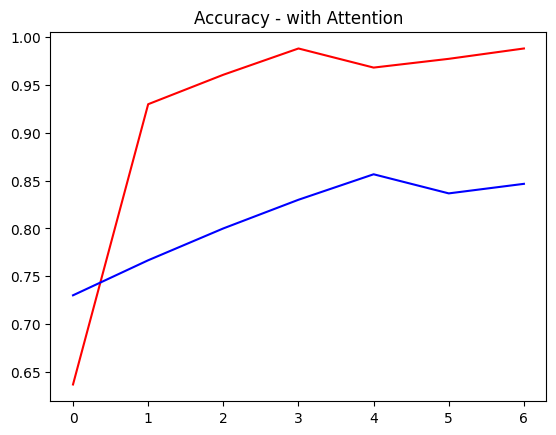
tic=time.time()  
histAtt = modelAtt.fit(x\_train,  
 epochs=15,  
 verbose=1,  
 callbacks=[early\_stopping],  
 validation\_data=x\_test  
 )  
toc=time.time()  
training\_time=toc-tic

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))

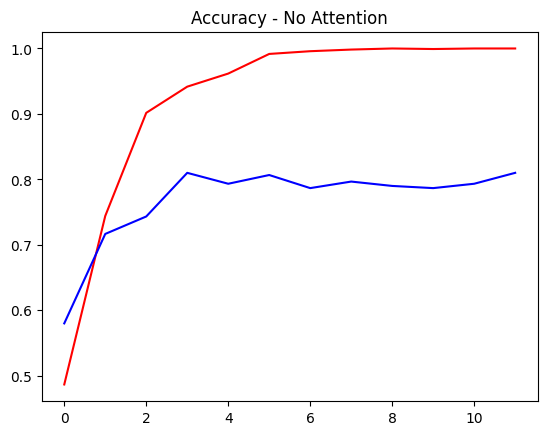
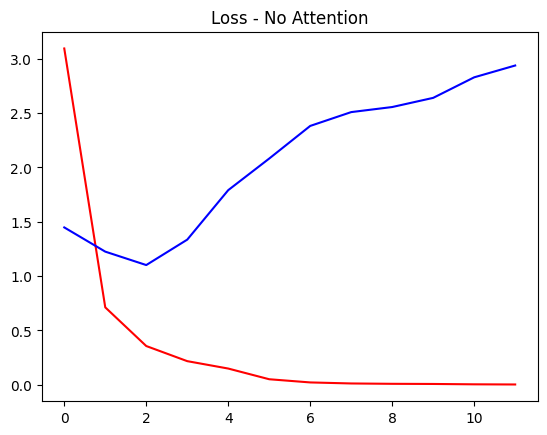
19/19 [==============================] - 0s 19ms/step - loss: 0.6919 - accuracy: 0.8467  
Training Time = 53.6 s  
Testing Time = 637.8 ms  
Test Loss = 69.20 %:  
Test Accuracy = 84.70 %:

## 2.2. Experiment Setup

The setup is the same as the MNIST dataset, but here the speech dataset had to be converted to a spectrogram first to change the problem to an image binary classification problem. Because the dimensions of the spectrogram is larger, the number of parameters of the neural networks increased significantly.

## 2.3. Comparison and Impact of Attention

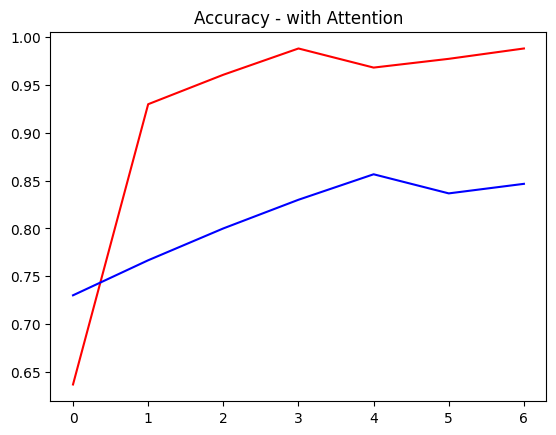
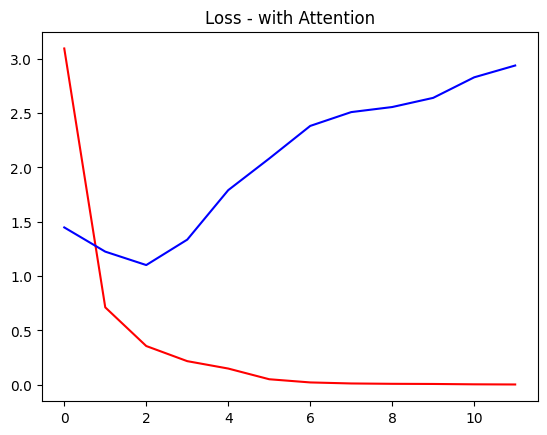
Results without attention:



In this dataset, probably the network architecture chosen isn’t the best to deal with it so the difference between the training and test results are very far.

For the best test accuracy: 81% at 15 epochs and minimum loss is 110.20%

Results with attention:



A very slight change that the blue curve was a little closer.

For the best test accuracy: 85.67% at 6 epochs and minimum loss is 51.03%

# 3. Future Work

First is using a well-known CNN architecture like LeNet-5, VGG, ResNet and insert the attention mechanism inside it to see the different in results.

We noticed that the network can get overfitted on training data which results in a bad training to test ratio, so might try to use other blocks that reduce less parameters.

Performing hyper parameters variations to know what is the best possible network for each of the 2 datasets. By doing a grid over activation functions, number of epochs, batch sizes and optimizer.

Implementing a better attention mechanism by implementing our own Attention class for the layer and perform the query, key, content and softmax method.