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. 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)

# 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  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

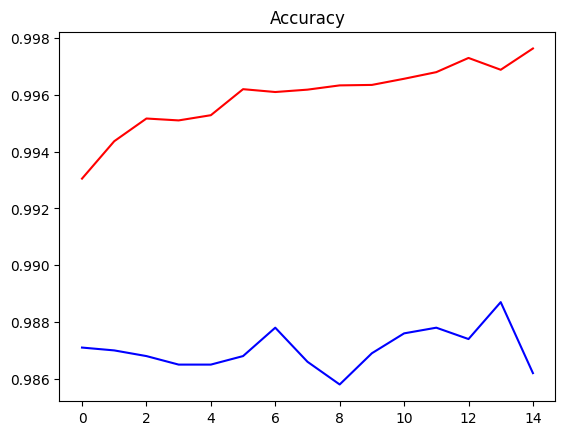
model.compile(optimizer=keras.optimizers.Adam(),  
 loss=keras.losses.categorical\_crossentropy,  
 metrics=['accuracy']  
 )  
early\_stopping = EarlyStopping(monitor='accuracy', patience=3)  
tic=time.time()  
hist = model.fit(x\_train,y\_train,  
 batch\_size=batch\_size,  
 epochs=15,  
 verbose=1,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

# 3. CNN with Attention

import torch.nn as nn

Design the CNN architecture

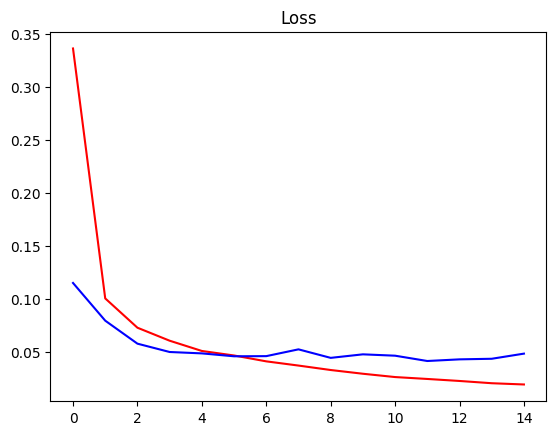
inputs = layers.Input(shape=input\_shape)  
conv = layers.Conv2D(32,kernel\_size=(3,3),activation='relu')(inputs)  
#Attention  
attention = layers.Conv2D(1, (3,3), padding='same', activation='sigmoid')(conv)  
attention\_mul = layers.Multiply()([conv, attention])  
##########  
pool = layers.MaxPool2D(pool\_size=(2,2))(attention\_mul)  
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  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

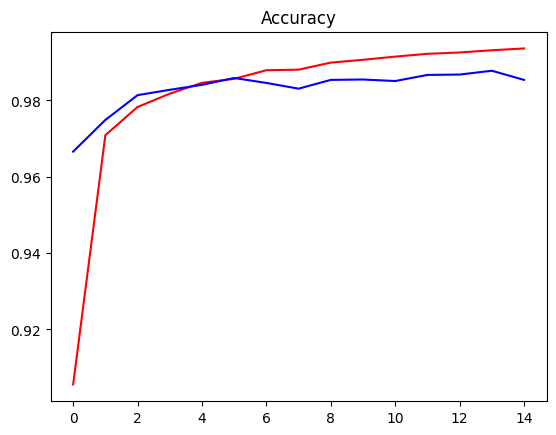
modelAtt.compile(optimizer=keras.optimizers.Adam(),  
 loss= keras.losses.CategoricalCrossentropy(),  
 metrics=['accuracy']  
 )  
tic=time.time()  
hist = modelAtt.fit(x\_train,y\_train,  
 batch\_size=batch\_size,  
 epochs=15,  
 verbose=1,  
 callbacks=[early\_stopping],  
 validation\_data=(x\_test,y\_test)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