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 54.6 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

# 1. Process MNIST dataset

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 = 128  
num\_classes = 10  
epochs = 4

Assign training and test data

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 [==============================] - 0s 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.layers import Dense,Flatten,Input  
from keras.layers import Conv2D,MaxPool2D,Multiply

model=Sequential()  
  
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"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 conv2d (Conv2D) (None, 26, 26, 32) 320   
   
 max\_pooling2d (MaxPooling2D (None, 13, 13, 32) 0   
 )   
   
 conv2d\_1 (Conv2D) (None, 11, 11, 64) 18496   
   
 max\_pooling2d\_1 (MaxPooling (None, 5, 5, 64) 0   
 2D)   
   
 flatten (Flatten) (None, 1600) 0   
   
 dense (Dense) (None, 32) 51232   
   
 dense\_1 (Dense) (None, 10) 330   
   
=================================================================  
Total params: 70,378  
Trainable params: 70,378  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

model.compile(optimizer=keras.optimizers.Adam(),  
 loss=keras.losses.categorical\_crossentropy,  
 metrics=['accuracy']  
 )  
model.fit(x\_train,y\_train,  
 batch\_size=batch\_size,  
 epochs=epochs,  
 verbose=1,  
 validation\_data=(x\_test,y\_test)  
 )

Epoch 1/4  
469/469 [==============================] - 21s 6ms/step - loss: 0.2862 - accuracy: 0.9122 - val\_loss: 0.0758 - val\_accuracy: 0.9765  
Epoch 2/4  
469/469 [==============================] - 2s 5ms/step - loss: 0.0760 - accuracy: 0.9772 - val\_loss: 0.0603 - val\_accuracy: 0.9807  
Epoch 3/4  
469/469 [==============================] - 2s 5ms/step - loss: 0.0539 - accuracy: 0.9830 - val\_loss: 0.0446 - val\_accuracy: 0.9864  
Epoch 4/4  
469/469 [==============================] - 2s 5ms/step - loss: 0.0423 - accuracy: 0.9869 - val\_loss: 0.0386 - val\_accuracy: 0.9878

<keras.callbacks.History at 0x7f4023c0d430>

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

313/313 [==============================] - 1s 2ms/step - loss: 0.0386 - accuracy: 0.9878  
Test Accuracy = 98.80 %:

# 3. CNN with Attention

Design the CNN architecture

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"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param # Connected to   
==================================================================================================  
 input\_1 (InputLayer) [(None, 28, 28, 1)] 0 []   
   
 conv2d\_2 (Conv2D) (None, 26, 26, 32) 320 ['input\_1[0][0]']   
   
 max\_pooling2d\_2 (MaxPooling2D) (None, 13, 13, 32) 0 ['conv2d\_2[0][0]']   
   
 conv2d\_3 (Conv2D) (None, 13, 13, 1) 33 ['max\_pooling2d\_2[0][0]']   
   
 multiply (Multiply) (None, 13, 13, 32) 0 ['max\_pooling2d\_2[0][0]',   
 'conv2d\_3[0][0]']   
   
 max\_pooling2d\_3 (MaxPooling2D) (None, 6, 6, 32) 0 ['multiply[0][0]']   
   
 conv2d\_4 (Conv2D) (None, 4, 4, 64) 18496 ['max\_pooling2d\_3[0][0]']   
   
 max\_pooling2d\_4 (MaxPooling2D) (None, 2, 2, 64) 0 ['conv2d\_4[0][0]']   
   
 conv2d\_5 (Conv2D) (None, 2, 2, 1) 65 ['max\_pooling2d\_4[0][0]']   
   
 multiply\_1 (Multiply) (None, 2, 2, 64) 0 ['max\_pooling2d\_4[0][0]',   
 'conv2d\_5[0][0]']   
   
 max\_pooling2d\_5 (MaxPooling2D) (None, 1, 1, 64) 0 ['multiply\_1[0][0]']   
   
 flatten\_1 (Flatten) (None, 64) 0 ['max\_pooling2d\_5[0][0]']   
   
 dense\_2 (Dense) (None, 32) 2080 ['flatten\_1[0][0]']   
   
 dense\_3 (Dense) (None, 10) 330 ['dense\_2[0][0]']   
   
==================================================================================================  
Total params: 21,324  
Trainable params: 21,324  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

modelAtt.compile(optimizer=keras.optimizers.Adam(),  
 loss= keras.losses.CategoricalCrossentropy(),  
 metrics=['accuracy']  
 )  
modelAtt.fit(x\_train,y\_train,  
 batch\_size=batch\_size,  
 epochs=epochs,  
 verbose=1,  
 validation\_data=(x\_test,y\_test)  
 )

Epoch 1/4  
469/469 [==============================] - 5s 6ms/step - loss: 0.6068 - accuracy: 0.8218 - val\_loss: 0.1601 - val\_accuracy: 0.9534  
Epoch 2/4  
469/469 [==============================] - 2s 5ms/step - loss: 0.1413 - accuracy: 0.9588 - val\_loss: 0.1090 - val\_accuracy: 0.9685  
Epoch 3/4  
469/469 [==============================] - 5s 10ms/step - loss: 0.1012 - accuracy: 0.9698 - val\_loss: 0.0805 - val\_accuracy: 0.9745  
Epoch 4/4  
469/469 [==============================] - 4s 9ms/step - loss: 0.0813 - accuracy: 0.9756 - val\_loss: 0.0698 - val\_accuracy: 0.9780

<keras.callbacks.History at 0x7f402375f1f0>

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

313/313 [==============================] - 2s 6ms/step - loss: 0.0698 - accuracy: 0.9780  
Test Accuracy = 97.80 %: