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
Requirement already satisfied: tensorflow-io-gcs-filesystem==0.32.0 in /usr/local/lib/python3.9/dist-packages (from te
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_acc
Epoch 2/4
469/469 [==============================] - 2s 5ms/step - loss: 0.0760 - accuracy: 0.9772 - val_loss: 0.0603 - val_accu
Epoch 3/4
469/469 [==============================] - 2s 5ms/step - loss: 0.0539 - accuracy: 0.9830 - val_loss: 0.0446 - val_accu
```

```
Epoch 4/4
469/469 [==============================] - 2s 5ms/step - loss: 0.0423 - accuracy: 0.9869 - val_loss: 0.0386 - val_accu
<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_accu
Epoch 2/4
469/469 [==============================] - 2s 5ms/step - loss: 0.1413 - accuracy: 0.9588 - val_loss: 0.1090 - val_accu
Epoch 3/4
469/469 [==============================] - 5s 10ms/step - loss: 0.1012 - accuracy: 0.9698 - val_loss: 0.0805 - val_acc
Epoch 4/4
469/469 [==============================] - 4s 9ms/step - loss: 0.0813 - accuracy: 0.9756 - val_loss: 0.0698 - val_accu
<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 %:
```

✓ 0s completed at 5:02 PM