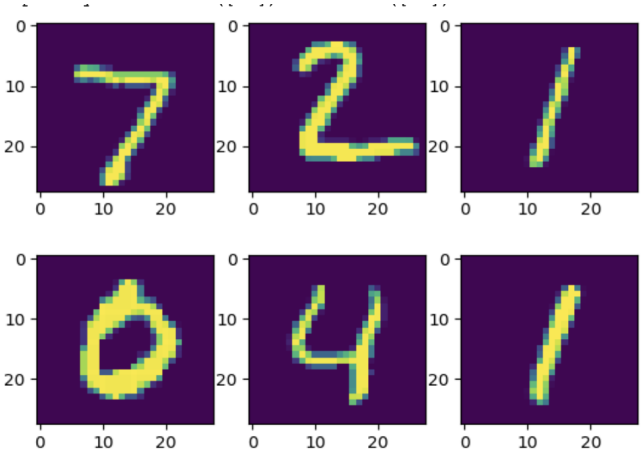
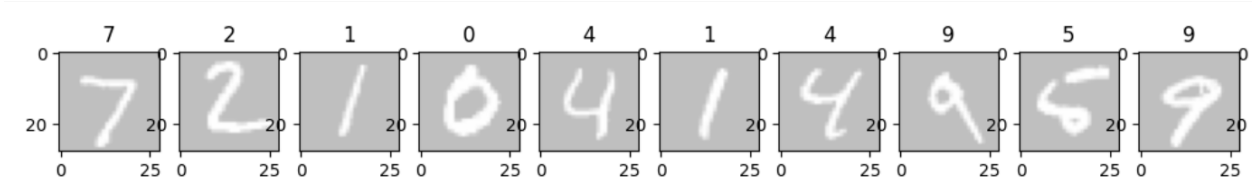
Computer Vision Assignment 4
M22MA002

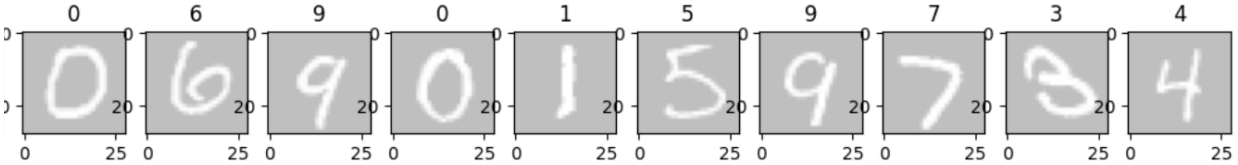
Q1. Perform image classification using CNN on the MNIST dataset. Follow the standard train and test split. Design an 8-layer CNN network (choose your architecture, e.g., filter size, number of channels, padding, activations, etc.).

Solution 1:-

- Import necessary libraries such as PyTorch, NumPy, Matplotlib, and torchvision.
- Define the device on which to run the code based on whether a CUDA-compatible GPU is available or not.
- Define two sets of image transformations using torchvision.transforms.Compose, one for training data and one for testing data.
- Load the MNIST dataset using torchvision.datasets.MNIST and apply the training and testing transformations to them using torch.utils.data.Subset.
- Define the data loaders using torch.utils.data.DataLoader, one for the training subset and one for the testing subset.
- Visualize some sample images from the testing subset using matplotlib.pyplot.imshow.







- Define a Convolutional Neural Network (CNN) using the nn.Module class. The CNN architecture includes two convolutional layers with ReLU activation functions, three max pooling layers, and two fully connected (linear) layers.

```
-
Net(
  (conv1): Conv2d(3, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(5, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(5, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(5, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(5, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv6): Conv2d(5, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv7): Conv2d(5, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv8): Conv2d(5, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (flat): Flatten(start_dim=1, end_dim=-1)
  (fc): Linear(in_features=45, out_features=10, bias=True)
)
```

1. Show the calculation of output filter size at each layer of CNN.

Using Formula : **Output_Size = (input_size - kernel_size + 2 * padding) / stride + 1**

Layer	**Input Size**	**Kernel Size**	**Stride**	**Padding**	**Output Size||**(applying formula:)
conv1	3x28x28	3x3	1	1	5x28x28 ||((28+2*1-3)/1)+1 = 28
conv2	5x28x28	3x3	1	1	5x28x28 ||((28+2*1-3)/1)+1 = 28
conv3	5x28x28	3x3	1	1	5x28x28 ||((28+2*1-3)/1)+1 = 28
maxpool	5x28x28	2x2	2	0	5x14x14 || ((28-2)/2)+1 = 14)
conv4	5x14x14	3x3	1	1	5x14x14 || ((14+2*1-3)/1)+1 = 14)
conv5	5x14x14	3x3	1	1	5x14x14 || ((14+2*1-3)/1)+1 = 14)
conv6	5x14x14	3x3	1	1	5x14x14 || ((14+2*1-3)/1)+1 = 14)
maxpool	5x14x14	2x2	2	0	5x7x7 || ((14-2)/2)+1 = 7)
conv7	5x7x7	3x3	1	1	5x7x7 || ((7+2*1-3)/1)+1 = 7)
conv8	5x7x7	3x3	1	1	5x7x7 || ((7+2*1-3)/1)+1 = 7)
maxpool	5x7x7	2x2	2	0	5x3x3 || ((7-2)/2)+1 = 3)
fc	5x3x3	-	-	-	10

- Define the loss function and the optimizer to be used during training. In this case, we use cross-entropy loss and stochastic gradient descent (SGD) optimizer.
- Train the model by iterating over each batch of training data and updating the model parameters based on the loss and optimizer.

- Evaluate the trained model on the testing subset by iterating over each batch of testing data and calculating the accuracy.

```
Epoch 1
-------------------------------
loss: 2.302793  [    0/60000]
loss: 2.299656  [ 6400/60000]
loss: 1.705746  [12800/60000]
loss: 1.735484  [19200/60000]
loss: 1.786470  [25600/60000]
loss: 1.564254  [32000/60000]
loss: 1.648763  [38400/60000]
loss: 1.645717  [44800/60000]
loss: 1.602869  [51200/60000]
loss: 1.609998  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 83.6%, Avg loss: 1.625750

Epoch 2
-------------------------------
loss: 1.610955  [    0/60000]
loss: 1.581167  [ 6400/60000]
loss: 1.679350  [12800/60000]
loss: 1.580261  [19200/60000]
loss: 1.665794  [25600/60000]
loss: 1.598458  [32000/60000]
loss: 1.613833  [38400/60000]
loss: 1.567804  [44800/60000]
loss: 1.600208  [51200/60000]
loss: 1.641178  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 89.4%, Avg loss: 1.567487

Epoch 3
-------------------------------
loss: 1.549946  [    0/60000]
loss: 1.546453  [ 6400/60000]
loss: 1.570320  [12800/60000]
loss: 1.520523  [19200/60000]
loss: 1.518022  [25600/60000]
loss: 1.505993  [32000/60000]
loss: 1.556441  [38400/60000]
loss: 1.490257  [44800/60000]
loss: 1.466957  [51200/60000]
loss: 1.511167  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 95.0%, Avg loss: 1.510825

Epoch 4
-------------------------------
loss: 1.513944  [    0/60000]
loss: 1.498846  [ 6400/60000]
loss: 1.593280  [12800/60000]
loss: 1.506817  [19200/60000]
loss: 1.536480  [25600/60000]
loss: 1.510194  [32000/60000]
loss: 1.523901  [38400/60000]
loss: 1.490133  [44800/60000]
loss: 1.498987  [51200/60000]
loss: 1.510159  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 95.8%, Avg loss: 1.504182

Epoch 5
-------------------------------
loss: 1.571882  [    0/60000]
loss: 1.513839  [ 6400/60000]
loss: 1.524679  [12800/60000]
loss: 1.513331  [19200/60000]
loss: 1.512370  [25600/60000]
loss: 1.494721  [32000/60000]
loss: 1.568997  [38400/60000]
loss: 1.526455  [44800/60000]
loss: 1.570550  [51200/60000]
loss: 1.542877  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 94.4%, Avg loss: 1.516549

Epoch 6
-------------------------------
loss: 1.506485  [    0/60000]
loss: 1.539337  [ 6400/60000]
loss: 1.508126  [12800/60000]
loss: 1.509050  [19200/60000]
loss: 1.495704  [25600/60000]
loss: 1.482530  [32000/60000]
loss: 1.540249  [38400/60000]
loss: 1.486751  [44800/60000]
loss: 1.510616  [51200/60000]
loss: 1.508262  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 96.0%, Avg loss: 1.500635
```

```
Epoch 7
-------------------------------
loss: 1.492411  [    0/60000]
loss: 1.478434  [ 6400/60000]
loss: 1.501948  [12800/60000]
loss: 1.508249  [19200/60000]
loss: 1.556912  [25600/60000]
loss: 1.470207  [32000/60000]
loss: 1.534584  [38400/60000]
loss: 1.470834  [44800/60000]
loss: 1.501104  [51200/60000]
loss: 1.476978  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 96.3%, Avg loss: 1.497113

Epoch 8
-------------------------------
loss: 1.502274  [    0/60000]
loss: 1.461270  [ 6400/60000]
loss: 1.461929  [12800/60000]
loss: 1.492575  [19200/60000]
loss: 1.477353  [25600/60000]
loss: 1.508714  [32000/60000]
loss: 1.492497  [38400/60000]
loss: 1.496876  [44800/60000]
loss: 1.557275  [51200/60000]
loss: 1.507560  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 96.4%, Avg loss: 1.496990

Epoch 9
-------------------------------
loss: 1.491760  [    0/60000]
loss: 1.480591  [ 6400/60000]
loss: 1.491834  [12800/60000]
loss: 1.497621  [19200/60000]
loss: 1.461173  [25600/60000]
loss: 1.461156  [32000/60000]
loss: 1.476774  [38400/60000]
loss: 1.522402  [44800/60000]
loss: 1.476781  [51200/60000]
loss: 1.492340  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 95.9%, Avg loss: 1.501446

Epoch 10
-------------------------------
loss: 1.504224  [    0/60000]
loss: 1.461171  [ 6400/60000]
loss: 1.492596  [12800/60000]
loss: 1.481747  [19200/60000]
loss: 1.495871  [25600/60000]
loss: 1.503958  [32000/60000]
loss: 1.498256  [38400/60000]
loss: 1.492608  [44800/60000]
loss: 1.507772  [51200/60000]
loss: 1.510793  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 95.7%, Avg loss: 1.504147
```

2. Calculate the number of parameters in your CNN. Model Summary:-

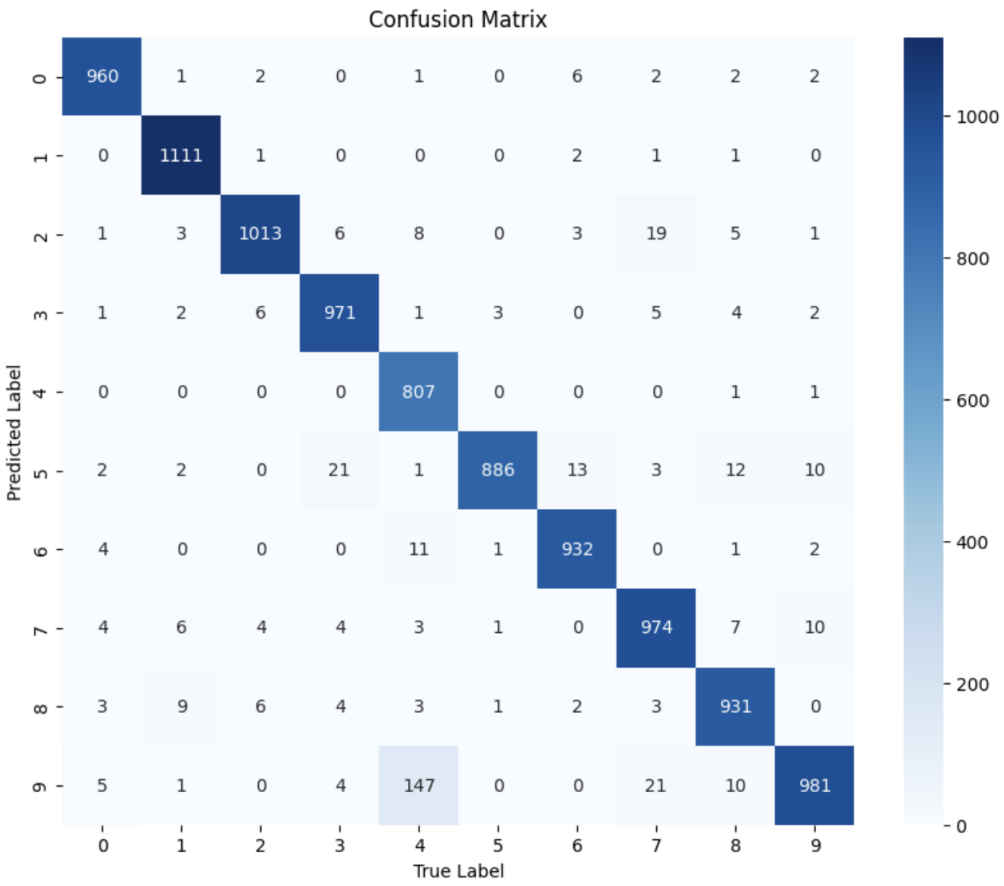
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 5, 28, 28]             140
            Conv2d-2            [-1, 5, 28, 28]             230
            Conv2d-3            [-1, 5, 28, 28]             230
         MaxPool2d-4            [-1, 5, 14, 14]               0
            Conv2d-5            [-1, 5, 14, 14]             230
            Conv2d-6            [-1, 5, 14, 14]             230
            Conv2d-7            [-1, 5, 14, 14]             230
         MaxPool2d-8              [-1, 5, 7, 7]               0
            Conv2d-9              [-1, 5, 7, 7]             230
           Conv2d-10              [-1, 5, 7, 7]             230
        MaxPool2d-11              [-1, 5, 3, 3]               0
          Flatten-12                   [-1, 45]               0
           Linear-13                   [-1, 10]             460
================================================================
Total params: 2,210
Trainable params: 2,210
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.13
Params size (MB): 0.01
Estimated Total Size (MB): 0.14
----------------------------------------------------------------
```

Number of parameters in conv1 layer = 5*28 = 140

Total number of parameters = 140+230+230+230+230+230+230+230+460= **2210.**

3. Report the following on test data: (should be implemented from scratch)
a. CNN Confusion matrix :-

```
[[9.6000e+02, 1.0000e+00, 2.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00,
        6.0000e+00, 2.0000e+00, 2.0000e+00, 2.0000e+00],
       [0.0000e+00, 1.1110e+03, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        2.0000e+00, 1.0000e+00, 1.0000e+00, 0.0000e+00],
       [1.0000e+00, 3.0000e+00, 1.0130e+03, 6.0000e+00, 8.0000e+00, 0.0000e+00,
        3.0000e+00, 1.9000e+01, 5.0000e+00, 1.0000e+00],
       [1.0000e+00, 2.0000e+00, 6.0000e+00, 9.7100e+02, 1.0000e+00, 3.0000e+00,
        0.0000e+00, 5.0000e+00, 4.0000e+00, 2.0000e+00],
       [0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 8.0700e+02, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 1.0000e+00, 1.0000e+00],
       [2.0000e+00, 2.0000e+00, 0.0000e+00, 2.1000e+01, 1.0000e+00, 8.8600e+02,
        1.3000e+01, 3.0000e+00, 1.2000e+01, 1.0000e+01],
       [4.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.1000e+01, 1.0000e+00,
        9.3200e+02, 0.0000e+00, 1.0000e+00, 2.0000e+00],
       [4.0000e+00, 6.0000e+00, 4.0000e+00, 4.0000e+00, 3.0000e+00, 1.0000e+00,
        0.0000e+00, 9.7400e+02, 7.0000e+00, 1.0000e+01],
       [3.0000e+00, 9.0000e+00, 6.0000e+00, 4.0000e+00, 3.0000e+00, 1.0000e+00,
        2.0000e+00, 3.0000e+00, 9.3100e+02, 0.0000e+00],
       [5.0000e+00, 1.0000e+00, 0.0000e+00, 4.0000e+00, 1.4700e+02, 0.0000e+00,
        0.0000e+00, 2.1000e+01, 1.0000e+01, 9.8100e+02]]
```

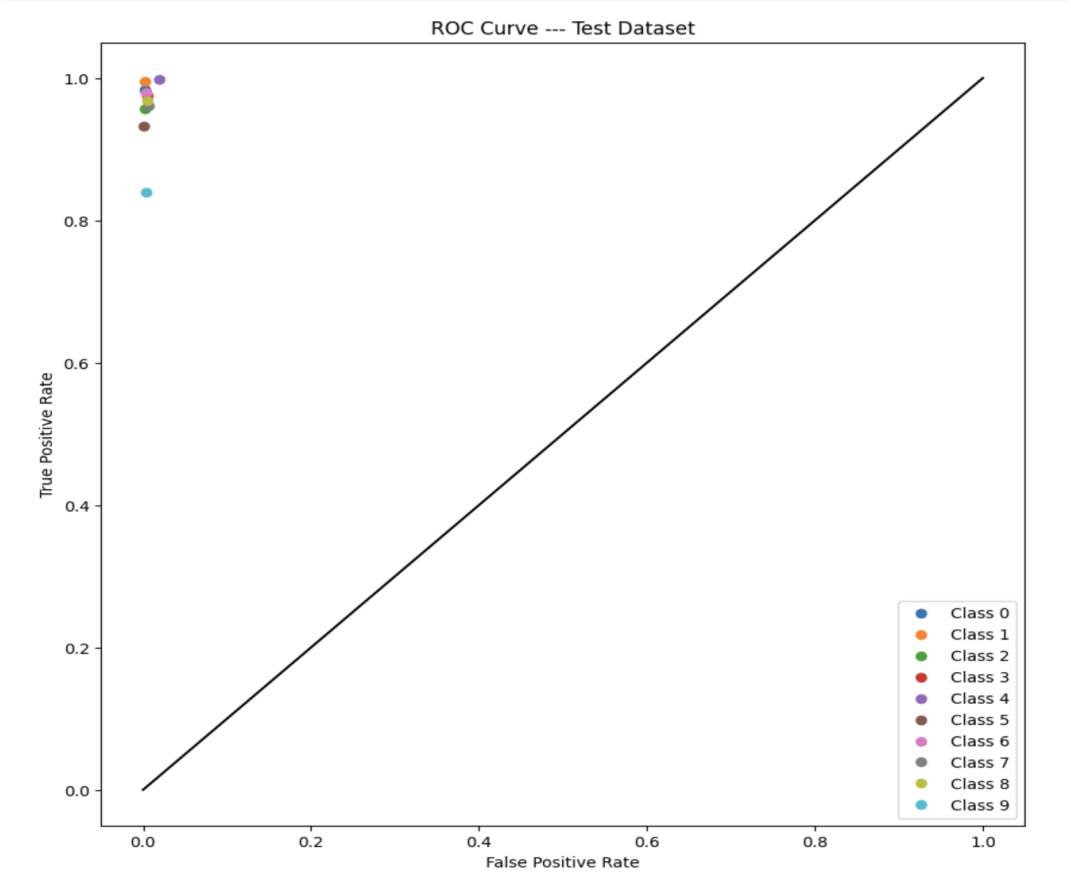


b. CNN Overall and class-wise accuracy.

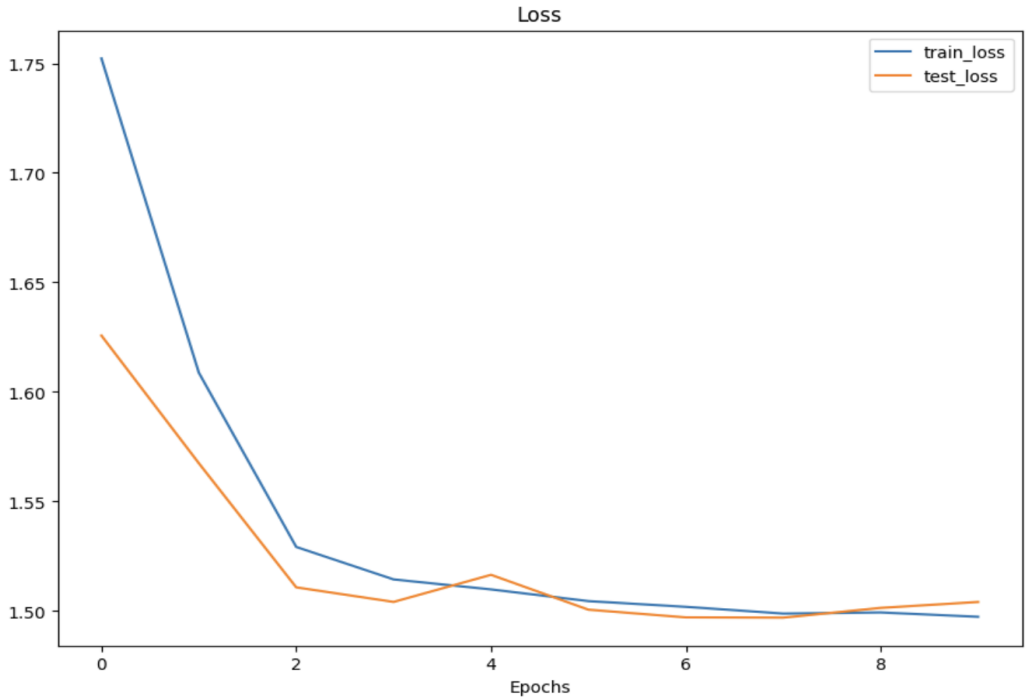
Overall and class-wise accuracy:
Overall accuracy: 0.95660001039505
Class 0 accuracy: 0.9836065769195557
Class 1 accuracy: 0.9955196976661682
Class 2 accuracy: 0.9565628170967102
Class 3 accuracy: 0.9758793711662292
Class 4 accuracy: 0.9975278377532959
Class 5 accuracy: 0.9326315522193909
Class 6 accuracy: 0.980021059513092
Class 7 accuracy: 0.9615004658699036
Class 8 accuracy: 0.9677754640579224
Class 9 accuracy: 0.8391788005828857

c. CNN ROC curve.

The ROC curve is a graphical representation of the trade-off between the true positive rate (TPR) and false positive rate (FPR) for different threshold values of a binary classifier. A good classifier will have an ROC curve that is close to the top left corner of the plot, indicating high TPR and low FPR for a wide range of threshold values.



4. CNN Report loss curve during training.



5. Replace your CNN with resnet18 and compare it with all metrics given in part 3. Comment on the final performance of your CNN and resnet18.

ResNet Architecture :- (Showing only layer 1 here, total ResNet model consists of 4 similar layers.)

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

ResNet Training Loss and Accuracy

- ResNet model is trained for 10 Epochs.

```
Epoch 1
-------------------------------
loss: 2.602721  [    0/60000]
loss: 0.206608  [ 6400/60000]
loss: 0.250168  [12800/60000]
loss: 0.139015  [19200/60000]
loss: 0.028307  [25600/60000]
loss: 0.110373  [32000/60000]
loss: 0.102260  [38400/60000]
loss: 0.084154  [44800/60000]
loss: 0.015657  [51200/60000]
loss: 0.146788  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 97.7%, Avg loss: 0.069212

Epoch 2
-------------------------------
loss: 0.134294  [    0/60000]
loss: 0.016584  [ 6400/60000]
loss: 0.055716  [12800/60000]
loss: 0.045388  [19200/60000]
loss: 0.053232  [25600/60000]
loss: 0.005694  [32000/60000]
loss: 0.072845  [38400/60000]
loss: 0.259426  [44800/60000]
loss: 0.089485  [51200/60000]
loss: 0.113797  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 98.9%, Avg loss: 0.036098

Epoch 3
-------------------------------
loss: 0.002682  [    0/60000]
loss: 0.028266  [ 6400/60000]
loss: 0.152501  [12800/60000]
loss: 0.003423  [19200/60000]
loss: 0.112255  [25600/60000]
loss: 0.004461  [32000/60000]
loss: 0.012947  [38400/60000]
loss: 0.023137  [44800/60000]
loss: 0.017747  [51200/60000]
loss: 0.073035  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 98.4%, Avg loss: 0.052695

Epoch 4
-------------------------------
loss: 0.212089  [    0/60000]
loss: 0.004403  [ 6400/60000]
loss: 0.019896  [12800/60000]
loss: 0.195652  [19200/60000]
loss: 0.001117  [25600/60000]
loss: 0.008525  [32000/60000]
loss: 0.042717  [38400/60000]
loss: 0.012200  [44800/60000]
loss: 0.013162  [51200/60000]
loss: 0.008677  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 99.0%, Avg loss: 0.035466

Epoch 5
-------------------------------
loss: 0.041638  [    0/60000]
loss: 0.010012  [ 6400/60000]
loss: 0.016529  [12800/60000]
loss: 0.004753  [19200/60000]
loss: 0.008237  [25600/60000]
loss: 0.006561  [32000/60000]
loss: 0.048608  [38400/60000]
loss: 0.024463  [44800/60000]
loss: 0.026339  [51200/60000]
loss: 0.039830  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 99.0%, Avg loss: 0.030116

Epoch 6
-------------------------------
loss: 0.002150  [    0/60000]
loss: 0.011224  [ 6400/60000]
loss: 0.003019  [12800/60000]
loss: 0.005839  [19200/60000]
loss: 0.107441  [25600/60000]
loss: 0.010211  [32000/60000]
loss: 0.116490  [38400/60000]
loss: 0.002416  [44800/60000]
loss: 0.000718  [51200/60000]
loss: 0.064193  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 99.1%, Avg loss: 0.029211
```

```
Epoch 7
------------------------------
loss: 0.069590  [    0/60000]
loss: 0.006384  [ 6400/60000]
loss: 0.055326  [12800/60000]
loss: 0.003800  [19200/60000]
loss: 0.159344  [25600/60000]
loss: 0.086674  [32000/60000]
loss: 0.110652  [38400/60000]
loss: 0.016597  [44800/60000]
loss: 0.008609  [51200/60000]
loss: 0.003215  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 98.8%, Avg loss: 0.040340
```

```
Epoch 8
------------------------------
loss: 0.001359  [    0/60000]
loss: 0.003671  [ 6400/60000]
loss: 0.014853  [12800/60000]
loss: 0.012307  [19200/60000]
loss: 0.005021  [25600/60000]
loss: 0.007704  [32000/60000]
loss: 0.001109  [38400/60000]
loss: 0.005054  [44800/60000]
loss: 0.051400  [51200/60000]
loss: 0.106300  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 98.9%, Avg loss: 0.033198
```

```
Epoch 9
------------------------------
loss: 0.005037  [    0/60000]
loss: 0.005094  [ 6400/60000]
loss: 0.004878  [12800/60000]
loss: 0.022373  [19200/60000]
loss: 0.005913  [25600/60000]
loss: 0.006963  [32000/60000]
loss: 0.053012  [38400/60000]
loss: 0.031737  [44800/60000]
loss: 0.154130  [51200/60000]
loss: 0.073286  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 99.2%, Avg loss: 0.026283
```

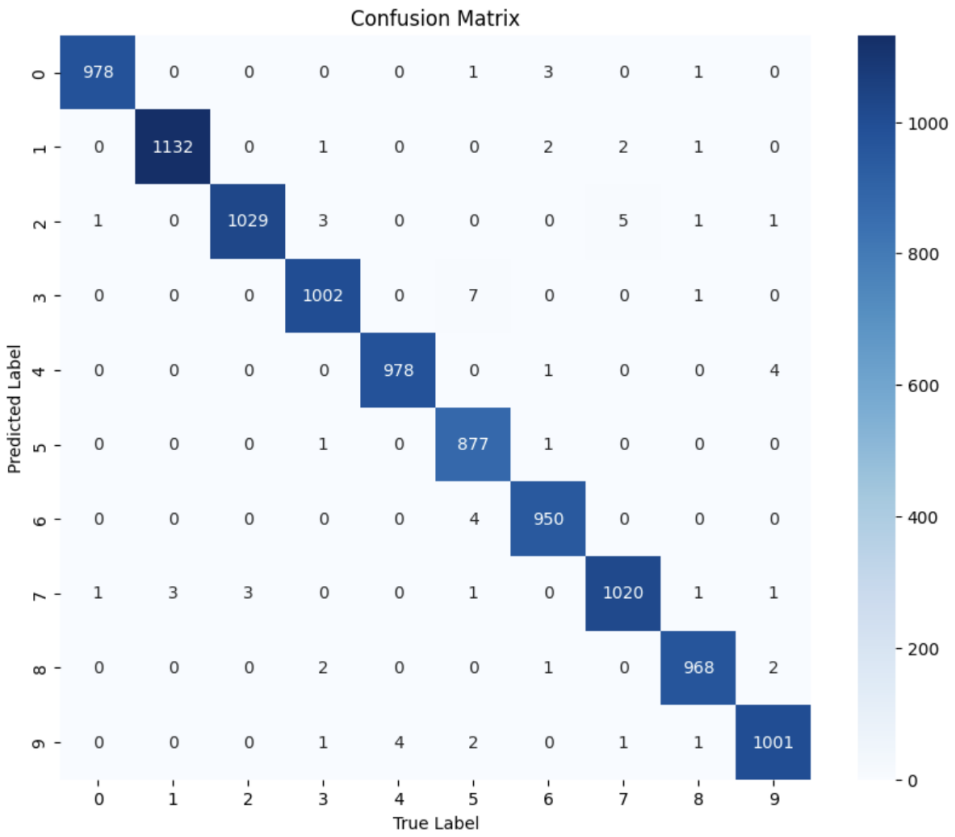
```
Epoch 10
------------------------------
loss: 0.001883  [    0/60000]
loss: 0.002640  [ 6400/60000]
loss: 0.001571  [12800/60000]
loss: 0.086904  [19200/60000]
loss: 0.021417  [25600/60000]
loss: 0.000527  [32000/60000]
loss: 0.016662  [38400/60000]
loss: 0.000321  [44800/60000]
loss: 0.023347  [51200/60000]
loss: 0.006319  [57600/60000]
size :10000, num_batches : 157
Test Error:
 Accuracy: 99.4%, Avg loss: 0.023475
```

Model Summary:-

```
       Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 5, 28, 28]             140
            Conv2d-2            [-1, 5, 28, 28]             230
            Conv2d-3            [-1, 5, 28, 28]             230
         MaxPool2d-4            [-1, 5, 14, 14]               0
            Conv2d-5            [-1, 5, 14, 14]             230
            Conv2d-6            [-1, 5, 14, 14]             230
            Conv2d-7            [-1, 5, 14, 14]             230
         MaxPool2d-8              [-1, 5, 7, 7]               0
            Conv2d-9              [-1, 5, 7, 7]             230
           Conv2d-10              [-1, 5, 7, 7]             230
        MaxPool2d-11              [-1, 5, 3, 3]               0
          Flatten-12                   [-1, 45]               0
           Linear-13                   [-1, 10]             460
================================================================
Total params: 2,210
Trainable params: 2,210
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.13
Params size (MB): 0.01
Estimated Total Size (MB): 0.14
----------------------------------------------------------------
```

Confusion Matrix of ResNet:-

[[9.7800e+02, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
3.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00],
[0.0000e+00, 1.1320e+03, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
2.0000e+00, 2.0000e+00, 1.0000e+00, 0.0000e+00],
[1.0000e+00, 0.0000e+00, 1.0290e+03, 3.0000e+00, 0.0000e+00, 0.0000e+00,
0.0000e+00, 5.0000e+00, 1.0000e+00, 1.0000e+00],
[0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0020e+03, 0.0000e+00, 7.0000e+00,
0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00],
[0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 9.7800e+02, 0.0000e+00,
1.0000e+00, 0.0000e+00, 0.0000e+00, 4.0000e+00],
[0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 8.7700e+02,
1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00],
[0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 4.0000e+00,
9.5000e+02, 0.0000e+00, 0.0000e+00, 0.0000e+00],
[1.0000e+00, 3.0000e+00, 3.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
0.0000e+00, 1.0200e+03, 1.0000e+00, 1.0000e+00],
[0.0000e+00, 0.0000e+00, 0.0000e+00, 2.0000e+00, 0.0000e+00, 0.0000e+00,
1.0000e+00, 0.0000e+00, 9.6800e+02, 2.0000e+00],
[0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 4.0000e+00, 2.0000e+00,
0.0000e+00, 1.0000e+00, 1.0000e+00, 1.0010e+03]]

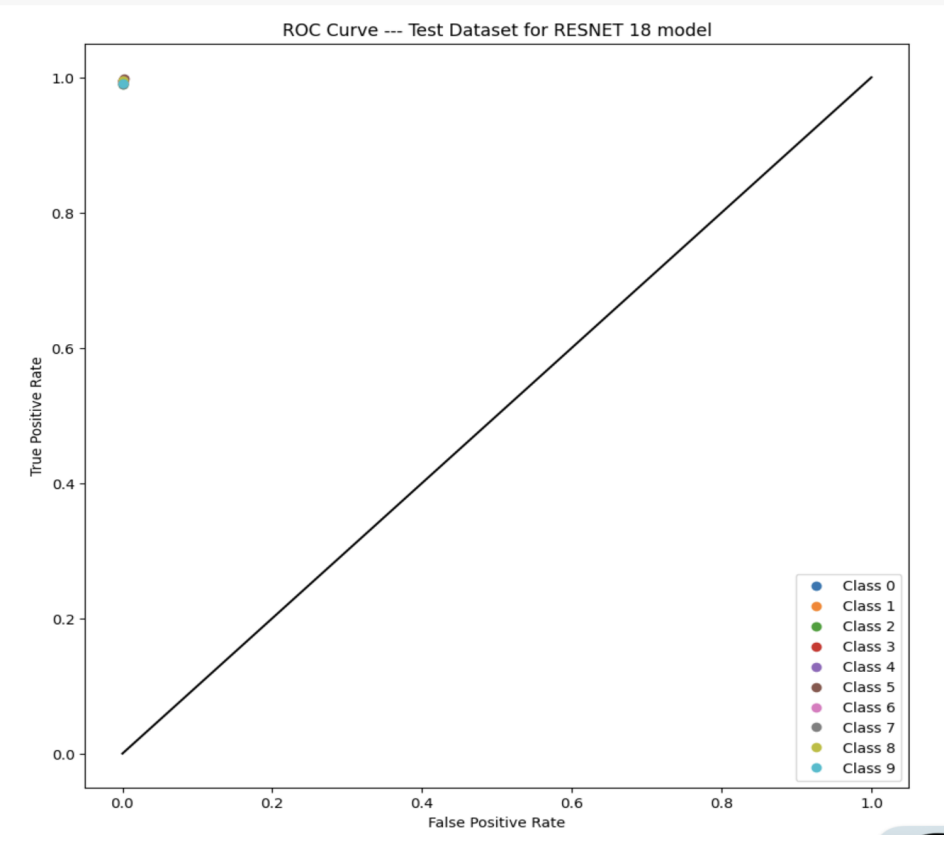


Overall and class-wise accuracy ResNet.

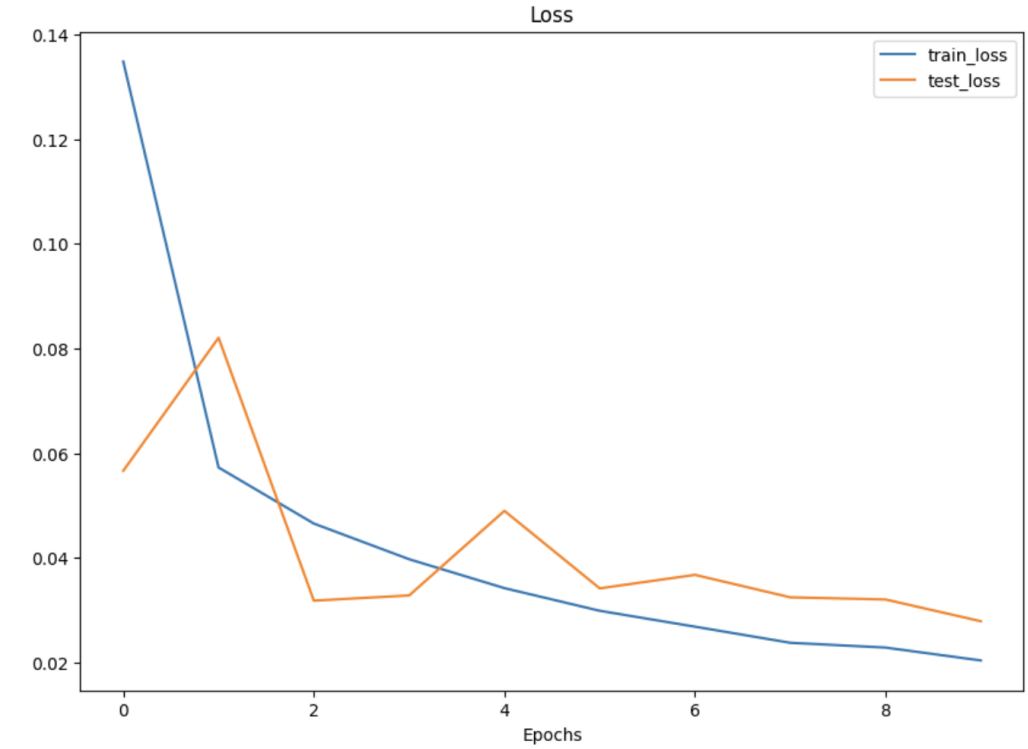
```
Overall and class-wise accuracy:
Overall accuracy: 0.9934999942779541
Class 0 accuracy: 0.9949135184288025
Class 1 accuracy: 0.994727611541748
Class 2 accuracy: 0.9894230961799622
Class 3 accuracy: 0.9920791983604431
Class 4 accuracy: 0.9949135184288025
Class 5 accuracy: 0.997724711894989
Class 6 accuracy: 0.9958071112632751
Class 7 accuracy: 0.9902912378311157
Class 8 accuracy: 0.9948612451553345
Class 9 accuracy: 0.9910891056060791
```

ROC Curve ResNet:-

- The ROC curve is a graphical representation of the trade-off between the true positive rate (TPR) and false positive rate (FPR) for different threshold values of a binary classifier.
- A good classifier will have an ROC curve that is close to the top left corner of the plot, indicating high TPR and low FPR for a wide range of threshold values.



Loss Curve of ResNet:-



Performance of your CNN and resnet18:

- The MNIST dataset was used to measure how well a custom CNN design and ResNet18 did at recognising words. On the MNIST dataset, both models were very accurate, but there were some clear differences in how they performed.
- The custom CNN design seemed to do a little better than ResNet18. ResNet18, on the other hand, is a deeper and more complicated model that might not be needed for simple tasks like MNIST classification. So, the model should be chosen based on the job at hand and the tools that are available.
- Compared to the bespoke CNN design, ResNet18's initial training loss was smaller, which shows that it started with a more accurate model. Also, ResNet18's loss went down more steadily over the course of training, while the custom CNN architecture's loss went down faster at first but then stopped going down.
- During training, both models lost less accuracy, but ResNet18 did so a little bit faster than the custom CNN design. Also, ResNet18's testing precision was a little bit higher generally, which suggests that it may work better with new data.
- In short, the results suggest that ResNet18 is easier to learn and better at adapting to new data than the bespoke CNN design.

Colab Notebook Link Q1

Q2. Download the Flicker8k dataset [images, captions]. Implement an encoder-decoder architecture for Image Captioning. For the encoder and decoder, you can use resnet/densnet/VGG and LSTM/RNN/GRU respectively. Perform the following tasks:
1. Split the dataset into train and test sets appropriately. You can further split the train set for validation. Train your model on the train set. Report loss curve during training.
2. Choose an existing evaluation metric or propose your metric to evaluate your model. Specify the reason behind your selection/proposal of the metric. Report the final results on the test set.

Ans:-

1. If the data directory is not present, the first section of code downloads and decompresses the Flicker8k_Dataset.zip file.
2. The second block of code downloads and unzips the Flickr8k_text.zip file to the data/Flickr8k_text directory if it does not already exist.
3. The code then generates the train_data.csv file, which contains the image titles, names, and paths.
4. Reading the captions in the data/Flickr8k_text/Flickr 8k.token.txt file retrieves the picture names and captions for each image. If the image cannot be accessed, it is not displayed.
5. Then, a Pandas DataFrame containing the image URLs, names, and captions is constructed and saved in the CSV file data/Flickr8k_text/train_data.csv. Existing CSV files are read into a DataFrame if the file already exists.
6. Include PyTorch and any related packages, as well as spacy, pandas, and PIL, among the imported packages.
7. Creates a vocabulary class capable of converting text to integers. It accepts a frequency threshold as an argument when constructing the vocabulary from the words in the input text. It also includes text tokenization and text numericization techniques.
8. Creates a class named FlickrDataset that inherits from the Dataset class in PyTorch. It accepts the root directory, the captions file, and the transform as parameters. It creates the vocabulary after initialising a Vocabulary object, loading the captions file using pandas, and then creating the object. In addition, it describes how to retrieve a dataset item, determine the dataset's length, and transform the captions numerically.
9. Creates the MyCollate class for use by the DataLoader. It accepts a pad index as input in order to assure that all captions are the same length.
10. Specifies the get_loader function, which accepts the root folder, annotation file, transform, batch size, number of workers, shuffle, and pin memory as parameters. It creates a FlickrDataset object with the dataset, batch size, number of workers, shuffle, pin memory, and MyCollate object as arguments, retrieves the pad index from the Vocabulary object, and then constructs a DataLoader object. Returns the DataLoader and dataset objects.
11. defining an image transformation pipeline based on transforms.Within the Compose() function, the image is resized to (128, 128) and transformed into a PyTorch tensor.

12. The path to the image directory and the location of the CSV file containing the captions are passed to an external module's get_loader() method, which loads the dataset.
13. Using torch.utils.data.random_split(), divide the dataset arbitrarily into training and test sets with a split ratio of 0.8 and a seed of 42.
14. Using torch.utils.data.DataLoader(), shuffle the training set, set the batch size to 32, and generate training and testing data loaders.
15. The training and testing set dimensions, as well as the associated data loaders, can be printed.
16. Matplotlib is used to visualise the first nine images and associated captions from the training set.
17. For image captioning, the code specifies the EncoderCNN, DecoderRNN, and CNNtoRNN neural network models.
18. The class EncoderCNN, which is derived from nn, defines the convolutional neural network used to extract image data.Module. In lieu of the completely connected layer, a linear layer with embed_size output features is used with a pre-trained ResNet-18 model. A dropout layer with a dropout probability of 0.5 and the ReLU activation function is applied to the output.
19. The class DecoderRNN is derived from nn.Module and define the recurrent neural network for creating captions. The word indices are embedded in the caption using an embedding layer, hidden states are generated based on the embeddings using a multi-layer LSTM, and the word indices of the next word in the caption are predicted using a linear layer. With a dropout probability of 0.52, the embeddings are subjected to a dropout layer.
20. CNNtoRNN is a class derived from nn.Module combines EncoderCNN and DecoderRNN to generate the entire image captioning model. It generates a list of word indices after receiving an image and a caption as input.
21. The forward() method of CNNtoRNN runs the image through the EncoderCNN to extract image features, followed by running the extracted image features and caption through the DecoderRNN to generate a list of word indices.
22. CNNtoRNN's caption_image() method constructs a caption for a provided image using beam search. It generates a list of words representing the generated caption and accepts an image, a vocabulary, and a maximum caption length as inputs.
23. During training, or EPOCHS, the model will cycle through the entire dataset fifteen times.
24. Model: CNNtoRNN(num_layers = 1, embed_size = 512, hidden_size = 512, vocab_size = len(dataset.vocab), size = 512).This line initialises the CNNtoRNN model, which accepts an image and generates its caption. A layer, a concealed size, and an embedding size of 512 are present in the model. The to(device) section instructs the model to execute on the GPU, if available, and the CPU, if not.
25. equal to loss_fn.CrossEntropyLoss(): This line initialises the loss function that will be utilised during training. Cross-Entropy loss, which is frequently applied to classification problems, quantifies the difference between the expected probability and the actual class labels.
26. lr = 0.001 weight_decay = 0; optimizer = torch.optimize.Adam In this line, the optimizer that will be used during training is initialised. In this circumstance, the Adam optimizer is

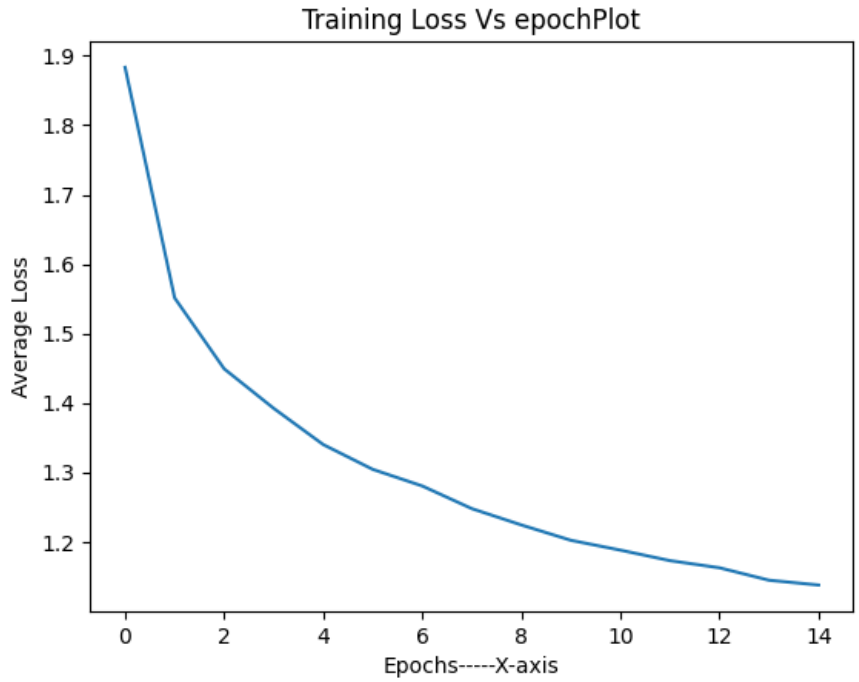
utilised with a learning rate of 0.001,. According to the params=model.parameters() specification, the model's parameters should be modified during training.

27. Finally we have drawn the loss curve and then evaluated the model
28. We have got the perplexity score and bleu score of the 7 random images with its actual and generated caption.

Results:-
n_epochs=15
lr=0.001
Embedded_size=512
Hidden_size=512

Vsualisation:-

Loss graph:-



Evaluation Metrics:-

1. **Perplexity:**
   Perplexity is a measurement used to evaluate the performance of language models. It quantifies how well a language model predicts a sequence of words or tokens given the preceding context.In simple terms, perplexity measures how surprised or uncertain a language model is when it encounters the next word in a sequence. A lower perplexity value indicates that the model is more confident in its predictions and has a better understanding of the language.
2. **BLEU_SCORE:**
   BLEU (Bilingual Evaluation Understudy) score is a metric used to evaluate the quality of machine-generated translations or text. It was originally developed for machine translation tasks but has since been adopted for other natural language processing tasks, including image captioning.The BLEU score ranges from 0 to 1, where 1 indicates a perfect match between the machine-generated text and the reference text. A higher BLEU score indicates a better quality of the generated text.

Result of the model:-

100%
1265/1265 [05:03<00:00, 7.81it/s]
Loss: 9.562185511345, Perplexity: 26.18829784430857

Actual Caption:
the biker is riding around a curve in the road .
Generated Caption:
a man on a bike is riding along a trail through the woods .
BLEU Score: 0.315

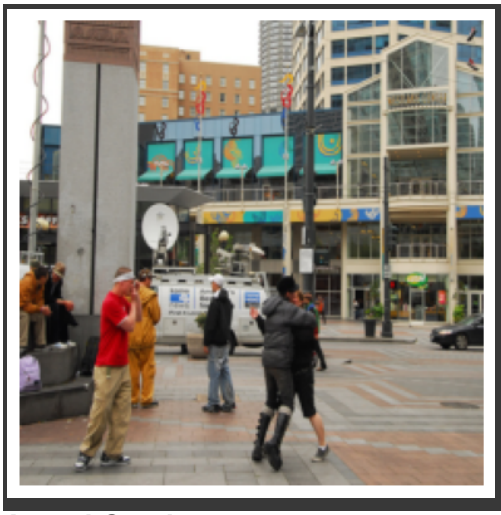
Actual Caption:
an older man does the splits on a hardwood floor .
Generated Caption:
a man in a white shirt and jeans is sitting on a wooden bench .
BLEU Score: 0.0020

Actual Caption:
a boy in a soccer uniform running on a field
Generated Caption:
a boy in a white uniform is playing soccer .
BLEU Score: 0.5134

Actual Caption:
a dog on the beach .
Generated Caption:
a dog runs across the sand .
BLEU Score: 0.0000

Actual Caption:
the man is waterskiing .
Generated Caption:
a man in a wetsuit surfing .
BLEU Score: 0.0000

Actual Caption:
a referee watches two basketball players playing basketball .
Generated Caption:
a basketball player in white is defending the ball in a game .
BLEU Score: 0.0000



Actual Caption:
the man in the red shirt is taking a photo of the couple that is hugging on the sidewalk .
Generated Caption:
a man in a white shirt and jeans is standing on a street corner with a cardboard sign .
BLEU Score: 0.0000

Conclusion:-

We have seen that the actual and generated captions are almost giving the same results.We have also noticed that the perplexity score is 26.18 which is upto the mark which indicates the logical words predicted after the word in the sequence.

Also the BLEU score has been considered for the evaluation parameter of the model as it compares the two sentences and measure the similarity score of the sentences word by word. And it helps in comparing with the other model as it ranges from0- 1 .So we can easily compare to other images.

Limitations:-

1. **Computational Constraints:-We have seen that BLEU score of most of the image are 0.000 due to the colab constraints we ran the model for only 15 epochs because for each epochs it requires almost 5 mins .Hence model does not perform so well that it gives the perfect BLEU score.**
   Although the predicted caption was relatable to the image.

References:-

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
PyTorch tutorial on image captioning by Yunjey Choi:
https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning
Tutorial on image captioning by Xavier Giro-i-Nieto:
https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2

Google colab link CODE 2

Q3. Use the dataset from Assignment 3 (Q. 4). Train YOLO object detection model (any version) on the train set. Compute the AP for the test set and compare the result with the HOG detector. Show some visual results and compare both of the methods.

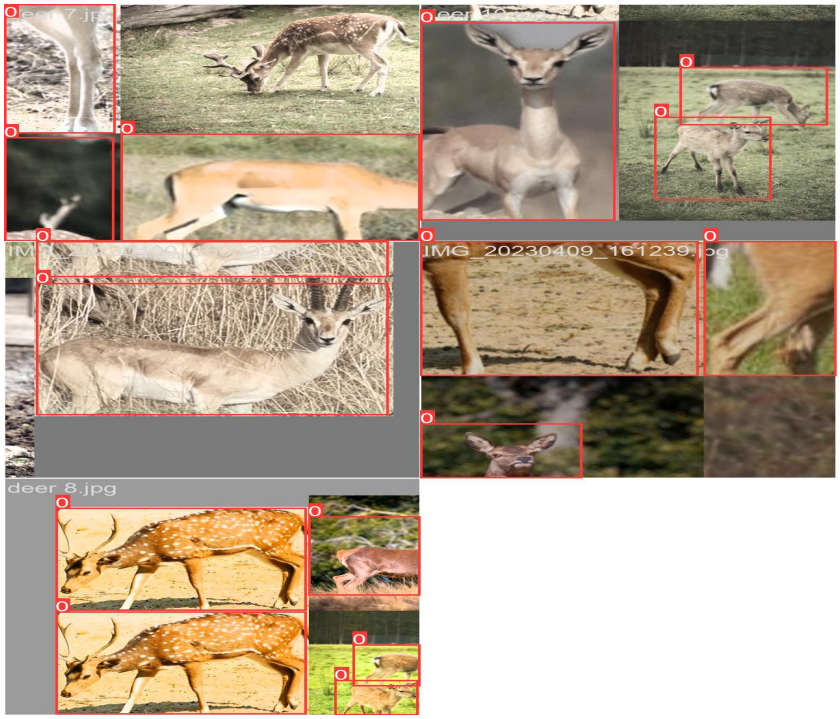
Solution 3:-

Annotated Dataset and yaml file as per the YOLO requirements used in this question is here.
We have arranged as per the directoried in the above file and zipped it.
Import it to gdrive and mount it to colab and then call it in the colab file
Create an annotated dataset using the Makesence tool.
This command installs the ultralytics package, which provides an implementation of the YOLOv8 object detection model and associated utilities.
ultralytics.checks(): This ensures the system meets the requirements for operating YOLOv8 by performing a series of tests.
import os imports the os module to enable operating system communication.
os.chdir(os.getcwd()): Current working directory is now the current directory.Train in yolo model=yolov8n.pt epochs=50, imgsz=640, batch=5, data="./Datasets/deer_dataset.yaml": This trains a YOLOv8 model with the following parameters on the deer dataset.
Considering that Yolov8n.pt is a unique model, this specifies the model architecture to use.
The location of the dataset configuration file is indicated by the expression data="./datasets/deer_dataset.yml".

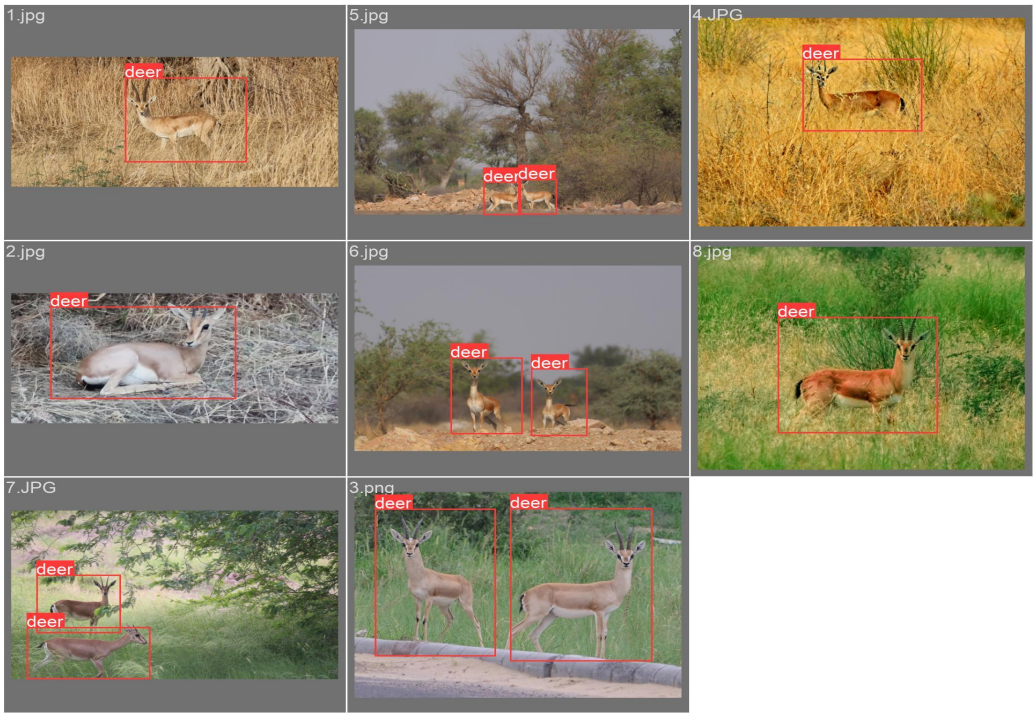
What is YOLO object detection model?
In computer vision, YOLO (You Only Look Once) is a prominent real-time object detection algorithm. It is renowned for its speed and precision in object detection in images and videos.The central concept of YOLO is to partition the input image and make predictions for each grid cell. YOLO conducts detection in a single pass, as opposed to traditional object detection algorithms that use multiple region proposals. Using a convolutional neural network (CNN) architecture, it predicts the bounding boxes and class probabilities directly.

Here we have used:-
YOLO V8.0.91 -pretrained model
Num_epochs-50
Batch_size-5
Training images- 27 images
Val images - 8 images
Labels- 0(DEER)

Training Data Batches :



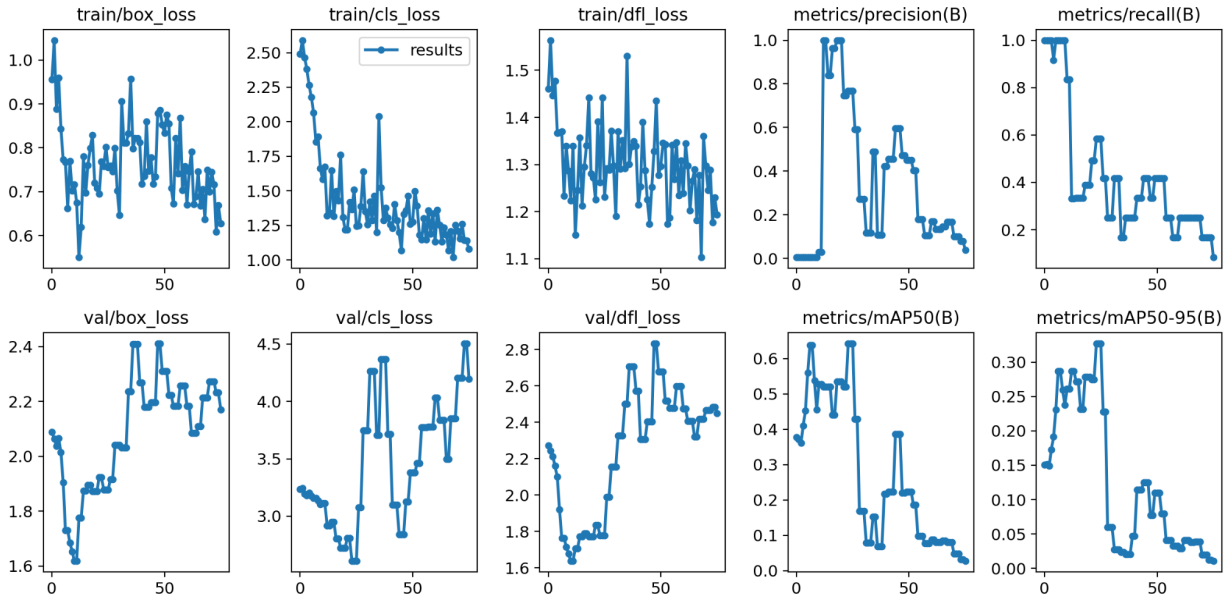
Testing Data Ground Truth:-



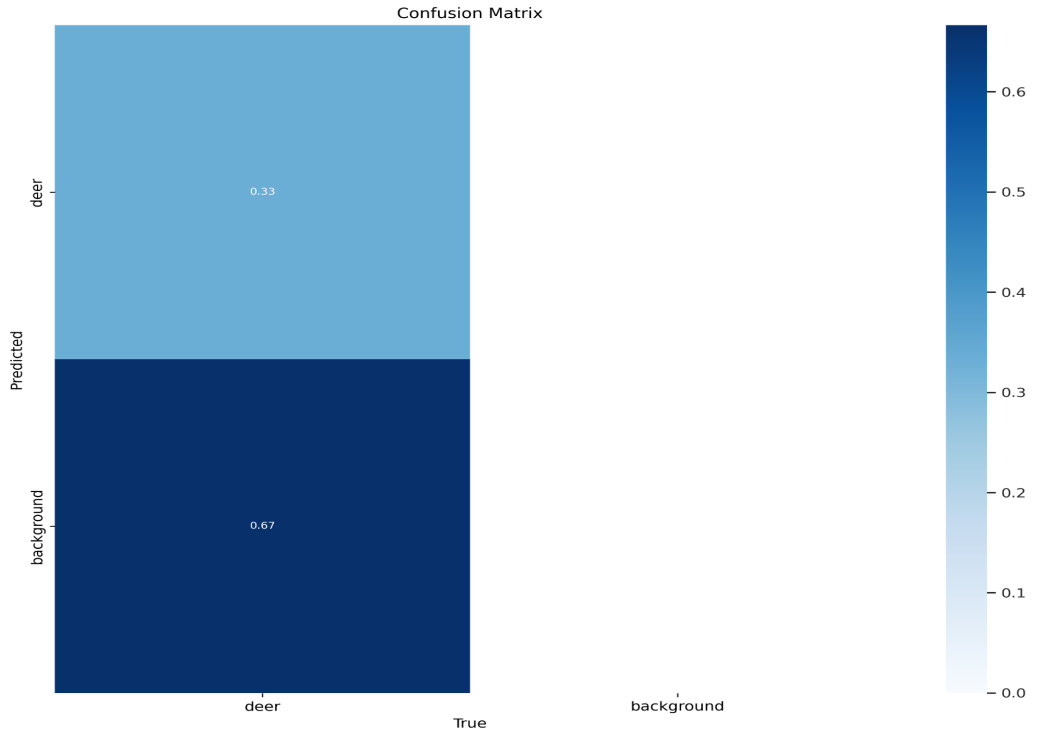
Testing Data Predictions:-

Results:-

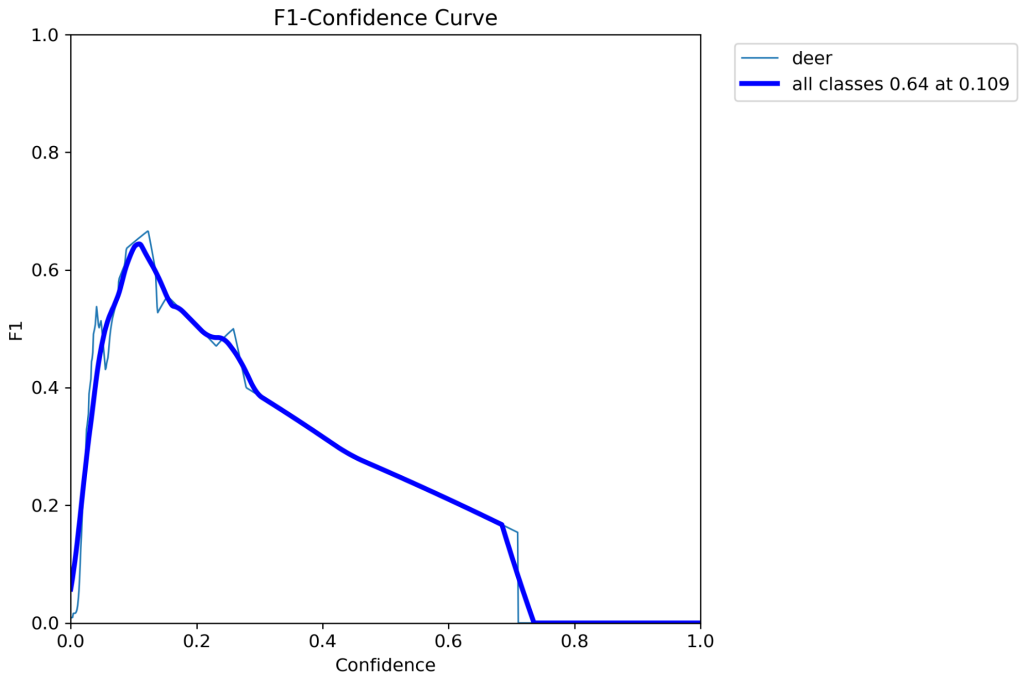
- **Train Loss vs Validation Loss Curves :-**



- **Confusion Matrix**



- **F1 Score Curve**



Result of the trained image in csv format generated by the yolo model:Reult.csv

	train/box_loss	train/cls_loss	train/dfl_loss	metrics/precision(B)	metrics/recall(B)	metrics/mAP50(B)	metrics/mAP50-95(B)	val/box_loss	val/cls_loss
0	0.95574	2.4917	1.4608	0.005	1	0.37835	0.15075	2.088	3.2349
1	1.0453	2.5902	1.5638	0.005	1	0.37122	0.15161	2.0644	3.2442
2	0.88833	2.4657	1.4468	0.005	1	0.36149	0.14888	2.038	3.1939
3	0.95987	2.38	1.4773	0.005	1	0.41046	0.17305	2.0667	3.1829
4	0.84316	2.2657	1.3664	0.00458	0.91667	0.45283	0.19147	2.0151	3.2037
5	0.77355	2.1772	1.3679	0.005	1	0.55976	0.23109	1.9052	3.1754
6	0.76723	2.0651	1.3705	0.005	1	0.63827	0.28681	1.7303	3.157
7	0.66232	1.8543	1.2336	0.005	1	0.63827	0.28681	1.7303	3.157
8	0.7694	1.8919	1.339	0.005	1	0.53835	0.25935	1.6845	3.1373
9	0.70272	1.6634	1.2866	0.005	1	0.45589	0.23764	1.6538	3.1025
10	0.7161	1.5835	1.2235	0.02923	0.83333	0.5279	0.26102	1.6195	3.115
11	0.67539	1.675	1.3397	0.02923	0.83333	0.5279	0.26102	1.6195	3.115
12	0.55027	1.3194	1.1502	1	0.33133	0.52017	0.28711	1.7755	2.9178
13	0.61948	1.3423	1.2458	1	0.33133	0.52017	0.28711	1.7755	2.9178
14	0.7799	1.6465	1.357	0.83891	0.33333	0.51984	0.27162	1.8742	2.9483
15	0.69726	1.3156	1.2126	0.83891	0.33333	0.51984	0.27162	1.8742	2.9483
16	0.76017	1.4955	1.2948	0.96203	0.33333	0.44048	0.23165	1.8955	2.8071

Comparison between HOG vs YOLO:-

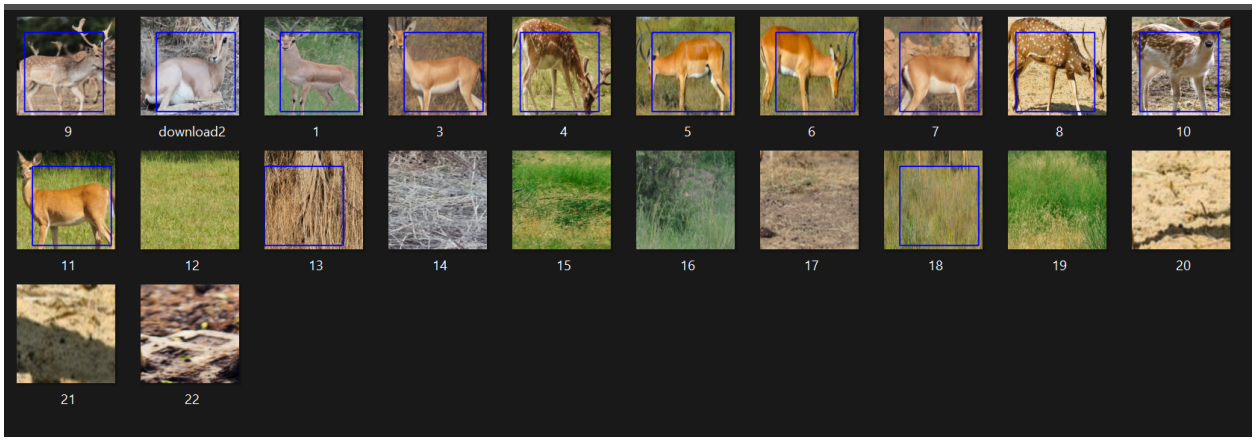


Fig1. Hog results

Fig2. YOLO result

We have seen that in case of HOG ,some of the non deer images are also detected as a deer and mostly in the deer it gives the good sliding window and detection.
In the YOLO some of the deer is not detected reason being small test size and also only 1 class label is used, also yolo is good for the fast movements like video and real time movement.
YOLO is known for its speed and real-time object detection capabilities. It can detect objects in an image or video stream very quickly.
YOLO directly predicts bounding boxes and class probabilities for multiple objects in a single pass, making it suitable for scenarios where real-time processing is essential.

HOG is a feature-based object detection method that focuses on detecting objects based on their local gradient patterns.
HOG is effective in detecting objects with distinguishable textures or shape characteristics.
It performs well in scenarios where high accuracy is important, such as pedestrian detection.

Overall HOG is accurate and YOLO is fast.

References:-

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
https://towardsdatascience.com/convolutional-neural-network-cnn-architecture-explained-in-plain-english-using-simple-diagrams-e5de17eacc8f
https://www.youtube.com/watch?v=2osIZ-dSPGE
https://stackoverflow.com/questions/2148543/how-to-write-a-confusion-matrix
https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning

Google ColabQ3