Question 1: find, read and summarize a paper (40 pts)

Paper Link

In the structure of this paper, where can you find below info?

- **Background** – '*Abstract*' |
- **Motivations** – '*Abstract*' and '*Introduction*'
- **Short Summary of the proposed work** – '*Introduction*'
- **Contribution highlights** – '*Experiments*'
- **Problem formulation** – *'Proposed method' and 'Joint localization, perception and prediction'*
- **Conclusion** – '*Quantitative results'* and '*Conclusion*'

What problem it is resolving?
The paper aims to tackle the problem of localization error in the modern autonomy pipeline. Localization errors can have serious consequences for perception-prediction (P2) and motional planning (MP) systems since they take high-definition maps as input assuming access to accurate online localization. However, if incorrect or faulty localization is taken as input by the P2 and MP systems then it can lead to missed detections, prediction errors, and bad planning which may cause collisions.

What are the input and output of the proposed method?
The input of the proposed method is the data from the LiDAR which is then converted to bird's eye view voxelization with the channels of the 2D input corresponding to the height dimension. Whereas the output is the localization and perception-prediction model.

The challenges of this problem?
The main challenges to solving this problem were – integration of a recursive Bayesian filter in the localizer, and a finer-grained evaluation of the errors concerning motional planning.

What are the purposes for the first and second figures?
The first figure depicts how a small amount of error in localization causes the 'self-driving vehicle' to go into the lane of opposite traffic resulting in a collision. Whereas the second figure depicts the effect of localization error on perception-prediction and motion planning.

What new concept its method introduced?
It proposed the solution of multi-task learning capable of jointly localizing against an HD map while also performing object detection and motion forecasting.

What counterparts it compared to and links of typical counterparts?
For the tasks focussed in the paper namely – perception, prediction, and localization; following are the existing approaches discussed in the paper:

Some existing models used in autonomous driving are capable of performing object detection and motion forecasting tasks jointly. Such models provide a number of benefits like – fast inference, uncertainty propagation, and improved performance.

The reference links are – [1] [2] [3] [4]

What aspect are included in the Experiment sections?
The following aspects were included in the experiment section. Firstly, it discussed the accuracy of the multi-task model on the joint localization and prediction-perception task. Secondly, it was showed how these metrics translate to a safe ride based on the motion planning metrics.

What results it concludes?
The main result concluded was that localization errors can be successfully deleted and corrected in less than 2ms of GPU time while performing multi-task learning of joint localization against an HD map.

Question 2): train an instance segmentation model with PyTorch tutorial and Google Collab; or you can deploy it in Palmetto Cluster. (60 pts)

(1) Show screenshots of successful setup, training, and inference on the colab. (15 points)

Installing pycocotools -

```
[1] %%shell

    pip install cython
    # Install pycocotools, the version by default in Colab
    # has a bug fixed in https://github.com/cocodataset/cocoapi/pull/354
    pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
```

```
Requirement already satisfied: cython in /usr/local/lib/python3.7/dist-packages (0.29.28)
Collecting git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI
  Cloning https://github.com/cocodataset/cocoapi.git to /tmp/pip-req-build-du27lhoe
  Running command git clone -q https://github.com/cocodataset/cocoapi.git /tmp/pip-req-build-du27lhoe
Requirement already satisfied: setuptools>=18.0 in /usr/local/lib/python3.7/dist-packages (from pycocotools==2.0) (57.4.0)
Requirement already satisfied: cython>=0.27.3 in /usr/local/lib/python3.7/dist-packages (from pycocotools==2.0) (0.29.28)
Requirement already satisfied: matplotlib>=2.1.0 in /usr/local/lib/python3.7/dist-packages (from pycocotools==2.0) (3.2.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0) (1.4.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0) (3.0.7)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0) (1.21.5)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib>=2.1.0->pycocotools==2.0) (3.10.0.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib>=2.1.0->pycocotools==2.0) (1.15.0)
Building wheels for collected packages: pycocotools
  Building wheel for pycocotools (setup.py) ... done
  Created wheel for pycocotools: filename=pycocotools-2.0-cp37-cp37m-linux_x86_64.whl size=264346 sha256=c956ac36098d1da8a73eb5cdbb85b6078b338df9247e3c2ab26e686fd72b9e40
  Stored in directory: /tmp/pip-ephem-wheel-cache-j1f8fzk5/wheels/e2/6b/1d/344ac773c7495ea0b85eb228bc66daec7400a143a92d36b7b1
Successfully built pycocotools
Installing collected packages: pycocotools
  Attempting uninstall: pycocotools
    Found existing installation: pycocotools 2.0.4
    Uninstalling pycocotools-2.0.4:
      Successfully uninstalled pycocotools-2.0.4
Successfully installed pycocotools-2.0
```

Downloading the Penn-Fudan dataset -

```
[2] %%shell

    # download the Penn-Fudan dataset
    wget https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip
    # extract it in the current folder
    unzip PennFudanPed.zip
```

```
--2022-03-31 23:56:00--  https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip
Resolving www.cis.upenn.edu (www.cis.upenn.edu)... 158.130.69.163, 2607:f470:8:64:5ea5::d
Connecting to www.cis.upenn.edu (www.cis.upenn.edu)|158.130.69.163|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 53723336 (51M) [application/zip]
Saving to: 'PennFudanPed.zip'

PennFudanPed.zip    100%[===================>]  51.23M  57.4MB/s    in 0.9s

2022-03-31 23:56:01 (57.4 MB/s) - 'PennFudanPed.zip' saved [53723336/53723336]

Archive:  PennFudanPed.zip
   creating: PennFudanPed/
  inflating: PennFudanPed/added-object-list.txt
   creating: PennFudanPed/Annotation/
```

Importing required libraries –

```
[3] import os
    import torch
    import numpy as np
    from PIL import Image
    import torch.utils.data
```

Displaying a sample from the dataset and its mask -

```
[4] Image.open('PennFudanPed/PNGImages/FudanPed00025.png')
```

```
[30] mask = Image.open('PennFudanPed/PedMasks/FudanPed00025_mask.png')
     # each mask instance has a different color, from zero to N, where
     # N is the number of instances. In order to make visualization easier,
     # let's adda color palette to the mask.
     mask.putpalette([
         0, 0, 0, # black background
         255, 0, 0, # index 1 is red
         255, 255, 0, # index 2 is yellow
         255, 153, 0, # index 3 is orange
     ])
     mask
```

```
[14] # let's train it for 10 epochs
     from torch.optim.lr_scheduler import StepLR
     num_epochs = 10

     for epoch in range(num_epochs):
         # train for one epoch, printing every 10 iterations
         train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=10)
         # update the learning rate
         lr_scheduler.step()
         # evaluate on the test dataset
         evaluate(model, data_loader_test, device=device)
```

Pre-processing the train & test dataset by calling the 'PennFudanDataset' class defined in the code -

```
[12] dataset = PennFudanDataset('PennFudanPed', get_transform(train=True))
     dataset_test = PennFudanDataset('PennFudanPed', get_transform(train=False))

     # split the dataset in train and test set
     torch.manual_seed(1)
     indices = torch.randperm(len(dataset)).tolist()
     dataset = torch.utils.data.Subset(dataset, indices[:-50])
     dataset_test = torch.utils.data.Subset(dataset_test, indices[-50:])

     # define training and validation data loaders
     data_loader = torch.utils.data.DataLoader(
         dataset, batch_size=2, shuffle=True, num_workers=4,
         collate_fn=utils.collate_fn)

     data_loader_test = torch.utils.data.DataLoader(
         dataset_test, batch_size=1, shuffle=False, num_workers=4,
         collate_fn=utils.collate_fn)
```

Constructing an optimizer and defining training parameters for the neural network -

```
[13] device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

     # our dataset has two classes only - background and person
     num_classes = 2

     # get the model using our helper function
     model = get_instance_segmentation_model(num_classes)
     # move model to the right device
     model.to(device)

     # construct an optimizer
     params = [p for p in model.parameters() if p.requires_grad]
     optimizer = torch.optim.SGD(params, lr=0.005,
                                 momentum=0.9, weight_decay=0.0005)

     # and a learning rate scheduler which decreases the learning rate by
     # 10x every 3 epochs
     lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                    step_size=3,
                                                    gamma=0.1)
```

Training the model for 10 epochs -

```
[14] # let's train it for 10 epochs
     from torch.optim.lr_scheduler import StepLR
     num_epochs = 10

     for epoch in range(num_epochs):
         # train for one epoch, printing every 10 iterations
         train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=10)
         # update the learning rate
         lr_scheduler.step()
         # evaluate on the test dataset
         evaluate(model, data_loader_test, device=device)
```

```
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.834
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.986
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.950
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.555
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.842
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.385
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.879
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.879
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.812
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.884
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.751
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.986
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.918
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.407
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.762
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.346
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.799
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.799
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.725
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.805
Epoch: [4]  [ 0/60]  eta: 0:02:00  lr: 0.000500  loss: 0.1394 (0.1394)  loss_classifier: 0.0175 (0.0175)  loss_box_reg: 0.0271 (0.0271)
Epoch: [4]  [10/60]  eta: 0:01:26  lr: 0.000500  loss: 0.2230 (0.2198)  loss_classifier: 0.0232 (0.0342)  loss_box_reg: 0.0434 (0.0545)
Epoch: [4]  [20/60]  eta: 0:01:07  lr: 0.000500  loss: 0.1785 (0.2073)  loss_classifier: 0.0262 (0.0311)  loss_box_reg: 0.0388 (0.0480)
Epoch: [4]  [30/60]  eta: 0:00:49  lr: 0.000500  loss: 0.1618 (0.1947)  loss_classifier: 0.0196 (0.0289)  loss_box_reg: 0.0253 (0.0432)
Epoch: [4]  [40/60]  eta: 0:00:32  lr: 0.000500  loss: 0.1618 (0.1907)  loss_classifier: 0.0199 (0.0281)  loss_box_reg: 0.0319 (0.0411)
Epoch: [4]  [50/60]  eta: 0:00:16  lr: 0.000500  loss: 0.1654 (0.1875)  loss_classifier: 0.0239 (0.0272)  loss_box_reg: 0.0356 (0.0401)
Epoch: [4]  [59/60]  eta: 0:00:01  lr: 0.000500  loss: 0.1781 (0.1921)  loss_classifier: 0.0239 (0.0276)  loss_box_reg: 0.0312 (0.0421)
```

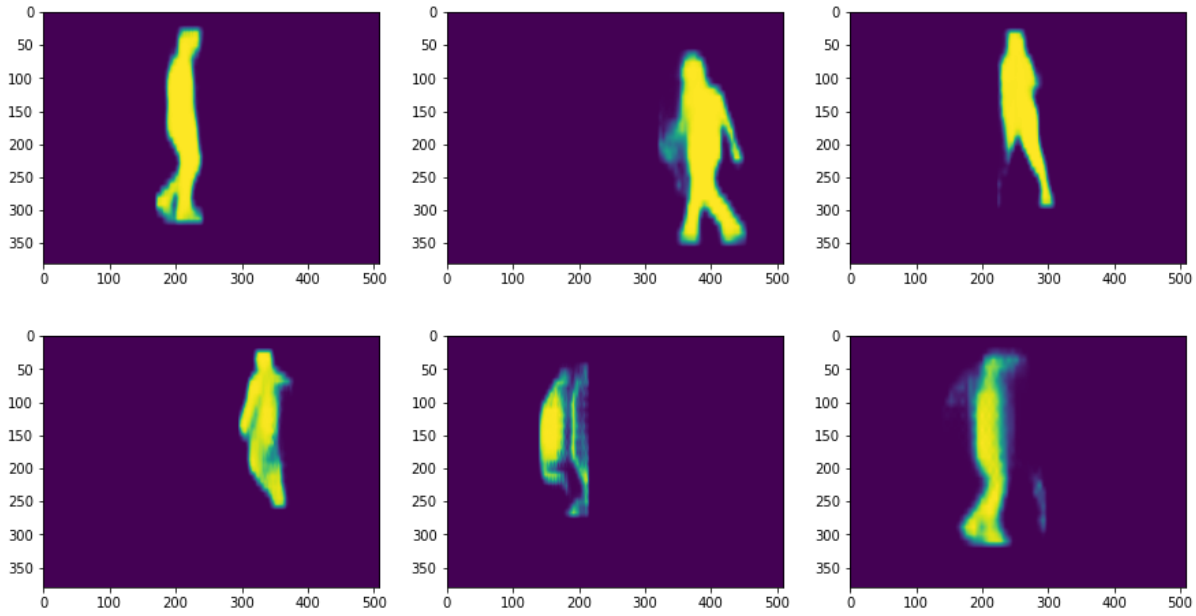
(2) Inference different images in the test set and show screenshots (5 points)

I. First test image -

```
[26] # pick one image from the test set
     img, _ = dataset_test[4]
     # put the model in evaluation mode
     model.eval()
     with torch.no_grad():
         prediction = model([img.to(device)])
```

```
[31] Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

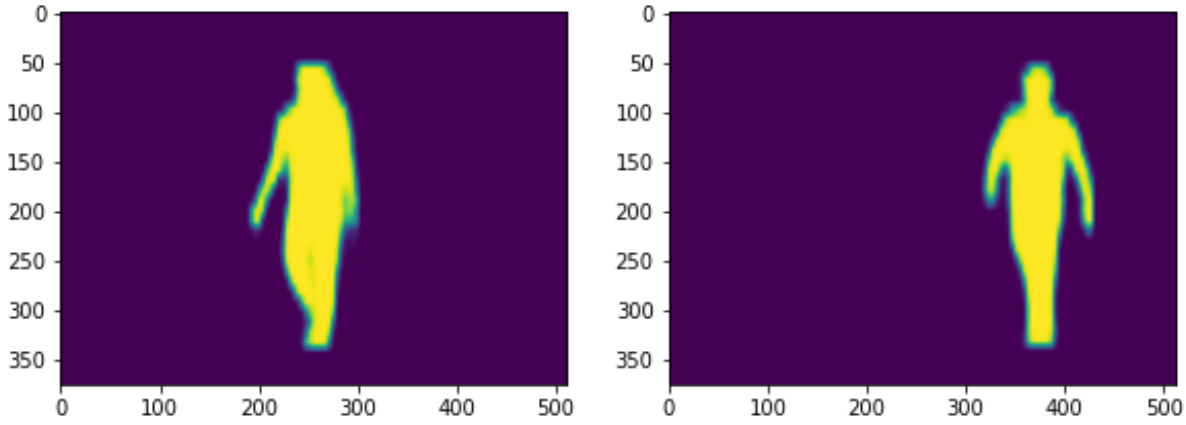
Predicted Masks -



II. Second test image –

```
[67] Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

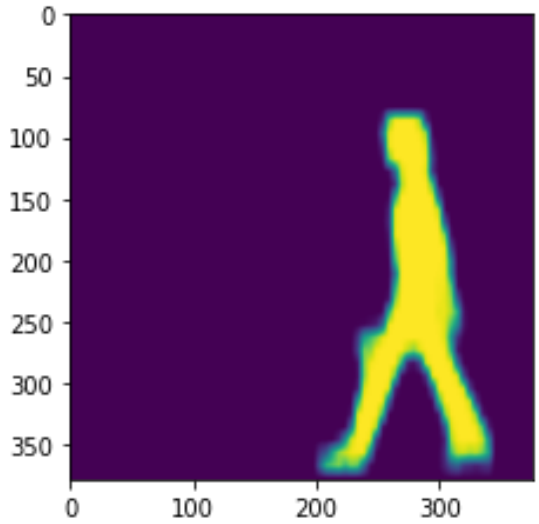
Predicted Masks –



III. Third test image –

```
Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

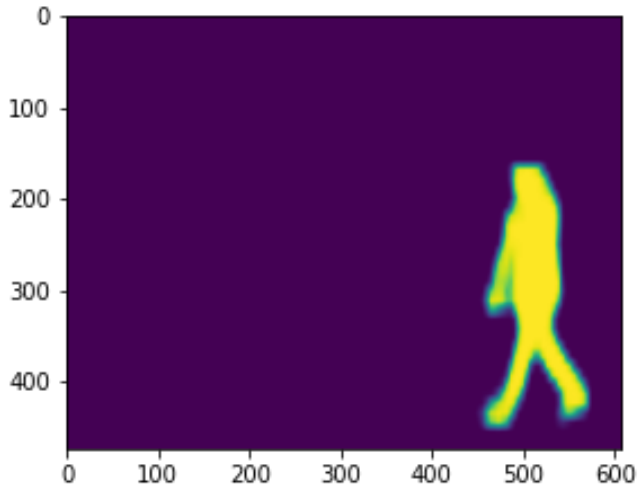
Predicted Masks –



IV. Fourth test image –

```
[99] Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

Predicted Masks –



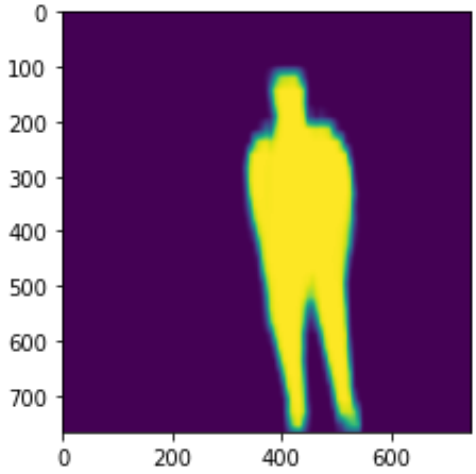
(3) Inference on your own image (10 points)

Converting my test image to tensor and passing it to the model -

```
[156] img = Image.open("/content/my_image.jpg")
      convert_tensor = T.ToTensor()
      img_tensor = convert_tensor(img, target=None)

      img, _ = img_tensor
      # put the model in evaluation mode
      model.eval()
      with torch.no_grad():
          prediction = model([img.to(device)])
```

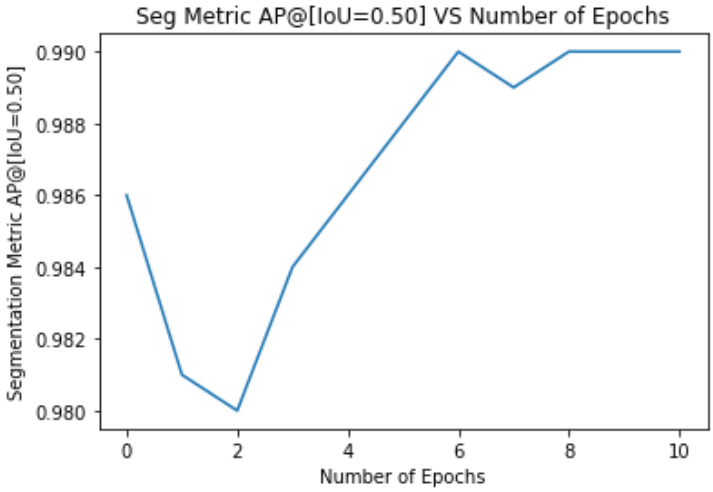
```
Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

Predicted Mask –



(4) Plot the segmentation metric AP@[IoU=0.50] against the number of training Epochs on a Graph. (15 points) (TIPS: if you cannot save accuracy during training, you can manually collect it from outputs.)

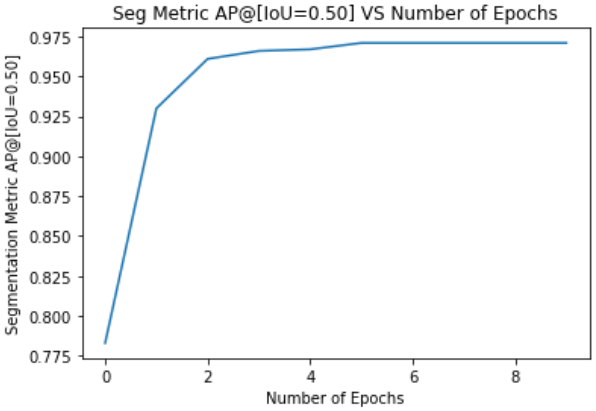
```
import numpy as np
import matplotlib.pyplot as plt

list = [0.986, 0.981, 0.980, 0.984, 0.986, 0.988, 0.990, 0.989, 0.990, 0.990, 0.990]
list = np.array(list)

plt.plot(list)
plt.title("")
title = plt.title("Seg Metric AP@[IoU=0.50] VS Number of Epochs")
ylabel = plt.ylabel("Segmentation Metric AP@[IoU=0.50]")
xlabel = plt.xlabel("Number of Epochs")
```



(5) Change the batch size, optimizer, learning rate etc... Plot and analyse its influence on accuracy. (15 points)

I. Changing the learning rate from 0.005 to 0.0001 and the optimizer to Adam –

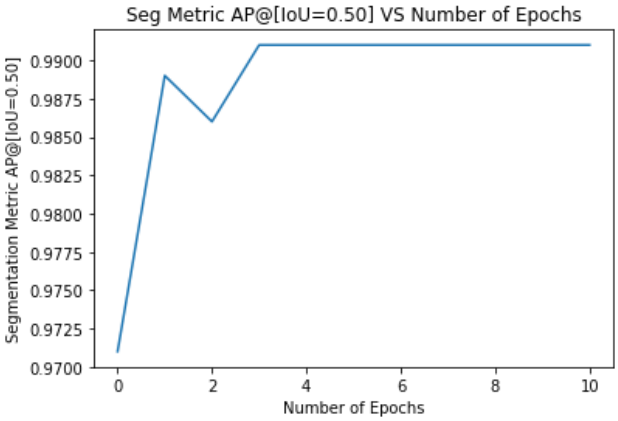
From the graph below it can be inferred that a low learning rate allows the model to learn faster, therefore leading to a better learning curve as seen as below. On the other hand, a smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train.



II. Changing optimizer to Adam and keeping the other parameters as following –

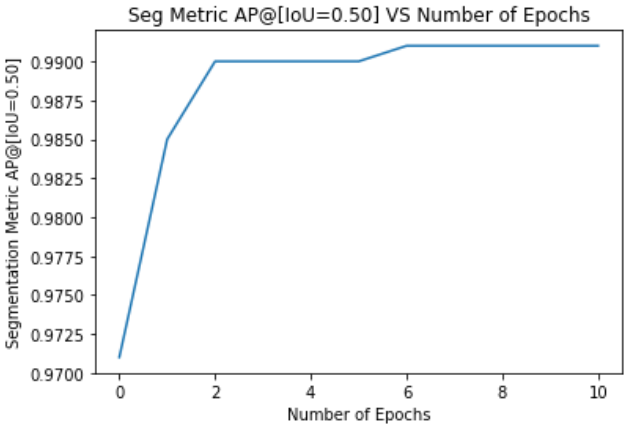
Learning rate = 0.0001
$\beta_1 = 0.9$
$\beta_2 = 0.999$
eps = 1e-08
Weight decay = 0

From the graph below it can be inferred that there is a slight increase in accuracy using Adam. The initial results were strong and after that the accuracy remained constant, but there is evidence that Adam converges to dramatically different minima compared to SGD (or SGD + momentum). Furthermore, a few of the test images had more predicted masks then there was when the optimizer was SGD.



III. Changing the batch size from 2 to 5 while keeping learning rate at 0.0001 –

Increasing the batch size reduced the training time slightly but the accuracy was almost the same as it was with batch size equal to 2. Furthermore, on increasing the batch size to higher numbers the accuracy was seen to be increased slightly.



Conclusion

In this task, I have worked with different models and based on that I have learned about the following parameters -

Learning Rate: - It is as an essential hyperparameter that defines the step size for the model. A lower learning rate was keeping the model from reaching the minima and would need more number of epochs. While a higher learning rate was making the model learn faster, but at times it may miss the minimum loss function and would only reach in the vicinity of it. On the other hand, a lower learning rate was giving a better chance to reach the minimum loss function, but as a tradeoff it needed higher number of epochs that means more computational power.

Batch Size: - Batch size essentially controls the accuracy of the error gradient during training of the neural network. A larger batch size was leading to faster training, but at the same time it was a trade-off in good training. In other words, small batch size leads to less accuracy of error gradient.

Number of Epochs: - My choice of this parameter was based on the limitation of computational resources available. However, I found that on average 10 epochs was leading to excellent accuracy.