F**inal project report**

**Introduction**

The purpose of this project is to design a convolutional neural network for semantic segmentation of photographic images. This designed network is later trained and tested on A data set of images so that we can achieve as high as possible mean Intersection over Union(mIoU) on the test dataset. The resulting optimized semantic segmented images can later be used in vision applications like for the development of a self-driving car which automatically analyzes the camera image to determine its acceleration and direction. Simply put, it would be much easier to determine those if we had a segmented image rather than just the raw camera images.

There are many possible ways to improve the performance (mIoU on test data) and I implemented the following methods to optimize and improve the mIoU on the test data.

• Optimize the network design

• Optimize the hyper-parameters

• Use an appropriate optimization method

• Employ data augmentation

• Employ regularization method

• Employ residual blocks

• Employ learning rate scheduler

in this project, I used

Optimize the network design part:

the first given network had only 3 levels of downsampling and upsampling, with no regularization methods. Therefore, I added one more downsampling and upsampling levels to the original network(which each have two convolutinal layers each). In doing so, the mIoU showed an increase from 0.196 to 0.233.

Optimize the hyper-parameters:

in this I part I tried to tune the hyper-parameters like learning rate, number of epochs and batch size. However, what I observed in the project is that there was no significant change in the value of the final mIoU I got unless the network design is changed.

Optimization method:

I used Adam and I set the learning rate to 0.0005 without the learning rate scheduler at first. However, since there was no much improvement, I tried using ReduceLROnPlateau with a patience of five. And later, I changed it to multistep learning rate, which performed a bit better than the ReduceLROnPleatu.

Data Augmentation part:

to increase the data that is fed to train the model, I used data augmentation such as horizontal flip, vertical flip and others more and it increase the final mIoU I got. However, I didn’t use them at the very last model because I was getting tensor stack error quite often.

Regularization part:

Since dropout was already incorporated into the network already had regularization. the only regularization I used is batch normalization, which is incorporated to the optimization technique.

These two regularization methods might help me prevent the over fitting of the model.

residual blocks:

while training, I added one more upsampling and downsampling(which is two additoinal convolutional layers in each.) So there was a problem of vanishing gradient and to prevent that I used residual blocks in both upsampling and downsampling methods(i.e. in both encoder and decoder).

**Procedure**

To achieve as high as possible mean Intersection over Union(mIoU) on the test data set, I applied many ways such as :

1. I first run the given code and I was able to get an mIoU of around 0.196

2. I then employed data regularization method of **Batch normalization** to standardizes the inputs to a layer for each mini-**batch**. This has the effect of stabilizing the learning process. However, it didn’t bring much of a difference in the value of mIoU. So I proceeded to use other optimization methods and then Just to see how the hyper parameters affect the performance, I changed the hyper parameters like batch\_size and n\_epoch to get a varying values of mIoU from 0.18 to 0.197 so there was no much change just by changing the hyper-parameters.

2. Then I added random crop and horizontal clip data augmentation to increase the data variety feed to the network training. Then surprisingly I got a final miou of around 0.226, which is a great improvement from the original value of 0.197.(however, I finally dropped the augmentation methods because I was getting an error). But till this point, I didn’t use learning rate scheduler. Then I tried to use a learning rate scheduler(MultiStepLR). However, this didn’t bring much of a differnece. There were even times where the final miou was smaller than the previous values I got after using learning rate scheduler.

4. Then I added two more convolutional layers both in downsampling and upsampling each. Yet it didn’t actually make a huge difference. One time I got an mIoU of around 0.233, which isn’t a huge improvement compared to the effect of data augmentation. This could have been due to vanishing gradient. So I tried to use a residual network in my architecture.

5. knowing that it’s not changing much I tried to use residual network, and I applied residual network in both up sampling and downsampling, which increeased the mIoU significantely from 0.233 to 0.259. while running the code in this network structure, mIoU increased very fast in the very first 60 epochs jumping from 0.04 to 0.259. However, mIoU wasn’t changing at all after the 60th epoch. This could have been due to a convergence at a very early stage. So the next step I took was to change the hyper-parameters like the learning rate. I also tried to change the mile stones of the multistep learning rate scheduler I used in my architecture. This actually improved the mIoU that was not changing after certain number of epochs. Changing the mile stones many times, I was able to get a maximum of 0.273 final mIoU. At this time I tried to increase the number of epochs, but the final miou was fluctuating and the final miou that was printed got lower. Then I again decreased the number of epochs to 150 and I saw an increase from 0.273 to 0.284.