Assignment 1 – Part 2 -Unet Implementation for Image Segmentation

*Did you train your network on GPU or CPU?

Yes, I have set the GPU as True and hence the network was trained on GPU.

*How big was the size of the images you trained the network on? (You are allowed to rescale the images in the dataset to speed up the training and help with memory usage. For the final results, use a size of at least 128x128.)
The testing is done on the original image which is a 1024* 1024 image and the image and labels are loaded using the Dataloader function.

The size of the image tensor is 572*572 which is sent in a 4D tensor to the model.

*Which data augmentation strategies did you use?

The data augmentations involved are the increase of Brightness, Vertical and the Horizontal Flip to the image are done. I have also implemented the Image mirroring by flipping the image up and down, rotating it by 90 degree and then cropping the image as per the output file size of 572*572.

*Did you deviate from the original network structure? (E.g. with the ReLU in the last step.)

No I have used the Unet architecture as such.

*How many epochs did you train?

I have trained the images for 40 epochs. I have provided the entire epoch run in the file name **Training_epoch_run.**

*How long did your training take?
The network took around 5 minutes for all the 40 epochs to run, generally seconds for every epoch.

OVERVIEW OF IMPLMENTATION

dataloader.py

The dataloader.py majorly handles the images and loading, the data augmentation, image mirroring and the Normalization of the image

The original images and labels are loaded in the dataloader.py file.

The data augmentations involved are the increase of Brightness, Vertical and the Horizontal Flip to the image are done. I have also implemented the Image mirroring by flipping the image up and down, rotating it by 90 degree and then cropping the image as per the outfile size of 572*572.

The data images and labels are converted into a numpy array.

model.py

model.py file closely follows the Unet Architecture. The contracting and the extracting steps are performed as per the Unet framework.

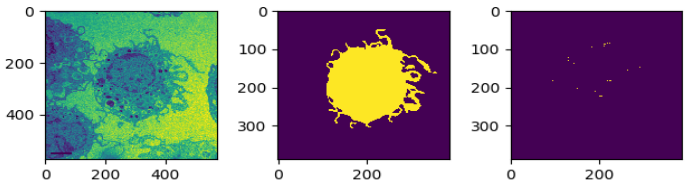
No deviations were made from the original network structure.
t 2 -Unet Implementation for Image Segmentation

train.py

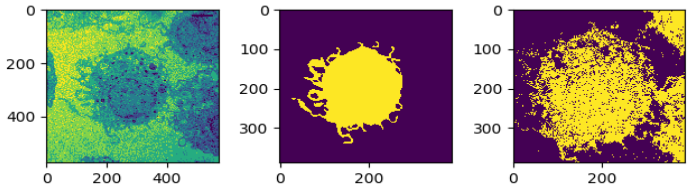
The numpy array of data images and labels are converted into a tensor and then fed to the network using the .cuda() function. The result output result output would be a 4D tensor and the labels are a 3D tensor labels used to calculate the loss function.

The loss function involves the implementation of the softmax and the cross entropy loss function.

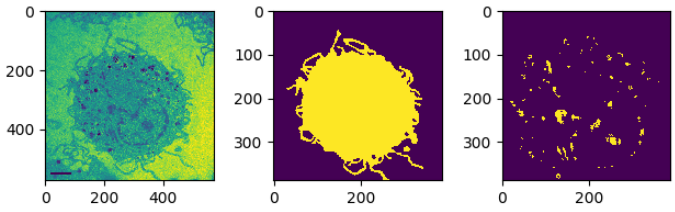
Without Data Augmentation



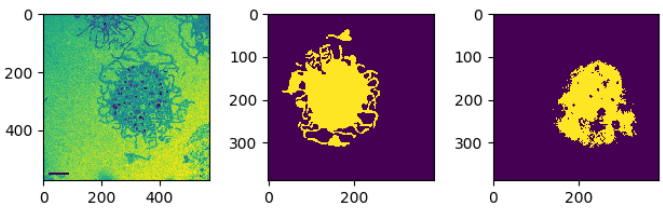
With Data Augmentation



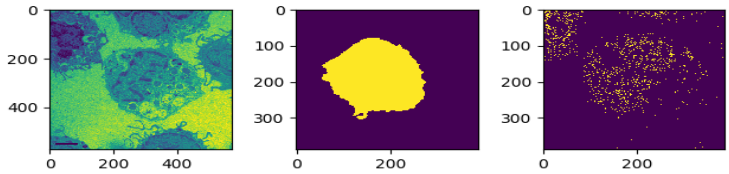
Without Data Augmentation



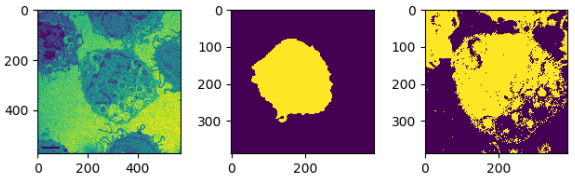
With Data Augmentation



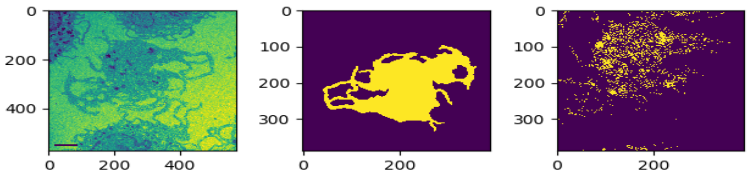
Without Data Augmentation



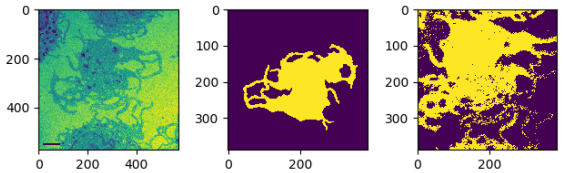
With Data Augmentation



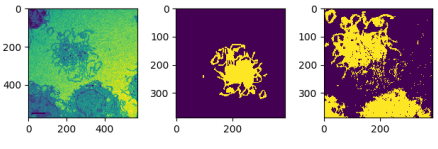
Without Data Augmentation

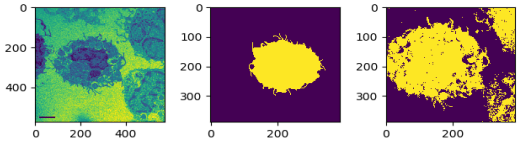


With Data Augmentation



I have added few more Images after the data augmentation for reference.





So, after the data augmentation is performed the results are way better as we could see above.

I have run the loss calculation for 40 epochs and I have provided the entire epoch run in the file name training_epoch_run. The loss has been decreasing and after running **40 epochs** the loss is **0.153668.**

```
Training sample 22 / 35 - Loss: 0.257232
tensor(0.1530, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 23 / 35 - Loss: 0.153032
tensor(0.1384, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 24 / 35 - Loss: 0.138355
tensor(0.1333, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 25 / 35 - Loss: 0.133316
tensor(0.2872, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 26 / 35 - Loss: 0.287250
tensor(0.1783, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 27 / 35 - Loss: 0.178313
tensor(0.1733, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 28 / 35 - Loss: 0.173315
tensor(0.1584, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 29 / 35 - Loss: 0.158443
tensor(0.1131, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 30 / 35 - Loss: 0.113079
tensor(0.1375, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 31 / 35 - Loss: 0.137473
tensor(0.1405, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 32 / 35 - Loss: 0.140526
tensor(0.0591, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 33 / 35 - Loss: 0.059090
tensor(0.1373, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 34 / 35 - Loss: 0.137304
tensor(0.2383, device='cuda:0', grad_fn=<DivBackward0>)
Training sample 35 / 35 - Loss: 0.238313
Checkpoint 40 saved !
Epoch 40 finished! - Loss: 0.153668
```