**Project 4 Report**

The objective of this project was to train a Generative Adversarial Network (GAN) on a dataset containing images of parked vehicles acquired by security cameras in a parking so that fake images that generated by the generator network looks very similar to the real ones. There are 46,125 images in total and each image in the dataset contains one vehicle. To accomplish this, we modified a pytorch implementation of vanilla GAN for our use case [1].

**Dataset and Network Structure:**

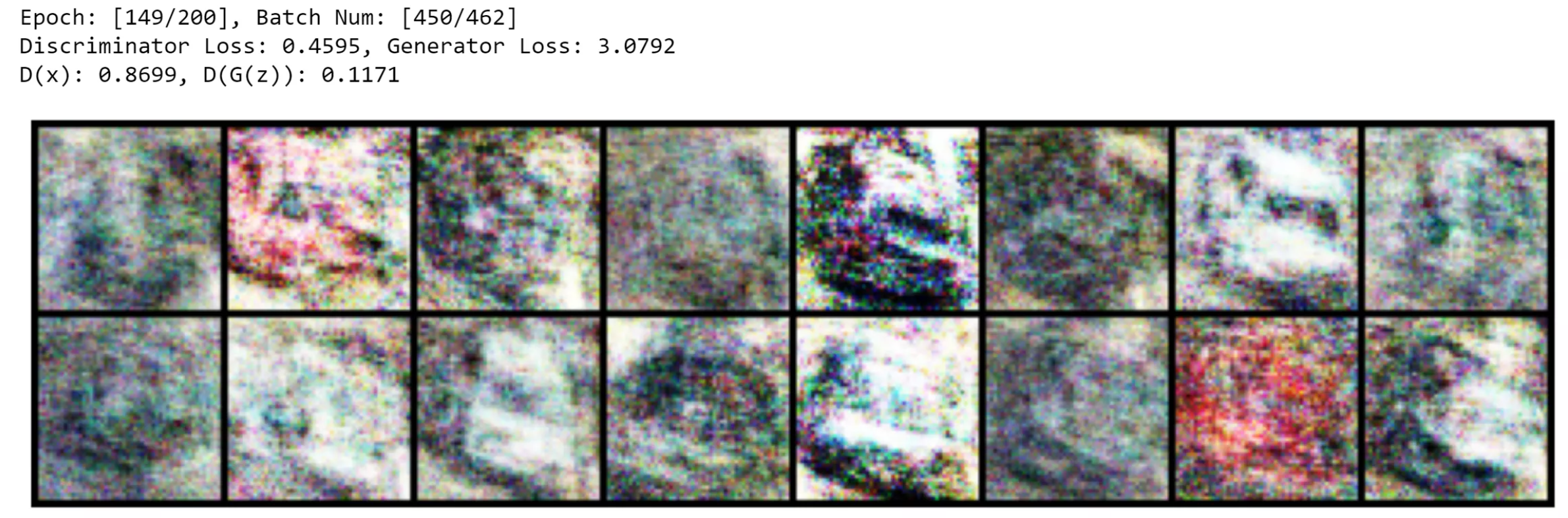
First, we had to create a custom class, named ‘VehicleDataset’, that inherits the pytorch DataSet class for our dataset and use case. This converts the images to tensor and enables transformations when loaded to the Dataloader object. We then used a pytorch transforms object to resize the images to 50x50 pixels and normalize them between -1 and 1 to help the training process. The Discriminator network was therefore defined to take a flattened image input of 7500 (3x50x50) with three hidden layers, each followed by a Leaky-Relu nonlinearity and a Dropout layer to prevent overfitting. A Sigmoid activation function is applied to the last layer to obtain a value between 0 and 1. The Generator network takes a latent variable vector of random noise as input and outputs a 7500 valued vector, with three hidden layers followed by a Leaky-Relu nonlinearity. The output layer has a Tanh activation function, which maps the resulting values in the (-1,1) range just like the normalized real image inputs. Adam was used as the optimization algorithm for both neural networks with a learning rate of 0.0002 and Binary Cross Entropy Loss was used as the loss function.

**Performance Evaluation:**

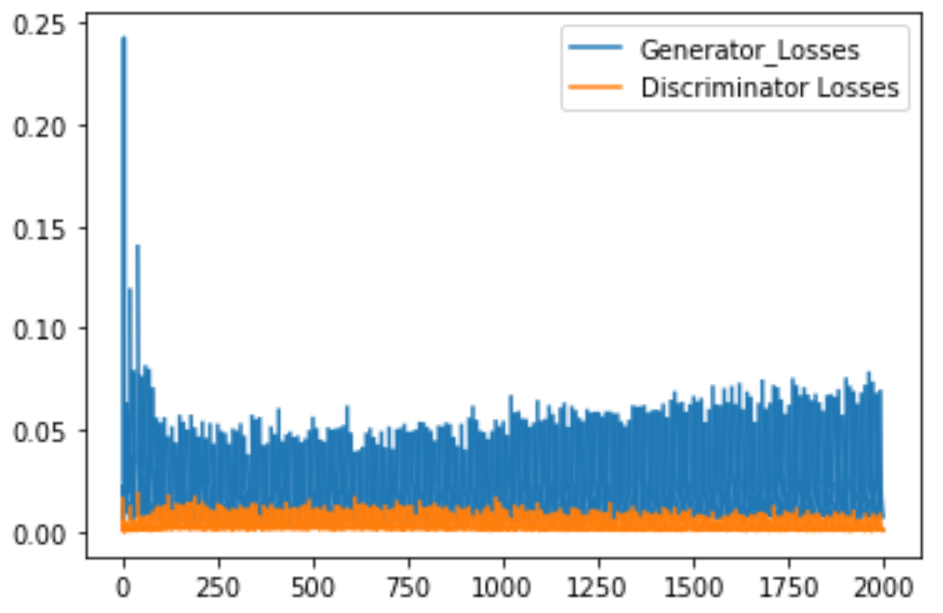
To log the training progress and visualize results, functions from the Logger class was used, which was imported from the utils.py file included in the project folder. In order to visualize the generator output, 16 samples of random noise were fed to the generator every 50 batches and the output images were displayed. The status logger displayed generator and discriminator losses every 50 batches as well as the output from each network. Furthermore, the trained weights of the generator were saved every 50 epochs in case we wanted to load pretrained weights to the generator network for re-training. Finally, the generator and discriminator losses per 50 batches were plotted using matplotlib.

**Results and Conclusion:**

The training was performed on an NVIDIA GeForce GTX 1050 gpu integrated within an ASUS Zenbook laptop. We trained for 200 epochs, which took around 1.5 hours. The best results were achieved around 150 epochs, as shown below:



We can see that the GAN was able to generate blurry yet discernible fake images of parked vehicles, which is pretty good considering the low resolution of the dataset. However, results could obviously be improved by incorporating convolutions and/or more hidden layers within the networks. Unfortunately, we could not implement these due time and resource constraints, but ultimately, we were able to successfully implement GAN to generate discernable synthetic images. The plot below shows that generator loss roughly decreases till around the 750th batch as expected, but then it roughly increases till the 2000th batch. The discriminator loss initially increases but then stays steady, which is not what we were hoping for as a steady increase would indicate a more successful training.



**Reference:**

1. https://github.com/diegoalejogm/gans/blob/master/1.%20Vanilla%20GAN%20PyTorch.ipynb