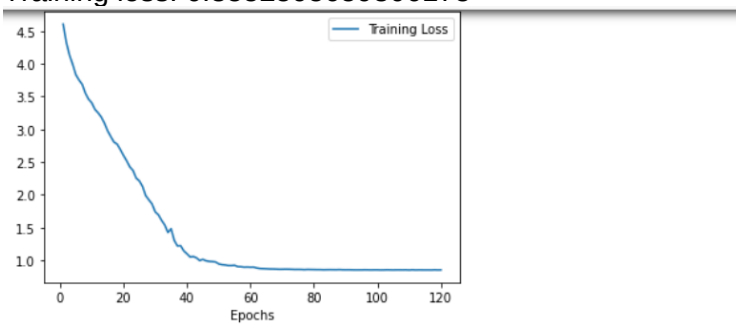
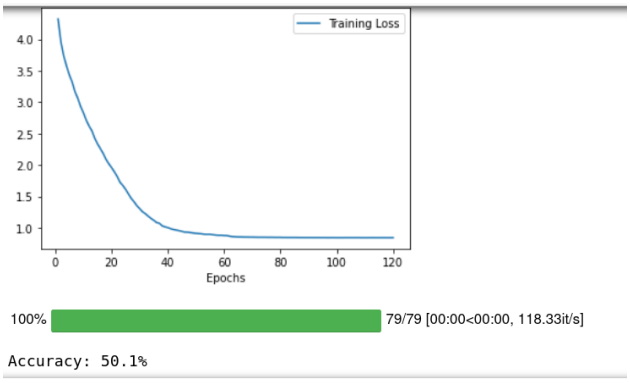
Chapter 1 was all covered in CSE291
Chapter 2 Deep Learning
Resnet18

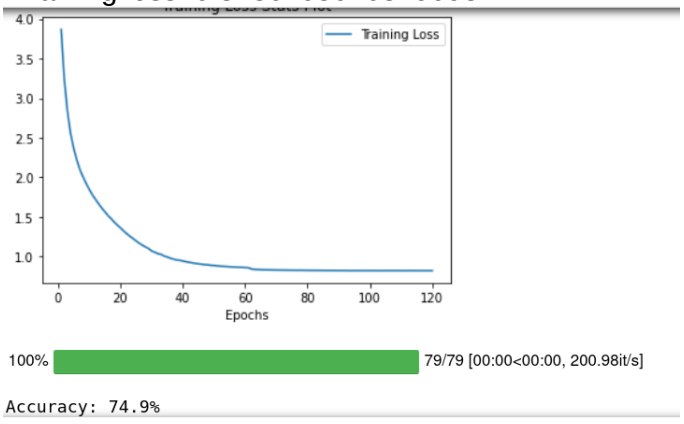
The graph is the training loss stats plot. At the bottom of the graph is the test accuracy.
When using 5% of training data
Training loss: 0.8552595659306175



When using 20% of training data
Training loss: 0.8422121099936657



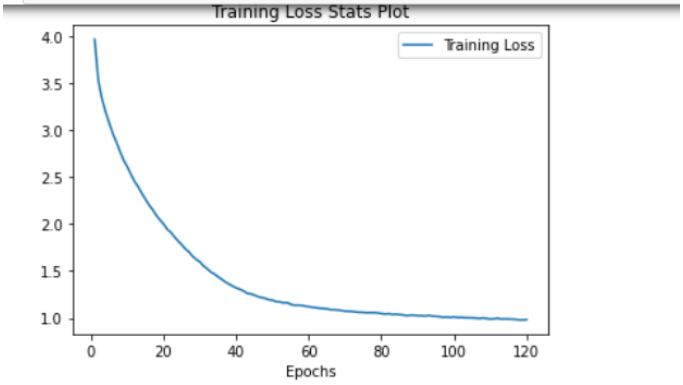
When using full dataset:
Training loss: 0.8236703016819098



In this case, test accuracy is 74.9%, which meets the expected result >= 71%.

VIT

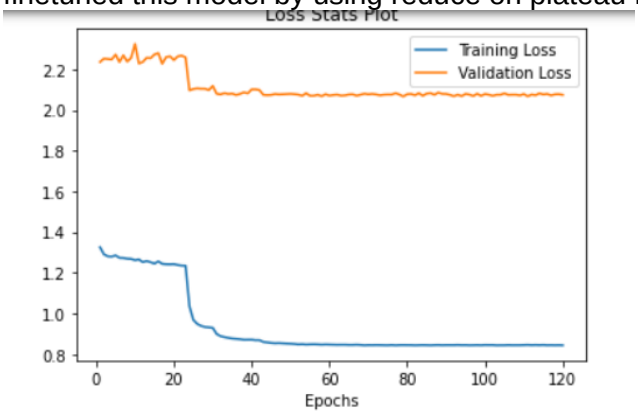
I choose patch_size = 4 and number of attention heads = 16.
I first used constant lr = 3e-4, with a linear warmup epochs = 5.




```
0%|          | 0/79 [00:00<?, ?it/s]

Accuracy: 59.4%
```

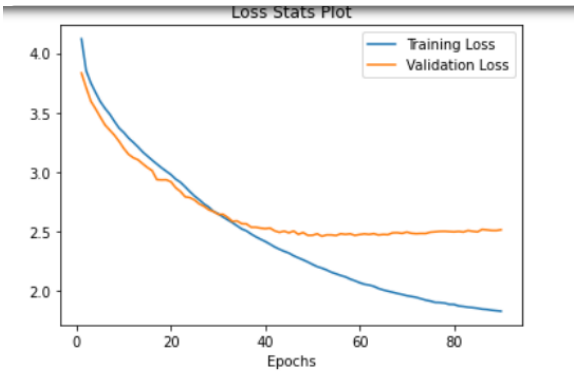
This performance is actually good in general. The test Accuracy is 59.4% Then I further finetuned this model by using reduce on plateau for learning rate




```
0%|          | 0/79 [00:00<?, ?it/s]

Accuracy: 63.9%
```

Here the x-axis is actually not started from 0, it should be started from 120 epochs. It achieved the best test accuracy of 63.9%.
I then tried lots of other learning rate schedulers, but none of them beats this result.
For example, as recommended to use a linear decay learning rate scheduler. I started with lr=1e-3 for 20 epochs and then learning rate linear decaying with a factor of 0.95. The result is shown below.

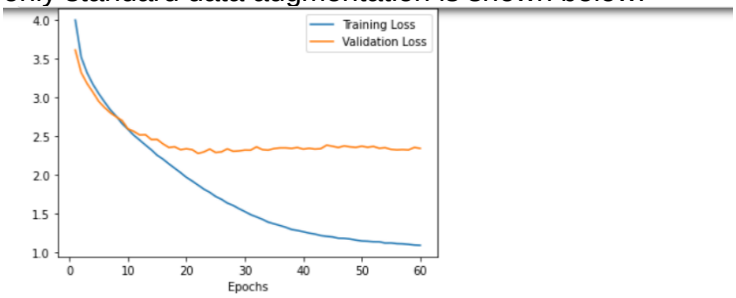



```
0%|          | 0/79 [00:00<?, ?it/s]
Accuracy: 50.5%
```

I also tried to use patch_size = 8, which is really big considering the image size is 32. And it gives worse test accuracy.
I tried to use the number of attention heads = 10 instead of 16. It gives comparable results.

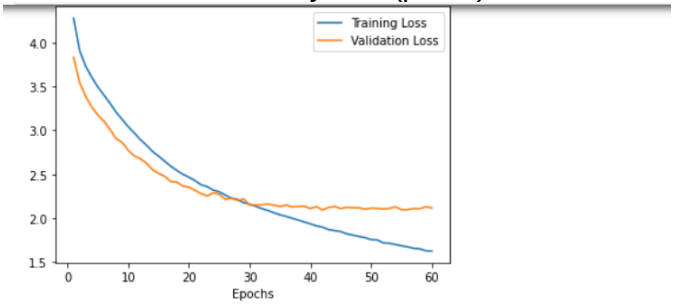
Investigate ColorJitter and Grayscale

For simplicity, I fixed patch_size = 4, number of attention heads=10. The learning rate is 3e-4, and training for 60 epochs. So without further training using reduce on plateau. The result using only standard data augmentation is shown below.



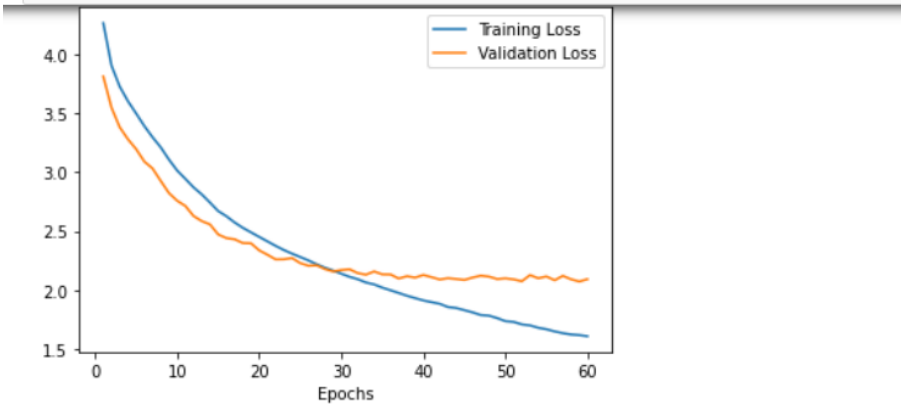

```
100% 79/79 [00:01<00:00, 79.86it/s]
Accuracy: 57.3%
```

After applying data augmentation transforms.ColorJitter(0.4, 0.4, 0.2, 0.1) and transforms.RandomGrayscale(p=0.2). The test accuracy significantly improved.




```
100% 79/79 [00:01<00:00, 79.40it/s]
Accuracy: 61.6%
```

Furthermore, I actually used torch.cuda.amp gradscaler and autocast to mix float16 and float32 for improving training speed through every model. But actually, as mentioned, it sacrificed some of the accuracy. When using only float32, the test accuracy is 62.2%.



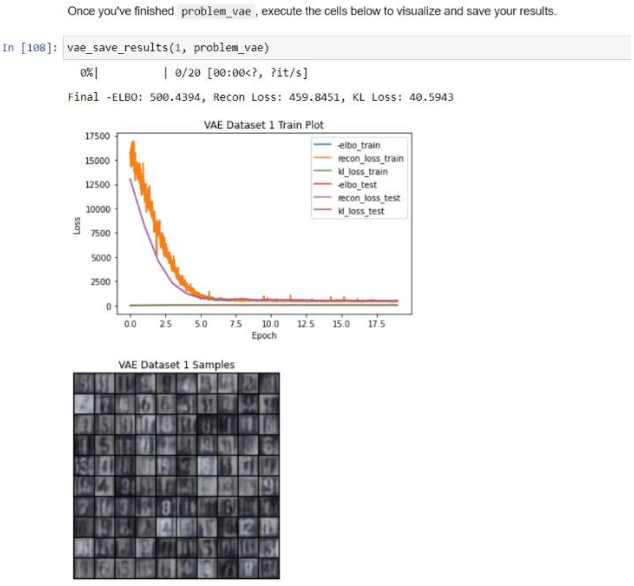
However, when using the automatic mixed precision package, training one epoch takes about 17 seconds. When using only float32, it takes 24 seconds to train one epoch. I'm glad I found this result. But for gradient stability, float32 is strongly important.

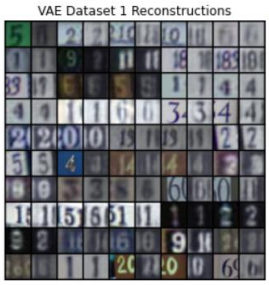
Instance Segmentation

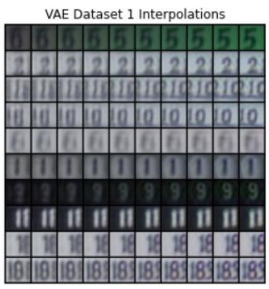
I chose to implement the DETR architecture. To utilize the pycocotools for conveniently loading the dataset, bounding boxes, and evaluating the final results, I used bop toolkit to transform bop dataset into the coco dataset convention. I finished the implementation with a reference on the facebook research.(Details inside "models" folder). However, even training one epoch could took several hours, due to the reason the dataset is very big the network is slow. So I decided not to wait for the network and didn't train.

Variational AutoEncoder

The result of VAE is presented below, you can go to the VAE_GAN notebook for more detail

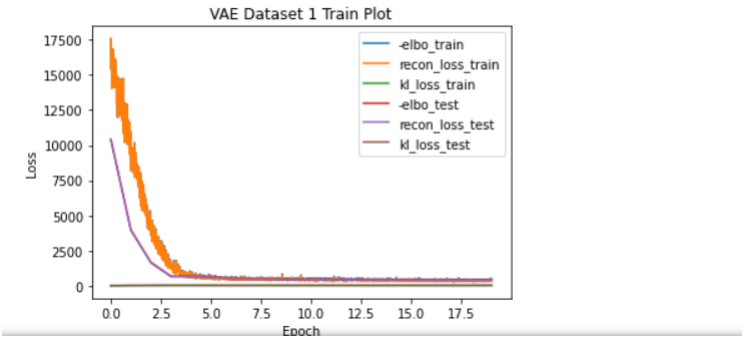


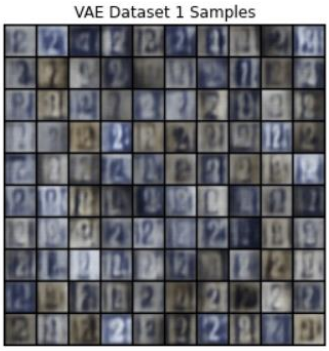


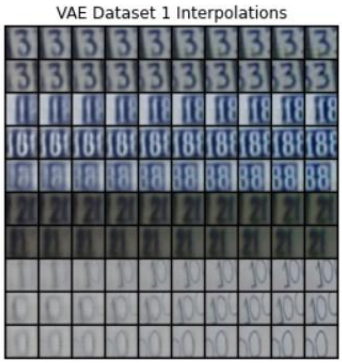


Conditional VAE

```
Final -ELBO: 436.1429, Recon Loss: 397.9991, KL Loss: 38.1438
```

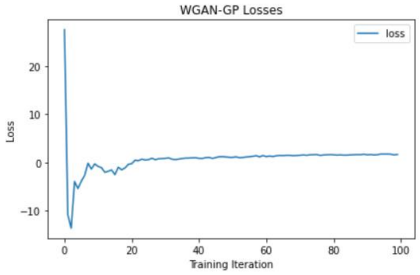




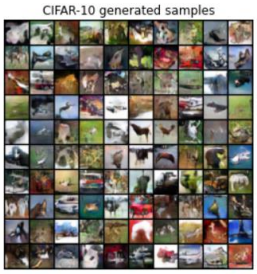


WGAN-GP with SNGAN-like Architecture

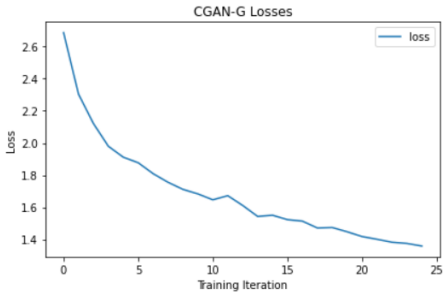
```
100% 100/100 [2:49:24<00:00, 99.84s/it]
.........Inception score: 7.178294
```

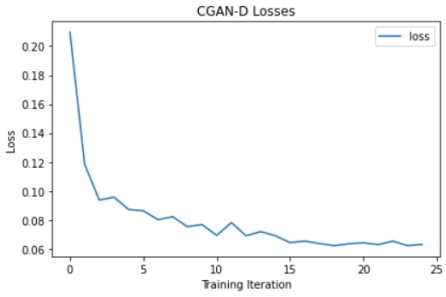



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```



CycleGAN






```
Clipping input data to the valid range for imshow with RGB data ([0..1]
```

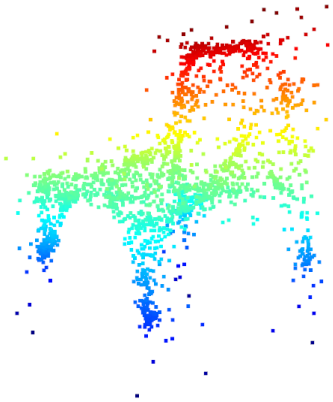

```
Clipping input data to the valid range for imshow with RGB data ([0..1]
```

PointNetGAN

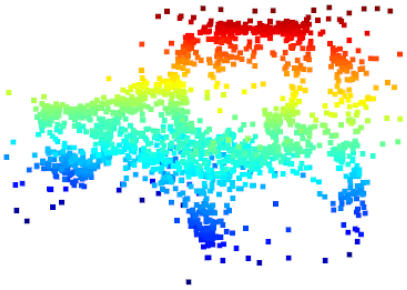
I implemented the WGAN-GP version of the GAN and then used PointNet-Mix by concatenating the max-pooling features and the average pooling features as a Discriminator. The generator is a simple MLP as paper discussed, the researching focus should be put into the Discriminator. The metric I used is the Chamfer distance since it's differentiable and very efficient.

Here I illustrate some examples generated after training only for **30** epochs. The results are not so good. The paper originally trained for 2000 epochs so I think the model still need further more training which takes much more time. There might also be other mistakes I might make for the network.

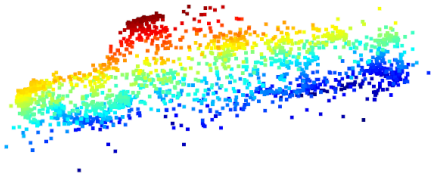
Chair



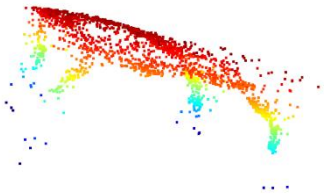
Wider Chair



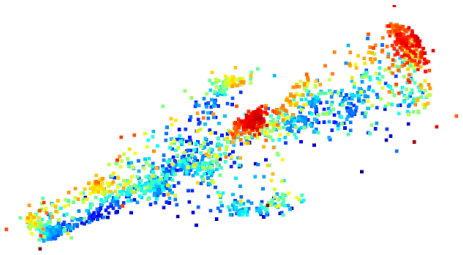
Car



Table

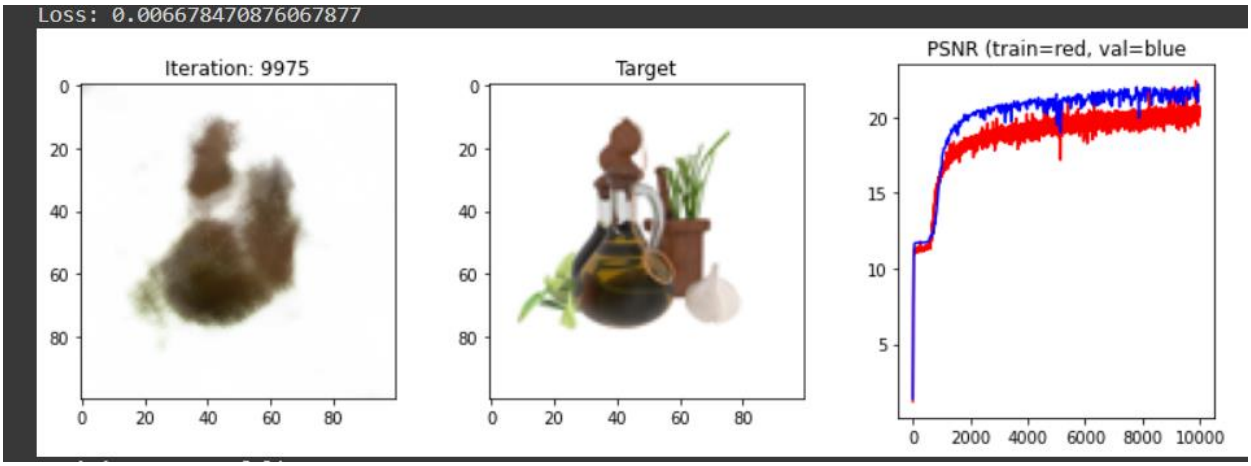


Air plane



NeRF

I trained nerf on the bottles dataset using only the 100 training images with a resolution of 100 * 100 due to my GPU memory consideration since I'm using google colab. The batch size is 2 **14 and the chunk size is 2**14. Below is a visualization of the predicted RGB image and the PSNR training. The result is clearly not so good.



I tried to using more epochs for training, but the google colab automatically shut down after 15000 epochs.