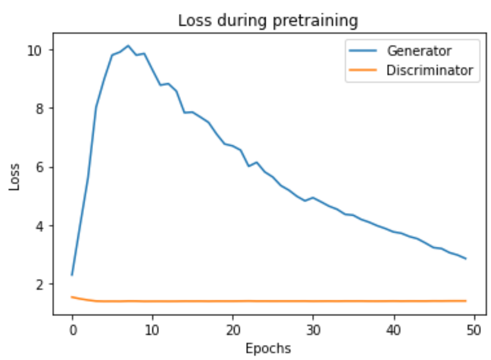
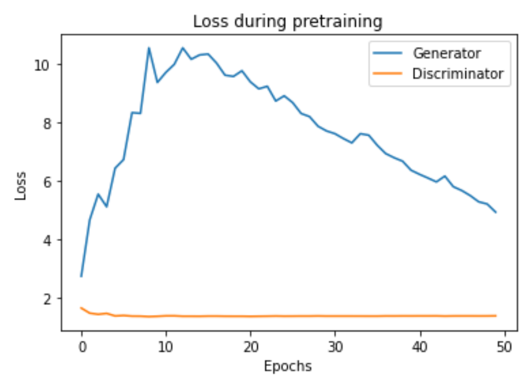
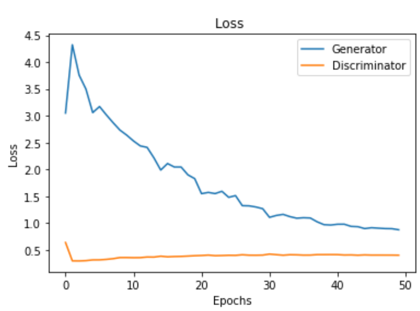
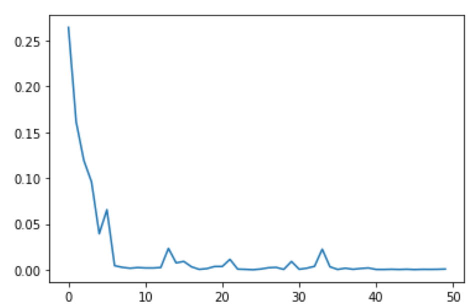
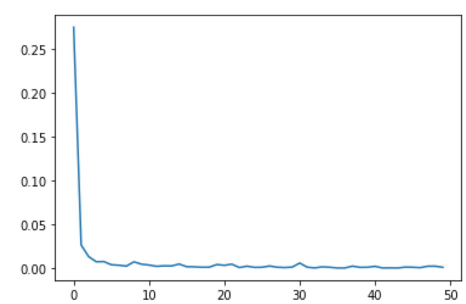
# Triplet GAN Network

Pre training is done in unsupervised manner using feature matching loss for generator and unsupervised loss for Discriminator:

| Loss plots during pre-training | |
|---|---|
| M = 16 | M = 32 |
|  |  |

During training triplet loss is added to the total loss of discriminator.

| | Loss plots during training | | |
|---|---|---|---|
| | **Generator and Discriminator** | **Unsupervised** | **Triplet** |
| M = 32 N = 100 |  |  |  |

| | N = 100, M = 16 | N = 200, M = 16 | N = 100, M = 32 |
|---|---|---|---|
| **Accuracy(%)** | 86.94 | 89.54 | 94.56 |
| **mAP** | 0.65 | 0.73 | 0.86 |

**Training Techniques :**

1. Training Generator and Discriminator alternatively. Did not train generator twice even if generator loss was high ( Accuracy - 76%)

2. Trained Generator twice for every batch if the generator loss was greater than 1. (Accuracy 85%)

**Experiments:**
Varying batch size, architectures.

**Problem:** During pre-training, discriminator loss became 0.000 and generator loss kept increasing.Training had to be stopped.
**Solution** :The chosen architecture had no noise layers in discriminator. Also the input was not infused with noise. Making the learning hard is one way to stabilize the discriminator. Picked a completely different architecture. Added noise to input of discriminator. This stabilized the pre-training process.
**Problem:**:
Increase of generator loss forced me to do early stopping:
**Solution**:Loss in GAN training is unstable and should be allowed to run for a greater number of epochs for stabilising. When I let the model run for more epochs I found generator loss coming down.
**Problem:** Discriminator loss becomes very small. Generator loss on the other was quite high and increasing thus producing poor results.
**Solution** :When generator loss is greater than 1 then generator is trained again. Thus if a generator has loss greater than 1 in any batch, the generator is trained twice as compared to discriminator.

**References:**
**1)[https://github.com/maciejzieba/tripletGAN](https://github.com/maciejzieba/tripletGAN)**
**2)[https://github.com/Sleepychord/ImprovedGAN-pytorch](https://github.com/Sleepychord/ImprovedGAN-pytorch)**

**Intermediate results**

**97%+, 75%+, 85%+**

**Architecture used:**

Discriminator

```
class Discriminator(nn.Module):
    def __init__(self, input_dim = 28 ** 2, output_dim = 16):
        super(Discriminator, self).__init__()
        self.input_dim = input_dim
        self.layers = torch.nn.ModuleList([
            LinearWeightNorm(input_dim, 1000),
            LinearWeightNorm(1000, 500),
            LinearWeightNorm(500, 250),
            LinearWeightNorm(250, 250),
            LinearWeightNorm(250, 250)]
        )
        self.final = LinearWeightNorm(250, output_dim, weight_scale=1)


    def forward(self, x, feature = False):
        x = x.view(-1, self.input_dim).cuda()
        noise = torch.randn(x.size()) * 0.3 if self.training else torch.Tensor([0])
        noise = noise.cuda()
        x = x + Variable(noise)
        for i in range(len(self.layers)):
            m = self.layers[i]
            x_f = F.relu(m(x))
            noise = torch.randn(x_f.size()) * 0.5 if self.training else torch.Tensor([0])
            noise = noise.cuda()
            x = (x_f + Variable(noise))
        if feature:
            return x_f, self.final(x)
        return self.final(x)
```

Generator

```
class Generator(nn.Module):
    def __init__(self, z_dim, output_dim = 28 ** 2):
        super(Generator, self).__init__()
        self.z_dim = z_dim
        self.fc1 = nn.Linear(z_dim, 500, bias = False)
        self.bn1 = nn.BatchNorm1d(500, affine = False, eps=1e-6, momentum = 0.5)
        self.fc2 = nn.Linear(500, 500, bias = False)
        self.bn2 = nn.BatchNorm1d(500, affine = False, eps=1e-6, momentum = 0.5)
        self.fc3 = LinearWeightNorm(500, output_dim, weight_scale = 1)
        self.bn1_b = Parameter(torch.zeros(500))
        self.bn2_b = Parameter(torch.zeros(500))
        nn.init.xavier_uniform(self.fc1.weight)
        nn.init.xavier_uniform(self.fc2.weight)

    def forward(self, batch_size):
        x = Variable(torch.rand(batch_size, self.z_dim), requires_grad = False, volatile = not self.training)

        x = x.cuda()
        x = F.softplus(self.bn1(self.fc1(x)) + self.bn1_b)
        x = F.softplus(self.bn2(self.fc2(x)) + self.bn2_b)
        x = F.softplus(self.fc3(x))
        return x
```