6135 - Assignment 3 - Practical Report

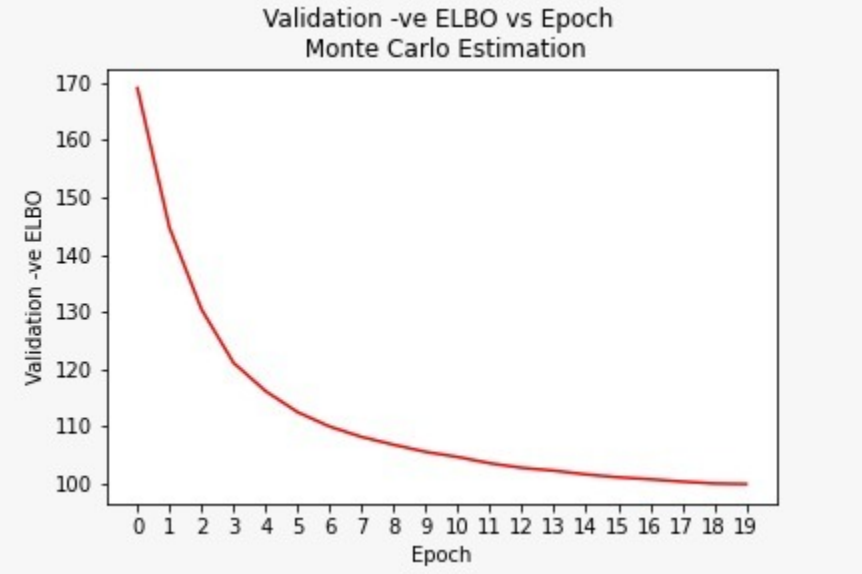
Problem 1

Answer 1.6:

- Train a VAE with the latent variable of 100 dimensions with an ADAM optimizer and 0.0003 learning rate for 20 epochs using the Monte Carlo estimate.
- The below is the ELBO on the validation set over epochs which is calculated as per the given equation,

$$\frac{1}{|\mathcal{D}_{\text{valid}}|} \sum_{\boldsymbol{x}_i \in \mathcal{D}_{\text{valid}}} \mathcal{L}_{\text{ELBO}}(\boldsymbol{x}_i) \geq -102$$

- For this, I trained the model which was given in the vae.ipynb file. For 20 epochs, passing each datapoint to vae module we get z_mean, z_logvar, x_mean which are then used to compute loss, and then the loss is passed backward and a step of the optimizer. After that, -ve ELBO is computed.
- We can infer that negative ELBO ranges from 100 to 170 with the set configuration, and the final value for ELBO after 20 epochs is -100.69.



Answer 1.7:

- Evaluate the log-likelihood of the trained VAE models by using importance sampling as per the below formula

$$\log p(\boldsymbol{x} = \boldsymbol{x}_i) \approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(\boldsymbol{x} = \boldsymbol{x}_i | \boldsymbol{z}_i^{(k)})\ p(\boldsymbol{z} = \boldsymbol{z}_i^{(k)})}{q_\phi(\boldsymbol{z} = \boldsymbol{z}_i^{(k)} | \boldsymbol{x}_i)}; \quad \text{for all } k\text{: } \boldsymbol{z}_i^{(k)} \sim q_\phi(\boldsymbol{z} | \boldsymbol{x}_i)$$

$$\text{and } \boldsymbol{x}_i \in \mathcal{D}.$$

- For this, I trained the model which was given in the vae.ipynb file. For each test data point, we find a sample from the posterior, Decode the samples, Reshape images and posterior to evaluate probabilities, Calculate all the probabilities, and then recombine accumulate them.
- The value of log(p(x)) received with this procedure is -95.52 which is quite similar to the analytical solution result which was -95.29.
- Running MC estimation is computationally heavy as compared to the analytical solution.

Problem 2

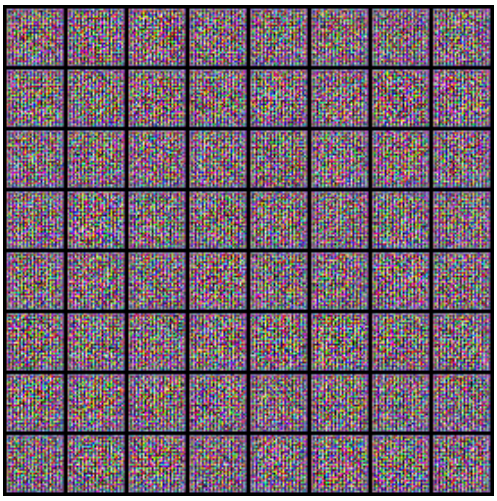
Answer 2.2:

- The below samples are generated by implementing GAN using Wasserstein distance and Lipchitz Penalty between two points x and y (real and generated samples respectively) using the below formulation:

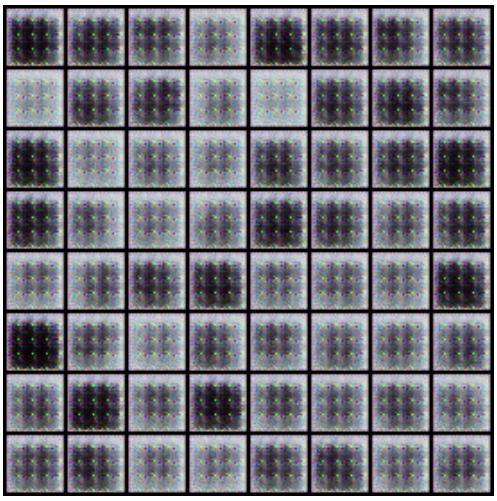
$$\mathbb{E}_{y\sim\nu}[f(y)] - \mathbb{E}_{x\sim\mu}[f(x)] + \lambda\mathbb{E}_{\hat{x}\sim\tau}[(\max\{0, \|\nabla f(\hat{x})\| - 1\}]^2$$

$$\text{where } \hat{x} = tx + (1-t)y \text{ for } t \sim U[0,1].$$

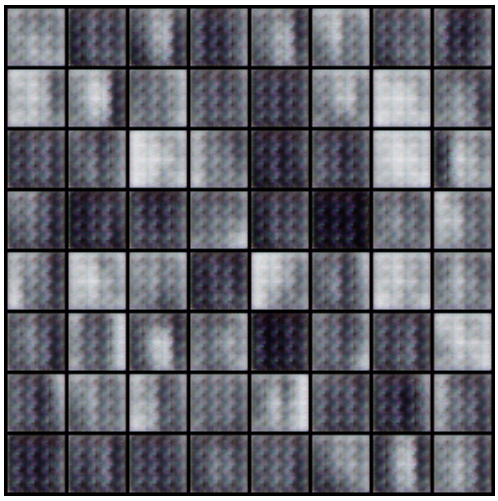
- Visual samples of 64 GAN generated images using Wasserstein distance on Street View House dataset (SVHN) for 50k epochs.



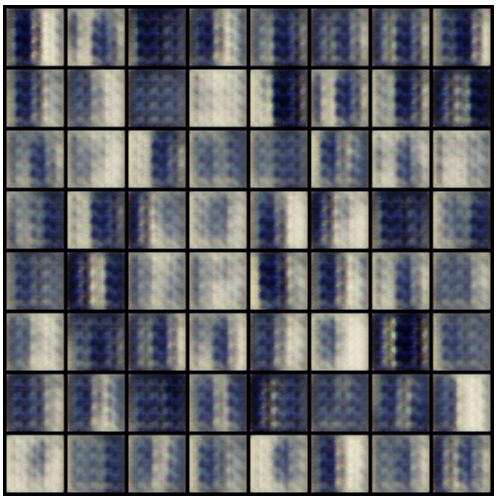
Epoch 0



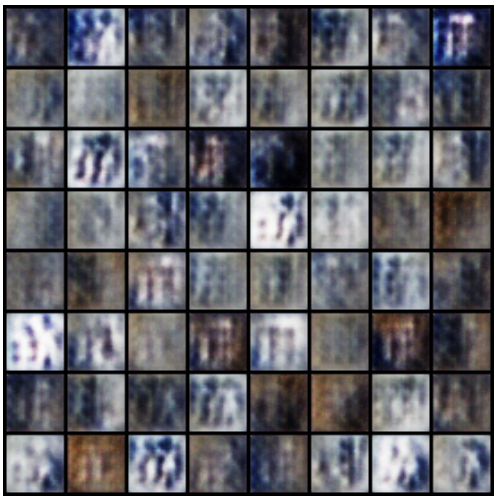
Epoch 100



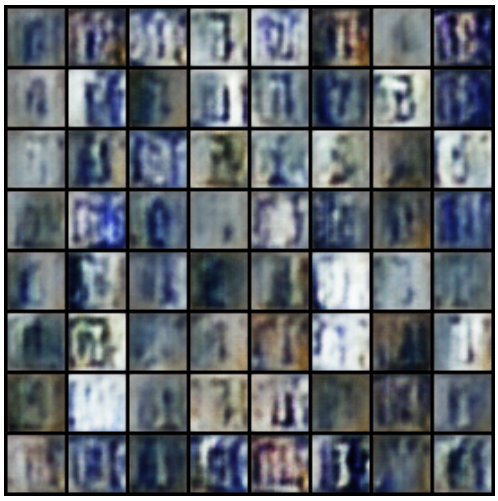
Epoch 500



Epoch 1000

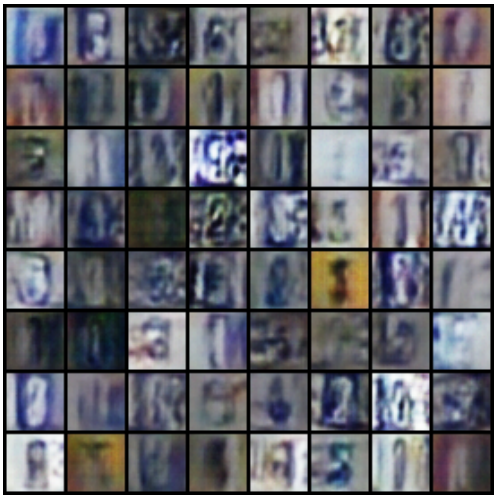


Epoch 2500



Epoch 5000

Epoch 10000



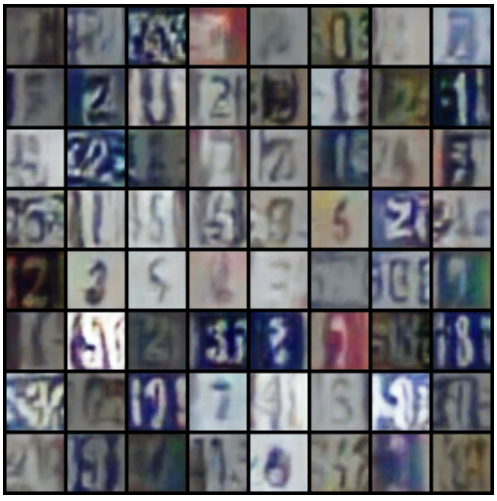
Epoch 15000

Epoch 20000

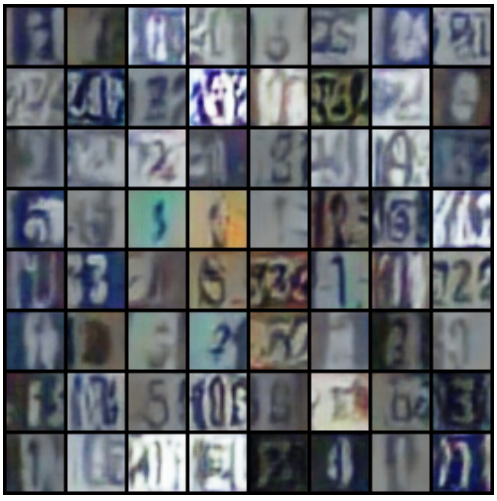
Epoch 25000

Epoch 30000

Epoch 35000



Epoch 40000



Epoch 45000

Epoch 49900

- If we look at the generated images above, during the initial 1000 iterations, the generator is still in the learning phase and has generated mostly blurry and less diverse noisy images which are not much distinguishable from each other.
- As we go further in the training, after 2k iterations, the generator starts understanding the basic structures and strokes of digits similar to SVHN dataset. The images are still blurry up to 10k epochs.

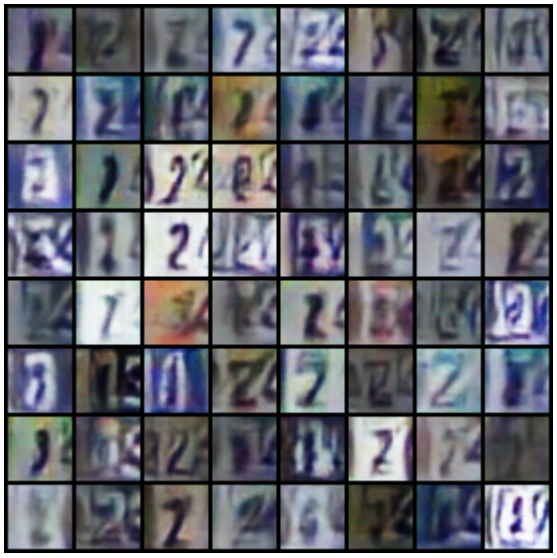
- Post that, we can see less blurry images which are distinguishable from each other have started generating. We can see images of numbers 3, 4, 2 here.
- After around 30k epochs, double and triple-digit numbers like 25, 18, 11 are also present and towards the end of training their resolution is getting increased and edges are getting sharper in various color combinations.

Answer 2.3:

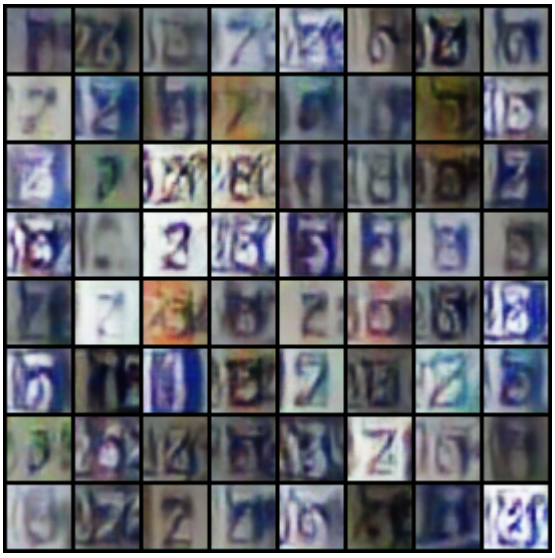
- Below are 64 randomly generated images for the different dimensions where a small epsilon (1e1) is added.
- We can notice that some of the images are clearly distinguishable from others while some are blurry and noisy which are hard to decode.
- We can clearly see the transition of numbers happening in the below images. From dimension 0 to 99, the image in the 4th row, 3rd column is initially blurry then it becomes digit 3 at dimension 10 and digit 2 at dimension 20. As we go further, it becomes number 9 and then 2 again.
- We can notice the transition in colors as well in some the images (i.e. 2nd row 4th column)

dimension = 0

dimension = 10

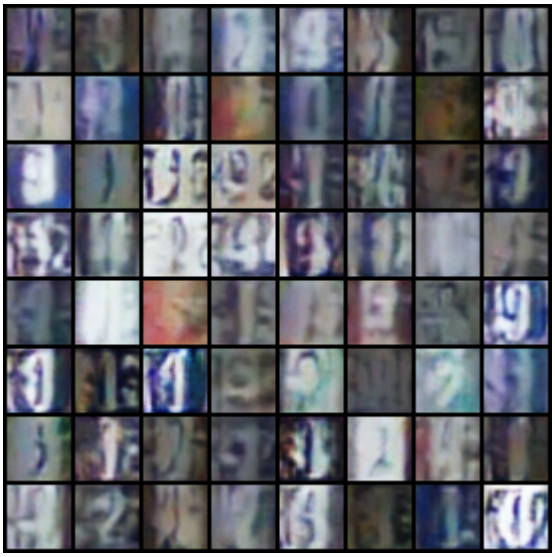


dimension = 20



dimension = 30

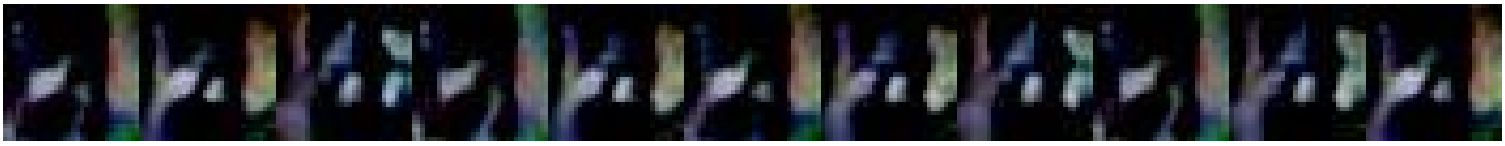
dimension = 80



dimension = 99

Answer 2.4.a:

- Compare interpolating in the data space and in the latent space for the values of alpha from 0, 0.1, 0.2, … 1. Here, The generated interpolation is passed to the generator to get x_alpha



- The above-generated images for the different values of alphas seem to be mostly noise which blurry edges representing the digit 2. Going from small alpha values like 0 or 0.5 to higher values such as 0.9 and 1, the generated images gets a bit sharper and more clear.
- Below are a few more samples with the above configuration to get an idea of how the digits are getting generated. We can clearly notice the transition between digits and colors here.

alpha = 0.1

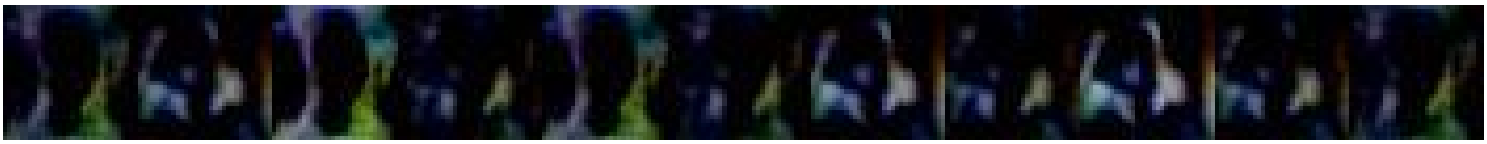
alpha = 0.5

alpha = 0.8

alpha = 1.0

Answer 2.4.b:

- Compare interpolating in the data space and in the latent space for the values of alpha from 0, 0.1, 0.2, … 1.
- The generated z0 and z1 from random distribution are passed to the generator to get x0 and x1



- The above-generated images for the different values of alphas seem to be mostly noise which blurry edges representing straight lines that resemble digit 1 initially and then slowly transition into digit 4. The third last image, alpha = 0.8 has a very clear image of 4.
- The below are a few more samples with the above configuration to get an idea of how the digits are getting generated.

alpha = 0.1

alpha = 0.5

alpha = 0.8

alpha = 1.0

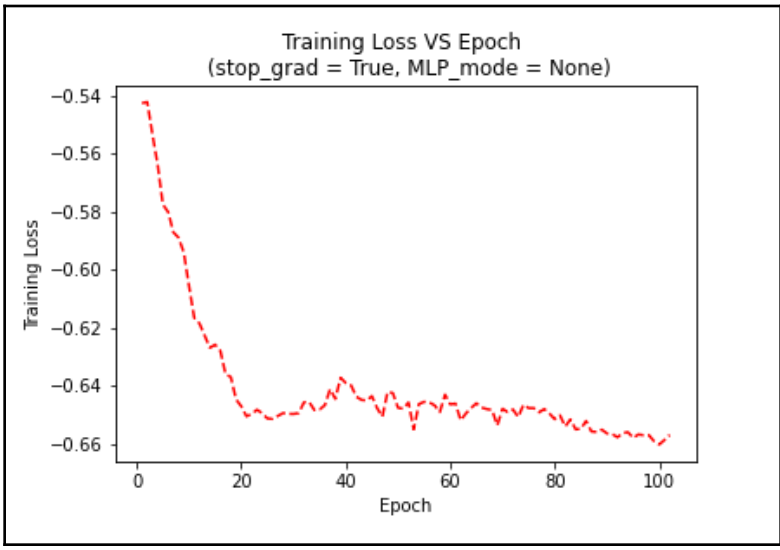
Overall, if we look at the output of 2.4.a and 2.4.b generated images,

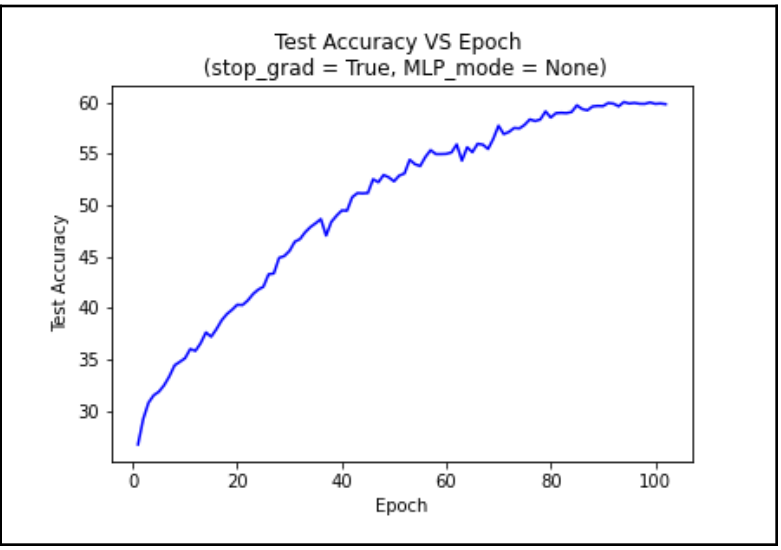
- When we are sampling 2 different images and then interpolating them in 2.4.b they transition from one to another and when alpha is near 0.5 it gives equal weightage to both the images resulting in an extremely noisy image whereas, as we go above or below this limit it gives more weightage to either of the samples and generate sharp images.
- The above phenomena are present in the case of 2.4.b but not in 2.4.a as there we are firstly interpolating 2 samples and then passing them to the generator once.
- Thus, in 2.4.b we see simply overlayed images that are less sharp as compared to 2.4.a.

Problem 3

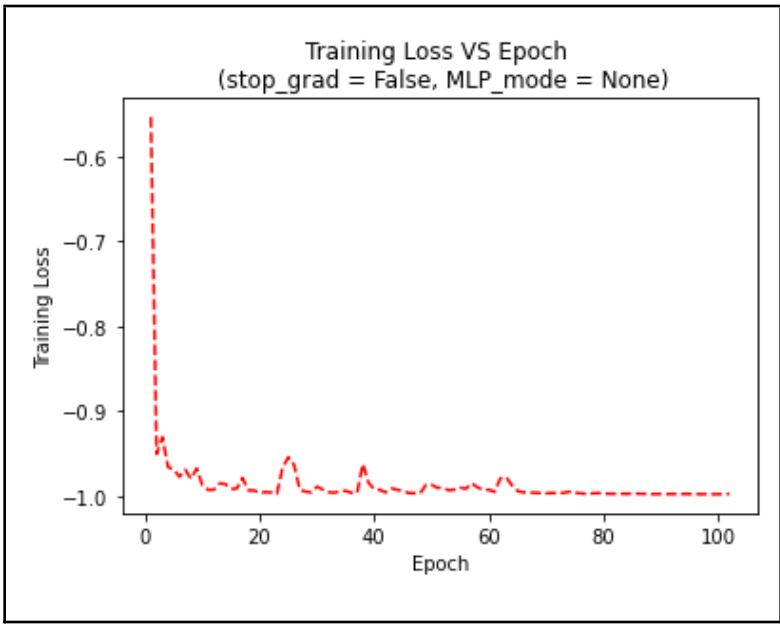
Answer 3.4:

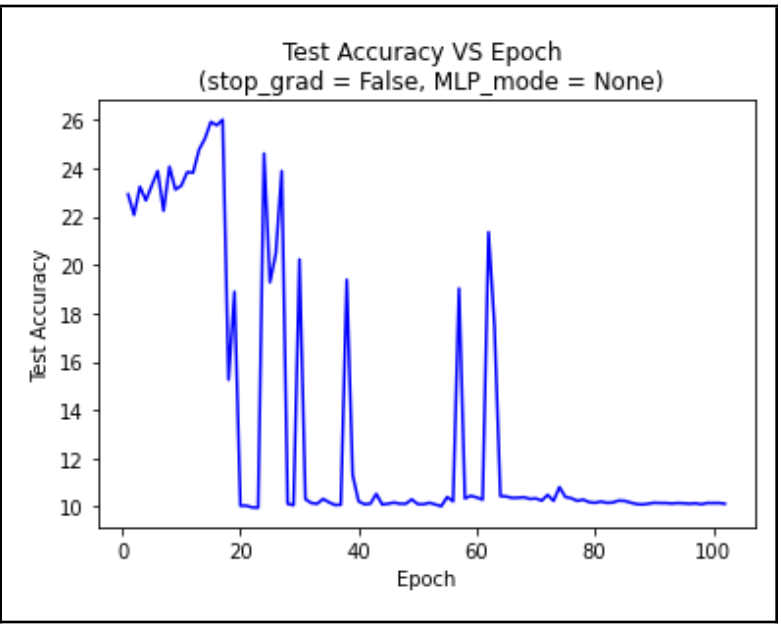
- Train model for 100 epochs **with gradient stopping**
- Training loss and Knn accuracy against training epochs are as follows.
- Interpretation:
- From the Training loss curve we can infer that it is in the range of -0.54 to -0.66 for 102 epochs. The loss is negative because we are using negative cosine similarity as a loss function.
- Overall, there is a slight decrease in the loss for 0-20 epochs and then it is almost the same.
- On the other hand, test accuracy is almost steadily increasing from 25% to 60% for the given configuration. We can conclude that the model is learning well.





- Train model for 100 epochs **without gradient stopping**
- Training loss and Knn accuracy against training epochs are as follows.
- Interpretation:
- As compared to the gradient stopping, where we do not use gradient stopping, the test accuracy is quite low. It fluctuates in the range of 10% to 26% and finally saturates at 10%.
- The train loss range is also high, loss magnitude starts from 0.6 and then goes up to 1.0. Here, we can not conclude that the model is learning well.





Answer 3.5:

- Train the model for 100 epochs while removing the predictor and replacing it with the identity mapping
- Training loss and Knn accuracy against training epochs are as follows.
- Interpretation:
- When we use MLP_mode as no_pred_mlp, in the q3_solution.py file we initialize predictor as an identity. When we use gradient stopping with this config, the test accuracy, again, similar to the last part is less. It's in the range of 10-26 % with a few spikes in the initial 40 epochs.
- The train loss range is also high, loss starts from -0.55 and then goes below up to -1.0. Here, we can not conclude that the model is learning well.