Representation Learning
Assignment 3 - Report
SimCLR + DDPM

Student Name: **Maryam Alipourhajiagha**

SimCLR

Transformers_Pool:

The transformations are applied to the unlabeled dataset (STL10) in the SimCLRDataset class to create augmented views of the images for contrastive learning and encourage the model to learn robust representations. I have applied the below tranformaions:

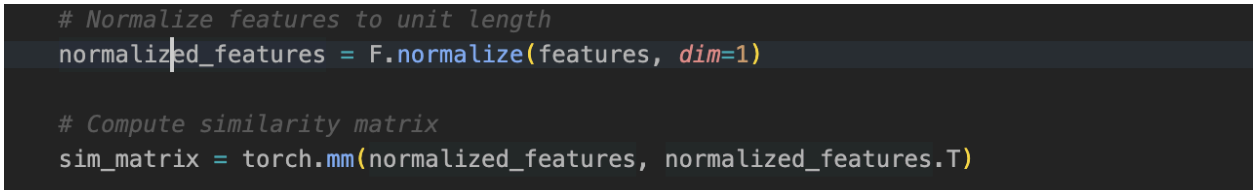
1. **RandomResizedCrop:** Randomly crops and resizes the image to a specified size. This helps the model learn features that are invariant to translation and scale changes.
2. **RandomHorizontalFlip:** Horizontally flips the image randomly. This augmentation increases the diversity of training samples and helps the model generalize better.
3. **GaussianBlur:** Applies Gaussian blur to the image with a randomly selected kernel size and sigma values. Blur augmentation smooths the image and encourages the model to focus on high-level features.
4. **ColorJitter:** Adjusts brightness, contrast, saturation, and hue of the image randomly. This augmentation introduces color variations, making the model invariant to color changes in the input data.
5. **RandomGrayscale:** Randomly converts the image to grayscale with a certain probability.
6. **ToTensor:** Converts the image to a PyTorch tensor, which is necessary for further processing.

Note: Due to insufficient resources, I changed the batch size to 2560.

SimCLR Loss:

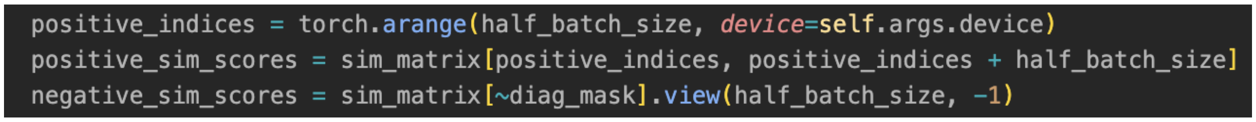
To implement the SimCLR loss function, I followed the algorithm described in the original paper. Here's how my implementation aligns with the paper's formulation:

1. **Similarity Matrix Calculation:** I compute the similarity matrix using the dot product of normalized feature vectors: $S_{ij} = \frac{Z_i^T Z_j}{||Z_i||\ ||Z_j||}$, where Z_i and Z_j are feature vectors.



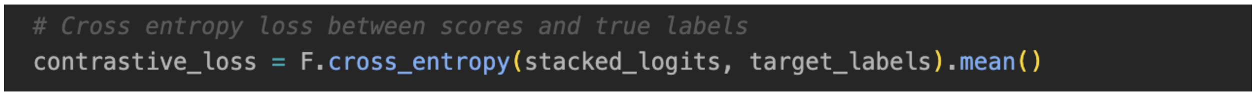
```
# Normalize features to unit length
normalized_features = F.normalize(features, dim=1)

# Compute similarity matrix
sim_matrix = torch.mm(normalized_features, normalized_features.T)
```

2. **Logit Scores:** Logit scores for positive and negative pairs are calculated using the similarity matrix. Positive scores are obtained for pairs $(2k - 1, 2k)$, and negative scores are obtained for all other pairs. (diag_mask filters out self-positives from the similarity matrix)



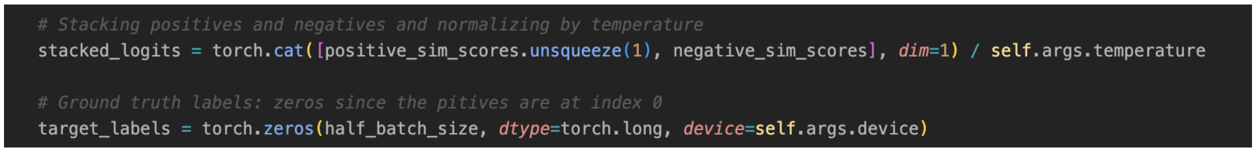
```
positive_indices = torch.arange(half_batch_size, device=self.args.device)
positive_sim_scores = sim_matrix[positive_indices, positive_indices + half_batch_size]
negative_sim_scores = sim_matrix[~diag_mask].view(half_batch_size, -1)
```

3. **Loss Computation:** I compute the loss function using the provided formula. The final loss L is the average of losses for positive pairs $(2k - 1, 2k)$ and $(2k, 2k - 1)$.



```
# Cross entropy loss between scores and true labels
contrastive_loss = F.cross_entropy(stacked_logits, target_labels).mean()
```

4. **Ground Truth Labels:** To prepare for calculating the loss with cross-entropy, I first stack the logit scores for positive and negative pairs together, ensuring each sample has corresponding logits for both types of pairs. Next, I generate ground truth labels where positives are assigned a label of 0 and negatives have labels greater than 0, aligning with the model's predicted logits.



```
# Stacking positives and negatives and normalizing by temperature
stacked_logits = torch.cat([positive_sim_scores.unsqueeze(1), negative_sim_scores], dim=1) / self.args.temperature

# Ground truth labels: zeros since the pitives are at index 0
target_labels = torch.zeros(half_batch_size, dtype=torch.long, device=self.args.device)
```

This implementation is equivalent to the loss presented in the class:

$$\ell_{i,j} = -\log \frac{\exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)},$$

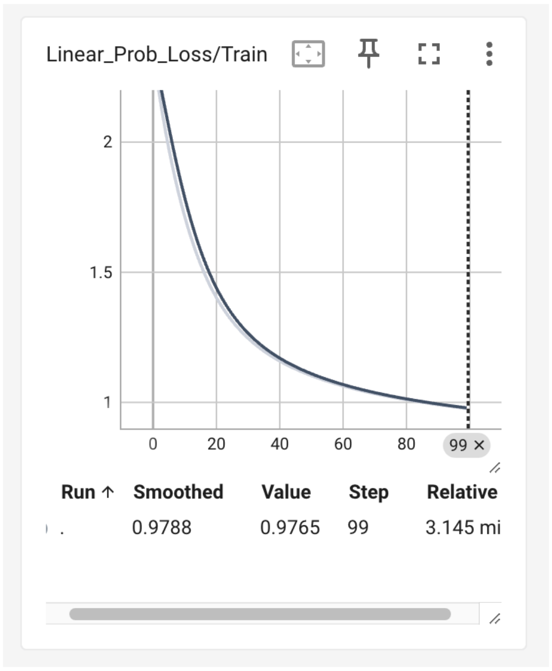
where $\mathrm{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \boldsymbol{v} / \|\boldsymbol{u}\| \|\boldsymbol{v}\|$

Computational efficiency:

My implementation leverages vectorized operations and PyTorch's optimized functions, such as matrix multiplication and normalization, to efficiently compute the similarity scores between pairs of feature vectors. These operations are inherently parallelizable and can effectively utilize GPU acceleration for faster computation. Additionally, the use of boolean masking with the diagonal mask efficiently filters out self-similarities from the similarity matrix, reducing unnecessary computations. Moreover, by utilizing the F.cross_entropy function, the loss computation benefits from optimized algorithms for calculating the cross-entropy loss, further enhancing computational efficiency.

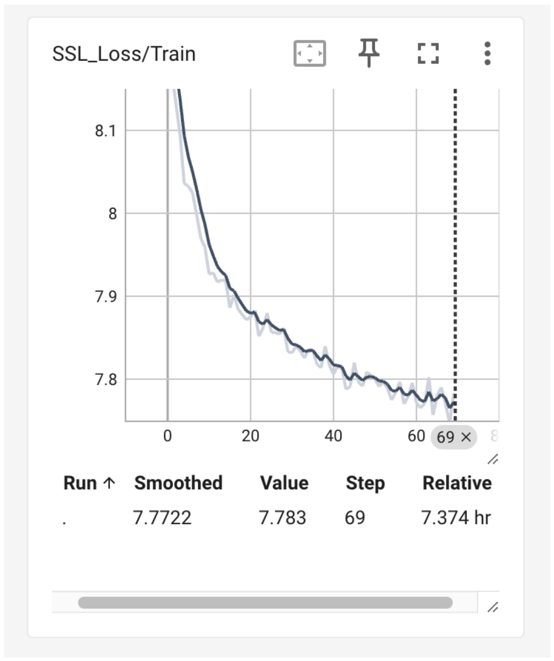
Learning Curves:

Downstream Task Train Loss:



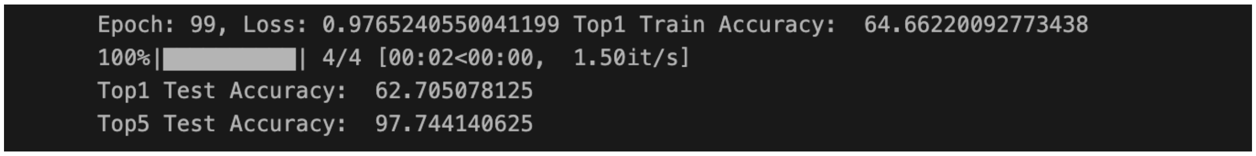
Run ↑	Smoothed	Value	Step	Relative
.	0.9788	0.9765	99	3.145 mi

SimCLR Train Loss:



Run ↑	Smoothed	Value	Step	Relative
.	7.7722	7.783	69	7.374 hr

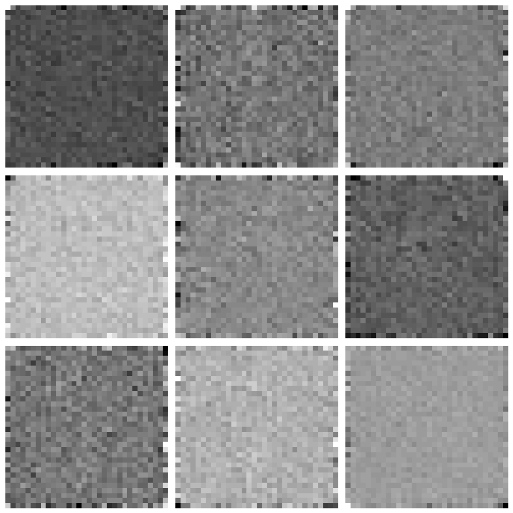
Downstream top1, top5 and Last Epoch Accuracy:



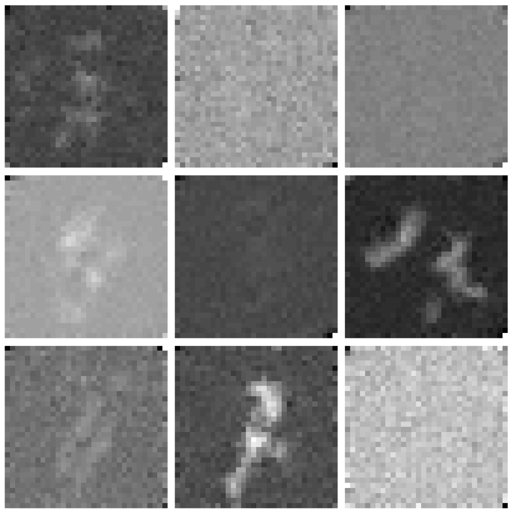
```
Epoch: 99, Loss: 0.9765240550041199 Top1 Train Accuracy:  64.66220092773438
100%|██████████| 4/4 [00:02<00:00,  1.50it/s]
Top1 Test Accuracy:  62.705078125
Top5 Test Accuracy:  97.744140625
```

DDPM

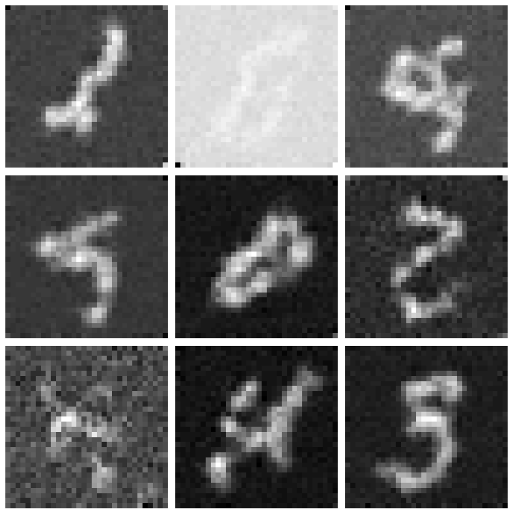
Epoch 1



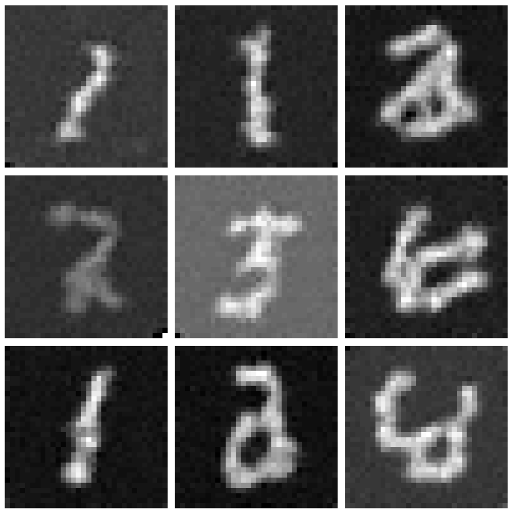
Epoch 2



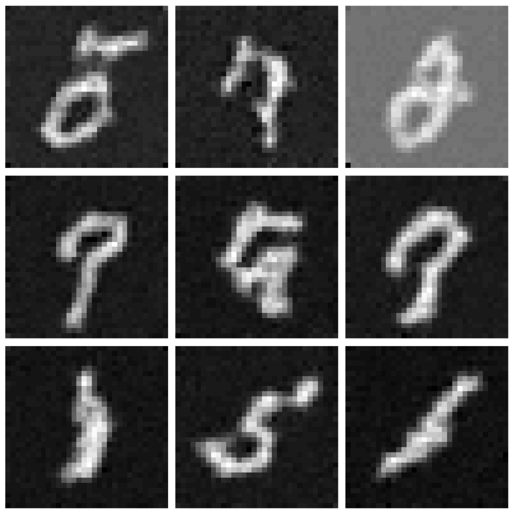
Epoch 3



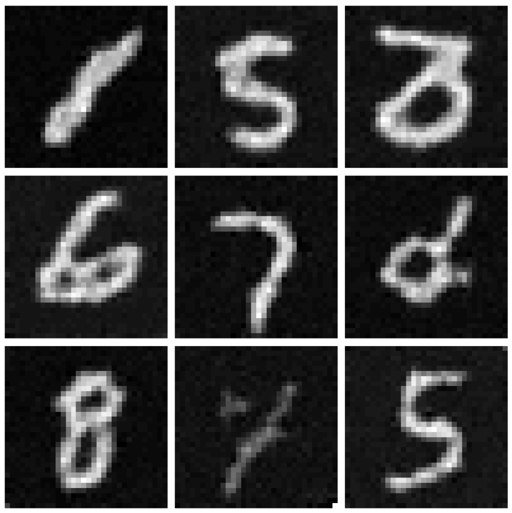
Epoch 4



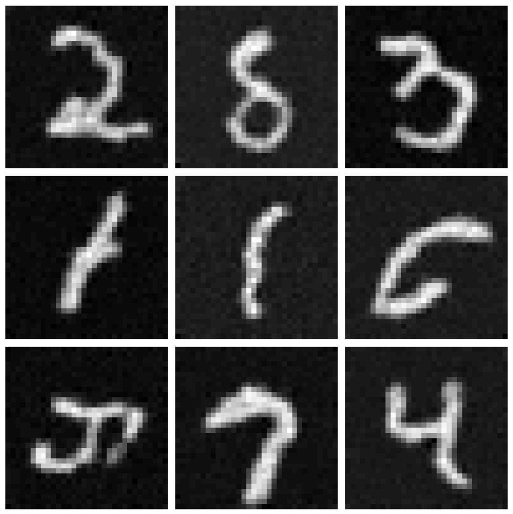
Epoch 5



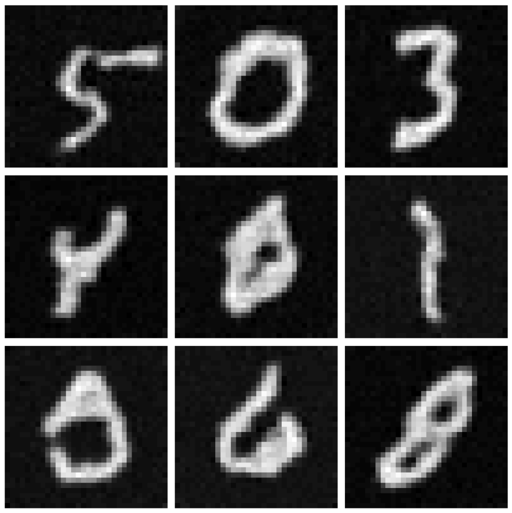
Epoch 6



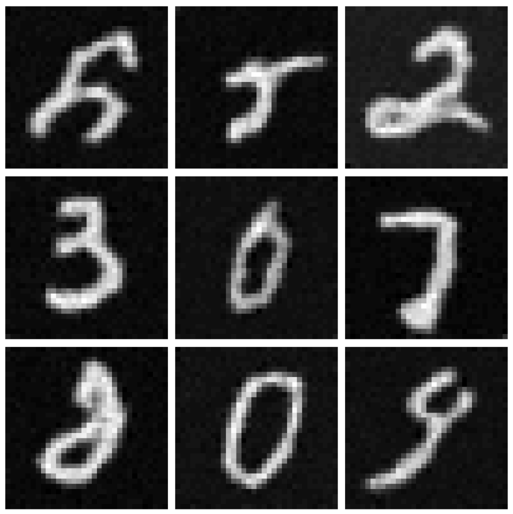
Epoch 7



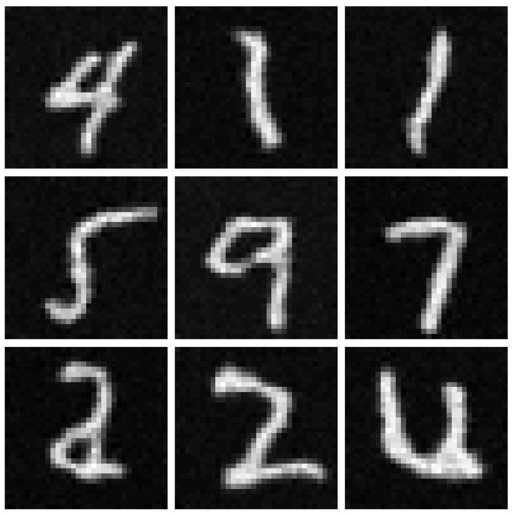
Epoch 8



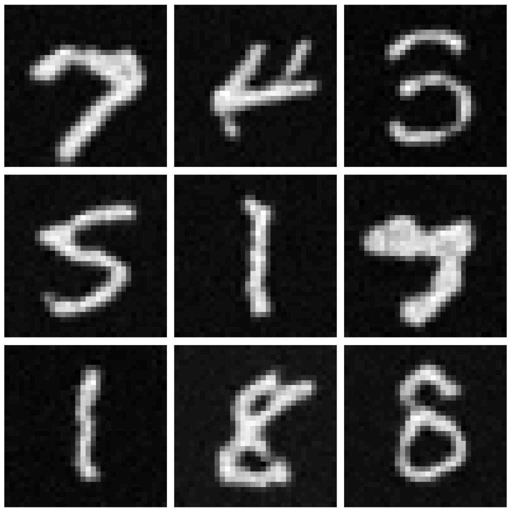
Epoch 9



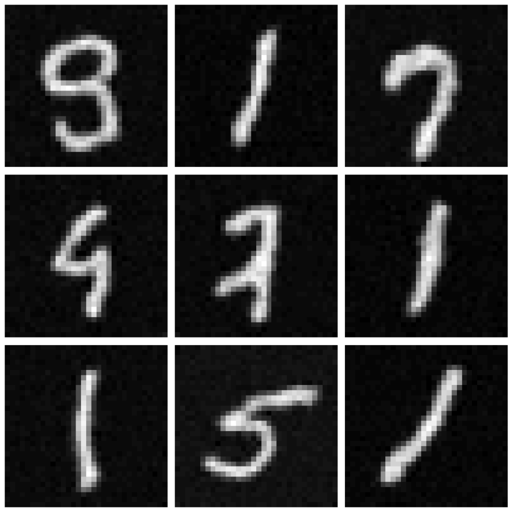
Epoch 10



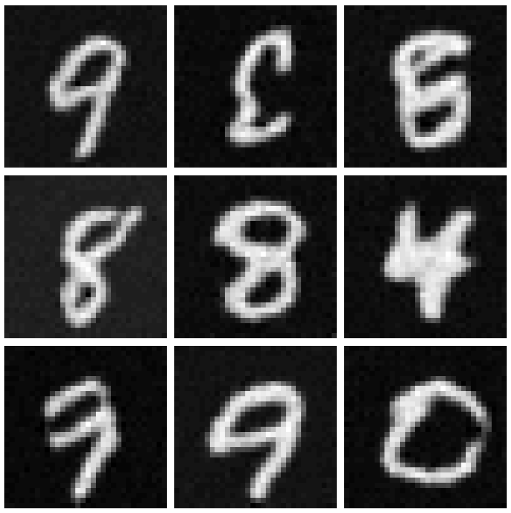
Epoch 11



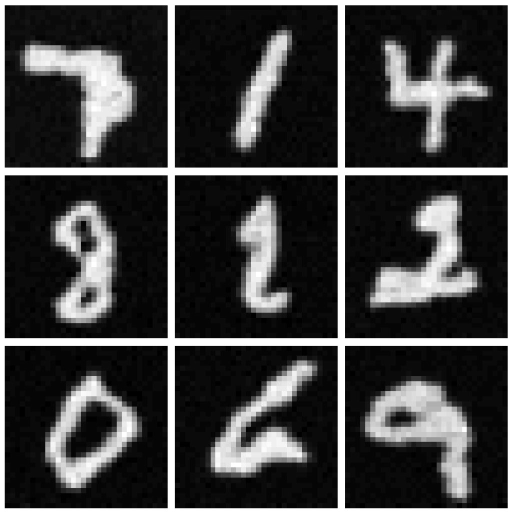
Epoch 12



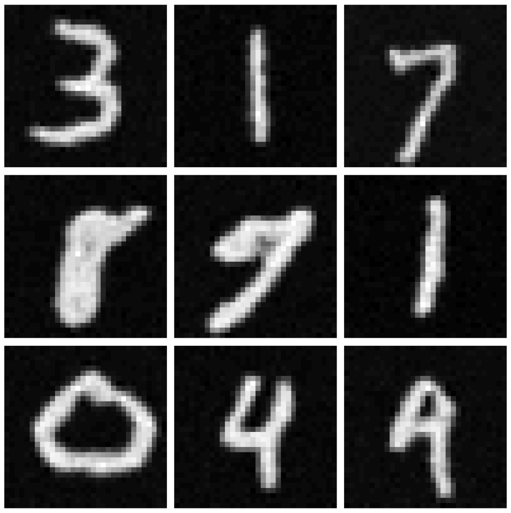
Epoch 13



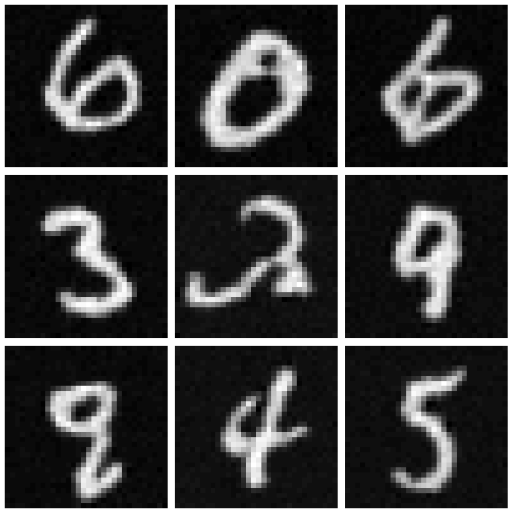
Epoch 14



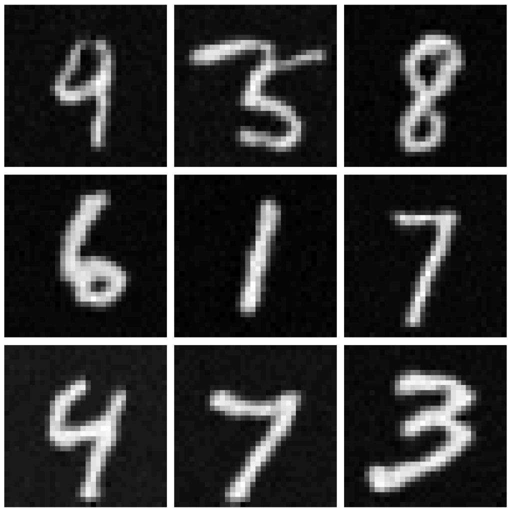
Epoch 15



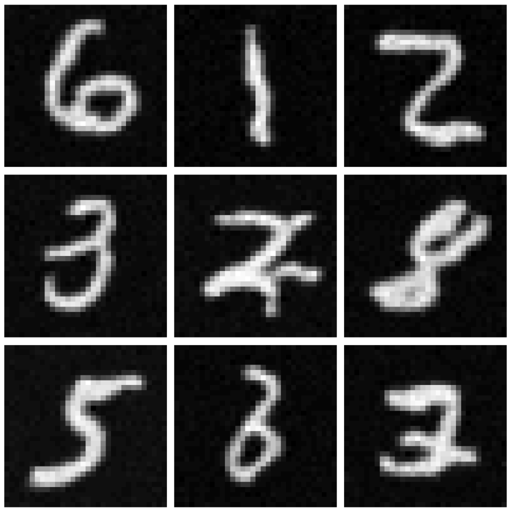
Epoch 16



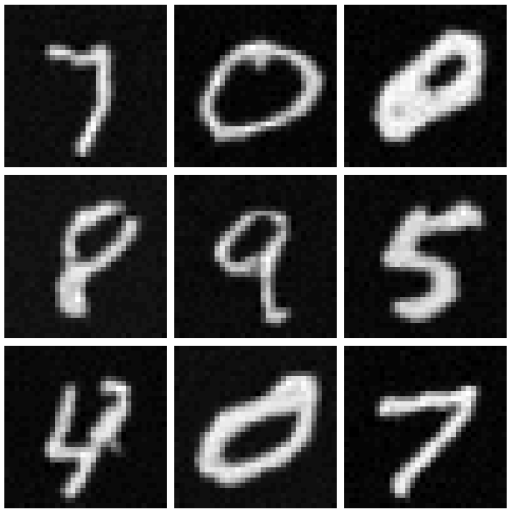
Epoch 17



Epoch 18



Epoch 19



Epoch 20