Deep Learning Project Report

Omer Trinin, Yonatan Azizi

April 2025

1 Reasoning

MNIST

Encoder

Layer	**Output Shape**	**Details**
Input	(1, 28, 28)	Grayscale image
Conv2d	(256, 28, 28)	Kernel: 7×7, Padding: 3, Stride: 1
BatchNorm2d	(256, 28, 28)	
ReLU	(256, 28, 28)	
Conv2d	(256, 28, 28)	Kernel: 7×7, Padding: 3, Stride: 1
BatchNorm2d	(256, 28, 28)	
ReLU	(256, 28, 28)	
MaxPool2d	(256, 14, 14)	Kernel: 2×2, Stride: 2
Conv2d	(1, 14, 14)	Kernel: 5×5, Padding: 2, Stride: 1
BatchNorm2d	(1, 14, 14)	
ReLU	(1, 14, 14)	
Flatten	(196,)	Converts $(1, 14, 14)$ to 196-D vector
Linear	(128,)	Fully connected layer
BatchNorm1d	(128,)	
ReLU	(128,)	

Table 1: Architecture of the encoder used across all MNIST tasks

- We used 3 layers according to your recommendations.
- We added an additional linear layer to reach 128 latent space.
- We experimented various kernel sizes and found out that kernel sizes of 7, 7 and 5 give the best results on the validation set with task 1.2.2. This architecture provides large receptive fields, despite the small number of layers, which turned out to be useful for MNIST and CIFAR10 classification.
- The number of channels is increased by the first layer from 1 to 256, kept as 256 by the second layer and reduced to 1 by the third layer. The experiments showed that this gave better results than a smaller number of channels. Thus it turned out that the usage of 256 channels was needed for capturing sufficiently complex features. The reduction into 1 channels at the last layer is needed to reach small latent space.
- Each layer includes BatchNorm2d which helps to speed up the learning and to prevent overfitting.
- A MaxPool2d was used in the second layer to reduces spatial resolution from 28×28 to 14×14. MaxPool2d helps the network to be less sensitive to small distortions in the input.
- We tried to use Dropout as well, but we decided to use it only in the classifier, after we saw it doesn't improve the results when added to the encoder.

Decoder

Layer	**Output Shape**	**Details**
Input	(128,)	Latent vector
Linear	(196,)	Fully connected layer
BatchNorm1d	(196,)	
ReLU	(196,)	
Unflatten	(1, 14, 14)	Reshape to image format
ConvTranspose2d	(256, 28, 28)	Kernel: 7×7, Padding: 3, Stride: 2, Output padding: 1
BatchNorm2d	(256, 28, 28)	
ReLU	(256, 28, 28)	
ConvTranspose2d	(256, 28, 28)	Kernel: 7×7, Padding: 3, Stride: 1
BatchNorm2d	(256, 28, 28)	
ReLU	(256, 28, 28)	
ConvTranspose2d	(1, 28, 28)	Kernel: 5×5, Padding: 2, Stride: 1
Sigmoid	(1, 28, 28)	Normalize output to $[0, 1]$

Table 2: Decoder architecture used in the reconstruction section

- The decoder is basically the inverse of the encoder.
- Additionally, we added a Sigmoid to normalize the output into $[0, 1]$.
- We experimented Dropout but it didn't improve the results.
- We experimented both ConvTranspose2d as well as Upsample with different modes (nearest, linear, bilinear). We chose ConvTranspose2d as it gave better results.

Classifier

Layer	Output Shape	Details
Input	(128,)	Latent vector
Linear	(512,)	Fully connected layer
BatchNorm1d	(512,)	
ReLU	(512,)	Activation function
Dropout	(512,)	Dropout ($p = 0.1$)
Linear	(256,)	Fully connected layer
BatchNorm1d	(256,)	
ReLU	(256,)	Activation function
Dropout	(256,)	Dropout ($p = 0.1$)
Linear	(128,)	Fully connected layer
BatchNorm1d	(128,)	
ReLU	(128,)	Activation function
Dropout	(128,)	Dropout ($p = 0.1$)
Linear	(10,)	Output logits for 10 classes

Table 3: Architecture of the classifier used across all MNIST tasks

- The classifier is an MLP with 3 fully connected layers and a final linear layer.
- The layer size is first increased from 128 to 512 and then reduced to 256 and then to 128. The first layer expansion allows the network to learn a more comprehensive set of features. The dimensionality reduction of next layers refines the learned features, makes the representation more compact, as preparation for the final classification.
- Additionally, we added a final linear layer that reduces the output to a 10-dimensional vector needed for the 10 output classes.
- Similarly to the encoder, each layer includes BatchNorm2d.
- Each layer also includes a Dropout layer. The Dropout forces the model to learn more robust features and prevents overfitting.

Contrastive projection head

Layer	Output Shape	Details
Input	(128,)	Latent vector
Linear	(128,)	Fully connected layer
ReLU	(128,)	Activation function
Linear	(128,)	Fully connected layer

Table 4: Projection head architecture used for contrastive learning

- We used for the projection head an architecture similar to the original SimCLR [1]

Contrastive loss function

We used the NT-Xnet (Normalized Temperature-scaled Cross Entropy Loss) similarly to the original SimCLR [1]. The temperature was set to 0.5.

Learning method

We used checkpoints and parameters of the model according to the validation dataset.

Task 1.2.1 learning algorithm

- We used Adam optimizer.
- We experimented various batch sizes and selected batch size of 64.

Task 1.2.2 learning algorithm

- We experimented Adam with various batch size.
- We choose batch size of 32.

Contrastive learning algorithm

- Following SimCLR [1], we experimented LARS with high learning rates from 1 to 4 and a large batch size of 512.
- We experimented LARS also with small learning rates of 0.0001, 0.001, 0.01 and 0.1.
- We experimented LARS also with scheduler of 10 epochs of LinearLR followed by CosineAnnealingLR scheduler.
- We experimented SGD with CosineAnnealingLR scheduler like reported in the git of [2]. We experimented it with batch sizes of 16, 32, 64, 128, 256 and 512.
- We experimented Adam with various batch sizes and learning rates.
- Based on these experiments we chose Adam batch size of 48.
  Note: in [1] they used a much larger batch size, which is the reason they had to use LARS. However, their dataset has 1000 classes which is probably the reason they needed a very large batch size.

Contrastive image augmentations

- We experimented with many augmentation methods and parameters.
- Unlike the CIFAR10 task, in this case relatively simple augmentations were sufficient.
- Unlike the CIFAR10 task, for MNIST it was sufficient to apply augmentations only for the contrastive learning, while there was no importance for applying augmentations in tasks 1.2.1 and 1.2.2.
- We choose the following augmentations:
  - Random Resized Crop with a scale range of $(0.6, 1)$
  - Randomly applied Gaussian blur with probability of 0.5 and kernel size 3.

Number of epochs

Task 1.2.1 - Reconstruct: 15
Task 1.2.1 - Classification: 20
Task 1.2: 10
Task 1.3 - Contrastive: 100
Task 1.3 - Classification: 20

CIFAR10

Since the architecture, learning algorithms, and reasoning are quite similar to MNIST, **we explain here only the differences.**

Encoder

Layer	Output Shape	Details
Input	(3, 32, 32)	RGB image
Conv2d	(256, 32, 32)	Kernel: 7×7, Padding: 3, Stride: 1
BatchNorm2d	(256, 32, 32)	
ReLU	(256, 32, 32)	
MaxPool2d	(256, 16, 16)	Kernel: 2×2, Stride: 2
Conv2d	(256, 16, 16)	Kernel: 7×7, Padding: 3, Stride: 1
BatchNorm2d	(256, 16, 16)	
ReLU	(256, 16, 16)	
MaxPool2d	(256, 8, 8)	Kernel: 2×2, Stride: 2
Conv2d	(2, 8, 8)	Kernel: 5×5, Padding: 2, Stride: 1
BatchNorm2d	(2, 8, 8)	
ReLU	(2, 8, 8)	
Flatten	(128,)	Converts $(2, 8, 8)$ to 128-D vector

Table 5: Encoder architecture for CIFAR10 tasks

- We used 3 layers according to your recommendations. We saw in the literature that more complicated architectures are sometimes used for CIFAR with contrastive learning, such as Resnet-18 and Resnet-50. However, we were able to surpass the required 60% accuracy with this simple model.
- In MNIST we reduced 28×28 image to 128 latent space by a single MaxPool2d and an additional linear layer. In CIFAR10 we reduced 32×32 image to 128 latent space by a two MaxPool2d and 2 output channels in the last convolution layer. One reason for these differences is that MNIST images are 28×28, so it is easier to reach 128 latent space with a linear layer added at the end, as it can easily reduce the latent space to any desired size.
- Another reason we used an additional linear layer only with MNIST is that an experiment with CIFAR10 showed that compressing the input image into a shape of (2, 8, 8) and then flattening it into a 128-dimensional latent space, without using a linear layer, gives better results. The reason for that it enabled adding convolution layers in the classifier, which helps to extract meaningful features for classification from the compressed image.
- Overall, this architecture achieved the best empirical performance on the classification part of task 1.2.1 and also showed strong results in the other tasks, outperforming many alternative architectures we experimented with.

Decoder

Layer	**Output Shape**	**Details**
Input	(2, 8, 8)	Latent vector reshaped
ConvTranspose2d	(256, 8, 8)	Kernel: 5×5, Padding: 2, Stride: 1
BatchNorm2d	(256, 8, 8)	
ReLU	(256, 8, 8)	
ConvTranspose2d	(256, 16, 16)	Kernel: 7×7, Padding: 3, Stride: 2, Output padding: 1
BatchNorm2d	(256, 16, 16)	
ReLU	(256, 16, 16)	
ConvTranspose2d	(256, 32, 32)	Kernel: 7×7, Padding: 3, Stride: 2, Output padding: 1
BatchNorm2d	(256, 32, 32)	
ReLU	(256, 32, 32)	
Conv2d	(3, 32, 32)	Kernel: 7×7, Padding: 3, Stride: 1
Tanh	(3, 32, 32)	Normalize output to $[-1, 1]$

Table 6: Decoder architecture used in the reconstruction section

- The decoder is the inverse of the encoder with a final convolution to smooth the reconstructed image, which also gave better results for the downstream task.
- A Tanh activation is used at the end to normalize the output into $[-1, 1]$. This is needed for computing reconstruction loss since we normalize the inputs.

Classifier

Layer	**Output Shape**	**Details**
Input	(2, 8, 8)	Input feature map
Conv2d	(256, 8, 8)	Kernel: 3×3, Padding: 1, Stride: 1
BatchNorm2d	(256, 8, 8)	
ReLU	(256, 8, 8)	
Dropout2d	(256, 8, 8)	Dropout ($p = 0.1$)
MaxPool2d	(256, 4, 4)	Pool size: 2×2
Conv2d	(512, 4, 4)	Kernel: 3×3, Padding: 1, Stride: 1
BatchNorm2d	(512, 4, 4)	
ReLU	(512, 4, 4)	
Dropout2d	(512, 4, 4)	Dropout ($p = 0.1$)
MaxPool2d	(512, 2, 2)	Pool size: 2×2
Flatten	(2048,)	Flatten feature maps
Linear	(512,)	Fully connected layer
BatchNorm1d	(512,)	
ReLU	(512,)	
Dropout	(512,)	Dropout ($p = 0.1$)
Linear	(256,)	Fully connected layer
BatchNorm1d	(256,)	
ReLU	(256,)	
Dropout	(256,)	Dropout ($p = 0.1$)
Linear	(128,)	Fully connected layer
BatchNorm1d	(128,)	
ReLU	(128,)	
Dropout	(128,)	Dropout ($p = 0.1$)
Linear	(10,)	Output logits for 10 classes

Table 7: Architecture of the classifier used across all CIFAR tasks

- The classifier includes 2 convolution layer, 3 fully connected layers and a linear layer.
- As noted in the encoder, adding the classifier convolution layers improved the results for the 1.2.1 task.
- the convolution layers increased the channels from 2 to 256 and then to 512, while reducing the imgage size from 8×8 to 4×4 and then to 2×4 by applying MaxPool2D.
- Other considerations are similar to those detailed in MNIST.

Contrastive loss function

Similarly to MNIST, we used the NT-Xnet loss. However, following experiments, we lowered the temperature to 0.15. A smaller temperature means the loss is more dominated by small distances, while widely separated representations do not impact much the loss. See discussion in blog [3]

Task 1.2.1 learning algorithm

Similarly to MNIST, we used Adam with a batch size of 64.

Task 1.2.2 learning algorithm

Similarly to MNIST, we used Adam, but increased the batch size from 32 to 64.

Contrastive learning algorithm

Similar to MNIST. we used Adam, but increased batch size from 48 to 128.

Augmentations

- Unlike MNIST, for CIFAR10, we used augmentation for training all 3 tasks and not only the contrastive task. The experiment showed that training with augmentation for tasks 1.2.1 and 1.2.2 was not significant for MNIST but significantly improved the results of CIFAR10. In fact, we were not able to surpass the required 60% accuracy in task 1.2.1 without it.
- For task 1.2.1, we experimented two different augmentation approaches. The first approach was using the augmented images just to enrich the training dataset. The second approach, inspired by diffusion models, was training the model to reconstruct the original image out of the augmented image. Since the second approach was more difficult to train, while the first approach surpassed the required 60% accuracy, we chose the simpler first approach.
- We used on-the-fly augmentation, meaning we generate new augmentations for each epoch rather than training with fixed augmentation. The reason is that this way we train on more diverse images as well as save memory.
- **After applying the augmentations, we normalized the images with the mean and the std of the CIFAR train dataset.**
- We did not augment the validation and the test datasets, but we normalized the validation and test images with the mean and the std of the CIFAR train dataset.
- We experimented with many different augmentation methods and parameters. The configurations that gave best results were different from MNIST contrastive learning.
- The best augmentation configurations also differ between CIFAR10 1.2.1 + 1.2.2 tasks and the CIFAR10 contrastive learning task.
- The choice of augmentation methods was inspired by the git of [2], but we modified it according to the experiment results to get better results with our architecture, which is weaker than the ResNet-18 used in [2].
- The selected augmentations for 1.2.1 and 1.2.2 tasks were
  - Random Resized Crop with a scale range of $(0.7, 1)$
  - Random Horizontal Flip with 0.5 probability
  - Randomly applied Color Jitter with probability 0.2, brightness 0.8, contrast 0.8, saturation 0.8 and hue 0.2.
- The selected augmentations for training the classifier in all 3 tasks were
  - Random Resized Crop with a scale range of $(0.7, 1)$
  - Random Horizontal Flip with 0.5 probability
- The selected augmentations for the contrastive learning task were
  - Random Resized Crop with a scale range of $(0.08, 1)$
  - Random Horizontal Flip with 0.5 probability
  - Randomly applied Color Jitter with probability 0.5, brightness 0.4, contrast 0.4, saturation 0.4 and hue 0.1.
  - Random Grayscale with probability 0.2
  - Randomly applied Gaussian Blur with probability 0.5 and kernel size 3

Number of epochs

Task 1.2.1 - Reconstruct: 20
Task 1.2.1 - Classification: 20
Task 1.2: 50
Task 1.3 - Contrastive: 200
Task 1.3 - Classification: 30

Overall:

- Training CIFAR10 required more epochs than training MNIST.
- Training the contrastive required **much** more epoches than other tasks.

Experiments methodology

- As detailed in the previous sections, we experimented with many
  - Network architectures
  - Optimizers
  - Learning rates
  - Batch sizes
  - Image augmentation methods
  - Contrastive learning temperatures
- The experiments were done based on a validation dataset (not the test dataset).
- To be able to run many experiments and compare many configurations, each experiment was done with 5-10 epochs.
- For tuning hyper-parameters of contrastive learning, experiments of 10 epochs were not sufficient . Thus, in these cases we first did 10 epochs experiments. Then selected the most promising configurations and compared those based on further experiments with 100 epochs.

2 Quantitative Results

Classification Accuracy

MNIST	Accuracy (%)		
Task	Train	Validation	Test
Self-Supervised Autoencoder (1.2.1)	99.34%	98.81%	98.47%
Classification-Guided Encoder (1.2.2)	99.17%	99.03%	99.14%
Structured Latent Space (1.2.3)	99.08%	98.90%	98.88%

CIFAR10	Accuracy (%)		
Task	Train	Validation	Test
Self-Supervised Autoencoder (1.2.1)	70.21%	63.15%	63.30%
Classification-Guided Encoder (1.2.2)	81.54%	81.44%	80.42%
Structured Latent Space (1.2.3)	72.38%	68.96%	69.64%

Reconstruction Error for Self-Supervised Autoencoders

	Mean Absolute Error		
Data Set	Train	Validation	Test
MNIST	0.0159	0.0162	0.0160
CIFAR10	0.296	0.297	0.295

Comparison and Analysis

For both MNIST and CIFAR10 datasets, the transfer learning with structured latent space encoders (1.2.3) gave better results compared to the transfer learning with self-supervised autoencoder (1.2.1). As expected, contrastive learning enhanced the separability of classes in the latent space, leading to better classification accuracy.

For both MNIST and CIFAR10 datasets, the classification-guided encoder (1.2.2) gave the best results. However, that doesn't mean that pretraining with self-supervised learning is not useful, since the comparison conditions were not the same. While in the classification-guided task (1.2.2) we trained the entire network, in the transfer learning tasks (1.2.1 and 1.2.3) the encoder weights were frozen, while only the decoder was trained on the classification task. We note that in exercise 3, training the entire pretrained network gave better results than training the decoder only (though that was with a different dataset, model, and task). We can also think of further transfer learning strategies, such as gradually unfreezing the encoder, but such research was not requested in this project. Anyway, the fact that the results of the transfer learning tasks (1.2.1 and 1.2.3) were not much below the classification-guided task (1.2.2), **despite the frozen encoder restriction**, shows that self-supervised learning is useful.

Overall, the results confirmed that pretraining self-supervised learning is useful for classification tasks and that doing it with contrastive learning is better than doing it with an image reconstruction autoencoder.

3 Qualitative Results

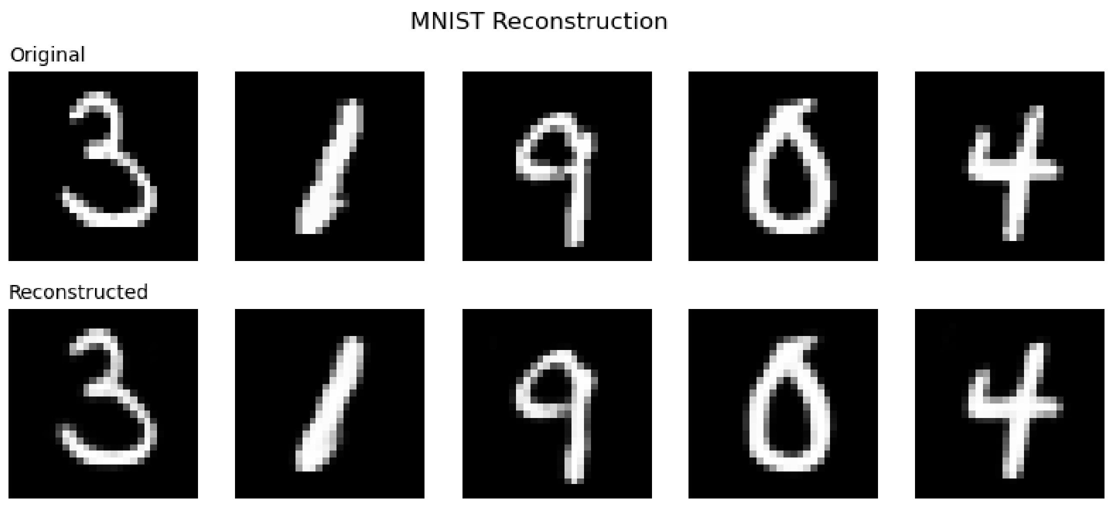


Figure 1: Reconstruction of 5 random images from the MNIST test dataset using the trained autoencoder. The top row shows the original images, while the bottom row shows their corresponding reconstructions.

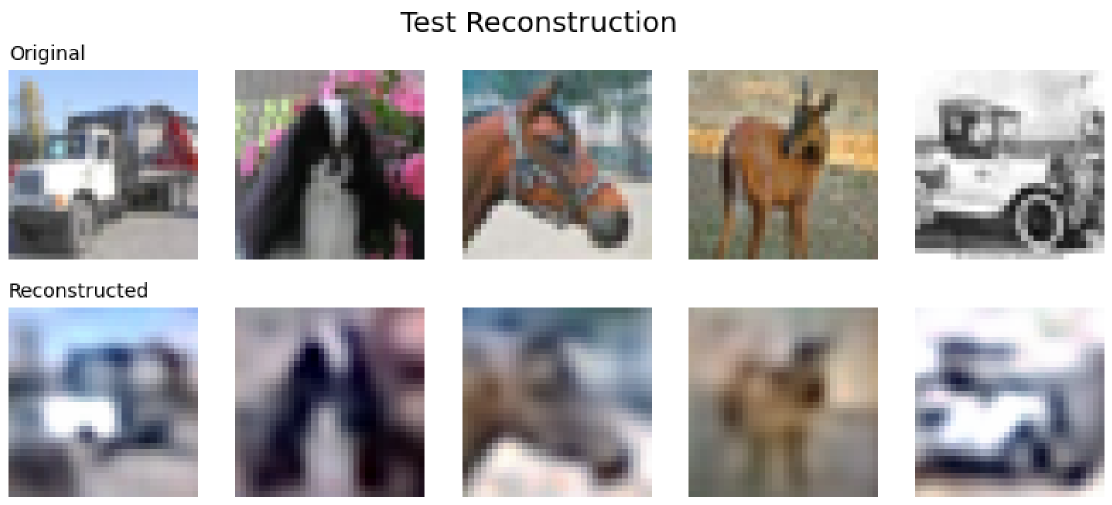


Figure 2: Reconstruction of 5 random images from the CIFAR10 test dataset using the trained autoencoder. The top row shows the original images, while the bottom row shows their corresponding reconstructions.

4 Linear Interpolation

Out of the 5 images above, we chose the image of 4 and the image of 5 because they are close on the t-SNE projections, and as we can see, the interpolation between them is smooth.

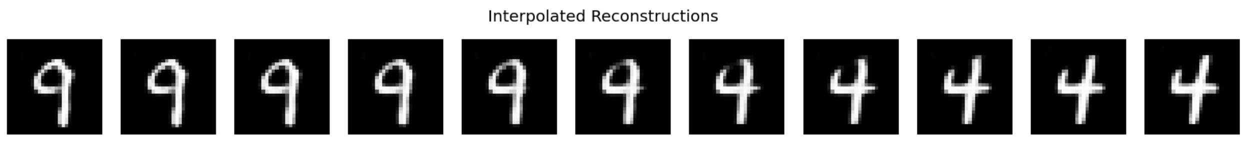


Figure 3: 10 steps linear interpolation between 4 and 9

Analysis

The received interpolations do seem like hand-written digits that could interpolate the two images in the image-domain. The transition between the digits 9 and 4 is smooth.

This is in fact above expectations. As discussed in the answer to question 10 of the dry part, unlike VAE, image reconstruction autoencoder doesn't guarantee smooth transitions and may result in images that are not similar to any digit. Nevertheless, in this example, we got good interpolation of the two digits in the image-domain with an image reconstruction autoencoder.

5 t-SNE Projections

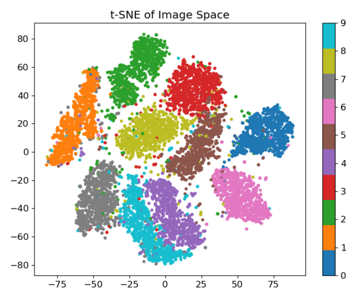
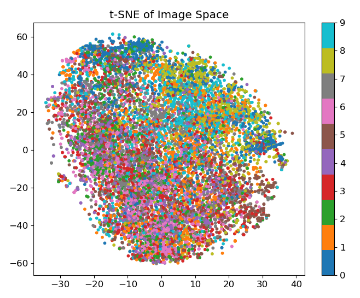
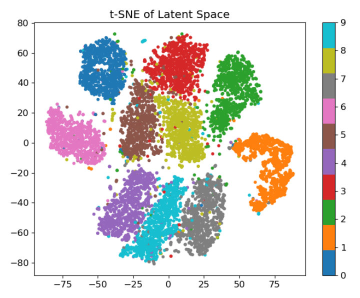
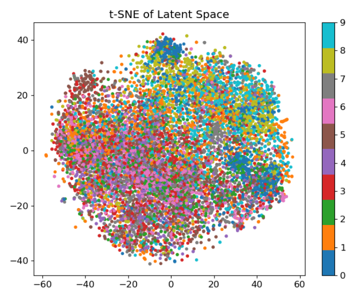
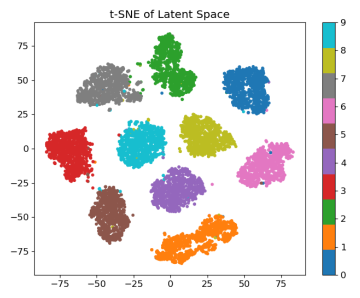
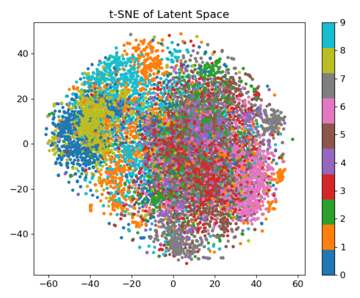
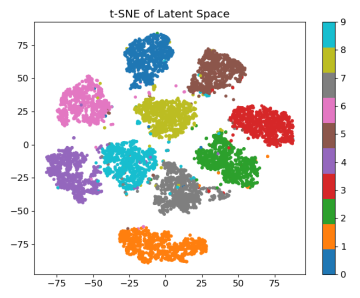
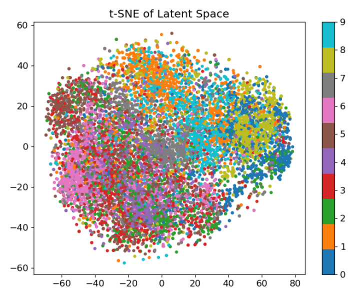
Task	MNIST	CIFAR
Image space		
1.2.1		
1.2.2		
1.2.3		

Table 8: t-SNE Projections of the test datasets from MNIST and CIFAR, including original image and latent space of the 3 tasks.

Analysis

When comparing the image spaces of MNIST and CIFAR10, we see that MNIST classes are already quite separated in image space, while CIFAR10 classes are very mixed. That explains why the CIFAR10 classification task is much more difficult and why its classification results were significantly lower.

The results of 1.2.1 seem quite similar to the image space in both datasets. That makes sense, since the autoencoder was trained to reconstruct the image without classes' separation intensive.

The results for 1.2.2 show much better separation between classes, for both datasets, compared to the image space and even slightly better separation than 1.2.3. That makes sense, as we trained a classification task, so as layers progress we expect the separation to improve. Yet, it is not obvious that the network was able to reach that separation after a few layers.

The results for 1.2.3 show much better separation between classes, for both datasets, compared to the image space. As expected, the contrastive learning helped shape the latent space to be more separable. That was done without using the class labels, which shows that contrastive learning is useful also for clustering and not only for classification.

References

[1] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[2] Weiran Huang, Mingyang Yi, and Xuyang Zhao. Towards the generalization of contrastive self-supervised learning, 2021.

[3] Lilian Weng. Contrastive representation learning. *lilianweng.github.io*, May 2021.