Part 1.4 Evaluation

Reasoning and quantitative results

MNIST – 1.2.1: Self-Supervised Autoencoding

Autoencoder Architecture

For the MNIST dataset, we chose a fully connected autoencoder because of the simplicity of the input data, which includes grayscale 28×28 images of handwritten digits.
The encoder consists of:

• A Flatten() layer to turn the 2D input image into a 784-dimensional vector (28x28)

• A fully connected layer projecting 784 → 256 with ReLU activation

• A final fully connected layer projecting 256 → 128 (the latent space)

The decoder mirrors this structure in reverse:

• A fully connected layer 128 → 256 with ReLU activation

• A final layer 256 → 784 with Sigmoid activation to constrain output between [0, 1]

• The result is reshaped back to (1, 28, 28)

We trained the autoencoder with L1 loss (MAE) in order to achieve sharper image reconstructions and better handle outliers.

We chose this architecture for several reasons.

First, the MNIST dataset is relatively simple, as it consists of low-resolution digits. This makes the use of complex convolutional architectures unnecessary.

Second, to minimize the risk of overfitting (especially in an unsupervised setting), we decided to use a shallower, fully connected model, since deeper or more expressive models would likely have overfit quickly.

Lastly, using a fully connected network offered greater interpretability, as it provided more control over the size and behavior of the latent space. Given the small image size and centered structure of MNIST digits, this design allowed us to observe how different features were represented in the latent space, without the added complexity of convolutional layers

Parameter Choices & Hyperparameters

Network Parameters:

• Latent Dimension: h = 128

As required by the assignment, we projected each image into a 128-dimensional latent space.

• Hidden Layer Size: 256

A simple intermediary size that balances representation capacity without excessive complexity.

• Depth: 2 layers per module

The encoder, decoder and classifier consist of two layers, resulting in a simple architecture with sufficient depth for MNIST without unnecessary complexity.

Hyperparameters:

• Learning Rate: 1e-3

Standard starting point for Adam optimizer; performed well without exploding or vanishing gradients.

• Optimizer: Adam

Adaptive and robust, especially effective for smaller datasets like MNIST.

• Loss Function: L1 Loss (MAE)

Chosen for its ability to produce less blurry reconstructions than MSE (L2).

• Batch Size: 128

Balanced stability and training speed, and fits comfortably in GPU memory.

• Epochs: 80

Based on early stopping (In our case, stopped after 18 epochs); empirically enough to converge without overfitting.

• Dropout: *Not used*

MNIST is simple enough that we did not need regularization in the encoder or decoder.

Quantitative Results

Training accuracy= 0.9892
Validation accuracy= 0.9723
Test accuracy= 0.9735
MAE loss= 0.0347

CIFAR-10 – 1.2.1: Self-Supervised Autoencoding

Autoencoder Architecture

For CIFAR-10, we designed a convolutional autoencoder to effectively capture spatial hierarchies in the 32×32 RGB images. CIFAR-10, unlike MNIST, consists of more complex, colored natural images, which makes convolutional layers more suitable for capturing spatial and hierarchical features.

Encoder

The encoder is made up of a series of three convolutional blocks that progressively downsample the input:

• Conv2d(3 → 64) with kernel size 4, stride 2, padding 1 → output size: 16×16

• Conv2d(64 → 128) → output size: 8×8

• Conv2d(128 → 256) → output size: 4×4

Each convolution is followed by ReLU activation.

Finally, the $256 \times 4 \times 4$ output is flattened and linearly projected to a latent space of size 128.

Decoder

The decoder mirrors the encoder using transposed convolutions to upsample:

• Linear layer maps latent vector 128 → 256×4×4

• ConvTranspose2d(256 → 128) → 8×8

• ConvTranspose2d(128 → 64) → 16×16

• ConvTranspose2d(64 → 3) → 32×32

The final layer uses Sigmoid activation to output normalized pixel values in the range [0, 1].

We trained the autoencoder using L1 loss (Mean Absolute Error) to encourage sharper reconstructions and better robustness to noise and outliers.

We chose architectural design for several reasons.
First, CIFAR-10 images are more complex compared with the images in MNIST, containing rich structure and color. This complexity makes convolutional layers essential for effectively learning hierarchical and spatial features.
Second, the use of strided convolutions and transposed convolutions provides a natural mechanism for downsampling and upsampling, which allows the model to compress and rebuild images without losing important details about their structure.
Lastly, the 128-dimensional latent space serves as a compact representation of the input, aligning with the assignment requirements and preserving meaningful image features.

Parameter Choices & Hyperparameters

Network Parameters

• Latent Dimension: h = 128

Per assignment requirement.

• Conv Layer Depths: 64 → 128 → 256

A common progression for image tasks, balancing depth and computational efficiency.

• Hidden Layer Sizes in Classifier:

256 → 128 → 10, with dropout for regularization.

Hyperparameters

• Learning Rate: 1e-3

A well-tested default for Adam, stable throughout training.

• Optimizer: Adam

Adaptive learning rate helped optimize both the AE and the classifier efficiently.

• Loss Function: L1 Loss (MAE)

Produced less blurry reconstructions than MSE and better visual quality.

• Batch Size: 128

Balanced performance and training time, especially on GPU.

• Epochs: 80

We used early stopping based on validation accuracy (stopped after 13 epoches). This was typically enough for convergence.

• Dropout: 0.5 in classifier only

Used to prevent overfitting on the more complex CIFAR-10 data during supervised classifier training.

Quantitative Results

Training accuracy= 0.5322
Validation accuracy= 0.5424
Test accuracy= 0.5534
MAE loss= 0.0424

MNIST – 1.2.2: Classification-Guided Encoding

Architecture Overview

In this section, we train the encoder **jointly with the classifier**, directly optimizing classification performance instead of reconstructing input images as in 1.2.1.

The encoder architecture remains the same:

• A Flatten() layer to convert the 28×28 input image into a 784-dimensional vector.

• A fully connected layer: 784 → 256, followed by ReLU.

• A final fully connected layer: 256 → 128, projecting into the latent space.

The classifier is a lightweight MLP:

• Fully connected layer: 128 → 64, followed by ReLU.

• Final layer: 64 → 10, outputting class logits for the 10 MNIST digits.

ReLU activation: We used ReLU activation in both the encoder and classifier to introduce non-linearity, allowing the model to learn more complex decision boundaries. ReLU is computationally efficient and reduces the risk of gradients becoming too small during training, making it especially effective for relatively shallow networks like ours on a simple dataset like MNIST.

We chose this architecture for several reasons, which include:

1. Task Alignment: The encoder is trained specifically for classification, allowing it to specialize in class-discriminative features.
2. Efficiency: By removing the decoder, the training focuses only on minimizing classification error, which accelerates convergence.
3. Supervised Latent Learning: The latent space is now shaped by label supervision, encouraging separability of digit classes in the learned embeddings.

Training Strategy

We trained the encoder and classifier jointly, using:

• Loss Function: Cross-entropy (standard for classification tasks).

• Optimizer: Adam, with a learning rate of 1e-3.

• Batch Size: 128

• Epochs: Max 80, with early stopping after 3 epochs without validation improvement(stopped after 10 epochs).

• Evaluation: Saved and evaluated the model using the best validation accuracy during training.

Quantitative Results

Training accuracy= 0.979
Validation accuracy= 0.9779
Test accuracy= 0.9937
MAE loss= 0.0190

CIFAR-10 – 1.2.2: Classification-Guided Encoding

Architecture Overview

Encoder

The encoder is a convolutional neural network designed to reduce spatial dimensions progressively while increasing the number of feature channels:

• **Conv2d(3 → 64), kernel_size=4, stride=2, padding=1**

Reduces spatial size from 32×32 to 16×16. The kernel size of 4 with stride 2 is chosen to aggressively downsample while still capturing local structure. Padding of 1 preserves the center alignment.

• **Conv2d(64 → 128), kernel_size=4, stride=2, padding=1**

Further reduces the spatial resolution from 16×16 to 8×8. ReLU activation is used to introduce non-linearity.

• **Conv2d(128 → 256), kernel_size=4, stride=2, padding=1**

Final downsampling to 4×4 spatial resolution. This results in a compact 256-channel feature map with spatial dimensions 4×4.

• **Flatten()**

Converts the 256×4×4 tensor into a 4096-dimensional vector.

• **Linear(4096 → 128)**

Projects the flattened vector into a 128-dimensional latent space.

Classifier

The classifier processes the latent vector using:

• **Linear(128 → 256)** with ReLU and Dropout(0.5)

Dropout is used to regularize and prevent overfitting.

• **Linear(256 → 128)** with ReLU and Dropout(0.5)

Another layer to refine the representation with added regularization.

• **Linear(128 → 10)**

Outputs the class logits.

Design Rationale

• **Kernel Size = 4**: Slightly larger than the standard 3×3 kernel. This allows the model to capture more spatial context at each step, which is beneficial during early downsampling.

• **Stride = 2**: Reduces spatial resolution by half at each layer. This eliminates the need for pooling layers and enables the network to reduce feature map size efficiently.

• **Padding = 1**: Maintains alignment and prevents excessive shrinking of the feature maps, allowing for consistent control over spatial dimensions.

• **Dropout in Classifier**: CIFAR-10 is more complex and has higher overfitting risk compared to MNIST. Dropout is added to mitigate this.

In this section we designed a convolutional encoder and classifier to perform classification-guided representation learning. The encoder consists of three convolutional layers with increasing channel depth (64 → 128 → 256), each using a kernel size of 4, stride of 2, and padding of 1. This architecture progressively reduces the input resolution from 32×32 to 4×4, extracting hierarchical features while preserving spatial structure. After flattening the final feature map (256×4×4 = 4096), we project it to a latent space of dimension 128 using a fully connected layer. This compact latent vector is then passed to the classifier, a three-layer MLP designed to promote separable representations. It uses two hidden layers (256 and 128 units), each followed by ReLU activations and 50% dropout to reduce overfitting, before mapping to the 10 CIFAR-10 classes. This design encourages the encoder to produce latent features that are both compact and highly discriminative, directly guided by the classification loss.

Training Strategy

• **Joint Training**: Encoder and classifier are optimized together to minimize cross-entropy loss.

• **Loss Function**: CrossEntropyLoss is used to guide classification.

• **Optimizer**: Adam optimizer with a learning rate of 1e-3, suitable for stable convergence on moderate-scale datasets.

• **Batch Size**: 128, which offers a balance between training stability and speed.

• **Early Stopping**: Validation accuracy is monitored; training halts if there is no improvement for 3 consecutive epochs. (stopped after 11 epochs)

Quantitative Results

Training accuracy= 0.6692
Validation accuracy= 0.6622
Test accuracy= 0.8227
MAE loss= 0.5279

Section 1.2.3 — Structured Latent Spaces (Contrastive Learning)

In this section, our goal was to learn a meaningful latent space without using labels during the encoder training phase. To do this, we used a **contrastive learning approach** inspired by SimCLR, where the model is encouraged to bring augmented views of the same image closer together in the latent space while pushing apart views of different images. After this self-supervised stage, we freeze the encoder and train a classifier using labeled data.

CIFAR-10

Encoder Design

For CIFAR-10, we chose to base our encoder on **ResNet18**, which is a well-known convolutional architecture that performs strongly on natural image datasets. CIFAR-10 images are relatively small (32×32), so we slightly adapted the original ResNet18:

• We replaced the first convolution with a smaller kernel (kernel_size=3, stride=1, padding=1) to better preserve spatial resolution in early layers.

• We removed the max-pooling layer (nn.Identity()) to retain more fine-grained features at the beginning of the network.

The final layers of ResNet (used for classification) were removed, and instead, we used a **projection head**:

• Flatten → Linear(512 → 512) → ReLU → Linear(512 → 128)

This projection head maps the final feature vector into a lower-dimensional latent space (128 dimensions), which is used for contrastive loss. The reason we use a projection head (instead of training directly on the features) is based on SimCLR findings: learning benefits from having a separate space for contrastive learning and classification, which helps the encoder learn more transferable representations.

Augmentations

We used the same set of augmentations proposed in SimCLR for natural images:

• RandomResizedCrop, RandomHorizontalFlip, ColorJitter, RandomGrayscale, and ToTensor.

These augmentations are critical. Without labels, the only "signal" for similarity is that two augmented views came from the same image. The model learns to ignore irrelevant variations like position, lighting, or color while preserving the content (e.g., a dog is still a dog after augmentation). This is how it builds a semantic understanding of the images.

Contrastive Loss

We used the **NT-Xent loss** (Normalized Temperature-scaled Cross Entropy Loss), which measures how similar positive pairs (two views of the same image) are relative to all other (negative) pairs in the batch. The temperature parameter (set to 0.5) controls how sharply the model focuses on the most similar/dissimilar samples.

Classifier

Once the encoder was trained, we froze its weights and trained a classifier on top. For consistency, we reused the same classifier as in section 1.2.2:

• Linear(128 → 256) → ReLU → Dropout → Linear(256 → 128) → ReLU → Dropout → Linear(128 → 10)

This architecture allows enough capacity to map the structured latent space to the label space while helping reduce overfitting using dropout.

Quantitative Results

Training accuracy= 0.7824
Validation accuracy= 0.7454
Test accuracy= 0.7499

Motivation Summary

CIFAR-10 contains diverse, complex visual features. Using a deep ResNet encoder enables the model to extract high-level representations. The contrastive approach, aided by strong augmentations, forces the encoder to learn these representations without labels. Then, using a fixed encoder and training only the classifier allows us to evaluate how well the latent space captures class-relevant information.

MNIST

Encoder Design

For MNIST, we took a simpler approach. Since MNIST images are 28×28 grayscale digits with low intra-class variation, we used a small MLP-based encoder:

• Flatten → Linear(784 → 256) → ReLU → Linear(256 → 128)

This architecture is computationally efficient and sufficient for the dataset. Our goal wasn't to over-engineer the model but to test whether contrastive learning could work even in such a low-complexity setting.

Augmentations

We designed the augmentations to match MNIST's characteristics:

• RandomResizedCrop, RandomRotation, RandomAffine, RandomGrayscale, and ToTensor.

These transformations slightly alter the digit's orientation and appearance without changing its identity, allowing the model to learn that a "3" is still a "3" even when rotated or stretched.

Contrastive Training

Similarly to CIFAR-10, we trained the encoder using NT-Xent loss. This helped the model cluster different augmentations of the same digit close together in the latent space.

Classifier

The classifier used here is the same one from section 1.2.1:

• Linear(128 → 64) → ReLU → Linear(64 → 10)

Since MNIST is a simpler database, we kept the classifier small and simple.

Quantitative Results

Training accuracy= 0.9708
Validation accuracy= 0.9658
Test accuracy= 0.9684

Motivation Summary

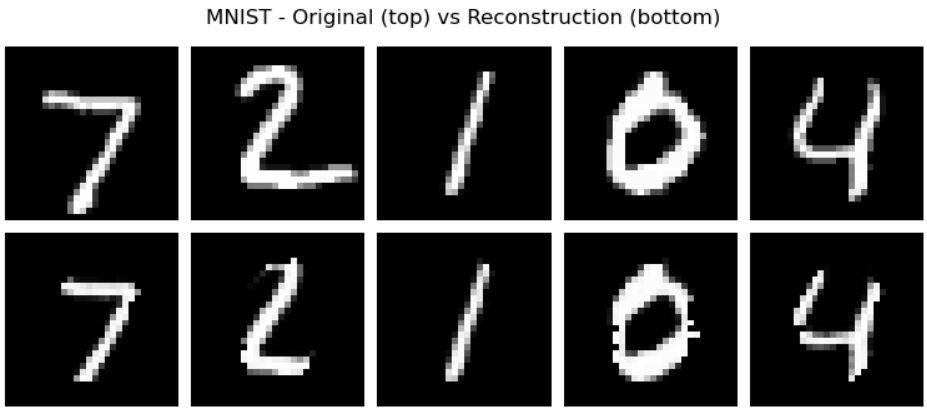
Although MNIST is a simpler dataset, it was important to test contrastive learning in this setting. Our hypothesis was that even a small encoder, when trained with contrastive loss and augmentations, could produce a useful latent space. The strong results support this: contrastive learning works even when the data isn't complex, as long as augmentations provide meaningful variation.

(cifar10 stopped after 27 epochs in 1.2.3 , mnist after 14 epochs)

Qualitative Evaluation (MNIST, Task 1.2.1):

The results demonstrate that our model successfully captures the essential structure of the digits. While the reconstructions are slightly smoothed—a common characteristic of autoencoders trained with L1 loss—the digit identities are preserved accurately.
This indicates that the encoder was able to extract meaningful latent representations, and the decoder learned to reconstruct coherent digit shapes from them.

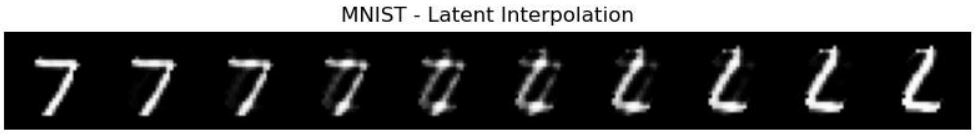
Given the simplicity of the MNIST dataset, our choice of a fully connected architecture with a 128-dimensional latent space proved sufficient for capturing the key visual patterns in the data without overfitting. The reconstructions confirm that the model did not merely memorize the training set but learned a generalizable encoding-decoding strategy. These qualitative results complement the quantitative evaluation and reinforce the effectiveness of our self-supervised autoencoding approach on MNIST.



Qualitative Results – CIFAR-10:

As shown in the figure, the reconstructions successfully preserve the high-level structure and overall color distributions of the original images. For instance, the silhouettes of boats, animals, and background colors remain recognizable. However, the outputs are significantly blurrier compared to MNIST reconstructions and lack fine-grained textures and details. This is expected due to the increased visual complexity of CIFAR-10 compared to MNIST, and the fact that the model was trained with an L1 reconstruction loss only, without perceptual or adversarial components. Despite this, the model demonstrates that it learns to encode and decode meaningful representations, capturing coarse object semantics. These results confirm that even a basic self-supervised setup can extract useful visual patterns from complex image domains like CIFAR-10.

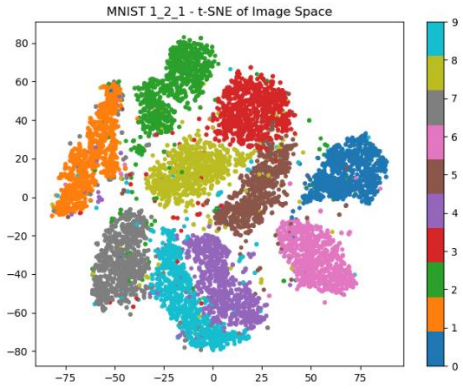
Linear interpolation mnist

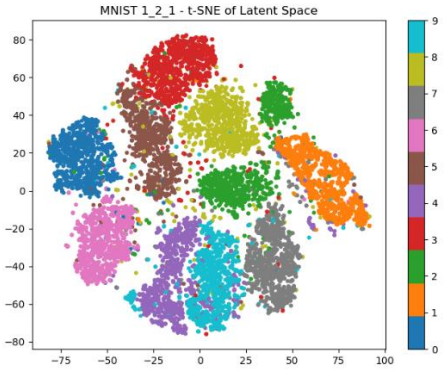


We selected a pair of digits from the MNIST test set (7 and 2), and performed a 10-step linear interpolation between their latent representations as encoded by our self-supervised encoder. The intermediate vectors were then decoded back into image space using the trained decoder. As seen in the figure, the transition is smooth and continuous, demonstrating that the latent space is not only compact but also semantically meaningful. The interpolated images resemble realistic handwritten digits, even though they do not correspond to real examples in the dataset. This suggests that the autoencoder has learned a structured representation of digit identity, where different digits lie on a continuum in the latent space. Such behavior is characteristic of well-trained generative models and highlights the effectiveness of self-supervised training in capturing data manifolds.

Comparison of t-SNE Projections for MNIST

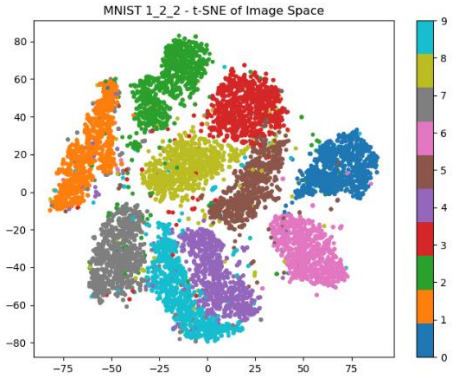
Section 1.2.1 — Self-Supervised Learning (Autoencoder)
• **Image Space**: Natural class structure appears, but with some mixing — expected since this is raw pixel space.
• **Latent Space**: Clusters begin to emerge, but there's noticeable overlap between digits. The encoder captures useful features, but since it's optimized for reconstruction and not discrimination, it doesn't strongly separate digit classes.
(especially between visually similar digits like 3 and 5 or 9 and 4).

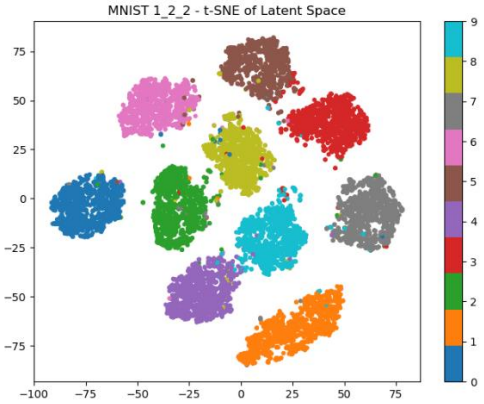




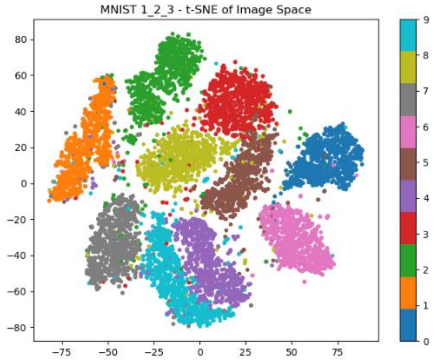
Section 1.2.2 — Classification-Guided Training
• **Image Space**: Identical to 1.2.1, since this is based on raw image data.
• **Latent Space**: Clusters are tightly packed and highly separable, with minimal overlap. Each digit class forms a distinct region, clearly benefiting from the supervised classification signal.

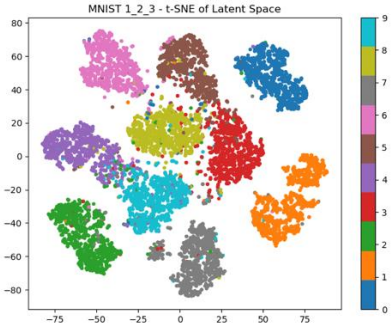
• **Conclusion**: This is the most separable latent space among the three — thanks to the classification loss directly encouraging class-level separation.





Section 1.2.3 — Contrastive Learning (SimCLR-style)
• **Image Space**: Again, unchanged from previous methods.
• **Latent Space**: Clusters are well-formed and clearly separated — better than in 1.2.1, but **not as compact or distinct as in 1.2.2**. Some digits, especially those with similar shapes (like 4 and 9), exhibit more overlap compared to the classifier-guided model.





Did section 1.2.3 help shape the latent space to be more separable?

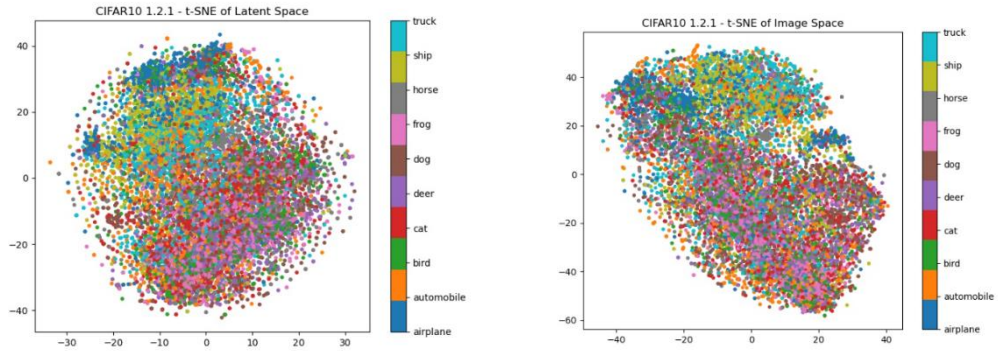
Yes, but not as effectively as classification-guided training (1.2.2).

The latent space in 1.2.3 is more organized than in 1.2.1, thanks to contrastive learning encouraging representations of different digits to spread apart. However, it doesn't reach the same level of class compactness and clarity as 1.2.2, where the encoder is directly trained to distinguish digit classes. This makes sense — contrastive learning uses instance-level supervision rather than explicit class labels, which is powerful, but slightly less effective for this simple dataset.

CIFAR-10 – t-SNE Analysis

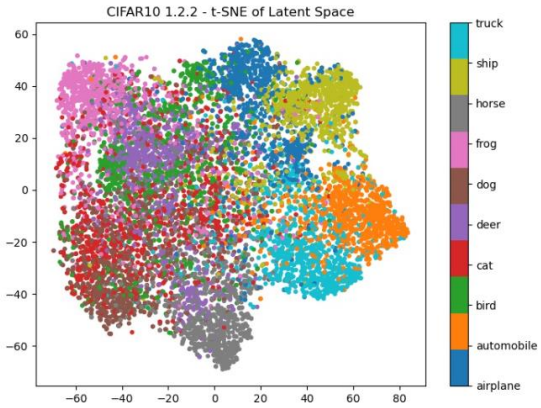
CIFAR-10 Class Labels:
0 - airplane
1 - automobile
2 - bird
3 - cat
4 - deer
5 - dog
6 - frog
7 - horse
8 - ship
9 - truck

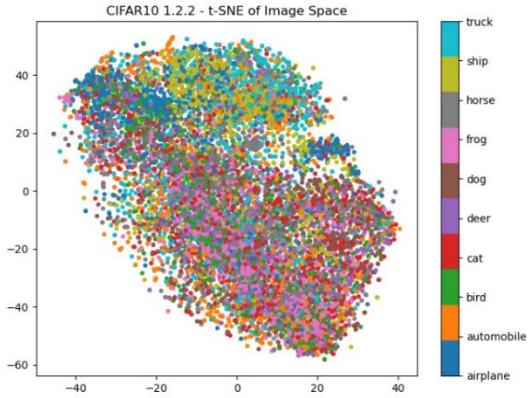
Self-Supervised (1.2.1)



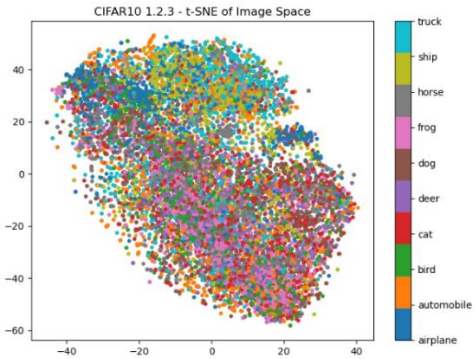
• **Image Space:** Entangled class representations — hard to distinguish even in raw image space.
• **Latent Space:** Shows no real separation — different classes are mixed together.
Unsurprising, as no labels are used to guide representation learning.

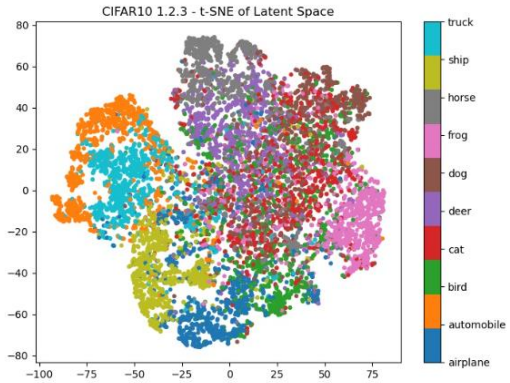
Classification-Guided (1.2.2)
• **Image Space:** Same as 1.2.1 — messy and overlapping.
• **Latent Space: Shows improved class separation.**
Adding classification loss helps form more distinct, class-aware embeddings. For example, the model does a pretty good job telling apart automobiles and trucks, but it seems to have a hard time figuring out the difference between cats and dogs.





Contrastive Learning (1.2.3)





• **Image Space:** No change from 1.2.1/1.2.2 — still messy.
• **Latent Space: Visibly improved** from 1.2.1. Not perfect, but **clearer clusters emerge**, especially for some classes.
Comparable to classification-guided training, though perhaps slightly noisier.
Similarly to part 1.2.3, the model here also does a pretty good job telling apart automobiles and trucks, but it seems to have a hard time figuring out the difference between cats and dogs.

Conclusion (CIFAR-10):
Contrastive learning (1.2.3) proves more effective on CIFAR-10 than on MNIST — likely due to its ability to learn invariances from complex and diverse data. It outperforms self-supervised training (1.2.1), and in some clusters, approaches or even matches the separation quality achieved by classification-guided training (1.2.2).
Notably, it also achieved higher test accuracy, supporting the visual evidence of better learned representations.