Exploring Denoising Autoencoder Architectures for Self-Supervised Learning.

Gerald White

Advisor: Yali Amit

Approved _____

Date _____

February 16th, 2023

Abstract

Modern deep learning networks rely on massive amounts of labeled data for training, which can be prohibitively costly and time consuming to generate. The field of self-supervised learning aims to circumvent this issue by using unsupervised pretraining to generate lower-dimensional data representations. Cutting-edge methods utilize two "views" or transformations of each data point to create an embedding space that maintains the underlying structure of the data distribution. Out of these methods, "contrastive" methods– which create positive pairs and negative pairs of each image in a batch– reign supreme despite their dependence on hyper-parameters like batch size. In this work, we propose a denoising autoencoder-based training architecture as an alternative to contrastive methods. The novelty of this method comes from utilizing a denoising autoencoder in combination with various types of embedding losses to generate representations. We perform sensitivity analysis to obtain the optimal weighting of the embedding loss with respect to the denoising reconstruction loss and then compare the resulting models to contrastive methods. We evaluate the learned representations via a standard linear evaluation protocol. In linear evaluation, our method achieves 67.3% classification accuracy, compared with 69.7% for the contrastive SimCLR despite our method not requiring the use of negative pairs. The code repository used for all experiments can be found at `https://github.com/artenyx/SSLProject`.

Contents

1 Introduction

1.1 Self-Supervised Learning

In the field of Deep Learning, labeled data is both of paramount importance and of high cost to generate. Modern state-of-the-art models rely on millions if not billions of labeled or annotated images for training [13] [17] [5]. This short supply and high demand creates a premium on labeled data for use in deep network training.

In computer vision, image-class data sets like ImageNet are the product of monumental crowd-sourcing operations that are not always feasible for smaller projects or niche data sets. On the other hand, images without labels are extremely plentiful across the internet. This discrepancy motivates the field of Self-Supervised Learning (SSL), which aims to use unlabeled data to train a representation space and reduce the downstream dependence on labeled data.

Self-supervised learning thus has two stages: an unsupervised representation learning through a pretext task, and a downstream supervised fine-tuning task. The goal of the representation learning stage is to form a representation space that captures the general structure of the data. For example, in a quality embeddings space we would expect the distance between the embeddings of a beagle and a golden retriever to be smaller than that of a beagle and a motorcycle.

The general framework of the representation learning process is that two "views" of the batch images are created by a predefined random augmentation pipeline and then compared in some way to train an embedding space. This embeddings space is defined by an encoder and a projector network, and when training is complete the projector is removed and the encoder generates representations. These representations can then be used for the downstream task.

During representation learning, we assume that we do not have access to large sets of labeled data and thus the computational resources required for training the representation space are not of paramount concern. For this reason, networks are often trained for upwards of 500 epochs. If this training is performed with a quality objective, the network should be able to efficiently encode any observation from the input distribution.

1.2 The Collapsing Problem and Contrastive Learning

One of the central problems in Self-Supervised Learning is the collapsing problem. [10] Simply put, complete collapse occurs when the network objective only incentivizes similarity between the two image views, and the network quickly learns to map all input to the same embedding vector. This mapping maximizes the similarity-based objective function while producing an uninformed network which has "tricked" our objective. Partial collapse, when some dimensions of the embeddings space are not utilized, can also occur because of a weak training objective.

State-of-the-art methods utilize contrastive learning to create representations. [13] [10] [5] Contrastive learning uses intra-batch relationships to train the network and avoid collapse. These schemes compare the embeddings of "positives", image views which stem from the same original image, and "negatives", image views which stem from different images. [4] The positives and negatives are treated differently in the loss function with the ultimate goal of embedding positives near and negatives far in the embedding space.

It follows that as the batch size increases, the number of negative pair relationships increases quadratically. For this reason contrastive frameworks run optimally at very high batch sizes, which can be unfavorable in some settings. [4] [6]

An illustrative example of this phenomenon is SimCLR, which uses its novel Information Noise Contrastive Estimation (InfoNCE) loss to push negative pairs far and positive pairs close in the embedding space. [4] [1] This framework was groundbreaking and laid the foundation for state-of-the-art methods like OpenAI's CLIP and Google's CoCa, which currently dominates the ImageNet benchmark. [13] [17] SimCLR uses encoder and projector networks to produce representations of the two views of an image. These representations are then fed to the InfoNCE loss. Because this loss function relies on the negative pairs in its loss function, it operates best at high batch sizes (4096 in the original paper). [4]

This coupling of performance to batch size in contrastive methods has generated interest in alternatives to contrastive methods that do not directly rely on the negative pairs. SimSiam for example, utilizes two separate branches with a projection layer only on one branch, followed by a similarity loss between the projected view and the non-projected view. [6] Boostrap your own latent or BYOL uses an online and a less frequently updated target network to bootstrap latent code which is used in downstream tasks. [9] [7]

In this work, we present another approach to self-supervised learning which avoids collapse without relying on the comparison of negative pairs. This framework combines a "positive-only" embedding loss with a decoder and a denoising reconstruction loss to disincentivize the network from producing a trivial embedding space.

1.3 Autoencoders

Autoencoders are comprised of an encoder network which generally maps the input to a lower dimensional latent code, followed by a decoder which attempts to reconstruct the original image given the output of the encoder. Because the size of the latent code is so small, the autoencoder must compress the input in a way that naturally can be used as a data representation.

The denoising autoencoder is structurally similar to its non-denoising counterpart, though it is tasked with "unlearning" an augmentation that was applied to its input. In other words, if a random augmentation pipeline is applied to an image, a perfect denoising autoencoder will return the original image before it was augmented.

The framework presented in this paper can be thought of as a denoising autoencoder with an added embedding loss to promote the creation of a quality representation space. By forcing the network to unlearn the augmentation pipeline, it must also learn the underlying structure of the data distribution. We show that this method performs better than its non-denoising counterpart, and then compare it to the SimCLR framework performed with the same encoder and projector networks. Notably, the proposed method does not use negative pairs, intense covariance computations as in Barlow Twins, or any clustering or bootstrapping in order to generate representations. [18]][9]

2 Methods

2.1 Embedding Training and Denoising Autoencoders

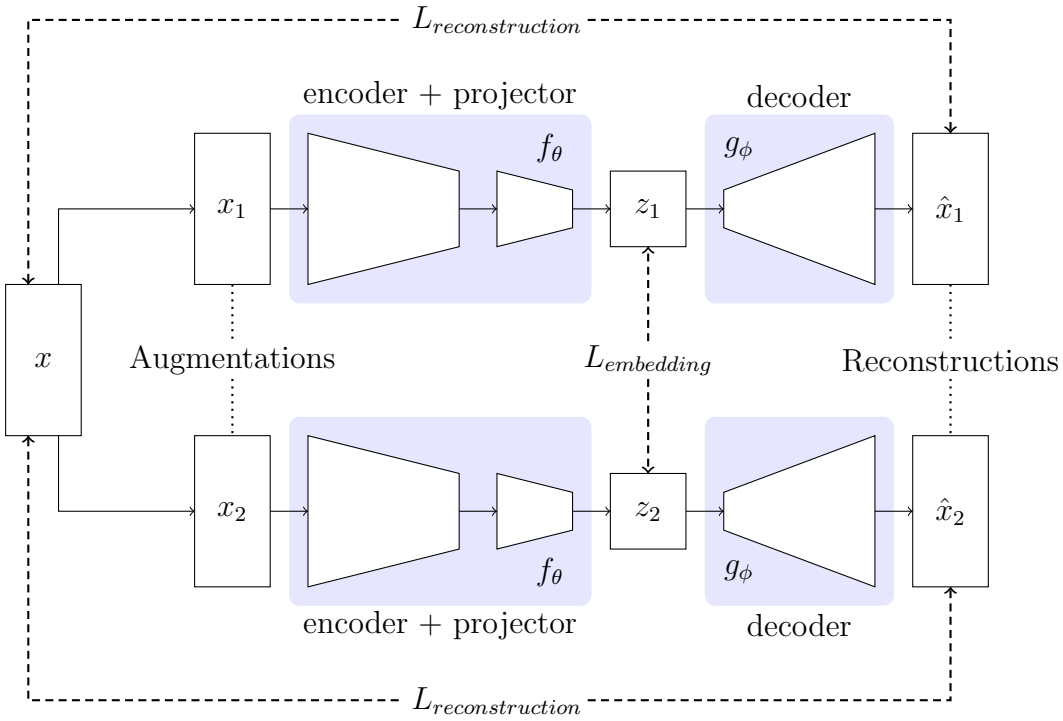


Figure 1: Denoising autoencoder self-supervised learning architecture. Two views are generated from an image and then embedded and reconstructed via the network. The distance between the two embeddings forms the embedding loss, and the distance between each reconstruction and the original unaugmented image forms the reconstruction loss.

The autoencoder is an intuitive choice for representation learning by the nature of its architecture. By forcing the network to reconstruct an image after it is compressed into a low dimensional space, the embedding must capture the salient features in order to produce any type of reconstruction of the data, whether denoised or otherwise. We should then be left with a quality representation space to be used in downstream tasks.

In practice this is not the case. With no constraints or penalties on the latent code, the autoencoder does not inherently organize the embedding space efficiently. This leads to weak downstream representations and poor performance.

One way to mitigate this problem is to take a classical self-supervised learning approach. To do this, we generate views of the input image and pass them through the autoencoder, while retaining the embeddings and reconstructions in order to back-propagate our loss signal. By incorporating a reconstruction loss, we avoid the collapsing problem while also including an

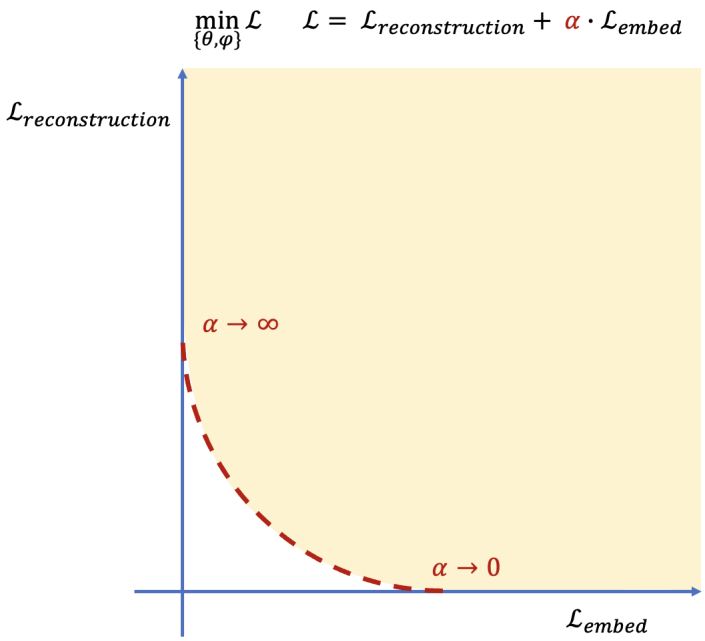


Figure 2: Pareto front diagram illustrating optimization priority in relation to hyper-parameter α.

embedding loss to ensure that we obtain quality downstream representations. This idea was previously explored with a non-denoising autoencoder and an L2 embedding loss. [3]

In this setting, the original image x is passed into some augmentation distribution to obtain two views $x_1, x_2 \sim D(x)$. These views are passed through the encoder and projector to generate embeddings $z_1 = f_\theta(x_1), z_2 = f_\theta(x_2)$, then through the decoder to obtain reconstructions $\hat{x}_1 = g_\phi(z_1), \hat{x}_2 = g_\phi(z_2)$. Here, the size of the latent code is represented by $|z| = d$. We can then parameterize the loss function as follows:

$$\mathcal{L} = \mathcal{L}_{reconstruction}(x_1, x_2, \hat{x}_1, \hat{x}_2) + \frac{\alpha}{d} \cdot \mathcal{L}_{embed}(z_1, z_2) \tag{1}$$

$$\mathcal{L} = ||x_1 - \hat{x}_1||_2 + ||x_2 - \hat{x}_2||_2 + \frac{\alpha}{d} \cdot ||z_1 - z_2||_2 \tag{2}$$

Since our element-wise L2 embedding loss only requires comparison between views which stem from the same image, we call this a "positive-only" loss function. While this loss function does not use negative pair computations, we generalize the formulation to allow the testing of various types of embedding losses in combination with a reconstruction loss.

$$\mathcal{L} = ||x_1 - \hat{x}_1||_2 + ||x_2 - \hat{x}_2||_2 + \frac{\alpha}{d} \cdot \mathcal{L}_{embed}(z_1, z_2) \tag{3}$$

The hyper-parameter α controls the balance between prioritization of the two portions of the loss function during training. Thus, at very high α values, we expect lousy reconstructions and potential collapse of the embedding space for the previously explained reasons. For very

low α values, we expect strong reconstructions yet a weak learned representation space. If the given embedding and reconstruction loss-type combinations are compatible, we would expect some value of α where there is higher downstream performance compared to when α approaches 0 and ∞.

An issue with this framework arises when a comparison is made between the effects of the two portions of the loss function on the encoder. As with other SSL methods, the embedding loss in Equation 3 incentivizes the encoder to embed the two views similarly or identically, but the reconstruction loss incentivizes the encoder to embed the two views differently, so that they can be reconstructed as their separate original views. Thus, the two portions of the loss function are not aligned and are competing during optimization.

This disconnect between the functions of the embedding and reconstruction losses inspire the framework presented in this paper. Rather than tasking the network with reconstructing the image views, we employ a denoising reconstruction loss which tasks the network with reconstructing the original unaugmented image. Under this formulation, the embedding loss and reconstruction loss are more aligned, as the reconstruction loss is also incentivizing the encoder to embed the views similarly, as both views will be reconstructed as the original unaugmented image. Thus, this framework prevents collapse by the inclusion of a decoder while aligning its reconstruction loss to the objective of representation learning. An overview of our method can be found in Figure 1.

While the typical autoencoder reconstruction loss is an element-wise L2 loss, this framework does not yet specify the type of embedding loss to use in combination with the reconstruction loss. By leaving this choice open, we can experiment with various positive-only embedding losses and test the performance of networks trained with these losses. Using a similar parameterization as above, we have the following loss function for this new architecture:

$$\mathcal{L} = ||x - \hat{x}_1||_2 + ||x - \hat{x}_2||_2 + \frac{\alpha}{d} \cdot \mathcal{L}_{embed}(z_1, z_2) \tag{4}$$

Because all terms of the loss function are positive-only, this method is theoretically independent of batch size and maintain a simple loss calculation. While an element-wise loss may be simplest, we will also experiment with using similarity losses which compare the entire embedding vector of the positive pair.

Thus, as our network learns to reconstruct the un-augmented image and embed the views close together, it is learning a strong representations space which can be used for downstream tasks. As expected, this framework performs definitively better than its non-denoising counterpart in a linear evaluation classification task. Our method also achieves comparable results to SimCLR run on the same encoding networks while relying on simple losses and not requiring the inclusion of negative pairs.

3 Experiments

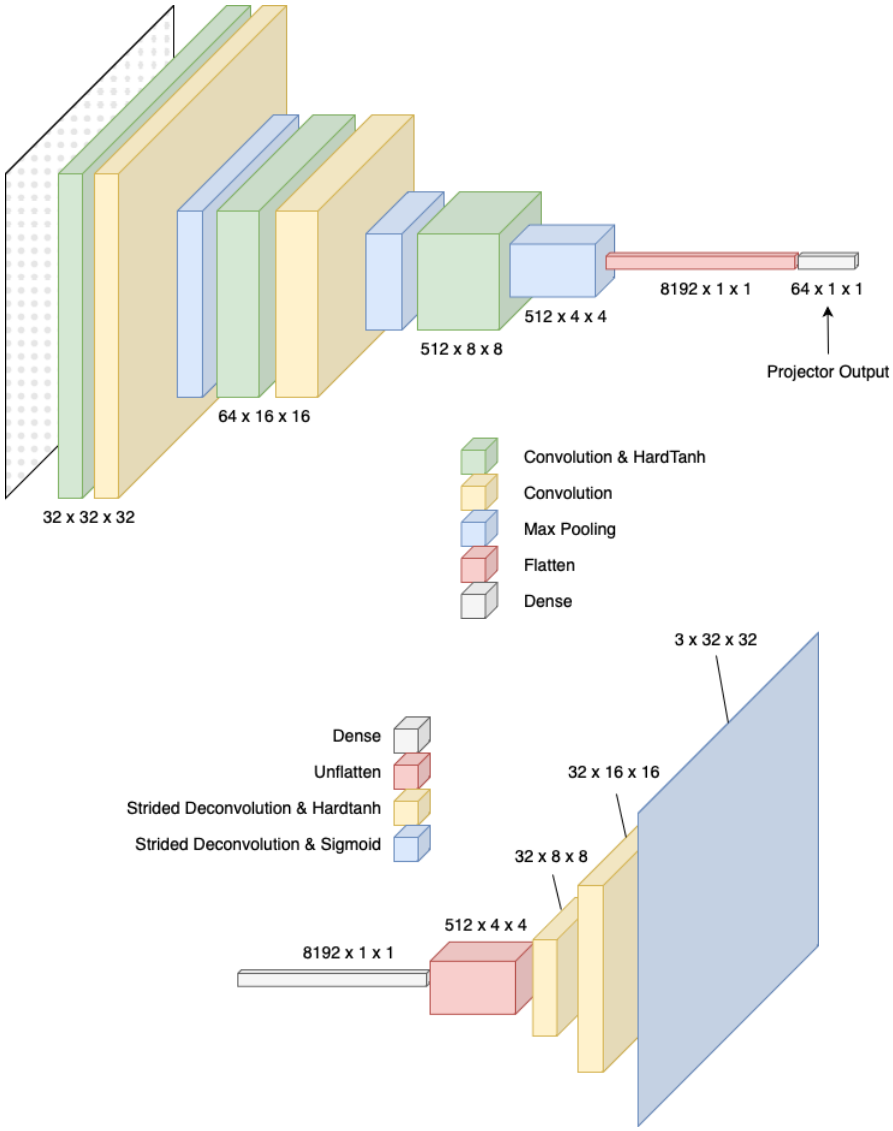


Figure 3: Overview of encoder (top) and decoder (bottom) networks as well a sizes of image tensors throughout layers (for a single image). All convolutions and pooling layers have 2 dimensional kernels.

As the goal of our experiments is to explore the viability of this new method by comparing performance to SOA, simple neural networks are used for the encoder, projector, and decoder.

The encoder network is comprised of 5 2D convolutional layers, each followed by either a hard hyperbolic tangent (Hardtanh) activation or a 2D max pooling as presented in [12]. The output of the encoder is called the representation and it is then flattened and input to the projector, which is a simple dense layer connecting the 8192-dimensional encoder output to the latent code size of 64. The output of the projector layer is called the embedding and it is used in the embedding loss calculation.

The decoder is much simpler than the encoder as it consists of a dense layer, which expands the dimension of the embedding (64) back to the size of the representation (8192), followed by an unflatten operation and three strided deconvolution layers with either Hardtanh or Sigmoid activation functions. There are about 2 million total trainable parameters in this network. Note that the decoder is only used when the network being trained is an autoencoder.

A summary of the layer-wise outputs of the network can be found in Figure 3 and full network details can be found in Table 3.

3.1 Training Routine

3.1.1 Augmentation Pipeline

	Transform Type	**Parameter Type**	**Value**
Default Pipeline	Random Crop	Window Size	28 x 28
	Resize	Window Size	32 x 32
	Random Horizontal Flip	Probability	0.8
	Random Color Jitter	Brightness	0.2
		Contrast	0.2
		Saturation	0.2
		Hue	0.05
Other Transforms	Gaussian Blur	Kernel	3
	Random Solarize	Threshold	0.75
		Probability	0.2

Table 1: Details of augmentation pipeline. Default pipeline is used for all experiments unless otherwise specified. Other transforms are used in some experiments at the given parameters.

Before any experiments were run, an augmentation pipeline was applied to each of the images in the representation learning data set (CIFAR-100). [11] During every epoch of

unsupervised training, two views of the image were drawn from the augmentation pipeline to be used for either autoencoder or SimCLR training.

Following the procedure laid out by [4], the PIL images were first converted to [0,1] tensors before they were passed to augmentations. [15] Then, a random 28-pixel square crop was taken of the image and the remaining image was resized back to a 32-pixel square. Then, the images were subject to a potential random horizontal flip and random color jitter which altered the brightness, contrast, hue and saturation of each image.

In some experiments, Gaussian Blur and Random Solarize were also included in the augmentation pipeline, though not by default. The full details of the pipelines can be seen in Table 1.

3.1.2 Representation Learning

Algorithm 1 PyTorch-style pseudo-code for unsupervised representation training of a siamese autoencoder network with an L2 embedding loss.

model = nn.Sequential(∗[encoder, projector, decoder])
optimizer = torch.optim.Adam(model.parameters(), lr = 0.0001)
crit_rec = nn.MSELoss() ▷ Set reconstruction criterion.
crit_emb = nn.MSELoss() ▷ Set embedding criterion.
for i in range(epochs_usl) **do**
  for x in usl_loader **do** ▷ Draw CIFAR100 images.
    $x_1, x_2 = D(x)$ ▷ Draw views from augmentation distribution
    $e_1, e_2 = \text{encoder}(x_1), \text{encoder}(x_1)$
    $z_1, z_2 = \text{projector}(e_1), \text{projector}(e_1)$
    $r_1, r_2 = \text{decoder}(z_1), \text{decoder}(z_1)$
    optimizer.zero_grad()
    $\text{loss} = \text{crit\_rec}(r_1, x) + \text{crit\_rec}(r_2, x) + \alpha \cdot \text{crit\_emb}(z_1, z_2)/d$
    loss.backwards()
    optimizer.step()
  end for
end for

To test this novel framework, all experiments begin with unsupervised pretraining of the given network. In this portion of the experiment, two views of the image-batch were extracted from the random augmentation pipeline shown in Table 1 and then fed through the network. The 256-dimensional view embeddings were retained for use in the loss function and passed through the decoder network to be reconstructed. The embeddings and reconstructions were then utilized to calculate a loss value which was used to update the network.

Unless otherwise noted, the default setting was to conduct representation training for 300 epochs on CIFAR-100. The input were of size $[b \times 3 \times 32 \times 32]$, where b is the batch size that

is being used. Algorithm 1 demonstrates the process for running representation learning with the denoising autoencoder configuration.

During this portion, the Adam optimizer was used with a cosine learning rate decay schedule and an initial learning rate of 0.0001. [14] The default batch size was 512 and the size of representations and embeddings were 8192 and 64 dimensions, accordingly.

3.1.3 Linear Evaluation

Algorithm 2 PyTorch-style pseudo-code for linear evaluation of learned representations.

```
model = nn.Sequential(*[encoder, nn.Linear(representation_dim, n_classes])
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
crit_le = nn.CrossEntropyLoss()                    ▷ Set linear evaluation criterion.
for i in range(epochs_le) do
    for x, y in le_loader do                       ▷ Draw CIFAR10 images and class labels.
        p = model(x)
        optimizer.zero_grad()
        loss = crit_le(p, y)
        loss.backwards()
        optimizer.step()
    end for
end for
```

For this portion of the Self-Supervised learning experiments, the projector and decoder networks were removed and the weights of the encoder network was frozen. Then, a new data loader was created using the linear evaluation dataset, which was CIFAR-10 in this case. This loader was created by passing the CIFAR-10 images through the encoder to generate representations and then pairing them to their respective labels.

Once the loaders were created, a one dimensional dense layer which took inputs of 8192 and generated outputs of 10 dimensions was randomly initialized. This miniature network was then trained on the representations to obtain the linear evaluation classification rate. Algorithm 2 provides an overview of this portion of training.

During hyper-parameter tuning (α and learning rates), a CIFAR-10 validation set was used to test performance of the networks, while during assessment time, a held-out test set was used. Linear evaluation was run by default for 200 epochs and the batch size was kept at 512. The Adam optimizer was used for this portion with a learning rate of 0.001 and no learning rate scheduler.

3.1.4 K-Means Clustering

Another newer way of evaluating the quality of the embedding space is by conducting some sort of clustering on the learned representations and measuring the distance between the points and their respective centroids.

This process was completed with the k-Means clustering package of Sci-Kit Learn with an L2 metric and 10 cluster centers for the 10 respective classes in the data set. The evaluation criterion used was the intertia of the clustering, or the average L2 distance between the representations and their cluster centers.

All experiments were conducted in PyTorch on a Linux Ubuntu machine and accelerated using 4 NVIDIA RTX A6000 GPUs. The complete code repository for the experiment can be found at `https://github.com/artenyx/SSLProject`.

4 Experiments

4.1 Denoising vs. Conventional Autoencoder

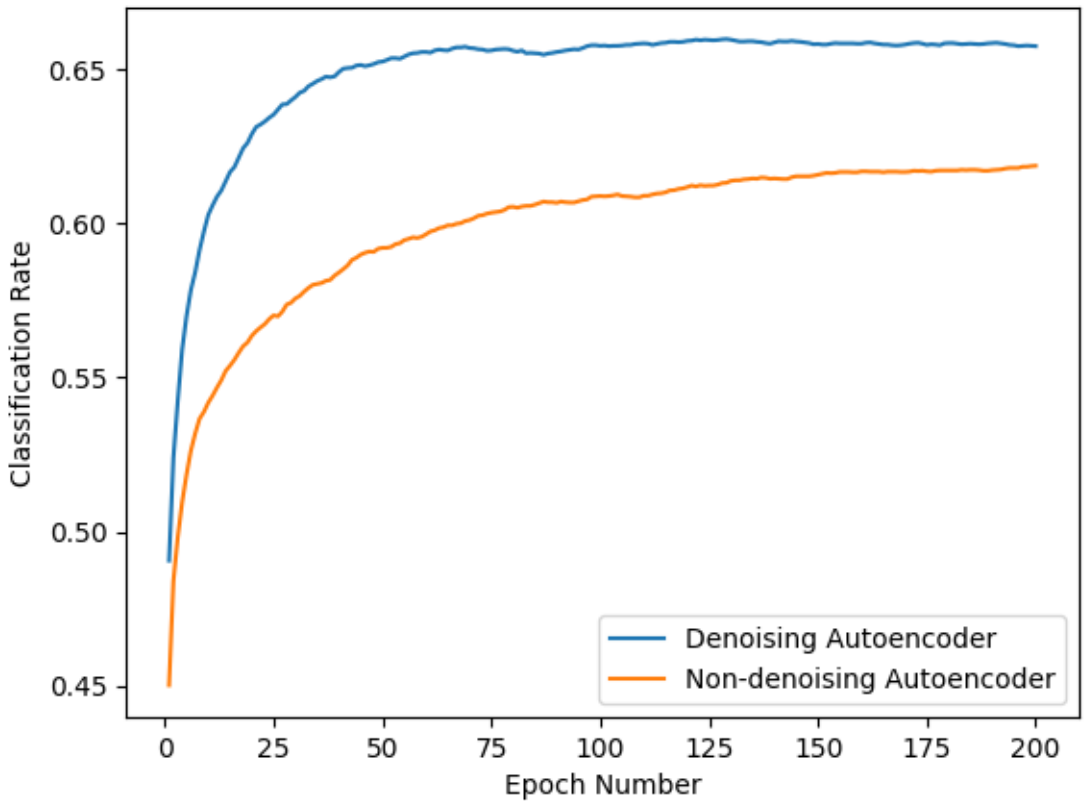


Figure 4: Linear evaluation trajectories for representations trained with a denoising vs a non-denoising autoencoder with no embedding losses. The autoencoder representation achieves a 61.2% classification rate while the denoising autoencoder representation achieves a 65.9% classification rate on CIFAR-10.

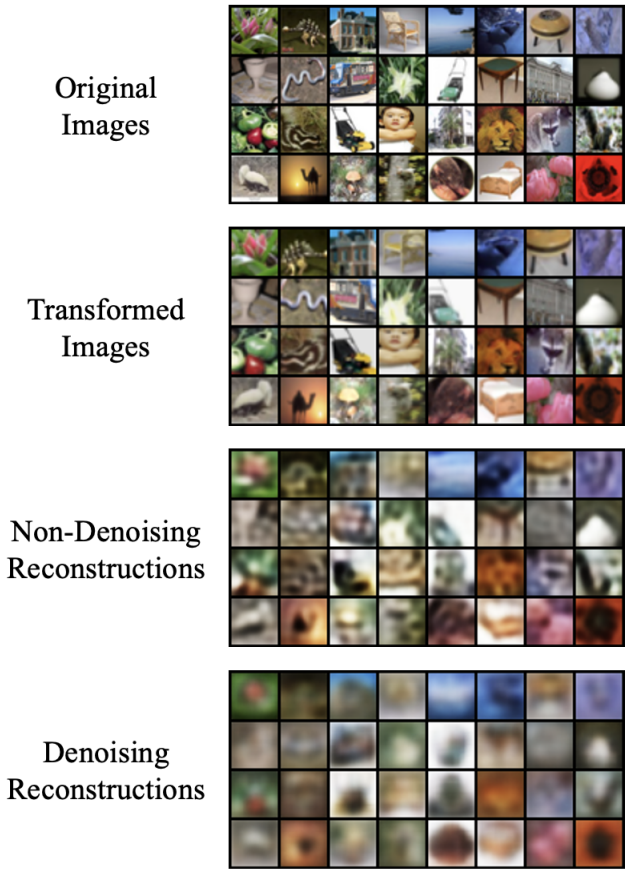


Figure 5: CIFAR-100 test-set images with reconstructions from denoising and non-denoising autoencoders. Networks trained for 300 epochs on CIFAR-100.

Some previous work has been done to study the effect of using a conventional autoencoder with an embedding loss and comparing against state of the art methods. [3] This work compared the performance of a non-denoising autoencoder base with a positive-only L2 embedding loss to SimCLR and did not achieve competitive performance to SimCLR. The main novelty of this method lies in utilizing a denoising reconstruction loss to increase network performance.

One way to interpret the denoising objective is that the network is incentivized to abstract the salient features of the non-augmented data distribution in order to reverse the random augmentation applied to the image. The model then learns to reconstruct the original unagumented image from the embeddings of the two augmented views. As previously discussed, this aligns the two portions of the loss function and thus allows the learning of a higher quality representation space.

Figure 4 shows the linear evaluation trajectory for representations trained by a traditional autoencoder compared with a denoising autoencoder. Not only does the denoising representation create faster convergence in linear evaluation, it also achieves about a 5% higher

classification rate.

As the reconstruction loss is the only aspect of the training routine that is varied in this experiment, it is clear that the denoising task produced stronger representations than its non-denoising counterpart. This then confirms the idea that a denoising reconstruction loss is more aligned with the SSL objective.

One interesting artifact of using decoders in the context of SSL, is that we can observe and draw insights from the reconstructions during training. Figure 5 shows one example of a set of unaugmented input images and a corresponding set of views, non-denoising reconstructions, and denoising reconstructions. As expected, the non-denoising autoencoder produces reconstructions that are much more faithful to the input views while the denoising reconstructions are more hazy and difficult to discern.

As shown in Figure 4, however, we obtain higher quality representations from the non-denoising framework. It follows that faithful reconstructions do not directly create stronger representations and thus utilizing a reconstruction loss like L2 may be overly restrictive for this setting. However, we leave this work to be explored in another setting.

The denoising reconstructions seem to show a hazy "average" of the augmentation distribution. Most likely as a product of the random cropping augmentation, most of the reconstructions are much more centered than their corresponding views, illustrating the network's unlearning of this transformation. For some images, like the lawnmower towards the bottom left of the image, the denoising autoencoder predicts somewhat of a superposition of the image if it were horizontally flipped, which indicated the network's modeling of the horizontal flip augmentation of the pipeline. We might then expect that the denoising autoencoder would be more transformation invariant than other methods, though we explore this question in further experiments.

4.2 Embedding Loss

As previously mentioned, this framework does not inherently define an embedding loss type to be used during training. Thus, three choices were experimented with in this work: element-wise L1 and L2 losses, and a negative cosine similarity (NCS) loss between positive pairs. While the element-wise losses may be unnecessarily strict in incentivizing the creation of identical embeddings for the two views, the NCS loss allows for more flexible embeddings as they do not have to be identical to maximize similarity. These three choices result in the following three equations for the embedding portion of the loss function:

$$\mathcal{L}_{embed}(z_1, z_2) = ||z_1 - z_2||_1 \qquad \text{L1 Loss} \tag{5}$$

$$\mathcal{L}_{embed}(z_1, z_2) = ||z_1 - z_2||_2 \qquad \text{L2 Loss} \tag{6}$$

$$\mathcal{L}_{embed}(z_1, z_2) = \frac{z_1 \cdot z_2}{||z_1||_2 \cdot ||z_2||_2} \qquad \text{NCS Loss} \tag{7}$$

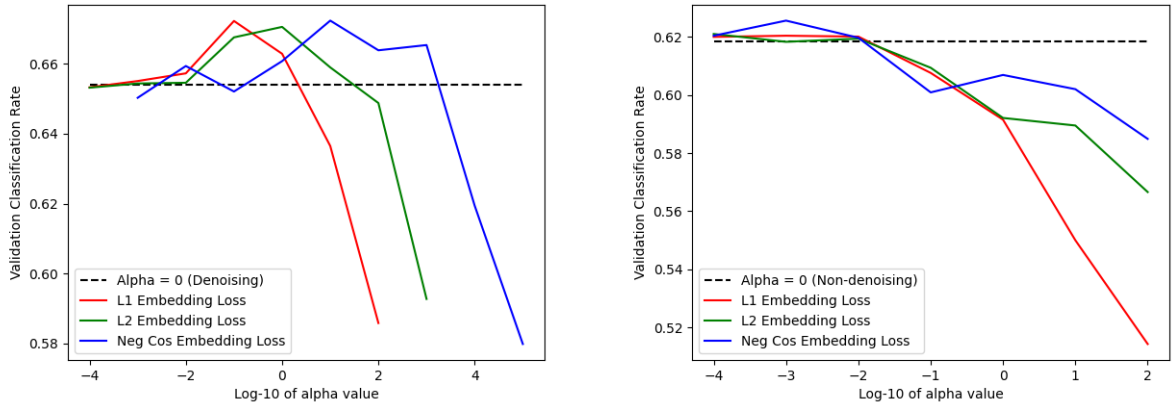


Figure 6: A comparison of classification rates on a validation set using a denoising autoencoder architecture (left) and a non-denoising autoencoder architecture (right) with various embedding losses.

For each type of embedding loss, a CIFAR-10 validation set was used to perform a sensitivity analysis and identify the optimal weighted combination of the reconstruction and embedding losses to be used in training. Values of α between 10^{-4} and 100 (unless larger values were necessary) were tested separately for each loss, as the loss types generally vary by scale on orders of magnitude. The linear evaluation classification rates on this validation set are reported and used to determine which α settings were optimal.

By our parameterization, high α values correspond to high weight placed on the embedding loss in comparison to the reconstruction loss, and vice versa. It follows that when $\alpha = 0$, the architecture is simply a denoising autoencoder scheme and the representations and reconstructions should have no effect on each other.

In Figure 6, we can see the performance of the networks under various loss types / α-values for a denoising scheme (left) and a non-denoising scheme (right). Here the dotted lines represent baseline performance when $\alpha = 0$.

The first takeaway from Figure 6 is how the non-denoising autoencoder does not seem to benefit much from the inclusion of an embedding loss across the tested types. This is further evidence for the idea that the non-denoising reconstruction loss is not aligned with the SSL objective of embedding the two views near each other in the embedding space.

In the denoising case, there is a more clear optimal balance between the embedding and reconstruction losses, indicating more clear alignment between the loss portions at different weightings. At low α values, we see that the performance is similar to when $\alpha = 0$ and as α grows, we see first an increase and then a sharp decline in performance. This decline can be explained with the pareto diagram included in Figure 2; as α becomes too large, the embedding space comes closer to a collapsing regime.

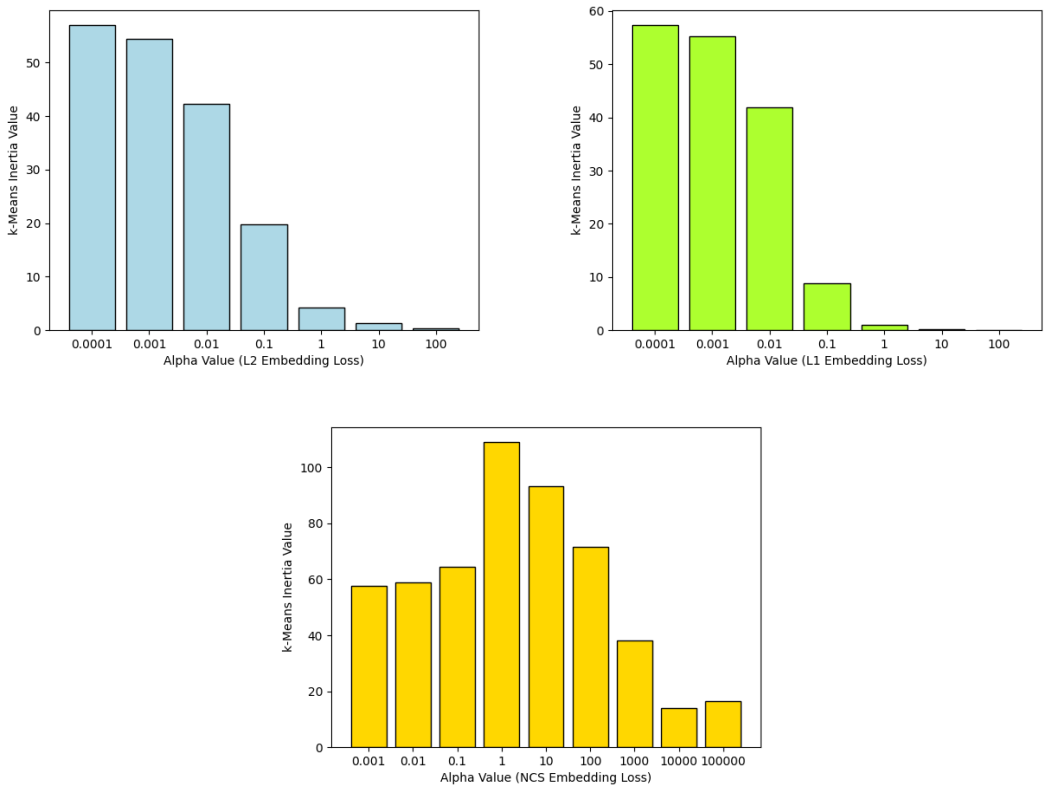


Figure 7: Inertia values (average L2-distance between point and cluster-center) for representations of CIFAR-10 images obtained by training siamese denoising autoencoders at various embedding-loss strengths.

Thus, the performance curves on the left of Figure 6 show the embedding and reconstruction losses performing similar functions and achieving optimal performance for different α values for each embedding loss type. By a small margin, the NCS embedding loss performs the best in the denoising experiments.

To better understand the function of increasing α, k-means clustering with 10 centers was conducted on the CIFAR-10 test set representations. These results are shown in Figure 7. As expected, the L1 and L2 plots show an inverse relationship between $\log_{10}(\alpha)$ and the inertia. This makes sense since as these element-wise penalties increase, the embeddings are forced to take on smaller values and the representations are more likely to be closer to their cluster centers.

Interestingly, the NCS loss, while having the highest performance, does not behave in the same way. There is not a rapid shrinking of inertia values as the value of α increases but rather an initial increase in inertia before an eventual decline. This most likely is due to the fact that NCS is not an element-wise loss, and thus does not constrain the magnitude of the elements of the embedding vectors. Further work could be done to explore why the NCS loss embeddings behave differently and what its relationship is to performance.

Method	Embedding Loss	Alpha Value	Top-1 Accuracy
SimCLR	None	–	**69.7**
Non-denoising AE	None	0	61.2
	L1	0.01	62.0
	L2	0.0001	62.1
	Negative Cosine Similarity	0.001	**62.6**
Denoising AE	None	0	65.9
	L1	0.1	67.1
	L2	1	67.1
	Negative Cosine Similarity	10	**67.3**

Table 2: Linear Evaluation Classification performance on CIFAR-10 for various embedding losses and SOA SimCLR.

4.3 Comparison with Contrastive Methods

4.3.1 Linear Evaluation Performance

As mentioned previously, a linear evaluation procedure was used to determine the strength of the learned representation space. After the linear evaluation network was trained for 200 epochs, the CIFAR-10 test set was used to assess the final performance of the different methods.

The out of the denoising autoencoder methods, the best performing networks (along with their $\alpha = 0$ counterparts) were chosen to be measured against SimCLR to assess performance. The results of these experiments are summarized in Table 2.

As shown in Table 2, in final classification rates the denoising architecture still handily outperforms the non-denoising one, with the NCS embedding loss achieving the highest performance among both schemes. While this denoising scheme does not ultimately outperform SimCLR (67.3% top-1 accuracy compared to 69.7%), it is competitive with this contrastive framework despite the lack of negative pair computations. This demonstrates that our framework can be feasible utilized for SSL.

While the denoising autoencoder method does not outperform SimCLR on under the base conditions, there are some settings where our method performs better than SimCLR. The experiments which illustrate two of these settings are included below.

4.3.2 Effects of Augmentation Strength

The augmentation pipeline is critical to the performance of SSL. Various works have shown the importance of the augmentation pipeline in learning strong representations from self-

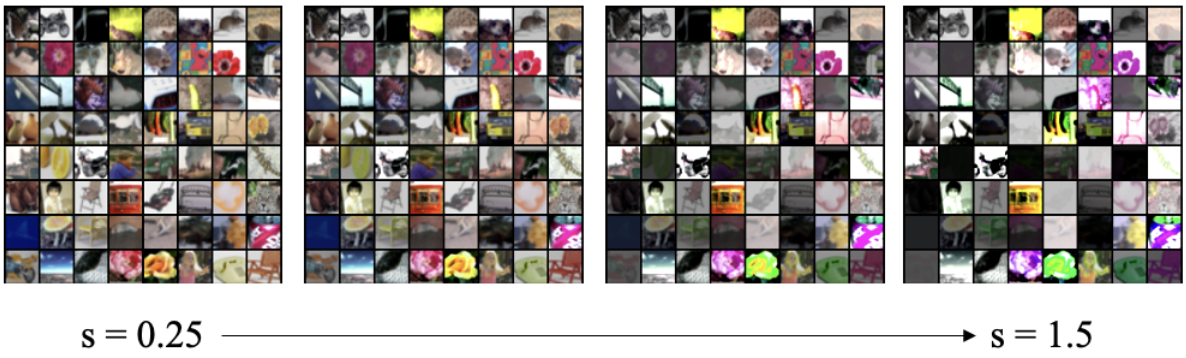


Figure 8: 8 x 8 CIFAR-100 images with varying levels of color jitter strength.

supervised learning. [4] [18] [2] [16] [15] The random color jitter is among the most common augmentations used in these experiments, and thus it was chosen for its own set of experiments to dig deeper into the relationship between augmentation strength and the quality of the representation space learned through these frameworks.

In this experiment, the augmentation strength or s is representative of the max value that each color property of the image can be jittered as shown below:

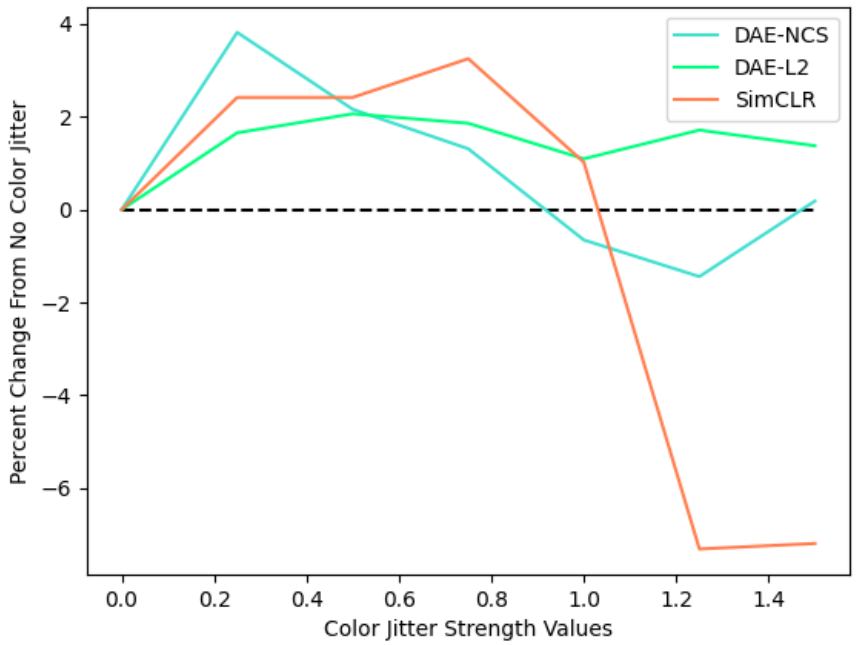
Brightness:	$0.8 \cdot s$
Contrast:	$0.8 \cdot s$
Saturation:	$0.8 \cdot s$
Hue:	$0.2 \cdot s$

Thus, if we have a higher augmentation strength, the image properties will be more likely to be intensely modified by the augmentation pipeline. Figure 8 shows output images of the augmentation pipeline with various values of s.

Augmentation strength values to test were chosen uniformly between 0 and 1.5, with 7 total strength settings tested. For each strength value, an entire SSL experiment was run and the CIFAR-10 classification rate was produced. The results of this experiment can be seen in Figure 9, which measures the change in classification rate from the baseline (no color jitter) to the strength that was tested in that experiment.

As we can see, each method benefits from some level of color jitter, though at high strength levels the methods diverge. The SimCLR method seems to break down at high levels of augmentation strength, losing around 7% performance from the baseline, and over 10% compared to peak performance. The autoencoders on the other hand do not collapse nearly as badly, and actually seem to retain their strong performance at high augmentation strengths. In fact, the L2 embedding denoising autoencoder barely suffers any performance compared to its peak, and still performs better at than baseline at high augmentation strength levels.

As shown at the bottom of 9, both denoising autoencoder methods outperform SimCLR at the highest strength level of $s = 1.5$. This indicates that the representations obtained



	DAE-L2	DAE-NCS	SimCLR
Top-1 acc. @ s = 1.5	66.8%	64.8%	63.2%

Figure 9: Percentage changes of classification rates when augmentation pipeline's color jitter strength is increased. Below the plot, top-1 accuracies in linear evaluation are shown for the three methods at the highest strength value. Data shown for NCS Denoising Autoencoder (AE-NCS), L2 Denoising Autoencoder (AE-L2), and SimCLR.

from the autoencoder methods may be more invariant to augmentations used in the augmentation pipeline, which can be a desirable quality for many settings including when the augmentation pipeline is not as well studied as this one. This invariance inherently makes sense, as during training the denoising autoencoder is learning an expected inverse of the augmentation pipeline.

4.3.3 Effects of Length of Representation Learning

Another way that we evaluate the strength of the method is to test how quickly the representation space is learned during training. To do this, we checkpoint the model during various points in pretraining and test the representation space via the linear evaluation protocol. If we see high top-1 accuracy early in the representation space, we might say that the method quickly learns a representation space.

In Figure 10, we see the results of this experiment for the denoising autoencoder with an L2 embedding loss and the contrastive SimCLR. While we already know that SimCLR outperforms the denoising autoencoders at the previously tested representation learning

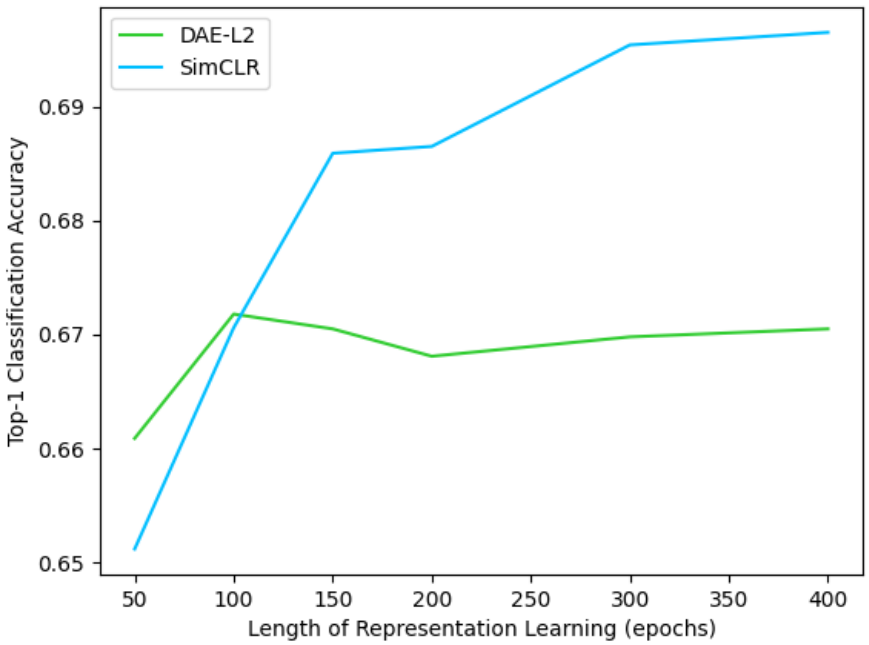


Figure 10: Top-1 accuracy after 200 epochs of linear evaluation plotted against varying lengths of representation training for a denoising autoencoder with L2 loss as well as SimCLR.

length (300 epochs), it seems that the denoising autoencoder finds a good representation space quicker than its contrastive counterpart. In fact, the top-1 accuracy when the length of representation is low (50 and 100 epochs) is actually higher for the denoising autoencoder, which also stabilizes much quicker during training. Thus, this method requires a smaller length of representation training to find a good representation space, a quality which can be desirable in settings where very large models are being used to create a representation space.

Thus, there are certain conditions where the denoising autoencoder architectures can be favorable to contrastive methods, while also leaving behind the dependence of negative pair learning.

4.3.4 Further Work

There are a few experiments that were left to of this work that could be extremely useful in evaluating this method's performance and understanding other positive-only alternatives to contrastive SSL. Although not explored in this paper, we leave these questions open for exploration in subsequent works.

First, the direct effect of batch size should be compared between the denoising autoencoder architectures and SimCLR. Initial experiments were conducted to answer this question, though the confounding effects of batch size and learning rate made it difficult to properly

assess this question.

Second, since we now know that faithful reconstructions are not inherently necessary to generate quality representations, it would be interesting to experiment with the type of loss calculation utilized for the denoising reconstruction term. Modifying this in conjunction with the embedding loss might give some intuition as to why certain types of embedding losses work better with certain kinds of reconstruction losses.

Lastly, although this question is largely explored with a non-denoising reconstruction loss in [8], it would be interesting to see the effects of using variational denoising training on the learned representation space. Performing this kind of training in combination with the augmentation invariance that we see with the denoising reconstruction loss may prove to create a representation space that can capture the distribution of various data sets.

5 Conclusions

The work lays out the framework for using denoising autoencoders to prevent collapse and generate representations in the context of self-supervised learning. Our method leaves behind the negative pair computations used in contrastive methods for a positive-only and simple to implement training scheme. Through taking the network to reconstruct the original image before the view was obtained from the augmentation pipeline, we show that the network learns transformation-invariant representations that require minimal training to achieve strong downstream performance. Although our method does not outperform the contrastive SimCLR on similar networks, it achieves comparable performance without the use of negative pairs.

We also show that there are some regimes, including high augmentation and low length representation training, where the denoising framework outperforms SimCLR by creating a higher quality representation space. Although there is still a gap between this method and the contrastive state-of-the-art, the research presented in this work sets a new path for further experiments and potentially more powerful frameworks based on the concept of denoising.

A Appendix

	Layer Type	Channels/ Nodes In	Channels/ Nodes Out	Kernel	Stride	Padding
Encoder	Conv2d HardTanh	3	32	3x3	1	1
	Conv2d MaxPool2d	32	32	3x3 2x2	1	1
	Conv2d HardTanh	32	64	3x3	1	1
	Conv2d MaxPool2d	64	64	3x3 2x2	1	1
	Conv2d MaxPool2d	64	512	3x3 2x2	1	1
Projector	Linear	8192	64			
Decoder (autoencoder only)	Linear HardTanh	64	8192			
	ConvT2d HardTanh	32	32	4x4	2	1
	ConvT2d Sigmoid	32	32	4x4	2	1

Table 3: Full network details. Decoder is only used in experiments where the trained network is an autoencoder.

References

[1] Philip Bachman, R Devon Hjelm, and William Buchwalter. "Learning Representations by Maximizing Mutual Information Across Views". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/ddf354219aac374f1d40b7e760ee5bb
Paper.pdf.

[2] Vivien Cabannes et al. *The SSL Interplay: Augmentations, Inductive Bias, and Generalization*. 2023. DOI: 10.48550/ARXIV.2302.02774. URL: https://arxiv.org/abs/2302.02774.

[3] Aoxue Chen. *Self-Supervised Learning via Autoencoder*. 2022.

[4] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 1597–1607. URL: https://proceedings.mlr.press/v119/chen20j.html.

[5] Xi Chen et al. "PaLI: A Jointly-Scaled Multilingual Language-Image Model". In: 2022.

[6] Xinlei Chen and Kaiming He. "Exploring Simple Siamese Representation Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 15750–15758.

[7] Debidatta Dwibedi et al. "With a Little Help From My Friends: Nearest-Neighbor Contrastive Learning of Visual Representations". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 9588–9597.

[8] Ioannis Gatopoulos and Jakub M. Tomczak. "Self-Supervised Variational Auto-Encoders". In: *Entropy* 23.6 (June 2021), p. 747. ISSN: 1099-4300. DOI: 10.3390/e23060747. URL: http://dx.doi.org/10.3390/e23060747.

[9] Jean-Bastien Grill et al. "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 21271–21284. URL: https://proceedings.neurips.cc/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf.

[10] Li Jing et al. "Understanding Dimensional Collapse in Contrastive Self-supervised Learning". In: (2021). DOI: 10.48550/ARXIV.2110.09348. URL: https://arxiv.org/abs/2110.09348.

[11] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[12] Yibo Yang Mufeng Tang and Yali Amit. "Biologically Plausible Training Mechanisms for Self-Supervised Learning in Deep Networks". In: *Frontiers in computational neuroscience* 16.789253 (2022). DOI: https://doi.org/10.3389/fncom.2022.789253.

[13] Alec Radford et al. "Learning Transferable Visual Models From Natural Language Supervision". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 8748–8763. URL: https://proceedings.mlr.press/v139/radford21a.html.

[14] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*. 2015. DOI: 10.48550/ARXIV.1506.01186. URL: https://arxiv.org/abs/1506.01186.

[15] Yonglong Tian et al. "What Makes for Good Views for Contrastive Learning?" In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 6827–6839. URL: https://proceedings.neurips.cc/paper/2020/file/4c2e5eaae9152079b9e95845750bb9ab-Paper.pdf.

[16] Diane Wagner et al. *On the Importance of Hyperparameters and Data Augmentation for Self-Supervised Learning*. 2022. DOI: 10.48550/ARXIV.2207.07875. URL: https://arxiv.org/abs/2207.07875.

[17] Jiahui Yu et al. *CoCa: Contrastive Captioners are Image-Text Foundation Models*. 2022. DOI: 10.48550/ARXIV.2205.01917. URL: https://arxiv.org/abs/2205.01917.

[18] Jure Zbontar et al. "Barlow Twins: Self-Supervised Learning via Redundancy Reduction". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 12310–12320. URL: https://proceedings.mlr.press/v139/zbontar21a.html.