Image Embedding using Contrastive Learning

Davis Arthur

1. Introduction

Contrastive learning is a technique used to produce a latent representation of data that is well-suited for downstream classification or clustering tasks. Contrastive learning starts from the assumption that some data points in a dataset are intrinsically similar to one another. A pair of similar points comprise a "positive pair" and a pair of dissimilar points comprise a "negative pair." The goal of contrastive learning is to find an embedding function that maps "positive pairs" close to one another and "negative pairs" far away from one another. The method used to determine positive and negative pairs depends on the setting in which contrastive learning is applied. This technique has proven to be particularly useful within the fields of computer vision and natural language processing.

Contrastive learning is most commonly applied in a self-supervised setting, where the ground truth labels of the training data are unknown. In this case, an artificially supervised task is created by first producing two or more augmented versions of each sample (e.g. two cropped versions of a particular image). A pair of augmented views are labelled positive relative to one another if they were generated from the same sample and negative if they were generated from different samples. More recently, contrastive methods have proven useful in the domain of supervised learning as well (Khosla et al., 2020). In this setting, two augmented views comprise a positive pair if they were produced from data points that share the same ground truth label. This means each augmented view may have multiple positive pairs in the dataset.

A number of different algorithms have been proposed for contrastive learning. These methods primarily vary in the construction of the augmented data set, the architecture of the embedding function, and the loss metric used to optimize the model. In this project, I have implemented three different contrastive learning algorithms, each using an open-source augmentation module from the PyTorch library and a simple convolutional neural network embedding scheme. The first two algorithms are applied in the self-supervised setting, and the third algorithm is applied in the supervised setting. I apply each algorithm to the Fashion MNIST dataset, and report their performance on downstream classification and clustering tasks.

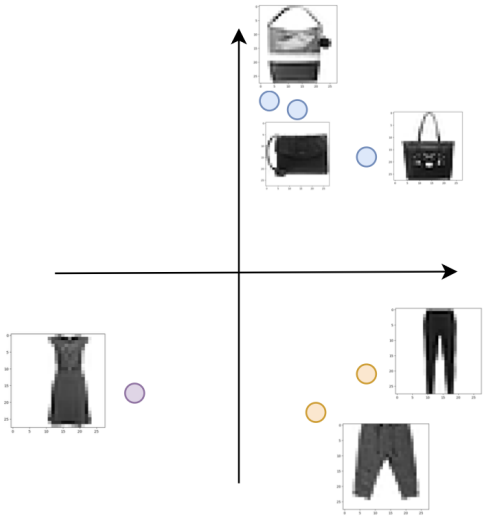


Figure 1. Two dimensional embedding of five images. This embedding would perform well on most contrastive loss functions assuming points of the same color form positive pairs, and points of different colors form negative pairs.

2. Problem Statement

Qualitatively, the goal of contrastive learning is to find an embedding that minimizes the distance between positive pairs while maximizing the distance between negative pairs. Unfortunately, there is no unanimous optimization problem that formalizes this description in the literature, since most practical contrastive learning algorithms are based on repeatedly minimizing a heuristic loss function defined over a subset of the data. Nevertheless, I have constructed a formal contrastive learning optimization problem that captures the key sentiments of each loss function I explored in this project.

To formulate this optimization problem, I introduce the following notation:

- $f : \mathbb{R}^m \to \mathbb{R}^p$: Embedding function.
- $\boldsymbol{X} \in \mathbb{R}^{n \times m}$: Input data set consisting of n samples each represented by an m dimensional vector.
- $\tilde{y}_{(\boldsymbol{x}_1, \boldsymbol{x}_2)}$: Contrastive label of the pair $(\boldsymbol{x}_1, \boldsymbol{x}_2)$. Equal to 1 for positive pairs, and -1 for negative pairs.

- $d : \mathbb{R}^p \times \mathbb{R}^p \rightarrow \mathbb{R}$: Function used to compute the distance between two embeddings.

Using this notation, the contrastive learning problem can be expressed as:

$$\underset{f}{\operatorname{argmin}} \sum_{(\boldsymbol{x}_1, \boldsymbol{x}_2) \in \boldsymbol{X}} \tilde{y}_{(\boldsymbol{x}_1, \boldsymbol{x}_2)} d(f(\boldsymbol{x}_1), f(\boldsymbol{x}_2)) \qquad (1)$$

3. Algorithms

3.1. Embedding Function

Most contrastive learning applications use an artificial neural network for embedding. When implementing the embedding network for my experiments, I followed an approach proposed by Khosla et. al. (Khosla et al., 2020). In their study, the embedding network is logically separated into an encoder and a projector. The encoder is a convolutional neural network (CNN) that maps each input image to a vector $\boldsymbol{r} \in \mathbb{R}^p$, and the projector is a small multilayer perceptron that maps $\boldsymbol{r}$ to a vector $\boldsymbol{z} \in \mathbb{R}^q$ (where $q \leq p < m$). During training, the output of the projector network is treated as the embedding, and during inference time, the output of the encoder network is treated as the embedding. Khosla et. al. do not explicitly explain the theory behind discarding the projector network during inference time, but it is well known that internal layers of neural networks often produce extremely informative representations of input data. I hypothesize that discarding the projector network and using the encoder embedding improves the stability of the contrastive model.

In the architecture proposed by Khosla et al., the outputs of the encoder and projector, $\boldsymbol{r}$ and $\boldsymbol{z}$, are both constrained to lie on the unit-hypersphere. This is useful because the inner product can then be used to measure the distance between two points. Similar embeddings should have an inner product close to 1, and distant embeddings should have an inner product close to -1. This constraint is also useful for deriving an important property of two of the loss functions discussed later in this section.

On large scale problems, a deep convolutional neural network (CNN) architecture such as ResNet or Visual Geometry Group (VGG) should be used for the encoder network. However, due to hardware constraints, I have opted to use a smaller CNN more appropriate for the Fashion MNIST dataset. When designing the network architecture, I tried to take inspiration from the more robust ResNet and VGG networks. In particular, my CNN is also characterized by a series of convolutional layers of increasing depth, and decreasing area. Additionally, nonlinearity is introduced to the network through pooling and rectified linear units (ReLU). The exact architecture of my encoder and projector networks are described in Figure 2 and Table 1.

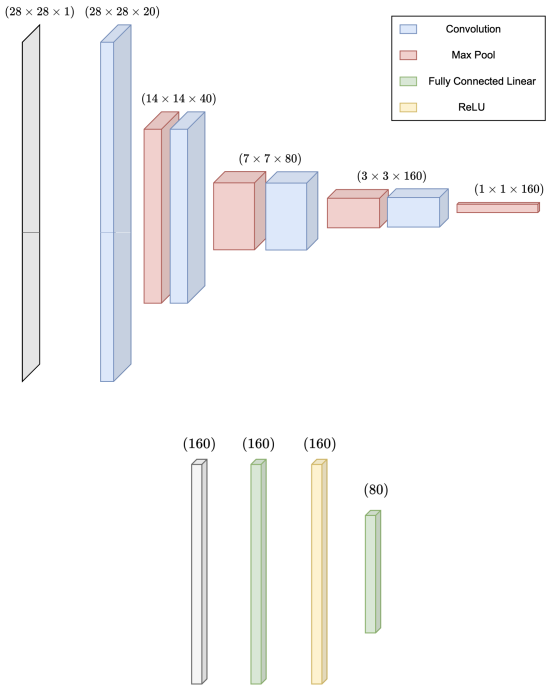


Figure 2. Architecture of encoder network (top) and projector network (bottom). The encoder maps a 28×28 grayscale image to a vector of length 160. The projector network maps a vector of length 160 to a vector of length 80.

OPERATION	INPUT	OUPUT	KERNEL
CONV2D	(28, 28, 1)	(28, 28, 20)	5×5
MAXPOOL2D	(28, 28, 20)	(14, 14, 20)	2×2
CONV2D	(14, 14, 20)	(14, 14, 40)	3×3
MAXPOOL2D	(14, 14, 40)	(7, 7, 40)	2×2
CONV2D	(7, 7, 40)	(7, 7, 80)	3×3
MAXPOOL2D	(7, 7, 40)	(3, 3, 40)	2×2
CONV2D	(3, 3, 80)	(3, 3, 160)	3×3
MAXPOOL2D	(3, 3, 160)	(1, 1, 160)	2×2
LINEAR	160	160	N/A
RELU	160	160	N/A
LINEAR	160	80	N/A

Table 1. Architecure of the encoder network (above dividing line) and projector network (below dividing line). The output of the encoder network is the input to the projector network.

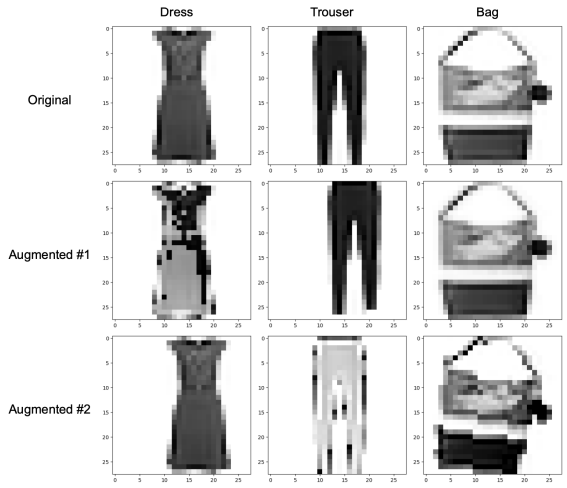


Figure 3. Three images from the Fashion MNIST dataset (top row). Two augmentations of each image (middle and bottom rows).

3.2. Augmentation Module

In both the self-supervised and supervised settings, an image augmentation scheme is used to create augmented views of each sample in the original data set. In the self-supervised setting, the augmentation scheme is necessary for creating positive samples. In the supervised setting, the augmentation scheme is useful for avoiding overfitting. I have chosen to create each augmented image using the PyTorch implementation of RandAugment (Cubuk et al., 2019) due to its simplicity and popularity in contrastive learning literature (Khosla et al., 2020). I have illustrated the behavior of RandAugment in Figure 3.

3.3. Triplet Loss

Triplet loss, as first described by Schroff (Schroff et al., 2015), is a simple heuristic loss function designed for contrastive learning. Triplet loss is defined over a triplet of points $(\boldsymbol{x}_a, \boldsymbol{x}_p, \boldsymbol{x}_n)$. The first point, $\boldsymbol{x}_a$, is referred to as the anchor sample, the second point, $\boldsymbol{x}_p$, is referred to as the positive sample, and the third point, $\boldsymbol{x}_n$, is referred to as the negative sample. The positive sample and the anchor sample comprise a positive pair, and the negative sample and the anchor sample comprise a negative pair. In my experiments, I apply triplet loss in the self-supervised setting, so each pair $(\boldsymbol{x}_a, \boldsymbol{x}_p)$ are two augmentations of the same image, and each pair $(\boldsymbol{x}_a, \boldsymbol{x}_n)$ are augmentations of different images. Let the embeddings of the anchor, positive, and negative samples be defined as f_a, f_p, and f_n respectively. Triplet loss is defined as follows:

$$L_{\text{tri}} = \max(||f_a - f_p||^2 - ||f_a - f_n||^2 + m, 0)$$

where m is a hyperparameter referred to as the margin.

For large datasets, computing the loss over all possible triplets can be computationally expensive and unnecessary. Instead, the embedding function is usually optimized over only a small batch of triplets each iteration. Interestingly, the convergence and performance of a contrastive model trained using triplet loss depends greatly on how triplets are sampled during batch gradient descent. To understand why, notice that the gradient of triplet loss is zero for any triplet that satisfies the following condition:

$$||f_a - f_p||^2 - ||f_a - f_n||^2 + m < 0$$

Such triplets are referred to as easy triplets because the positive sample and negative sample are relatively easy to classify based on their relationship with the anchor point. Since these triplets do not contribute to the gradient, they are not useful for improving the embedding function. Moreover, as the embedding function improves, easy triplets become increasingly common. In general, this issue can lead to extremely slow or sub-optimal convergence.

To avoid this shortcoming, many triplet loss algorithms propose strategies for identifying hard triplets during the construction of each batch (Schroff et al., 2015). Other loss functions, such as supervised contrastive loss (Section 3.4) have been designed with the explicit intention of avoiding the convergence issues of triplet loss. Since I have implemented triplet loss and self-supervised contrastive loss, I have chosen to follow the naive triplet sampling strategy described in Algorithm 1 when optimizing the embedding network using triplet loss.

Algorithm 1 Triplet Loss BGD

Randomly initialize network parameters $\boldsymbol{\theta}^{(0)}$
for $t = 1 \dots T$ **do**
  $L \leftarrow 0$
  $\boldsymbol{X}_b \leftarrow$ Randomly sample a batch from $\boldsymbol{X}$
  for $i = 1, \dots, b$ **do**
    $\boldsymbol{x}_a, \boldsymbol{x}_p \leftarrow$ Augmented versions of ith sample in $\boldsymbol{X}_b$
    $\boldsymbol{x}_n \leftarrow$ Augmented version of any other $\boldsymbol{X}_b$ sample
    $L \leftarrow \max(||f_a - f_p||^2 - ||f_a - f_n||^2 + m, 0)$
  end for
  $\boldsymbol{g} \leftarrow$ Compute $\nabla_{\boldsymbol{\theta}^{(t)}} L$ using backpropagation
  $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma^{(t)} \boldsymbol{g}$
end for

3.4. Self-Supervised Contrastive Loss

Tian et al. introduce the self-supervised contrastive loss function, which naturally extends triplet loss to include more than one negative sample (Tian et al., 2019). The other major difference is that self-supervised contrastive loss measures the similarity of two embeddings by their inner product instead of the Euclidean distance between them. Given a tuple of data points containing an anchor

sample, $\boldsymbol{x}_a$, a positive sample, $\boldsymbol{x}_p$, and $b-2$ negative samples, $(\boldsymbol{x}_n^{(1)}, \ldots, \boldsymbol{x}_n^{(b-2)})$, self-supervised contrastive loss is defined as follows:

$$\begin{aligned} L_{\text{ssc}} &= -\log\left(\frac{\exp(f_a^T f_p/\tau)}{\sum_{j\in\mathcal{A}}\exp(f_a^T f_j/\tau)}\right) \\ &= \log\left(\sum_{j\in\mathcal{A}}\exp(f_a^T f_j/\tau)\right) - \frac{1}{\tau} f_a^T f_p \end{aligned}$$

where $\mathcal{A}$ is the set containing all samples in the tuple excluding the anchor point, and τ is a hyperparameter referred to as temperature.

As indicated by the name, self-supervised contrastive loss is designed for the self-supervised setting, which means the anchor sample and positive sample are created by augmenting the same image, and the all negative samples are created by augmenting other images. Similar to triplet loss, the embedding function is typically optimized over a small batch of tuples during each iteration. Fortunately, the convergence of a contrastive model trained using self-supervised contrastive loss is not very sensitive to the tuple sampling strategy.

The stability of self-supervised contrastive loss is due to a property Khosla et al. refer to as intrinsic hard sample mining. Qualitatively, this property means the gradient of the loss function over a particular tuple is substantially more sensitive to samples that are hard to classify than samples that are easy to classify. This means that if b is large enough, most tuples will contribute to the gradient, since most tuples will contain at least one hard sample. It is worth noting that this property holds under the assumption that the embedding function embeds each sample onto the unit hypersphere. This assumption is relatively mild, since an embedding scheme of this nature is common when embedding similarity is measured using the inner product.

To mathematically derive the intrinsic hard sample mining property, first let $\mathcal{N}$ be the set of all negative samples in a tuple, and let P_{aj} be defined by the following equation:

$$P_{aj} = \frac{\exp(f_a^T f_j/\tau)}{\sum_{j\in A(i)}\exp(f_a^T f_j/\tau)}$$

Using this notation, the gradient of self-supervised contrastive loss with respect to the embedding f_a is given by:

$$\begin{aligned} \boldsymbol{\nabla}_{f_a} L_{\text{ssc}} &= -\frac{1}{\tau}\left(f_p - \sum_{j\in\mathcal{A}} f_j P_{aj}\right) \\ &= \frac{1}{\tau}\left(f_p(P_{ap}-1) + \sum_{n\in\mathcal{N}} f_n P_{an}\right) \end{aligned} \tag{2}$$

Next, express the constrained embedding f_a in terms of an unconstrained vector $\boldsymbol{w}_a$:

$$f_a(\boldsymbol{w}_a) = \frac{\boldsymbol{w}_a}{||\boldsymbol{w}_a||}$$

The derivative of f_a with respect to $\boldsymbol{w}_a$ is given by:

$$\begin{aligned} f_a'(\boldsymbol{w}_a) &= \frac{1}{||\boldsymbol{w}_a||}\left(\boldsymbol{I} - \frac{\boldsymbol{w}_a \boldsymbol{w}_a^T}{||\boldsymbol{w}_a||^2}\right) \\ &= \frac{1}{||\boldsymbol{w}_a||}\left(\boldsymbol{I} - f_a f_a^T\right) \end{aligned} \tag{3}$$

Finally, the gradient of the loss function with respect to $\boldsymbol{w}_a$ can be computed using chain rule.

$$\boldsymbol{\nabla}_{\boldsymbol{w}_a} L_{\text{ssc}} = f_a'(\boldsymbol{w}_a)^T \boldsymbol{\nabla}_{f_a} L_{\text{ssc}} \tag{4}$$

Plugging Equation 2 and 3 into Equation 4 yields:

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{w}_a} L_{\text{ssc}} =& \frac{1}{\tau||\boldsymbol{w}_a||}(P_{ap}-1)(f_p - (f_a^T f_p) f_a) \\ &- \frac{1}{\tau||\boldsymbol{w}_a||}\sum_{n\in\mathcal{N}} P_{an}(f_n - (f_a^T f_n) f_a)) \end{aligned} \tag{5}$$

Notice that each term in Equation 5 corresponds with a single positive or negative sample from the tuple. Additionally, the effect the jth sample has on the gradient computation is dependent on the factor $(f_j - (f_a^T f_j) f_a)$. The magnitude of this factor can be expressed as follows:

$$||f_j - (f_a^T f_j) f_a|| = \sqrt{1 - (f_a^T f_j)^2}$$

For easy positives, $f_a^T f_p \approx 1$ and for easy negatives $f_a^T f_n \approx -1$. In both cases, the magnitude of the sample factor is close to zero.

$$\begin{aligned} f_a^T f_p \approx 1 \Rightarrow ||f_p - (f_a^T f_p) f_a|| \approx 0 \\ f_a^T f_n \approx -1 \Rightarrow ||f_n - (f_a^T f_n) f_a|| \approx 0 \end{aligned}$$

Alternative for hard positives, $f_a^T f_p \approx 0$, and for hard negatives $f_i^T f_n \approx 0$. In both cases, the magnitude of the sample factor is close to one.

$$\begin{aligned} f_a^T f_p \approx 0 \Rightarrow ||f_p - (f_a^T f_p) f_a|| \approx 1 \\ f_a^T f_n \approx 0 \Rightarrow ||f_n - (f_a^T f_n) f_a|| \approx 1 \end{aligned}$$

This property implies that hard samples will have a much larger impact than easy samples in each gradient update.

To train the contrastive model using self-supervised loss, I used batch gradient descent in a similar manner to the triplet loss algorithm. The specific procedure for tuple construction is outlined in Algorithm 2.

Algorithm 2 Self-Supervised Contrastive Loss BGD

Randomly initialize $\boldsymbol{\theta}^{(0)}$
for $t = 1 \dots T$ **do**
  $L \leftarrow 0$
  $\boldsymbol{X}_b \leftarrow$ Randomly sample a batch from $\boldsymbol{X}$
  for $i = 1, \dots, b$ **do**
    $\boldsymbol{x}_a, \boldsymbol{x}_p \leftarrow$ Augmented versions of ith sample in $\boldsymbol{X}_b$
    $\mathcal{N} \leftarrow$ Augmented versions of all other $\boldsymbol{X}_b$ samples
    $\mathcal{A} \leftarrow \mathcal{N} \cup \{\boldsymbol{x}_p\}$
    $L \leftarrow L - \log\left(\frac{\exp(f_a^T f_p/\tau)}{\sum_{j\in\mathcal{A}}\exp(f_a^T f_j/\tau)}\right)$
  end for
  $\boldsymbol{g} \leftarrow$ Compute $\nabla_{\boldsymbol{\theta}^{(t)}} L$ using backpropagation
  $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma^{(t)}\boldsymbol{g}$
end for

3.5. Supervised Contrastive Loss

Khosla et. al propose the supervised contrastive (SupCon) loss function, which naturally extends self-supervised contrastive loss to include multiple positive samples in each loss computation (Khosla et al., 2020). SupCon loss is exclusively applied in the supervised setting where it easy to identify multiple positive pairs for each sample. SupCon loss is defined over a tuple of points with one anchor point, $\boldsymbol{x}_a$, and an arbitrary number of positive and negative samples. Let $\mathcal{P}$ denote the set of all positive samples in the tuple, and let $\mathcal{A}$ denote the set of all samples in the tuple excluding the anchor. The SupCon loss is given by:

$$L_{\text{sup}} = -\frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\log\left(\frac{\exp(f_a^T f_p/\tau)}{\sum_{j\in\mathcal{A}}\exp(f_a^T f_j/\tau)}\right)$$

Khosla et. al. show that SupCon also exhibits the hard sample mining property exhibited by self-supervised contrastive loss (Khosla et al., 2020).

Algorithm 3 SupCon Loss BGD

Randomly initialize $\boldsymbol{\theta}^{(0)}$
for $t = 1 \dots T$ **do**
  $L \leftarrow 0$
  $\boldsymbol{X}_b, \boldsymbol{y}_b \leftarrow$ Randomly sample a batch from $(\boldsymbol{X}, \boldsymbol{y})$
  for $i = 1, \dots, b$ **do**
    $\boldsymbol{x}_a, y_a \leftarrow i$th sample in $\boldsymbol{X}_b, \boldsymbol{y}$
    $\mathcal{P} \leftarrow$ All samples in $\boldsymbol{X}_b$ with label $y = y_a$
    $\mathcal{A} \leftarrow$ All samples in $\boldsymbol{X}_b$ excluding $\boldsymbol{x}_a$
    $L \leftarrow L - \frac{1}{|P|}\sum_{p\in\mathcal{P}}\log\left(\frac{\exp(f_a^T f_p)}{\sum_{j\in\mathcal{A}}\exp(f_a^T f_j)}\right)$
  end for
  $\boldsymbol{g} \leftarrow$ Compute $\nabla_{\boldsymbol{\theta}^{(t)}} L$ using backpropagation
  $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma^{(t)}\boldsymbol{g}$
end for

4. Experiments

I trained the embedding network on the Fashion MNIST dataset using each of the three algorithms introduced in the preceding section. The Fashion MNIST dataset consists of 70000 grayscale images of 28×28 pixel resolution. The dataset is divided into 10 classes based on the type of fashion apparel. Of the 70000 images, I used 60000 for training the embedding network. The other 10000 images were reserved for testing. Images from the Fashion MNIST dataset are depicted in Figures 1 and 3.

4.1. Self-Supervised Learning

In the self-supervised setting, I trained the embedding network once using triplet batch gradient descent (Algorithm 1) and once using self-supervised contrastive batch gradient descent (Algorithm 2). During the triplet loss experiment, I used a margin value of $m = 0.5$. I chose to use a relatively small margin, since I constrained each embedding to have a norm of exactly 1. This implies:

$$||f_a - f_p||^2 - ||f_a - f_n||^2 \leq 4$$

for all triplets. I also observed that this margin value outperformed $m = 0.25, 1.0$, and 2.0 in preliminary testing. In the self-supervised contrastive loss experiment, I used a temperature of $\tau = 0.1$. I chose this value because it was used in the original SupCon study (Khosla et al., 2020). Additionally, when I used a larger temperature of 1.0, the quality of the embedding failed to discernibly improve after 10 epochs of training.

I ran each algorithm for 30 epochs over the training set using a batch size of 30 points. The PyTorch implementation of the ADADELTA gradient descent optimizer was used to update the parameters based on the loss over each batch. This optimizer was chosen because it is robust to noisy gradient computations, and it does require a manually fine tuned learning rate (Zeiler, 2012).

Self-supervised contrastive learning is commonly used to generate an embedding of data useful for clustering. To evaluate the performance of Algorithm 1 and Algorithm 2, I clustered the embedding of the training set into 10 clusters after each epoch of training. I then computing the adjusted rand index between the clustering assignments and the ground truth labels. I used the Scikit-Learn library to compute the adjusted rand index and perform k-means. The adjusted rand index quantifies the similarity between two partitionings of a dataset, ignoring permutations (Yeung & Ruzzo, 2001). An adjusted rand index close to 1 indicates the partitionings are nearly identical, while a value close to 0 indicates the partitionings are uncorrelated. I report the results in Figure 4. Unfortunately, because k-means is a random algorithm, this metric is noisy even when evaluated twice on the same embedding. Nevertheless, the triplet

embedding and self-supervised contrastive embedding significantly outperform a clustering of the raw dataset.

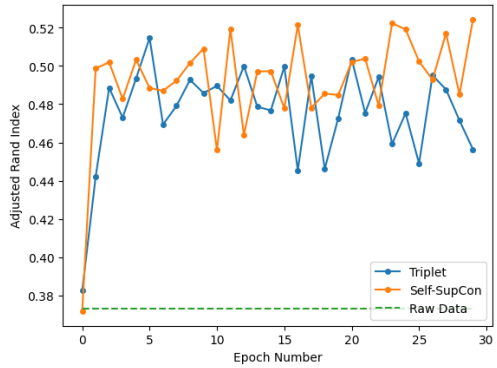


Figure 4. Adjusted rand index of the clustering produced by k-means on the embedded version of the training dataset produced by triplet loss (blue), and the embedded version of the training dataset produced by self-supervised contrastive loss (orange).

To visualize the quality of each embedding, I also used the Scikit-Learn implementation of principle component analysis (PCA) to project the final embedded version of the test dataset into a two-dimensional space. These projections are shown in Figure 5. The PCA projections are useful because they give intuition into which classes are harder to separate. For instance, images of ankle boots, sandals, and sneakers are mapped close to one another in each embedding. Meanwhile, images of trousers are mapped far away from all other images.

4.2. Supervised Learning

In the supervised setting, I ran the SupCon algorithm (Algorithm 3) for 30 epochs over the training set using a batch size of 30 points. Again, the PyTorch implementation of the ADADELTA gradient descent optimizer was used to update the embedding network parameters after each batch. I also used a temperature parameter of $\tau = 0.1$.

In the supervised setting, contrastive learning is commonly used for classification. Therefore, to evaluate the performance of the SupCon algorithm, I trained a linear SVM classifier on the embedded version of the training set. I then tested the accuracy of this classifier on the embedded version of the test set. In Figure 6, I report the test set accuracy of the SVM after each epoch of contrastive model training. I also report this metric for the triplet loss and self-supervised contrastive loss algorithms since the performance of a linear SVM may give intuition into the separability of the embeddings these algorithms produce.

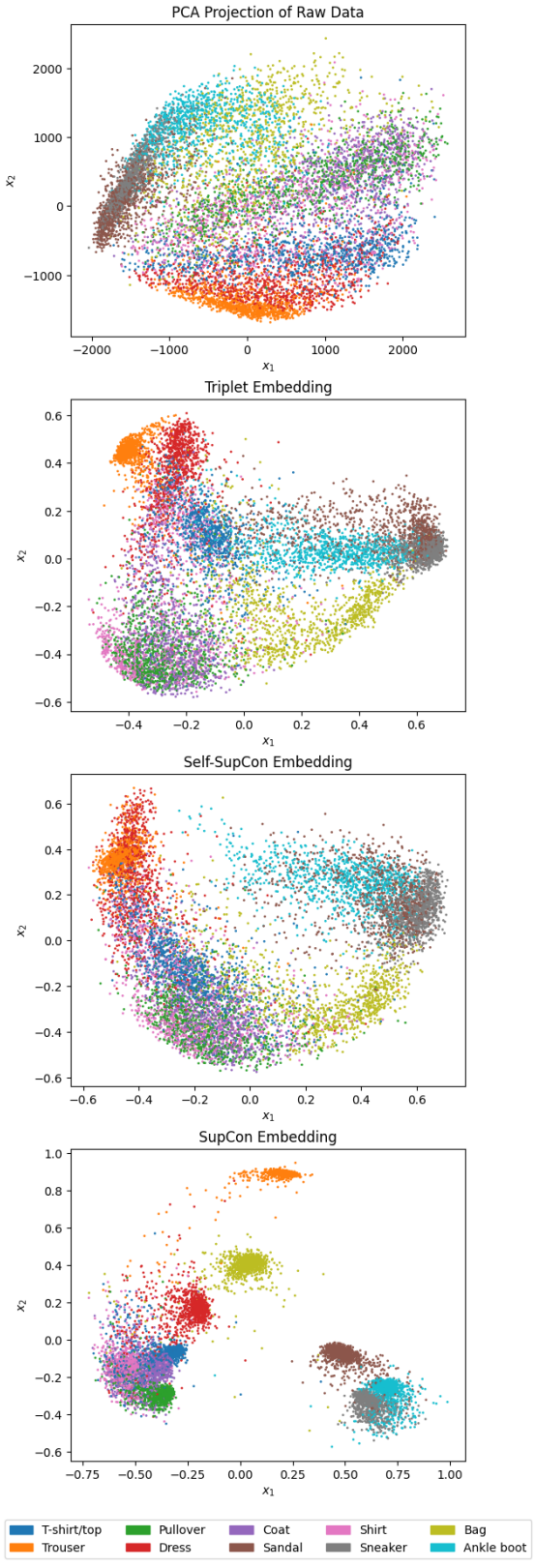


Figure 5. Two dimensional PCA projection of the raw test set and each embedded version of the test set.

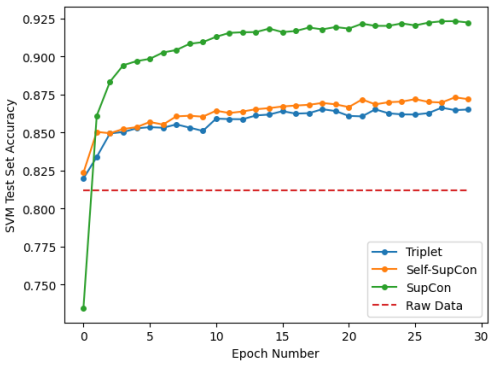
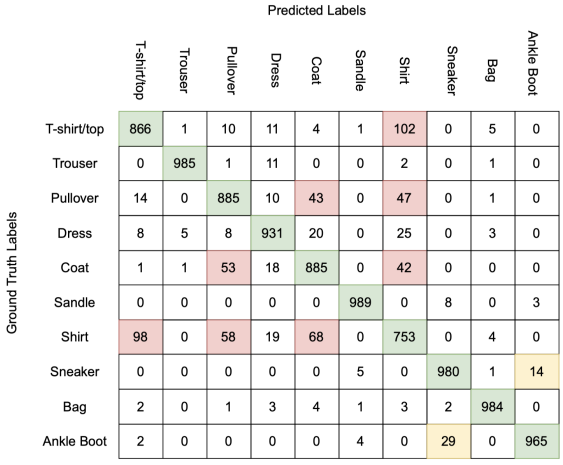


Figure 6. Test set accuracy achieved by an SVM classifier trained on the embedded version of the training dataset produced by triplet loss (blue), self-supervised contrastive loss (orange), and SupCon loss (green).

Finally, to provide a more in-depth perspective of the SupCon embedding, I also plotted the confusion matrix in Figure 7. The C_{ij} entry in the confusion matrix is defined as the number of images with ground truth label i and predicted label j. Notice that the most common misclassifications occur between the T-shirt/top class and the shirt class. Coats and pullovers are also commonly misclassified with each other or one of the two shirt classes. These misclassifications are unsurprising since there is a large overlap in the PCA projections of these classes in the SupCon embedding.



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandle	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/top	866	1	10	11	4	1	102	0	5	0
Trouser	0	985	1	11	0	0	2	0	1	0
Pullover	14	0	885	10	43	0	47	0	1	0
Dress	8	5	8	931	20	0	25	0	3	0
Coat	1	1	53	18	885	0	42	0	0	0
Sandle	0	0	0	0	0	989	0	8	0	3
Shirt	98	0	58	19	68	0	753	0	4	0
Sneaker	0	0	0	0	0	5	0	980	1	14
Bag	2	0	1	3	4	1	3	2	984	0
Ankle Boot	2	0	0	0	0	4	0	29	0	965

Figure 7. Confusion matrix produced by the class predictions of the SVM classifier trained on the SupCon embedding. Accurate predictions are highlighted in green. Common misclassifications are highlighted in yellow and red.

5. Conclusion

Contrastive learning is an efficient and powerful method for extracting informative features from a labelled or unlabelled dataset. In the self-supervised setting, I showed that triplet loss and self-supervised contrastive loss can be used to train a simple CNN to produce a feature rich low-dimensional embedding of the Fashion MNIST dataset. On a downstream classification task, the triplet and self-supervised contrastive embeddings achieved 5.3% and 6.0% advantages respectively over the raw dataset. In the supervised setting, the benefits of contrastive learning were even more expressed. On the same classification task, the SupCon embedding outperformed the raw dataset by 11.0%.

I am relatively surprised that the difference in performance of triplet loss and self-supervised contrastive loss was not larger, given that triplet loss has known convergence issues. I believe this result may be an artifact of the simplicity of the fashion MNIST dataset. In the future, I am interested in applying both contrastive learning algorithms to a larger, more challenging dataset such as CIFAR10. To handle the increased complexity, I would also opt to use a larger, more standardized encoder network architecture such as VGG11 or ResNet18. I attempted to perform such experiments during the course of this project, but unfortunately the training times were prohibitively large on my current hardware.

Finally, I am interested in experimenting with the augmentation module. Throughout this project, I used the PyTorch implementation of RandAugment to create the augmented versions of each image (Cubuk et al., 2019). However, I could create my own augmentation scheme to investigate which transformations (e.g. random horizontal flip, cropping, Guassian blur, etc.) are most beneficial to the contrastive model. Alternatively, on the CIFAR10 or ImageNet dataset, I could use the AutoAugment scheme introduced by (Cubuk et al., 2018).

References

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data, 2018. URL https://arxiv.org/abs/1805.09501.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical automated data augmentation with a reduced search space, 2019. URL https://arxiv.org/abs/1909.13719.

Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., and Krishnan, D. Supervised contrastive learning, 2020. URL https://arxiv.org/abs/2004.11362.

Schroff, F., Kalenichenko, D., and Philbin, J. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.

doi: 10.1109/cvpr.2015.7298682. URL https://doi.org/10.1109%2Fcvpr.2015.7298682.

Tian, Y., Krishnan, D., and Isola, P. Contrastive multiview coding, 2019. URL https://arxiv.org/abs/1906.05849.

Yeung, K. Y. and Ruzzo, W. Details of the adjusted rand index and clustering algorithms supplement to the paper "an empirical study on principal component analysis for clustering gene expression data" (to appear in bioinformatics). *Science*, 17, 01 2001.

Zeiler, M. D. Adadelta: An adaptive learning rate method, 2012. URL https://arxiv.org/abs/1212.5701.