



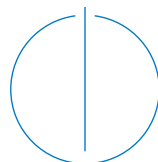
DEPARTMENT OF INFORMATICS

Technische Universität München

Bachelor's Thesis in Informatics

**Comparison of distance metrics for MDS based  
NLDR using CNNs**

Eric Fuchs





DEPARTMENT OF INFORMATICS

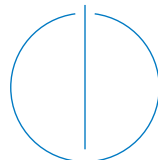
Technische Universität München

Bachelor's Thesis in Informatics

**Comparison of distance metrics for MDS based  
NLDR using CNNs**

**Vergleich von Distanzmaßen für MDS-basierte  
NLDR mit CNNs**

Author: Eric Fuchs  
Supervisor: Prof. Dr. Hans-Joachim Bungartz  
Advisor: Severin Reiz, M.Sc.  
Submission Date: 17.02.2020



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 17.02.2020

Eric Fuchs

---

## Abstract

The L1 and L2 distance metrics can be used when training convolutional neural nets to perform nonlinear dimensionality reduction on image datasets, generating embedded spaces in a similar manner as with multidimensional scaling. The choice between them is often made arbitrarily. We trained encoder/decoder network pairs as Regressors, Autoencoders, Siamese networks, and with a triplet loss before applying them to Classification, Outlier detection, Interpolation, and Denoising. The experimental results were interpreted, subjectively where necessary, leading to the conclusion that using the euclidean or the manhattan distance during training matters less than the choice of training configuration. The L2 distance appeared minimally favorable.

# Contents

<b>Abstract</b>	<b>iv</b>
<b>1. Introduction</b>	<b>1</b>
<b>2. Theory</b>	<b>2</b>
2.1. Principal Component analysis . . . . .	2
2.2. Multidimensional scaling . . . . .	2
2.3. Isomap . . . . .	3
<b>3. Implementation</b>	<b>5</b>
3.1. The Fashion-MNIST dataset . . . . .	5
3.2. Used Software . . . . .	7
3.3. Convolutional Neural Networks . . . . .	7
3.4. Regression on precomputed embedding . . . . .	11
3.5. Autoencoder . . . . .	13
3.6. Siamese . . . . .	14
3.7. Triplets . . . . .	15
<b>4. Evaluation</b>	<b>18</b>
4.1. Classification . . . . .	18
4.2. Outlier detection . . . . .	19
4.3. Interpolation . . . . .	21
4.4. Denoising . . . . .	23
<b>5. Conclusion</b>	<b>26</b>
<b>Appendix</b>	<b>27</b>
<b>A. Reprojected Images</b>	<b>28</b>
<b>Bibliography</b>	<b>28</b>

# 1. Introduction

Many problems arise when dealing with high-dimensional data like images: The storage of the large amount of values is cumbersome, the high dimensionality increases the runtime of algorithms like nearest neighbor search, and the special relationship of sample vectors is often irrepresentative of their similarity in terms of the features of objects depicted in the images.

Multidimensional scaling (MDS) is a class of nonlinear dimensionality reduction (NLDR) techniques that offer a solution to this problem. Algorithms like Isomap can find low dimensional coordinates for samples so that their spacial relationship better reflects their similarity.

Shortcomings of this approach are the large memory requirement that usually scales unfavorably with the number of samples in the dataset, and the inability to map new data into or back out of the discovered embedded space.

Convolutional neural networks (CNN) can be trained to solve the out-of-sample extension problem given such an embedding. By defining loss functions to mimic the optimization goals of MDS algorithms, they can also learn embedded spaces directly, with minimal need for memory thanks to batched training.

In all of these approaches, some comparison of images in their original, high-dimensional space needs to take place. The L1 and L2 distance metrics are commonly picked for this function.

This work aims to compare their usefulness for this purpose. To this end, we first provide a theoretical background for multidimensional scaling and related algorithms. This is followed by information on the dataset, software, and neural network architecture used in the experiments, as well as the four employed training configurations that optimize the networks weights in different ways. Four use cases of dimensionality reduction are described next, with interpreted results of the eight trained pairs of encoder and decoder networks.

A summarization of these results is provided in the conclusion.

## 2. Theory

### 2.1. Principal Component analysis

Principal Component Analysis [13] is a popular method of dimensionality reduction. At its core is a simple matrix, multiplication with which is used to map data points to an embedded space. It assumes the data occupy a euclidean space, leaving no room for different distance metrics. Geometrically, PCA can be viewed as translation and rotation of the data. After centering the mean on the origin the data are projected onto a new set of orthogonal basis vectors, chosen to point in the direction in which the data exhibits the most variance, excluding variance along the previously selected vectors. Actual reduction in the number of dimensions is then achieved by cutting off all basis vectors after the desired dimensionality is reached.

One way of computing the matrix mapping vectors to their principal components is:

1. Given a matrix  $X$  in which every sample is recorded as one of  $N$  row vectors, compute the mean

$$\bar{x}_i = \frac{1}{N} \sum_{j=1}^N x_{ji}$$

and center the data matrix by subtracting it from all rows

$$x_{ij}^c = x_{ij} - \bar{x}_j$$

2. Calculate the covariance matrix of the centered data  $X^{cT} X^c$
3. Find the eigenvectors and eigenvalues of the covariance matrix, and assemble the eigenvectors - in order of decreasing magnitude of the corresponding eigenvalues - as column vectors into a matrix  $W$ .

By storing the mean of the data  $\bar{x}$  that was used to center it and the transformation matrix  $W$ , new data  $\tilde{x}$  can be mapped to a predetermined embedded space as  $(\tilde{x} - \bar{x}) \cdot W$ .

### 2.2. Multidimensional scaling

Multidimensional scaling [16, 10, 1] was largely developed in the field of psychology. There, experimental data was often obtained by asking human subjects to give numerical values for different attributes of a stimulus. Averaged over multiple subjects these

values would provide a dataset with a number of samples equal to the number of stimuli and number of dimensions equal to the number of attributes ready for analysis.

Problems with this approach include the inherent inaccuracy with which humans map abstract concepts to a numerical range, and the necessity for researchers to anticipate the exact number and kind of attributes necessary to analyze the data beforehand.

While the first can be mitigated by increasing the number of subjects, the second one can be solved by collecting information on the similarity or dissimilarity of pairs of stimuli instead of specific attributes of individual samples. MDS then provides a way to derive vector coordinates for the stimuli so that their pairwise euclidean distances approximately match the obtained dissimilarities, even if they contain erroneous values.

Numerous methodologies have been proposed for estimating the necessary number of dimensions and finding coordinates adhering to the distance matrix. As an example, one algorithm for finding an embedding with MDS [17] is as follows:

1. Given the  $N \times N$  distance matrix  $D$ , compute the double centered, squared distance matrix

$$B = -\frac{1}{2}HSH$$

where  $s_{ij} = d_{ij}^2$  is the element-wise squared distance matrix and  $h_{ij} = \begin{cases} 1-1/N & \text{if } i=j \\ -1/N & \text{else} \end{cases}$  is the so-called centering matrix.

2. Assuming the number of dimensions desired for the embedded space is  $m$ , compute the  $m$  largest positive eigenvalues  $\lambda_i$  of  $B$  and their corresponding eigenvectors  $\mathbf{e}_i$ .
3. Construct the matrix containing the final coordinates of the  $N$  samples as  $m$ -dimensional row vectors.

$$x_{ij} = (\mathbf{e}_j)_i \cdot \sqrt{\lambda_j}$$

### 2.3. Isomap

The Isomap algorithm [15] seeks to find an embedding that unravels a manifold presumed to exist in the higher dimensional data. To do this, the geodesic distances between data points are approximated. It consists of the following three steps:

1. Construct a neighborhood graph from the input data. This takes the form of a large matrix where every entry  $a_{ij}$  is the distance between samples  $i$  and  $j$  if  $i$  and  $j$  are neighbors, positive infinity otherwise. Two points  $i$  and  $j$  are considered neighbors either if the distance between them is smaller than a given threshold ( $\epsilon$ -Isomap) or if one is among the others  $k$  nearest neighbors ( $k$ -Isomap).
2. Find the shortest paths for all pairs of points on the graph. This replaces the infinite entries in the matrix with real numbers.



3. Perform regular MDS on the obtained distance matrix.

If the datapoints on the manifold are sufficiently dense, the global distances between neighboring points are nearly identical to their geodesic distances. After that, these connections between neighbored points form a lattice along the manifold. The shortest route along the edges of this lattice is then a reasonable approximation of the geodesic distance for points that are very far from each other.

## 3. Implementation

### 3.1. The Fashion-MNIST dataset

The Fashion-MNIST dataset [19] consists of 70 000 labeled images of various articles of clothing, split into a training set of 60 000 samples and a testing set of 10 000 samples. Each sample is assigned one of 10 categories (see Figure 3.1), and both the training set and the test set contain samples of every category in equal proportion. The Images are square, 28 by 28 pixel rasters of 8-bit grayscale values. The white Background is represented by the number 0, the color black is stored as 255.

As a preprocessing step, we converted the values to 32-bit floating point numbers and divided them by 255, effectively mapping them to the  $[0; 1]$ -Range. No further preprocessing was made.



Figure 3.1.: 100 Images of the Fashion-MNIST training set, 10 samples randomly selected for each of the 10 classes. These are, in columns from left to right: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.

## 3.2. Used Software

In this work, a number of preexisting software packages was used to implement and run the experiments. Most notably they include:

- Keras [3] (<https://keras.io/>)  
For the construction, training, and usage of neural networks.
- scikit-learn [14] (<https://scikit-learn.org/stable/>)  
For its implementations of PCA, Isomap, and nearest neighbor search and classification.

## 3.3. Convolutional Neural Networks

the Architecture of the encoder network (Figure 3.2a) was adapted from [8]. There, its hyperparameters like the number of layers and number of filters per layer were algorithmically chosen to provide reasonable accuracy on specific image classification tasks with a minimal number of trainable parameters.

While image classification and dimensionality reduction are different tasks, they both consist of detecting image features and encoding them in a small vector. It can therefore be assumed that a given network architecture is similarly suited for both.

Notable changes to the architecture proposed by [8] specifically for image classification on the fashion-MNIST dataset are:

- Increased number of output neurons in the last dense layer from 10 to 20
- Leaky ReLU activation functions
- No activation function (or a linear activation function) in the last layer

By foregoing an activation function in the last layer samples in the embedded space are not constrained to lie in a hypersphere around the origin or only have positive values. This is necessary when training the encoder on a precomputed embedding (see section 3.4) or on specified distances (see section 3.6).

The decoder network's architecture (Figure 3.2b) is, for the most part, symmetric to the encoder's. In place of the commonly used transpose convolution, regular forward convolution layers were employed. This avoids checkerboard artifacts in the generated images [12]. As inverse of the max pooling layers in the encoder serve Upsampling layers with a nearest-neighbor scaling function.

The activation function in the last layer of the decoder is a novel modified version of the hard-sigmoid activation function (Equation 3.1). Just like with leaky ReLU, the sections of

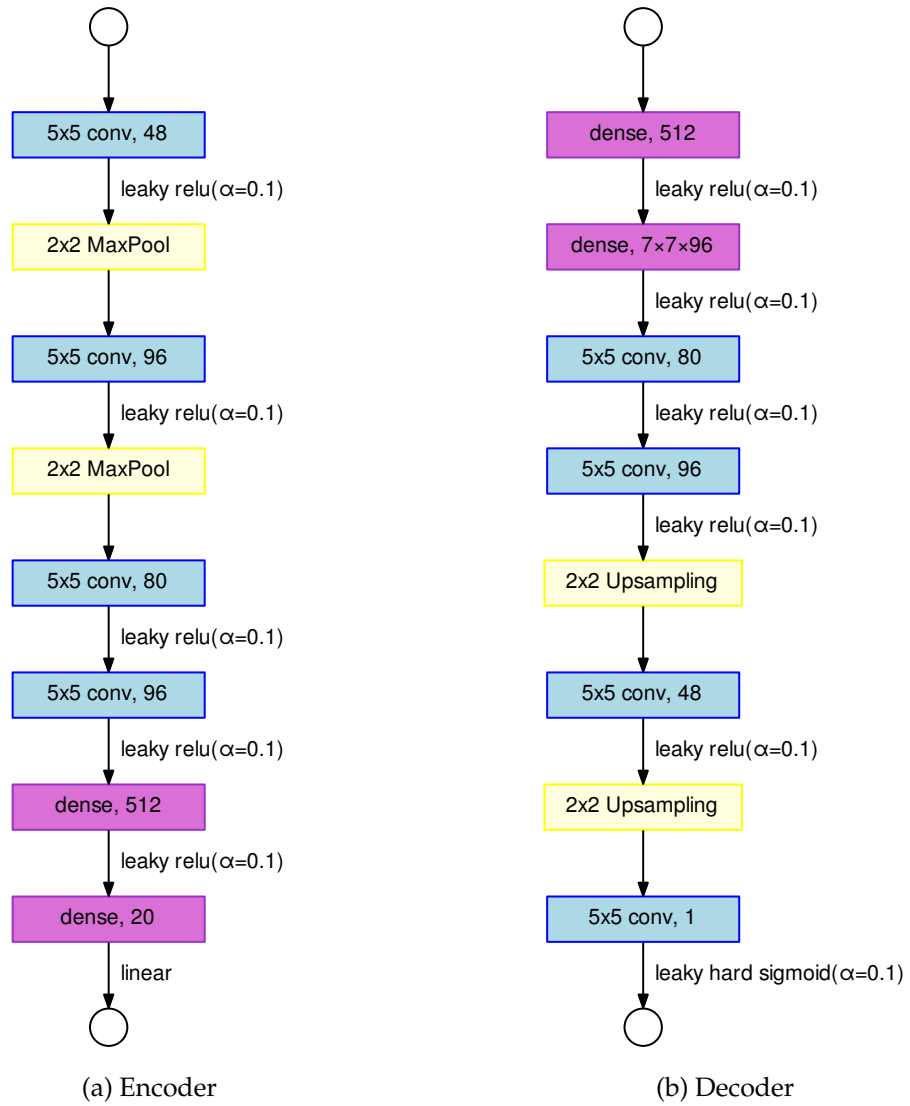


Figure 3.2.: Diagrams giving an overview of the used convolutional neural networks. For more detailed information, see Table 3.1 and Table 3.2

### 3. Implementation

---

the function that would otherwise be constant were slightly tilted to avoid losing gradient information during training due to zeros in the derivative.

$$\text{leakyHardSigmoid}(x, 0 \leq \alpha \leq 1) = \begin{cases} (0.2\alpha)x + (2.5 \times 0.2\alpha) & \text{if } x > 2.5 \\ (0.2\alpha)x + (1 - 2.5 \times 0.2\alpha) & \text{if } x < -2.5 \\ 0.2x + 0.5 & \text{else} \end{cases} \quad (3.1)$$

The network weights were initialized from a uniform random distribution within the range  $\left[-\sqrt{\frac{6}{n_l}}; \sqrt{\frac{6}{n_l}}\right]$  where  $n_l$  is the number of inputs of the layer being initialized, which has the same variance as the normal distributed initialization proposed by He et al. [5].<sup>1</sup>

During training, the Adam optimizer [9] is used to update the networks weights. It's parameters are set to the following commonly used<sup>2</sup> values: learning rate = 0.001,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\hat{\epsilon} = 1 \times 10^{-7}$ .

---

<sup>1</sup> Details on the `he_uniform` initialization can be found at [https://www.tensorflow.org/api\\_docs/python/tf/keras/initializers/he\\_uniform](https://www.tensorflow.org/api_docs/python/tf/keras/initializers/he_uniform)

<sup>2</sup> They are the default values used by Keras, see [https://www.tensorflow.org/api\\_docs/python/tf/keras/optimizers/Adam](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam). Kingma and Ba [9] originally recommended  $\hat{\epsilon} = 1 \times 10^{-8}$  instead.

### 3. Implementation

---

Table 3.1.: All hyperparameters of the encoder network’s layers. A Reshape-layer is implicitly needed before the first densely connected layer. The network encompasses a total of nearly 3 million trainable weights, as evidenced by the right-most column.

Layer type	Kernel size	Filter count	Input dimensions	Output dimensions	Parameter count
Convolution	5x5	48	28, 28, 1	28, 28, 48	1 248
Max Pooling	2x2		28, 28, 48	14, 14, 48	0
Convolution	5x5	96	14, 14, 48	14, 14, 96	115 296
Max Pooling	2x2		14, 14, 96	7, 7, 96	0
Convolution	5x5	80	7, 7, 96	7, 7, 80	192 080
Convolution	5x5	96	7, 7, 80	7, 7, 96	192 096
Dense		512	4704	512	2 408 960
Dense		20	512	20	10 260
Total:					2 919 940

Table 3.2.: All hyperparameters of the decoder network’s layers. A Reshape-layer is implicitly needed before the first convolutional layer. Like the encoder network, the decoder has just short of 3 million trainable weights.

Layer type	Kernel size	Filter count	Input dimensions	Output dimensions	Parameter count
Dense		512	20	512	10 752
Dense		4704	512	4704	2 413 152
Convolution	5x5	80	7, 7, 96	7, 7, 80	192 080
Convolution	5x5	96	7, 7, 80	7, 7, 96	192 096
Upsampling	2x2		7, 7, 96	14, 14, 96	0
Convolution	5x5	48	14, 14, 96	14, 14, 48	115 248
Upsampling	2x2		14, 14, 48	28, 28, 48	0
Convolution	5x5	1	28, 28, 48	28, 28, 1	1 201
Total:					2 924 529

### 3.4. Regression on precomputed embedding

The most straightforward way of training the encoder and decoder networks is to first obtain a definitive embedding of the training data from an algorithm like those discussed in chapter 2. Subsequently, The two neural networks can be separately trained to map samples of the dataset in both directions using the embedding as desired output for the encoder and input for the decoder. This way of using convolutional neural networks for out-of-sample extension was previously explored by Mishne et al. [11].

Computing the embedding requires large computational resources as discussed in chapter 2, but having the desired output defined directly as well as minimizing the number of layers between it and the input simplifies the backpropagation of errors and speeds up training.

Given the assumption that euclidian distance between points in embedded space is meaningful, mean squared error is a natural choice for the loss function used when training the encoder.

The decoder's loss function compares data in image-space. There, only the manhattan distance and the euclidean distance are in the scope of this work. The former can be implemented by using the mean absolute error as a loss function, the latter - as in the encoder's case - with the mean squared error loss.

To obtain an embedding we chose the Isomap algorithm with the 5 nearest samples - as measured by L2 - being considered neighbors (see section 2.3). To save computational resources, Isomap was only applied to a subset of the available training data. For each of the 10 labels, 3000 samples were randomly selected, effectively cutting the dataset in half.

The resultant points in embedded space were centered by subtracting the mean vector from all data points and scaled by dividing all values by the variance of the flattened data.

Both networks were trained on batches of 32 samples. Training of the encoder was done after 75 epochs, the decoder was trained for 25 epochs. An epoch in this case refers to using every one of the 30 000 samples exactly once - in random order - during training. Since the number of data points is not evenly divisible by the batch size, the last batch in every epoch contains less than 32 samples.



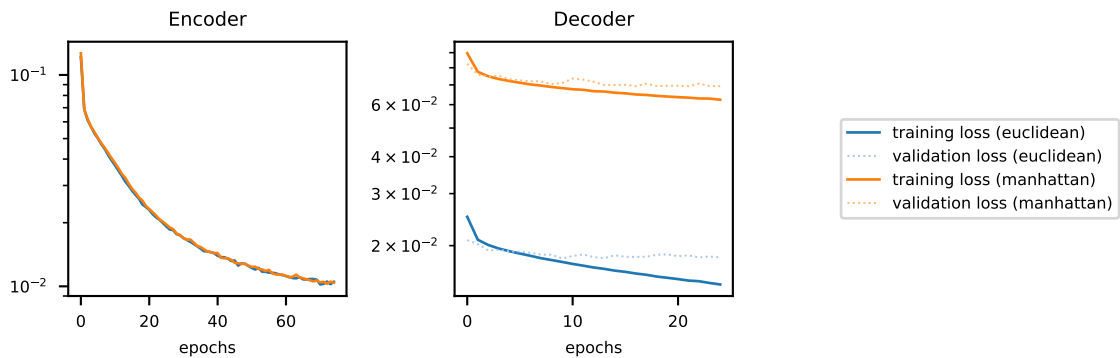


Figure 3.3.: Logarithmic plots of the loss computed in the last batch of every epoch. Since the encoder in this case does not depend on the chosen distance metric, the loss plots differ only mildly due to different random weight initializations and shuffled training data. For the decoders plot, a validation loss was computed after every epoch of training by using the 10 000 images of the test set without updating the weights. The two distance metrics exhibit a difference in scale, but otherwise seem to lead to similar training progress. The validation loss visibly stops improving after very few epochs. The risk of overfitting is likely exasperated by the reduction in dataset size that was necessary to obtain the Isomap embedding.

### 3.5. Autoencoder

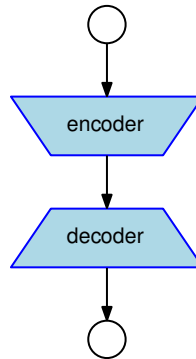


Figure 3.4.: Training configuration for the autoencoder: The encoder and decoder networks are connected sequentially, forming a larger network that maps images into and back out of the embedded space. This enables training to minimize the reprojection error without any additional constraints regarding the embedding.

By attaching the last layer of the encoder as input to the first layer of the decoder a new, bigger neural network (Figure 3.4) is created. This combined network is then trained to approximate the identity function by providing it with identical samples from the training data as both input and desired output. [6]

Given that the loss function for the autoencoder compares data in image-space just like in the decoder above, the same ones were included in the experiments.

Training is done on all 60 000 samples of the training set, in random order, and split into 1875 batches of size 32. It was found that 75 epochs of training were sufficient.

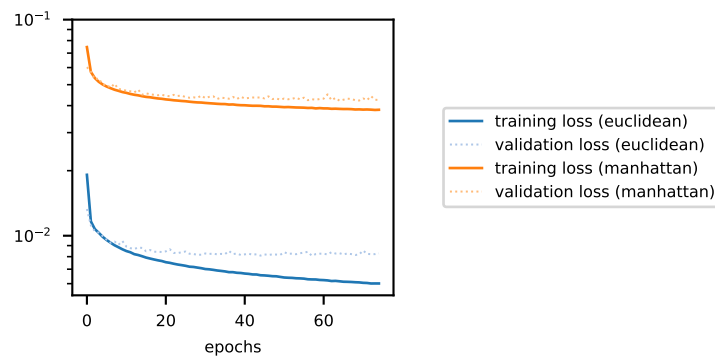


Figure 3.5.: Logarithmic loss plots showing the training progress of the autoencoder.

### 3.6. Siamese

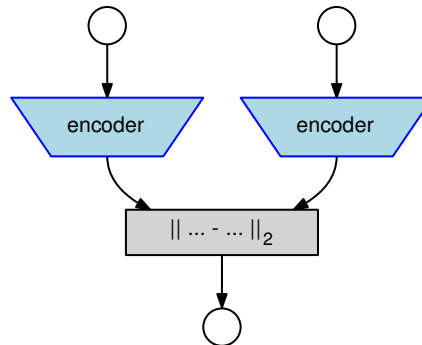


Figure 3.6.: Siamese training configuration: Two instances of the encoder with shared weights map two input images into the embedded space. The euclidean distance between these points is computed as part - and output - of the larger neural network being trained.

Just like in classical multidimensional scaling, the goal of training a Siamese Neural Network [2, 4] is to find an embedding in which the pairwise euclidean distances between samples approximate dissimilarities precomputed or known for the training data.

The network being trained consists of two instances of the same encoder network with shared weights whose outputs are fed into a euclidean distance operator whose output in turn is used as final output of the bigger network (see Figure 3.6). This way it predicts - for two samples given as input - their euclidean distance in the embedded space.

For every epoch, two random derangements of the training set's samples are generated and cut into batches of 512 pairs. The last batch of every epoch is necessarily smaller. Since the only distance metrics compared in this work are the euclidean and the manhattan distance, which are cheap to compute, the desired output values the network is being trained to approximate can be generated on-demand for the image pairs in every batch.

The mean squared error was chosen as loss function to avoid learning overly accurate distance relationships on a subset of image-pairs at the cost of outliers by penalizing higher errors more strongly.

The training process was finished after 50 epochs.

Subsequently, the now trained encoder was used to map all images in the training set into the embedded space. The resulting coordinates along with their corresponding images were then used to train the decoder network.

The decoder's loss function is chosen to be equivalent to the distance metric being considered. Training with a batch size of 32 is completed after 75 epochs.

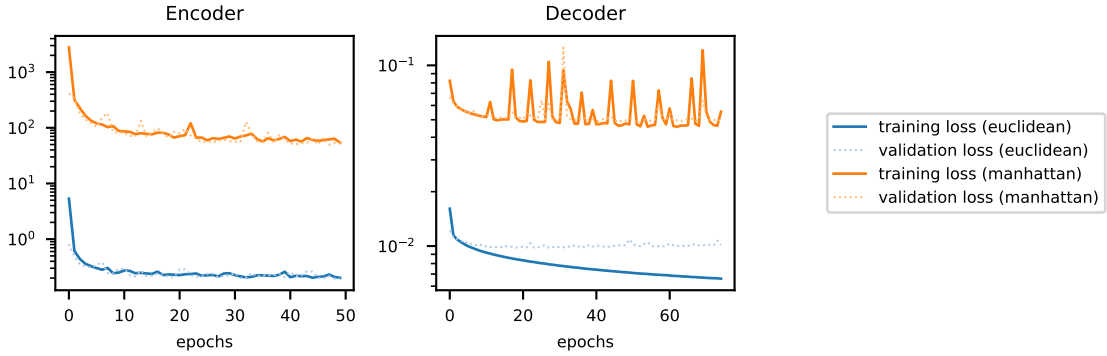


Figure 3.7.: Logarithmic loss plots of the siamese networks training progress. The larger scale of the manhattan distance destabilizes the optimization of the decoder.

### 3.7. Triplets

Instead of approximating given distances in embedded space like the Siamese Neural Network, the triplet loss facilitates replication of known neighborhood relationships in the data [7]. Using the neighborhood information, triplets of inputs (an “anchor”, a “puller”, and a “pusher”) are assembled so that anchor and puller are neighbors and anchor and pusher are not. These are fed into three instances of the encoder network with shared weights, allowing the computation of the euclidean distances between the two pairs in embedded space (see Figure 3.8). To preserve the neighborhood relationship, the distance between anchor and pusher should be larger than the distance between anchor and puller. In order to optimize toward this goal a comparator function turning the two distances into a minimizable loss is needed. This comparator, together with the two distance computations, is then referred to as a “triplet loss”.

From the different comparator functions established in previous works we chose the one proposed by Wohlhart and Lepetit [18]:

$$c_{\text{triplet}}(s_{\text{anchor}}, s_{\text{puller}}, s_{\text{pusher}}) = \max\left(0, 1 - \frac{\|\text{enc}(s_{\text{anchor}}) - \text{enc}(s_{\text{pusher}})\|_2}{\|\text{enc}(s_{\text{anchor}}) - \text{enc}(s_{\text{puller}})\|_2 + m}\right),$$

where  $s_i$  are the images in a triplet and  $\text{enc}(\cdot)$  denotes the encoder. The free parameter  $m$  sets a margin by which the pusher needs to be further away from the anchor than the puller to satisfy the loss function. In this experiment we set  $m = 0.01$ .

During training, we observed that using solely the triplet loss leads to continuous expansion of the cloud of points projected into embedded space from epoch to epoch. To counter this, we implemented an additional loss to directly oppose this phenomenon by minimizing the euclidean distance of all embedded points from the origin. This is equiv-

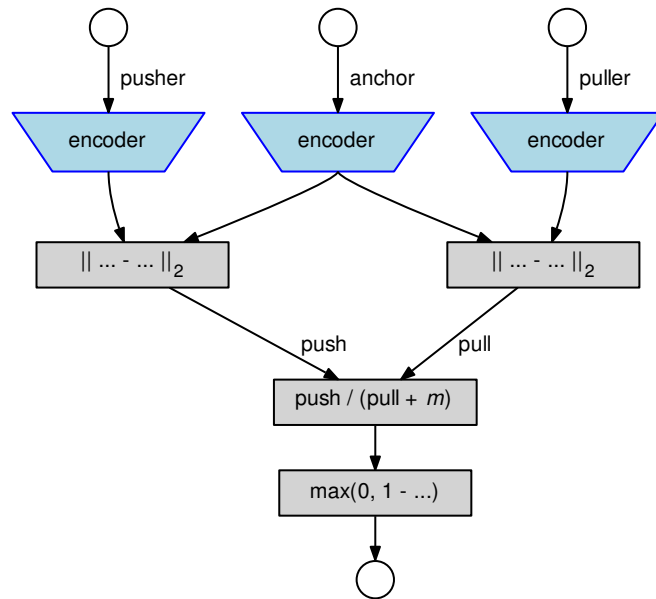


Figure 3.8.: Training configuration with triplet loss: Three input images (puller, anchor, and pusher) are fed into three instances of the encoder network with shared weights. The euclidean distances in embedded space between the anchor images' coordinates and those of the other two are computed and then compared in a loss function. If the distance between anchor and pusher is higher than that between anchor and puller by some margin  $m$ , the objective of the loss function is fulfilled.

### 3. Implementation

alent to adding an L2 activity loss to the last layer of the encoder. The exact definition of the activity loss used is the following:

$$c_{\text{activity}} = w \times (\| \text{enc}(s_{\text{anchor}}) \|_2 + \| \text{enc}(s_{\text{puller}}) \|_2 + \| \text{enc}(s_{\text{pusher}}) \|_2)$$

Through the weighting parameter  $w$  the expanding and contracting effects of the two losses can be balanced to cancel out. We found a weight value of  $w = 1 \times 10^{-5}$  to yield acceptable results.

We used both distance metrics we compare in this work separately to determine the five nearest neighbors in image-space to every sample in the training set. Images are in this case not considered to be their own neighbors. One training epoch then consists of all 60 000 training images in random order as anchors, one of their 5 neighbors chosen with equal probability as pullers, and one non-neighbor randomly chosen from the 59 994 other samples as “pusher”. The encoder was trained on 75 of these epochs, cut into batches of 512 triplets.

The decoder was trained for 100 epochs in exactly the same manner as for the siamese network (see section 3.6).



Figure 3.9.: Logarithmic plots of the triplet loss during training and on the validation set (left), the activity loss and the sum of the losses that the optimizer aims to minimize (middle), and the training and validation loss computed while subsequently fitting the decoder (right).

## 4. Evaluation

The ability to map data to a lower dimensional embedded space and back has a variety of uses. In this chapter, a variety of these are introduced and used to evaluate the relative usefulness of using the euclidean or the manhattan distance metric to train convolutional neural nets in any of the previously introduced ways.

Since PCA is also capable of all the same basic operations - determining an embedding for a given dataset and mapping new data into and out of it - it will serve as a benchmark to compare the neural networks against.

Reprojection quality plays an important role in Interpolation (section 4.3) and Denoising (section 4.4), and therefore is not additionally considered by itself. For completeness, a few reprojected images of the fashion-MNIST test set are provided in Figure A.1.

### 4.1. Classification

A simple way to classify data given labeled examples is to find the example that is the most similar to the new sample and use the provided label for it as well. Algorithms exist to enable a nearest neighbor search without iterating over all provided data, but this limits the choice of distance metric, and storing the training data in its entirety is not desirable.

Mapping the labeled examples into a low-dimensional embedded space dramatically decreases the storage requirement and computational cost of the neighbor search with minimal decreases in accuracy.

The accuracy scores in Table 4.1 are just the percentage of images in the test set that were correctly labeled. When interpreting these results, it is important to keep in mind that nearest neighbor classification is sensitive to even small amounts of noise. An excellent example of this is the “precomputed” row: Since only the encoder network is relevant to classification and the chosen distance metric plays no role in its training (see section 3.4), the scores of the “Euclidean” and “Manhattan” columns in this row only differ due to the retraining of the same network on the same data with different random initializations.

The siamese network shows improved scores when trained using the euclidean rather than the manhattan distance metric, while for the triplets network the opposite is true. This suggests that distance metric and training setup can not be chosen entirely independently from one another.

In these experiments, the triplets networks score consistently - and independently of the chosen distance metric - worse than their siamese counterparts. A likely explanation of this is that they were not trained sufficiently (see Figure 3.9). The difficulty of training using

Table 4.1.: Accuracy of  $n \in \{1, 5, 10, 20\}$  nearest neighbor classification. The best and worst score are highlighted in bold. Results of  $n$ -NN in image space are labeled “reference”, all other results are computed in the 20-dimensional embedded spaces. The nearest neighbor search always uses euclidean distance. The categorization into “Euclidean” and “Manhattan” refers to the neural network training process (see chapter 3).

	Euclidean				Manhattan			
	$n = 1$	$n = 5$	$n = 10$	$n = 20$	$n = 1$	$n = 5$	$n = 10$	$n = 20$
reference	84.97	85.54	85.15	84.15				
PCA	82.12	84.45	84.77	84.33				
precomputed	<b>79.59</b>	82.21	82.84	82.58	79.95	82.50	82.96	82.83
Autoencoder	85.96	87.31	<b>87.91</b>	87.30	85.94	87.28	87.41	87.12
Siamese	83.95	85.74	85.92	85.48	82.43	84.67	84.93	84.53
Triples	81.03	83.11	83.06	82.77	81.87	84.17	84.09	83.74

the triplets loss compared to the siamese architecture is nonetheless a valuable result.

The autoencoder, while slightly better when trained using the L2 rather than the L1 metric, outperforms the reference scores using either of them. A possible explanation for this is that the autoencoder found an embedding that is based less on the L2 or L1 distance between samples and more on their similarity in terms of the basic shapes like edges and corners that its convolutional layers can easily detect and reproduce. This embedding then clusters samples of the same class together more clearly than they were in the original, high-dimensional image-space, regardless of the used distance metric.

The scores of the 8 trained and tested neural networks vary more pronounced by the training configuration that was employed than by the chosen distance metric. Still, for the autoencoder and the siamese network - the two best performing configurations in this experiment - the euclidean distance yielded slightly better results.

## 4.2. Outlier detection

Often, datasets contain a small number of samples that do not “fit in” with the rest. Their uniqueness and high dissimilarity to other data can complicate further processing, so the ability to automatically identify them is very useful. When interpreting the data as a cloud of points in multi-dimensional space, outliers can be defined as those points that inhabit regions of that space that exhibit a lower density of points.

For the same reasons as discussed in section 4.1 mapping the data into a lower-dimensional embedded space is advantageous.

In the embedded space, the sample density can be estimated through a nearest neighbor search. Interpreting the distance to the  $n$ -th nearest neighbor as the radius of a hyper-



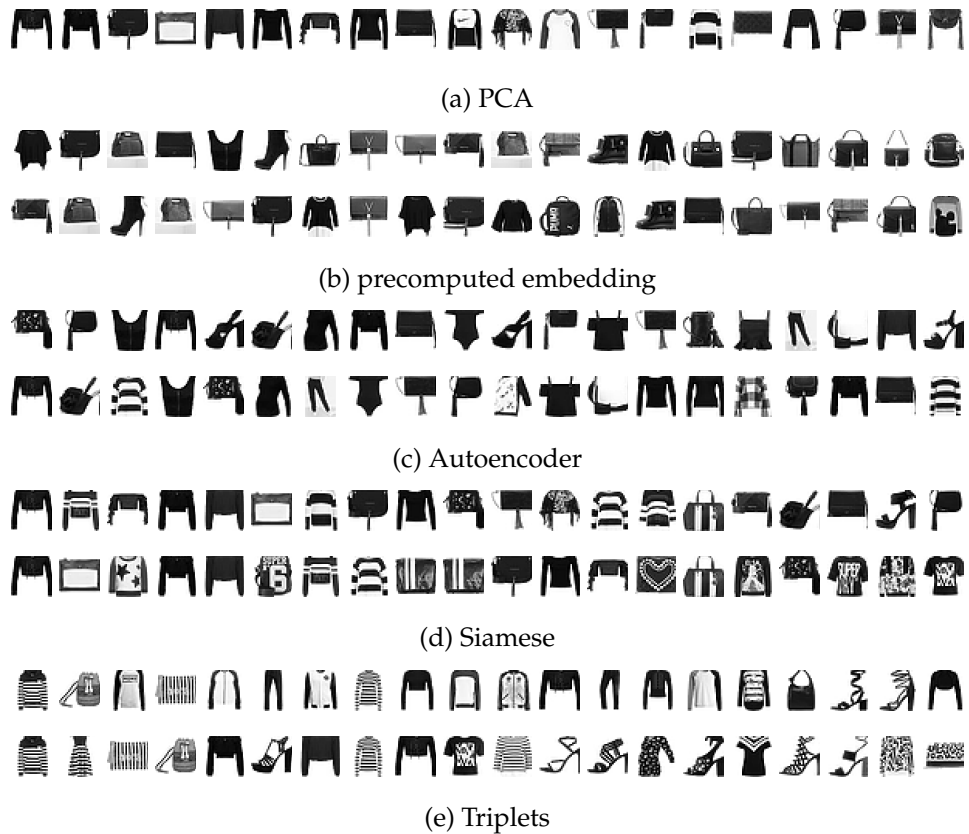


Figure 4.1.: Images identified as likely outliers in the fashion-MNIST training set. Density of samples is estimated through a 20-NN search in the embedded space. The 20 images with the lowest density are shown in every single row, with the density increasing from left to right. For the neural nets (b - e), the top and bottom row of images are obtained using the version trained with the euclidean and the manhattan distance metric, respectively. Compare Figure 3.1 for random images of the same set serving as examples of non-outliers.

sphere centered on the point in question, the density at that point can be approximated as  $n$  divided by the volume of the hypersphere. Through the free parameter  $n$  the maximum number of points in a cluster of outliers can be controlled.

For example: Setting  $n = 1$ , two samples that are close to each other but far away from all others would still have a high density and consequently not count as outliers. Setting  $n = 2$  in the same scenario expands the hyperspheres all the way to the other points' cluster. The significantly lower density then allows the two points to be identified as outliers.

The number of samples necessary for a cluster to count as legitimate part of the data rather than a collection of outliers is subjective. We settled on  $n = 20$ .

By sorting the dataset in order of increasing density, likely outliers can quickly be reviewed. Since the density, as defined above, is inversely proportional to - and only to - the distance to the  $n$ -th nearest neighbor, its calculation can be skipped. Sorting is then done in order of decreasing distance.

Figure 4.1 shows this method of outlier detection being applied to the Fashion-MNIST training set. Since it lacks definitive labeling of outliers, the results can only be interpreted subjectively. Comparison with the randomly chosen images of the dataset in Figure 3.1 suggests that all methods successfully find images that are unusual in some way, be it shape, texture, pose, or the inclusion of a background.

The two rows of 4.1b show the effect of retraining a network. A number of images are identified as outliers both times, if in a slightly different order.

The same observation can also be made in the pairs of rows 4.1c - 4.1e, which suggests the choice of distance metric between L1 and L2 plays a nearly negligible role in the result of outlier detection.

While the different training configurations show a great amount of variability in the kind of images that are classified as outliers, the siamese network (4.1d) exhibits noticeably similar results to PCA (4.1a), especially when using the L2 metric (top row). This is to be expected, given that both optimize for the preservation of euclidean distances in the embedded space.

### 4.3. Interpolation

In some datasets, interpolation between samples is very straightforward. Consider, for example, a dataset describing different rectangles by storing their width and height as two-dimensional vectors. The average of two such vectors then perfectly represents a new rectangle that is a mixture of the two originals. In fact, any point on the straight line connecting the samples in feature-space can be used to create rectangles relating to the other two to varying degrees.

With datasets consisting of images however, results of this type of interpolation are often less than desirable. In a dataset consisting of images of rectangles, linear combinations of sample vectors do not lead to an image of a rectangle at all. Rather, the two source

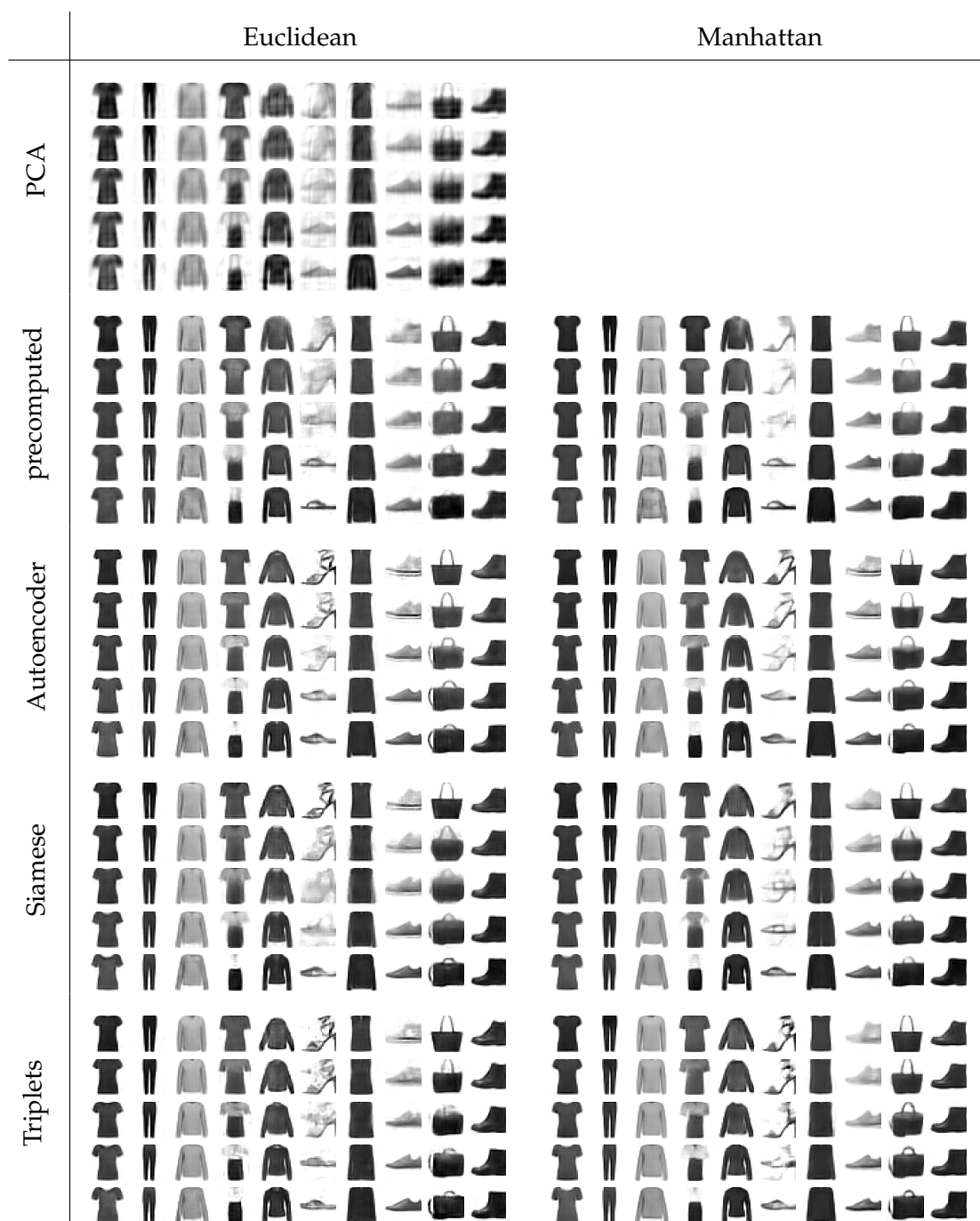


Figure 4.2.: Interpolated images of the fashion-MNIST training set. Rows show 0%, 25%, 50%, 75%, and 100% linear combinations of vectors in embedded space after reprojection.

rectangles appear overlaid on top of another with transparency, creating a sort of “plus”-shape.

Ideally, a nonlinear dimensionality reduction method would be able to infer the feature-space of the first example from the images of the second. Meaningful interpolation can then be achieved by

1. mapping the source Images to the embedded space,
2. linearly interpolating between the low-dimensional vectors, and
3. reprojecting the newly obtained vectors into the original image space.

This process was applied using all 8 trained neural nets and, for reference, PCA. Figure 4.2 shows the generated results. For each of the 10 classes, 2 images were randomly selected and interpolated between. Interpolation between different classes was avoided deliberately, because the quality of those is difficult to judge (What *should* an object halfway between a T-shirt and a shoe look like?). The two samples in the “Sandal”-class (6-th column from the left) nevertheless provide insight into the type of failures the different networks produce for interpolations between very different inputs.

By definition, the interpolations obtained by PCA are just linear combinations of the reprojections. All the neural nets seem to somewhat successfully incorporate nonlinearities into their mappings. This can be seen for example in the handle of the bag (second column from the right), which properly changes length instead of duplicating.

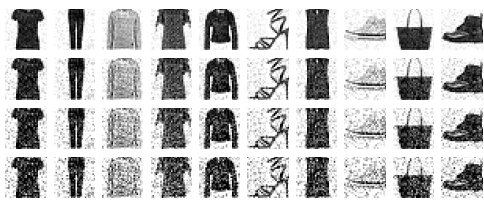
A difference between using the L1 or the L2 metric can be observed in the sandal: Trained with the euclidean distance as loss the decoders tend to blur the image, with the manhattan distance entire lines can be removed or added.

## 4.4. Denoising

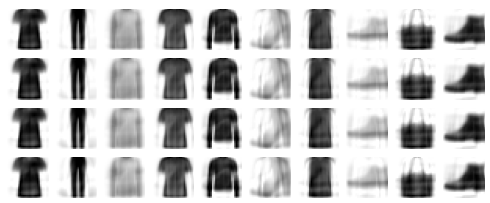
Usually when projecting images to an embedded space and back, the goal is for as much detail as possible to be preserved. The ability of neural nets to generalize to unseen data is, however, limited. This property of the reprojection procedure can be used to remove unwanted artifacts - like noise - from input images.

To test this, we picked one sample per class from the training set, and generated a pattern of standard normal noise for each of them. These noise patterns were then multiplied by a range of scalar factors before being added to their images. By not choosing images from the test set, the neural nets varying ability to reproject unseen data (see Figure A.1) is eliminated as a source of errors in the output.

The results of this batch of experiments can be seen in Figure 4.3. The effective mean and standard variation of the applied noise are  $\mu = 0$  and  $\sigma \in \left\{ \frac{2}{33} \approx 0.06, \frac{4}{33} \approx 0.12, \frac{6}{33} \approx 0.18, \frac{8}{33} \approx 0.24 \right\}$ . The pixel values of the noisy images in (4.3a) are clipped to the  $[0; 1]$ -range, but the neural nets received the unclipped values as input.



(a) noisy input images



(b) PCA



(c) precomputed embedding (euclidean)



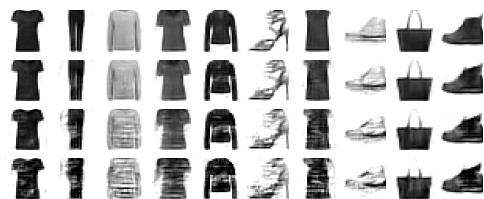
(d) precomputed embedding (manhattan)



(e) Autoencoder (euclidean)



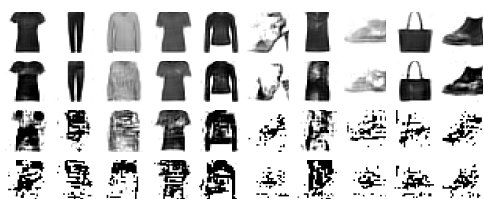
(f) Autoencoder (manhattan)



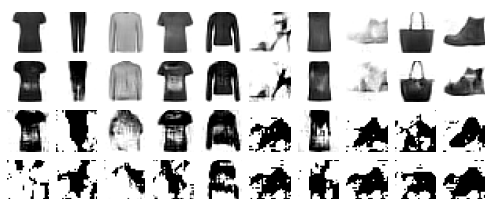
(g) Siamese (euclidean)



(h) Siamese (manhattan)



(i) Triplets (euclidean)



(j) Triplets (manhattan)

Figure 4.3.: Denoised images of the fashion-MNIST training set. (a) shows the input images. The same pattern of gaussian noise is applied in every row, with increasing magnitude.

Figure 4.3b shows the results of using PCA (fitted to the entire training set) to denoise the images. While these reprojections are the least affected by high levels of noise, their quality is noticeably poor even with low levels of noise.

Overall, the networks trained using the manhattan metric appear to perform slightly better on this task than their counterparts trained using L2 distance. However, the quality of the denoised images, as well as the maximum level of noise that can be filtered out, vary much more strongly between the different training configurations. This suggests once more that the L1 and L2 distance are similarly suited to finding embedded spaces and training neural networks, and that the choice between them plays an inferior role to the choice of training configuration.

## 5. Conclusion

Convolutional neural networks are a valuable tool for nonlinear dimensionality reduction of Images. To train them, a distance metric for use in the image-space must first be chosen. Common choices for this are the euclidean and the manhattan distance. In this work, we implemented a variety of training configurations and tested them - using both metrics - on multiple applications of dimensionality reduction.

Although not all training setups led to smooth, fast converging progress during fitting of the models, the key result of the experiments remains valid:

The effect of choosing the L1 or the L2 metric for training has a negligibly small effect on the performance of the network in question.

Going into more detail, the euclidean distance produced slightly higher scores when classifying images. When identifying likely outliers, neither distance metric could be judged to provide an even minimally better suited embedding. During interpolation, it was revealed that L1 and L2 tend to produce different types of errors in output images: Where the decoder networks trained using the euclidean distance blur images, their counterparts trained using the manhattan distance tended to omit or erroneously add features like lines. In the denoising experiments, the outputs of networks trained with L1 - subjectively - appeared to be of higher quality.

Letting the numerically small, but objective difference in classification score outweigh the subjective judgment on the denoised images quality, and considering blurring a more desirable failure mode in image output than the addition or removal of details, the euclidean distance metric can be said to be preferable.

# Appendix



## A. Reprojected Images



(a) original images



(b) PCA



(c) precomputed embedding (euclidean)



(d) precomputed embedding (manhattan)



(e) Autoencoder (euclidean)



(f) Autoencoder (manhattan)



(g) Siamese (euclidean)



(h) Siamese (manhattan)



(i) Triplets (euclidean)



(j) Triplets (manhattan)

Figure A.1.: 20 images fashion-MNIST test set, reprojected using PCA (b) and every trained neural net (c - j). Two samples were randomly chosen for each of the 10 classes and arranged in columns. For reference, the original images are shown in (a).

# Bibliography

- [1] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling*. Springer New York, 2005. doi: 10.1007/0-387-28981-x. URL <https://doi.org/10.1007/0-387-28981-x>.
- [2] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a " siamese " time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.
- [3] François Chollet et al. Keras. <https://keras.io>, 2015.
- [4] Sumit Chopra, Raia Hadsell, Yann LeCun, et al. Learning a similarity metric discriminatively, with application to face verification. In *CVPR (1)*, pages 539–546, 2005.
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [6] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [7] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.
- [8] Md Mosharaf Hossain, Douglas A Talbert, Sheikh K Ghafoor, and Ramakrishnan Kannan. Fawca: a flexible-greedy approach to find well-tuned cnn architecture for image recognition problem. In *Proceedings of the Int'l Conf. Data Science*, pages 214–219, 2018.
- [9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
- [10] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a non-metric hypothesis. *Psychometrika*, 29(1):1–27, 1964.
- [11] Gal Mishne, Uri Shaham, Alexander Cloninger, and Israel Cohen. Diffusion nets. *Applied and Computational Harmonic Analysis*, 47(2):259–285, 2019.
- [12] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi: 10.23915/distill.00003. URL <http://distill.pub/2016/deconv-checkerboard>.

- [13] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572, 1901.
- [14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [15] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.
- [16] Warren S Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.
- [17] Florian Wickelmaier. An introduction to mds. *Sound Quality Research Unit, Aalborg University, Denmark*, 46(5):1–26, 2003.
- [18] Paul Wohlhart and Vincent Lepetit. Learning descriptors for object recognition and 3d pose estimation. *CoRR*, abs/1502.05908, 2015. URL <http://arxiv.org/abs/1502.05908>.
- [19] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.