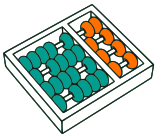


MC934/MO434 – Deep Learning
Institute of Computing – University of Campinas

UNICAMP

Final Project Report

"Constrastive Learning"

Nome	RA	Curso	E-mail
David Aparco Cardenas	230293	Doutorado em Ciência da Computação	d230293@dac.unicamp.br

Submission Date: December 21, 2021

Project Summary

In the present work, we address the problem of face verification using a Siamese neural network (SiNN) architecture. The professor provided a baseline and our objective was to improve it through different steps. First, we evaluated and discussed the role of the hyperparameters of the contrastive loss function, namely contrastive threshold, margin and dimension of the embedding metric space. Next, we investigated the literature, seeking for state-of-the-art techniques to improve the baseline model. We came across three different loss functions, namely a modified version of the contrastive loss function, the triplet loss and the quadruplet loss, which improved the baseline results by either providing a better accuracy or providing a better class separation in the embedding metric space. To assess the impact of the changes for class separation, we used t-SNE, a non-linear projection technique, to project a high dimensional space onto 2D. Furthermore, we used the feature extractors of a VGG-16 and a ResNet-18 network aiming to improve the baseline model. However, we were not successful in obtaining a gain in performance, obtaining only similar or worst results than the baseline model. Lastly, we implemented a transfer learning technique using the Corel dataset to build a classifier model by using the convolutional layers of a SiNN model as feature extractor and replacing the projection head of the SiNN by an MLP classifier. We achieved to obtain good results for all the models we tested.

1 Introduction

Recently, Contrastive Learning has received increased attention by researchers and practitioners due to its success in achieving state-of-the-art performance in self-supervised representation learning, leading to its widespread use in multiple applications and input domains including image, video, text, audio and others, with contributions spanning across many fields including natural language processing, computer vision and audio processing among others [6].

It is of common knowledge that a good choice of data representation, or features, is directly related to the performance of a machine learning model. Furthermore, it is also known that some sets of features are representative of a dataset and are suitable as input to train an effective classifier or predictor.

Therefore, *representation learning* [1], which refers to the process of learning a parametric mapping from the raw input data to a feature vector aiming to capture and extract more abstract and useful high-level concepts, was proposed to avoid the tedious and time-consuming task of feature engineering and improve the performance of downstream tasks. In this context, often the input data is represented in a high-dimensional space (*e.g.* images, video, sound, text), whereas the embedded representations are located in a lower dimension manifold. In contrast to most *dimensionality reduction* techniques that convert high-dimensional inputs to lower-dimensional encodings, representation learning must also learn a mapping that generalizes on unseen data samples.

Contrastive learning can be succinctly described as learning by *comparing* among different samples. Such comparison is carried out between positive pairs of "similar" inputs and negative pairs of "dissimilar" inputs. In contrast to supervised techniques, contrastive learning methods make no use of a class label y for every input sample $\mathbf{x}$. Instead, they only need to define the similarity distribution to sample positive inputs $\mathbf{x}^+ \sim p^+(\cdot|\mathbf{x})$, and a data distribution for negative inputs $\mathbf{x}^- \sim p^-(\cdot|\mathbf{x})$, with respect to an input sample $\mathbf{x}$. The primary objective of contrastive learning is to encode "similar" samples into representations that are close together, whereas "dissimilar" samples should be mapped further apart in the embedding space. Accordingly, the learning process consisting of contrasting between samples of positive and samples of negative pairs will bring closer together representations of positive pairs, while pushing apart representations of negative pairs.

In this context, *siamese networks* stand out as one of the most widespread used contrastive learning-based methods [3]. A siamese network consist of two identical base networks with shared weights to perform a non-linear embedding ϕ from the input domain $\mathscr{X}$ (*e.g.*, images) to some Euclidean space $\mathbb{R}^n$, *i.e.*, $\phi : \mathscr{X} \to \mathbb{R}^n$ [8].

In the present project, we address the problem of face verification using a siamese network architecture. The baseline is provided by the professor and our objective is to improve it by implementing new ideas found in the related work literature and modifying the hyperparameters of the network.

The implementation of the siamese network and the experiments in this project were done using PyTorch [7].

2 Problem Definition and Baseline

The dataset employed in this study is the Olivetti face dataset containing a set of face images taken between April 1992 and April 1994 at AT&T Laboratories Cambridge. It contains ten different images for each of 40 different people, comprising a total of 400 images in .pgm format. The images are quantized to 256 grey levels with dimensions of 92×112 pixels. Figure 1 shows some example images of this dataset.

The architecture of the siamese network used in this project is illustrated in Figure 3. This contrastive learning approach aims to learn a representation by maximizing the agreement of the encoded embeddings between augmented views of two images of the same subject, while simultaneously minimizing the agreement between views generated from different subjects. Image augmentation techniques are

Figure 1. Some images of the Olivetti dataset. Each row contains ten images of the same subject.

applied to the training images generating views, so the model avoids maximizing agreement through low-level visual cues.

Next, we explain the components of the siamese network shown in Figure 3. In this project, new views are generated from the original images using the following image augmentation techniques:

- `transforms.RandomHorizontalFlip()`
- `transforms.RandomAffine(5, (0.01,0.2), scale=(0.9,1.1))`

Let $\mathcal{T}$ be the set of *image transformation* operations listed above. Then, for a pair of images $(\mathbf{x}, \mathbf{y})$, two views are generated by applying the set of transformations on the images: $\mathbf{x}^* = \mathcal{T}(\mathbf{x})$, $\mathbf{y}^* = \mathcal{T}(\mathbf{y})$

(a) **(b)** **(c)** **(d)** **(e)**

Figure 2. Some images of the Olivetti dataset and the views generated using $\mathcal{T}$.

Figure 2 shows some example of the views generated with $\mathcal{T}$. Once the views have been generated, a process of resizing to 100×100 and normalization with mean and standard deviation of 0.5 is performed on both images $\mathbf{x}^*$ and $\mathbf{y}^*$. Then, a *feature encoder* $e(\cdot)$ extract the feature vectors from the augmented images $\mathbf{v}_x = e(\mathbf{x}^*)$ and $\mathbf{v}_y = e(\mathbf{y}^*)$.

The encoder is defined by two convolutional blocks cb_1 and cb_2, such that each convolutional block is constituted by a convolutional layer, batch normalization, ReLU and max-pooling. The number of filters in each convolutional block are $n(cb_1) = 16$ and $n(cb_2) = 128$.

A feature vector $\mathbf{v} \in \mathbb{R}^{128\times25\times25}$ is generated and then fed into a *projection head* $h(\cdot)$ comprised of a multi-layer perceptron (MLP) constituted by three fully connected layers of sizes 512, 256 and 64. The metric embeddings $\mathbf{z}_x = h(\mathbf{v}_x)$ and $\mathbf{z}_y = h(\mathbf{v}_y)$, where $z_* \in \mathbb{R}^{64}$ is obtained for each of the pair of input images.

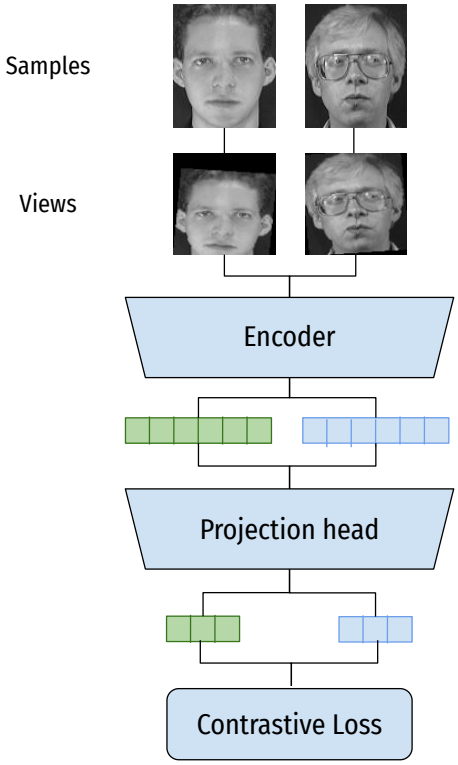


Figure 3. Illustration of the siamese network architecture used in thid work.

A batch of these metric embeddings pairs $\{(\mathbf{z}_x, \mathbf{z}_y)\}$ are then fed into the *contrastive loss* function which encourages the distance of the embeddings of similar subjects to be small, and the distance of the different subjects to be large. The contrastive loss function introduced in [5] is used. Let $\mathbf{z}_x$, $\mathbf{z}_y$ be a pair of metric embeddings fed into the contrastive loss function. Let l be a binary label assigned to this pair, such that $l = 0$ if $\mathbf{z}_x$, $\mathbf{z}_y$ are deemed similar, and $l = 1$ if they are deemed dissimilar. The parameterized distance function D_w between metric embeddings is defined as the Euclidean distance. That is,

$$D_w(\mathbf{z}_x, \mathbf{z}_y) = \|\mathbf{z}_x - \mathbf{z}_y\| \tag{1}$$

The input of the contrastive loss function is the triplet $(\mathbf{z}_x, \mathbf{z}_y, l)$ and the loss function is defined as in Equation 2, where $m > 0$. The margin m defines a radius around $\mathbf{z}_*$. Thus, different subjects contribute to the loss only if their distance is within this radius. The first term of the summation is the partial loss for a pair of similar subjects, while the second term accounts for the partial loss of a pair of dissimilar subjects.

$$L(\mathbf{z}_x, \mathbf{z}_y, l) = (1-l)\frac{1}{2}(D_w(\mathbf{z}_x, \mathbf{z}_y))^2 + (l)\frac{1}{2}\{\max(0, m - D_w(\mathbf{z}_x, \mathbf{z}_y))\}^2 \tag{2}$$

A contrastive threshold τ is defined to assess whether a pair of metric embeddings belong to the same subject. The euclidean distance of the embeddings is computed and compared to τ . If the distance is lower than τ, we say that both embeddings correspond to the same subject. Otherwise, we say they correspond to different subjects.

The baseline hyperparameters for contrastive threshold τ, margin m and dimension of the embedding metric space d are shown in Table 1.

hyperparameter	*value*
τ	1.1
m	2.0
d	64

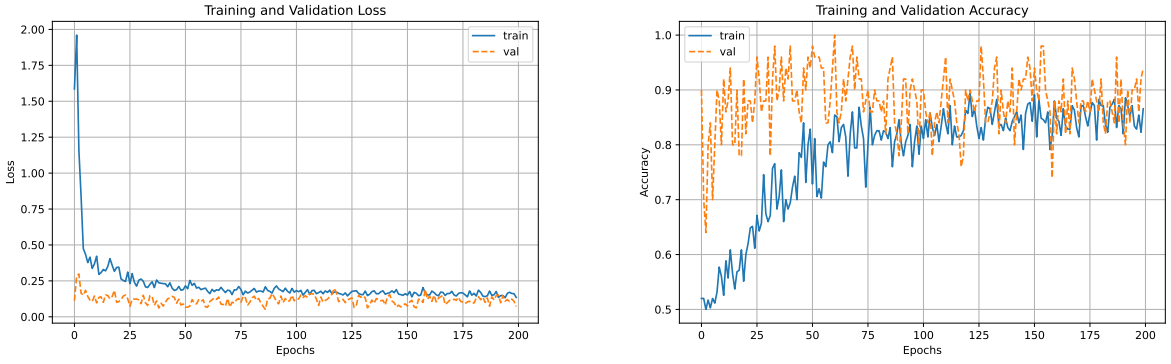
Table 1. Baseline hyperparameters for contrastive threshold τ, margin m and dimension of the embedding metric space d.

The dataset is split into training and validation sets with 35 and 5 subjects, comprising 350 and 50 images, respectively. The siamese neural network (SiNN) with baseline hyperparameters was executed for 200 epochs. Table 2 shows the mean and standard deviation of both training and validation accuracies from 3 executions.

model	*training accuracy*	*validation accuracy*
SiNN (baseline)	0.848571 ± 0.026741	0.908000 ± 0.044900

Table 2. Training and validation accuracy for the baseline SiNN model. The mean and standard deviation are computed from 3 different executions.

Figure 4 shows the loss and accuracy curves on the training and validation sets. It can be observed that high oscillation occurs for the accuracy metric across iterations. This fact can be explained due to the low number of samples used for training the model. Furthermore, it can be seen that there may be happening a case of underfitting since the loss curve of the validation set is lower than the loss curve of the training set. On the other hand, the accuracy curve of the validation set is slightly higher than then accuracy curve of the training set. This behavior may be explained due to the small size of the validation set.



(a) Training and validation loss

(b) Training and validation accuracy

Figure 4. Curves of the accuracy and loss of training and validation datasets for the SiNN (baseline) model.

Figure 5 shows the projection in a bidimensional space of the input MLP, the first hidden MLP layer, the second hidden MLP layer and the output MLP layer using the non-linear projection technique t-SNE [11] algorithm. The input MLP layer is equivalent to the flattening of the output of the convolutional layers. The samples corresponding to the same subject are colored with the same color. The projection of the input MLP layer (see Figure 5.a) shows a separation of the dataset into groups, where each group represent the same subject (same color). However, the output of a CNN is a high-dimensional and sparse feature space, and therefore it is very sensitive to the curse of high dimensionality where

every object has a similar distance to any other object in the metric space. Thus, we lose the ability of distinguishing objects that belong to the same group (subject). To circumvent this issue, we reduce the dimensionality of the output of a CNN using a multilayer perceptron (projection head). It can be seen in Figure 5.d that the projection of the output MLP layer shows the samples of the same group closer than in Figure 5.a. However, the separation of the groups is less clear, with some clusters overlapping.

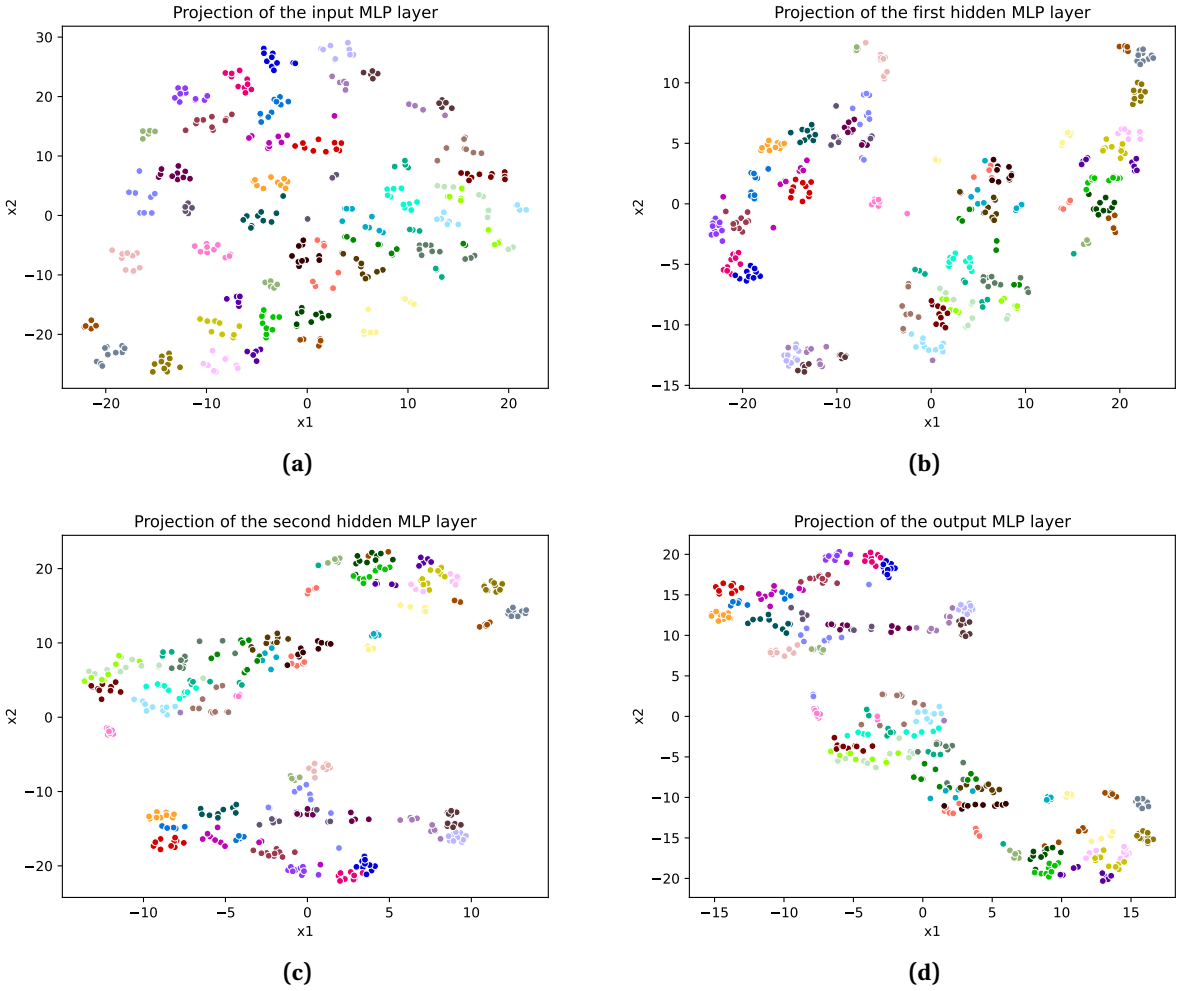


Figure 5. Projections in 2D of (a) the input MLP layer, (b) the first hidden MLP layer, (c) the second hidden MLP layer and (d) the output MLP layer. Samples of the same subject are colored with the same color.

In Section § 3, we evaluate and discuss the role of the following hyperparameters: contrastive threshold τ, margin m and dimension of the embedding metric space d.

3 Evaluation of τ, m and d

In this section, we present a brief study of the τ, m and d hyperparameters, discussing their role in the contrastive loss function and making experiments with different values to analyze their influence on the model's accuracy. The values corresponding to each metric are modified while keeping the same baseline values for the other hyperparameters.

3.1 Contrastive threshold τ

The τ hyperparameter defines the threshold from which it can be decided whether a pair of metric embeddings correspond to the same subject through their Euclidean distance. If the distance is lower than the threshold, we say that they are close in the embedded metric space, and therefore represent the same subject. Otherwise, we say they represent different subjects. We tested five different values of τ, centred around the baseline $\tau = 1.10$: 0.60, 0.85, 1.10, 1.35 and 1.60.

model	*training accuracy*	*validation accuracy*
$\text{SiNN}_{\tau=0.60}$	0.655428 ± 0.053714	0.731999 ± 0.054552
$\text{SiNN}_{\tau=0.85}$	0.767428 ± 0.023532	0.847999 ± 0.054552
$\text{SiNN}_{\tau=1.10}$	0.848571 ± 0.026741	0.908000 ± 0.044900
$\text{SiNN}_{\tau=1.35}$	0.834857 ± 0.020396	0.847999 ± 0.049959
$\text{SiNN}_{\tau=1.60}$	0.825714 ± 0.031350	0.851999 ± 0.057410

Table 3. Training and validation accuracies of the baseline SiNN model with different values of contrastive threshold τ. The table shows the mean and standard deviation from 3 executions.

Table 3 shows the results of the experiments. It can be seen that for lower values of τ, the performance of the model decreases significantly. The same behaviour occurs when the contrastive threshold is higher than the baseline τ value. This fact can be explained by the following: if the contrastive threshold is too low, the model may misclassify a pair of similar metric embeddings as dissimilar, and in the other way around, if the threshold is too high, the model may misclassify a pair of dissimilar representations as similar. In both cases, the learning process is negatively affected by this issue. Therefore, we can conclude that $\tau = 1.10$ is a suitable contrastive threshold value for our SiNN model.

3.2 Margin m

The m hyperparameter, as we discussed before, takes an important role in the partial loss function (see Equation 3) for a pair of dissimilar points. The margin m can be seen as a radius defined around the metric embeddings (see Figure 6), such that dissimilar points contribute to the loss function L only if D_w is within this radius.

$$L_D(\mathbf{z}_x, \mathbf{z}_y, l) = (l)\frac{1}{2}\{\max(0, m - D_w(\mathbf{z}_x, \mathbf{z}_y))\}^2 \tag{3}$$

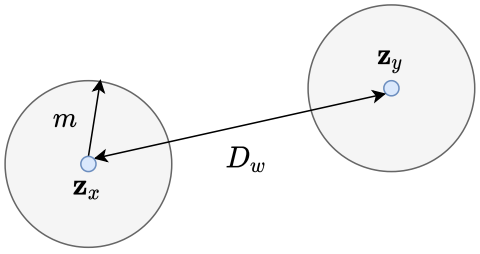


Figure 6. The margin m defines a radius around a metric embedding.

We tested five different values of m, centred around the baseline $m = 2.0$: 1.0, 1.5, 2.0, 2.5 and 3.0. Table 4 shows the results of the experiments. It can be observed that for lower values of m, the performance of the model decreases (*e.g.*, $m = 1.0$) and a similar behavior is found for higher values of m. For high values of m, the loss will be higher since more pair of points will contribute to the loss function,

which may need a larger number of epochs or a larger learning rate. In contrast, for low values of m, less pair of points will contribute to the loss function which will mislead the learning process. Therefore, we can conclude that $m = 1.5$ and $m = 2.0$ are suitable margin values for our SiNN model.

model	*training accuracy*	*validation accuracy*
$\text{SiNN}_{m=1.0}$	0.631999 ± 0.024538	0.787999 ± 0.073321
$\text{SiNN}_{m=1.5}$	0.817142 ± 0.021983	0.891999 ± 0.065238
$\text{SiNN}_{m=2.0}$	0.848571 ± 0.026741	0.908000 ± 0.044900
$\text{SiNN}_{m=2.5}$	0.789142 ± 0.048027	0.895999 ± 0.052763
$\text{SiNN}_{m=3.0}$	0.756571 ± 0.023726	0.835999 ± 0.063749

Table 4. Training and validation accuracies of the baseline SiNN model with different values of margin m. The table shows the mean and standard deviation from 3 executions.

3.3 Dimension of the embedding metric space d

The d hyperparameter represents the dimension of the MLP output layer or, in other words, the dimension of the embedded metric space. Since the objective of the MLP (projection head) in the SiNN is that of dimensionality reduction, it must be considerably lower than the dimension of the flattened output convolutional layer. We tested five different values of d, centred around the baseline $d = 64$: 16, 32, 64, 96 and 128.

model	*training accuracy*	*validation accuracy*
$\text{SiNN}_{d=16}$	0.848571 ± 0.031558	0.891999 ± 0.029933
$\text{SiNN}_{d=32}$	0.846285 ± 0.036203	0.843999 ± 0.026533
$\text{SiNN}_{d=64}$	0.848571 ± 0.026741	0.908000 ± 0.044900
$\text{SiNN}_{d=96}$	0.836571 ± 0.025834	0.895999 ± 0.032000
$\text{SiNN}_{d=128}$	0.875428 ± 0.013229	0.879999 ± 0.047328

Table 5. Training and validation accuracies of the baseline SiNN model with different values of dimension of the embedding metric space d. The table shows the mean and standard deviation from 3 executions.

The results of this experiment are shown in Table 5. It can be observed that the model achieves a similar performance for all five values of d, which suggests that any of those values for d can be used to generate the embedding metric space.

In Section § 4, we make a literature review aiming to improve the baseline by modifying the components of the SiNN architecture.

4 Improving the baseline

4.1 Contrastive Loss L with Cosine Similarity

In [10], Suprapto and Polela used the contrastive loss function L with a cosine similarity metric instead of Euclidean distance for measuring text similarity. The cosine similarity is defined in Equation 4 for a pair of metric embeddings $\mathbf{z}_x$ and $\mathbf{z}_y$. Thus, we find interesting to test whether the cosine similarity distance may be suitable in the context of images. Therefore, we plug this distance in the contrastive loss function L replacing the standard Euclidean distance.

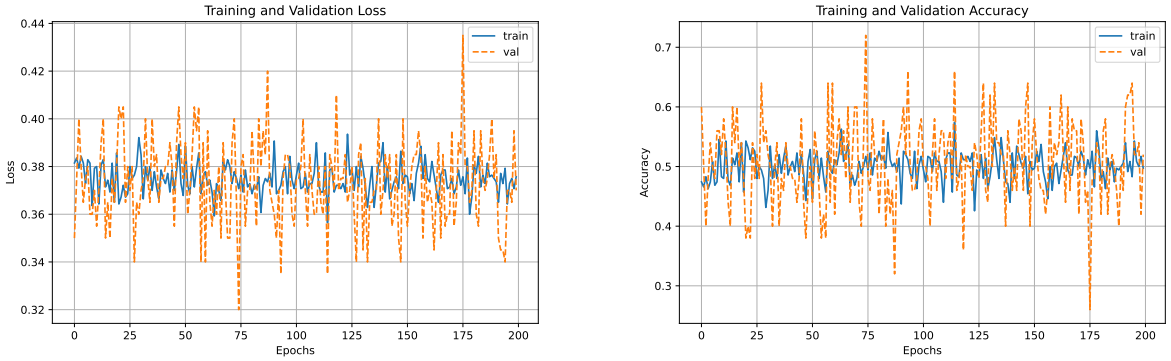
$$D_w(\mathbf{z}_x, \mathbf{z}_y) = \frac{\mathbf{z}_x \cdot \mathbf{z}_y}{\|\mathbf{z}_x\| \|\mathbf{z}_y\|} \tag{4}$$

We executed the experiment using the same hyperparameters used for the baseline. Table 6 shows the mean and standard deviation of the model's accuracy from three different executions. It can be observed that the model's performance declines significantly, suggesting that the cosine similarity is not suitable for this particular problem.

model	***training accuracy***	***validation accuracy***
SiNN (baseline)	0.848571 ± 0.026741	0.908000 ± 0.044900
SiNN_{cos}	0.496190 ± 0.0221721	0.479999 ± 0.056568

Table 6. Training and validation accuracy for the SiNN model using cosine similarity. The mean and standard deviation are computed from 3 different executions.

Figure 7 shows the learning curves for the loss and accuracy for both training and validation sets. It can be seen that the cosine similarity does not reflect the actual distance of a pair of points in the embedding metric space, negatively affecting the minimization of the loss function, thereby thwarting the learning process.



(a) Training and validation loss

(b) Training and validation accuracy

Figure 7. Curves of the accuracy and loss of training and validation datasets for the SiNN model using cosine similarity.

4.2 Contrastive Loss L^*

Chopra et al. introduced a contrastive loss function L^* in [4] for face verification. The loss function L^* is defined in Equation 5. It can be observed that this contrastive loss function receives the same set of parameters as the baseline contrastive loss function. However, it introduces another hyperparameter replacing the margin m. The constant Q is set to the upper bound of $D_w(\mathbf{z}_x, \mathbf{z}_y)$. In the same fashion as in the baseline loss function, the first term of the summation accounts for the partial loss function for a similar pair of metric embeddings, while the second term accounts for the partial loss function for a dissimilar pair of metric embeddings. Both terms were designed in a way such that the minimization of L^* will decrease the energy of the first term and increase the energy of the second term.

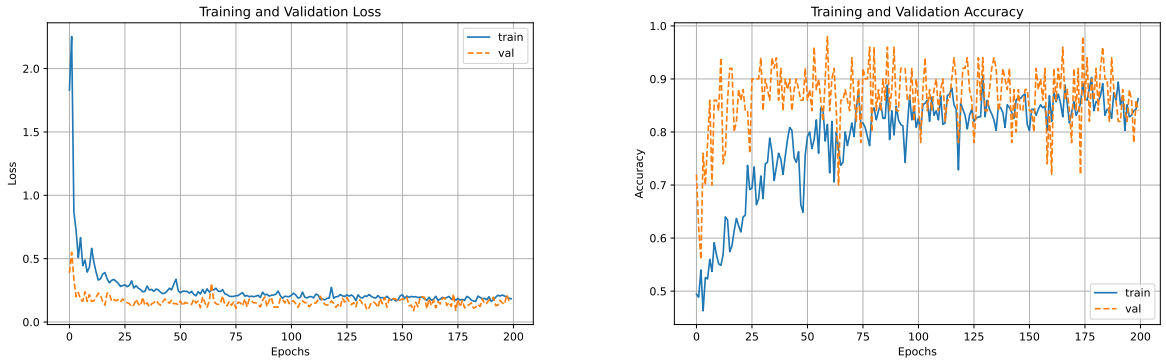
$$L^*(\mathbf{z}_x, \mathbf{z}_y, l) = (1-l)\frac{2}{Q}(D_w(\mathbf{z}_x, \mathbf{z}_y))^2 + (l)2Qe^{-\frac{-2.77}{Q}D_w(\mathbf{z}_x, \mathbf{z}_y)} \tag{5}$$

To evaluate the influence of the hyperparameter Q on the effectiveness of the model, we tested three different values of Q: 1.5, 2.0 and 2.5. Table 7 shows the results of this experiment, where it can be observed that $Q = 2.0$ is the value that achieves the best accuracy for both the training and validation sets.

model	***training accuracy***	***validation accuracy***
$\text{SiNN}_{L^*,Q=1.5}$	0.861904 ± 0.017817	0.879999 ± 0.043204
$\text{SiNN}_{L^*,Q=2.0}$	0.871428 ± 0.016329	0.913333 ± 0.024944
$\text{SiNN}_{L^*,Q=2.5}$	0.816190 ± 0.042335	0.853333 ± 0.052493

Table 7. Training and validation accuracies of the SiNN model using L^* as contrastive loss function and different values of Q. The table shows the mean and standard deviation from 3 executions.

Table 8 shows the loss and accuracy curves for the SiNN model with L^*, where $Q = 2.0$. It can be seen that the model successfully achieves to minimize the loss and maximize the accuracy along iterations.



(a) Training and validation loss

(b) Training and validation accuracy

Figure 8. Curves of the accuracy and loss of training and validation datasets for the SiNN model with L^*.

Figure 9 shows the projection in 2D using t-SNE of the MLP layers. It can be seen that L^* attains a slightly better performance in separating points of different classes at the input MLP layer. It can also be seen that in the next layers, points of different classes start exhibiting some overlapping.

4.3 Triplet Loss

The Triplet loss [9] takes three inputs data to compare at each time step, in contrast to the standard contrastive loss which takes only two. The first sampled object $\mathbf{x}$ is called *anchor*, which is used as point of comparison for the remaining two objects. Next, we sample a *positive* object $\mathbf{y}$, which is known to be similar or share the same label with the anchor object. Lastly, we sample a *negative* object $\mathbf{w}$, which is known to be dissimilar or have a different label than the anchor object. In this context, our SiNN will consist of three twin networks with tied weights, where each of the networks will generate a metric embedding for each object. Let $\mathbf{z}_x$, $\mathbf{z}_y$ and $\mathbf{z}_w$ be the metric embeddings generated from each of the networks for $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{w}$, respectively.

Let d_1 be the Euclidean distance between $\mathbf{z}_x$ and $\mathbf{z}_y$, *i.e.*, $d_1 = \|\mathbf{z}_x - \mathbf{z}_y\|$, and let d_2 be the Euclidean distance between $\mathbf{z}_x$ and $\mathbf{z}_w$, *i.e.*, $d_2 = \|\mathbf{z}_x - \mathbf{z}_w\|$. Therefore, the Triplet loss function (TL) is defined as in Equation 6.

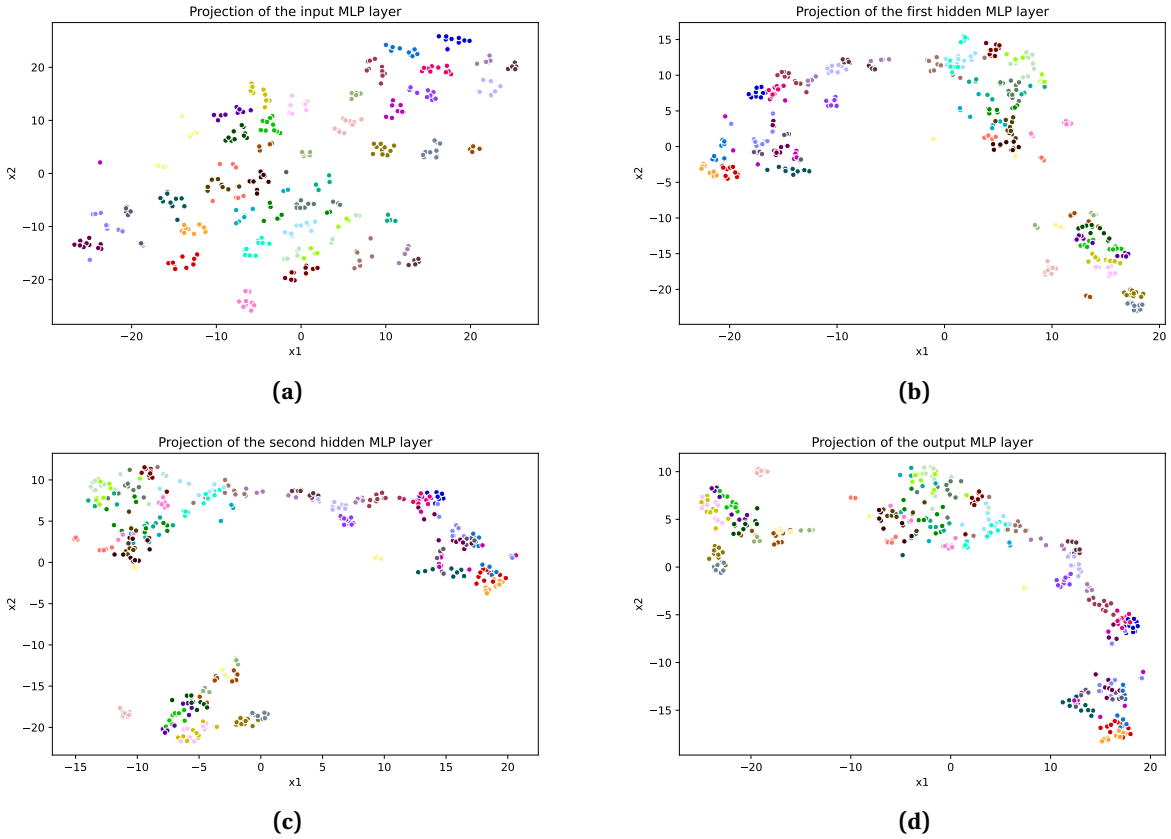


Figure 9. Projections in 2D of (a) the input MLP layer, (b) the first hidden MLP layer, (c) the second hidden MLP layer and (d) the output MLP layer for the SiNN model with L^*. Samples of the same subject are colored with the same color.

$$TL(d_1, d_2) = \max(d_1^2 - d_2^2 + m, 0) \tag{6}$$

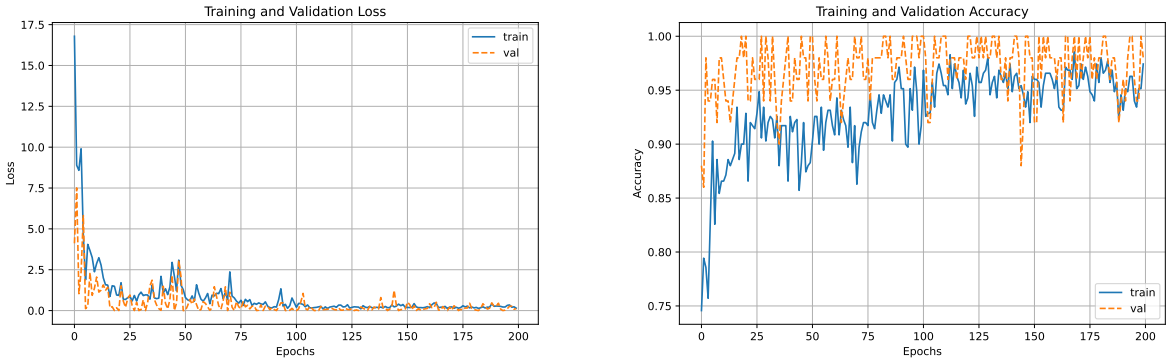
where m is the margin parameter. At each learning iteration, the anchor $\mathbf{z}_x$ is pushed closer to the similar object $\mathbf{z}_y$, while being pushed away from the dissimilar object $\mathbf{w}$ until the distance between them surpasses the value of the margin m. Therefore, aiming to minimize the loss, the model adopt two strategies: minimize d_1^2, and maximize d_2^2 until it surpasses the value of m. The accuracy is computed comparing the distances d_1 and d_2, thereby, we say that a triplet of metric embeddings is accurate if $d_1 < d_2$.

model	***training accuracy***	***validation accuracy***
$\text{SiNN}_{TL,m=2.0}$	0.951428 ± 0.004040	0.986666 ± 0.018856

Table 8. Training and validation accuracy for the baseline SiNN model with Triplet loss. The mean and standard deviation are computed from 3 different executions.

We empirically set the value of $m = 2.0$ and execute the model 3 times to compute the mean and standard deviation of the accuracy for both the training and validation sets. Table 8 shows the values of the training and validation accuracy for this model. The learning curves for the loss and accuracy are presented in Figure 10, where it can be seen that the learning process achieves its objective of minimizing the loss and maximizing the accuracy. Figure 11 shows the projections in 2D of the layers of the

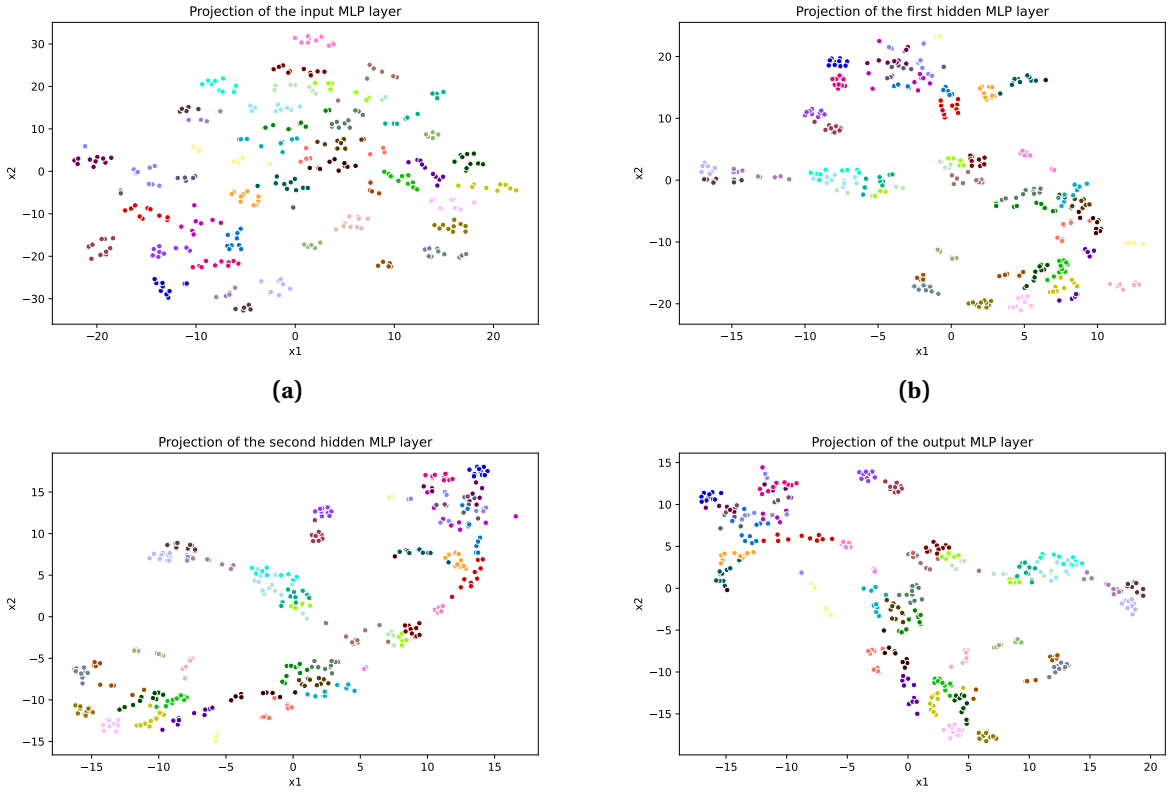
MLP. It can be seen that in the input MLP layer, there exists a clear separation between classes, such that similar points are close, and dissimilar points are apart from each other. In subsequent layers, this separation is maintained with the difference that clusters of similar points are becoming more compact.



(a) Training and validation loss

(b) Training and validation accuracy

Figure 10. Curves of the accuracy and loss of training and validation datasets for the SiNN model with Triplet loss.



(a)

(b)

(c)

(d)

Figure 11. Projections in 2D of (a) the input MLP layer, (b) the first hidden MLP layer, (c) the second hidden MLP layer and (d) the output MLP layer for the SiNN model with Triplet loss. Samples of the same subject are colored with the same color.

4.4 Quadruplet Loss

The Quadruplet loss [2] takes four inputs data at each time step. Analogously to the Triplet loss, we need to sample an *anchor* $\mathbf{x}$, a positive object $\mathbf{y}$ and a negative object $\mathbf{w}$. Additionally, we will need to sample a fourth object $\mathbf{v}$, this will be another negative object dissimilar to the previous three ones. Our SiNN will consist of four identical networks with tied weights, where each of the network will generate a metric embedding for each object. Let $\mathbf{z}_x$, $\mathbf{z}_y$, $\mathbf{z}_w$ and $\mathbf{z}_v$ be the metric embeddings generated from each of the networks for $\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$ and $\mathbf{v}$, respectively.

Let d_1 be the Euclidean distance between $\mathbf{z}_x$ and $\mathbf{z}_y$, *i.e.*, $d_1 = \|\mathbf{z}_x - \mathbf{z}_y\|$, let d_2 be the Euclidean distance of $\mathbf{z}_x$ and $\mathbf{z}_w$, *i.e.*, $d_2 = \|\mathbf{z}_x - \mathbf{z}_w\|$, and let d_3 be the Euclidean distance of $\mathbf{z}_w$ and $\mathbf{z}_v$. Therefore, the Quadruplet loss function (QL) is defined as in Equation 7.

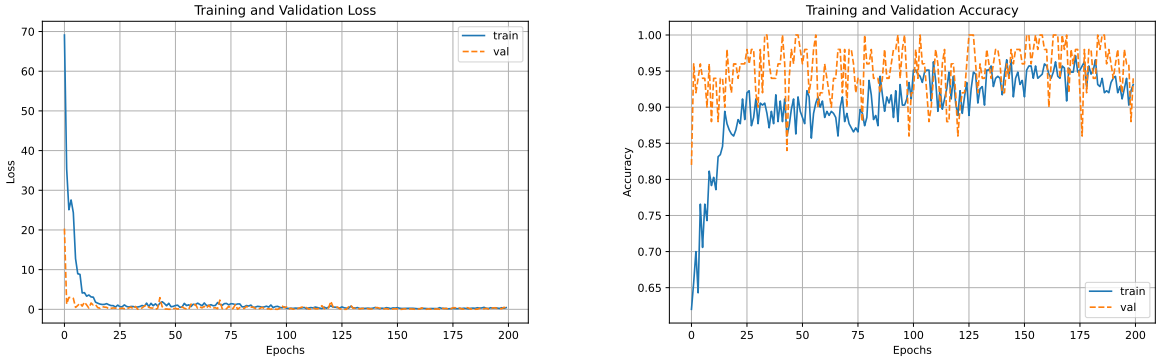
$$QL(d_1, d_2, d_3) = \max(d_1^2 - d_2^2 + m_1, 0) + \max(d_1^2 - d_3^2 + m_2, 0) \tag{7}$$

where m_1 and m_2 are margin parameters. At each learning iteration, the similar objects $\mathbf{z}_x$ and $\mathbf{z}_y$ are pushed together, while the dissimilar objects $\mathbf{z}_x$, $\mathbf{z}_w$ are pushed apart from each other. Moreover, an additional dissimilar object $\mathbf{z}_v$ is included in the loss. This helps the network to better understand the notion of dissimilarity, such that the two dissimilar objects $\mathbf{z}_w$ and $\mathbf{z}_v$ are also pushed apart from each other. Therefore, aiming to minimize the loss, the model adopt three strategies: minimize d_1^2, maximize d_2^2 until it surpasses the value of m_1 and maximize d_3^2 until it surpasses the value of m_2. The accuracy is computed comparing the distances d_1 and d_2 and d_1 and d_3, thereby, we say that a triplet of metric embeddings is accurate if $d_1 < d_2 \wedge d_1 < d_3$.

model	***training accuracy***	***validation accuracy***
$\text{SiNN}_{QL,m_1=2.0,m_2=1.0}$	0.935238 ± 0.019564	0.959999 ± 0.016329

Table 9. Training and validation accuracy for the SiNN model with Quadruplet Loss. The mean and standard deviation are computed from 3 different executions.

We empirically set the values of $m_1 = 2.0$, $m_2 = 1.0$ and execute the model 3 times to compute the mean and standard deviation of the accuracy for both the training and validation sets. Table 9 shows the values of the training and validation accuracy for this model. The learning curves for the loss and accuracy are presented in Figure 12, where it can be seen that the model successfully learns to minimize the loss while maximizing the accuracy.



(a) Training and validation loss

(b) Training and validation accuracy

Figure 12. Curves of the accuracy and loss of training and validation datasets for the SiNN model with Quadruplet Loss.

Figure 11 shows the projections in 2D of the layers of the MLP of the SiNN. It can be seen that in the input MLP layer, there exists a clear separation between classes, such that similar points are closer, and dissimilar points are apart from each other. In subsequent layers, this separation is maintained with the difference that clusters of similar points are becoming more compact. It is worth noting that the performance of the Quadruplet loss is the best among its counterparts, achieving more compact clusters in the projection of the output MLP layer.

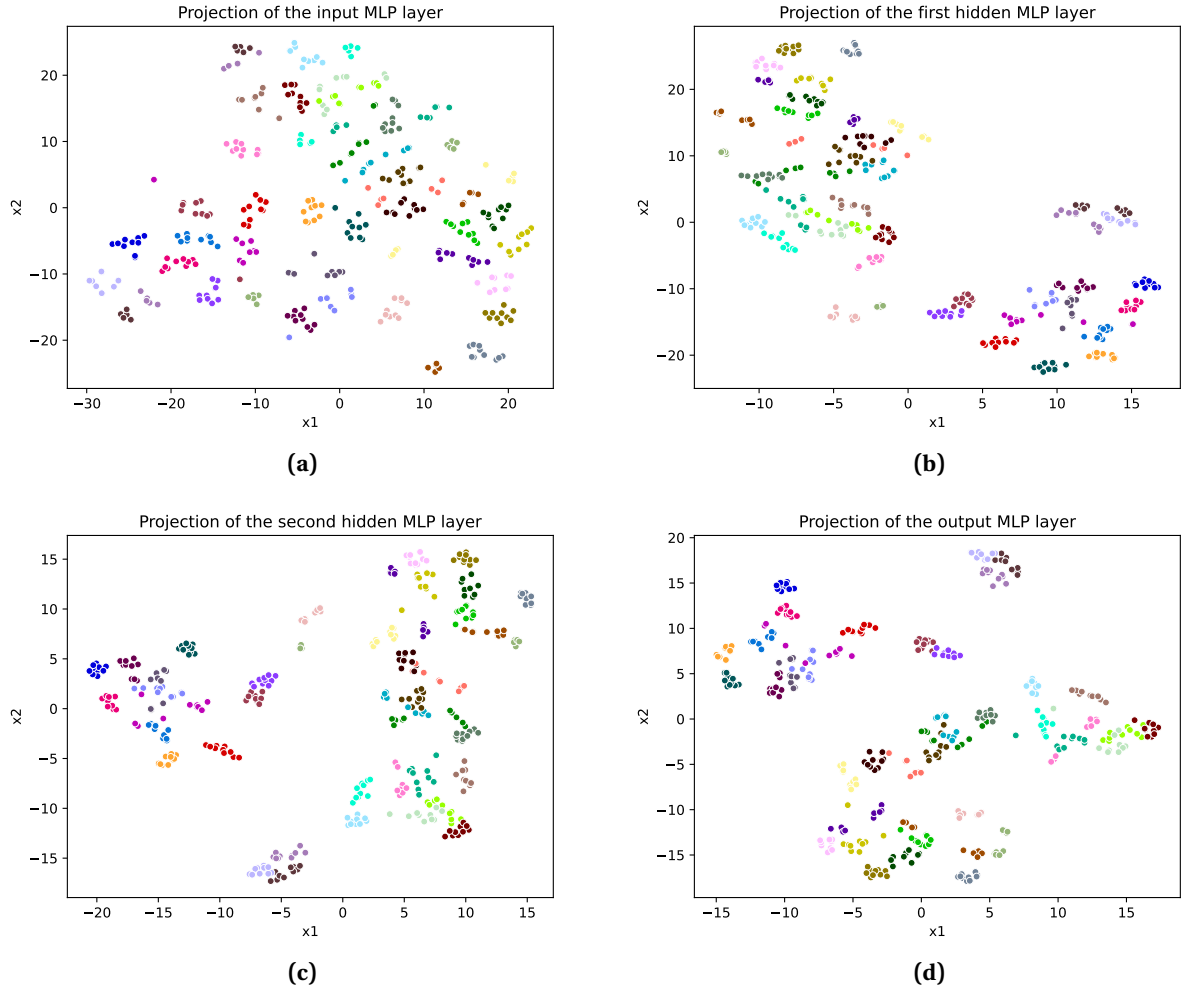


Figure 13. Projections in 2D of (a) the input MLP layer, (b) the first hidden MLP layer, (c) the second hidden MLP layer and (d) the output MLP layer for the SiNN model with Quadruplet Loss. Samples of the same subject are colored with the same color.

5 SiNN with VGG-16 and ResNet-18

In this section, we will make some experiments by replacing the convolutional layers of the SiNN network with a backbone comprised of the feature extractors of VGG-16 and ResNet-18 aiming to improve the performance of the model by leveraging the pretrained convolutional layers of such networks. We obtained the VGG-16 and ResNet-18 networks with pretrained weights from the torchvision library.

In this experiment, we considered four models to compare against the baseline:

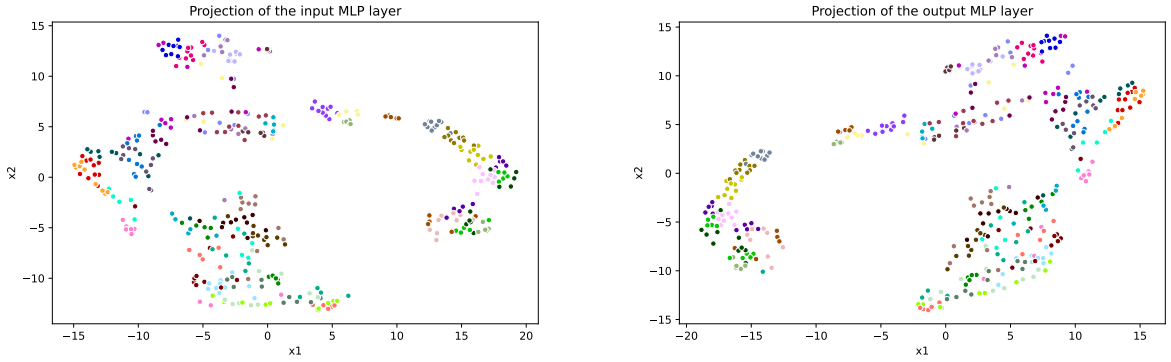
1. SiNN training the backbone of VGG-16 (SiNN_{VGG-16})
2. SiNN freezing the backbone of VGG-16 (SiNN^{f}_{VGG-16})
3. SiNN training the backbone of ResNet-18 ($\text{SiNN}_{ResNet-18}$)
4. SiNN using the frozen backbone of ResNet-18 ($\text{SiNN}^{f}_{ResNet-18}$)

The main problem we faced during this experiment was that the input of the pretrained networks expected an image with 3 channels. However, we are working with grayscale images with only a single channel. Thus, to address this problem, we averaged the channels of the kernels in the input layer into a single channel.

model	***training accuracy***	***validation accuracy***
SiNN (baseline)	0.848571 ± 0.026741	0.908000 ± 0.044900
SiNN_{VGG-16}	0.789523 ± 0.045014	0.799999 ± 0.000000
SiNN^{f}_{VGG-16}	0.839047 ± 0.020381	0.833333 ± 0.041096
$\text{SiNN}_{ResNet-18}$	0.812380 ± 0.030446	0.886666 ± 0.057348
$\text{SiNN}^{f}_{ResNet-18}$	0.864761 ± 0.015879	0.693333 ± 0.049888

Table 10. Training and validation accuracies of the SiNN model using the backbones of VGG-16 and ResNet-18. The table shows the mean and standard deviation from 3 executions.

Table 10 shows the mean and standard deviation of the accuracies from 3 iterations for both the training and validation sets. It can be observed that among the four models, the model that achieves the better result is $\text{SiNN}_{ResNet-18}$ with trainable parameters. However, none of the models is able to obtain a better performance in terms of accuracy than the baseline.



(a) **(b)**

Figure 14. Projections in 2D of (a) the input MLP layer and (b) the output MLP layer for the $\text{SiNN}_{ResNet-18}$ model. Samples of the same subject are colored with the same color.

Figure 14 shows the projections of the input MLP layer and the output MLP layer for the $\text{SiNN}_{ResNet-18}$ model. It can be observed that even though there is indeed some class separation in the projection of the input MLP layer, there are regions with high class overlapping. Besides, we see that the same occurs in the projection of the output MLP layer with regions of high overlapping as well.

6 Transfer Learning for Image Classification

In this section, our objective is to create a predictive model using the convolutional layers of a SiNN as backbone. For this purpose, we will use the Corel dataset, which comprises 355 images in 6 classes. Figure 15 shows some images in the Corel dataset.

Figure 15. Some images of the Corel dataset. Each column represents a different class.

We can broadly divide this experiment into two parts:

1. First, we train a SiNN model using the Corel dataset
2. Next, we replace the projection head of the SiNN model with an MLP classifier, and then we retrain the model freezing the backbone weights.

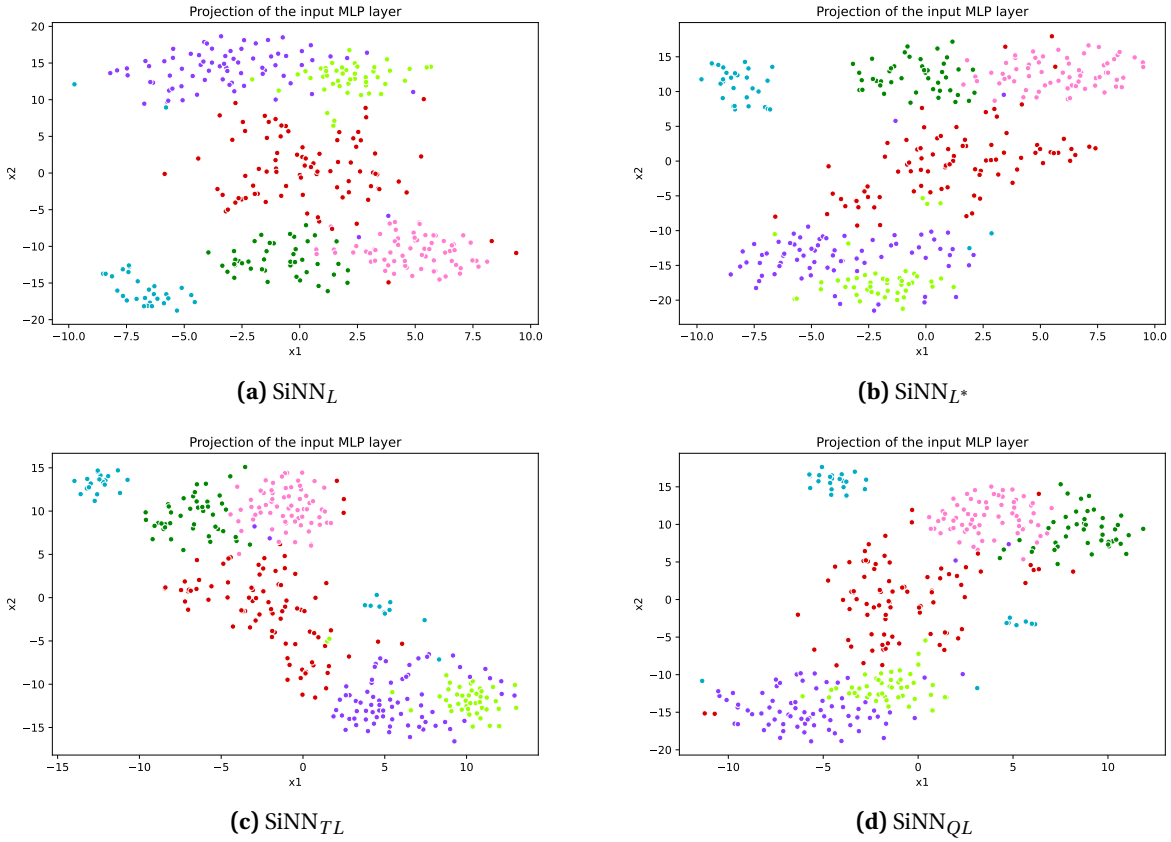
For the first part, we tested four different SiNN models: SiNN with L loss (SiNN$_L$), SiNN with L^* loss (SiNN$_{L^*}$), SiNN with Triplet loss (SiNN$_{TL}$) and SiNN with Quadruplet loss (SiNN$_{QL}$).

For the first part of the experiment, we split the dataset into training (80%) and validation (20%) sets. The same architecture and hyperparameters as in previous experiments were used for all four models. Figure 16 shows the projections of the first input MLP layer, which is equivalent to the output convolutional layer, for all four models. Therefore, these points represent the features the MLP classifier will receive as input during the second part of the experiment. From the figure, it can be observed that all four models achieve to obtain a good separation among the classes.

For the second part of the experiment we partition the dataset into training (40%), validation (20%) and test sets (40%). The MLP used for classification is constituted by a hidden layer with 128 units and the decision layer with 6 units. We only used 40 epochs for the training of the classifier, in contrast to the 200 epochs used for the training of the SiNN models.

Table 11 shows the mean and standard deviation of the accuracy for the training, validation and test sets from 3 executions. It can be observed that the SiNN$_{L^*}$ model achieves the best performance on this task. It is worth noting that actually all the models achieve a good performance. However, there is a slight difference for which the model using the L^* loss becomes the winner.

Figure 17 shows the projection of the decision layer of the MLP classifier for all four models. It can be seen that all models achieve a good separation of the classes. However, the model SiNN$_{L^*}$ attains the



(a) SiNN$_L$ **(b)** SiNN$_{L^*}$

(c) SiNN$_{TL}$ **(d)** SiNN$_{QL}$

Figure 16. Projections in 2D of the input MLP layer for (a) SiNN$_L$, (b) SiNN$_{L^*}$, (c) SiNN$_{TL}$ and (d) SiNN$_{QL}$. Samples of the same class are colored with the same color.

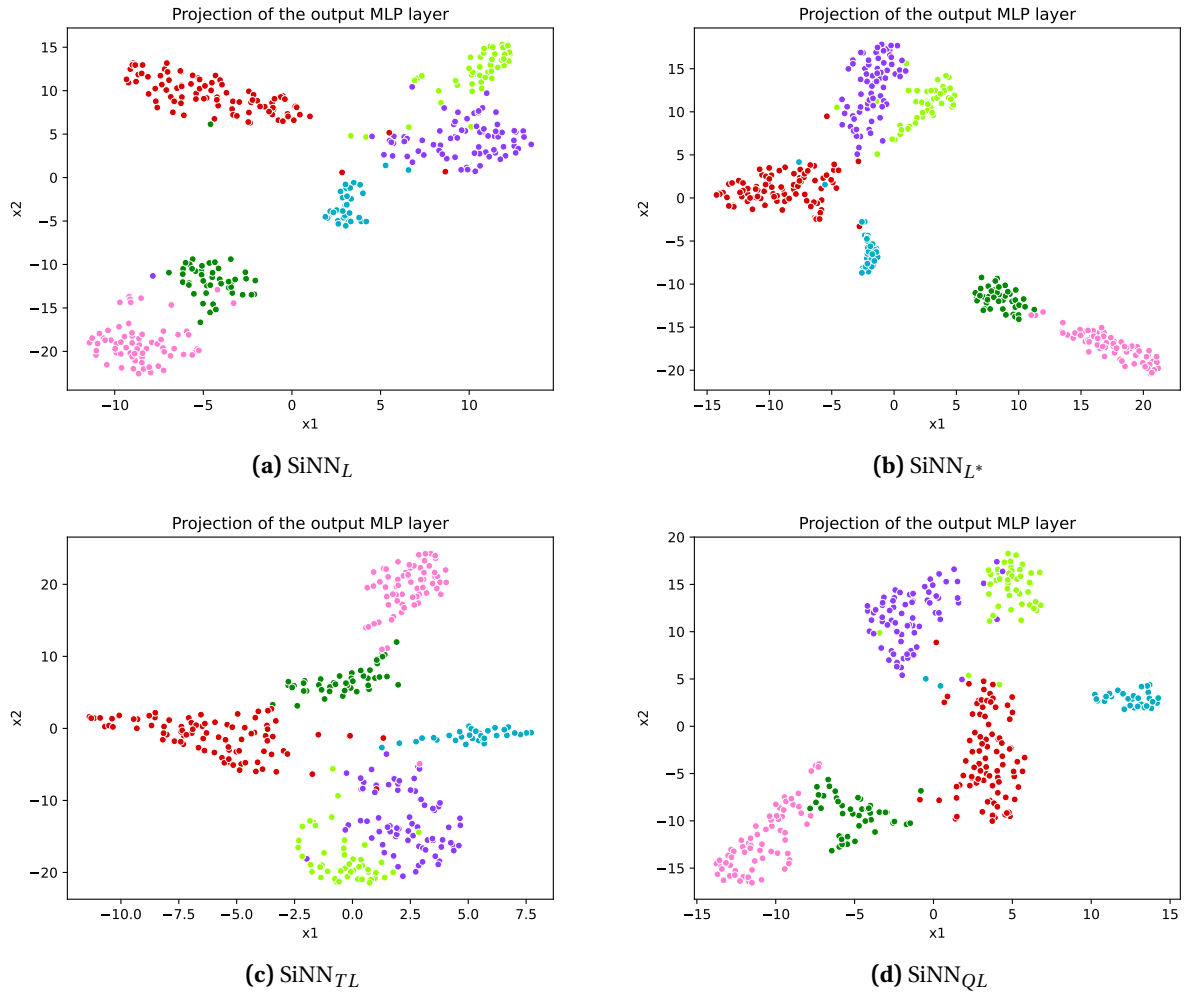
model	***training accuracy***	***validation accuracy***	***testing accuracy***
SiNN$_L$	0.976525 ± 0.011969	0.929577 ± 0.011499	0.924882 ± 0.023939
SiNN$_{L^*}$	0.988262 ± 0.008783	0.943661 ± 0.019918	0.924882 ± 0.013278
SiNN$_{TL}$	0.985915 ± 0.005749	0.943661 ± 0.000000	0.920187 ± 0.013279
SiNN$_{QL}$	0.976525 ± 0.011969	0.938967 ± 0.006639	0.934272 ± 0.017566

Table 11. Training and validation accuracies of the classifier using SiNN$_L$, SiNN$_{L^*}$, SiNN$_{TL}$ and SiNN$_{QL}$ as feature extractors. The table shows the mean and standard deviation from 3 executions.

best separation among classes coinciding with its accuracy values in Table 11. In the other cases, there is some overlapping that negatively influence the performance of the models.

7 Conclusion

In the present project, we implemented a solution for the task of face verification based on a Siamese neural network architecture. Based on a baseline model, our objective was to improve its performance in the aforementioned task. We carried out a discussion and evaluation of the hyperparameters of the contrastive loss. Furthermore, we investigated the literature and implemented three different loss func-



(a) SiNN$_L$ **(b)** SiNN$_{L^*}$

(c) SiNN$_{TL}$ **(d)** SiNN$_{QL}$

Figure 17. Projections in 2D of the output of the MLP classifier for (a) SiNN$_L$, (b) SiNN$_{L^*}$, (c) SiNN$_{TL}$ and (d) SiNN$_{QL}$. Samples of the same class are colored with the same color.

tions, namely, a modifier version of the contrastive loss, the triplet loss and the quadruplet loss. The inclusion of these losses in the training of the SiNN model improved the baseline, achieving a better class separation in the projection of the embedding metric space. We used t-SNE as a non-linear projection tool to project the activations of the neural network layers. We also experimented by using the backbone as feature extractor of a pretrained VGG-16 and ResNet-18 networks, aiming to improve the baseline. However, we did not obtain any significant improvement, achieving only similar or worst results. Lastly, we perform transfer learning using the Corel dataset. We trained four different SiNN models using different losses, then we replaced the projection head from all four models by an MLP classifier with a single hidden layer and then, we proceeded to retrain the models freezing the weights of the feature extractor. We achieved good accuracy results for all four models, showcasing the effectiveness of the transfer learning technique.

This course has broadened my perspective and knowledge in deep learning, giving me the tools and understanding to implement my own models and address a wide variety of problems using these powerful tools.

References

[1] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[2] W. Chen, X. Chen, J. Zhang, and K. Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 403–412, 2017.

[3] D. Chicco. Siamese neural networks: An overview. *Artificial Neural Networks*, pages 73–94, 2021.

[4] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[5] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[6] P. H. Le-Khac, G. Healy, and A. F. Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 2020.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[8] S. K. Roy, M. Harandi, R. Nock, and R. Hartley. Siamese networks: The tale of two manifolds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3046–3055, 2019.

[9] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[10] Suprapto and J. A. Polela. The influence of loss function usage at siamese network in measuring text similarity. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 11(12), 2020.

[11] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.