Data Science Project Two Report

Hang Zheng 520021911347

April 28, 2023

Contents

1 Introduction

In this project, I mainly uses KNN(k-Nearest Neighbor) to do image classification based on different distance measurement methods, including basic distance such as Euclidean Distance, Manhattan Distance and Chebyshev Distance, and some relatively more complicated metric learning methods.

The dataset for the project is the Animals with Attributes (AwA2) dataset, which consists of 37322 images of 2048 dim pre-extracted deep learning features belonging to 50 categories. For each category, I randomly split 60% images into the training set and 40% into the test set.

Furthermore, I do experiment on the parameters of each method trying to find out the optimal distance measurement methods and the best classification accuracy.

2 Experimental Principle

In this section, I will give a brief introduction to the principle of the distance measurement methods I mentioned above.

2.1 K-Nearest Neighbors(KNN)

KNN is a basic classification and regression algorithm. Its basic idea is to find the k nearest training samples to a test sample in the feature space, and then predict the class of the test sample based on the class information of these k training samples.

The key factors in KNN are the the distance measurement method and the appropriate k value. There are many basic distance measurement methods like Euclidean distance as well as some more well-designed metric learning methods and the choice of distance measurement will significantly affect the performance of KNN. As for the k value, it is usually determined by methods such as cross-validation.

The specific steps of the KNN algorithm are as follows:

1. Calculate the distance between the test sample and each training sample
2. Select the k nearest training samples according to the distance
3. Predict the class of the test sample (classification problem) or the value (regression problem) based on the class information of these k training samples

2.2 Distance Metrics

2.2.1 Minkowski Distance

Minkowski Distance is a metric in normed vector space. It is a generalization of the well-known Euclidean distance. The Minkowski distance of order p between two points $x = (x_1, x_2, \cdots, x_n)$ and $y = (y_1, y_2, \cdots, y_n)$ is defined as:

$$d_{Minkowski}(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

The following figure Fig. 1 shows unit circles (the set of all points which are at the unit distance from the center) with different values of p.

2.2.2 Euclidean Distance

The Euclidean distance is the "ordinary" straight-line distance in Euclidean space. It is also a specialization of Minkowski distance where p = 2. Thus, it is defined as:

$$d_{Euclidean}(x, y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

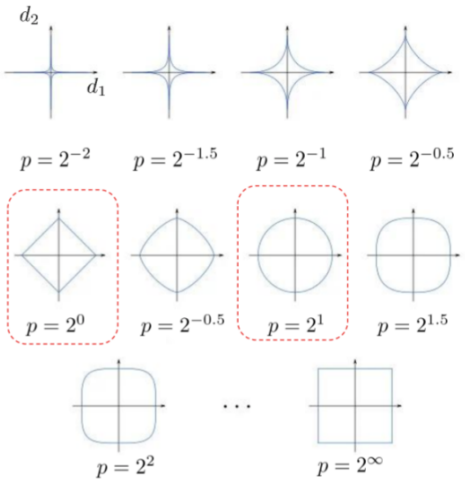


Figure 1: Unit Circles in Minkowski Distance

2.2.3 Manhattan Distance

The Manhattan distance is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. It is a specialization of Minkowski distance where p = 1. Thus, it is defined as:

$$d_{Manhattan}(x, y) = \left| \sum_{i=1}^{n} |x_i - y_i| \right|$$

2.2.4 Chebyshev Distance

The Chebyshev distance is a metric defined on a vector space where the distance between two points is the maximum of their differences along any coordinate dimension. It is actually the Minkowski distance when $p \to \infty$ and thus is defined as:

$$d_{Chebyshev}(x, y) = \lim_{p \to \infty} \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} = \max_i (|x_i - y_i|)$$

2.3 Metric Learning

2.3.1 Neighborhood Component Analysis(NCA)

NCA (Neighborhood Component Analysis) is a popular Metric Learning algorithm that aims to improve the classification performance of high-dimensional data by optimizing the distance metric in a low-dimensional space. NCA is a supervised learning algorithm commonly used to handle classification problems with high-dimensional features.

The core idea of NCA is to learn a linear transformation to map the original high-dimensional data into a low-dimensional space such that the distances between samples in the low-dimensional space are similar to those in the original high-dimensional space. Therefore, **the goal of NCA is to maximize the distances between adjacent samples belonging to different classes, while minimizing the distances between samples belonging to the same class**.

Specifically, the target function of NCA is defined as:

$$J(W) = \sum_{i=1}^{n} \frac{1}{\sum_{j=1}^{n} p_{ij}} \sum_{j=1}^{n} p_{ij} ||Wx_i - Wx_j||^2$$

where p_{ij} represents the probability that sample i and sample j belong to the same class, which can be calculated by:

$$p_{ij} = \begin{cases} \frac{1}{|y|}, & \text{if } y_i = y_j \\ 0, & \text{otherwise} \end{cases}$$

where $|y|$ represents the number of classes.

To maximize the objective function $J(W)$, we need to compute its gradient and update the parameter W using optimization methods such as gradient descent. Specifically, the gradient of the objective function $J(W)$ can be expressed as:

$$\frac{\partial J(W)}{\partial W} = \sum_{i=1}^{n}(\frac{\partial p_i}{\partial W}||Wx_i - \sum_{j=1}^{n} p_{ij}Wx_j||^2 + 2(Wx_i - \sum_{j=1}^{n} p_{ij}Wx_j)(x_i - x_j)^T)$$

where $p_i = \sum_{j=1}^{n} p_{ij}$.

Therefore, we can use the following update rule to update the parameter W:

$$W_{t+1} = W_t + \eta \frac{\partial J(W)}{\partial W}$$

where η denotes the learning rate.

By iteratively updating the parameter W, we can obtain an optimal linear transformation matrix that maps the original high-dimensional data to a low-dimensional space for better classification.

In a word, the process of NCA algorithm could be summarized as followed:

1. Initializing the linear transformation matrix W;
2. For each sample i in the training set, computing the distances between i and all other samples, as well as the class labels of i and all other samples;
3. For the current sample i, computing the sum of distances between i and all samples belonging to the same class, as well as the sum of distances between i and all samples belonging to different classes;
4. Updating the linear transformation matrix W based on the distance information of the current sample i;
5. Repeating steps 2-4 until W converges or reaches the maximum number of iterations.

2.3.2 Siamese networks

Siamese networks are a type of **neural network-based** Metric Learning algorithm, whose main purpose is to learn a distance metric function that minimizes the distance between similar samples and maximizes the distance between dissimilar samples. The name Siamese network comes from the twin neural networks it contains, which share the same weights and parameters.

The input of a Siamese network is usually a pair of samples, such as two images. This pair of samples is fed into two sub-networks, each of which consists of multiple layers, including convolutional layers, pooling layers, fully connected layers, etc. The two sub-networks have the same structure and parameters, but they process different inputs. After processing the inputs, the two sub-networks compare the output feature vectors to calculate their distance or similarity. This is usually done using common distance metrics such as Euclidean distance, Manhattan distance, or cosine similarity.

In practice, we usually uses **a single network instead of two identical networks** for that they share the same network architecture and weights. During the training of a Siamese network, we need to provide a pair of samples and their similarity/dissimilarity labels. If two samples are similar, their similarity label is 1, otherwise it is 0. During the training process, we use these labels to calculate the loss function between the predicted values and the true values.

Contrastive Loss Contrastive Loss is a commonly used loss function in Siamese networks, whose main purpose is to minimize the distance between similar samples and maximize the distance between dissimilar samples. The calculation of Contrastive Loss is as follows:

Suppose there is a pair of samples (x_1, x_2), and their similarity label is y, where $y = 1$ indicates similarity and $y = 0$ indicates dissimilarity. Let d_{ij} be the distance between samples x_i and x_j (e.g., Euclidean distance or Manhattan distance), then the calculation of Contrastive Loss is as follows:

$$L_{ij} = \frac{1}{2}(1 - y_{ij})d_{ij}^2 + \frac{1}{2}y_{ij}\max(m - d_{ij}, 0)^2$$

Here, m is a hyperparameter that controls the boundary of the minimum distance between similar samples.

- When $y_{ij} = 0$, L_{ij} represents the loss between dissimilar samples, and we want the distance d_{ij} between them to be greater than m, otherwise the loss will increase.
- When $y_{ij} = 1$, L_{ij} represents the loss between similar samples, and we want the distance d_{ij} between them to be as small as possible, otherwise the loss will increase.

By minimizing the sum of L_{ij} for all pairs of samples, we can train the parameters of the Siamese network to learn a distance metric function.

2.3.3 Triplet Loss

Triplet loss is a commonly used loss function in Metric Learning, which is used to train a neural network to learn a distance metric function, so that the distance between similar samples is small and the distance between dissimilar samples is large under this metric. The idea of Triplet loss is to train the network by comparing **the distances between three samples, including an anchor sample, a positive sample, and a negative sample.**

Specifically, given an anchor sample x_a, a positive sample x_p, and a negative sample x_n, their feature vectors are $f(x_a)$, $f(x_p)$, and $f(x_n)$, respectively, where $f(\cdot)$ represents the feature extraction function of the network. We want the distance $d_{ap} = ||f(x_a) - f(x_p)||$ between the anchor sample and the positive sample to be small in the feature space, while the distance $d_{an} = ||f(x_a) - f(x_n)||$ between the anchor sample and the negative sample to be large.

The calculation of Triplet loss is as follows:

$$L_{triplet} = \max(0, d_{ap} - d_{an} + margin)$$

where $margin$ is a hyperparameter that controls how much larger the distance between the anchor sample and the negative sample should be compared to the distance between the anchor sample and the positive sample.

- When $d_{ap} - d_{an} + margin < 0$, $L_{triplet} = 0$, indicating that the distances between the three samples already satisfy the requirement and there is no need to update the network parameters.
- When $d_{ap} - d_{an} + margin > 0$, $L_{triplet} = d_{ap} - d_{an} + margin > 0$, indicating that the network parameters need to be updated to reduce d_{ap} or increase d_{an}, so that the distances between the three samples better meet the requirement.

The training process of Triplet loss usually involves **selecting triplets of samples** from a large dataset for training, where the positive samples are from the same class as the anchor samples while the negative samples are from different classes from the anchor samples. Since Triplet loss requires selecting a suitable number of high-quality triplet samples for training, some techniques are needed to improve training efficiency and stability in practical applications, such as **online hard sample mining**, dynamic adjustment of margin, etc.

In this project, I implemented **online hard sample mining**, so here I will give a brief introduction to it.

Online hard sample mining Online hard sample mining is a technique used to improve the training effectiveness of Metric Learning models. Its main idea is to **select the hardest samples** for training in each batch, so that the model can pay more attention to those difficult-to-distinguish samples.

Specifically, the steps are as followed:

- For each training batch, we first calculate the Triplet loss for all triplet samples;
- Select the hardest samples as the training samples for the current batch;
- Typically, we choose a certain number of hardest samples for training, such as selecting the top K samples with the highest loss value.

By training with the hardest samples, we can make the model pay more attention to those difficult-to-distinguish samples, thereby improving the model's generalization ability and robustness.

3 Experimental Result

3.1 Baseline

In this experiment, I use the basic KNN model to straightly do classification on the original 2048-dim features. Then, I use PCA and LDA respectively to reduce the feature dimentionality to 48 and compare the classification accuracy to the baseline KNN algorithm.

In this part, I set the distance metric of KNN to 'euclidean' and the result is as shown in Table 1

k	KNN(%)	KNN+PCA(%)	KNN+LDA(%)
1	87.46404442	87.14964212	90.46090039
2	85.99237407	85.90541173	90.2735969
3	88.43400896	88.206569	91.95263897
4	88.42063014	88.54772895	91.9058131
5	88.74172185	89.10295003	92.19345776
6	**88.86882066**	89.04943474	92.09980601
7	88.78854773	89.39059469	92.26704127
8	88.75510068	89.21667001	92.39414008
9	88.71496421	**89.40397351**	92.30048833
10	88.80192655	89.12301826	92.34062479
15	88.55441836	89.16984414	92.44765536
20	88.17981136	88.90226771	**92.54130711**
50	86.3870493	87.71824202	92.44096595
100	84.49394608	86.23988227	92.26704127
200	84.49394608	84.03237675	91.93257074
1000	72.22556693	76.93491203	87.06267978

Table 1: Comparison of Baseline KNN and Dimensionality Reduction + KNN

From the result, we can see that:

1. Dimensionality Reduction methods is useful in this task, indicated from that PCA and LDA bring performance improvement with 0.54% and 3.67% respectively.
2. With the k value increasing, the classification accuracy goes higher at the first stage and then starts to go down. Indicating that when using KNN algorithm, a proper k is of great importance for the classification accuracy.
3. When using dimensionality Reduction methods, especially LDA, the best k is greater than that in the original KNN, which confirms that the data dimensionality reduction methods are effective in bringing the data belonging to the same class closer.

3.2 K-fold cross-validation

In this experiment, I conducted 5-fold cross-validation on three hyper-parameters of the KNN algorithm:

- The k value: takes k nearest neighbors into account.
- The weights: to determine the weight of each nearest neighbor sample, where 'uniform' means each neighbor shares the same weight while 'distance' means the weight of the nearest neighbor sample is inversely proportional to the distance between the test sample.
- The normalization flag: a self-defined parameter indicating whether to normalize the data.

The result is as shown in Table 2.

k	weight = 'uniform'		weight = 'distance'	
	norm(%)	no_norm(%)	norm(%)	no_norm(%)
1	87.4126	88.514	87.4126	88.514
2	85.7484	87.1365	87.3992	88.5274
3	88.0236	89.4037	88.4738	89.7226
4	88.1629	89.4386	88.9535	90.0844
5	88.5837	89.8647	88.916	90.1032
6	88.3639	89.7146	**89.05**	**90.2881**
7	**88.5944**	**89.8673**	88.9267	90.1353
8	88.4497	89.6744	88.9294	90.1755
9	88.4229	89.8566	88.707	90.063
10	88.3344	89.7816	88.7659	90.1755

Table 2: K-fold Cross Validation on k, weight and norm

From the result, we can see that:

1. Weights 'distance' is better than weights 'uniform' in this task. The reason for it maybe that the data is relatively dense and the weights 'distance' could better classify the data when it's dense.
2. Normalization could slightly improve the performance of KNN in this task by about 0.5%.

3.3 Basic Distance Metric

In this experiment, I uses KNN with three different basic distance metric, 'euclidean', 'manhattan' and 'chebyshev', together with different weights methods to conduct classification.

The result is as shown in Table 3.

k	weight = 'uniform'			weight = 'distance'		
	Euclidean(%)	Manhattan(%)	Chebyshev(%)	Euclidean(%)	Manhattan(%)	Chebyshev(%)
1	87.46404442	87.11619506	76.43320623	87.46404442	87.11619506	76.43320623
2	85.99237407	85.62445649	74.70064887	87.46404442	87.11619506	76.43320623
3	88.43400896	87.59114322	77.38979196	88.68820657	88.0192655	78.32630945
4	88.42063014	87.59114322	78.00521774	89.02936651	88.22663723	78.83470466
5	88.74172185	**87.8921667**	78.29955181	88.98923005	**88.27346311**	79.08221286
6	**88.86882066**	87.5777644	78.12562713	**89.26349589**	88.2667737	78.98856111
7	88.78854773	87.88547729	78.46678708	88.97585123	88.13967489	78.96849288
8	88.75510068	87.73162084	**78.59388588**	89.02936651	88.10622784	**79.26282694**
9	88.71496421	87.72493143	78.44671884	88.87551007	87.95906081	79.14910696
10	88.80192655	87.73831025	78.22596829	89.06281357	87.96575022	79.13572814
15	88.55441836	87.14295271	78.09218008	88.72834303	87.41052913	78.62733293
20	88.17981136	86.58104221	77.71757308	88.43400896	86.83523982	78.15907419
50	86.3870493	84.40029433	75.2826276	86.53421634	84.57421901	75.81778045
100	84.49394608	82.4336076	72.29915044	84.67456017	82.58077463	72.82761389
200	82.04562178	78.94173523	68.91430865	82.27975115	79.40330457	69.75048498
1000	72.22556693	64.15144826	57.10080942	73.32931969	66.37902201	59.38858787

Table 3: KNN with different Basic Distance Metric

From the result, we can see that:

1. Weights 'distance' is better than weights 'uniform' in this task.
2. Euclidean distance metric is optimal for this task, Manhattan distance metric is slightly worse while Chebyshev distance metric is the worst (nearly 10% worse).

3.4 NCA

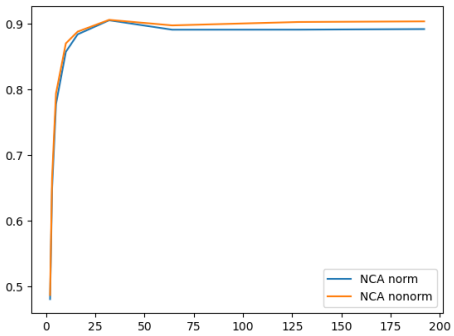
In this experiment, I conducted NCA on the data and take the transformed data as the input of KNN to do classification. Before NCA, we can choose whether to do data normalization or not.
the result is as shown in Table 4.

output dim	NCA_nonorm(%)	NCA_norm(%)
2	48.7056	48.07679
3	66.45929	64.86722
5	79.38993	77.77109
10	87.00248	85.71811
16	88.79524	88.38049
32	**90.588**	**90.52111**
64	89.74513	89.08957
128	90.26022	89.09626
192	90.35387	89.18322

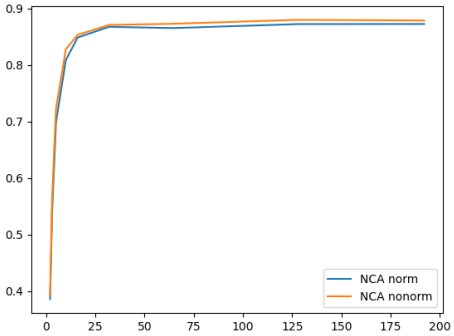
Table 4

More detailed statistic are shown in Fig 2.
From the result, we can see that:

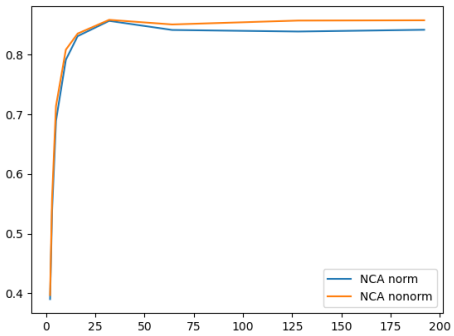
1. Even when reducing the data to a low dimentionality, KNN with NCA-transformed data still has a relatively good performance, which confirms the effectiveness of NCA.
2. NCA metric learning method is more effective than those basic distance metric like euclidean distance.
3. Data normalization is of no use and even harmful when conducting NCA.
4. The performance of NCA increases in the first stage and then slightly goes down as the output dim increases.



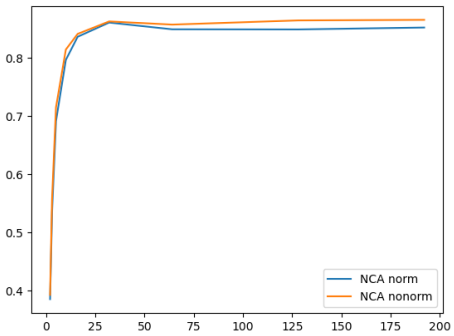
(a) Accuracy



(b) Precision



(c) Recall



(d) F1 Score

Figure 2: Detailed statistic in NCA experiment

3.5 Siamese network

I conducted two experiments with Siamese network and here I will give detailed illustration for both of them.

3.5.1 Experiment on output dimentionality

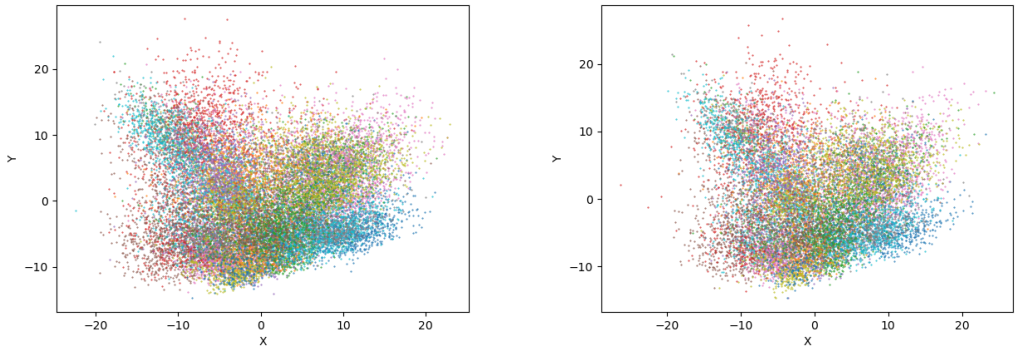
In Siamese network method, there's a simple network built up with several linear layers connected with relu function. The output dimensionality is designed to be a hyper-parameter and different output dim will significantly affect the performance of the KNN. In this view, the Siamese network could be seen as a dimensionality reduction method just like PCA. So in this experiment, I use Siamese together with PCA and LDA to reduce the data dimensionality and take the transformed data as the input of KNN, comparing the performance of the three algorithm.

I conduct the experiment with different setting of output dim and the result is as shown in Table 5.

output dim	Siamese(%)	PCA(%)	LDA(%)
2	82.1	19.6	25
5	92	53.1	53.6
16	92.1	83.5	78.2
32	92	87.9	88.9
64	92.1	89.4	-
128	92	89.7	-
1024	91.3	88.9	-

Table 5

The distribution of training dataset and testing dataset reduced to 2d using PCA, LDA and Siamese network are shown respectively in Fig 3, Fig 4 and Fig 5.



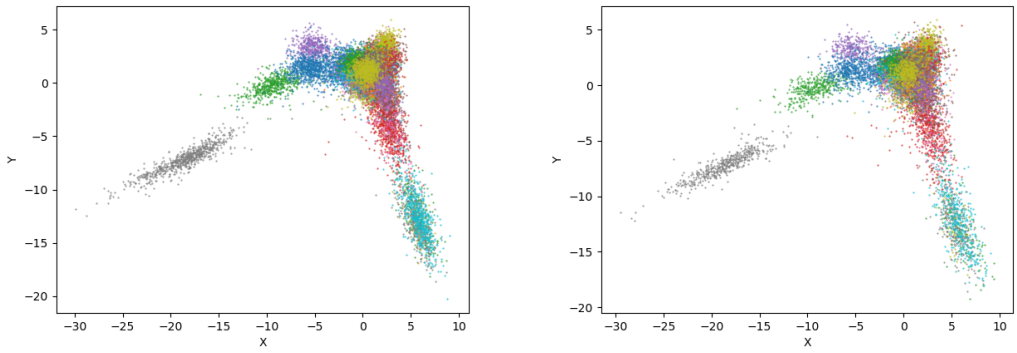
(a) Training data

(b) Testing data

Figure 3: Distribution of training and testing dataset reduced to 2d by PCA

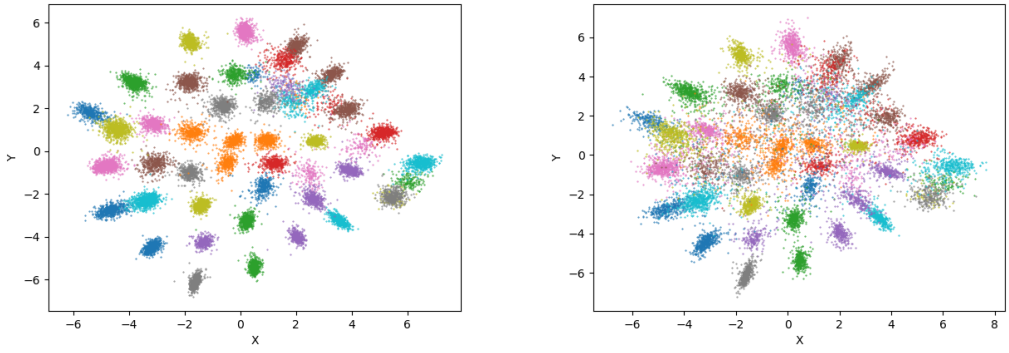
From the result, we can see that:

1. Siamese network method is much more effective compared to PCA and LDA, especially when the target output dimensionality is low.
2. Siamese network could reduce the dimensionality of the data greatly and still keeps the data belonging to the same class close and the data belonging to different classes far away with each other.
3. A vary low dimensionality (like 5) is enough for Siamese network to keep the classification information.



(a) Training data (b) Testing data

Figure 4: Distribution of training and testing dataset reduced to 2d by LDA



(a) Training data (b) Testing data

Figure 5: Distribution of training and testing dataset reduced to 2d by Siamese network

3.5.2 Experiment on sampling parameter ϵ

In the Siamese network, each step there should be a couple of data put into the two networks. For each data in a batch, the original algorithm will randomly select another data in the dataset to act as the second data in the data couple. In order to make the network learn more efficiently, I set a hyper-parameter ϵ to control the construction of the data couple.

Specifically, the meaning of ϵ is:

- With probability ϵ, the selected data will have the same label with the original data.
- With probability $1 - \epsilon$, the selected data is randomly picked from the dataset.

By tuning the parameter ϵ, the learning efficiency of the algorithm will also be affected. The result is as shown in Table 6.

epsilon	Accuracy
0.05	92.3
0.1	92.5
0.2	90.7
0.5	88.2
0.8	82.2
1.0	3.2

Table 6: Accuracy with different Epsilon in Siamese Network

From the result, we can see that the classification accuracy goes high in the first stage and then goes down as the value of ϵ increases. This meet our expectation as well for that we can infer:

1. A smaller value of ϵ means the composition of data couple will have more randomness and the network are more likely to see different data label combination. This will benefit the robustness of the algorithm but also make the training of the network more difficult. The network will takes more data and steps to be well trained.

2. A larger value of ϵ means a higher probability that the data couple are from the same class, thus it will be more easy for the network to learn the metric to project the data from the same class to closer positions in the new space. But also, the too high value of ϵ will harm the robustness of the algorithm and all the network sees is the data come from the same classes, which will lead to the situation that the network has no idea how to make the data from different classes far away from each other in the new space. So now maybe the most easy way for the network to minimize the loss is to project all the data to a same single point, which is obviously not what we want.

3.6 Triplet loss

I conducted two experiments with Siamese network and here I will give detailed illustration for both of them.

3.6.1 Experiment on output dimentionality

Just as that in Siamese network, in Triplet loss method, there's also a neural network with almost the same architecture as the network in Siamese network. So it could also be seen as a dimensionality reduction method.

I conduct the experiment with different setting of output dim and the result is as shown in Table 7.

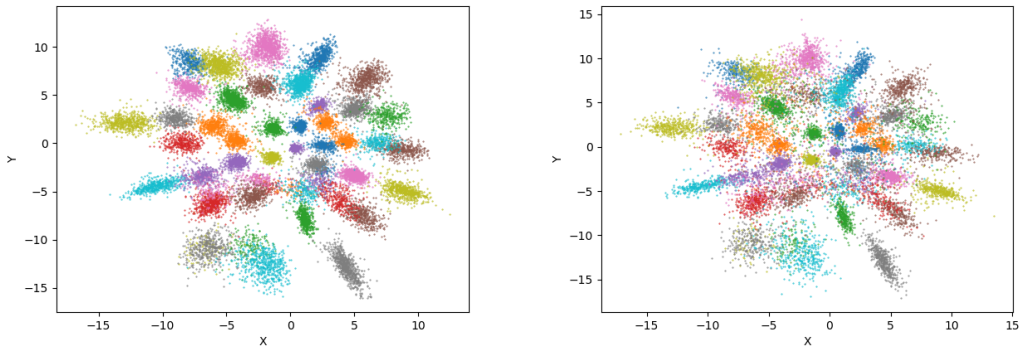
The distribution of training dataset and testing dataset reduced to 2d using Triplet loss are shown in Fig 6.

Comparing to the result of Siamese network in Table 5 and Fig 5, we can see that:

1. Triplet loss method is almost of the same good performance with the Siamese network and are much better than PCA and LDA when conducted on classification task.

output dim	Triplet(%)	Siamese(%)	PCA(%)	LDA(%)
2	78.2	82.1	19.6	25
5	89.5	92	53.1	53.6
16	92.3	92.1	83.5	78.2
32	91.3	92	87.9	88.9
64	92.6	92.1	89.4	-
128	92.7	92	89.7	-
1024	91.7	91.3	88.9	-

Table 7



(a) Training data

(b) Testing data

Figure 6: Distribution of training and testing dataset reduced to 2d by Triplet loss

2. More detailed, Triplet loss is slightly worse than the Siamese network when the output dimensionality is low but outperform the Siamese network when the output dimensionality is high (like greater than 16).

3. From the distribution figures Fig 5 and Fig 6, we can also see that the Triplet loss method doesn't separate the data points from different classes away as well as that in Siamese network. This may result from that the Triplet network is more difficult to train and here it's still under-trained.

3.6.2 Experiment on Online hard sample mining

As introduced in section 2.3.3, Online hard sample mining is a method to improve the training efficiency of the Triplet network. I implemented Online hard sample mining in this project in which for every anchor sample, I randomly pick 10 negative samples and calculate the most difficult one to act as the negative sample put into the real training process.

Here I conducted an experiment on it. The result is as shown in Table 8.

output dim	Accuracy	Training Time(s)	Accuracy(Ohsm)	Training Time(Ohsm)(s)
2	78.2	3232.6731045246124	80.3	14692.793508529663
16	92.3	3293.970034599304	92.0	14823.191870212555
32	91.3	3302.925131082535	92.0	14615.924565553665
64	92.6	3219.535559654236	91.7	14714.675187587738
128	92.7	3266.530208826065	92.6	14161.639461994171

Table 8

From the result, we can see that:

1. Online hard sample mining doesn't help with this task and brings no obvious performance improvement. The reason for this may be that this task is too easy for the Triplet network and there's no need to use such a improvement method.

2. Online hard sample mining is time-consuming. The process to find the most difficult sample in the candidate list takes a lot of calculation and makes Online hard sample much slower than the original algorithm.

4 Conclusion

In this paper, we introduced the common basic distance metric and implemented three another metric learning methods, namely NCA, Siamese network and Triplet loss. The best performance comes from Triplet loss method with output dim set as 128, achieves accuracy 92.7% on testing set.

Furthermore, I think that uses the combination of PCA/LDA before metric learning methods like Siamese network or Triplet loss may bring an further improvement on the classification accuracy. Due to time limitation I didn't conduct it and illustrate it here.