Deep Metric Learning with Data Summarization

Wenlin Wang[†]✉, Changyou Chen[†], Wenlin Chen[‡], Piyush Rai[♯], and Lawrence Carin[†]

[†]Dep. of Electrical & Computer Engineering, Duke University
[‡]Dept. of Computer Science & Engineering, Washington Univeristy in St. Louis
[♯]Dept. of Computer Science & Engineering, IIT Kanpur
{ww107,cc448,piyush.rai,lcarin}@duke.edu, dtccwl@gmail.com

Abstract. We present Deep Stochastic Neighbor Compression (DSNC), a framework to compress training data for instance-based methods (such as k-nearest neighbors). We accomplish this by inferring a smaller set of *pseudo-inputs* in a new feature space learned by a deep neural network. Our framework can equivalently be seen as *jointly* learning a nonlinear distance metric (induced by the deep feature space) and learning a compressed version of the training data. In particular, compressing the data in a deep feature space makes DSNC robust against label noise and issues such as within-class multi-modal distributions. This leads to DSNC yielding better accuracies and faster predictions at test time, as compared to other competing methods. We conduct comprehensive empirical evaluations, on both quantitative and qualitative tasks, and on several benchmark datasets, to show its effectiveness as compared to several baselines.

1 Introduction

In machine learning problems there are situations for which the massive data scale renders learning algorithms infeasible to run in a reasonable amount of time. One solution is to first summarize the data in the form of a small set of *representative* data points that best characterize and represent the original data, and then run the original algorithm on this subset of the data. This may be desirable due to the requirement of making a fast prediction at test time, in problems where the predictions depend on the entire training data, e.g., k-nearest neighbors (kNN) classification [8,4] or kernel methods such as SVMs [21]. For example, in traditional kNN classification, the prediction cost for each test example scales linearly in the number of training examples, which can be expensive if the number of training examples is large. Traditional approaches to speed-up such methods usually rely on cleverly designed data structures or select a compact subset of the original data (e.g., via subsampling [4]). Although such methods may reduce the storage requirements and/or prediction time, the performance tends to suffer, especially if the original data is high-dimensional and/or noisy.

Recently [24] introduced Stochastic Neighbor Compression (SNC), which learns a set of *pseudo-inputs* for kNN classification by minimizing a stochastic 1-nearest neighbor classification error on the training data. Compared to the data sub-sampling approaches, SNC achieves impressive improvements in test accuracy when using these pseudo-inputs as the new training set. However, since SNC performs data compression in the original data space (or in a linearly transformed lower-dimensional space), it may perform poorly when the data in the original space are highly non-separable and noisy.

Motivated by this, we present Deep Stochastic Neighbor Compression (DSNC), a new framework to jointly perform data summarization akin to the methods like SNC, while also learning a *nonlinear* feature representation of the data via a deep learning architecture. Our framework is based on optimizing an objective function that is designed to learn nonlinear transformations that preserve the neighborhood structure in the data (based on label information), while simultaneously learning a small set of pseudo-inputs that summarize the entire data. Note that, due to the neighborhood preserving property, our framework can also be viewed as performing a nonlinear (deep) distance metric learning [22], while also learning a summarized version of the original data. The data summarization aspect also makes DSNC much faster than other metric learning based approaches which need all the training data. In DSNC, the data summarization and feature learning, both, are performed jointly through backpropagation [31] using stochastic gradient descent, making our framework readily scalable to large data sets. Moreover, our framework is also more general than standard feedforward neural networks which perform simultaneous feature learning and classification but are not designed to learn a summary of the data which may be useful in its own right.

In our comprehensive empirical studies, DSNC achieves superior classification accuracies on the seven datasets we used in the experiments, outperforming SNC by a significant margin. For example, with DSNC, 1-NN is able to achieve 0.67% test error on MNIST with only ten compressed data samples (one per class) on a 20-dimensional feature space, compared to 7.71% for SNC. We also report qualitative experiments (via visualization) showing that DSNC is effective in learning a good summary of the data.

2 Background

Throughout this paper, we denote vectors as bold, lower-case letters, and matrices as bold, upper-case letters. $\|\cdot\|$ applied to a vector denotes the standard vector norm, $[\mathbf{X}]_{ij}$ means the (i,j)-th element of matrix $\mathbf{X}$. We denote the training data $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, where $\mathbf{X} \in \mathbb{R}^{D \times N}$ are N observed data samples of dimensionality D with corresponding labels $\mathbf{Y} = \{y_1, ..., y_N\} \in \mathcal{Y}^N$, with $\mathcal{Y}$ as a discrete set of possible labels.

To motivate our proposed framework DSNC (described in Section 3), we first provide an overview of Neighborhood Components Analysis (NCA) [16,32] and Stochastic Neighbor Compression (SNC) [24], which our proposed framework builds on.

2.1 Neighborhood Components Analysis

Neighborhood Components Analysis (NCA) [16] is a distance metric learning method that learns a mapping $f(\cdot|\mathbf{W})$ with parameters $\mathbf{W}$ to optimize the k-nearest-neighbors classification objective. The optimization is based on preserving the Euclidean distance $d_{ij} = \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|^2$ in the transformed space for $\mathbf{x}_i$ and $\mathbf{x}_j$, based on their original neighborhood relationship in the original space. Specifically, soft neighbor assignments are used in NCA to directly optimize the mapping f for kNN classification performance. The probability p_{ij} that $\mathbf{x}_i$ is assigned to $\mathbf{x}_j$ as its stochastic nearest-neighbor is modeled with a softmax over distances between $\mathbf{x}_i$ and the other training samples, *i.e.*, $p_{ij} = \frac{\exp(-d_{ij})}{\sum_{k:k\neq i}\exp(-d_{ik})}$. The objective of NCA is to maximize the expected

number of correctly classified points, expressed here as a log-minimization problem: $\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} - \sum_{i=1}^{N} \log(p_i)$, where p_i is the probability that the mapped sample $f(\mathbf{x}_i|\mathbf{W})$ is correctly classified with label y_i, *i.e.*, $p_i = \sum_{j:y_i=y_j} p_{ij}$. Although NCA can learn a distance metric adaptively from data, the entire training data still needs to be stored, making it computationally and storage-wise expensive at test time. To extend NCA with nonlinear transformations, [32] defines $f(\cdot|\mathbf{W})$ to be a feedforward neural network parameterized by weights $\mathbf{W}$.

2.2 Stochastic Neighbor Compression (SNC)

Stochastic Neighbor Compression (SNC) is an improvement over NCA by learning a compressed kNN training set by optimizing a soft neighborhood objective [24] . The goal in SNC is to find a subset of $m \ll N$ compressed samples $\mathbf{Z} = [\mathbf{z}_1, ..., \mathbf{z}_m]$ with labels $\hat{\mathbf{Y}} = [\hat{y_1}, ..., \hat{y_m}]$, to best approximate the kNN decision rule on the original set of training samples $\mathbf{X}$ and labels $\mathbf{Y}$. Different from NCA, a compressed set $\mathbf{Z}$ needs to be learned from the whole data. The objective is to maximize the stochastic nearest-neighbor accuracy with respect to $\mathbf{Z}$, *i.e.*, $\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} - \sum_{i=1}^{N} \log(p_i)$, where the probability of a correct assignment between a training sample $\mathbf{x}_i$ and the compressed neighbors $\mathbf{z}_i$ is defined as $p_i = \sum_{j:y_i=y_j} \frac{\exp(-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_j\|^2)}{\sum_{k=1}^{m} \exp(-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_k\|^2)}$, where γ is the width of the Gaussian kernel. Given such probabilities, the objective of SNC is constructed as in the case of NCA and is optimized w.r.t. the m pseudo-inputs $\mathbf{Z}$. In [24], a *linear* metric learning extension of this approach was also considered, which defines $p_i = \sum_{j:y_i=y_j} \frac{\exp(\|-\mathbf{A}(\mathbf{x}_i - \mathbf{z}_j)\|^2)}{\sum_{k=1}^{m} \exp(-\gamma^2 \|-\mathbf{A}(\mathbf{x}_i - \mathbf{z}_k)\|^2)}$, in which the pseudo-inputs will be learned in the linearly transformed space. However, in the case of noisy and highly non-separable data sets, the linear transformation may not be able to learn a good set of pseudo-inputs. Our proposed framework, on the other hand, is designed to learn these pseudo-inputs, while simultaneously learning a nonlinear feature representation for these.

3 Deep Stochastic Neighbor Compression

Our proposed framework *Deep Stochastic Neighbor Compression* (DSNC) is based on the idea of summarizing/compressing data in a *nonlinear* feature space learned via a deep feedforward neural network. Although methods like SNC (Section 2.2) can achieve a significant data compression, the inferred pseudo-inputs $\mathbf{Z}$ still belong to the *original* feature space, or a linear subspace of the original data. In contrast, DSNC learns $\mathbf{Z}$ in a more expressive, nonlinear feature space. Note that, in our framework, nonlinear feature learning naturally corresponds to a nonlinear (deep) metric learning.

DSNC consists of a deep feedforward neural network architecture which jointly learns a compressed set $\mathbf{Z} \in \mathbb{R}^{d \times m}$ with *m* pseudo-inputs ($m \ll N$), along with a deep feature mapping $f(\cdot|\mathbf{W})$ from the original feature space $\mathbb{R}^D$ to a transformed space $\mathbb{R}^d$. The procedure is illustrated in Figure 1. The set $\mathbf{Z}$ consisting of the inferred pseudo-inputs and the deep feature representation $f(\cdot|\mathbf{W})$ are used as a reference set and feature transformation, respectively, at test-time of an instance based method such as kNN classification. In the following we describe the key components of DSNC.

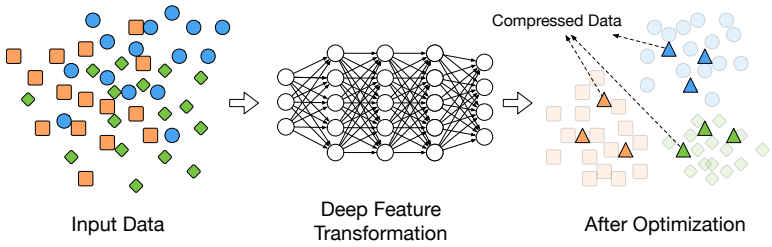

Fig. 1. A conceptual illustration of DSNC, which transforms the data via a deep feedforward neural net while simultaneously learning the pseudo-inputs that summarize the original data.

3.1 Deep Stochastic Reference Set

Let $f(\cdot|\mathbf{W}) : \mathbb{R}^D \rightarrow \mathbb{R}^d$ be a deep neural network mapping function, with $\mathbf{W}$ as the set of parameters from all layers of the network.[1] Similar to SNC, we aim to learn a compressed set of pseudo-inputs, $\mathbf{Z} = [\mathbf{z}_1, \cdots, \mathbf{z}_m]$ with $\mathbf{z} \in \mathbb{R}^d$, such that $\mathbf{Z}$ summarizes the original training set in the deep feature space. To this end, akin to SNC, we define the probability that input $\mathbf{x}_i$ chooses $\mathbf{z}_j$ as its nearest reference vector as:

$$p_{ij} = \frac{\exp(-\gamma^2 \|f(\mathbf{x}_i) - \mathbf{z}_j\|^2)}{\sum_{k=1}^{m} \exp(-\gamma^2 \|f(\mathbf{x}_i) - \mathbf{z}_k\|^2)} \,. \tag{1}$$

In the optimization, in addition to learning the parameters of a deep neural network, the compressed set $\mathbf{Z}$ is also learned from the data. This is done by first initializing $\mathbf{Z}$ with m randomly sampled examples from $\mathbf{X}$, noted as $\mathbf{X}^{'}$, and then computing their deep representation via $\mathbf{Z} = f(\mathbf{X}^{'})$, while recording their original labels. Note that while learning f and $\mathbf{Z}$, these labels are fixed throughout, while $\mathbf{Z}$ and the parameters $\mathbf{W}$ of the deep mapping f are learned jointly with the objective defined below.

3.2 DSNC Objective

To define an objective function for DSNC, we would like to ensure $p_i \triangleq \sum_{j:y_i=y_j} p_{ij} = 1$ for all $\mathbf{x}_i \in \mathbf{X}$, where p_{ij} is defined in (1). This means that the probability p_{ij} corresponding to an input $\mathbf{x}_i$ and a pseudo-input $\mathbf{z}_j$, both having different labels, is zero. We then define the KL-divergence between the "perfect" distribution "1" and p_i as

$$KL(1\|p_i) = -\log(p_i) \tag{2}$$

We wish to find a compressed set $\mathbf{Z}$ such that as many training inputs as possible are classified correctly in the deep feature space. In other words, we would like p_i to be close to 1 for all $\mathbf{x}_i \in \mathbf{X}$. This leads to the following objective:

$$\tilde{\mathcal{L}}(\mathbf{Z}, \mathbf{W}) = -\sum_{i=1}^{n} \log(p_i) \,, \tag{3}$$

[1] For conciseness, we will typically omit the parameters $\mathbf{W}$ from the notation for the mapping function, *i.e.*, $f(\cdot) \triangleq f(\cdot|\mathbf{W})$.

Algorithm 1 DSNC in pseudo-code

1: **Input:** $\{\mathbf{X}, \mathbf{Y}\}$; compressed data set size $\mathbf{m}$
2: Initialize $\mathbf{Z}$ by sampling $\mathbf{m}$ inputs from $\mathbf{X}$, uniformly in each class, and forwarding into the initialized deep neural network $f(\cdot)$
3: Learn $\mathbf{Z}$ and the deep networks $f(\cdot)$ with back-propagation using gradients in (5) and (6)
4: Return $f(\cdot)$ and $\mathbf{Z}$

where $\mathbf{W}$ denotes the parameters of the deep feedforward neural network.

There are two possible issues that may arise while optimizing the objective (3) for DSNC and need to be properly accounted for. First, since we are jointly learning the deep feature map f and the compressed set $\mathbf{Z}$, without any constraints, it is possible that the mapped samples $f(\mathbf{x}_i)$ are on a different scale than the compressed samples $\mathbf{Z}$ in the deep feature space, while achieving a small value for the objective function (3). To handle this issue, we encourage the distance between $f(\mathbf{x}_i)$ and $\mathbf{z}_j$ to be *small* to avoid an inhomogeneous distribution in the feature space.

Second, it is also possible that all the compressed data samples with the same label collapse into a single point since our objective aims to maximize the classification accuracy. As a result, we also penalize the distribution of the compressed samples to encourage a multi-modal distribution for each label. This is done by *maximizing* the pair-wise distance between two pseudo-inputs $\mathbf{z}_i$ and $\mathbf{z}_j$ with the same label. Consequently, the DSNC objective function combines the KL-divergence term $\tilde{\mathcal{L}}(\mathbf{Z}, \mathbf{W})$ with two additional regularization terms to account for these, and is given by

$$\begin{aligned}\mathcal{L}(\mathbf{Z}, \mathbf{W}) = &- \sum_{i=1}^{n} \log(p_i) + \lambda_1 \underbrace{\sum_{i=1}^{n}\sum_{j=1}^{m} \|f(\mathbf{x}_i) - \mathbf{z}_j\|^2}_{R_1} \\ &- \lambda_2 \underbrace{\sum_{i=1}^{m}\sum_{j=1}^{m} \delta(\hat{y_i}, \hat{y_j}) \|\mathbf{z}_i - \mathbf{z}_j\|^2}_{R_2} \end{aligned} \tag{4}$$

where λ_1 and λ_2 are regularization coefficients and the delta function $\delta(\hat{y_i}, \hat{y_j}) = 1$ if $\hat{y_i} = \hat{y_j}$, and 0 otherwise. $\{\hat{y_i}\}$ are the labels for the compressed set $\mathbf{Z}$. R_1 regularizes the compressed samples to be close to the training data in the deep feature space, while R_2 encourages compressed samples with the same label to dissociate.

3.3 Learning with stochastic gradient descent

The objective function (4) can be easily optimized via the back-propagation algorithm with stochastic gradient descent [7]. We adopt the RMSProp algorithm [35].

Specifically, there are two components that need to be updated: the parameters $\mathbf{W}$ of the deep neural network, and the compressed set $\mathbf{Z}$. Parameters $\mathbf{W}$ are updated by back-propagation, which requires the gradient of the objective with respect to the output $f(\mathbf{X})$, which is then back-propagated down the neural network. The compressed set $\mathbf{Z}$

can be simply updated with a stochastic gradient descent step. The stochastic gradients for both $\mathbf{Z}$ and $f(\mathbf{X})$ have simple and compact forms. To write down the gradients, we first define the following matrices $\{\mathbf{Q}, \mathbf{P}, \hat{\mathbf{P}}\} \in \mathbb{R}^{n\times m}$, $\mathbf{Q}_1 \in \mathbb{R}^{m\times m}$, $\mathbf{P}_1 \in \mathbb{R}^{d\times n}$, and $\{\mathbf{P}_2, \mathbf{Q}_2\} \in \mathbb{R}^{d\times m}$ as

$$
\begin{aligned}
&[\mathbf{Q}]_{ij} = (\delta_{y_i,\hat{y_j}} - p_i), \quad [\mathbf{Q}_2]_{ij} = \sum_{i=1}^{m} \mathbf{Q}_{ij} \\
&[\mathbf{Q}_1]_{ij} = \delta(\hat{y_i}, \hat{y_j}), \qquad [\mathbf{P}]_{ij} = \frac{p_{ij}}{p_i}, \qquad [\hat{\mathbf{P}}]_{ij} = p_{ij} \\
&[\mathbf{P}_1]_{ik} = \sum_{j}^{m} \mathbf{z}_{ij}, \qquad [\mathbf{P}_2]_{jk} = \sum_{i}^{n} \mathbf{x}_{ji}
\end{aligned}
$$

Here, p_{ij} is defined in (1), x_{ij} and z_{ij} denote the corresponding elements of row/column i/j in X/Z. After some careful algebra, the gradient of $\mathcal{L}$ with respect to the compressed set $\mathbf{Z}$ and $f(\mathbf{X})$ can then be conveniently represented in matrix operations with the above defined symbols, *i.e.*,

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} = &-2\gamma^2 \left(\mathbf{X} \left(\mathbf{Q} \circ \mathbf{P} \right) - \mathbf{Z} \,\mathrm{diag}\left(\left(\mathbf{Q} \circ \mathbf{P} \right)^T \mathbf{1}_n \right) \right) \\
&+ 2\lambda_1 \left(n\mathbf{Z} - \mathbf{P}_2 \right) + 2\lambda_2 \left(\mathbf{Z}\mathbf{Q}_1 - \mathbf{Q}_2 \circ \mathbf{Z} \right) \qquad (5) \\
\frac{\partial \mathcal{L}}{\partial f(\mathbf{X})} = &-2\gamma^2 \mathbf{Z} \left(\mathbf{Q} \circ \mathbf{P} - \hat{\mathbf{P}} \right)^T + 2\lambda_1 \left(m\mathbf{X} - \mathbf{P}_1 \right) \,. \qquad (6)
\end{aligned}
$$

where $\circ$ is the Hadamard (element-wise) product, $\mathbf{1}_n$ is the $n \times 1$ vector of all ones, and $\mathrm{diag}(\cdot)$ is the diagonal operator placing a vector along the diagonal of an otherwise 0 matrix. Given the gradients, learning is straightforward by applying the RMSProp algorithm on $\mathbf{Z}$ and the back-propagation for learning W, described in Algorithm 1.

3.4 Relationship with deep neural net with softmax output

We now show how DSNC is related to a deep neural network with a softmax output. Note they are comparable only when $m = |Y|$, *i.e.*, the number of pseudo-inputs is equal to the number of classes. Note that, for a deep neural network with a softmax output, the corresponding probability for (1) can be written as

$$
p_{ij} = \frac{\exp(f^T(\mathbf{x}_i)\mathbf{z}_j)}{\sum_{k=1}^{|Y|} \exp(f^T(\mathbf{x}_i)\mathbf{z}_k)} \,.
$$

Note that the Euclidean distance in DSNC is replaced by an inner product in softmax function above. When $\gamma^2 = \frac{1}{2}$ and $||f(\mathbf{x}_i)||_2^2 = ||\mathbf{z}_j||_2^2 = 1$, the probability that $\mathbf{x}_i$ belongs to "class" $\mathbf{z}_j$, as given by (1) can be written as

$$
p_{ij} = \frac{\exp(-\frac{1}{2}\|f(\mathbf{x}_i)\|^2)\exp(-\frac{1}{2}\|\mathbf{z}_j\|^2)\exp(f^T(\mathbf{x}_i)\mathbf{z}_j)}{\sum_{k=1}^{|Y|} \exp(-\frac{1}{2}\|f(\mathbf{x}_i)\|^2)\exp(-\frac{1}{2}\|\mathbf{z}_k\|^2)\exp(f^T(\mathbf{x}_i)\mathbf{z}_k)} = \frac{\exp(f^T(\mathbf{x}_i)\mathbf{z}_j)}{\sum_{k=1}^{|Y|} \exp(f^T(\mathbf{x}_i)\mathbf{z}_k)}
$$

which exactly recovers the softmax output. Therefore, a deep neural network with a softmax output can be viewed as a special case of our DSNC framework.

4 Related Work

Our work is aimed at improving the accuracies of instance based methods, such as kNN, by learning highly discriminative feature representations (equivalently, learning a good distance metric), while also speeding up the test-time predictions. It is therefore related to both feature/distance-metric learning algorithms, as well as data summarization/compression algorithms for instance based methods.

In the specific context of kNN methods, there have been several previous efforts to speed up kNN's test-time predictions. The vast majority of these methods reduce to speeding up the retrieval of k nearest neighbors without modifying the training set. These include space partition algorithms such as ball-trees [6,30] and kd-trees [5], as well as approximate neighbor search like local-sensitive hashing [3,14]. Our paper addresses the problem from the perspective of data compression that reduces the size of the training set. Note that data compression approach is orthogonal to prior efforts on fast retrieval approaches, and thus these two methodologies could be combined.

Perhaps the most straightforward idea for data compression is subsampling the dataset. The seminal work in this area is Condensed Nearest Neighbors (CNN) proposed by [18]. It starts off with two sets, S and T, where S contains an instance of the training set and T contains the rest. CNN repeatedly scans T, looking for an instance in T that is misclassified using the data in S. This instance is then moved from T to S. This process continues until no more data movement can be made. Since this work, there have been several variants of CNN, including MCNN to address the order dependent issue of CNN [11], post-processing method [13], and fast CNN (FCNN) [4]. With these methods, the compressed training set is always a subset of the original training set, which is not necessarily a good representation. Recently, [24] introduce Stochastic Neighbor Compression (SNC), which learns a synthetic set as the compressed set. Assuming the synthetic set is presented as the *design variables*, SNC uses stochastic neighborhood [16,20,26] to model the probability of each training instance being correctly classified by the synthetic set. The synthetic set is obtained through numerical optimization, where the objective is to minimize the KL-divergence between the modeled distribution and the "perfect" distribution in which all training instances are correctly classified.

Among other works on summarizing/compressing massive data sets for machine learning problems includes methods such as coresets [1] for geometric problems (e.g., k-means/k-median clustering, nearest neighbor methods, etc). Kernel methods are also known to have the problem of having to store the entire training data in the memory and being slow at test time, and several methods such as landmarks based approximations [40,21] have been proposed to address these issues. However all these methods can only perform data compression by learning a set of representatives in the original feature space, and are not suited for data sets that are high-dimensional and exhibit significant nonlinearities.

All of the above methods operate on the original data space, not embracing the superior expressive power of deep learning. With unprecedented generalization performance, deep learning has achieved great success in various important applications, including speech recognition [19,17,29], natural language processing [28,27,9], image labeling [23,12,34,39], and object detection [15,36]. Recently, the kNN classifier

has been equipped with deep learning in modern face recognition systems, such as FaceNet [33]. In particular, kNN performs classification on the space mapped by a convolutional net [25]. However, Facenet trains the convolutional net to reflect the actual similarity between images/faces, rather than the accuracy performance of kNN. [32] introduce a method to train a deep neural net for kNN to perform well on the transformed space. Though inspired by this work, our DSNC is fundamentally different in that it not only optimizes the kNN performance but also simultaneously learns a compressed set in a new nonlinear feature space learned by a feedforward deep neural network.

5 Results

We present experimental results on seven benchmark datasets, including four from [24], *i.e.*, MNIST, YALEFACE, ISOLET, ADULT; and three additional, more complex datasets, *i.e.*, 20NEWS, CIFAR10 and CIFAR100. Some statistics are listed in Table 1. Since YALEFACE has no predefined test set, we report the average performance over 10 splits. All other results are reported on predefined test sets. We begin by describing the experimental settings, and then evaluate the test errors, compression ratios, feature representations, sensitivity to hyper-parameters and visualization of distributions of the test sets in the deep feature space. Our code is publicly available at `http://people.duke.edu/~ww107/`.

Table 1. Summary of datasets used in the evaluation.

Dataset	n	$\|Y\|$	d (d_L)	N_{max}
MNIST	60000	10	784 (164)	600
YALEFACE	1961	38	8064 (100)	100
ISOLET	3898	26	617 (172)	100
ADULT	32562	2	123 (50)	100
20NEWS	11314	20	2000 (100)	100
CIFAR10	50000	10	3072 (200)	100
CIFAR100	50000	100	3072 (200)	300

5.1 Experimental setting

To explore the advantages of our deep-learning-based method, we use raw features as the input for DSNC and the corresponding reference deep neural networks[2]. For MNIST, YALEFACE (rescaled to 48×42 pixels [37]), CIFAR10 and CIFAR100, we adopt convolutional neural networks, while ISOLET, ADULT and 20NEWS are fitted with feedforward neural networks. ReLU is adopted as the activation function after hidden layers for all models. Details of the network structures are shown in Table 2. When comparing

[2] The same network structure as DSNC except with a softmax-output.

Table 2. The feedforward neural network structure used for each dataset. 'Ck' ('Hk') indicates a convolutional (fully-connected) layer with k filters (hidden units). The variable d represents the dimensionality of the output feature representation of the inferred pseudo-inputs $\mathbf{Z}$.

Dataset	Network Structure
MNIST	C20-C50-Hd
YALEFACE	C20-C50-Hd
ISOLET	H500-H500-Hd
ADULT	H200-H200-Hd
20NEWS	H800-H800-Hd
CIFAR10	C64-C128-C256-C128-Hd
CIFAR100	C64-C128-C256-C128-Hd

the error with varying compressed ratios in Section 5.3, we fix d in Hd to be d_L in Table 1, and the time comparing the error with varying dimensions in Section 5.4, we keep the compared size m to be N_{max}.

DSNC is implemented using Torch7 [10] and trained on NVIDIA GTX TITAN graphics cards with 2688 cores and 6GB of global memory. We verify the implementation by numerical gradient checking, and optimize using stochastic gradient descent with RMSprop, using mini-batch in size of 100 . For all the datasets, we randomly select 20% of the training data for cross-validation of hyper-parameters λ_1 and λ_2 and early stopping. In contrast to SNC, our DSNC is not sensitive to γ. Thus we use a constant value 1 for all DSNC experiments set up.

With SNC we follow a similar setup to [24]. For ISOLET and MNIST, the dimensionality is reduced with LMMN as described in [38]. For YALEFACE, we follow [38] and first rescale the images to 48×42 pixels, then reduce the dimensionality with PCA, while omitting the leading five principal components which largely account for lighting variations. Finally we apply large margin nearest neighbor (LMNN) to reduce the dimensionality further to $d = 100$. For CIFAR10 and CIFAR100, we use LMNN to reduce the dimensionality to $d = 200$. In fact, the dimensionality of SNC is determined by LMNN. The parameters used for comparing the test error with varying compression rates and dimensionality are exactly the same as DSNC as we described before. Parameters are listed in Table 1. Notice that LMNN is used as the pre-processing step for all the methods except our DSNC and the corresponding reference networks.

5.2 Baselines

We experiment with two versions of DSNC, one uses the compressed data $\mathbf{Z}$ as the kNN reference during testing, denoted by $Compression$, the other uses the entire training data, denoted by ALL. We compare DSNC against the following related baselines, where the 1-nearest neighbor rule is adopted for all kNN methods.

- kNN without compression, with/without dimensionality reduction with LMNN;
- kNN using *Stochastic Neighbor Compression* (SNC) [24];
- Approximate kNN with *Locality-Sensitive Hashing* (LSH) [14,2];

– kNN using *CNN* [18] and *FCNN* [4] dataset compression;
– Deep neural network classifier with the same network structure as DSNC.

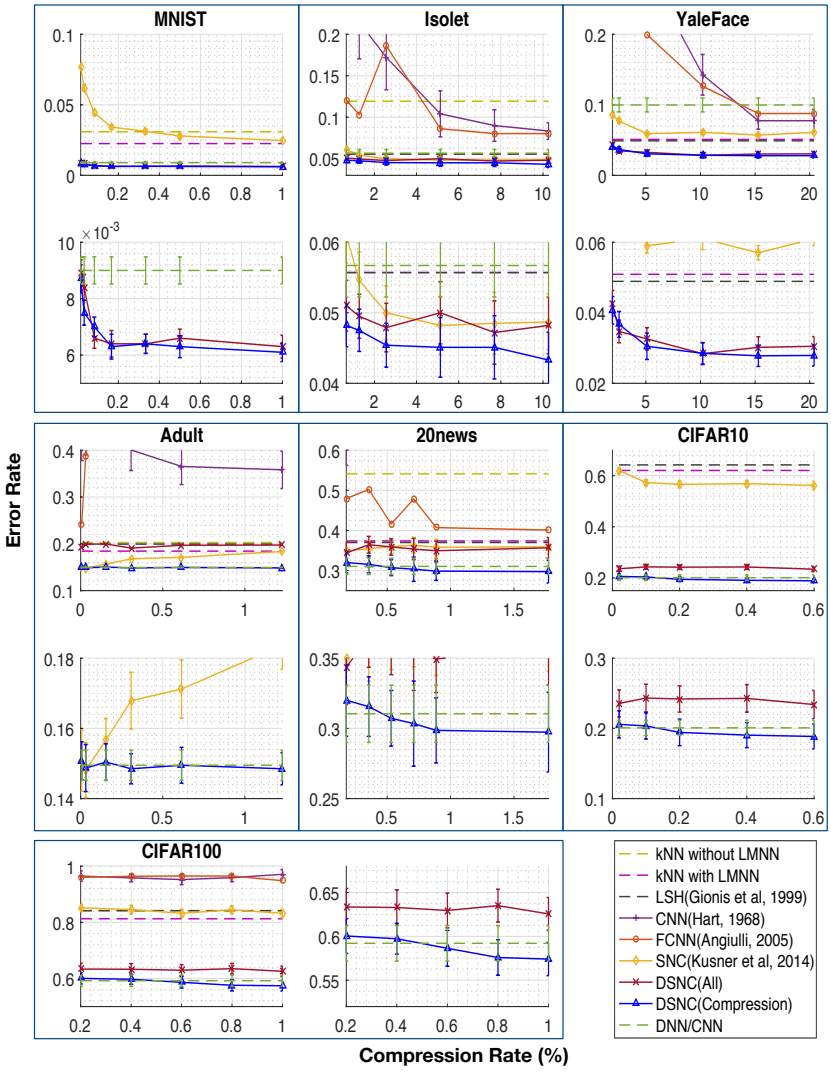

Fig. 2. Test error with varying dataset compression rates. The images below or on the right side in the blue rectangle are the zoom in images

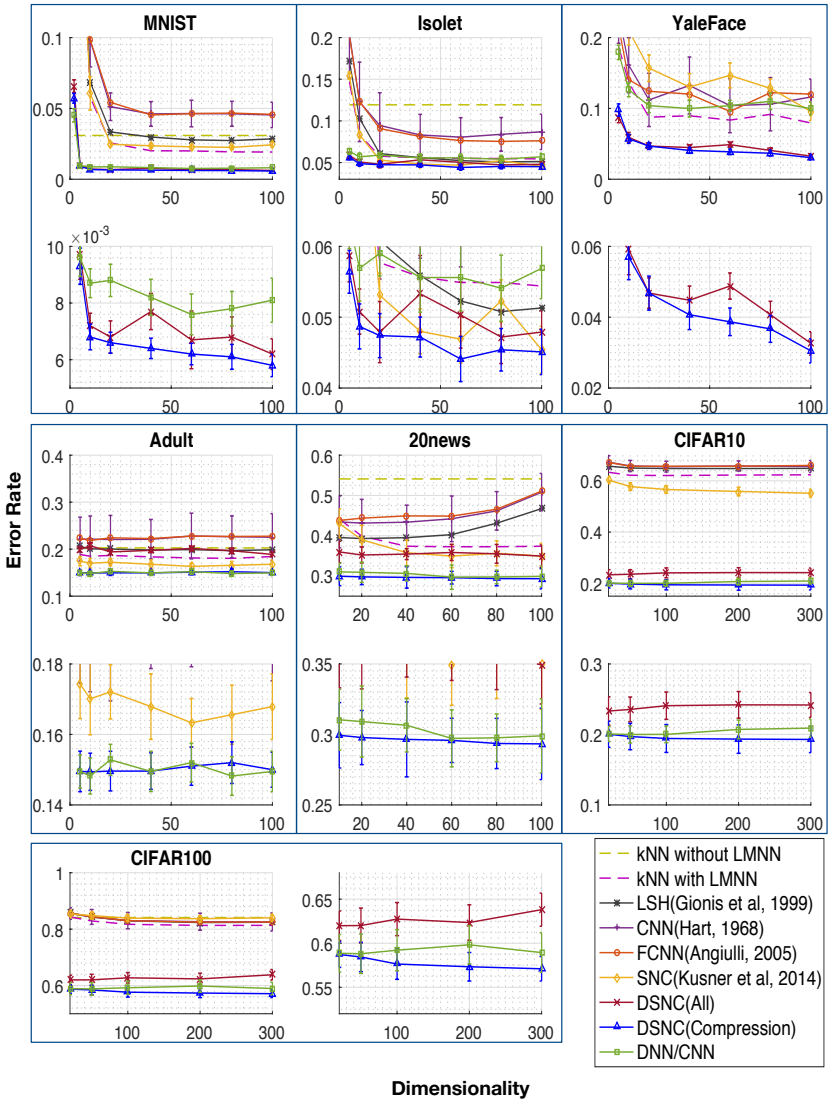

Fig. 3. Test error rates after mapping into different size of feature space. Zoom in images are organized the same as Figure 2

5.3 Errors with varying compression ratios

In this section we experiment with varying compressed ratio of the dataset, defined as the ratio between the compressed data size and the whole data size. The results are plotted in Figure 2. Several conclusions can be drawn from the results: 1) DSNC outperforms other methods on all data sets. The gap between DSNC and SNC is huge for

all the data sets, which indicates the advantage of learning the nonlinear feature space for data compression. 2) DSNC emerges as a stable compression method that is robust to the compression ratio. This is especially true when the compression ratio is small, for example, when the compression data size equals the number of classes ($m = |Y|$), DSNC still performs well, yielding significantly lower errors than LSH, CNN, FCNN and SNC. And, generally, with increasing m, test errors tend to decrease to a certain degree. 3) Compared with reference deep neural networks with softmax outputs, DSNC exhibits better performances on most datasets except ADULT, but with smaller gaps than the other methods. A possible reason could be that the task is binary classification and multi-modality within class distributions may be not that explicit in the dataset. It is notable when $m = |Y|$, DSNC degrades to the reference neural network using Euclidean distance as the metrics in softmax. We can see on 20NEWS, CIFAR10 and CIFAR100, the reference neural networks perform better. However, with an adaptive m, DSNC can always surpass the reference neural networks; while the observation on $YaleFace$ is particular surprising, as there is a big performance gap between DSNC and the corresponding convolutional neural networks. This indicates our motivation of learning a representative feature space for data compression to be effective, as DSNC has more degrees of freedom to adapt the compression data to a weak feature presentation.

5.4 Errors with varying feature dimensions

Next we investigate the impact of feature dimensions on the classification accuracy. To test the adaptive ability of DSNC to extremely low-dimensional feature spaces, we vary the feature space dimensions from 10 to 300 on CIFAR10 and CIFAR100, and from 10 to 100 on the other datasets. The results are plotted in Figure 3. We can see from the figure that the performance does not deteriorate when learning with a deep nonlinear transformation, *i.e.*, DSNC and DNN/CNN yield almost the same test errors with different feature dimensions on all the datasets, while other methods produce significantly worse performance when the feature dimension is low. Particularly, for MNIST and ISOLET, a 20 dimensional space is found to be powerful enough to express the dataset, while for CIFAR100, a nonlinear transformation into a 100-dimensional space obtains an accuracy that is close to the optimal performance. Interestingly, we also notice that using the compressed data outperforms the one using the entire mapped data. This is because our objective optimizes directly on the compressed data set, which can effectively filter out the noise in the original data set consisting of all the observations.

5.5 Sensitivity to hyper-parameters

In contrast to SNC, it is found that our model is not sensitive to the parameter γ in the stochastic neighborhood term. However, the hyper-parameters λ_1 and λ_2 do influence the performance of DSNC, because they control different behaviors of the objective. Specifically, λ_1 tends to pull the compressed data closer to the training sets in the deep feature space, while λ_2 pushes the compressed data with the same label to be far away from each other, such that they do not collapse into a single point and tend to capture the within-class multi-modality. We visualize the effects for these hyper-parameters by embedding the compressed data into 2-dimensional space using tSNE [26]. We use the

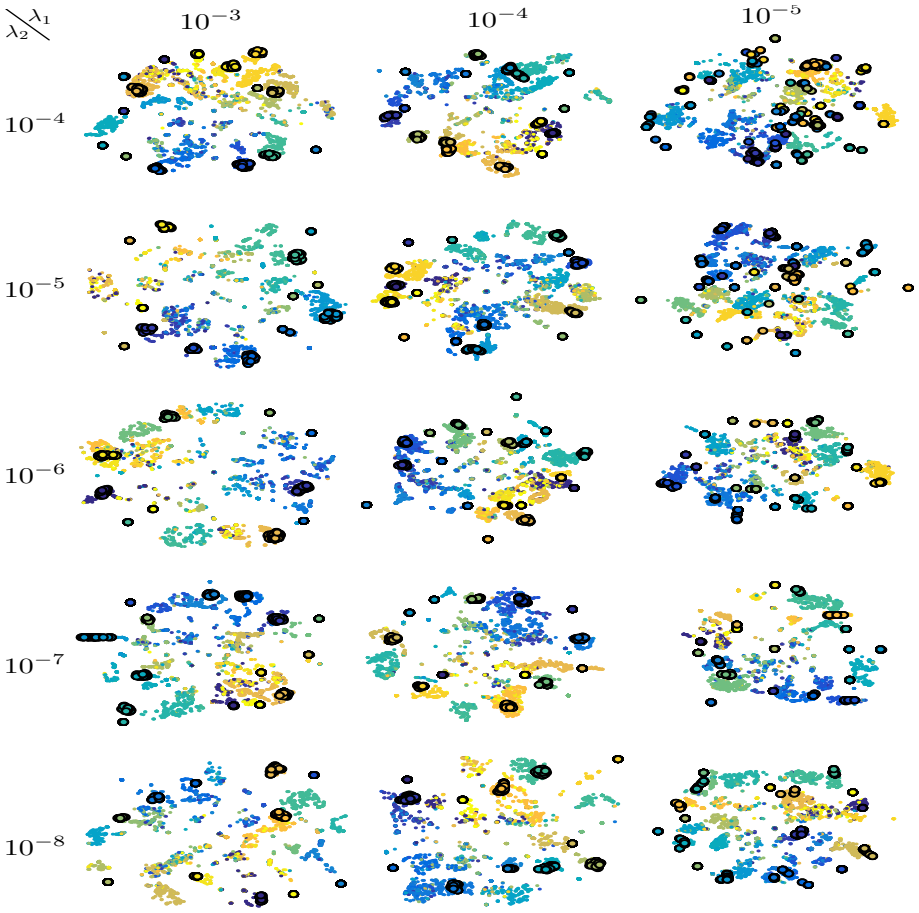

Fig. 4. tSNE visualization on 20NEWS with varying λ_1 and λ_2, compresseion size 500 (black circle), color indicates categories

20NEWS dataset for visualization in Figure 4. Consistent with our intuition, we find that with increasing λ_1, the compressed data tends to be condense, and far way from the training data; while increasing λ_2 generally pushes the compressed data in the same class to distribute more scatteringly. This indicates that if we want a larger compressed set of pseudo-inputs (i.e., m is large), a larger value of λ_2 should be set. The accuracies with different values of λ_1 and λ_2 are summarized in Table 3, which indicates suitable choices for λ_1 and λ_2 is essential for good performance.

5.6 Comparison of DSNC with SNC and SOFTMAX

In order to further understand the advantage of DSNC over SNC and softmax-based deep neural networks (SOFTMAX), we visualize them on MNIST. We adopt the same

Table 3. Test errors on 20news with varying the hyper-parameters λ_1 and λ_2 under the networks structure H800-H800-H100. The compressed size m fixes to be 100.

λ_2 \ λ_1	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
10^{-4}	30.36	28.75	30.48	32.80	34.10
10^{-5}	29.96	**28.47**	30.33	32.86	33.50
10^{-6}	30.80	29.27	29.77	33.55	32.70
10^{-7}	29.12	28.74	29.80	32.84	33.20
10^{-8}	29.02	28.54	30.68	33.86	33.27

models as the above experiments with a reference set consisting of $m = 100$ pseudo-inputs. This gives us cleaner results in the visualization. The inferred pseudo-inputs in the feature space are plotted in Figure 5. It can be clearly seen that DSNC is able to learn both separable feature space and representative data, whereas for the SNC, the compressed data does not seem to be separable. In terms of SOFTMAX, even though it can learn centered clusters, its tendency to only learn unimodal within-class distributions lead to poor performance around the decision boundary.

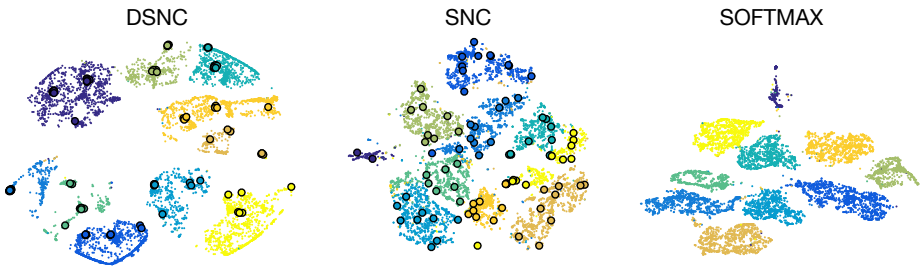

Fig. 5. Comparison of DSNC (left) with SNC (middle), SOFTMAX (right) on MNIST dataset. Circles represent the reference set.

6 Conclusion

We propose DSNC to jointly learn a deep feature space and a subset of compressed data that best represents the whole data. The algorithm consists of a deep neural network component for feature learning, on top of which an objective is proposed to optimize the kNN criteria, leading to a natural extension of the popular softmax-based deep neural networks. We test DSNC on a number of benchmark datasets, obtaining significantly improved performance compared to existing data compression algorithms.

Acknowledgments This research was supported in part by ARO, DARPA, DOE, NGA and ONR.

References

1. Agarwal, P.K., Har-Peled, S., Varadarajan, K.R.: Geometric approximation via coresets. Combinatorial and computational geometry 52, 1–30 (2005)
2. Aly, M., Munich, M., Perona, P.: Indexing in large scale image collections: Scaling properties and benchmark. In: Applications of Computer Vision (WACV), 2011 IEEE Workshop on. pp. 418–425. IEEE (2011)
3. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on. pp. 459–468. IEEE (2006)
4. Angiulli, F.: Fast condensed nearest neighbor rule. In: Proceedings of the 22nd international conference on Machine learning. pp. 25–32. ACM (2005)
5. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9), 509–517 (1975)
6. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In: Proceedings of the 23rd international conference on Machine learning. pp. 97–104. ACM (2006)
7. Bottou, L.: Stochastic gradient descent tricks. Tech. rep., Microsoft Research, Redmond, WA (2012)
8. Cayton, L.: Fast nearest neighbor retrieval for bregman divergences. In: Proceedings of the 25th international conference on Machine learning. pp. 112–119. ACM (2008)
9. Chen, W., Grangier, D., Auli, M.: Strategies for training large vocabulary neural language models. arXiv preprint arXiv:1512.04906 (2015)
10. Collobert, R., Kavukcuoglu, K., Farabet, C.: Torch7: A matlab-like environment for machine learning. In: BigLearn, NIPS Workshop. No. EPFL-CONF-192376 (2011)
11. Devi, V.S., Murty, M.N.: An incremental prototype set building technique. Pattern Recognition 35(2), 505–513 (2002)
12. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: Decaf: A deep convolutional activation feature for generic visual recognition. arXiv preprint arXiv:1310.1531 (2013)
13. Gates, G.: The reduced nearest neighbor rule. IEEE Transactions on Information Theory 18(3), 431–433 (1972)
14. Gionis, A., Indyk, P., Motwani, R., et al.: Similarity search in high dimensions via hashing. In: VLDB. vol. 99, pp. 518–529 (1999)
15. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR (2014)
16. Goldberger, J., Hinton, G.E., Roweis, S.T., Salakhutdinov, R.: Neighbourhood components analysis. In: Advances in neural information processing systems. pp. 513–520 (2004)
17. Graves, A., Mohamed, A.R., Hinton, G.: Speech recognition with deep recurrent neural networks. In: ICASSP (2013)
18. Hart, P.E.: The condensed nearest neighbor rule. Information Theory, IEEE Transactions on 14, 515–516 (1968)
19. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. Signal Processing Magazine, IEEE 29(6), 82–97 (2012)
20. Hinton, G.E., Roweis, S.T.: Stochastic neighbor embedding. In: Advances in neural information processing systems. pp. 833–840 (2002)
21. Hsieh, C.J., Si, S., Dhillon, I.S.: Fast prediction for large-scale kernel machines. In: Advances in Neural Information Processing Systems. pp. 3689–3697 (2014)

22. Hu, J., Lu, J., Tan, Y.P.: Discriminative deep metric learning for face verification in the wild. In: CVPR (2014)
23. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012)
24. Kusner, M., Tyree, S., Weinberger, K.Q., Agrawal, K.: Stochastic neighbor compression. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14). pp. 622–630 (2014)
25. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation 1(4), 541–551 (1989)
26. Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of Machine Learning Research 9(2579-2605), 85 (2008)
27. Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010. pp. 1045–1048 (2010)
28. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS (2013)
29. Mohamed, A.r., Sainath, T.N., Dahl, G., Ramabhadran, B., Hinton, G.E., Picheny, M.A.: Deep belief networks using discriminative features for phone recognition. In: ICASSP (2011)
30. Omohundro, S.M.: Five balltree construction algorithms. International Computer Science Institute Berkeley (1989)
31. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Tech. rep., DTIC Document (1985)
32. Salakhutdinov, R., Hinton, G.E.: Learning a nonlinear embedding by preserving class neighbourhood structure. In: International Conference on Artificial Intelligence and Statistics. pp. 412–419 (2007)
33. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. arXiv preprint arXiv:1503.03832 (2015)
34. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229 (2013)
35. Tieleman, T., Hinton, G.E.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. Tech. rep., Coursera: Neural Networks for Machine Learning (2012)
36. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. arXiv preprint arXiv:1411.4555 (2014)
37. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In: ICML (2009)
38. Weinberger, K.Q., Blitzer, J., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. In: Advances in neural information processing systems. pp. 1473–1480 (2005)
39. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: ECCV (2014)
40. Zhang, K., Tsang, I.W., Kwok, J.T.: Improved nyström low-rank approximation and error analysis. In: Proceedings of the 25th international conference on Machine learning. pp. 1232–1239. ACM (2008)