Joint Unsupervised Learning of Deep Representations and Image Clusters

Jianwei Yang, Devi Parikh, Dhruv Batra
Virginia Tech
{jw2yang, parikh, dbatra}@vt.edu

Abstract

In this paper, we propose a recurrent framework for joint unsupervised learning of deep representations and image clusters. In our framework, successive operations in a clustering algorithm are expressed as steps in a recurrent process*, stacked on top of representations output by a Convolutional Neural Network (CNN). During training, image clusters and representations are updated jointly: image clustering is conducted in the forward pass, while representation learning in the backward pass. Our key idea behind this framework is that good representations are beneficial to image clustering and clustering results provide supervisory signals to representation learning. By integrating two processes into a single model with a unified weighted triplet loss function and optimizing it end-to-end, we can obtain not only more powerful representations, but also more precise image clusters. Extensive experiments show that our method outperforms the state-of-the-art on image clustering across a variety of image datasets. Moreover, the learned representations generalize well when transferred to other tasks. The source code can be downloaded from* `https://github.com/jwyang/joint-unsupervised-learning`.

1. Introduction

We are witnessing an explosion in visual content. Significant recent advances in machine learning and computer vision, especially via deep neural networks, have relied on supervised learning and availability of copious annotated data. However, manually labelling data is a time-consuming, laborious, and often expensive process. In order to make better use of available unlabeled images, clustering and/or unsupervised learning is a promising direction.

In this work, we aim to address image clustering and representation learning on unlabeled images in a unified framework. It is a natural idea to leverage cluster ids of images as supervisory signals to learn representations and in turn the representations would be beneficial to image clustering. At a high-level view, given a collection of n_s unlabeled images $\boldsymbol{I} = \{I_1, ..., I_{n_s}\}$, the global objective function for learning image representations and clusters can be written as:

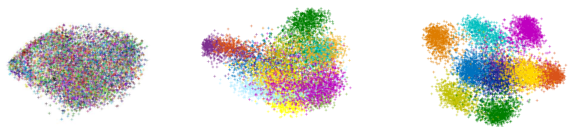
(a) Initial stage (b) Middle stage (c) Final stage

Figure 1: Clustering outputs for MNIST [32] test set at different stages of the proposed method. We conduct PCA on the image representations and then choose the first three dimensions for visualization. Different colors correspond to different clusters. Samples are grouped together gradually and more discriminative representations are obtained.

$$\underset{\boldsymbol{y},\boldsymbol{\theta}}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{y}, \boldsymbol{\theta}|\boldsymbol{I}) \tag{1}$$

where $\mathcal{L}(\cdot)$ is a loss function, $\boldsymbol{y}$ denotes the cluster ids for all images, and $\boldsymbol{\theta}$ denotes the parameters for representations. If we hold one in $\{\boldsymbol{y}, \boldsymbol{\theta}\}$ to be fixed, the optimization can be decomposed into two alternating steps:

$$\underset{\boldsymbol{y}}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{y}|\boldsymbol{I}, \boldsymbol{\theta}) \tag{2a}$$

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{I}, \boldsymbol{y}) \tag{2b}$$

Intuitively, (2a) can be cast as a conventional clustering problem based on fixed representations, while (2b) is a standard supervised representation learning process.

In this paper, we propose an approach that alternates between the two steps – updating the cluster ids given the current representation parameters and updating the representation parameters given the current clustering result. Specifically, we cluster images using agglomerative clustering[16] and represent images via activations of a Convolutional Neural Network (CNN).

The reason to choose agglomerative clustering is threefold: 1) it begins with an over-clustering, which is more reliable in the beginning when a good representation has not yet been learned. Intuitively, clustering with representations from a CNN initialized with random weights are not reliable, but nearest neighbors and over-clusterings are often acceptable; 2) These over-clusterings can be merged as better representations are learned; 3) Agglomerative clustering is a recurrent process and can naturally be interpreted in a recurrent framework.

Our final algorithm is farily intuitive. We start with an intial over-clustering, update CNN parameters (2b) using image cluster labels as supervisory signals, then merge clusters (2a) and iterate until we reach a stopping criterion. An outcome of the proposed framework is illustrated in Fig. 1. Initially, there are 1,762 clusters for MNIST test set (10k samples), and the representations (image intensities) are not that discriminative. After several iterations, we obtain 17 clusters and more discriminative representations. Finally, we obtain 10 clusters which are well-separated by the learned representations and interestingly correspond primarily to the groundtruth category labels in the dataset, even though the representation is learnt in an unsupervised manner. To summarize, the major contributions of our work are:

1 We propose a simple but effective end-to-end learning framework to jointly learn deep representations and image clusters from an unlabeled image set;
2 We formulate the joint learning in a recurrent framework, where merging operations of agglomerative clustering are expressed as a forward pass, and representation learning of CNN as a backward pass;
3 We derive *a single loss function* to guide agglomerative clustering and deep representation learning, which makes optimization over the two tasks seamless;
4 Our experimental results show that the proposed framework outperforms previous methods on image clustering and learns deep representations that can be transferred to other tasks and datasets.

2. Related Work

Clustering Clustering algorithms can be broadly categorized into hierarchical and partitional approaches [24]. Agglomerative clustering is a hierarchical clustering algorithm that begins with many small clusters, and then merges clusters gradually [12, 16, 30]. As for partitional clustering methods, the most well-known is K-means [36], which minimizes the sum of square errors between data points and their nearest cluster centers. Related ideas form the basis of a number of methods, such as expectation maximization (EM) [7, 37], spectral clustering [40, 47, 61], and non-negative matrix factorization (NMF) based clustering [1, 8, 60].

Deep Representation Learning Many works use raw image intensity or hand-crafted features [9, 18, 19, 23, 42, 50] combined with conventional clustering methods. Recently, representations learned using deep neural networks have presented significant improvements over hand-designed features on many computer vision tasks, such as image classification [29, 44, 46, 49], object detection [13, 14, 20, 43], etc. However, these approaches rely on supervised learning with large amounts of labeled data to learn rich representations. A number of works have focused on learning representations from unlabled image data. One class of approaches cater to reconstruction tasks, such as autoencoders [21, 28, 33, 41, 53], deep belief networks (DBN) [31], etc. Another group of techniques learn discriminative representations after fabricating supervisory signals for images, and then finetune them supervisedly for downstream applications [10, 11, 55]. Unlike our approach, the fabricated supervisory signal in these previous works is not updated during representation learning.

Combination A number of works have explored combining image clustering with representation learning. In [51], the authors proposed to learn a non-linear embedding of the undirected affinity graph using stacked autoencoder, and then ran K-means in the embedding space to obtain clusters. In [52], a deep semi-NMF model was used to factorize the input into multiple stacking factors which are initialized and updated layer by layer. Using the representations on the top layer, K-means was implemented to get the final results. Unlike our work, they do not jointly optimize for the representation learning and clustering.

To connect image clustering and representation learning more closely, [58] conducted image clustering and codebook learning iteratively. However, they learned codebook over SIFT feature [35], and did not learn deep representations. Instead of using hand-crafted features, Chen [2] used DBN to learn representations, and then conducted a nonparametric maximum margin clustering upon the outputs of DBN. Afterwards, they fine-tuned the top layer of DBN based on clustering results. A more recent work on jointly optimizing two tasks is found in [56], where the authors trained a task-specific deep architecture for clustering. The deep architecture is composed of sparse coding modules which can be jointly trained through back propagation from a cluster-oriented loss. However, they used sparse coding to extract representations for images, while we use a CNN. Instead of fixing the number of clusters to be the number of categories and predicted labels based on softmax outputs, we predict the labels using agglomerative clustering based on the learned representations. In our experiments we show that our approach outperforms [56].

3. Approach

3.1. Notation

We denote an image set with n_s images by $\boldsymbol{I} = \{I_1, ..., I_{n_s}\}$. The cluster labels for this image set are $\boldsymbol{y} = \{y_1, ..., y_{n_s}\}$. $\boldsymbol{\theta}$ are the CNN parameters, based on which we obtain deep representations $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_{n_s}\}$ from $\boldsymbol{I}$. Given the predicted image cluster labels, we or-

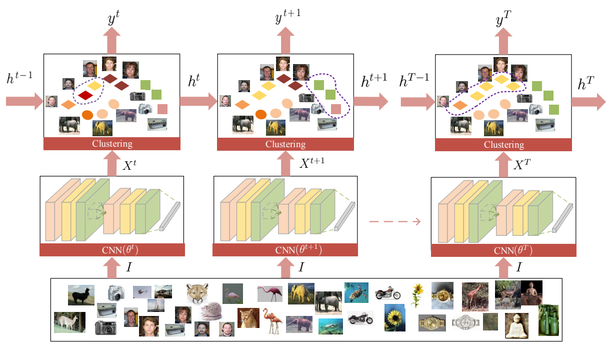

Figure 2: Proposed recurrent framework for unsupervised learning of deep representations and image clusters.

ganize them into n_c clusters $\boldsymbol{\mathcal{C}} = \{\mathcal{C}_1, ..., \mathcal{C}_{n_c}\}$, where $\mathcal{C}_i = \{\boldsymbol{x}_k | y_k = i, \forall k \in 1, ..., n_s\}$. $\mathcal{N}_i^{K_s}$ are the K_s nearest neighbours of $\boldsymbol{x}_i$, and $\mathcal{N}_{\mathcal{C}_i}^{K_c}$ is the set of K_c nearest neighbour clusters of $\mathcal{C}_i$. For convenience, we sort clusters in $\mathcal{N}_{\mathcal{C}_i}^{K_c}$ in descending order of affinity with $\mathcal{C}_i$ so that the nearest neighbour $\text{argmax}_{\mathcal{C} \in \boldsymbol{\mathcal{C}}^t} \mathcal{A}(\mathcal{C}_i, \mathcal{C})$ is the first entry $\mathcal{N}_{\mathcal{C}_i}^{K_c}[1]$. Here, $\mathcal{A}$ is a function to measure the affinity (or similarity) between two clusters. We add a superscript t to $\{\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\mathcal{C}}\}$ to refer to their states at timestep t. We use $\boldsymbol{\mathcal{Y}}$ to denote the sequence $\{\boldsymbol{y}^1, ..., \boldsymbol{y}^T\}$ with T timesteps.

3.2. Agglomerative Clustering

As background, we first briefly describe conventional agglomerative clustering [16, 30]. The core idea in agglomerative clustering is to merge two clusters at each step until some stopping conditions. Mathematically, it tries to find two clusters $\mathcal{C}_a$ and $\mathcal{C}_b$ by

$$\{\mathcal{C}_a, \mathcal{C}_b\} = \underset{\mathcal{C}_i, \mathcal{C}_j \in \boldsymbol{\mathcal{C}}, i \neq j}{\text{argmax}} \mathcal{A}(\mathcal{C}_i, \mathcal{C}_j) \tag{3}$$

There are many methods to compute the affinity between two clusters [16, 30, 38, 62, 64]. More details can be found in [24]. We now describe how the affinity is measured by $\mathcal{A}$ in our approach.

3.3. Affinity Measure

First, we build a directed graph $G =< \mathcal{V}, \mathcal{E} >$, where $\mathcal{V}$ is the set of vertices corresponding to deep representations $\boldsymbol{X}$ for $\boldsymbol{I}$, and $\mathcal{E}$ is the set of edges connecting vertices. We define an affinity matrix $\boldsymbol{W} \in \mathbb{R}^{n_s \times n_s}$ corresponding to the edge set. The weight from vertex $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$ is defined by

$$\boldsymbol{W}(i,j) = \begin{cases} exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2}{\sigma^2}), & \text{if } \boldsymbol{x}_j \in \mathcal{N}_i^{K_s} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $\sigma^2 = \frac{a}{n_s K_s} \sum_{\boldsymbol{x}_i \in \boldsymbol{X}} \sum_{\boldsymbol{x}_j \in \mathcal{N}_i^{K_s}} ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2$. This way to build up a directed graph can be found in many previous works such as [62, 64]. Here, a and K_s are two predefined parameters (their values are listed in Table 2). After constructing a directed graph for samples, we then adopt the graph degree linkage in [62] to measure the affinity between cluster $\mathcal{C}_i$ and $\mathcal{C}_j$, denoted by $\mathcal{A}(\mathcal{C}_i, \mathcal{C}_j)$.

3.4. A Recurrent Framework

Our key insight is that agglomerative clustering can be interpreted as a recurrent process in the sense that it merges clusters over multiple timesteps. Based on this insight, we propose a recurrent framework to combine the image clustering and representation learning processes.

As shown in Fig. 2, at the timestep t, images $\boldsymbol{I}$ are first fed into the CNN to get representations $\boldsymbol{X}^t$ and then used in conjunction with previous hidden state $\boldsymbol{h}^{t-1}$ to predict current hidden state $\boldsymbol{h}^t$, i.e, the image cluster labels at timestep t. In our context, the output at timestep t is $\boldsymbol{y}^t = \boldsymbol{h}^t$. Hence, at timestep t

$$\boldsymbol{X}^t = f_r(\boldsymbol{I}|\boldsymbol{\theta}^t) \tag{5a}$$

$$\boldsymbol{h}^t = f_m(\boldsymbol{X}^t, \boldsymbol{h}^{t-1}) \tag{5b}$$

$$\boldsymbol{y}^t = f_o(\boldsymbol{h}^t) = \boldsymbol{h}^t \tag{5c}$$

where f_r is a function to extract deep representations $\boldsymbol{X}^t$ for input $\boldsymbol{I}$ using the CNN parameterized by $\boldsymbol{\theta}^t$, and f_m is a merging process for generating $\boldsymbol{h}^t$ based on $\boldsymbol{X}^t$ and $\boldsymbol{h}^{t-1}$.

In a typical Recurrent Neural Network, one would unroll all timesteps at each training iteration. In our case, that would involve performing agglomerative clustering until we obtain the desired number of clusters, and then update the CNN parameters by back-propagation.

In this work, we introduce a *partial unrolling* strategy, *i.e.*, we split the overall T timesteps into multiple periods, and unroll one period at a time. The intuitive reason we unroll partially is that the representation of the CNN at the beginning is not reliable. We need to update CNN parameters to obtain more discriminative representations for the following merging processes. In each period, we merge a number of clusters and update CNN parameters for a fixed number of iterations at the end of the period. An extreme case would be one timestep per period, but it involves updating the CNN parameters too frequently and is thus time-consuming. Therefore, the number of timesteps per period (and thus the number of clusters merged per period) is determined by a parameter in our approach. We elaborate on this more in Sec. 3.6.

3.5. Objective Function

In our recurrent framework, we accumulate the losses from all timesteps, which is formulated as

$$\mathcal{L}(\{\boldsymbol{y}^1, ..., \boldsymbol{y}^T\}, \{\boldsymbol{\theta}^1, ..., \boldsymbol{\theta}^T\}|\boldsymbol{I}) = \sum_{t=1}^{T} \mathcal{L}^t(\boldsymbol{y}^t, \boldsymbol{\theta}^t | \boldsymbol{y}^{t-1}, \boldsymbol{I}) \tag{6}$$

Here, $\boldsymbol{y}^0$ takes each image as a cluster. At timestep t, we find two clusters to merge given $\boldsymbol{y}^{t-1}$. In conventional agglomerative clustering, the two clusters are determined by finding the maximal affinity over all pairs of clusters. In this paper, we introduce a criterion that considers not only the affinity between two clusters but also the local structure surrounding the clusters. Assume from $\boldsymbol{y}^{t-1}$ to $\boldsymbol{y}^t$, we merged a cluster $\mathcal{C}_i^t$ and its nearest neighbour. Then the loss at timestep t is a combination of negative affinities, that is,

$$\mathcal{L}^t(\boldsymbol{y}^t, \boldsymbol{\theta}^t | \boldsymbol{y}^{t-1}, \boldsymbol{I}) = -\boldsymbol{\mathcal{A}}(\mathcal{C}_i^t, \mathcal{N}_{\mathcal{C}_i^t}^{K_c}[1]) \tag{7a}$$

$$-\frac{\lambda}{(K_c-1)} \sum_{k=2}^{K_c} \left(\boldsymbol{\mathcal{A}}(\mathcal{C}_i^t, \mathcal{N}_{\mathcal{C}_i^t}^{K_c}[1]) - \boldsymbol{\mathcal{A}}(\mathcal{C}_i^t, \mathcal{N}_{\mathcal{C}_i^t}^{K_c}[k]) \right) \tag{7b}$$

where λ weighs (7a) and (7b). Note that $\boldsymbol{y}^t$, $\boldsymbol{y}^{t-1}$ and $\boldsymbol{\theta}^t$ are not explicitly presented at the right side, but they determine the loss via the image cluster labels and affinities among clusters. On the right side of the above equation, there are two terms: 1) (7a) measures the affinity between cluster $\mathcal{C}_i$ and its nearest neighbour, which follows conventional agglomerative clustering; 2) (7b) measures the difference between affinity of $\mathcal{C}_i$ to its nearest neighbour cluster and affinities of $\mathcal{C}_i$ to its other neighbour clusters. This term takes the local structure into account. See Sec. 3.5.1 for detailed explanation.

It is hard to simultaneously derive the optimal $\{\boldsymbol{y}^1, ..., \boldsymbol{y}^T\}$ and $\{\boldsymbol{\theta}^1, ..., \boldsymbol{\theta}^T\}$ that minimize the overall loss in Eq. (6). As aforementioned, we optimize iteratively in a recurrent process. We divide T timesteps into P partially unrolled periods. In each period, we fix $\boldsymbol{\theta}$ and search optimal $\boldsymbol{y}$ in the forward pass, and then in the backward pass we derive optimal $\boldsymbol{\theta}$ given the optimal $\boldsymbol{y}$. Details will be explained in the following sections.

3.5.1 Forward Pass

In forward pass of the p-th ($p \in \{1, ..., P\}$) partially unrolled period, we update the cluster labels with $\boldsymbol{\theta}$ fixed to $\boldsymbol{\theta}^p$, and the overall loss in period p is

$$\mathcal{L}^p(\boldsymbol{\mathcal{Y}}^p | \boldsymbol{\theta}^p, \boldsymbol{I}) = \sum_{t=t_p^s}^{t_p^e} \mathcal{L}^t(\boldsymbol{y}^t | \boldsymbol{\theta}^p, \boldsymbol{y}^{t-1}, \boldsymbol{I}) \tag{8}$$

where $\boldsymbol{\mathcal{Y}}^p$ is the sequence of image labels in period p, and $[t_p^s, t_p^e]$ is the corresponding timesteps in period p. For optimization, we follow a greedy search similar to conventional agglomerative clustering. Starting from the time step t_p^s, it finds one cluster and its nearest neighbour to merge so that $\mathcal{L}^t$ is minimized over all possible cluster pairs.

In Fig. 3, we present a toy example to explain the reason why we employ the term (7b). As shown, it is often the case that the clusters are densely populated in some regions while sparse in some other regions. In conventional agglomerative clustering, it will choose two clusters with largest affinity (or smallest loss) at each time no mater

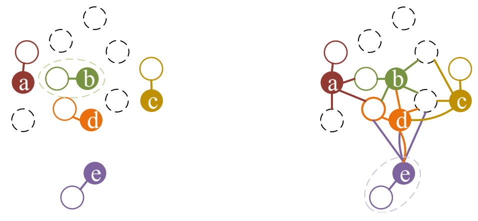

Figure 3: A toy illustration of (a) conventional agglomerative clustering strategy and (b) the proposed one. For simplification, we use a single circle to represent a cluster/sample. In conventional agglomerative clustering, node *b* and its nearest neighbour are chosen to merge because they are closest to each other; while node *e* is chosen in our proposed strategy considering the local structure.

where the clusters are located. In this specific case, it will choose cluster $\mathcal{C}_b$ and its nearest neighbour to merge. In contrast, as shown in Fig. 3(b), our algorithm by adding (7b) will find cluster $\mathcal{C}_e$, because it is not only close to it nearest neighbour, but also relatively far away from its other neighbours, i.e., the local structure is considered around one cluster. Another merit of introducing (7b) is that it will allow us to write the loss in terms of triplets as explained next.

3.5.2 Backward Pass

In forward pass of the p-th partially unrolled period, we have merged a number of clusters. Let the sequence of optimal image cluster labels be given by $\boldsymbol{\mathcal{Y}}_*^p = \{\boldsymbol{y}_*^t\}$, and clusters merged in forward pass are denoted by $\{[\mathcal{C}_*^t, \mathcal{N}_{\mathcal{C}_*^t}^{K_c}[1]]\}$, $t \in \{t_p^s, ..., t_p^e\}$. In the backward pass, we aim to derive the optimal $\boldsymbol{\theta}$ to minimize the losses generated in forward pass. Because the clustering in current period is conditioned on the clustering results of all previous periods, we accumulate the losses of all p periods, i.e.,

$$\mathcal{L}(\boldsymbol{\theta} | \{\boldsymbol{\mathcal{Y}}_*^1, ..., \boldsymbol{\mathcal{Y}}_*^p\}, \boldsymbol{I}) = \sum_{k=1}^{p} \mathcal{L}^k(\boldsymbol{\theta} | \boldsymbol{\mathcal{Y}}_*^k, \boldsymbol{I}) \tag{9}$$

Minimizing (9) w.r.t $\boldsymbol{\theta}$ leads to representation learning on $\boldsymbol{I}$ supervised by $\{\boldsymbol{\mathcal{Y}}_*^1, ..., \boldsymbol{\mathcal{Y}}_*^p\}$ or $\{\boldsymbol{y}_*^1, ..., \boldsymbol{y}_*^{t_p^e}\}$. Based on (7a) and (7b), the loss in Eq. 9 is reformulated to

$$-\frac{\lambda}{K_c-1} \sum_{t=1}^{t_p^e} \sum_{k=2}^{K_c} \left(\lambda' \boldsymbol{\mathcal{A}}(\mathcal{C}_*^t, \mathcal{N}_{\mathcal{C}_*^t}^{K_c}[1]) - \boldsymbol{\mathcal{A}}(\mathcal{C}_*^t, \mathcal{N}_{\mathcal{C}_*^t}^{K_c}[k]) \right) \tag{10}$$

where $\lambda' = (1 + 1/\lambda)$. (10) is a loss defined on clusters of points, which needs the entire dataset to estimate, making it difficult to use batch-based optimization. However, we show that this loss can be approximated by a sample-based loss, enabling us to compute unbiased estimators for the gradients using batch-statistics.

Algorithm 1 Joint Optimization on $\boldsymbol{y}$ and $\boldsymbol{\theta}$

```
Input:
    $\boldsymbol{I}$: = collection of image data;
    $n_c^*$: = target number of clusters;
Output:
    $\boldsymbol{y}^*, \boldsymbol{\theta}^*$: = final image labels and CNN parameters;
 1: $t \leftarrow 0; p \leftarrow 0$
 2: Initialize $\boldsymbol{\theta}$ and $\boldsymbol{y}$
 3: repeat
 4:     Update $\boldsymbol{y}^t$ to $\boldsymbol{y}^{t+1}$ by merging two clusters
 5:     if $t = t_p^e$ then
 6:         Update $\boldsymbol{\theta}^p$ to $\boldsymbol{\theta}^{p+1}$ by training CNN
 7:         $p \leftarrow (p+1)$
 8:     end if
 9:     $t \leftarrow t+1$
10: until Cluster number reaches $n_c^*$
11: $\boldsymbol{y}^* \leftarrow \boldsymbol{y}^t$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}^p$
```

The intuition behind reformulation of the loss is that agglomerative clustering starts with each datapoint as a cluster, and clusters at a higher level in the hierarchy are formed by merging lower level clusters. Thus, affinities between clusters can be expressed in terms of affinities between datapoints. We show in the supplement that the loss in (10) can be approximately reformulated as

$$\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}_*^{t_p^e}, \boldsymbol{I}) = -\frac{\lambda}{K_c - 1} \sum_{i,j,k} (\gamma \mathcal{A}(\boldsymbol{x}_i, \boldsymbol{x}_j) - \mathcal{A}(\boldsymbol{x}_i, \boldsymbol{x}_k)) \tag{11}$$

where γ is a weight whose value depends on λ' and how clusters are merged during the forward pass. $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are from the same cluster, while $\boldsymbol{x}_k$ is from the neighbouring clusters, and their cluster labels are merely determined by the final clustering result $\boldsymbol{y}_*^{t_p^e}$. To further simplify the optimization, we instead search $\boldsymbol{x}_k$ in at most K_c neighbour samples of x_i from other clusters in a training batch. Hence, the batch-wise optimization can be performed using conventional stochastic gradient descent method. Note that such triplet losses have appeared in other works [45, 54]. Because it is associated with a weight, we call (11) the weighted triplet loss.

3.6. Optimization

Given an image dataset with n_s samples, we assume the number of desired clusters n_c^* is given to us as is standard in clustering. Then we can build up a recurrent process with $T = n_s - n_c^*$ timesteps, starting by regarding each sample as a cluster. However, such initialization makes the optimization time-consuming, especially when datasets contain a large number of samples. To address this problem, we can first run a fast clustering algorithm to get the initial clusters. Here, we adopt the initialization algorithm proposed in [63] for fair comparison with their experiment results. Note that other kind of initializations can also be used, e.g. K-means. Based on the algorithm in [63], we obtain a number of clusters which contain a few samples for each (average is about 4 in our experiments). Given these initial clusters, our optimization algorithm learns deep representations and clusters. The algorithm is outlined in Alg. 1. In each partially unrolled period, we perform forward and backward passes to update $\boldsymbol{y}$ and $\boldsymbol{\theta}$, respectively. Specifically, in the forward pass, we merge two clusters at each timestep. In the backward pass, we run about 20 epochs to update $\boldsymbol{\theta}$, and the affinity matrix W is also updated based on the new representation. The duration of the p-th period is $n_p = ceil(\eta \times n_c^s)$ timesteps, where n_c^s is the number of clusters at the beginning of current period, and η is a parameter called *unrolling rate* to control the number of timesteps. The less η is, the more frequently we update $\boldsymbol{\theta}$.

4. Experiments

4.1. Image Clustering

We compare our approach with 12 clustering algorithms, including K-means [36], NJW spectral clustering (SC-NJW) [40], self-tuning spectral clustering (SC-ST)[61], large-scale spectral clustering (SC-LS) [3], agglomerative clustering with average linkage (AC-Link)[24], Zeta function based agglomerative clustering (AC-Zell) [64], graph degree linkage-based agglomerative clustering (AC-GDL) [62], agglomerative clustering via path integral (AC-PIC) [63], normalized cuts (N-Cuts) [47], locality preserving non-negative matrix factorization (NMF-LP) [1], NMF with deep model (NMF-D) [52], task-specific clustering with deep model (TSC-D) [56].

For evaluation, we use a commonly used metric: normalized mutual information (NMI) [59]. It ranges in $[0, 1]$. Larger value indicates more precise clustering results.

4.1.1 Datasets

We evaluate the clustering performance on two handwritten digit image datasets (MNIST [32] and USPS[1]), two multi-view object image datasets (COIL20 and COIL100 [39]), and four face image datasets (UMist [17], FRGC-v2.0[2], CMU-PIE [48], Youtube-Face (YTF)) [57]. The number of samples and categories, and image size are listed in Table 1. MNIST consists of training set (60,000) and testing set (10,000). To compare with different approaches, we experiment on the full set (MNIST-full) and testing set (MNIST-test), separately. For face image datasets such as UMist, CMU-PIE, we use the images provided as is without any changes. For FRGC-v2.0 and YTF datasets, we first

[1]`http://www.cs.nyu.edu/~roweis/data.html`
[2]`http://www3.nd.edu/~cvrl/CVRL/Data_Sets.html`

Table 1: Datasets used in our experiments.

Dataset	*MNIST*	*USPS*	*COIL20*	*COIL100*	*UMist*	*FRGC-v2.0*	*CMU-PIE*	*YTF*
#Samples	70000	11000	1440	7200	575	2462	2856	10000
#Categories	10	10	20	100	20	20	68	41
Image Size	28×28	16×16	128×128	128×128	112×92	32×32	32×32	55×55

Table 2: Hyper-parameters in our approach.

Hyper-parameter	K_s	a	K_c	λ	γ	η
Value	20	1.0	5	1.0	2.0	0.9 or 0.2

crop faces and then resize them to a constant size. In FRGC-v2.0 dataset, we randomly choose 20 subjects. As for YTF dataset, we choose the first 41 subjects which are sorted by their names in alphabet order.

4.1.2 Experimental Setup

All the hyper-parameters and their values for our approach are listed in Table 2. In our experiments, K_s is set to 20, the same value to [62]. a and λ are simply set to 1.0. We search the values of K_c and γ for best performance on MNIST-test set. The unrolling rate η for first four datasets is 0.9; and 0.2 for face datasets. The target cluster number n_c^* is set to be the number of categories in each dataset.

We use Caffe [26] to implement our approach. We stacked multiple combinations of convolutional layer, batch normalization layer, ReLU layer and pooling layer. For all the convolutional layers, the number of channels is 50, and filter size is 5×5 with stride = 1 and padding = 0. For pooling layer, its kernel size is 2 and stride is 2. To deal with varying image sizes across datasets, the number of stacked convolutional layers for each dataset is chosen so that the size of the output feature map is about 10×10. On the top of all CNNs, we append an inner product (*ip*) layer whose dimension is 160. *ip* layer is followed by a L2-normalization layer before being fed to the weighted triplet loss layer or used for clustering. For each partially unrolled period, the base learning rate is set to 0.01, momentum 0.9, and weight decay 5×10^{-5}. We use the *inverse* learning rate decay policy, with *Gamma*=0.0001 and *Power*=0.75. Stochastic gradient descent (SGD) is adopted for optimization.

4.1.3 Quantitative Comparison

We report NMI for different methods on various datasets. Results are averaged from 3 runs. We report the results by re-running the code released by original papers. For those that did not release the code, the corresponding results are borrowed from the papers. We find the results we obtain are somewhat different from the one reported in original papers. We suspect that these differences may be caused by the different experimental settings or the released code is changed from the one used in the original paper. For all test algorithms, we conduct L2-normalization on the image intensities since it empirically improves the clustering performance. We report our own results in two cases: 1) the straight-forward clustering results obtained when the recurrent process finish, denoted by OURS-SF; 2) the clustering results obtained by re-running clustering algorithm after obtaining the final representation, denoted by OURS-RC. The quantitative results are shown in Table 3. In the table cells, the value before '/' is obtained by re-running code while the value after '/' is that reported in previous papers.

As we can see from Table 3, both OURS-SF and OURS-RC outperform previous methods on all datasets with noticeable margin. Interestingly, we achieved perfect results (NMI = 1) on COIL20 and CMU-PIE datasets, which means that all samples in the same category are clustered into the same group. The agglomerative clustering algorithms, such as AC-Zell, AC-GDL and AC-PIC perform better than other algorithms generally. However, on MNIST-full test, they all perform poorly. The possible reason is that MNIST-full has 70k samples, and these methods cannot cope with such large-scale dataset when using image intensity as representation. However, this problem is addressed by our learned representation. We show that we achieved analogous performance on MNIST-full to MNIST-test set. In most cases, we can find OURS-RC performs better on datasets that have room for improvement. We believe the reason is that OURS-RC uses the final learned representation over the entire clustering process, while OURS-SF starts with image intensity, which indicates that the learned representation is more discriminative than image intensity. [3]

4.1.4 Generalization Across Clustering Algorithms

We now evaluate if the representations learned by our joint agglomerative clustering and representation learning approach generalize to other clustering techniques. We re-run all the clustering algorithms without any changes of parameters, but using our learned deep representations as features. The results are shown in Table 4. It can be seen that all

[3]We experimented with hand-crafted features such as HOG, LBP, spatial pyramid on a subset of the datasets with some of the better clustering algorithms from Table 3, and found that they performed worse than image intensity.

Table 3: Quantitative clustering performance (NMI) for different algorithms using image intensities as input.

Dataset	*COIL20*	*COIL100*	*USPS*	*MNIST-test*	*MNIST-full*	*UMist*	*FRGC*	*CMU-PIE*	*YTF*
K-means [36]	0.775	0.822	0.447	0.528	0.500	0.609	0.389	0.549	0.761
SC-NJW [40]	0.860/0.889	0.872/0.854	0.409/0.690	0.528/0.755	0.476	0.727	0.186	0.543	0.752
SC-ST [61]	0.673/0.895	0.706/0.858	0.342/0.726	0.445/0.756	0.416	0.611	0.431	0.581	0.620
SC-LS [49]	0.877	0.833	0.681	0.756	0.706	0.810	0.550	0.788	0.759
N-Cuts [47]	0.768/0.884	0.861/0.823	0.382/0.675	0.386/0.753	0.411	0.782	0.285	0.411	0.742
AC-Link [24]	0.512	0.711	0.579	0.662	0.686	0.643	0.168	0.545	0.738
AC-Zell [64]	0.954/0.911	0.963/0.913	0.774/0.799	0.810/0.768	0.017	0.755	0.351	0.910	0.733
AC-GDL [62]	0.945/0.937	0.954/0.929	0.854/0.824	0.864/0.844	0.017	0.755	0.351	0.934	0.622
AC-PIC [63]	0.950	0.964	0.840	0.853	0.017	0.750	0.415	0.902	0.697
NMF-LP [1]	0.720	0.783	0.435	0.467	0.452	0.560	0.346	0.491	0.720
NMF-D [52]	0.692	0.719	0.286	0.243	0.148	0.500	0.258	0.983/0.910	0.569
TSC-D [56]	-/0.928	-	-	-	-/0.651	-	-	-	-
OURS-SF	**1.000**	0.978	0.858	0.876	0.906	**0.880**	0.566	0.984	**0.848**
OURS-RC	**1.000**	**0.985**	**0.913**	**0.915**	**0.913**	0.877	**0.574**	**1.00**	**0.848**

Table 4: Quantitative clustering performance (NMI) for different algorithms using our learned representations as inputs.

Dataset	*COIL20*	*COIL100*	*USPS*	*MNIST-test*	*MNIST-full*	*UMist*	*FRGC*	*CMU-PIE*	*YTF*
K-means [36]	0.926	0.919	0.758	0.908	0.927	0.871	0.636	0.956	0.835
SC-NJW [40]	0.915	0.898	0.753	0.878	0.931	0.833	0.625	0.957	0.789
SC-ST [61]	0.959	0.922	0.741	0.911	0.906	0.847	**0.651**	0.938	0.741
SC-LS [49]	0.950	0.905	0.780	0.912	**0.932**	**0.879**	0.639	0.950	0.802
N-Cuts [47]	0.963	0.900	0.705	0.910	0.930	0.877	0.640	0.995	0.823
AC-Link [24]	0.896	0.884	0.783	0.901	0.918	0.872	0.621	0.990	0.803
AC-Zell [64]	**1.000**	0.989	0.910	0.893	0.919	0.870	0.551	**1.000**	0.821
AC-GDL [62]	**1.000**	0.985	0.913	**0.915**	0.913	0.870	0.574	**1.000**	**0.84**2
AC-PIC [63]	**1.000**	**0.990**	**0.914**	0.909	0.907	0.870	0.553	**1.000**	0.829
NMF-LP [1]	0.855	0.834	0.729	0.905	0.926	0.854	0.575	0.690	0.788

clustering algorithms obtain more precise image clusters by using our learned representation. Some algorithms like K-means, AC-Link that performed very poorly with raw intensities perform much better with our learned representations, and the variance in performance across all clustering algorithms is much lower. These results clearly demonstrate that our learned representation is not over-fitting to a single clustering algorithm, but generalizes well across various algorithms. Interestingly, using our learned representation, some of the clustering algorithms perform even better than AC-GDL we build on in our approach.

4.2. Transferring Learned Representation

4.2.1 Cross-Dataset Clustering

Table 5: NMI performance across COIL20 and COIL100.

Layer	*data*	top(*ip*)	top-1	top-2
COIL20 → COIL100	0.924	0.927	**0.939**	0.934
COIL100 → COIL20	0.944	0.949	**0.957**	0.951

Table 6: NMI performance across MNIST-test and USPS.

Layer	*data*	top(*ip*)	top-1	top-2
MNIST-test → USPS	0.874	0.892	0.907	**0.908**
USPS → MNIST-test	0.872	0.873	**0.886**	-

In this section, we study whether our learned representations generalize across datasets. We train a CNN based on our approach on one dataset, and then cluster images from another (but related) dataset using the image features extracted via the CNN. Specifically, we experiment on two dataset pairs: 1) multi-view object datasets (COIL20 and COIL100); 2) hand-written digit datasets (USPS and MNIST-test). We use the representation learned from one dataset to represent another dataset, followed by agglomerative clustering. Note that because the image sizes or channels are different across datasets, we resize the input images and/or expand the channels before feeding them to CNN. The experimental results are shown in Table 5 and 6. We use the representations from top *ip* layer and also the *convolutional* or *pooling* layers (top-1, top-2) close to top

Table 7: Face verification results on LFW.

#Samples	10k	20k	30k	50k	100k
Supervised	**0.737**	**0.746**	0.748	**0.764**	0.770
OURS	0.728	0.743	**0.750**	0.762	0.767

layer for image clustering. In two tables, compared with directly using raw image from the *data* layer, the clustering performance based on learned representations from all layers improve, which indicates that the learned representations can be transferred across these datasets. As perhaps expected, the performance on target datasets is worse compared to learning on the target dataset directly. For COIL20 and COIL100, a possible reason is that they have different image categories. As for MNIST and USPS, the performance beats OURS-SF, but worse than OURS-RC. We find transferring representation learned on MNIST-test to USPS gets close performance to OURS-RC learned on USPS.

4.2.2 Face Verification

We now evaluate the performance of our approach by applying it to face verification. In particular, the representation is learned on Youtube-Face dataset and evaluated on LFW dataset [22] under the restricted protocol. For training, we randomly choose about 10k, 20k, 30k, 50k, 100k samples from YTF dataset. All these subsets have 1446 categories. We implement our approach to train CNN model and cluster images on the training set. Then, we remove the L2-normalization layer and append a softmax layer to fine-tune our unsupervised CNN model *based on the predicted image cluster labels*. Using the same training samples and CNN architecture, we also train a CNN model with a softmax loss supervised by the groundtruth labels of the training set. According to the evaluation protocol in [22], we run 10-fold cross-validation. The cosine similarity is used to compute the similarity between samples. In each of 10 cross-validations, nine folds are used to find the optimal threshold, and the remaining one fold is used for evaluation. The average accuracy is reported in Table. 7. As shown, though no groundtruth labels are used for representation learning in our approach, we obtain analogous performance to the supervised learning approach. Our approach even (slightly) beats the supervised learning method in one case.

4.3. Image Classification

Recently, unsupervised representation learning methods are starting to achieve promising results for a variety of recognition tasks [4, 5, 25, 34]. We are interested in knowing whether the proposed method can also learn useful representation for image classification. We experiment with CIFAR-10 [27]. We follow the pipeline in [5], and base our experiments on their publicly available code. In this

Table 8: Image classification accuracy on CIFAR-10.

#Samples	K-means [5]	conv1	conv2	conv1&2
5k	62.81%	63.05%	63.10%	**63.50**%
10k	68.01%	68.30%	68.46%	**69.11**%
25k	74.01%	72.83%	72.93%	**75.11**%
50k (full set)	76.59%	74.68%	74.68%	**78.55**%

pipeline, codebook with 1600 codes is build upon 6×6 ZCA-whitened image patches, and then used to code the training and testing samples by extracting 1,600-d feature from each of 4 image quadrants. Afterwards, a linear SVM [6] is applied for image classification on 6,400-d feature. In our approach, the only difference is that we learn a new representation from 6×6 patches, and then use these new representations to build the codebook with 1,600 codes. The CNN architecture we use contains two convolutional layers, each of which is combined with a ReLu and a pooling layer, followed by an inner product layer. Both convolutional layers have 50 3×3 filters with pad = 1. The kernel size of pooling layer is 2, and the stride is 2. To save on training time, 40k randomly extracted patches are extracted from 50k training set and used in all the experiments.

Classification accuracies on test set with different settings are shown in Table 8. We vary the number of training samples and evaluate the performance for representations from different layers. As we can see, the combination of representations from the first and second convolutional layer achieve the best performance. We also use the representation output by inner product layer to learn the codebook. However, it performs poorly. A possible reason is that it discards spatial information of image patches, which may be important for learning a codebook. When using 400k randomly extracted patches to learn the codebook, [5] achieved 77.9%. However, it is still lower than what we achieved. This performance also beats several other methods listed in [4, 15, 25, 34].

5. Conclusion

In this paper, we have proposed an approach to jointly learn deep representations and image clusters. In our approach, we combined agglomerative clustering with CNNs and formulate them as a recurrent process. We used a partially unrolling strategy to divide the timesteps into multiple periods. In each period, we merged clusters step by step during the forward pass and learned representation in the backward pass, which are guided by a single weighted triplet-loss function. The extensive experiments on image clustering, deep representation transfer learning and image classification demonstrate that our approach can obtain more precise image clusters and discriminative representations that generalize well across many datasets and tasks.

References

[1] D. Cai, X. He, X. Wang, H. Bao, and J. Han. Locality preserving nonnegative matrix factorization. In *IJCAI*, volume 9, pages 1010–1015, 2009.

[2] G. Chen. Deep learning with nonparametric clustering. *arXiv preprint arXiv:1501.03084*, 2015.

[3] X. Chen and D. Cai. Large scale spectral clustering with landmark-based representation. In *AAAI*, 2011.

[4] A. Coates and A. Y. Ng. Selecting receptive fields in deep networks. In *NIPS*, pages 2528–2536, 2011.

[5] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*, pages 215–223, 2011.

[6] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[8] C. Ding, T. Li, M. Jordan, et al. Convex and semi-nonnegative matrix factorizations. *IEEE TPAMI*, 32(1):45–55, 2010.

[9] C. Doersch, A. Gupta, and A. A. Efros. Mid-level visual element discovery as discriminative mode seeking. In *NIPS*, pages 494–502, 2013.

[10] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, pages 1422–1430, 2015.

[11] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *NIPS*, pages 766–774, 2014.

[12] Y. Gdalyahu, D. Weinshall, and M. Werman. Self-organization in vision: stochastic clustering for image segmentation, perceptual grouping, and image database organization. *IEEE TPAMI*, 23(10):1053–1074, 2001.

[13] R. Girshick. Fast r-cnn. In *ICCV*, pages 1440–1448, 2015.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587. IEEE, 2014.

[15] I. J. Goodfellow, A. Courville, and Y. Bengio. Spike-and-slab sparse coding for unsupervised feature discovery. *arXiv preprint arXiv:1201.3382*, 2012.

[16] K. C. Gowda and G. Krishna. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern recognition*, 10(2):105–112, 1978.

[17] D. B. Graham and N. M. Allinson. Characterising virtual eigensignatures for general purpose face recognition. In *Face Recognition*, pages 446–456. Springer, 1998.

[18] D. Han and J. Kim. Unsupervised simultaneous orthogonal basis clustering feature selection. In *IEEE CVPR*, pages 5016–5023, 2015.

[19] B. Hariharan, J. Malik, and D. Ramanan. Discriminative decorrelation for clustering and classification. In *ECCV*, pages 459–472. Springer, 2012.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, pages 346–361. Springer, 2014.

[21] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[22] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[23] H.-C. Huang, Y.-Y. Chuang, and C.-S. Chen. Affinity aggregation for spectral clustering. In *CVPR*, pages 773–780. IEEE, 2012.

[24] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[25] Y. Jia, C. Huang, and T. Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. In *CVPR*, pages 3370–3377. IEEE, 2012.

[26] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[27] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

[28] A. Krizhevsky and G. E. Hinton. Using very deep autoencoders for content-based image retrieval. In *ESANN*. Citeseer, 2011.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[30] T. Kurita. An efficient agglomerative clustering algorithm using a heap. *Pattern Recognition*, 24(3):205–209, 1991.

[31] Q. V. Le. Building high-level features using large scale unsupervised learning. In *ICASSP*, pages 8595–8598. IEEE, 2013.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, pages 609–616. ACM, 2009.

[34] T.-H. Lin and H. Kung. Stable and efficient representation learning with nonnegativity constraints. In *ICML*, pages 1323–1331, 2014.

[35] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, volume 2, pages 1150–1157. IEEE, 1999.

[36] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[37] G. McLachlan and D. Peel. *Finite mixture models*. John Wiley & Sons, 2004.

[38] J. F. Navarro, C. S. Frenk, and S. D. White. A universal density profile from hierarchical clustering. *The Astrophysical Journal*, 490(2):493, 1997.

[39] S. A. Nene, S. K. Nayar, H. Murase, et al. Columbia object image library (coil-20). Technical report, Technical Report CUCS-005-96, 1996.

[40] A. Y. Ng, M. I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. *NIPS*, 2:849–856, 2002.

[41] M. A. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *CVPR*, pages 1–8. IEEE, 2007.

[42] K. Rematas, B. Fernando, F. Dellaert, and T. Tuytelaars. Dataset fingerprints: Exploring image collections through data mining. In *IEEE CVPR*, pages 4867–4875, 2015.

[43] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.

[44] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, pages 1–42, 2014.

[45] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, pages 815–823, 2015.

[46] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[47] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE TPAMI*, 22(8):888–905, 2000.

[48] T. Sim, S. Baker, and M. Bsat. The cmu pose, illumination, and expression (pie) database. In *FG*, pages 46–51. IEEE, 2002.

[49] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[50] S. Singh, A. Gupta, and A. Efros. Unsupervised discovery of mid-level discriminative patches. *ECCV*, pages 73–86, 2012.

[51] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu. Learning deep representations for graph clustering. In *AAAI*, pages 1293–1299, 2014.

[52] G. Trigeorgis, K. Bousmalis, S. Zafeiriou, and B. Schuller. A deep semi-nmf model for learning hidden representations. In *ICML*, pages 1692–1700, 2014.

[53] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pages 1096–1103. ACM, 2008.

[54] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. In *CVPR*, pages 1386–1393. IEEE, 2014.

[55] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, pages 2794–2802, 2015.

[56] Z. Wang, S. Chang, J. Zhou, and T. S. Huang. Learning a task-specific deep architecture for clustering. In *arXiv preprint arXiv:1509.00151*, 2015.

[57] L. Wolf, T. Hassner, and I. Maoz. Face recognition in unconstrained videos with matched background similarity. In *CVPR*, pages 529–534. IEEE, 2011.

[58] P. Xie and E. Xing. Integrating image clustering and codebook learning. In *AAAI*, 2015.

[59] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273. ACM, 2003.

[60] S. Zafeiriou and M. Petrou. Nonlinear non-negative component analysis algorithms. *IEEE TIP*, 19(4):1050–1066, 2010.

[61] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *NIPS*, pages 1601–1608, 2004.

[62] W. Zhang, X. Wang, D. Zhao, and X. Tang. Graph degree linkage: Agglomerative clustering on a directed graph. In *ECCV*, pages 428–441. Springer, 2012.

[63] W. Zhang, D. Zhao, and X. Wang. Agglomerative clustering via maximum incremental path integral. *Pattern Recognition*, 46(11):3056–3065, 2013.

[64] D. Zhao and X. Tang. Cyclizing clusters via zeta function of a graph. In *NIPS*, pages 1953–1960, 2009.