

---

# A Winner-Take-All Method for Training Sparse Convolutional Autoencoders

---

Alireza Makhzani, Brendan Frey  
Department of Electrical and Computer Engineering  
University of Toronto  
{makhzani, frey}@psi.toronto.edu

## Abstract

We explore combining the benefits of convolutional architectures and autoencoders for learning deep representations in an unsupervised manner. A major challenge is to achieve appropriate sparsity among hidden variables, since neighbouring variables in each feature map tend to be highly correlated and a suppression mechanism is therefore needed. Previously, deconvolutional networks and convolutional predictive sparse decomposition have been used to construct systems that have a recognition pathway and a data generation pathway that are trained so that they agree and so that the hidden representation is sparse. We take a more direct approach and describe a way to train convolutional autoencoders layer by layer, where in each layer sparsity is achieved using a winner-take-all activation function within each feature map. Learning is computationally efficient and we show that our method can be used to train shallow and deep convolutional autoencoders whose representations can be used to achieve classification rates on the MNIST, CIFAR-10 and NORB datasets that are competitive with the state of the art.

## 1 Introduction

Convolutional architectures have been used very successfully to advance the state of the art in discriminative classification [1], often by training them using new regularization techniques that have been developed in the deep learning community [2, 3]. Convolutional networks are well-suited to data where the same pattern or combination of patterns may occur at different locations or time points in the input space. For example, convolutional networks continue to perform better than non-convolutional architectures on the CIFAR-10 object classification task [1, 3] and new classification records are regularly achieved using convolutional architectures. In parallel with research on discriminative convolutional architectures, neural networks trained in an unsupervised fashion, such as stacked autoencoders, deep Boltzmann machines [4] and deep belief networks [5], have been developed and used to produce representations that have enabled leaps forward in classification accuracy for several tasks. While discriminative deep learning methods usually outperform these techniques, it is still widely recognized that unsupervised learning algorithms that can extract useful features are needed for solving problems with limited or weak label information, such as video recognition or pedestrian detection [6]. An advantage of unsupervised learning algorithms is the ability to use them in semi-supervised scenarios where the amount of labeled data is limited. Here, we explore how to incorporate the advantages of convolutional architectures within an unsupervised, autoencoder framework.

Recently, it has been observed that when representations are learnt in a way that encourages sparsity, improved performance is obtained on classification tasks. However, there are several practical problem with conventional sparse coding methods for large image sizes. First, it is well known that the images are smooth and highly structured, so that their variations cannot be captured by directly

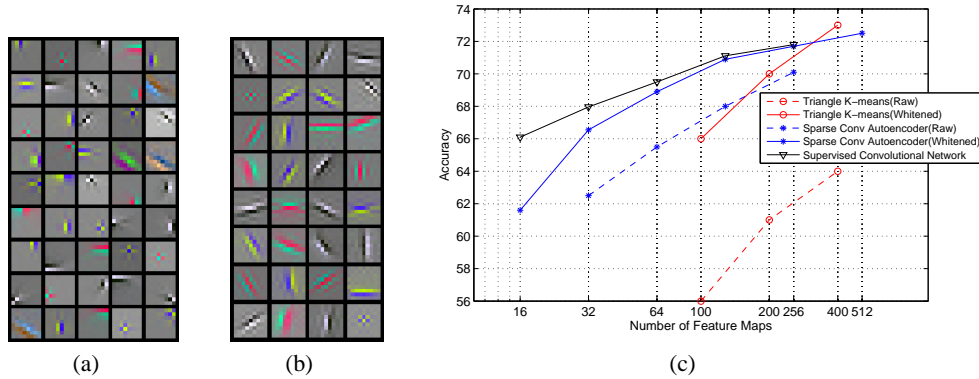


Figure 1: (a) Filters of a sparse autoencoder with 1000 hidden units, learnt from whitened CIFAR-10 random patches in isolation. (b) All the 32 filters learned by our sparse convolutional autoencoder from the entire CIFAR-10 images. (c) Classification rate of the features obtained by convolutional training on the entire image and patch-based training in isolation.

applying sparse coding. One way to solve this problem is to extract random image patches from input images and then train an unsupervised feature learning algorithm on these patches in isolation [7]. Once training is complete, the learnt filters can be used in a convolutional fashion to obtain representations of images. Fig. 1(a) shows the filters of a sparse non-convolutional autoencoder [8] with 1000 hidden units that was trained on CIFAR-10 image patches. Other unsupervised learning algorithms such as  $k$ -means clustering and sparse RBMs result in similar filters. As discussed in [7, 9], the main problem with this approach is that if the receptive field is small, this method will not capture relevant features (imagine the extreme of  $1 \times 1$  patches). Increasing the receptive field size is problematic, because then a very large number of features are needed to account for all the position-specific variations within the receptive field. For example, we see that in Fig. 1(a), the model allocates different filters to represent the same horizontal edge appearing at different locations within the receptive field. As a result, the learnt features are essentially shifted versions of each other, which results in redundancy between filters.

Unsupervised methods that make use of convolutional architectures can be used to address this problem, including convolutional RBMs [10], convolutional DBNs [11, 10], deconvolutional networks [12] and convolutional predictive sparse decomposition (PSD) [9, 6]. These methods learn features from the entire image in a convolutional fashion. In this setting, filters learn to collaborate with each other to reconstruct overlapping patches in the entire input image and thus the redundancy among the filters is reduced. The filters can focus on learning the shapes (*i.e.*, “what”), because the location information (*i.e.*, “where”) is encoded into feature maps.

In this paper, we propose an unsupervised learning algorithm for training convolutional autoencoders with a very strong sparsity constraint on the activity within each feature map. Our work is similar in spirit to deconvolutional networks [12] and convolutional PSD [9, 6], but whereas the approach in that work is to break apart the recognition pathway and data generation pathway, but learn them so that they are consistent, we describe a technique for directly learning a convolutional autoencoder. Fig. 1(b) shows the filters that were learnt in a convolutional fashion using our proposed sparse convolutional autoencoder, with just 32 filters and the same receptive field size as the patch-based model of Fig. 1(a). In this case, we can see the learnt filters are more diverse and global and only one filter is allocated for detecting each orientation.

We found that our proposed learning algorithm produces representations that can approach the classification rates achieved using supervised methods, and can exceed the state of the art in unsupervised learning when the number of feature maps is small. Fig. 1(c) compares the classification accuracy of the patch-based training approach versus training filters convolutionally on the entire image. In this figure, we plot the result of patch-based training using the  $k$ -means algorithm on the raw and whitened CIFAR-10 dataset as reported in [7]<sup>1</sup>. Other patch-based approaches such as

<sup>1</sup>[7] only reports the cross-validation performance. The test performance could be up to 2% more.

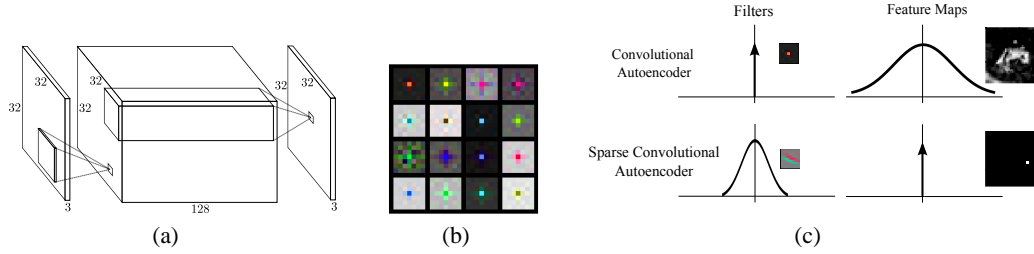


Figure 2: (a) Convolutional autoencoder that is trained on the entire image. (b) Filters of a non-regularized convolutional autoencoder, which uselessly copy the input image to the feature map. (c) The sparse convolutional autoencoder learns dense features with sparse feature maps.

those that use sparse autoencoders or RBMs have similar trends. We also plot the performance of our proposed shallow sparse convolutional autoencoder on this dataset. From this figure, we can see that, for example, if the number of feature maps is 128 we can get up to 12% improvement on the raw CIFAR-10 images, or by using 32 filters on the whitened CIFAR-10 we can do as well as the  $k$ -means method with 100 features. These results demonstrate that our method can capture the statistics within the receptive field much more efficiently when the number of feature maps is small. However, our results also suggest that both methods perform asymptotically similarly when a large number of feature maps is used (more than 500). We think this is because in this regime, both methods capture most of the statistics in the receptive field and thus the classification performance is the similar.

In Fig. 1(c), we also plot the performance of a shallow discriminative convolutional network that is trained in a purely supervised fashion and has the same architecture of our sparse convolutional autoencoder (*i.e.* number of feature maps, filter width/stride and pooling width/stride). Interestingly, we observe that our sparse convolutional autoencoder approaches the same accuracy when more than 32 feature maps are used. This shows that the unsupervised features of our shallow sparse convolutional autoencoder are as discriminative as the features of a supervised convolutional network with the same architecture.

We first describe our winner-take-all method and then we compare our technique with other approaches, including deconvolutional networks [12].

## 2 Description of the Algorithm

A shallow convolutional autoencoder maps an input vector  $\mathbf{x}$  to a set of feature maps in a convolutional fashion where feature map  $i$  is computed using  $\mathbf{z}_i = f(P_i * \mathbf{x} + b_i)$ . The encoder filters and biases are  $\{P_i, b_i\}$  and  $f$  is the activation function, *e.g.*, linear, sigmoidal or ReLU. We assume that the boundaries of the input image are zero-padded, so that each feature map has the size as the input. The hidden representation is then mapped linearly to the output using  $\hat{\mathbf{x}} = \sum_i (W_i * \mathbf{z}_i + b'_i)$  where  $W_i$  is  $i$ -th decoder or reconstruction filter. The parameters are optimized to minimize the mean square error,  $\|\hat{\mathbf{x}} - \mathbf{x}\|_2^2$ , summed over all training points.

Fig. 2(a) shows a diagram of a convolutional autoencoder for  $32 \times 32$  input images and 128 feature maps. Fig. 2(b) shows the filters of this autoencoder trained on CIFAR-10 images without using any regularization. We can see that in this case, both of the learnt encoder and decoder filters are essentially delta functions or color detectors located in the middle of the receptive field. These filters create a linear combination of the color channels in each feature map and then the decoder filters make a perfect reconstruction of the input using the feature maps. **These delta-features are useless for downstream applications and highlight the need for proper regularization, such as sparsity.**

### 2.1 Sparse convolutional autoencoders

Our sparse convolutional autoencoder is based on a convolutional autoencoder with a ReLU activation function. In the feedforward phase, after computing the feature maps  $\mathbf{z}_i = f(P_i * \mathbf{x} + b_i)$ , rather than reconstructing the input from all of the hidden units of the feature maps, we identify the

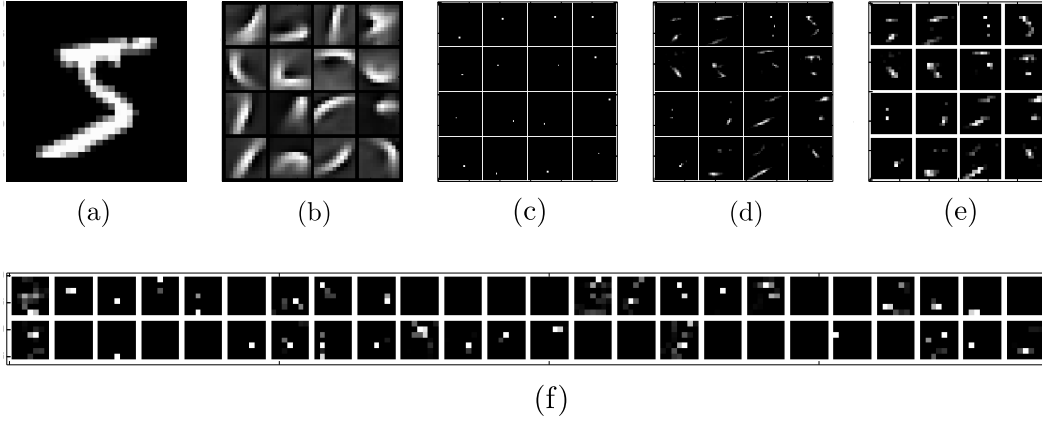


Figure 3: The sparse convolutional autoencoder with 16 first layer feature maps and 128 second layer feature maps trained on MNIST: (a) A MNIST training case as the input of the model. (b) Decoder weights of the first layer learnt on MNIST. (c) The 16 feature maps of the first layer for a particular input while training. (d) Feature maps of the first layer with ReLU function after turning off the sparsity constraint. (e) Feature maps of the first layer after overlapping max-pooling. (f) The 48 out of 128 feature maps of the second layer after max-pooling (final representation).

single largest hidden activity within each feature map, and set the rest of the activities as well as their derivatives to zero. This results in a sparse representation whose sparsity level is the number of feature maps. The decoder then reconstructs the output using only the active hidden units in the feature maps and the reconstruction error is only backpropagated through these hidden units as well. Note that once the largest activities are selected, the function computed by the network is linear. So at training time, the only non-linearity comes from the selection of the largest activities. This selection step acts as a regularizer that prevents the use of an overly large number of hidden units within each feature map, when reconstructing the input. Fig. 2(c) compares the filters and feature maps of a non-sparse convolutional autoencoder with the sparse convolutional autoencoder. In the sparse convolutional autoencoder, instead of learning delta functions as the filters, we force each feature map to be a delta function located at the maximum of the map, which result in much more useful encoder and decoder filters.

Once the weights are trained, the resulting filters may be used for learning to perform downstream classification tasks. Consistent with other representation learning approaches such as triangle  $k$ -means [7] and deconvolutional networks [13], we observed that using a softer sparsity constraint at the test time results in a better classification performance. So, in the sparse convolutional autoencoder, in order to find the final representation of the input image, we simply turn off the sparsity regularizer and thus the feature maps are computed as  $z_i = f(P_i * x + b_i)$  where  $f$  is the ReLU function. After that, we max-pool the feature maps and use this representation for classification tasks or in training deeper layers as discussed in the next section. Fig. 3 shows a sparse convolutional autoencoder that was trained on MNIST.

A problem that arises in almost all sparse coding algorithms, such as OMP, K-SVD [14] or even in  $k$ -means clustering, is the problem of dead atoms or empty clusters when learning the dictionary. This problem happens when we are enforcing strong sparsity, in which case the sparse coding algorithm greedily assigns a set of atoms to training cases in the first few epochs, without learning a decomposition of the input, and then in subsequent inference steps, those atoms will be picked and re-enforced and other atoms will not be adjusted. This problem can prevent sparse coding algorithms from scaling up to larger dictionary sizes. However, our approach ensures that every filter will be updated after visiting every training case, because we force all the feature maps to have exactly one active unit. In the largest model that we trained on CIFAR-10, our representation can achieve a sparsity level of 0.1% over one million hidden units. To the best of our knowledge achieving this sparsity level at this scale is not possible with any other method. One important consequence of this approach is that it avoids ‘dead filters’ that can’t be recovered during learning. To confirm this, we trained a model by setting all but the  $k$  largest hidden units across all the feature maps to zero,

during the feedforward phase. A delta-function filter was learned and the corresponding feature map took up almost all of the  $k$  active units for every input, while the rest of the filters and feature maps were dead. We also tried an experiment in which we relaxed the winner-take-all constraint to a ‘top- $n$ ’ constraint and we found that the model learns less sharp filters, resulting in slightly worse performance.

We found that tying the convolution and deconvolution weights hurt the generalization performance of our algorithm (data not shown). We think this is because the autoencoder is already very regularized by the strong sparsity constraint and tying the weights results in too much regularization. So, in the rest of the paper, we assume that the network has two different sets of recognition and reconstruction weights denoted by  $P$  and  $W$  respectively. The training algorithm of the sparse convolutional autoencoder is summarized as follows.

|   |
|---|
| <p><b>Sparse Convolutional Autoencoders:</b></p> <p><b>Training (Unsupervised):</b></p> <ol style="list-style-type: none"> <li>1) Perform the feedforward phase and compute <math display="block">\mathbf{z}_i = f(P_i * \mathbf{x} + b_i)</math> </li> <li>2) Find the largest activation within each feature map and set the rest of the hidden units in that feature map to zero. <math display="block">\mathbf{z}_{i(\Gamma_i)^c} = 0 \quad \text{where} \quad \Gamma_i = \arg \max(\mathbf{z}_i)</math> </li> <li>3) Compute the output and the error using the sparsified <math>\mathbf{z}_i</math>. <math display="block">\hat{\mathbf{x}} = \sum W_i * \mathbf{z}_i + b'_i</math> <math display="block">E = \ \mathbf{x} - \hat{\mathbf{x}}\ _2^2</math> </li> </ol> <p>3) Backpropagate the error only through the largest activations defined by <math>\Gamma_i</math> and iterate.</p> <p><b>Sparse Encoding (for supervised training or training the next layer):</b></p> <ol style="list-style-type: none"> <li>1) Turn off the sparsity constraint and compute the features using <math>\mathbf{z}_i = f(P_i * \mathbf{x} + b_i)</math>.</li> <li>2) Pool the features using overlapping max-pooling to find the final representation.</li> </ol> |
|---|

## 2.2 Deep sparse convolutional autoencoders

The sparse convolutional autoencoder can be used as a building block to form a hierarchy. In order to train the hierarchical model, we first train a sparse convolutional autoencoder and then, as explained above, we turn off the sparsity regularizer and use the ReLU activation function followed by max-pooling to obtain feature maps. We then fix the feature maps and train another sparse convolutional autoencoder to obtain the deep feature maps. This procedure is repeated to find multi-level representations.

It should be noted that in general, in unsupervised learning algorithms, the deep layers only marginally improve the classification performance as opposed to supervised learning where the deeper layers are much more helpful. We observed the same issue in deep sparse convolutional autoencoders as well. While shallow sparse convolutional autoencoders can achieve almost the same classification rate of a shallow supervised model with the same architecture (Fig. 1(c)), we found that in order to improve the classification rate using multiple levels, it is crucial to have many more dimensions (*i.e.* feature maps) in the deeper layers than the shallow layers, so as to enable linear separability. This is in contrast with the supervised case where the classification rate can be improved using deeper layers with the same dimensionality as the shallow layers. One reason for this difference could be that in the unsupervised case, deep layers are supposed to model the statistics of a larger spacial region in the input domain (*e.g.*, longer digit strokes in the case of MNIST) and thus the number of feature maps should increase as we go deeper, so as to account for combinations of parts. We will discuss the effect of using deeper architectures in the experiment section, but Fig. 3(f) shows the deep feature maps of a sparse convolutional autoencoder that was trained on MNIST.

## 3 Experiments

In this section, we evaluate the performance of sparse convolutional autoencoders (SCAs) on the MNIST, CIFAR-10 and NORB datasets using shallow and deep networks. (We will use the acronym SCA to refer to our method to save space, although we realize that there are other ways of training a sparse convolutional autoencoder.) Here, we only compare the performance of SCAs with other *unsupervised* feature learning methods. The state-of-the-art in supervised learning is generally better.



Figure 4: SCA decoder filters for 128 feature maps trained on non-whitened CIFAR-10 images.

While most of the conventional sparse coding algorithms require complex matrix operations such as matrix inversion or SVD decomposition, SCAs only require the *max* and *argmax* operations in addition to matrix multiplication which are all efficiently implemented in most GPU libraries. We used the publicly available *gnumpy* library [15] as well as Alex Krizhevsky’s *cuda-convnet* library [1]. The largest network that we studied was trained on CIFAR-10 and has 512 maps in the first layer and 4096 maps in the second layer. Training each layer took around 6 hours on a single Nvidia GTX 680 GPU.<sup>2</sup>

### 3.1 MNIST

For MNIST, the shallow SCA architecture has 128 filters with a filter width of 7 and a stride of 1. After training, we use max-pooling with a pooling stride of 3 and a pooling width of 5 to obtain the final  $128 \times 10 \times 10$  representation. Logistic regression is then applied to this representation for classification. In the deep architecture, we train another 2048 feature maps on top of the pooled feature maps of the first layer. Then the final representation is obtained by max pooling the 2048 feature maps with a pooling stride of 2 and a width of 3 to obtain the final  $128 \times 5 \times 5$  representation.

Highly competitive error rates of 0.64% and 0.48% are achieved using the shallow and deep architectures. To the best of our knowledge, this is the best classification rate obtained with a model trained in a purely unsupervised fashion for both shallow and deep architectures. Table 1 compares the performance of SCAs with other unsupervised convolutional feature learning algorithms and Fig. 3 shows a sparse convolutional autoencoder that was trained on MNIST.

|   | Error rate |
|---|------------|
| Shallow Deconvolutional Net with Differentiable Gaussian Pooling (16 maps) [13] | 1.38%      |
| Deep Deconvolutional Net with Differentiable Gaussian Pooling [13]              | 0.84%      |
| Deep Convolutional Belief Networks [16]   | 0.82%      |
| Ranzato et al. [17]   | 0.64%      |
| Shallow Sparse Convolutional Autoencoder, 16 maps                               | 1.15%      |
| Shallow Sparse Convolutional Autoencoder, 128 maps                              | 0.64%      |
| Deep Sparse Convolutional Autoencoder, 128 and 2048 maps                        | 0.48%      |

Table 1: Accuracy of unsupervised feature learning methods on MNIST.

### 3.2 CIFAR-10

On the CIFAR-10 dataset, we use global contrast normalization and ZCA whitening to preprocess the dataset as described in [7]. In the shallow architecture of our network, we use 512 feature maps, with a filter width of 9 and a stride of 1, followed by overlapping max-pooling with a stride of 4 and a width of 5, to obtain the final  $512 \times 8 \times 8$  representation. In the deep architecture, we train 4096 feature maps on top of the first layer and then use max pooling with a stride of 2 and a width of 3 to obtain the final  $4096 \times 4 \times 4$  representation. Fig. 4 and Fig. 1(b) show the filters learned from the raw and whitened images. Table 2 compares the performance of SCAs with other unsupervised feature learning methods, and shows that SCAs perform similarly to convolutional DBNs [18] and spike-and-slab sparse coding [19].

It was shown in [7] that the triangle *k*-means method can achieve classification rates up to 79.6% if a much larger number of feature maps, *e.g.*, 4000, is used in the first layer. In the largest network that

<sup>2</sup>Note that although the second layer has 8 times more feature maps, the computation time of both layers is still the same since the second layer is trained on the pooled feature maps.





Figure 5: Decoder filters of a sparse convolutional autoencoder with 32 filters NORB dataset.

we trained, we used 512 feature maps in the first layer, which achieved almost the same classification rate as the triangle  $k$ -means algorithm with the same number of feature maps. However, as we discussed in Section 1 and Fig. 1(c), considerable improvements over  $k$ -means can be achieved using SCAs when a small number of feature maps ( $< 500$ ) is used. Also, in our deep network, we used full connectivity between layers, but it was recently shown [20] that for triangle  $k$ -means, by learning the connectivity between layers using a similarity metric on the feature maps, a  $\sim 3\%$  gain in classification rate is achievable.

|   | Accuracy |
|---|----------|
| 3-Way Factored RBM (3 layers) [21]                                  | 65.3%    |
| Mean-covariance RBM (3 layers) [22]                                 | 71.0%    |
| Spike-and-Slab RBM [23]   | 76.7%    |
| Spike-and-Slab Sparse Coding [19]                                   | 78.3%    |
| Convolutional Deep Belief Net (2 layers) [18] ( <i>fine-tuned</i> ) | 78.9%    |
| Triangle $k$ -means (1600 maps) [7]                                 | 77.9%    |
| Shallow Sparse Convolutional Autoencoder, 512 maps                  | 72.7%    |
| Deep Sparse Convolutional Autoencoder, 512 and 4096 maps            | 78.1%    |

Table 2: Accuracy of unsupervised feature learning methods on CIFAR-10.

### 3.3 NORB

We also explored the small NORB normalized-uniform dataset, which contains 24,300 training and test examples with two views, each of size  $96 \times 96$  pixels. We take the inner  $64 \times 64$  pixels of each view and resize it to  $32 \times 32$  pixels. This size preserves the details of the image and also is small enough to allow fast training. We use global contrast normalization and ZCA whitening to preprocess the dataset as described in [7] and separate the two views to form a training set of 48,600 images. We learn a representation using this dataset and then at the test time, we concatenate the feature maps of both views to form the final representation of the binocular image.

In the shallow SCA architecture, we use 128 feature maps with a filter width of 9 and a stride of 1, and then use overlapping max pooling with a stride of 4 and a width of 6. In the deep architecture, we train 2048 feature maps on top of the first layer and then use max pooling with a stride 2 and a width of 3 to obtain the final representation. Fig. 5 shows the learnt dictionaries of SCAs trained on raw and whitened NORB images. In the whitened case, we can see that the sparse convolutional autoencoder allocates only one filter to each orientation as opposed to the non-convolutional feature learning methods. Table 3 compares the performance of SCAs with other unsupervised feature learning methods. SCAs perform significantly better than other deep learning methods, although the previously published triangle  $k$ -means result is significantly better still.

|  | Error rate |
|--|------------|
| Deep Boltzmann Machines [4]                              | 7.2%       |
| Third-order RBM [21]                                     | 6.5%       |
| Deep Belief Network [5]                                  | 5.0%       |
| Triangle $k$ -means [7] (4000 feature maps)              | 2.7%       |
| Shallow Sparse Convolutional Autoencoder, 128 maps       | 6.6%       |
| Deep Sparse Convolutional Autoencoder, 128 and 2048 maps | 4.7%       |

Table 3: Accuracy of unsupervised feature learning methods on NORB.

## 4 Discussion

**Filter redundancy.** After first conceiving of our method, we hypothesized that a major disadvantage would be that the winner-take-all constraint prevents the architecture from re-using the same feature multiple times in an image. If a particular feature is crucial for reconstructing different parts of the same input image, multiple filters containing the same feature will need to be learnt. We originally thought that this would prevent our method from achieving good classification accuracies, but the experimental results proved us wrong. The fact that the winner-take-all training method achieves state-of-the-art accuracies indicates that this hypothesis needs to be examined more closely. One advantage of the learning algorithm is that it is extremely fast, making it possible to learn large numbers of feature maps. Possibly, this compensates from the extremely sparse winner-take-all representation.

**Relationship to deconvolutional networks and convolutional PSD.** Deconvolutional networks [13] are top down models for generating images from a set of feature maps under a sparsity constraint. The image is reconstructed by first unpooling the sparse feature maps and then convolving them with a set of learnt filters. There is no direct way of mapping the image to the feature maps and inference of the sparse feature maps requires solving an optimization problem with the iterative ISTA algorithm, which is costly. Convolutional PSD [9] was introduced to address this problem. The generative model is still trained using the ISTA algorithm, but then a parameterized non-linear encoder or recognition network is separately trained to explicitly predict the sparse code using a soft thresholding operator.

Deconvolutional networks and convolutional PSD can be viewed as the generative and encoder paths of a convolutional autoencoder. Our contribution is to propose a specific winner-take-all approach to training a convolutional autoencoder, in which both paths are trained jointly. The main difference between our SCA and deconvolutional networks is that we directly impose sparsity using an  $\ell_0$  projection on the feature maps of a convolutional autoencoder which gives us a direct link from the image to the feature maps. In contrast to convolutional PSD, where this link is learnt separately to predict the solution of the ISTA algorithm, the encoder path of the SCA is trained and learnt jointly with the decoder part of the autoencoder to find the sparse feature maps. In this way, the costly operations of sparse recovery algorithms such as ISTA/FISTA is eliminated, yielding an algorithm that is much faster than both the deconvolutional network and convolutional PSD algorithms and just slightly slower than a standard convolutional autoencoder without any regularization. As a result of this speed-up, we can train much larger networks.

**Relationship to maxout networks.** When we initially described our approach to colleagues, they confused it with maxout [3], which is an activation function that performs a *max* operation over a set of linear units and represents the maximum of these units as the final activation. In SCAs, we also find the maximum over a set of linear units within each feature map during training. However, the main difference is that whereas the maxout activity is passed to the next layer using weights that are the same regardless of which unit gave the maximum, in SCAs, the winner-take-all feature maps retain the location information of all the units within the feature map and these units have different connectivity and effect on the subsequent layers.

## 5 Conclusion

We proposed a method for training sparse convolutional autoencoders by directly imposing a winner-take-all sparsity constraint on the feature maps. Unlike related approaches, such as deconvolutional networks and convolutional PSD, our method jointly trains the encoder and decoder paths and does not require an iterative optimization technique during learning. One concern with the winner-take-all approach is that features cannot be reused, but our experimental results show that this concern may not be too limiting. We found that adding more features improves classification, but also that training deep convolutional architectures achieves better results. We performed experiments on MNIST, CIFAR-10 and NORB datasets and showed that the classification rates of sparse convolutional autoencoders are competitive with the state-of-the-art results in unsupervised feature learning. Interestingly, the shallow sparse convolutional autoencoder achieves the same classification rate as a supervised convolutional network with the same architecture. However, our results suggest that better unsupervised learning principles may be needed to make the deep unsupervised algorithms work as well as the deep supervised algorithms.



## References

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *NIPS*, vol. 1, p. 4, 2012.
- [2] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.
- [3] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, “Maxout networks,” *ICML*, 2013.
- [4] R. Salakhutdinov and G. E. Hinton, “Deep boltzmann machines,” in *International Conference on Artificial Intelligence and Statistics*, pp. 448–455, 2009.
- [5] V. Nair and G. E. Hinton, “3d object recognition with deep belief nets,” in *NIPS*, pp. 1339–1347, 2009.
- [6] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, “Pedestrian detection with unsupervised multi-stage feature learning,” in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 3626–3633, IEEE, 2013.
- [7] 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*, 2011.
- [8] A. Makhzani and B. Frey, “k-sparse autoencoders,” *International Conference on Learning Representations, ICLR*, 2014.
- [9] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. LeCun, “Learning convolutional feature hierarchies for visual recognition,” in *NIPS*, vol. 1, p. 5, 2010.
- [10] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 609–616, ACM, 2009.
- [11] A. Krizhevsky, “Convolutional deep belief networks on cifar-10,” *Unpublished*, 2010.
- [12] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, “Deconvolutional networks,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2528–2535, IEEE, 2010.
- [13] M. D. Zeiler and R. Fergus, “Differentiable pooling for hierarchical feature learning,” *arXiv preprint arXiv:1207.0151*, 2012.
- [14] M. Aharon, M. Elad, and A. Bruckstein, “K-svd: Design of dictionaries for sparse representation,” *Proceedings of SPARS*, vol. 5, pp. 9–12, 2005.
- [15] T. Tieleman, “Gnumpy: an easy way to use gpu boards in python,” 2010.
- [16] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 609–616, ACM, 2009.
- [17] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. Lecun, “Unsupervised learning of invariant feature hierarchies with applications to object recognition,” in *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on*, pp. 1–8, IEEE, 2007.
- [18] A. Krizhevsky, “Convolutional deep belief networks on cifar-10,” 2010.
- [19] I. J. Goodfellow, A. Courville, and Y. Bengio, “Spike-and-slab sparse coding for unsupervised feature discovery,” *arXiv preprint arXiv:1201.3382*, 2012.
- [20] A. Coates and A. Y. Ng, “Selecting receptive fields in deep networks,” in *NIPS*, 2011.
- [21] A. Krizhevsky, G. E. Hinton, *et al.*, “Factored 3-way restricted boltzmann machines for modeling natural images,” in *International Conference on Artificial Intelligence and Statistics*, pp. 621–628, 2010.
- [22] M. Ranzato and G. E. Hinton, “Modeling pixel means and covariances using factorized third-order boltzmann machines,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2551–2558, IEEE, 2010.
- [23] Y. Bengio, A. C. Courville, and J. S. Bergstra, “Unsupervised models of images by spike-and-slab rbms,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1145–1152, 2011.