Autoconvolution for Unsupervised Feature Learning

Boris Knyazev *
Bauman Moscow State Technical University
bknyazev@bmstu.ru

Erhardt Barth
Institut für Neuro- und Bioinformatik
University of Lübeck
barth@inb.uni-luebeck.de

Thomas Martinetz
Institut für Neuro- und Bioinformatik
University of Lübeck
martinetz@inb.uni-luebeck.de

Abstract

In visual recognition tasks, supervised learning shows excellent performance. On the other hand, unsupervised learning exploits cheap unlabeled data and can help to solve the same tasks more efficiently. We show that the recursive autoconvolutional operator, adopted from physics, boosts existing unsupervised methods to learn more powerful filters. We use a well established multilayer convolutional network and train filters layer-wise. To build a stronger classifier, we design a very light committee of SVM models. The total number of trainable parameters is also greatly reduced by using shared filters in higher layers. We evaluate our networks on the MNIST, CIFAR-10 and STL-10 benchmarks and report several state of the art results among other unsupervised methods.

1 Introduction

Large-scale visual tasks can now be solved with big deep neural networks, if thousands of labeled samples are available and if training time is not an issue. Efficient GPU implementations of standard computational blocks make training and following usage feasible.

A major drawback of supervised neural networks is that they heavily rely on labeled data. It is true that in real applications it does not really matter which methods are used to achieve the desired outcome. But in some cases, labeling can be an expensive process. On the other side, natural data are full of abstract features unrelated to object classes. Unsupervised learning exploits abundant amounts of these cheap unlabeled data and can help to solve the same tasks more efficiently. In general, unsupervised learning is important to move towards artificial intelligence [1]. Another drawback of supervised methods and some recent unsupervised developments [2, 3, 4] is their excessive use of data augmentation. In this regard, unsupervised methods are shown to be able to learn these and more complex transformations, e.g., by "watching" videos [5].

In this work, we learn a visual representation model for image classification free from the aforementioned pitfalls. As a result, our work can potentially be easily applied to any natural data (e.g., audio). We improve on the previous works in which network filters (or weights) are learned layer-wise without label information [6, 7, 8, 9, 10, 11, 4, 12]. These methods are particularly suitable for the tasks with only few labeled training samples, such as STL-10 [7], as well as variants of MNIST [13] and CIFAR-10 [14] with smaller training sets. In addition, on full variants of these datasets we demonstrate that unsupervised learning is steadily becoming comparable with and even outperforms its supervised counterpart, including some convolutional neural networks (CNNs) trained by back-

*Alternative e-mail: borknyaz@gmail.com

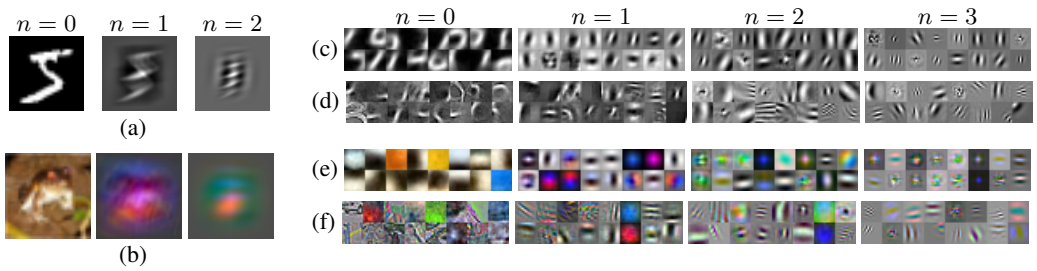

Figure 1: (a),(b): Recursive autoconvolution (2) of orders $n = 0, 1, 2$ applied to an MNIST (*a*) and CIFAR-10 (*b*) sample. (c)-(f): Examples of filters of size 13×13 learned using k-means with $n = 0, 1, 2, 3$ on the MNIST (*c,d*) and CIFAR-10 (*e,f*) patches with (*d,f*) and without (*c,e*) whitening.

propagation on thousands of labeled samples [15, 16]. This way, we provide further evidence that unsupervised learning is suitable for building efficient visual representations.

The main contribution of this work is adaptation of the recursive autoconvolution operator. Concretely, we demonstrate that this operator can be used to encourage existing clustering methods (e.g., k-means) or other learning methods (e.g., independent component analysis (ICA)) to train more powerful filters (Fig. 1) that resemble the ones learned by CNNs (Sections 3 and 4.1). Our second novelty is a method for building a committee of support vector machines (SVM) trained on several subsets of data projected by principal component analysis (PCA) (Section 4.6). Finally, we substantially reduce the total number of learned filters in higher layers of the network without loss of classification accuracy (Sections 4.4 and 5.3). In the end, we report several state of the art results among unsupervised methods while keeping computational cost relatively low (Section 5.3).

2 Related Work

Unsupervised learning is used quite often as an additional regularizer in the form of weights initialization [17] or reconstruction cost [18, 19], or as an independent visual model trained on still images [3, 14] or image sequences [5].

However, to a larger extent, our method is related to another series of works [6, 7, 8, 9, 10, 2, 12] and, in particular, [11, 4], in which filters (or some basis) are learned layer-wise and, contrary to the methods above, neither backpropagation, fine tuning nor data augmentation is used. The only exceptions are [4, 2], in which image transformations are used.

In these works, learning filters with clustering methods, such as k-means, is a standard approach [7, 2, 12, 4, 11]. For this reason, and to make comparison of our results easier, k-means is also adopted in this work as a default method. Moreover, clustering methods can learn overcomplete dictionaries without additional modifications, such as done for ICA in [10]. Nevertheless, since ICA [20] is also a common practice to learn filters, we conduct a couple of simple experiments with this method to probe our novel idea more thoroughly. In contrast to various coding schemes [6, 7, 8, 9, 12], popular in unsupervised learning, our forward pass is built upon a well established supervised method - a multilayer convolutional network [13] and its rectifying and pooling blocks.

Recently, this framework was successfully transfered to unsupervised learning in [11, 4]. In their works, as well as in our work, the forward pass is mostly kept standard, while methods to learn stronger (in terms of classification) filters are developing. For instance, in [11], k-means is enhanced by introducing convolutional clustering. Convolutional extension of clustering and coding methods is one of the ways to reduce redundancy in filters and improve classification, e.g., convolutional sparse coding [21]. In this work, we suggest another concept of making filters more powerful, namely, by recursive autoconvolution applied to image patches before unsupervised learning.

Autoconvolution (or self-convolution) and its properties seem to be first analyzed in physics (in spectroscopy [22]) and later in function optimization [23] as the problem of deautoconvolution arose. This operator also appeared in visual tasks to extract invariant patterns [24]. But, to the best of our knowledge, its recursive version, used as a pillar in this work, was first suggested in [25] for parametric description of images and temporal sequences. By contrast, we use this operator to extract patterns, which are then treated as convolution kernels in a multilayer CNN, which we refer to as an AutoCNN.

3 Autoconvolution

We first describe the routine for processing arbitrary discrete data based on autoconvolution. It is convenient to consider autoconvolution in the frequency domain. According to the convolution theorem, for N-dimensional discrete signals $\mathbf{X}$ and $\mathbf{Y}$, such as images ($N = 2$): $\mathcal{F}(\mathbf{X} * \mathbf{Y}) = k\mathcal{F}(\mathbf{X}) \circ \mathcal{F}(\mathbf{Y})$, where $\mathcal{F}$ - the N-dimensional forward discrete Fourier transform (DFT), $\circ$ - point-wise matrix product, k - a normalizing coefficient (which will be ignored further, since we apply normalization afterwards). Hence, autoconvolution is defined as

$$\mathbf{X} * \mathbf{X} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{X})^2), \tag{1}$$

where $\mathcal{F}^{-1}$ - the N-dimensional inverse DFT. Because of squared frequencies, phase information is mixed and the inverse operation becomes ill-posed. A lot of work has been devoted to that problem, e.g., [23]. In this work, we focus only on the forward operator.

To extract patterns from $\mathbf{X}$, it is necessary to make sure that $mean(\mathbf{X}) = 0$ and $std(\mathbf{X}) > 0$ before computing (1). Also, to compute linear autoconvolution, $\mathbf{X}$ is first zero-padded (however, for such images as in MNIST, this step might be optional). We also tried to work with autocorrelation defined as $\mathbf{X} \star \mathbf{X} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{X}) \circ conj[\mathcal{F}(\mathbf{X})]))$, but in our experience filters extracted with autoconvolution are more powerful.

3.1 Recursive Autoconvolution

We adopt the recursive autoconvoution operator, proposed earlier in [25] by simple extension of (1):

$$\mathbf{X}_n = \mathbf{X}_{n-1} * \mathbf{X}_{n-1} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{X}_{n-1})^2), \tag{2}$$

where $n = [0, n_{max}]$ - an index of the recursive iteration (or autoconvoution order). If $n = 0$, $\mathbf{X}_0$ equals the input, i.e. a raw image patch. For $n = 1$ expression (2) becomes equal (1). In our work, we limit $n_{\max} = 4$ as higher orders do not lead to better classification results.

In [25], image patterns extracted using this operator were used for parametric description of images. In this work, we use extracted patterns as convolution kernels, i.e. filters, because we noticed that applying (2) with $n > 1$ to images provides sparse wavelet like patterns, which are usually learned by a CNN in the first layer (see [16], Fig. 3 or [5], Fig. 6) or by other unsupervised learning methods, e.g. ICA [see 10, Fig. 1] or sparse coding [see 6, Fig. 2].

One of the issues with recursive autoconvoution is that the size of $\mathbf{X}$ is doubled after each iteration n due to zero-padding. To deal with that, we simply take the central part of the result or resize (subsample) it to its original size after each iteration (Fig. 1 (a),(b), where the second option is picked). We randomly choose one of these options to make the set of patches richer.

According to our statistics of extracted patches, autoconvolution order n is inversely proportional to the joint spatial σ_{xy} and frequency σ_{uv} resolution, i.e. $n \sim 1/(\sigma_{xy}\sigma_{uv})$, where $\sigma_{xy} = \sigma_x\sigma_y = \sqrt{D_1 D_2}$ and D_1, D_2 - are eigenvalues of the weighted covariance matrix of $\mathbf{X}$ in the spatial domain; analogously for σ_{uv}. Therefore, to cover a wider range of spatio-frequency properties, patches extracted with several orders are combined into one global set.

We are now ready to describe the architecture of a multilayer convolutional network (AutoCNN), which we used for image classification.

4 Autoconvolutional Multilayer Architecture

4.1 Learning Filters

The baseline method for unsupervised filter learning is chosen to be k-means as in [11, 4]. Specifically, for some layer l of an AutoCNN and for some training sample we have input $\mathbf{X}_{in}^{(l)} \in \mathbb{R}^{a_l \times a_l \times K_{l-1}}$, where K_{l-1} - the number of filters (or channels) in the previous layer. To learn filters for this layer l, we take random patches $\mathbf{X}^{(l)} \in \mathbb{R}^{s_l \times s_l \times d_l}$ from a subset of training samples and apply n-order recursive autoconvolution (2), where d_l - the depth of filters, which is $\leq K_{l-1}$ due to grouping described below in Section 4.4. Only squared inputs, patches and filters are considered for simplicity. To learn more powerful filters, we take results of several orders (e.g., in case $n = [0, 3]$ we have 4

patches instead of 1) and combine them into one global set of autoconvolutional patches. Note that in case $n = 0$, we extract more patches to make the total number of input data points for k-means about the same as for combinations of orders. We also experience with each of the orders independently to determine which orders contribute the most during classification (Fig. 1 (c)-(f)). In this global set, all patches are first scaled to have values in the range [0,1] , then they are ZCA-whitened as in [14, 7, 9, 11, 4]. For this set, k-means clustering (or another method, e.g., ICA) is applied, which produces a set of K_l data points (a dictionary) $\mathbf{D}^{(l)} \in \mathbb{R}^{s_l \times s_l \times d_l \times K_l}$. These data points are first l_2-normalized and then used as convolution kernels (filters) for layer l.

4.2 Preprocessing and Convolution

For image preprocessing we use standard techniques like ZCA-whitening and standardization [2] with few modifications. First, only colored datasets and only in the first layer are whitened, because for such clean grayscale images as in MNIST it is not reasonable. Second, in contrast to [11] or [7], where global or local normalization techniques are used, we use batch standardization (i.e. the entire batch is treated as a single vector) with a typical batch size of 125 to divide the dataset into equal batches. Additionally, since for higher layers we have groups of connected feature maps (as described below in Section 4.4), we standardize groups independently before convolutions. After preprocessing, inputs are convolved with filters with zero-padding, so that sizes of inputs and responses are equal.

4.3 Response Rectification, Normalization and Pooling

We experiment with two popular rectifiers which are applied to filter responses: $\max(0, \mathbf{x})$ (or ReLU), which is used by default (unless otherwise specified), and absolute values $|\mathbf{x}|$. Between layers we use local contrast normalization (LCN), as in [4]. Then, simple max-pooling over squared (of size m_l) disjoint spatial regions within a feature map is used. In addition, nonlinear "rootsift" normalization $(\text{sign}(\mathbf{x})\sqrt{|\mathbf{x}|/\|\mathbf{x}\|_2})$ applied after pooling turned out to be beneficial for colored datasets. This normalization as well as LCN are used only to report final classification results.

4.4 Selecting Connections between Layers

In CNNs trained with backpropagation, typically depth d_l of filters equals the number of filters in the previous layer, i.e. $d_l = K_{l-1}$. Alternatively, connections between feature maps and filters can also be sparse [13] or learned along with the network weights [11].

For unsupervised learning methods, such as k-means, it is hard to learn discriminative filters, if d_l is too high (e.g., 64 in [15]). Therefore, in this work, we follow the practice, established in unsupervised CNNs [11, 4], to divide feature maps into n_g relatively small groups, so that filters $\mathbf{D}^{(l)} \in \mathbb{R}^{s_l \times s_l \times d_l \times K_l}$ are learned independently for each group, where $d_l \ll K_{l-1}$. Convolutions are concatenated for all groups, so that responses of size $a_l/m_l \times a_l/m_l \times n_g K_l$ are forward passed to the next layer. For $n_g = 1$ this is a typical CNN architecture. Our contribution is that for $l > 1$, instead of treating groups independently, we learn filters for all groups altogether as described in more detail in Section 5.3.

Contrary to [11, 4], where feature groups are formed randomly or learned in a supervised fashion, we use the approach from [9]. Specifically, from all K_l feature maps random n_g maps are first chosen, for each of them the closest $(d_l - 1)$ features are then found. The similarity is computed as correlation between squared vectors $S(\mathbf{x}_j, \mathbf{x}_k) = \text{corr}(\mathbf{x}_j^2, \mathbf{x}_k^2)$ for some whitened feature maps $\mathbf{x}_j$ and $\mathbf{x}_k$. To improve connections, we repeat this heuristic procedure several times and select few variants of connections with the smallest total distance S. From these variants, we select the one with the maximum number of unique feature maps connected to the next layer. This way, we try to find a compromise between diversity of filters and their connectivity to each other, which is necessary to form invariant groups [10]. Our connection scheme is, in fact, incomplete. That is, only part of feature maps is propagated to higher layers, even if $n_g d_l > K_{l-1}$. But as we apply a multidictionary approach similarly to [11, 12], this is not an issue, because eventually features from all layers are concatenated and fed to a classifier.

[2] Along this work, vector $\mathbf{x}$ is considered standardized if its $mean(\mathbf{x}) = 0$ and $std(\mathbf{x}) = 1$.

4.5 Dimension Reduction and Classification

In [11], fully connected layers with dropout are trained on top of the unsupervised features. In this work, we first apply principal component analysis (PCA) together with whitening and then train a classifier on the projected data. For particularly large features (>30k and up to 300k in our experiments) random PCA [26] turned out to be extremely useful.

For classification, we use an SVM with the RBF kernel with a one-vs-one multiclass variant (unless otherwise specified). Feature vectors are standardized before classification. In all experiments, the SVM regularization constant was $C = 16$. The width of the RBF kernel was chosen to be $\gamma = 1/p_j$, where p_j is the dimensionality of the projected feature vector, the input of the SVM (see also the next section). For better final classification results for CIFAR-10 and STL-10, we also apply a one-vs-all method (we use a very efficient GPU implementation from [27]).

4.6 SVM Committee

A committee of models tends to give better classification results [4]. In this work, we build a committee of J models with just one set of filters, so that the models in the committee are determined only by their SVM. The filters, the PCA matrix and other parameters are fixed for all models. After the images were processed with a learned AutoCNN, PCA is performed once, and $p_j, j = 1, ..., J$ first principal components are chosen. Then for each j, an SVM is trained on the features projected on the p_j first principal components. After all iterations j are complete, the SVM scores within the committee are averaged.

5 Experiments

5.1 Experimental Setup

We evaluate our method on several image classification benchmarks: MNIST [13], CIFAR-10 [14] and STL-10 [7]. To demonstrate that unsupervised learning is particularly effective for datasets with few training samples, such as STL-10, which has only 100 images per class, we test our method on smaller versions of MNIST and CIFAR-10, namely MNIST (100), MNIST (300) and CIFAR-10 (400) with just 100, 300 and 400 images per class respectively, using the same experimental protocol as in previous works, e.g. [11, 3]: average results, and their standard deviations on the test set using 10 random subsets (folds) from the training set, are reported. For STL-10 these folds are predefined.

While in previous works all labeled training samples are typically used as unlabeled data during unsupervised learning, we found that it is enough to use at most 10k samples to learn filters and 20k - to perform PCA in all experiments, including STL-10, which contains 100k unlabeled samples.

We train models with one and two layers according to the proposed architecture (Section 4), which turned out to be sufficient to confirm effectiveness of our method. All reported classification results are average errors for MNIST and accuracies for all other datasets in percent, unless otherwise stated.

5.2 Model Parameters Validation

For model validation 3 random sets of 10k training and 10k test samples are drawn from the original training sets in case of MNIST and CIFAR-10, and 1k training and 1k test samples in case of STL-10. Average cross-validation results using the baseline model for 3 folds and among 3 different PCA dimensionalities ($p_j = 50, 100, 150$) are reported. The baseline model uses ReLU and max-pooling of size (m_1) 4 for MNIST, 8 for CIFAR-10 and 16 for STL-10.

First, for different filter sizes (s_1) we observe that autoconvolution (2) greatly improves classification, if neither filters nor input images are whitened (Fig. 2 (a)-(c)). We also show that in case of CIFAR-10 and STL-10 (Fig. 2 (b),(c)) whitening applied both to input images and filters considerably enhances results, which are shown in single markers for $n = 0$ and for several $n \geq 0$ only for the best s_1. Next, we conducted tests with whitening applied only to filters. It turned out to be an optimal scenario only for MNIST, and the results in this case are shown only for this dataset, similarly in single markers (Fig. 2 (a)). For CIFAR-10 and STL-10 this scenario is still more beneficial than completely without whitening, and the advantage of $n \geq 0$ over $n = 0$ in this case is much more noticeable than in case of whitening both. The number of filters in these experiments is fixed to $K_1 = 96$. Overall, the best

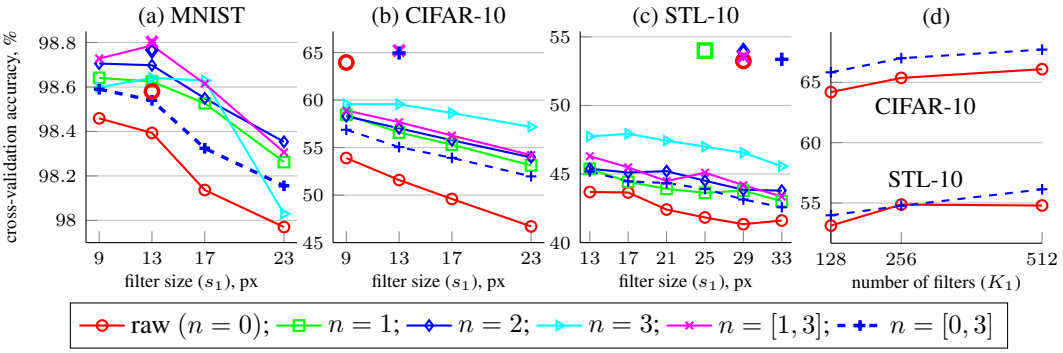

Figure 2: Comparison of results for a single layer baseline model learned with ($n \geq 0$) and without ($n = 0$) recursive autoconvolution, (a)-(c): without (solid and dashed lines) and with (in single markers for the best s_1 only) whitening, (d): with whitening applied both to input images and filters.

performance with recursive autoconvolution is superior than without it for a wide range of filter sizes, so, evidently, better filter size or preprocessing are not the reasons for our accuracy increase.

Given the obtained results (Fig. 2 (a)-(c)), for the next experiments we apply whitening both to images and filters for CIFAR-10 and STL-10 and whitening only to filters for MNIST. For CIFAR-10 and MNIST we set $s_1 = 13$ and for STL-10 $s_1 = 29$ px. Using these parameters, we estimated that the effect of applying recursive autoconvolution scales well with the number of filters (Fig. 2 (d)), although for STL-10 we observe some noise, probably, because we used too few folds and a rather small number of k-means iterations.

Nevertheless, our results without whitening and with whitening applied only to filters are very relevant in practice, because in some tasks whitening of inputs is difficult to be performed or not applicable [10], e.g., in case of large inputs we would need to compute a huge covariance matrix or in case of MNIST whitening is harmful.

By further cross-validation we tuned other model parameters. In detail, an error is decreased for MNIST by using $|\mathbf{x}|$ instead of ReLU (from 1.17% to 0.98%). In its turn, for CIFAR-10 and STL-10 a parametric rectifier $\max(0.25, \min(25, \mathbf{x}))$, "rootsift" normalization (see Section 4.3) and larger max-pooling ($m_1 = 20$ for STL-10 in case of one layer) altogether yield 0.5-1.5%. For all datasets, a committee of $J = 4 - 20$ SVM models (introduced in Section 4.6) in the ranges $p_j = [50, 400]$ for MNIST and $p_j = [30, 1500]$ for colored datasets gains another 0.05-0.10% and 1-3% respectively (Table 1). Using these tuning parameters we report our test results in Tables 1 and 2. Additionally to these parameters, in Table 3 results for CIFAR-10 and STL-10 are achieved with LCN, $|\mathbf{x}|$ for the second layer and a one-vs-all SVM, which together give about 1.5-2%.

5.3 Multilayer Performance

For a multilayer AutoCNN we determine if applying recursive autoconvolution is reasonable for higher layers (Table 1). It turned out, that for all datasets the results are consistently better with $n \geq 0$ either for the first or second layer and, in particular, for both layers. For all cases in these experiments all model settings are kept the same. The architectures in our experiments have short notations. For instance, "128c13-2p→32g-4ch-64c9-6p" denotes a two layer network with 128 filters of size $s_1 = 13$ and max-pooling of size $m_1 = 2$ in the first layer, and 64 filters of size $s_2 = 9$ and depth $d_2 = 4$ with $m_2 = 6$ in the second layer, whereas feature maps of the first layer are connected into $n_g = 32$ groups. Thus, in these experiments (Table 1), for MNIST, CIFAR-10 and STL-10 we design small (given that unsupervised models tend to be much larger) two layer networks: 128c13-2p→32g-2ch-64c9-2p, 128c13-2p→32g-2ch-64c11-4p and 96c29-4p→24g-2ch-64c17-6p respectively. In all experiments with 2 layer networks, for layer 1 we use combinations $n = [1, 3]$ for MNIST and $n = [0, 4]$ for others, while for layer 2 we found $n = [2, 3]$ to be optimal.

We next investigate connections from the first to the second layer. In previous works [11, 4], filters of higher layers are distinct for each group (learned independently), i.e. the total number of filters in layer 2 is defined as $K_2 n_g$, as described in Section 4.4. We discovered that using the same filters for all groups has no negative effect on classification accuracy, while it speeds up learning and

Table 1: Classification results on test sets with 2 layer networks with a single SVM/committee

Layer 1	Layer 2	MNIST(300)	CIFAR-10(400)	STL-10	max #filters (STL-10)
Raw	Raw	1.37 / 1.32	62.5 / 64.4	54.0 / 55.9	128 (112)
Autoconv	Raw	1.32 / 1.24	64.1 / 66.8	55.4 / 58.0	128 (112)
Raw	Autoconv	1.12 / 1.08	64.1 / 66.3	56.0 / 58.0	128 (112)
Autoconv	Autoconv	1.05 / 1.04	66.4 / 68.9	57.4 / 60.1	128 (112)
Autoconv	Autoconv+independ.	1.03 / 1.01	66.7 / 69.0	56.7 / 59.9	2112 (1584)
Autoconv	Autoconv+multidict.	1.04 / 1.01	68.3 / 70.5	59.9 / 62.7	192 (160)

significantly reduces the number of trainable parameters (Table 1). In this case, patches from all groups are concatenated before clustering and the total number of filters in layer 2 becomes equal K_2, i.e. the same filters are shared between all groups. To exploit features from both layers, the multidictionary approach is employed as in [11, 12]. First layer features are obtained according to a single layer architecture with a larger pooling size, so that sizes of the 1 and 2 layer feature maps are equal. Features of both layers are then combined into a large vector and passed to PCA and SVMs.

Finally, we evaluate our method on the test datasets (Tables 2 and 3, where the number of filters in dictionaries or convolutional layers, i.e. with filters larger than 1×1, is specified if known, the best results within a group of methods are in bold, parameters for STL-10 in parentheses). For MNIST, CIFAR-10 and STL-10 we increase the networks to 192c13-2p$\rightarrow$32g-3ch-64c9-2p, 1024c13-2p$\rightarrow$128g-4ch-160c11-4p and 1024c29-4p$\rightarrow$192g-4ch-160c17-8p respectively. However, at the same time, we prune the first layer features so that only the filters connected to layer 2 are used (in Tables 2 and 3, an approximate number of filters in layer 1 is indicated). For the full tests on MNIST and CIFAR-10 we report the average for 10 independent tests (as done, for instance, in [19]) and, in addition, the results of averaging the SVM scores of all $J \times 10$ models (denoted as "comb").

Table 2: Comparison of final classification errors on MNIST

Model	MNIST (100)	MNIST (comb)
1-2 Layer, unsupervised, no data augmentation		
Sparse coding (169) [6]	–	0.59
C-SVDDNet (400+multiscale+SIFT) [28]	–	**0.35**
CONV-WTA (128-2048) [29]	1.92	0.48
1 layer AutoCNN (256c13-4p)	1.97 ± 0.14	0.42 ± 0.02 (0.42)
2 layer AutoCNN (85-64)	$\mathbf{1.46 \pm 0.07}$	0.42 ± 0.03 (0.38)
Supervised and semi-supervised state of the art		
Ladder Network (full cost)[19]	$\mathbf{0.84 \pm 0.08}$	0.57 ± 0.02
Kernel CNN (12-400) [16]	2.05	**0.39**
Convolutional cluster. (96-1536) [11]	2.5	0.5
Stochastic pooling (64-64-64) [15]	~ 4.1	0.47

Notably, in most of the previous works, the results are very good either for simple grayscale images (MNIST) or more complex colored datasets (CIFAR-10, STL-10), or for smaller or larger datasets only. The only exception seems to be Ladder Networks [19], which are much larger and deeper than our models and use a supervised cost. Also, in [28] better results are achieved for MNIST with a single layer, but in our experience it can be quite easy to fine tune to such a simple task. Besides, we report the average for 10 tests, and for some runs our error was close to their results. For STL-10 comparable results are obtained in [12, 28], however our model has several times fewer features compared to [12], while in [28] the most important model component is handcrafted SIFT features, whereas we learn features from data. Our single layer model is especially effective for CIFAR-10, for which we obtain an accuracy better than in all previous multilayer unsupervised models (without data augmentation) and even better than a two layer CNN [16]. Our two layer network with a smaller total number of filters outperforms our single layer (except for the full tests on MNIST), which suggests the importance of depth in our case in the same way as in supervised CNNs. It is also superior than many other supervised and unsupervised models, including a large 3 layer CNN in [3] (in case of full data), which relies on excessive data augmentation and a 3 layer supervised CNN [15] based on an advanced pooling scheme. Other previous works, showing higher accuracies, are either fully

Table 3: Comparison of final classification accuracies on the test datasets

Model	CIFAR-10 (400)	CIFAR-10 (comb)	STL-10
1-3 Layer, unsupervised, no data augmentation			
Sparse coding/OMP (1600/6000) [8]	66.4 ± 0.8 [9]	81.5	59 ± 0.8
NOMP-20 (3200-6400-6400) [12]	72.2 ± 0.4	82.9	67.9 ± 0.6
C-SVDDNet (500+multiscale+SIFT) [28]	–	–	68.23 ± 0.5
Convolut. clustering, unsup. (96-1536) [11]	–	–	65.4
CONV-WTA (256-1024-4096) [29]	–	80.1	–
1 layer AutoCNN (1024c13(29)-8(20)p)	71.5 ± 0.2	83.2 ± 0.2 (83.3)	62.5 ± 0.6
1 layer ICA (1024c19(29)-8(20)p)	67.9 ± 0.3	81.1 ± 0.1 (81.3)	58.8 ± 0.7
1 layer AutoCNN-ICA (1024c19(29)-8(20)p)	71.5 ± 0.5	83.1 ± 0.1 (83.5)	61.9 ± 0.5
2 layer AutoCNN (400(500)-160)	$\mathbf{75.1 \pm 0.3}$	$\mathbf{85.0 \pm 0.2}$ **(85.4)**	$\mathbf{68.7 \pm 0.5}$
Supervised, semi-supervised or with data augmentation state of the art			
Committee of networks (300-5625) [4]	–	–	68.0 ± 0.55
View-invariant k-means (3 layers, 6400) [2]	72.6 ± 0.7	81.9	63.7
Exemplar-CNN (92-256-512) [3]	77.4 ± 0.2	84.3	$\mathbf{75.4 \pm 0.3}$
SWWAE (8-10 layers) [18]	–	**92.23**	74.33
Ladder Network (Γ-model, 7 layers) [19]	$\mathbf{79.60 \pm 0.47}$	–	–
Kernel CNN (12-800+50-800) [16]	–	82.18	62.32
Stochastic pooling (64-64-64) [15]	~ 65	84.87	–

or semi-supervised. In this work, we show competitive results both for simple and more complex datasets, as well as both for smaller and larger ones using the same model with few tuning parameters.

To further advocate for our method, we checked if recursive autoconvolution is able to improve other learning methods. For this purpose, we learned filters with ICA [20] on patches with $n = 0$ and $n \geq 0$ (AutoCNN-ICA) using the same procedure as with k-means (see Section 4.1). For CIFAR-10 we use larger filters since we were not able to learn overcomplete dictionaries. The gap between these two results is pronounced (Table 3), which strongly confirms the effectiveness of our method.

Note that the numbers of model filters presented in Tables 2 and 3 do not always reflect the total number of trainable parameters (N_{model}) nor the total computational cost. While our 2 layer model has seemingly many more filters than some CNNs, its maximum number of filter channels (d_l) is only 4, while in CNNs $d_l = K_{l-1}$ (i.e. 64-512). Thus, for convolutional layers of our large model for CIFAR-10 $N_{model} = 400 \times 13 \times 13 \times 3 + 160 \times 11 \times 11 \times 4 \approx 280$k, while in [15] it is ≈ 200k, which is smaller only because of smaller filters. Most other CNNs, presented in Tables 2 and 3, have many more parameters. Training our large two layer network on CIFAR-10 takes about 100 minutes on NVIDIA GTX 980 Ti in a Matlab implementation. Our smaller two layer network for MNIST is trained in about 25 minutes, while large single layer models can be trained in about 15 minutes on CIFAR-10 and in just 3-4 minutes on MNIST (including an SVM committee in all cases).

6 Conclusion

The importance of unsupervised learning in visual tasks is increasing and development is driven by the necessity to better exploit massive amounts of unlabeled data. We propose a novel unsupervised feature learning method and report superior results in several image classification tasks among the works not relying on data augmentation or supervised fine tuning. We adopt recursive autoconvolution and demonstrate its great utility for unsupervised learning methods, such as k-means and ICA. We argue that it can be integrated into other learning methods, including recently devised convolutional clustering, to boost their performance since recursive autoconvolution reveals complex image patterns. Furthermore, we significantly reduce the total number of trainable parameters by using shared filters and propose a simple method to build a committee of SVM models. As a result, the proposed autoconvolutional network performs better than most of the unsupervised, and several supervised, models in various classification tasks with only few but also with thousands of labeled samples.

Acknowledgments

This work is jointly supported by the German Academic Exchange Service (DAAD) and the Ministry of Education and Science of the Russian Federation (project number 3708).

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[2] Ka Y Hui. Direct modeling of complex invariances for visual object features. In *ICML*, 2013.

[3] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *NIPS*, pages 766–774, 2014.

[4] Bogdan Miclut. Committees of deep feedforward networks trained with few data. In *Pattern Recognition*, pages 736–742. Springer, 2014.

[5] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2015.

[6] Kai Labusch, Erhardt Barth, and Thomas Martinetz. Simple method for high-performance digit recognition based on sparse coding. *Neural Networks, IEEE Transactions on*, 19(11):1985–1989, 2008.

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

[8] Adam Coates and Andrew Y Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, pages 921–928, 2011.

[9] Adam Coates and Andrew Y Ng. Selecting receptive fields in deep networks. In *NIPS*, 2011.

[10] Quoc V Le, Alexandre Karpenko, Jiquan Ngiam, and Andrew Y Ng. ICA with reconstruction cost for efficient overcomplete feature learning. In *NIPS*, pages 1017–1025, 2011.

[11] Aysegul Dundar, Jonghoon Jin, and Eugenio Culurciello. Convolutional clustering for unsupervised learning. *arXiv preprint arXiv:1511.06241v2*, 2016.

[12] Tsung-Han Lin and HT Kung. Stable and efficient representation learning with nonnegativity constraints. *Journal of Machine Learning Research*, 2014.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[15] Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[16] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In *NIPS*, pages 2627–2635, 2014.

[17] Tom Le Paine, Pooya Khorrami, Wei Han, and Thomas S Huang. An analysis of unsupervised pre-training in light of recent advances. *arXiv preprint arXiv:1412.6597*, 2014.

[18] Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015.

[19] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *NIPS*, pages 3532–3540, 2015.

[20] Aapo Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *Neural Networks, IEEE Transactions on*, 10(3):626–634, 1999.

[21] Hilton Bristow, Anders Eriksson, and Simon Lucey. Fast convolutional sparse coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 391–398, 2013.

[22] V Dose, Th Fauster, and H-J Gossmann. The inversion of autoconvolution integrals. *Journal of Computational Physics*, 41(1):34–50, 1981.

[23] Rudolf Gorenflo and Bernd Hofmann. On autoconvolution and regularization. *Inverse Problems*, 1994.

[24] Janne Heikkilä. Multi-scale auto-convolution for affine invariant pattern recognition. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 1, pages 119–122. IEEE, 2002.

[25] BA Knyazev and VM Chernenkiy. Convolutional sparse coding for static and dynamic images analysis. *Science & Education of Bauman MSTU/Nauka i Obrazovanie of Bauman MSTU*, 1(11), 2014.

[26] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[27] Andrew Cotter, Nathan Srebro, and Joseph Keshet. A GPU-tailored approach for training kernelized SVMs. In *SIGKDD*, pages 805–813. ACM, 2011.

[28] Dong Wang and Xiaoyang Tan. Unsupervised feature learning with C-SVDDNet. *arXiv preprint arXiv:1412.7259*, 2014.

[29] Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. In *NIPS*, pages 2773–2781, 2015.

[30] John G Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *JOSA A*, 2(7):1160–1169, 1985.

[31] Andrea Vedaldi and Brian Fulkerson. VLFeat: An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1469–1472. ACM, 2010.

[32] Andrea Vedaldi and Karel Lenc. MatConvNet: Convolutional neural networks for matlab. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 689–692. ACM, 2015.

[33] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

Appendix

A.1 Comments to Section 3 (Autoconvolution)

In Fig. 3 (*left*), we show the images from Fig. 1 (a),(b) in their original sizes to illustrate that image size is doubled after each iteration of recursive autoconvolution (2). To collect patches of equal size, which is necessary for k-means and ICA, we take the central part of the result or resize (subsample) it to its original size after each iteration. Matlab demo code to compute recursive autoconvolution (2) is available at `https://github.com/bknyaz/nips2016`.

Next, as it is mentioned in Section 3.1, according to statistics of extracted patches, autoconvolution order n is inversely proportional to the joint spatial and frequency resolution, i.e. $n \sim 1/(\sigma_{xy}\sigma_{uv})$ (Fig. 3 (*right*)). This implies that recursive autoconvolution provides sparse wavelet-like filters, which seems to be very important for classification, because learning filters with $n > 0$ significantly improves classification results. However, it is also beneficial to have few filters with a very large joint resolution (e.g., $\sigma_{xy}\sigma_{uv} > 0.1$) typically learned with $n = 0$, because combinations of orders $n = [0, 3]$ and $n = [0, 4]$ show the best classification performance on CIFAR-10 and STL-10 (Fig. 7 (b),(c)). Examples of learned sparse and dense filters are visualized in Fig. 1 (c)-(f) and Fig. 4. Notice, that values of $\sigma_{xy}\sigma_{uv}$ are constrained by the theoretical limit [see 30, expression (2)], namely, $\sigma_{xy}\sigma_{uv} \geq 1/(16\pi^2)$, although in practice, due to discretization this constraint can be violated.

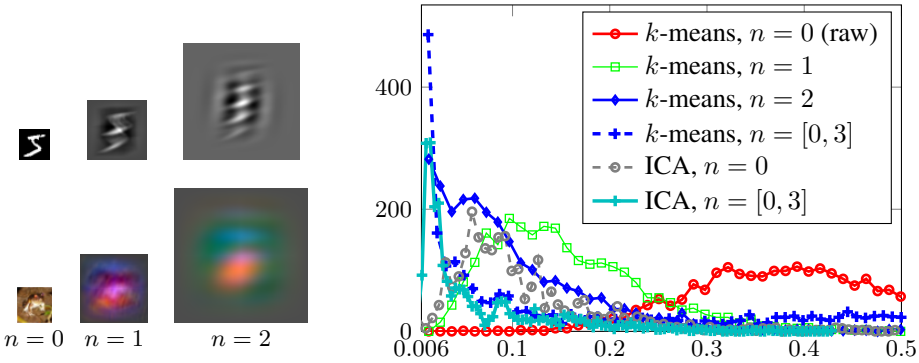

Figure 3: *(left)*: In addition to Fig. 1 (a),(b), we show the results of applying recursive autoconvolution of orders $n = 0, 1, 2$ to an MNIST (*top*) and CIFAR-10 (*bottom*) sample in original sizes. *(right)*: Distributions of the joint spatial and frequency resolution $\sigma_{xy}\sigma_{uv}$ depending on n for whitened filters learned on CIFAR-10 in the first layer with k-means and ICA.

A.2 Comments to Section 4.1 (Learning Filters)

In ZCA-whitening, we keep the number of dimensionalities so that 99% of data variance is preserved, a regularization value of 0.05 is also applied. For clustering with k-means, we use either the Approximated Nearest Neighbors or ELKAN algorithms with the Euclidean distance, implemented in VLFeat [31], which are equally good and fast. In other works (e.g., [12]), spherical k-means is often employed, however, we did not find it advantageous.

Examples of filters learned with k-means are visualized in Fig. 4. We can observe that filters for STL-10 are noisy, probably, because they are much larger (29×29) and, therefore, it is difficult for k-means to find the centroids. But on the other hand, if we decrease the filter size, then they resemble the filters learned on CIFAR-10, but the classification performance drops (Table 6), probably, because STL-10 images are relatively big and, therefore, they need larger receptive fields. Thus, one of the compromise solutions for STL-10 is to process smaller crops rather than whole images (as confirmed by the experiments, see Table 6), but in this work, we do not consider data augmentation (including cropping) and focus on images as a whole.

To train filters with ICA, the code provided at `http://research.ics.aalto.fi/ica/imageica` is applied. We use identical model settings both in case $n = 0$ and $n = [0, 4]$ (AutoCNN-ICA), 5k iterations of the ICA algorithm and similar numbers (50k-60k) of data points (image patches). These settings are also almost identical to the ones used with k-means, the only difference is that with ICA

(a) MNIST, $n = [1, 3]$ (b) CIFAR-10, $n = [0, 4]$ (c) STL-10, $n = [0, 4]$

Figure 4: Examples of 128 filters in the first layer of our AutoCNN learned using k-means with ZCA-whitening. For MNIST and CIFAR-10 filters are 13×13, for STL-10 they are 29×29. Filters are sorted according to their joint spatial and frequency resolution $\sigma_{xy}\sigma_{uv}$ (see Section 3.1), so that the top left corner corresponds to the minimum value and the bottom right corner to the maximum.

for CIFAR-10 we had to increase the filter size to $s_1 = 19$ (both with $n = 0$ and $n = [0, 4]$) to learn non-overcomplete dictionaries.

A.3 Comments to Section 4.2 (Preprocessing and Convolution)

To implement batch standardization for test samples, during training means and variances of all batches are collected, and then during inference (on the test set) their average values are used to standardize single test samples. To compute convolutions we use MatConvNet [32].

A.4 Comments to Section 4.6 (SVM Committee)

In this section, we show an example of training an SVM committee, proposed in our work, on top of multidictionary features of a 2 layer AutoCNN 1024c13-2p→128g-4ch-160c11-5p (see Table 4, where Acc1: an accuracy of a single SVM model j, Acc2: an accuracy of a committee of SVM models from 1 to j). The best results for a single SVM model and for the committee are in bold. Filters, the PCA matrix (of size $M \times P$) and all other model parameters are fixed within the committee. In this example, the dimensionality of feature vectors before projection $M = 3 \times 3 \times (1024 + 128 \times 160) = 193536$ and $P = 1000$ - the maximum dimensionality of feature vectors after projection.

Table 4: An example of classification results (the best among 10 independent tests) with the proposed committee of SVM models on the CIFAR-10 test set using all training data

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p_j	50	75	100	125	150	200	250	300	350	400	500	600	700	800	900	1000
Acc1, %	75.87	79.66	81.29	82.22	83.38	83.64	83.61	83.87	83.89	**83.92**	83.64	83.55	83.30	83.19	82.93	83.00
Acc2, %	75.87	79.27	80.97	82.40	83.18	83.52	84.07	84.43	84.61	84.82	84.94	85.03	85.26	85.17	85.23	**85.29**

For the full tests on MNIST and CIFAR-10, we report the average of 10 independent tests and, in addition, the result of averaging the SVM scores of all 10 models (denoted as "comb" in Tables 2, 3, 5 and 6). Note that for each of these 10 models (tests) an SVM committee is trained as well, as mentioned in the end of Section 5.2. To give an example, if we train a committee with $J = 16$ models (like p_j =[50:25:150,200:50:400,500:100:1000] in Table 6, where we use a Matlab notation for arrays of dimensionalities) for each independent test, then the "comb" result (e.g., 85.15% in Table 6) corresponds to averaging $16 \times 10 = 160$ SVM models. While running all 10 tests can be time consuming, running a single test (i.e. with $J = 16$ SVM models) is computationally cheap, because most of the computation time is spent on learning filters and the forward pass, which must be performed only once per test.

For classification with a one-vs-one multiclass SVM, we use implementation from [33].

A.5 Additional Data for Section 5.2 (Model Parameters Validation)

On Fig. 5, 6 and 7 we show full experimental data, which are supplementary to Fig. 2. The results shown in single markers in Fig. 5 support our argument made in Section 5.2 that for the presented image datasets whitening applied only to filters is more beneficial than completely (neither to filters nor to inputs) without whitening. For CIFAR-10 and STL-10 the advantage of $n \geq 0$ over $n = 0$ in this case (only filters are whitened) is much more noticeable than in case of whitening applied to both

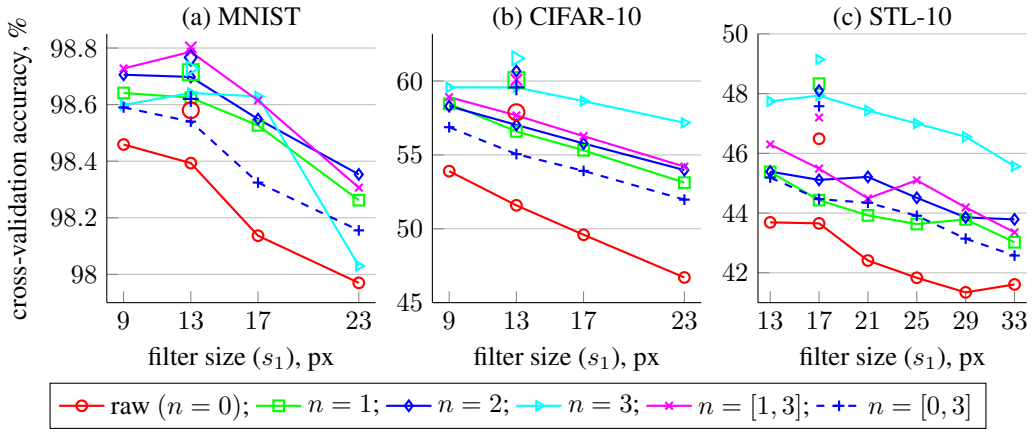

Figure 5: Comparison of classification results obtained with a single layer baseline model learned with ($n \geq 0$) and without ($n = 0$) recursive autoconvolution depending on filter size (s_1) without whitening (solid and dashed lines) and with whitening applied only to filters (in single markers for one of the values s_1 only).

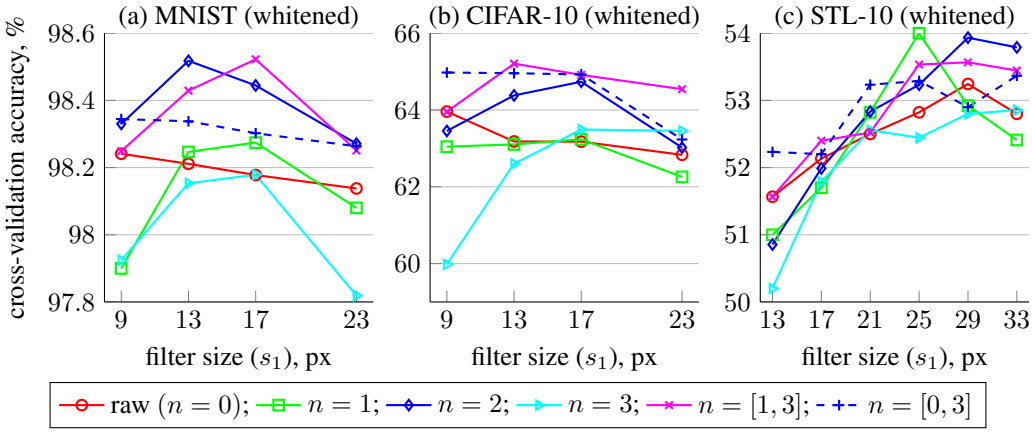

Figure 6: Comparison of classification results obtained with a single layer baseline model learned with ($n \geq 0$) and without ($n = 0$) recursive autoconvolution depending on filter size (s_1) with whitening applied both to input images and filters.

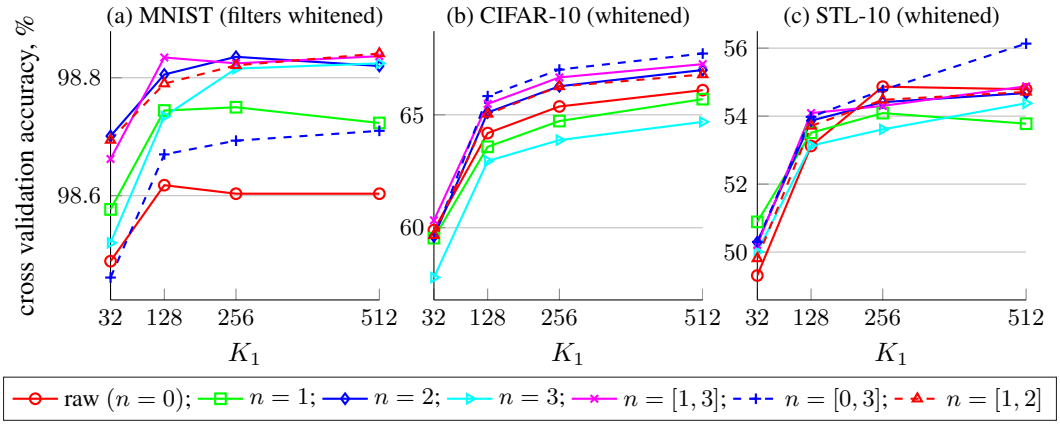

Figure 7: Comparison of classification results obtained with a single layer baseline model learned with ($n \geq 0$) and without ($n = 0$) recursive autoconvolution depending on the number of filters (K_1) with whitening applied both to input images and filters for CIFAR-10 and STL-10, and applied only to filters for MNIST.

(see Fig. 6 (b),(c) for comparison). Also, it is clear that for MNIST whitening of input images has a negative effect, given the results in Fig. 5 (a) and 6 (a).

A.6 Additional Data for Section 5.3 (Multilayer Performance)

To achieve final classification results on MNIST, reported in Tables 2 and 5, we use a rectifier $|\mathbf{x}|$ for both layers and a one-vs-one SVM (a committee of $J = 4 - 11$ models). Local contrast normalization (LCN), "rootsift" normalization, a parametric rectifier and a one-vs-all SVM is not employed for MNIST, because we have not found these methods helpful for this dataset. We use a one-vs-all classifier implemented on a GPU [27] only to report timings in Section 5.3.

To achieve final classification results on CIFAR-10 and STL-10, reported in Tables 3 and 6, we use a parametric ReLU rectifier $\max(0.25, \min(25, \mathbf{x}))$ on the first and $\max(0.25, \min(25, |\mathbf{x}|))$ on the second layer, a one-vs-all SVM (a committee of $J = 8 - 20$ models) if not specified otherwise, with LCN between layers and with "rootsift" normalization after each layer. LCN is a quite expensive operation (in our implementation) and, in our experience, has a positive effect only in case of many feature maps (i.e. $K_1 > 128$), so we employ it only for final tests with large networks.

For 2 layer AutoCNNs, the number of parameters in convolutional layers (N_{model}) varies considerably depending on whether we use all features from layer 1 (a full network) for classification or only those connected to the second layer (a pruned network). For instance, for a large CIFAR-10 network in the former case we have $N_{model} = 1024 \times 13 \times 13 \times 3 + 160 \times 11 \times 11 \times 4 \approx 600$k, while in the latter case $N_{model} = 400 \times 13 \times 13 \times 3 + 160 \times 11 \times 11 \times 4 \approx 280$k, where the value of 400 is an approximate number of filters connected to the second layer in our experiments. Note that it is smaller than $128 \times 4 = 512$, because some feature maps are connected to several groups (see Section 4.4). Thus, pruned networks have much fewer parameters and, as reported in Tables 5 and 6, they preserve or even improve the classification accuracy of their full counterparts. The architectures of the full and pruned networks are initially the same and correspond to the ones used for the experiments in Tables 2 and 3, except that the full network for CIFAR-10 has larger pooling $m_2 = 5$. The 2 layer network for MNIST with 128-64 filters is the same that we used to obtain the results in Table 1.

The extended experimental results in Tables 5 and 6 support our arguments made in Sections 4.5 and 5.3 that a one-vs-all SVM is advantageous compared to one-vs-one for the CIFAR-10 and STL-10 datasets, and it is better to go deeper than use a larger single layer network.

Table 5: An extended version of Table 2 (Comparison of final classification errors on MNIST) with additional results and details

Model	MNIST (100)	MNIST (300)	MNIST (comb)
1 Layer, unsupervised, no data augmentation			
PCA dimensionality, p_j	[50 70 90 100]	[50 70 90 100 120 150 200]	[50 70 90 100 120 150 200 250]
AutoCNN (16c13-4p)	2.58 ± 0.17	1.85 ± 0.27	0.60 ± 0.03 (0.53)
AutoCNN (64c13-4p)	2.13 ± 0.17	1.28 ± 0.09	0.46 ± 0.02 (0.44)
AutoCNN (128c13-4p)	2.08 ± 0.26	1.15 ± 0.05	0.47 ± 0.02 (0.45)
AutoCNN (256c13-4p)	1.97 ± 0.14	1.17 ± 0.10	0.42 ± 0.02 (0.42)
AutoCNN (384c13-4p)	1.95 ± 0.09	1.23 ± 0.14	0.44 ± 0.02 (0.42)
PCA dimensionality, p_j	[50 70 90 100 120 150 200 250]		
AutoCNN (256c13-4p)	1.90 ± 0.17	1.22 ± 0.11	–
$n = 0$ (256c7-4p)	2.28 ± 0.11	1.38 ± 0.09	–
$n = 0$ (256c9-4p)	2.20 ± 0.21	1.36 ± 0.10	–
$n = 0$ (256c11-4p)	2.17 ± 0.16	1.34 ± 0.07	–
$n = 0$ (256c13-4p)	2.21 ± 0.14	1.36 ± 0.10	–
2 Layers, unsupervised, no data augmentation			
PCA dimensionality, p_j	[50 70 90 100 120 150 200 250]		
2 layer AutoCNN (128-64)	1.54 ± 0.12	1.01 ± 0.05	0.46 ± 0.02 (0.39)
PCA dimensionality, p_j	[50 70 90 100 120 150 200 250 300 350 400]		[50 70 120 200 250 300 350 400]
2 layer AutoCNN (192-64)	1.47 ± 0.07	0.97 ± 0.11	–
2 layer AutoCNN (85-64)	1.46 ± 0.07	0.97 ± 0.11	0.42 ± 0.03 (0.38)

The number of models (J) in a proposed SVM committee varies (Tables 5 and 6), because sometimes it is better to use more models and sometimes fewer depending on the number of features, size of the training dataset, multiclass classifier and unsupervised method. We have not particularly fine-tuned

Table 6: An extended version of Table 3 (Comparison of final classification accuracies on CIFAR-10 and STL-10) with additional results and details. Parameters for STL-10 in parentheses.

Model	CIFAR-10 (400)	CIFAR-10 (comb)	STL-10
1 Layer, unsupervised, no data augmentation			
PCA dimensionality, p_j	[50 75 100 125 150 200 250 300]		[30:10:100,120,150:50:250,300:100:600]
AutoCNN (128c13(c29))-onevsone	68.21 ± 0.43	79.50	60.03 ± 0.63
AutoCNN (1024c13(c29))-onevsone	69.97 ± 0.39	81.62	61.82 ± 0.55
AutoCNN (2048c13(c29))-onevsone	69.77 ± 0.32	81.39	61.57 ± 0.54
AutoCNN (1024c13(c29))	71.59 ± 0.21	82.74 ± 0.11 (82.93)	62.52 ± 0.59
PCA dimensionality, p_j	[50:25:150,200:50:350]		[30:10:100,120,150:50:250,300:100:600]
AutoCNN (128c13(c29))	69.45 ± 0.42	81.04 ± 0.17 (82.01)	61.04 ± 0.70
AutoCNN (2048c13(c29))	71.32 ± 0.22	83.12 ± 0.11 (83.24)	62.01 ± 0.60
$n = 0$ (1024c7(c17))	69.53 ± 0.30	–	60.13 ± 0.62
$n = 0$ (1024c9(c21))	69.97 ± 0.32	–	60.67 ± 0.70
$n = 0$ (1024c11(c25))	69.75 ± 0.41	–	60.69 ± 0.77
$n = 0$ (1024c13(c29))	69.70 ± 0.28	–	60.87 ± 0.59
PCA dimensionality, p_j	[50:25:150,200:50:350]	[50:25:150,200:50:350,400,500,600]	[30:10:100,120,150:50:250,300:100:600]
AutoCNN (1024c13(c29))	71.53 ± 0.20	83.18 ± 0.16 (83.30)	62.52 ± 0.59
ICA (1024c19(29))	67.85 ± 0.34	81.09 ± 0.14 (81.30)	58.75 ± 0.67
AutoCNN-ICA (1024c19(29))	71.52 ± 0.50	83.13 ± 0.14 (83.46)	61.89 ± 0.53
2 Layers, unsupervised, no data augmentation			
PCA dimensionality, p_j	[50:25:150,200:50:400,500:100:1000]		[60,70,120,150,300:100:1000]
AutoCNN (1024-160)	74.70 ± 0.41	84.97 ± 0.18 (85.15)	68.22 ± 0.52
PCA dimensionality, p_j	[50:25:150,200:50:400,500:100:1000]		[30,40,70,80,120,150,250,300:100:1500]
2 layer AutoCNN (400(500)-160)	75.12 ± 0.32	85.01 ± 0.18 (85.43)	68.69 ± 0.53
1-2 Layers, unsupervised + cropping			
PCA dimensionality, p_j			[50:10:100,120,150:50:250,300:100:600]
1 layer AutoCNN (1024c19-16p)	–	–	63.74 ± 0.49
PCA dimensionality, p_j			[60,70,120,150,300:100:1500]
2 layer AutoCNN (1024-160)	–	–	69.14 ± 0.26

the value of J and dimensionalities (p_j) in the committees and tried to make them more or less consistent among all experiments.

Next, on the test datasets we double-check (in additional to cross-validation) that filter size (s_1) was not the reason for our performance increase. We present several results obtained with $n = 0$ using the settings identical to $n = [1, 3]$ (MNIST) and $n = [0, 4]$ (CIFAR-10 and STL-10), but with smaller filters (typically used in previous works) (Tables 5 and 6). Note, that for two layer AutoCNNs, the performance increase of using $n \geq 0$ compared to $n = 0$ is more pronounced than for single layer models (see Table 1).

For the sake of curiosity and motivated by our promising results on small (32×32) CIFAR-10 images with the suggested method, we also conducted a couple of experiments with a cropped version of STL-10. But, we do not report these results in Table 3, because we consider only unsupervised learning methods without data augmentation, and it would be unfair to compare our results with others who do not employ cropping. In these experiments, we randomly take 10 crops of size 64×64 from the original (96×96) unlabeled and training samples, and 16 crops with fixed spatial positions from the test samples. We take only 5k original unlabeled samples. As a result, we have 50k cropped unlabeled, 10k training (for each of the 10 predefined folds) samples and 128k test samples. To obtain labels and an accuracy for the original 8k test samples of STL-10, for each test sample we average the SVM scores obtained for 16 crops (so, in total we train $16 \times J$ SVM models) and take the maximum value to predict the label. We designed a 2 layer network 1024c19-4p→128g-4ch-160c11-4p, which is similar to the one used for STL-10 without cropping. The results show significant improvements compared to the experiments without cropping (Table 6).