An Analysis of Single-Layer Networks in Unsupervised Feature Learning

Adam Coates
Stanford University
Computer Science Dept.
353 Serra Mall
Stanford, CA 94305

Honglak Lee
University of Michigan
Computer Science and Engineering
2260 Hayward Street
Ann Arbor, MI 48109

Andrew Y. Ng
Stanford University
Computer Science Dept.
353 Serra Mall
Stanford, CA 94305

Abstract

A great deal of research has focused on algorithms for learning features from unlabeled data. Indeed, much progress has been made on benchmark datasets like NORB and CIFAR by employing increasingly complex unsupervised learning algorithms and deep models. In this paper, however, we show that several simple factors, such as the number of hidden nodes in the model, may be more important to achieving high performance than the learning algorithm or the depth of the model. Specifically, we will apply several off-the-shelf feature learning algorithms (sparse auto-encoders, sparse RBMs, K-means clustering, and Gaussian mixtures) to CIFAR, NORB, and STL datasets using only single-layer networks. We then present a detailed analysis of the effect of changes in the model setup: the receptive field size, number of hidden nodes (features), the step-size ("stride") between extracted features, and the effect of whitening. Our results show that large numbers of hidden nodes and dense feature extraction are critical to achieving high performance—so critical, in fact, that when these parameters are pushed to their limits, we achieve state-of-the-art performance on both CIFAR-10 and NORB using only a single layer of features. More surprisingly, our best performance is based on K-means clustering, which is extremely fast, has no hyper-parameters to tune beyond the model structure itself, and is very easy to implement. Despite the simplicity of our system, we achieve accuracy beyond all previously published results on the CIFAR-10 and NORB datasets (79.6% and 97.2% respectively).

Appearing in Proceedings of the 14^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2011, Fort Lauderdale, FL, USA. Volume 15 of JMLR: W&CP 15.

1 Introduction

Much recent work in machine learning has focused on learning good feature representations from unlabeled input data for higher-level tasks such as classification. Current solutions typically learn multi-level representations by greedily "pre-training" several layers of features, one layer at a time, using an unsupervised learning algorithm [11, 8, 18]. For each of these layers a number of design parameters are chosen: the number of features to learn, the locations where these features will be computed, and how to encode the inputs and outputs of the system. In this paper we study the effect of these choices on single-layer networks trained by several feature learning methods. Our results demonstrate that several key ingredients, orthogonal to the learning algorithm itself, can have a large impact on performance: whitening, large numbers of features, and dense feature extraction can all be major advantages. Even with very simple algorithms and a single layer of features, it is possible to achieve state-of-the-art performance by focusing effort on these choices rather than on the learning system itself.

A major drawback of many feature learning systems is their complexity and expense. In addition, many algorithms require careful selection of multiple hyper-parameters like learning rates, momentum, sparsity penalties, weight decay, and so on that must be chosen through cross-validation, thus increasing running times dramatically. Though it is true that recently introduced algorithms have consistently shown improvements on benchmark datasets like NORB [16] and CIFAR-10 [13], there are several other factors that affect the final performance of a feature learning system. Specifically, there are many "meta-parameters" defining the network architecture, such as the receptive field size and number of hidden nodes (features). In practice, these parameters are often determined by computational constraints. For instance, we might use the largest number of features possible considering the running time of the algorithm. In this paper, however, we pursue an alternative strategy: we employ very *simple* learning algorithms and then more care-

fully choose the network parameters in search of higher performance. If (as is often the case) larger representations perform better, then we can leverage the speed and simplicity of these learning algorithms to use larger representations.

To this end, we will begin in Section 3 by describing a simple feature learning framework that incorporates an unsupervised learning algorithm as a "black box" module within. For this "black box", we have implemented several off-the-shelf unsupervised learning algorithms: sparse auto-encoders, sparse RBMs, K-means clustering, and Gaussian mixture models. We then analyze the performance impact of several different elements in the feature learning framework, including: (i) whitening, which is a common pre-process in deep learning work, (ii) number of features trained, (iii) step-size (stride) between extracted features, and (iv) receptive field size.

It will turn out that whitening, large numbers of features, and small stride lead to uniformly better performance regardless of the choice of unsupervised learning algorithm. On the one hand, these results are somewhat unsurprising. For instance, it is widely held that highly over-complete feature representations tend to give better performance than smaller-sized representations [32], and similarly with small strides between features [21]. However, the main contribution of our work is demonstrating that these considerations may, in fact, be *critical* to the success of feature learning algorithms—potentially more important even than the choice of unsupervised learning algorithm. Indeed, it will be shown that when we push these parameters to their limits that we can achieve state-of-the-art performance, outperforming many other more complex algorithms on the same task. Quite surprisingly, our best results are achieved using K-means clustering, an algorithm that has been used extensively in computer vision, but that has not been widely adopted for "deep" feature learning. Specifically, we achieve the test accuracies of 79.6% on CIFAR-10 and 97.2% on NORB—better than all previously published results.

We will start by reviewing related work on feature learning, then move on to describe a general feature learning framework that we will use for evaluation in Section 3. We then present experimental analysis and results on CIFAR-10 [13] as well as NORB [16] in Section 4.

2 Related work

Since the introduction of unsupervised pre-training [8], many new schemes for stacking layers of features to build "deep" representations have been proposed. Most have focused on creating new training algorithms to build single-layer models that are composed to build deeper structures. Among the algorithms considered in the literature are sparse-coding [22, 17, 32], RBMs [8, 13], sparse RBMs [18], sparse auto-encoders [7, 25], denoising auto-encoders [30], "factored" [24] and mean-covariance [23] RBMs, as well as many others [19, 33]. Thus, amongst the many components of feature learning architectures, the unsupervised learning module appears to be the most heavily scrutinized.

Some work, however, has considered the impact of other choices in these feature learning systems, especially the choice of network architecture. Jarret et al. [11], for instance, have considered the impact of changes to the "pooling" strategies frequently employed between layers of features, as well as different forms of normalization and rectification between layers. Similarly, Boureau et al. have considered the impact of coding strategies and different types of pooling, both in practice [3] and in theory [4]. Our work follows in this vein, but considers instead the structure of single-layer networks—before pooling, and orthogonal to the choice of algorithm or coding scheme.

Many common threads from the computer vision literature also relate to our work and to feature learning more broadly. For instance, we will use the K-means clustering algorithm as an alternative unsupervised learning module. K-means has been used less widely in "deep learning" work but has enjoyed wide adoption in computer vision for building codebooks of "visual words" [5, 6, 15, 31], which are used to define higher-level image features. This method has also been applied recursively to build multiple layers of features [1]. The effects of pooling and choice of activation function or coding scheme have similarly been studied for these models [15, 28, 21]. Van Gemert et al., for instance, demonstrate that "soft" activation functions ("kernels") tend to work better than the hard assignment typically used with visual words models.

This paper will compare results along some of the same axes as these prior works (e.g., we will consider both 'hard' and 'soft' activation functions), but our conclusions differ somewhat: While we confirm that some feature-learning schemes are better than others, we also show that the differences can often be outweighed by other factors, such as the number of features. Thus, even though more complex learning schemes may improve performance slightly, these advantages can be overcome by fast, simple learning algorithms that are able to handle larger networks.

3 Unsupervised feature learning framework

In this section, we describe a common framework used for feature learning. For concreteness, we will focus on the application of these algorithms to learning features from images, though our approach is applicable

to other forms of data as well. The framework we use involves several stages and is similar to those employed in computer vision [5, 15, 31, 28, 1], as well as other feature learning work [16, 19, 3].

At a high-level, our system performs the following steps to learn a feature representation:

1. Extract random patches from unlabeled training images.
2. Apply a pre-processing stage to the patches.
3. Learn a feature-mapping using an unsupervised learning algorithm.

Given the learned feature mapping and a set of labeled training images we can then perform feature extraction and classification:

1. Extract features from equally spaced sub-patches covering the input image.
2. Pool features together over regions of the input image to reduce the number of feature values.
3. Train a linear classifier to predict the labels given the feature vectors.

We will now describe the components of this pipeline and its parameters in more detail.

3.1 Feature Learning

As mentioned above, the system begins by extracting random sub-patches from unlabeled input images. Each patch has dimension w-by-w and has d channels,[1] with w referred to as the "receptive field size". Each w-by-w patch can be represented as a vector in $\mathbb{R}^N$ of pixel intensity values, with $N = w \cdot w \cdot d$. We then construct a dataset of m randomly sampled patches, $X = \{x^{(1)}, ..., x^{(m)}\}$, where $x^{(i)} \in \mathbb{R}^N$. Given this dataset, we apply the pre-processing and unsupervised learning steps.

3.1.1 Pre-processing

It is common practice to perform several simple normalization steps before attempting to generate features from data. In this work, we assume that every patch $x^{(i)}$ is normalized by subtracting the mean and dividing by the standard deviation of its elements. For visual data, this corresponds to local brightness and contrast normalization.

After normalizing each input vector, the entire dataset X may optionally be whitened [10]. While this process is commonly used in deep learning work (e.g., [24]) it is less frequently employed in computer vision. We will present experimental results obtained both with and without whitening to determine whether this component is generally necessary.

3.1.2 Unsupervised learning

After pre-processing, an unsupervised learning algorithm is used to discover features from the unlabeled data. For our purposes, we will view an unsupervised learning algorithm as a "black box" that takes the dataset X and outputs a function $f : \mathbb{R}^N \to \mathbb{R}^K$ that maps an input vector $x^{(i)}$ to a new feature vector of K features, where K is a parameter of the algorithm. We denote the kth feature as f_k. In this work, we will use several different unsupervised learning methods[2] in this role: (i) sparse auto-encoders, (ii) sparse RBMs, (iii) K-means clustering, and (iv) Gaussian mixtures. We briefly summarize how these algorithms are employed in our system.

1. **Sparse auto-encoder:** We train an auto-encoder with K hidden nodes using back-propagation to minimize squared reconstruction error with an additional penalty term that encourages the units to maintain a low average activation [18, 7]. The algorithm outputs weights $W \in \mathbb{R}^{K \times N}$ and biases $b \in \mathbb{R}^K$ such that the feature mapping f is defined by:

$$f(x) = g(Wx + b), \tag{1}$$

   where $g(z) = 1/(1 + exp(-z))$ is the logistic sigmoid function, applied component-wise to the vector z.

   There are several hyper-parameters used by the training algorithm (e.g., weight decay, and target activation). These parameters were chosen using cross-validation for each choice of the receptive field size, w.[3]

2. **Sparse restricted Boltzmann machine:** The restricted Boltzmann machine (RBM) is an undirected graphical model with K binary hidden variables. Sparse RBMs can be trained using the contrastive divergence approximation [9] with the same type of sparsity penalty as the auto-encoders. The training also produces weights

[1] For example, if the input image is represented in (R,G,B) colors, then it has three channels.

[2] These algorithms were chosen since they can scale up straight-forwardly to the problem sizes considered in our experiments.

[3] Ideally, we would perform this cross-validation for every choice of parameters, but the expense is prohibitive for the number of experiments we perform here. This is a major advantage of the K-means algorithm, which requires no such procedure.

W and biases b, and we can use the same feature mapping as the auto-encoder (as in Equation (1))—thus, these algorithms differ primarily in their training method. Also as above, the necessary hyper-parameters are determined by cross-validation for each receptive field size.

3. **K-means clustering:** We apply K-means clustering to learn K centroids $c^{(k)}$ from the input data. Given the learned centroids $c^{(k)}$, we consider two choices for the feature mapping f. The first is the standard 1-of-K, hard-assignment coding scheme:

$$f_k(x) = \begin{cases} 1 & \text{if } k = \arg\min_j ||c^{(j)} - x||_2^2 \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

This is a (maximally) sparse representation that has been used frequently in computer vision [5]. It has been noted, however, that this may be too terse [28]. Thus our second choice of feature mapping is a non-linear mapping that attempts to be "softer" than the above encoding while also keeping some sparsity:

$$f_k(x) = \max\{0, \mu(z) - z_k\} \quad (3)$$

where $z_k = ||x - c^{(k)}||_2$ and $\mu(z)$ is the mean of the elements of z. This activation function outputs 0 for any feature f_k where the distance to the centroid $c^{(k)}$ is "above average". In practice, this means that roughly half of the features will be set to 0. This can be thought of as a very simple form of "competition" between features.

We refer to these in our results as `K-means (hard)` and `K-means (triangle)` respectively.

4. **Gaussian mixtures:** Gaussian mixture models (GMMs) represent the density of input data as a mixture of K Gaussian distributions and is widely used for clustering. GMMs can be trained using the Expectation-Maximization (EM) algorithm as in [1]. We run a single iteration of K-means to initialize the mixture model.[4] The feature mapping f maps each input to the posterior membership probabilities:

$$f_k(x) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \cdot \exp\left(-\frac{1}{2}(x - c^{(k)})^\top \Sigma_k^{-1}(x - c^{(k)})\right)$$

where Σ_k is a diagonal covariance and ϕ_k are the cluster prior probabilities learned by the EM algorithm.

[4]When K-means is run to convergence we have found that the mixture model does not learn features substantially different from the K-means result.

3.2 Feature Extraction and Classification

The above steps, for a particular choice of unsupervised learning algorithm, yield a function f that transforms an input patch $x \in \mathbb{R}^N$ to a new representation $y = f(x) \in \mathbb{R}^K$. Using this feature extractor, we now apply it to our (labeled) training images for classification.

3.2.1 Convolutional extraction

Using the learned feature extractor $f : \mathbb{R}^N \to \mathbb{R}^K$, given any w-by-w image patch, we can now compute a representation $y \in \mathbb{R}^K$ for that patch. We can thus define a (single layer) representation of the entire image by applying the function f to many sub-patches. Specifically, given an image of n-by-n pixels (with d channels), we define a $(n - w + 1)$-by-$(n - w + 1)$ representation (with K channels), by computing the representation y for each w-by-w "subpatch" of the input image. More formally, we will let $y^{(ij)}$ be the K-dimensional representation extracted from location i, j of the input image. For computational efficiency, we may also "step" our w-by-w feature extractor across the image with some step-size (or "stride") s greater than 1. This is illustrated in Figure 1.

3.2.2 Classification

Before classification, it is standard practice to reduce the dimensionality of the image representation by pooling. For a stride of $s = 1$, our feature mapping produces a $(n-w+1)$-by-$(n-w+1)$-by-K representation. We can reduce this by summing up over local regions of the $y^{(ij)}$'s extracted as above. This procedure is commonly used (in many variations) in computer vision [15] as well as deep feature learning [11].

In our system, we use a very simple form of pooling. Specifically, we split the $y^{(ij)}$'s into four equal-sized quadrants, and compute the sum of the $y^{(ij)}$'s in each. This yields a reduced (K-dimensional) representation of each quadrant, for a total of $4K$ features that we use for classification.

Given these pooled ($4K$-dimensional) feature vectors for each training image and a label, we apply standard linear classification algorithms. In our experiments we use (L2) SVM classification. The regularization parameter is determined by cross-validation.

4 Experiments and Analysis

The above framework includes a number of parameters that can be changed: (i) whether to use whitening or not, (ii) the number of features K, (iii) the stride s, and (iv) receptive field size w. In this section, we present our experimental results on the impact of these parameters on performance. First, we will evaluate the effects of these parameters using cross-validation

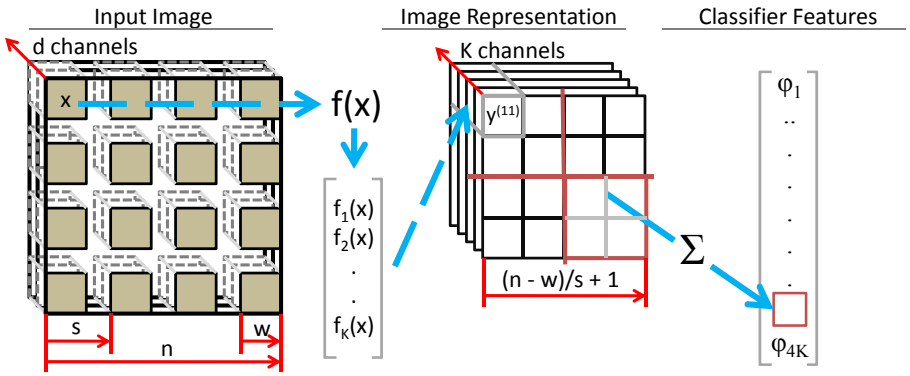


Figure 1: Illustration showing feature extraction using a w-by-w receptive field and stride s. We first extract w-by-w patches separated by s pixels each, then map them to K-dimensional feature vectors to form a new image representation. These vectors are then pooled over 4 quadrants of the image to form a feature vector for classification. (For clarity we have drawn the leftmost figure with a stride greater than w, but in practice the stride is almost always smaller than w.)

on the CIFAR-10 training set. We will then report the results achieved on both CIFAR-10 and NORB test sets using each unsupervised learning algorithm and the parameter settings that our analysis suggests is best overall (i.e., in our final results, we use the same settings for all algorithms).[5]

Our basic testing procedure is as follows. For each unsupervised learning algorithm in Section 3.1.2, we will train a single-layer of features using either whitened data or raw data and a choice of the parameters K, s, and w. We then train a linear classifier as described in Section 3.2.2, then test the classifier on a holdout set (for our main analysis) or the test set (for our final results).

4.1 Visualization

Before we present classification results, we first show visualizations of the learned feature representations. The bases (or centroids) learned by sparse autoencoders, sparse RBMs, K-means, and Gaussian mixture models are shown in Figure 2 for 8 pixel receptive fields. It is well-known that autoencoders and RBMs yield localized filters that resemble Gabor filters and we can see this in our results both when using whitened data and, to a lesser extent, raw data. However, these visualizations also show that similar results can be achieved using clustering algorithms. In particular, while clustering raw data leads to centroids consistent with those in [6] and [29], we see that clustering whitened data yields sharply localized filters that are very similar to those learned by the other algorithms. Thus, it appears that such features are easy to learn with clustering methods (without any parameter tweaking) as a result of whitening.

[5] To clarify: The parameters used in our final evaluation are those that achieved the best (average) cross-validation performance across all models: whitening, 1 pixel stride, 6 pixel receptive field, and 1600 features.

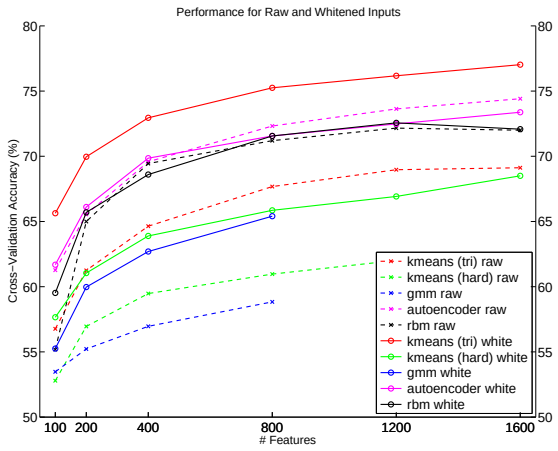


Figure 3: Effect of whitening and number of bases (or centroids).

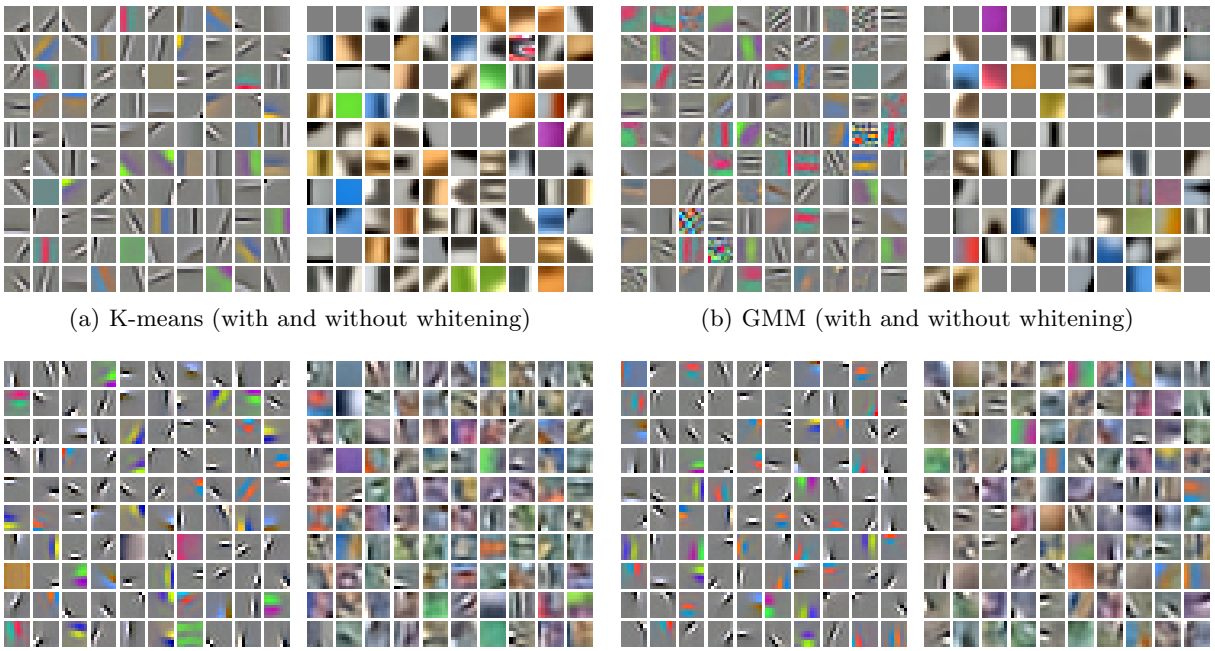
4.2 Effect of whitening

We now move on to our characterization of performance on various axes of parameters, starting with the effect of whitening[6], which visibly changes the learned bases (or centroids) as seen in Figure 2. Figure 3 shows the performance for all of our algorithms as a function of the number of features (which we will discuss in the next section) both with and without whitening. These experiments used a stride of 1 pixel and 6 pixel receptive field.

For sparse autoencoders and RBMs, the effect of whitening is somewhat ambiguous. When using only 100 features, there is a significant benefit of whitening for sparse RBMs, but this advantage disappears with larger numbers of features. For the clustering algorithms, however, we see that whitening is a crucial pre-process since the clustering algorithms cannot handle the correlations in the data.[7]

[6] In our experiments, we use Zero-phase whitening [2]

[7] Our GMM implementation uses diagonal covariances



(a) K-means (with and without whitening)

(b) GMM (with and without whitening)

(c) Sparse Autoencoder (with and without whitening)

(d) Sparse RBM (with and without whitening)

Figure 2: Randomly selected bases (or centroids) trained on CIFAR-10 images using different learning algorithms. Best viewed in color.

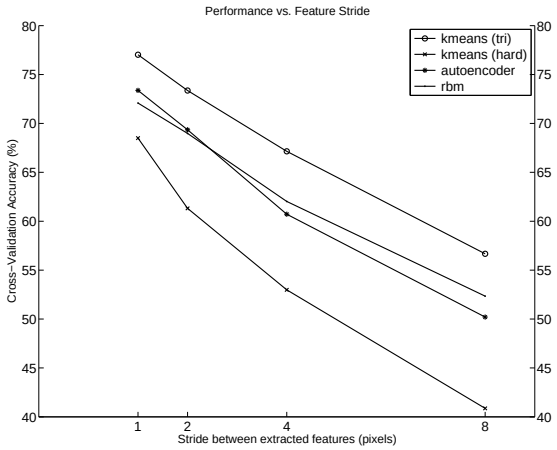


Figure 4: Effect of stride.

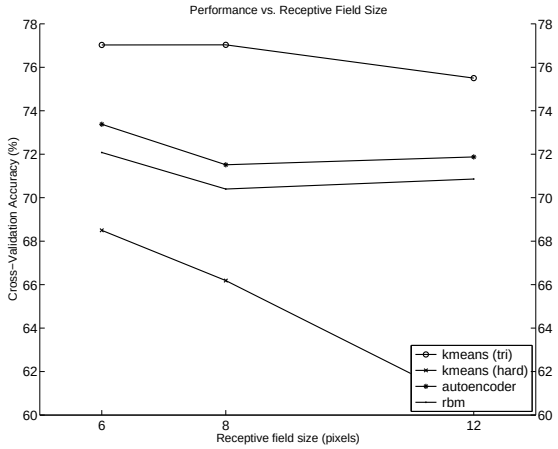


Figure 5: Effect of receptive field size.

Clustering algorithms have been applied successfully to raw pixel inputs in the past [6, 29] but these applications did not use whitened input data. Our results suggest that improved performance might be obtained by incorporating whitening.

4.3 Number of features

Our experiments considered feature representations with 100, 200, 400, 800, 1200, and 1600 learned features.[8] Figure 3 clearly shows the effect of increasing the number of learned features: all algorithms generally achieved higher performance by learning more features as expected.

Surprisingly, K-means clustering coupled with the "triangle" activation function and whitening achieves the highest performance. This is particularly notable since K-means requires no tuning whatsoever, unlike the sparse auto-encoder and sparse RBMs which require us to choose several hyper-parameters for best results.

4.4 Effect of stride

The "stride" s used in our framework is the spacing between patches where feature values will be extracted (see Figure 1). Frequently, learning systems will use

and K-means uses Euclidean distance.

[8] We found that training Gaussian mixture models with more than 800 components was often difficult and always extremely slow. Thus we only ran this algorithm with up to 800 components.

a stride $s > 1$ because computing the feature mapping is very expensive. For instance, sparse coding requires us to solve an optimization problem for each such patch, which may be prohibitive for a stride of 1. It is reasonable to ask, then, how much this compromise costs in terms of performance for the algorithms we consider (which all have the property that their feature mapping can be computed extremely quickly). In this experiment, we fixed the number of features (1600) and receptive field size (6 pixels), and vary the stride over 1, 2, 4, and 8 pixels. The results are shown in Figure 4. (We do not report results with GMMs, since training models of this size was impractical.)

The plot shows a clear downward trend in performance with increasing step size as expected. However, the magnitude of the change is striking: for even a stride of $s = 2$, we suffer a loss of 3% or more accuracy, and for $s = 4$ we lose at least 5%. These differences can be significant in comparison to the choice of algorithm. For instance, a sparse RBM with stride of 2 performed comparably to the simple hard-assignment K-means scheme using a stride of 1—one of the simplest possible algorithms we could have chosen for unsupervised learning (and certainly much simpler than a sparse RBM).

4.5 Effect of receptive field size

Finally, we also evaluated the effect of receptive field size. Given a scalable algorithm, it's possible that leveraging it to learn larger receptive fields could allow us to recognize more complex features that cover a larger region of the image. On the other hand, this increases the dimensionality of the space that the algorithm must cover and may require us to learn more features or use more data. As a result, given the same amount of data and using the same number of features, it is not clear whether this is a worthwhile investment. In this experiment, we tested receptive field sizes of 6, 8, and 12 pixels. For other parameters, we used whitening, stride of 1 pixel, and 1600 bases (or centroids).

The summary results are shown in Figure 5. Overall, the 6 pixel receptive field worked best. Meanwhile, 12 pixel receptive fields were similar or worse than 6 or 8 pixels. Thus, if we have computational resource to spare, our results suggest that it is better to spend it on reducing stride and expanding the number of learned features.

Unfortunately, unlike for the other parameters, the receptive field size does appear to require cross validation in order to make an informed choice. Our experiments do suggest, though, that even very small receptive fields can work well (with pooling) and are worth considering. This is especially important if reducing the input size allows us to use a smaller stride or more features which both have large positive impact on results.

Table 1: Test recognition accuracy on CIFAR-10

Algorithm	Accuracy
Raw pixels (reported in [13])	37.3%
3-Way Factored RBM (3 layers) [24]	65.3%
Mean-covariance RBM (3 layers) [23]	71.0%
Improved Local Coord. Coding [33]	74.5%
Conv. Deep Belief Net (2 layers) [14]	78.9%
Sparse auto-encoder	73.4%
Sparse RBM	72.4%
K-means (Hard)	68.6%
K-means (Triangle)	77.9%
K-means (Triangle, 4000 features)	**79.6%**

Table 2: Test recognition accuracy (and error) for NORB (normalized-uniform)

Algorithm	Accuracy (error)
Conv. Neural Network [16]	93.4% (6.6%)
Deep Boltzmann Machine [26]	92.8% (7.2%)
Deep Belief Network [20]	95.0% (5.0%)
(Best result of [11])	94.4% (5.6%)
Deep neural network [27]	**97.13% (2.87%)**
Sparse auto-encoder	96.9% (3.1%)
Sparse RBM	96.2% (3.8%)
K-means (Hard)	96.9% (3.1%)
K-means (Triangle)	97.0% (3.0%)
K-means (Triangle, 4000 features)	**97.21% (2.79%)**

4.6 Final classification results

We have shown that whitening, a stride of 1 pixel, a 6 pixel receptive field, and a large number of features works best on average across all algorithms for CIFAR-10. Using these parameters we ran our full pipeline on the entire CIFAR-10 training set, trained a SVM classifier and tested on the standard CIFAR-10 test set. Our final test results on the CIFAR-10 data set with these settings are reported in Table 1 along with results from other publications. Quite surprisingly, the `K-means (triangle)` algorithm attains very high performance (77.9%) with 1600 features. Based on this success, we sought to improve the results further simply by increasing the number of features to 4000. Using these features, our test accuracy increased to 79.6%.

Based on our analysis here, we have also run each of these algorithms on the NORB "normalized uniform" dataset. We use all of the same parameters as for CIFAR-10, including the 6 pixel receptive field size. The results are summarized in Table 2. Here, all of the algorithms achieve very high performance. Again, K-means with the "triangle" activation achieves the highest performance. When using 4000 features as for

Table 3: Test recognition accuracy on STL-10

Algorithm	Accuracy
Raw pixels	31.8% (±0.62%)
K-means (Triangle 1600 features)	51.5% (±1.73%)

CIFAR, we achieve 97.21% accuracy. We note, however, that the other results are very similar regardless of the algorithm used. This suggests that the main source of performance here is from our choice of network structure, not from the particular choice of unsupervised learning algorithm.

Finally, we also ran our system on the new STL-10 dataset[9]. This dataset uses higher resolution (96x96) images, but allows many fewer training examples (100 per class), while providing a large unlabeled training set—thus forcing algorithms to rely heavily on acquired prior knowledge of image statistics. We applied the same system as used for CIFAR on downsampled versions of the STL images (32x32 pixels). In this case, the performance is much lower than on the comparable CIFAR dataset on account of the small labeled datasets: 51.5% (±1.73%) (compared to 31.8% (±0.62%) for raw pixels). This suggests that the method proposed here is strongest when we have large labeled training sets as with NORB and CIFAR.

5 Discussion

Our results above may seem inexplicable considering the simplicity of the system—it is not clear, on first inspection, exactly what in our experiments allows us to achieve such high performance compared to prior work. We believe that the main explanation for the performance gain is, in fact, our choice of network parameters since almost all of the algorithms performed favorably relative to previous results.

Each of the network parameters (feature count, stride and receptive field size) we've tested potentially confers a significant benefit on performance. For instance, large numbers of features (regardless of how they're trained) gives us many non-linear projections of the data. Unlike simple linear projections, which have limited representational power, it is well-known that using extremely large numbers of non-linear projections can make data closer to linearly separable and thus easier to classify. Hence, larger numbers of features may be uniformly beneficial, regardless of the training algorithm.

The dramatic impact of changes to the stride parameter may be partly explained by the work of Boureau [4]. By setting the stride small, a larger number of samples are incorporated into each pooling area which was shown both theoretically and empirically to improve results. It is also likely that high-frequency features (edges) are more accurately identified using a dense sampling.

Finally, the receptive field size, which we chose by cross-validation appears to be important as well. It appears that large receptive fields result in a space that is simply too large to cover effectively with a small number of nonlinear features. For instance, because our features often include shifted copies of edges, increasing the receptive field size also increases the amount of redundancy we can expect in our filters. This caveat might be ameliorated by training convolutionally [19, 16, 12]. Note that small receptive fields might also increase the number of samples used in pooling and thus have a small effect similar to using a smaller stride.

6 Conclusion

In this paper we have conducted extensive experiments on the CIFAR-10 dataset using multiple unsupervised feature learning algorithms to characterize the effect of various parameters on classification performance. While confirming the basic finding that more features and dense extraction are useful, we have shown more importantly that these elements can, in fact, be as important as the unsupervised learning algorithm itself. Surprisingly, we have shown that even the K-means clustering algorithm—an extremely simple learning algorithm with no parameters to tune—is able to achieve state-of-the-art performance on both CIFAR-10 and NORB datasets when used with the network parameters that we have identified in this work. We've also shown more generally that smaller stride and larger numbers of features yield monotonically improving performance, which suggests that while more complex algorithms may have greater representational power, simple but fast algorithms can be highly competitive.

References

[1] A. Agarwal and B. Triggs. Hyperfeatures multilevel local coding for visual recognition. In *European Conference on Computer Vision*, 2006.

[2] A. Bell and T. J. Sejnowski. The 'independent components' of natural scenes are edge filters. *Vision Research*, 37, 1997.

[3] Y. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *CVPR*, 2010.

[4] Y. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *International Conference on Machine Learning*, 2010.

[5] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of key-

[9] http://cs.stanford.edu/∼acoates/stl10

points. In *ECCV Workshop on Statistical Learning in Computer Vision*, 2004.

[6] L. Fei-Fei and P. Perona. A Bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition*, 2005.

[7] I. Goodfellow, Q. Le, A. Saxe, H. Lee, and A. Ng. Measuring invariances in deep networks. In *NIPS*, 2009.

[8] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[9] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[10] A. Hyvarinen and E. Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

[11] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *International Conference on Computer Vision*, 2009.

[12] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. Lecun. Learning convolutional feature hierarchies for visual recognition. In *Advances in Neural Information Processing Systems*, 2010.

[13] A. Krizhevsky. Learning multiple layers of features from Tiny Images. Master's thesis, Dept. of Comp. Sci., University of Toronto, 2009.

[14] A. Krizhevsky. Convolutional Deep Belief Networks on CIFAR-10. Unpublished manuscript, 2010.

[15] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition*, 2006.

[16] Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR*, 2004.

[17] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *NIPS*, 2007.

[18] H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief net model for visual area V2. In *NIPS*, 2008.

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

[20] V. Nair and G. E. Hinton. 3D object recognition with deep belief nets. In *NIPS*, 2009.

[21] E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In *European Conference on Computer Vision*, 2006.

[22] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[23] M. Ranzato and G. E. Hinton. Modeling Pixel Means and Covariances Using Factorized Third-Order Boltzmann Machines. In *Computer Vision and Pattern Recognition*, 2010.

[24] M. Ranzato, A. Krizhevsky, and G. E. Hinton. Factored 3-way Restricted Boltzmann Machines for Modeling Natural Images. In *AISTATS 13*, 2010.

[25] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *NIPS*, 2007.

[26] R. Salakhutdinov and G. E. Hinton. Deep Boltzmann Machines. In *AISTATS 12*, 2009.

[27] R. Uetz and S. Behnke. Large-scale object recognition with CUDA-accelerated hierarchical neural networks. In *Intelligent Computing and Intelligent Systems*, 2009.

[28] J. C. van Gemert, J. M. Geusebroek, C. J. Veenman, and A. W. M. Smeulders. Kernel codebooks for scene categorization. In *European Conference on Computer Vision*, 2008.

[29] M. Varma and A. Zisserman. A statistical approach to material classification using image patch exemplars. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006.

[30] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

[31] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *International Conference on Computer Vision*, 2005.

[32] J. Yang, K. Yu, Y. Gong, and T. S. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Vision and Pattern Recognition*, 2009.

[33] K. Yu and T. Zhang. Improved local coordinate coding using local tangents. In *International Conference on Machine Learning*, 2010.