What is the Best Multi-Stage Architecture for Object Recognition?

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato and Yann LeCun
The Courant Institute of Mathematical Sciences
New York University, 715 Broadway, New York, NY 10003, USA
koray@cs.nyu.edu

Abstract

In many recent object recognition systems, feature extraction stages are generally composed of a filter bank, a non-linear transformation, and some sort of feature pooling layer. Most systems use only one stage of feature extraction in which the filters are hard-wired, or two stages where the filters in one or both stages are learned in supervised or unsupervised mode. This paper addresses three questions: 1. How does the non-linearities that follow the filter banks influence the recognition accuracy? 2. does learning the filter banks in an unsupervised or supervised manner improve the performance over random filters or hard-wired filters? 3. Is there any advantage to using an architecture with two stages of feature extraction, rather than one? We show that using non-linearities that include rectification and local contrast normalization is the single most important ingredient for good accuracy on object recognition benchmarks. We show that two stages of feature extraction yield better accuracy than one. Most surprisingly, we show that a two-stage system with random filters *can yield almost 63% recognition rate on Caltech-101, provided that the proper non-linearities and pooling layers are used. Finally, we show that with supervised refinement, the system achieves* state-of-the-art *performance on NORB dataset* (5.6%) *and unsupervised pre-training followed by supervised refinement produces good accuracy on Caltech-101* ($> 65\%$)*, and the lowest known error rate on the undistorted, unprocessed MNIST dataset* (0.53%)*.*

1. Introduction

Over the last few years, considerable efforts have been devoted to designing appropriate feature descriptors for object recognition. Many recent proposals use dense features extracted on regularly-spaced patches over the input image. The vast majority of these systems use a feature extraction process composed of a filter bank (generally based on oriented edge detectors), a non-linear operation (quantization, winner-take-all, sparsification, normalization, and/or point-wise saturation), and a pooling operation that combines nearby values in real space or feature space through a max, average, or histogramming operator. For example, the SIFT operator applies oriented edge filters to a small patch and determines the dominant orientation through a winner-take-all operation. Finally, the resulting sparse vectors are added (pooled) over a larger patch to form local orientation histograms. Several recognition architectures use a single stage of such features followed by a supervised classifier. Particular embodiments of the single-stage systems use SIFT features [19, 13], HoG [6], Geometric Blur [5], and models inspired by the architecture of the mammalian primary visual cortex [24], to mention a few. Other models use two or more successive stages of such feature extractors, followed by a supervised classifier. This includes convolutional networks globally trained in purely supervised mode with gradient descent [10], convolutional networks trained in supervised mode with an auxiliary task [3], or trained in purely unsupervised mode [25, 11, 18]. Multi-stage systems also include HMAX-type models [28, 22] in which the first layer is hardwired with Gabor filters, and the second layer is trained in unsupervised mode by storing randomly-picked output configurations from the first stage into filters of the second stage. All of these models essentially differ by whether they have one or two (or more) feature extraction stages, by the type of non-linearity used after the filter banks, the method used to pick the filters (hard-wired, unsupervised, supervised), and the top-level classifier (linear or more sophisticated).

This paper addresses three questions: 1. How do the non-linearities that follow the filter banks influence the recognition accuracy? 2. Does learning the filter banks in an unsupervised or supervised manner improve the performance over hard-wired filters or even random filters? 3. Is there any advantage to using an architecture with two successive stages of feature extraction, rather than with a single stage? To address these questions, we experimented with various combinations of architectures (with 1 or 2 stages of feature extraction), non-linearities, filter types, filter learning methods (random, unsupervised and supervised). We use a recently-proposed unsupervised feature learning method called Predictive Sparse Decomposition (PSD), based on an encoder-decoder architecture with sparsity constraints on the feature vector [12]. Results are presented on the well-known Caltech-101 dataset [7], on the NORB object dataset [15], and on the MNIST dataset of handwritten digits [14].

At first glance, one may think that training a complete system in a purely supervised manner (using gradient descent) is bound to fail on dataset with small number of labeled samples such as Caltech-101, because the number of parameters greatly outstrips the number of samples. One may also think that the filters need to be carefully hand-picked (or trained) to produce good performance, and that the details of the non-linearity play a somewhat secondary role. These intuitions, as it turns out, are wrong.

1.1. Modules for dense feature extraction

A common choice for the filter bank of the first stage is Gabor Wavelets [28, 22, 24]. Other proposals use simple oriented edge detection filters such as gradient operators, including SIFT [19], and HoG [6]. Another set of methods learn the filters by adapting them to the statistics of the input data with unsupervised learning [25, 11, 18]. When trained on natural images these filters are Gabor-like edge detectors. The advantage of learning methods is that they provide a way to learn the filters in the subsequent stages of the feature hierarchy. While prior knowledge about image statistics point to the usefulness of oriented edge detectors at the first stage, there is no similar prior knowledge that would allow to design sensible filters for the second stage in the hierarchy. Hence the second stage *must be learned*. A number of methods have been proposed to learn filters in a multi-stage vision system. The simplest method, which is a kind of patch memorization, is to set the filters to randomly-picked configurations of outputs of the previous stage [28, 22]. One of the oldest methods is to simply learn the filters in a supervised fashion using gradient descent [14, 10, 3]. The main issue with the purely supervised global training approach is that the number of parameters to be adjusted is very large, perhaps too large relative to the available number of training samples for most applications. Finally, one can train the filters in an unsupervised fashion by following the so-called "deep belief network" strategy [8, 4, 26, 9, 25, 17]: the filters are trained so that representations at one stage can be reconstructed from the representation at the next stage under sparsity constraints [25, 11] or using the so-called contrastive divergence method [18]. The main problem with the unsupervised approach is that the filters are learned independently of the task, although a few authors have proposed methods that combine unsupervised and supervised criteria to alleviate the problem [21, 27, 4].

The second ingredient of a feature extraction system is the non-linearity. Convolutional networks use a simple point-wise sigmoid function after the filter banks [14], while models that are strongly inspired by biology have included rectifying non-linearities, such as positive part, absolute value, or squaring functions [24], often followed by a local contrast normalization [24], which is inspired by divisive normalization models [20]. SIFT uses a rectification followed by winner-take-all operation over orientation, which is an extreme form of normalization. The last step is the pooling layer that can be applied over space [14, 13, 25, 3], over scale and space [28, 22, 24], or over similar feature types and space [11]. This layer builds robustness to small distortions by computing an average or a max of the filter responses within the pool.

The accuracy of single-stage systems on the Caltech-101 dataset, after training on 30 labeled samples per category varies with the details of the architecture and the filters. SIFT-based systems yield accuracies around 50% when fed to linear classifiers [11], and around 65% when using more sophisticated classifiers such as the Pyramid Match Kernel SVM (PMK-SVM) [13, 31, 11]. The V1-like model of Pinto et al. yields around 60% with a linear classifier following PCA [24]. These methods are similar in the fact that they use hand-crafted oriented edge filters.

In recent years, a few authors have experimented with filter-learning methods on Caltech-101. Kavukcuoglu et al. [11] report recognition rates similar to SIFT using a single-stage feature extractor fed to either a linear classifier or a PMK-SVM. Several authors have proposed systems with two stages of learned feature extractors, each of which comprises filter banks, non-linearities, and pooling. This includes convolutional networks using supervised training [10] and unsupervised training [25] yielding recognition rates in the mid 50's, and supervised training using auxiliary "pseudo-tasks" to regularize the system [3] yielding 67.2% recognition rate. HMAX-type architectures have yielded rates in the mid-40's to mid-50's [28, 22], and stacked Restricted Boltzmann Machines [17, 18] have yielded 65.4% with a PMK-SVM classifier on top. While the best results on Caltech-101 have been obtained by combining a large number of different feature families [29], the present study concerns systems with a single feature family, hence results will be compared with other work in which a single feature family is used. Better absolute numbers can be obtained by combining the features presented here with others, as described in [29].

2. Model Architecture

This section describes how to build a hierarchical feature extraction and classification system with fast (feed-forward) processing. The hierarchy stacks one or several feature extraction stages, each of which consists of filter bank layer, non-linear transformation layers, and a pooling layer that combines filter responses over local neighborhoods using an average or max operation, thereby achieving invariance to small distortions.

Filter Bank Layer - F_{CSG}: the input of a filter bank layer is a 3D array with n_1 2D *feature maps* of size $n_2 \times n_3$. Each component is denoted x_{ijk}, and each feature map is denoted x_i. The output is also a 3D array, y composed of m_1 feature maps of size $m_2 \times m_3$. A filter in the filter bank k_{ij} has size $l_1 \times l_2$ and connects input feature map x_i to output feature map y_j. The module computes:

$$y_j = g_j \tanh(\sum_i k_{ij} * x_i) \tag{1}$$

where $\tanh$ is the hyperbolic tangent non-linearity, $*$ is the 2D discrete convolution operator and g_j is a trainable scalar coefficient. By taking into account the borders effect, we have $m_1 = n_1 - l_1 + 1$, and $m_2 = n_2 - l_2 + 1$. This layer is denoted by F_{CSG} because it is composed of a set of convolution filters (C), a sigmoid/tanh non-linearity (S), and gain coefficients (G). In the following, superscripts are used to denote the size of the filters. For instance, a filter bank layer with 64 filters of size 9x9, is denoted as: $64F_{CSG}^{9\times9}$.

Rectification Layer - R_{abs}: This module simply applies the absolute value function to all the components of its input: $y_{ijk} = |x_{ijk}|$. Several rectifying non-linearities were tried, including the positive part, and produced similar results.

Local Contrast Normalization Layer - N: This module performs local subtractive and divisive normalizations, enforcing a sort of local competition between adjacent features in a feature map, and between features at the same spatial location in different feature maps. The subtractive normalization operation for a given site x_{ijk} computes: $v_{ijk} = x_{ijk} - \sum_{ipq} w_{pq}.x_{i,j+p,k+q}$, where w_{pq} is a Gaussian weighting window (of size 9x9 in our experiments) normalized so that $\sum_{ipq} w_{pq} = 1$. The divisive normalization computes $y_{ijk} = v_{ijk}/\max(c, \sigma_{jk})$ where $\sigma_{jk} = (\sum_{ipq} w_{pq}.v^2_{i,j+p,k+q})^{1/2}$. For each sample, the constant c is set to the $mean(\sigma_{jk})$ in the experiments. The denominator is the weighted standard deviation of all features over a spatial neighborhood. The local contrast normalization layer is inspired by computational neuroscience models [24, 20].

Average Pooling and Subsampling Layer - P_A: The purpose of this layer is to build robustness to small distortions, playing the same role as the complex cells in models of visual perception. Each output value is $y_{ijk} = \sum_{pq} w_{pq}.x_{i,j+p,k+q}$, where w_{pq} is a uniform weighting window ("boxcar filter"). Each output feature map is then subsampled spatially by a factor S horizontally and vertically. In this work, we do not consider pooling over feature types, but only over the spatial dimensions. Therefore, the numbers of input and output feature maps are identical, while the spatial resolution is decreased. Disregarding the border effects in the boxcar averaging, the spatial resolution is decreased by the down-sampling ratio S in both directions, denoted by a superscript, so that, an average pooling layer with 4x4 down-sampling is denoted: $P_A^{4\times4}$.

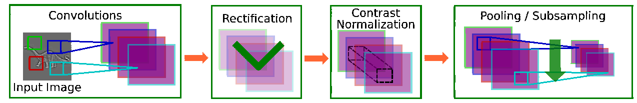

Figure 1. A example of feature extraction stage of the type $F_{CSG} - R_{abs} - N - P_A$. An input image (or a feature map) is passed through a non-linear filterbank, followed by rectification, local contrast normalization and spatial pooling/sub-sampling.

Max-Pooling and Subsampling Layer - P_M: building local invariance to shift can be performed with any symmetric pooling operation. The max-pooling module is similar to the average pooling, except that the average operation is replaced by a max operation. In our experiments, the pooling windows were non-overlapping. A max-pooling layer with 4x4 down-sampling is denoted $P_M^{4\times4}$.

2.1. Combining Modules into a Hierarchy

Different architectures can be produced by cascading the above-mentioned modules in various ways. An architecture is composed of one or two stages of feature extraction, each of which is formed by cascading a filtering layer with different combinations of rectification, normalization, and pooling. Recognition architectures are composed of one or two such stages, followed by a classifier, generally a multinomial logistic regression.

$\mathbf{F_{CSG} - P_A}$ This is the basic building block of traditional convolutional networks, alternating tanh-squashed filter banks with average down-sampling layers [14, 10]. A complete convolutional network would have several sequences of "F_{CSG} - P_A" followed by by a linear classifier.

$\mathbf{F_{CSG} - R_{abs} - P_A}$ The tanh-squashed filter bank is followed by an absolute value non-linearity, and by an average down-sampling layer.

$\mathbf{F_{CSG} - R_{abs} - N - P_A}$ The tanh-squashed filter bank is followed by an absolute value non-linearity, by a local contrast normalization layer and by an average down-sampling layer.

$\mathbf{F_{CSG} - P_M}$ This is also a typical building block of convolutional networks, as well as the basis of the HMAX and other architectures [28, 25], which alternate tanh-squashed filter banks with max-pooling layers.

3. Training Protocol

Given a particular architecture, a number of training protocols have been considered and tested. Each protocol is identified by a letter R, U, R^+, or U^+. A single letter (e.g. R) indicates an architecture with a single stage of feature extraction, followed by a classifier, while a double letter (e.g. RR) indicates an architecture with two stages of feature extraction followed by a classifier:

Random Features and Supervised Classifier - R and RR: The filters in the feature extraction stages are *set to random values and kept fixed* (no feature learning takes place), and the classifier stage is trained in supervised mode.

Unsupervised Features, Supervised Classifier - U and UU. The filters of the feature extraction stages are trained using the *unsupervised* PSD algorithm, described in section 3.1, and kept fixed. The classifier stage is trained in supervised mode.

Random Features, Global Supervised Refinement - $\mathbf{R^+}$ and $\mathbf{R^+R^+}$. The filters in the feature extractor stages are *initialized with random values*, and the *entire system (feature stages + classifier) is trained in supervised mode by gradient descent*. The gradients are computed using back-propagation, and all the filters are adjusted by stochastic online updates. This is identical to the usual method for training supervised convolutional networks.

Unsupervised Feature, Global Supervised Refinement - $\mathbf{U^+}$ and $\mathbf{U^+U^+}$. The filters in the feature extractor stages are *initialized with the PSD unsupervised learning algorithm*, and the *entire system (feature stages + classifier) is then trained (refined) in supervised mode by gradient descent*. The system is trained the same way as *random features with global refinement* using online stochastic updates. This is reminiscent of the "deep belief network" strategy in which the stages are first trained in unsupervised mode one after the other, and then globally refined using supervised learning [8, 4, 26]

For instance, a traditional convolutional network with a single stage initialized at random [14] would be denoted by an architecture motif like "$F_{CSG} - P_A$", and the training protocol would be denoted by R^+. The stages of a convolutional network with max-pooling would be denoted by "$F_{CSG} - P_M$". A system with two such stages trained in unsupervised mode, and the classifier (only) trained in supervised mode, as in [25], is denoted UU.

3.1. Unsupervised Training of Filter Banks using Predictive Sparse Decomposition

In order to learn the filter coefficients (g, k) in the filter bank layers (see eq. 1), an unsupervised learning algorithm is required. We used the Predictive Sparse Decomposition algorithm of [12], which has the following characteristics: 1. it produces efficient, feed-forward filter banks that include a point-wise non-linearity; 2. the training procedure is deterministic (no sampling required, as with Restricted Boltzmann Machines); 3. it learns to produce high-dimensional *sparse features*, which are suitable for subsequent pooling, and which enhance class discriminability. Although the filter banks are eventually applied to entire images, the PSD algorithm trains them on individual patches (or stacks of patches from multiple input feature maps) whose size is equal to the size of the filters. The starting point of PSD is the well-known sparse coding algorithm proposed by Olshausen and Field [23] which, unfortunately does not produce direct filters, but "reverse" filters (or dictionary elements). Inputs are approximated as a sparse linear combination of these dictionary elements. The coefficients constitute the feature representation. The method learns the optimal dictionary that can be used to reconstruct a set of training samples under sparsity constraints on the feature vector. For a given input X (a vectorized patch or stack of patches), and a matrix W whose columns are the dictionary elements, feature vector Z^* is obtained by minimizing the following energy function:

$$E_{OF}(X,Z,W) = \|X - WZ\|_2^2 + \lambda\|Z\|_1 \quad (2)$$

$$Z^* = \arg\min_Z E_{OF}(X,Z,W) \quad (3)$$

where λ is a sparsity hyper-parameter. Given a set of training samples $X^i, i = 1 \dots P$, learning proceeds by minimizing the loss $\mathcal{L}_{OF}(W) = 1/P \sum_{i=1}^{P} \min_z E_{OF}(X^i, Z, W)$ using stochastic gradient descent or a similar procedure. After learning, for any input X, one needs to run a rather expensive optimization algorithm to find Z^* (the so-called "basis pursuit" problem, which is convex, but non-quadratic [16, 2]). To alleviate the problem, the PSD method [12] trains a simple (feed-forward) regressor (or *encoder*) to approximate the sparse solution Z^* for all X in the training set. The regressor $C(X, K)$ takes the form of eq. 1 on a patch the size of the filters (K collectively denotes all the filter coefficients). During training, the feature vector Z^* is obtained by minimizing the energy function $E_{PSD}(X, Z, W, K)$, defined as follows:

$$E_{PSD} = \|X - WZ\|_2^2 + \lambda\|Z\|_1 + \|Z - C(X,K)\|_2^2 \quad (4)$$

$$Z^* = \arg\min_Z E_{PSD}(X,Z,W,K) \quad (5)$$

As with Olshausen and Field [23], learning proceeds by minimizing the loss $\mathcal{L}_{PSD}(W,K) = 1/P \sum_{i=1}^{P} \min_z E_{PSD}(X^i, Z, W, K)$. The learning procedure simultaneously optimizes W (dictionary) and K (filters). Once training is complete, the feature vector for a given input is simply obtained with $Z^* = C(X, K)$, hence the process is extremely fast (feed-forward).

4. Experiments

In this section, various architectures and training protocols are compared on the Caltech 101 [7], MNIST [1] and NORB [15] datasets. Our purpose is to determine whether two stages are better than one stage, which non-linearities are preferable, and which training protocol makes a difference.

Images from the Caltech-101 dataset were pre-processed with a procedure similar to [24]. The steps are: $1)$ converting to gray-scale (no color) and resizing to 151×151 pixels. $2)$ subtracting the image mean and dividing by the image standard deviation, $3)$ applying subtractive/divisive normalization (N layer with $c = 1$). $4)$ zero-padding the shorter side to 143 pixels.

Single Stage System: $[64.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R/N/P^{5\times5}}]$ - log_reg					
	$\mathbf{R_{abs} - N - P_A}$	$\mathbf{R_{abs} - P_A}$	$\mathbf{N - P_M}$	$\mathbf{N - P_A}$	$\mathbf{P_A}$
$\mathbf{U^+}$	54.2%	50.0%	44.3%	18.5%	14.5%
$\mathbf{R^+}$	54.8%	47.0%	38.0%	16.3%	14.3%
$\mathbf{U}$	52.2%	43.3%(±1.6)	44.0%	17.2%	13.4%
$\mathbf{R}$	53.3%	31.7%	32.1%	15.3%	12.1%(±2.2)
$\mathbf{G}$	52.3%				
Two Stage System: $[64.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R/N/P^{5\times5}}] - [256.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R/N/P^{4\times4}}]$ - log_reg					
	$\mathbf{R_{abs} - N - P_A}$	$\mathbf{R_{abs} - P_A}$	$\mathbf{N - P_M}$	$\mathbf{N - P_A}$	$\mathbf{P_A}$
$\mathbf{U^+U^+}$	65.5%	60.5%	61.0%	34.0%	32.0%
$\mathbf{R^+R^+}$	64.7%	59.5%	60.0%	31.0%	29.7%
$\mathbf{UU}$	63.7%	46.7%	56.0%	23.1%	9.1%
$\mathbf{RR}$	62.9%	33.7%(±1.5)	37.6%(±1.9)	19.6%	8.8%
$\mathbf{GT}$	55.8%				
Single Stage: $[64.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R_{abs}/N/P_A^{5\times5}}]$ - PMK-SVM					
$\mathbf{U}$	64.0%				
Two Stages: $[64.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R_{abs}/N/P_A^{5\times5}}] - [256.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R_{abs}/N}]$ - PMK-SVM					
$\mathbf{UU}$	52.8%				

Table 1. Average recognition rates on Caltech-101 with 30 training samples per class. Each row contains results for one of the training protocols, and each column for one type of architecture. All columns use an F_{CSG} as the first module, followed by the modules shown in the column label. The error bars for all experiments are within 1%, except where noted.

All results are recognition rates averaged over classes, after training with 30 samples per class, and averaged over 5 drawings of the training set. To adjust hyperparameters, a validation set of 5 samples per class was taken out of the training sets. The hyper-parameters were selected to maximize the performance on the validation set. Then, the system was trained over the entire training set. The final error value is computed as the average error over categories to account for differences in the number of instances per category (as is standard protocol for Caltech-101). The background category was also included.

Using a Single Stage of Feature Extraction: The first stage is composed of an F_{CSG} layer with 64 filters of size 9×9 ($64F^{9\times9}_{CSG}$), followed by an abs rectification (R_{abs}), a local contrast normalization layer (N) and an average pooling layer with 10×10 boxcar filter and 5×5 down-sampling ($P_A^{5\times5}$). The output of the first stage is a set of 64 features maps of size 26×26. This output is then fed to a multinomial logistic regression classifier that produces a 102-dimensional output vector representing a posterior distribution over class labels. Lazebnik's PMK-SVM classifier [13] was also tested.

Using Two Stages of Feature Extraction: In two-stage systems, the second-stage feature extractor is fed with the output of the first stage. The first layer of the second stage is an F_{CSG} module with 256 output features maps, each of which combines a random subset of 16 feature maps from the previous stage using 9×9 kernels. Hence the total number of convolution kernels is $256 \times 16 = 4096$. The average pooling module uses a 6×6 boxcar filter with a 4×4 down-sampling step. This produces an output feature map of size $256\times4\times4$, which is then fed to a multinomial logistic regression classifier. The PMK-SVM classifier was also tested.

Table 1 summarizes the results for the experiments.
1. The most astonishing result is that systems with *random filters and no filter learning whatsoever achieve decent performance* (53.3% for R and 62.9% for RR), as long as they include absolute value rectification and contrast normalization ($R_{abs} - N - P_A$).
2. Comparing experiments from rows R vs R^+, RR vs R^+R^+, U vs U^+ and UU vs U^+U^+, we see that supervised fine tuning consistently improves the performance, particularly with weak non-linearities: 62.9% to 64.7% for RR, 63.7% to 65.5% for UU using $R_{abs} - N - P_A$ and 35.1% to 59.5% for RR using $R_{abs} - P_A$.
3. It appears clear that two-stage systems (UU, U^+U^+, RR, R^+R^+) are systematically and significantly better than their single-stage counterparts (U, U^+, R, R^+). For instance, 54.2% obtained by U^+ compared to 65.5% obtained by U^+U^+ using $R_{abs} - N - P_A$. However, when using P_A architecture, adding second stage without supervised refinement does not seem to help. This may be due to cancellation effects of the P_A layer when rectification is not present.
4. It seems that unsupervised training (U, UU, U^+, U^+U^+) does not seem to significantly improve the performance (comparing with (R, RR, R^+, R^+R^+) if both rectification and normalization are used (62.9% for RR versus 63.7% for UU). When contrast normalization is removed, the performance gap becomes significant (35.1% for RR versus 47.8% for UU). If no supervised refinement is performed, it looks as if appropriate architectural components are a good

substitute for unsupervised training.

5. It is clear that abs rectification is a crucial component for achieving good performance, as shown with the U^+U^+ protocol by comparing columns $N - P_A$ (31.0%), $R_{abs} - P_A$ (60.0%), and $R_{abs} - N - P_A$ (65.5%). However, using max pooling seems to alleviate the need for abs rectification, confirming the hypothesis that average pooling without rectification falls victim to cancellation effects between neighboring filter outputs. This is an extremely important fact for users of convolutional networks, in which rectification has not been traditionally used.

6. A single-stage system with PMK-SVM reaches the same performance as a two-stage system with logistic regression (around 65%) as shown in the last two rows of Table 1. It looks as if the pyramid match kernel is able to play the same role as a second stage of feature extraction. Perhaps it is because PMK first performs a K-means based vector quantization, which can be seen as an extreme form of sparse coding, followed by local histogramming, which is a form of spatial pooling. Hence, the PM kernel is conceptually similar to a second stage based on sparse coding and pooling as recently pointed out in [30]. Furthermore, these numbers are similar to the performance of the original PMK-SVM system which used dense SIFT features (64.6%) [13]. Again, this is hardly surprising as the SIFT module is conceptually very similar to our feature extraction stage. When using features extracted using UU architecture, the performance of PMK-SVM classifier drops significantly. This behaviour might be caused by the very small spatial density (18×18) of features at second layer.

7. The last row of single stage system represents F_{CSG} kernels that are initialized with Gabor functions ($\mathbf{G}$). The last row of two stage system represents first layer gabor functions, followed by a second layer where kernels are initialized with templates from first layer outputs ($\mathbf{GT}$) as in the HMAX model [28, 22]. Suprisingly, the performance is considerably worse than with random filters.

4.1. NORB Dataset

Caltech-101 is very peculiar in that many objects have distinctive textures and few pose variations. More importantly, the particularly small size of the training set favors methods that minimize the role learning and maximize the role of clever engineering. A diametrically opposed object dataset is NORB [15]. The "Small NORB" dataset has 5 object categories (humans, airplanes, cars, trucks, animals) 5 object instances for training, and 5 different object instances for test. Each object instance has 972 samples (18 azimuths, 9 elevations, and 6 illuminations), for a total of 24300 training samples and 24300 test samples (4860 per class). Each image is 96×96 pixels, grayscale. Experiments were conducted to elucidate the importance of the non-linearity, and the performance of random filter systems when many labeled samples are available.

Figure 2. Several examples from NORB dataset

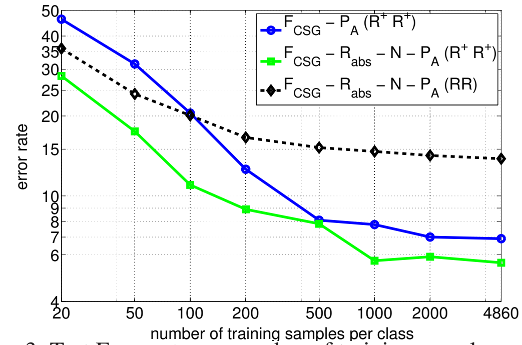

Figure 3. Test Error rate vs. number of training samples per class on NORB Dataset. Although pure random features perform surprisingly good when training data is very scarce, for large number of training data learning improves the performance significantly. Absolute value rectification (R_{abs}) and local normalization (N) is shown to improve the performance in all cases.

Only the RR and R^+R^+ protocols were used with 8 feature maps with 5×5 filters at the first stage, 4×4 average pooling followed by 24 feature maps with 6×6 filters, each of which combines input from 4 randomly picked stage-1 feature maps, followed by 3×3 average pooling. The last stage is a 5-way multinomial logistic regressor.

Several systems with various non-linearities were trained on subsets of the training set with 20, 50, 100, 200, 500, 1000, 2000, and 4860 training samples per class. The results are shown in figure 3 in log-log scale. The green curve (bottom) uses abs and normalization, while the blue curve (middle) uses neither. Both are trained in purely supervised mode from random initial conditions (R^+R^+). It appears that the use of abs and normalization makes a big difference when labeled samples are scarce, but the difference diminishes as the number of training samples increases. Training seems to compensate for architectural simplicity, or conversely architectural sophistication seems to compensate for lack of training. Still the error rate when trained on the full training set is 5.6% with abs and normalization, but 6.9% with neither ([15] reported 6.6%).

More interesting is the behavior of the system with random filters: While its error rate is comparable to that of the network trained in supervised mode for small training sets (in the "Caltech-101 regime"), the error rate remains high as samples are added. Hence, *while random filters perform well on Caltech-101, they would most likely not perform as well as learned filters on tasks with more labeled samples.*

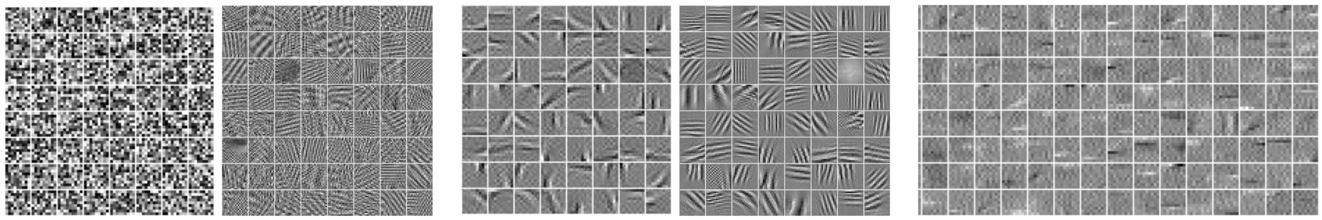

Figure 4. Left: random stage-1 filters, and corresponding optimal inputs that maximize the response of each corresponding complex cell in a $F_{CSG} - R_{abs} - N - P_A$ architecture. The small asymmetry in the random filters is sufficient to make them orientation selective. Middle: same for PSD filters. The optimal input patterns contain several periods since they maximize the output of a complete stage that contains rectification, local normalization, and average pooling with down-sampling. Shifted versions of each pattern yield similar activations. Right panel: subset of stage-2 filters obtained after PSD and supervised refinement on Caltech-101. Some structure is apparent.

4.2. Random Filter Performance

Perhaps the most astonishing result is the surprisingly good performance obtained with random filters with few labeled samples. The NORB experiments show that random filters yield sub-par performance when labeled samples are abundant. But the experiments also show that random filters seem to require the presence of abs and normalization. To explore why random filters work at all, we used gradient descent to find the *optimal input patterns* that maximize each complex cell (after pooling) in a $F_{CSG} - R_{abs} - N - P_A$ stage. The surprising finding is that the optimal stimuli for random filters are oriented gratings (albeit a noisy and faint ones), similar to the optimal stimuli for trained filters. As shown in fig 4, it appears that random weights, combined with the abs/norm/pooling creates a spontaneous orientation selectivity.

4.3. Handwritten Digits Recognition

As a sanity check for the overall training procedures and architectures, experiments were run on the MNIST dataset, which contains 60,000 gray-scale 28x28 pixel digit images for training and 10,000 images for testing. An architecture with two stages of feature extraction was used: the first stage produces 32 feature maps using 5×5 filters, followed by 2x2 average pooling and down-sampling. The second stage produces 64 feature maps, each of which combines 16 feature maps from stage 1 with 5×5 filters (1024 filters total), followed by 2×2 pooling/down-sampling. The classifier is a 2-layer fully-connected neural network with 200 hidden units, and 10 outputs. The loss function is equivalent to that of a 10-way multinomial logistic regression (also known as cross-entropy loss). The two feature stages use abs rectification and normalization.

The parameters for the two feature extraction stages are first trained with PSD as explained in Section 3.1. The classifier is initialized randomly. The whole system is fine-tuned in supervised mode (the protocol could be described as $(U^+U^+R^+R^+)$). A validation set of size 10,000 was set apart from the training set to tune the only hyper-parameter: the sparsity constant λ. Nine different values were tested between 0.1 and 1.6 and the best value was found to be 0.2. The system was trained with a form of stochastic gradient descent on the 50,000 non-validation training samples until the best error rate on the validation set was reached (this took 30 epochs). It was then tuned for another 3 epochs on the whole training set. A **test error rate of 0.53% was obtained**. To our knowledge, *this is the best error rate ever reported on the original MNIST dataset, without distortions or preprocessing*. The best previously reported error rate was 0.60% [26].

5. Conclusions

This paper addressed the following three questions:

1. how do the non-linearities that follow the filter banks influence the recognition accuracy. The surprising answer is that using a rectifying non-linearity is the single most important factor in improving the performance of a recognition system. This might be due to several reasons: a) the polarity of features is often irrelevant to recognize objects, b) the rectification eliminates *cancellations* between neighboring filter outputs when combined with average pooling. Without a rectification what is propagated by the average down-sampling is just the noise in the input. Also introducing a local normalization layer improves the performance. It appears to make supervised learning considerably faster, perhaps because all variables have similar variances (akin to the advantages introduced by whitening and other decorrelation methods)

2. does learning the filter banks in an unsupervised or supervised manner improve the performance over hard-wired filters or even random filters: the most surprising result is that *random filters* used in a two-stage system with the proper non-linearities yield 62.9% recognition rate on Caltech-101. Experiments on NORB show that this surprising performance is only seen in the limit of very small training set sizes. We have also shown that the optimal input patterns for a randomly initialized stage are very similar to the optimal inputs for a stage that use learned filters. The second important result is that global supervised learning of the filters yields good recognition rate if the proper non-linearities are used. It was thought that the dismal performance of supervised convolutional networks on Caltech-101 was due to overparameterization, but it seems to be due to an inadequate non-linear layer. The last interesting point is that unsupervised pre-training followed by supervised refinement yields the best overall accuracy, although the improvement over purely supervised is rather small.

3. is there any advantage to using an architecture with two successive stages of feature extraction, rather than with a single stage: the experiments clearly show that two stages are better than one. The performance of our two-stage system is similar to that of the best single-stage systems based on SIFT and PMK-SVM, perhaps because the PM Kernel is conceptually similar to our feature extractions stage.

There are reasons to hope that better learning methods with refined architectures and more powerful classifiers may yield even better accuracy. The ability to learn a hierarchy of filters allows us to apply the method to any signal or image type with strong local dependencies without having to rely on expert knowledge to produce appropriate filters.

References

[1] http://yann.lecun.com/exdb/mnist/.

[2] M. Aharon, M. Elad, and A. Bruckstein. K-svd and its non-negative variant for dictionary design. In *Proc. of the SPIE conference wavelets*, volume 5914, 2005.

[3] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo tasks. In *European Conference on Computer Vision*, 2008.

[4] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*. MIT Press, 2007.

[5] A. C. Berg, T. L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. In *CVPR*, 2005.

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. of Computer Vision and Pattern Recognition*, 2005.

[7] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPR Workshop*, 2004.

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

[9] G. Hinton and T. Sejnowski. *Unsupervised Learning: Foundations of Neural Computation*. MIT press, 1999.

[10] F.-J. Huang and Y. LeCun. Large-scale learning with svm and convolutional nets for generic object categorization. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press, 2006.

[11] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. LeCun. Learning invariant features through topographic filter maps. In *CVPR'09*. IEEE, 2009.

[12] K. Kavukcuoglu, M. Ranzato, and Y. LeCun. Fast inference in sparse coding algorithms with applications to object recognition. Technical report, Computational and Biological Learning Lab, Courant Institute, NYU, 2008. Tech Report CBLL-TR-2008-12-01.

[13] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, June 2006.

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

[15] Y. LeCun, F.-J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

[16] H. Lee, A. Battle, R. Raina, and A. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, 2006.

[17] H. Lee, E. Chaitanya, and A. Y. Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems*, 2007.

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

[19] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[20] S. Lyu and E. Simoncelli. Nonlinear image representation using divisive normalization. In *CVPR*, pages 1–8, 2008.

[21] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Discriminative learned dictionaries for local image analysis. In *CVPR*, 2008.

[22] J. Mutch and D. Lowe. Multiclass object recognition with sparse, localized features. In *CVPR*, 2006.

[23] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: a strategy employed by v1? *Vision Research*, 37:3311–3325, 1997.

[24] N. Pinto, D. D. Cox, and J. J. DiCarlo. Why is real-world visual object recognition hard? *PLoS Computational Biology*, 4(1), 2008.

[25] M. Ranzato, F. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007.

[26] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *NIPS 2006*. MIT Press, 2006.

[27] M. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. In *Internal Conference of Machine Learning*, 2008.

[28] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *CVPR*, 2005.

[29] M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *ICCV*, 2007.

[30] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009.

[31] H. Zhang, A. C. Berg, M. Maire, and J. Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *CVPR*, 2006.