Computer Science Technical Report TR2010-935, December 2010
Courant Institute of Mathematical Science, New York University

Learning Image Decompositions with Hierarchical Sparse Coding

Matthew D. Zeiler and Rob Fergus
Dept. of Computer Science, Courant Institute, New York University
{zeiler,fergus}@cs.nyu.edu

Abstract

We present a hierarchical model that learns image decompositions via alternating layers of convolutional sparse coding and max pooling. When trained on natural images, the layers of our model capture image information in a variety of forms: low-level edges, mid-level edge junctions, high-level object parts and complete objects. To build our model we rely on a novel inference scheme that ensures each layer reconstructs the input, rather than just the output of the layer directly beneath, as is common with existing hierarchical approaches. This scheme makes it possible to robustly learn multiple layers of representation and we show a model with 4 layers, trained on images from the Caltech-101 dataset. We use our model to produce image decompositions that, when used as input to standard classification schemes, give a significant performance gain over low-level edge features and yield an overall performance competitive with leading approaches.

1. Introduction

Discovering good representations for images is the key to recognizing the objects they contain. Local region descriptors such as SIFT and HOG have catalyzed dramatic improvements in recognition performance over the last few years. However, these descriptors capture limited information from the image; essentially finding edges and then pooling them. Building higher-order representations that capture corners, junctions and common object parts would have great potential to improve recognition performance.

Many leading object detection algorithms use multi-layered object representations, most notably the approach of Felzsenszwalb *et al.* [3]. In this framework, the model parts exist at multiple scales with the dependencies between them capturing their relative location. The local appearance for each part relies on a discriminatively trained filter to match within a HOG pyramid. However, given the latent part positions, the model is linear in its parameters which limits its complexity and hence performance.

Multi-layered non-linear models are potentially far more powerful but are difficult to train effectively. The most obvious example is the Convolutional Neural Network (ConvNets) [10] which performs well on certain tasks such as classifying handwritten digits, but less so on more challenging data. The reasons are two-fold: (i) they still suffer from the problem of vanishing gradients[1], which prevents very deep networks from being learned; (ii) they must be trained in a supervised manner and have many parameters, thus require an inconveniently large amount of labeled data. The latter objection has been overcome by Deep Belief Networks (DBNs) [6] which incorporate an unsupervised pre-training phase to initialize the network parameters. Convolutional DBNs [11] have shown promising results, but are still difficult to train. While ConvNets use back-propagation to minimize the loss function relative to the input, the unsupervised training schemes, such as contrastive divergence, cannot do this. Instead, each layer of the DBN attempts to reconstruct the output of the layer below and the model is built greedily in a layer-by-layer fashion. The problem with this approach is that the upper layers of the model have no direct connection back to the input, thus the reconstruction errors accumulate as the number of layers increases.

In this paper we introduce a novel hierarchical model that is non-linear and overcomes many of the issues outlined above. Instead of using explicit nonlinear functions in the model, we instead use sparsity [12, 13] to iteratively reweight a linear system, thus generating non-linear behavior. This formulation has a number of advantages: (i) we can train each layer's loss function directly with respect to the input, irrespective on the number of layers; (ii) the training is unsupervised and efficient, being predominantly based on linear conjugate gradients; (iii) nevertheless, the resulting model is highly non-linear; (iv) it a top-down generative model which performs explaining away at each layer.

Each layer of the model captures information at different image scales, ranging from edges up to entire scenes. Correspondingly, the model features at each layer increase in sophistication, from edges, to corners, curves and junctions, to model parts and finally to entire objects and scenes.

[1] ConvNets and neural networks use multiple layers of saturating non-linearities. When back-propagating through them the gradients become so attenuated that they do not reach more than a few layers from the top, thus the first layers of the network remain untrained.

The structure of each layer is based on a Deconvolutional Network [18], which is essentially a convolutional form of sparse coding [2, 8]. However, the overall topology, training and inferences schemes differ significantly from those used in [18].

Our model performs a decomposition of the full image, in the spirit of Zhu and Mumford [20] and Tu and Zhu [16]. This differs from other hierarchical models, such as Fidler and Leonardis [4] and Zhu *et al.* [19], that only model a stable sub-set of image structures at each level, rather than all pixels. Another key aspect of our approach is that we learn the decomposition from natural images. Several other hierarchical models such as the HMax model of Serre *et al.* [14, 15] and Guo *et al.* [5] use hand-crafted features at each layer.

2. Approach

Our model produces an over-complete image representation that can be used as input to standard object classifiers. Unlike many image representations, ours is learned from natural images and, given a new image, requires inference to compute. The model decomposes an image in a hierarchical fashion using multiple alternating layers of convolutional sparse coding (deconvolution [18]) and max-pooling. Each of the deconvolution layers attempts to directly minimize the reconstruction error of the input image under a sparsity constraint on an over-complete set of feature maps. The cost function $C_l(y)$ for layer l comprises two terms: (i) a likelihood term that keeps the reconstruction of the input $\hat{y}_l$ close to the original input image y; (ii) a regularization term that penalizes the ℓ_1 norm of the 2D feature maps $z_{k,l}$ on which the reconstruction $\hat{y}_l$ depends. The weighting between the two terms is controlled by λ_l:

$$C_l(y) = \frac{\lambda_l}{2}\|\hat{y}_l - y\|_2^2 + \sum_{k=1}^{K_l} |z_{k,l}|_1 \quad (1)$$

Unlike existing approaches [2, 8, 18], our convolutional sparse coding layers attempt to directly minimize the reconstruction error of the input image, rather than the output of the layer below.

Deconvolution: Consider the first layer of the model, as shown in Fig. 1. The reconstruction $\hat{y}_1$ (comprised of c color channels) is formed by convolving each of the 2D feature maps $z_{k,1}$ with filters $f_{k,1}^c$ and summing them:

$$\hat{y}_1^c = \sum_{k=1}^{K_1} z_{k,1} * f_{k,1}^c \quad (2)$$

where $*$ is the 2D convolution operator. The filters f are the parameters of the model common to all images. The feature maps z are latent variables, specific to each image. Since $K_1 > 1$ the model is over-complete, but the regularization term in Eqn. 1 above ensures that there is a unique solution. We describe the inference scheme used to discover the optimal z_1 and the closely related learning approach for estimating f_1 in Sections 2.1 and 2.2 respectively. For notational brevity, we combine the convolution and summing operations of layer l into a single matrix F_l and convert the multiple 2D maps $z_{k,l}$ into a single vector z_l:

$$\hat{y}_1 = F_1 z_1 \quad (3)$$

Pooling: On top of each deconvolutional layer, we perform a max-pooling operation on the feature maps z from the layer below. This pooling occurs both spatially (within each 2D z map) and also between adjacent maps. In the case of the first layer, each element in the pooled map $p_{j,1}(x,y)$[2] is the absolute maximum (preserving sign) over a neighborhood $\mathtt{N}(x,y,j)$ in z_1. A typical neighborhood is 3×3 spatially (non-overlapping) and 2 in the k dimension:

$$[p_{j,1}(x,y), s_{j,1}(x,y)] = \max_{\forall x',y',k' \in \mathtt{N}(x,y,j)} |z_{k',1}(x',y')| \text{sign}(z_{k',1}(x',y')) \quad (4)$$

The location of the maxima within each pooling region is recorded in switch variables s_1. For brevity we express the pooling operation in matrix form, where $p_1 = P_{s_1} z_1$, P_{s_1} being a binary selection matrix (for switch settings s_1). The corresponding unpooling operation $U_{s_1} = P_{s_1}^T$ takes the elements in p_1 and places them in z_1 at the locations specified by s_1, the remaining elements being set to zero: $\hat{z}_1 = U_{s_1} p_1$. The pooling operations mean that as we ascend the model each element in the feature maps can reconstruct larger and larger regions of the input image.

Multiple Layers: The architecture remains the same for higher layers in the model but the number of feature maps K_l may vary. At each layer we reconstruct the input through the filters and switches of the layers below. We define a reconstruction operator R_l that takes feature maps z_l from layer l and alternately convolves (F) and unpools them (U_s) down to the input:

$$\hat{y}_l = F_1 U_{s_1} F_2 U_{s_2} \ldots F_l z_l = R_l z_l \quad (5)$$

Note that $\hat{y}_l$ depends only on the feature maps z_l, not the maps from any lower layers. In other words, when we project down to the image, we do not impose sparsity on any of the intermediate layer reconstructions $\hat{z}_{l-1}, \ldots, \hat{z}_1$. However, the reconstruction operator R_l does depend on the pooling switches in the intermediate layers $(s_{l-1} \ldots s_1)$ since they determine the unpooling operations $U_{s_{l-1}} \ldots U_{s_1}$. The switches are configured by the values of $z_{l-1} \ldots z_1$ from previous iterations. Fig. 1 illustrates two layers of deconvolution and pooling within our model.

2.1. Inference

For a given layer l, inference involves finding the feature maps z_l that minimize $C_l(y)$, given an input image y and filters f. The model structure outlined above is designed

[2] j is the map index $(1 \ldots K)$ and x, y are the spatial coordinates.

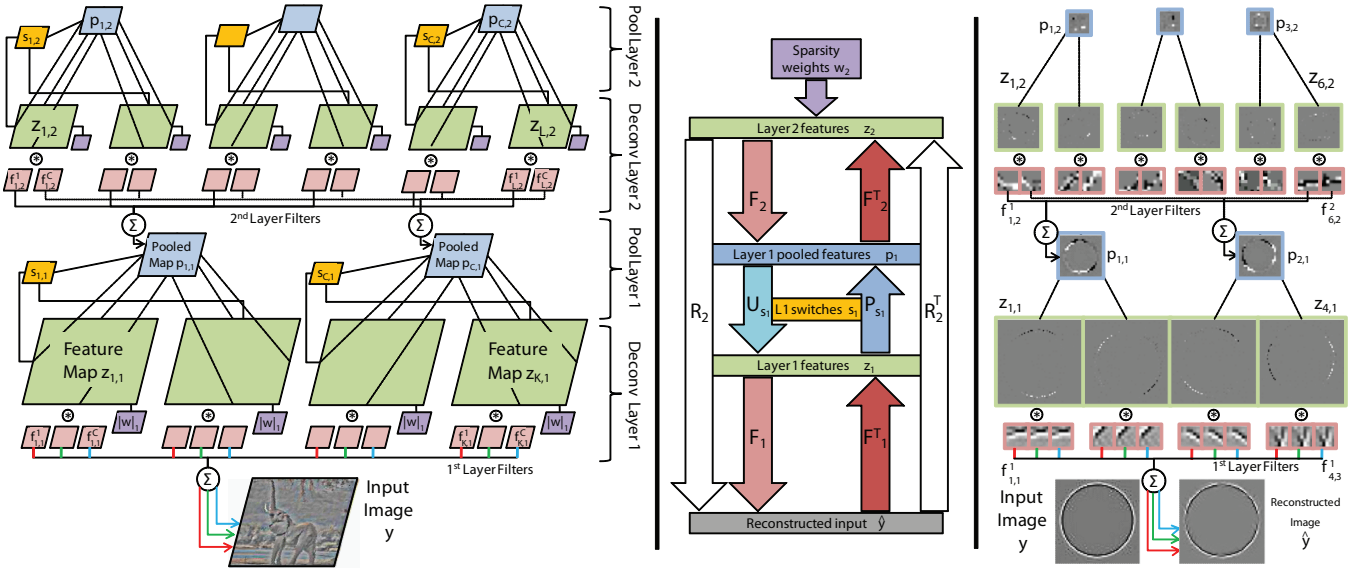

Figure 1. Left: A visualization of two layers of our model. Each layer consists of a deconvolution and a max-pooling. The deconvolution layer is a convolutional form of sparse coding that decomposes input image y into feature maps z_1 (green) and learned filters f_1 (red), which convolve together and sum to reconstruct y. The filters have c planes, each used to reconstruct a different channel of the input image. Each z map is penalized by a per-element ℓ_1 sparsity term (purple). The max-pooling layer pools within and between feature maps, reducing them in size and number to give pooled maps p (blue). The locations of the maxima in each pooling region are recorded in switches s (yellow). The second deconvolution/pooling layer is conceptually identical to the first, but now has two input channels rather than three. In practice, we have many more feature maps per layer and have up to 4 layers in total. Middle: A block diagram view of the inference operations within the model for layer 2. See Section 2.1 for explanation. Right: A toy instantiation of the model on the left, trained using a single input image of a (contrast-normalized) circle. The switches and sparsity terms are not shown. Note the sparse feature maps (green) and effect of the pooling operations (blue). Since the input is grayscale, the planes of the 1st layer filters are identical.

to make inference tracible for large models with many hundreds of feature maps.

For each layer we need to solve a large ℓ_1 convolutional sparse coding problem and we adapt the continuation approach of Zeiler *et al.* [18]. This introduces a set of auxiliary variables w (one for each element in z) to separate the likelihood and regularization terms:

$$C_l(y) = \frac{\lambda_l}{2}\|\hat{y}_l - y\|_2^2 + \sum_{k=1}^{K_l} |z_{k,l}|_1 \quad (6)$$

$$= \frac{\lambda_l}{2}\|\hat{y}_l - y\|_2^2 + \frac{\beta_l}{2}\sum_{k=1}^{K_l}\|z_{k,l} - w_{k,l}\|_2^2 + \sum_{k=1}^{K_l}|w_{k,l}|_1 \quad (7)$$

where β_l is the continuation parameter. $C_l(y)$ can then be minimized in an alternating fashion. First we fix $z_{k,l}$ to give a separable 1D problem for each element in $w_{k,l}$ (**w-subproblem**). Then we fix $w_{k,l}$ and solve for $z_{k,l}$, which is a quadratic problem that can be solved very efficiently (**z-subproblem**). Starting with a small value of β_l, we alternate between the two subproblems, increasing β_l until it strongly ties $z_{k,l}$ to $w_{k,l}$.

w-subproblem: Given fixed $z_{k,l}$, each element of the optimal $w_{k,l}$ can be found in closed form:

$$w_{k,l} = \max(|z_{k,l}| - \frac{1}{\beta_l}, 0)\frac{z_{k,l}}{|z_{k,l}|} \quad (8)$$

z-subproblem: Given fixed $w_{k,l}$, we need to minimize the following expression with respect to z_l:

$$\frac{\lambda_l}{2}\|\hat{y}_l - y\|_2^2 + \frac{\beta_l}{2}\sum_{k=1}^{K_l}\|z_{k,l} - w_{k,l}\|_2^2 \quad (9)$$

Since our model contains no explicit non-linearities, the derivative of $\hat{y}_l$ with respect to z_l is linear, given fixed switch settings $s_1 \ldots s_{l-1}$ and is equal to R_l^T:

$$\frac{\partial \hat{y}_l}{\partial z_l} = F_l^T P_{s_{l-1}} \ldots F_2^T P_{s_1} F_1^T = R_l^T \quad (10)$$

Intuitively, R_l^T takes a signal at the input and alternately filters it (F^T) and pools it (P_s) up to layer l. Taking derivatives of Eqn. 9 and setting equal to zero, we obtain the following linear system in z_l:

$$(\lambda_l R_l^T R_l + \beta_l I) z_l = \lambda_l R_l^T y + \beta_l w_l \quad (11)$$

The left-hand side of the system is never explicitly formed. Instead, we compute the matrix-vector product by mapping z_l back to the input (via R_l) and then passing it forward again up to layer l (via R_l^T), and finally adding β_l to all elements. The right hand side of the system is formed by propagating y upto layer l and adding the per-element weights $\beta_l w_l$. Both the reconstruction R and propagation R^T operations are very quick, just consisting of convolutions, summations, pooling and unpooling operations, all of which are

amenable to parallelization. This makes it possible to efficiently solve the system in Eqn. 11 using linear conjugate gradients (CG), even with massively over-complete layers where z_l may be up to 10^5 in length.

While the z-subproblem is linear, the model as a whole is not. The non-linearity arises from two sources: (i) sparsity, as induced in the w-subproblem, and (ii) the settings of the switches s which change the pooling/unpooling within R_l.

We perform both inference and learning in a layer-by-layer fashion, starting at the bottom. As we move up, the filters and switches of layers below the current one are held fixed. During inference, after solving the z-subproblem (Eqn. 11), we update the switch settings for the current layer using Eqn. 4. Additionally, since we want a reconstruction that is consistent with these switch settings (for layers above), we perform a pool/unpool operation on z_l:

$$z_l \leftarrow U_{s_l} P_{s_l} z_l \tag{12}$$

2.2. Learning

In learning the goal is to estimate the filters f in the model, which are shared across all images $Y = \{y^1, \ldots y^i, \ldots, y^N\}$. For a given layer l, we perform inference to compute z_l^i. Taking derivatives of Eqn. 1 and setting to zero, we obtain the following linear system in f_l:

$$\sum_{i=1}^{N} {z_l^i}^T {P^i_{s_{l-1}}}^T {R^i}^T_{l-1} \hat{y}^i = \sum_{i=1}^{N} {z_l^i}^T {P^i_{s_{l-1}}}^T {R^i}^T_{l-1} y^i \tag{13}$$

where $\hat{y}^i$ is the reconstruction of the input using the current value of f_l. We solve this system using linear conjugate gradients. As with inference, the matrix-vector product of left-hand side is computed efficiently by mapping down to the input and back up again using the R_l and R_l^T operations. After solving Eqn. 13, we normalize f_l to have unit length.

The overall algorithm for learning all layers of the model is given in Algorithm 1. The procedure for inference is identical, except the f_l update on line 14 is not performed. In practice, we find that just two CG iterations for the z_l and f_l systems (lines 7,14) are sufficient, rather than solving to convergence.

2.3. Relation to existing models

We now explain how our model differs from other feature learning approaches. The key differences are: (i) we train each layer to reconstruct the input y, not the layer immediately below (i.e. z_{l-1}) like most other approaches; (ii) by careful design of our model and its inference scheme, we are able to do this efficiently while keeping the model non-linear. The first difference is vital in practice, since without it the reconstruction constraint is too weak to drive the learning of good filters. The second difference makes it possible train models with a large number of feature maps, needed for a good representation.

Algorithm 1 Learning Image Decompositions.

Require: Training set Y, # layers L, # epochs E
Require: Regularization weights λ_l, # feature maps K_l
Require: Continuation parameters: $\beta_l^{\text{Inc}}, \beta_l^{\text{Max}}$
1: **for** $l = 1 : L$ **do**
2: Init. features/filters: $z_l^i \sim \mathcal{N}(0, \epsilon)$, $f_l \sim \mathcal{N}(0, \epsilon)$, $w_l^i = 0$
3: **for** epoch $= 1 : E$ **do**
4: **for** $i = 1 : N$ **do**
5: $\beta_l = 1/\lambda_l$
6: **while** $\beta_l < \beta_l^{\text{Max}}$ **do**
7: Update z_l^i by solving Eqn. 11 (z-prob) using CG
8: Update w_l^i using Eqn. 8 (w-prob)
9: Update switches s_l^i using Eqn. 4
10: Pool/unpool z_l^i, using Eqn. 12
11: $\beta_l = \beta_l \cdot \beta_l^{\text{Inc}}$
12: **end while**
13: **end for**
14: Update f_l by solving Eqn. 13 using CG
15: **end for**
16: **end for**
17: Output: filters f, feature maps z and switches s.

Convolutional Networks [10] apply the filters f to the image y, rather than the feature maps z as we do. In other words, each layer in a convolutional network is bottom-up (feed-forward), while ours is top-down (generative). Although slower, each of our sparse coding layers performs explaining away, thus has more modeling power than a linear transformation plus non-linearity. These non-linearities, which complicate training significantly in ConvNets, are absent in our model. Finally, ConvNets are purely supervised, while our approach is unsupervised. Predictive Sparse Decomposition [7] adds a sparse coding component to ConvNets that allows unsupervised training, however each layer only reconstructs the layer beneath, unlike in our model.

Deep Belief Networks [6, 11], composed of multiple layers of Restricted Boltzmann Machines, are a popular approach with similarities to ours. However, each RBM layer has a factored representation that does not directly perform explaining away. Training is relatively slow and they only reconstruct the output of the layer below.

The closest approaches to ours are those based on convolutional sparse coding [2, 8, 18]. Our deconvolutional layers are the same as those used in Zeiler *et al.* [18], but their model does not include any form of pooling, and they only attempt to reconstruct the layer below. Furthermore, to assist training it is common [8, 10, 18] to manually impose sparse connectivity between the feature maps of different layers. In contrast, our model has full connectivity between layers, which allows our model to learn more complex structures.

3. Application to object recognition

Our model is purely unsupervised and so must be combined with a classifier to perform object recognition. In view of its simplicity and performance, we use the Spatial Pyramid Matching (SPM) of Lazebnik *et al.* [9].

Given a new image, performing inference with our model decomposes it into multiple layers of feature maps and switch configurations. We now describe a novel approach for using this decomposition in conjunction with the SPM classifier. While the filters are shared between images, the switch settings are not, thus the feature maps of two images are not directly comparable since they use different bases R_l. This makes direct use of the higher-level feature maps problematic and we propose a different approach.

We examine the activations of single elements in the feature maps in the top level of the model (by projecting them back to the input pixel domain), and these produce a soft decomposition of the original image. Shown in Fig. 3, they contain good reconstructions of select image structures, as extracted by the model, while neighboring content is suppressed. For each image i, we take the set of M largest absolute activations from the top layer feature maps and project each one *separately* down to the input to create M different images $(\hat{y}^{i,1}, \ldots, \hat{y}^{i,M})$, each containing various image parts generated by our model. This is only practical for high layers with large receptive fields.

Instead of directly inputting $\hat{y}^{i,1}, \ldots, \hat{y}^{i,M}$ to the SPM, we instead use the corresponding reconstructions of the 1st layer feature maps (i.e. $\hat{z}_1^{i,1}, \ldots, \hat{z}_1^{i,M}$), since activations at this layer are roughly equivalent to unnormalized SIFT features (the standard SPM input [9]). After computing separate pyramids for each $\hat{z}_1^{i,m}$, we average all M of them to give a single pyramid for each image. We can also apply SPM to the actual 1st layer feature maps z_1^i, which are far denser and have even coverage of the image[3]. The pyramids of the two can be combined to boost performance.

4. Experiments

We train our model on the entire training set of 3060 images from the Caltech 101 dataset (30 images per class).

Pre-processing: Each image is converted to gray-scale and resized to 150×150 (zero padding to preserve the aspect ratio). Local subtractive and divisive normalization (i.e. the patch around each pixel should have zero mean and unit norm) is applied using a 13×13 Gaussian filter with $\sigma = 5$.

Model architecture: We use a 4 layer model, with 7×7 filters, $E = 10$ epochs of training, $\beta_l^{\text{Inc}} = 10$ and $\beta_l^{\text{Max}} = 1000/\lambda_l$ at all layers. Various parameters, timings and statistics are shown in Table 1. Due to the efficient inference scheme, we are able to train with many more feature maps and more data than other approaches, such as [2, 8, 11]. By the 4th layer, the receptive field of each feature map element (z pixel field) covers the entire image, making it suitable for the novel feature extraction process described in Section 3. At lower layers of the model, the representation has many latent variables (i.e. z's and s's) but as we ascend, the number drops. Counterbalancing this trend, the number of filter parameters grows dramatically as we ascend and the top layers of the model are able to learn object specific structures.

Property	Layer 1	Layer 2	Layer 3	Layer 4
# Feature maps	15	50	100	150
Pooling size	3x3x3	3x3x2	3x3x2	3x3x1
λ_l	1	0.01	0.001	0.0001
Inference time (s)	0.6	1.1	1.5	13.5
z pixel field	7x7	21x21	63x63	189x189
Feature map dims	156x156	58x58	26x26	15x15
# Filter Params	735	7,350	122,500	367,500
Total # z & s	378,560	178,200	71,650	37,500

Table 1. Parameter settings (top 3 rows) and statistics (lower 5 rows) of our model.

Timings: With 3060 training images and $E = 10$ epochs, it takes around 48 hours to train the entire 4 layer model. For inference, a single epoch suffices in the lower layers (to set the switches), but 5 are needed at the top layer. The total inference time per image is 16.7 secs (see Table 1 for per layer breakdown). Both these timings are for a Matlab implementation on a single six-core CPU. As previously discussed, the inference can easily be parallelized, thus a dramatic speed improvement could be expected from a GPU implementation.

4.1. Model visualization

The top-down nature of our model makes it easy to inspect what it has learned. In Fig. 2 we visualize the filters in the model by taking each feature map separately and picking the single largest absolute activation over the entire training set. Using the switch settings particular to that activation we projecting it down to the input pixel space. At layer 1, we see a range of oriented Gabors of differing frequencies, as well as some DC filters. In layer 2, a range of edge junctions and curves can be seen, built from combinations of the 1st layer filters. For a few selected filters (highlighted in color), we show the 25 largest activations across all images, not just the largest. Each group shows clustering with a certain degree of variation, as produced by the varying switch settings for that particular activation. E.g. the sliding configuration of the T-junction (green box). Reflecting their large receptive field, the filters in layer 3 show a range of complex compositions. The highlighted boxes show the model able to cluster quite complex structures. Note that the groupings produced are quite different to a pixel-space clustering of image patches since they are: (i) far from rectangular in shape; (ii) utilize the constrained

[3] Specific details: pixel spacing=2, patch size=16, codebook size=2000.

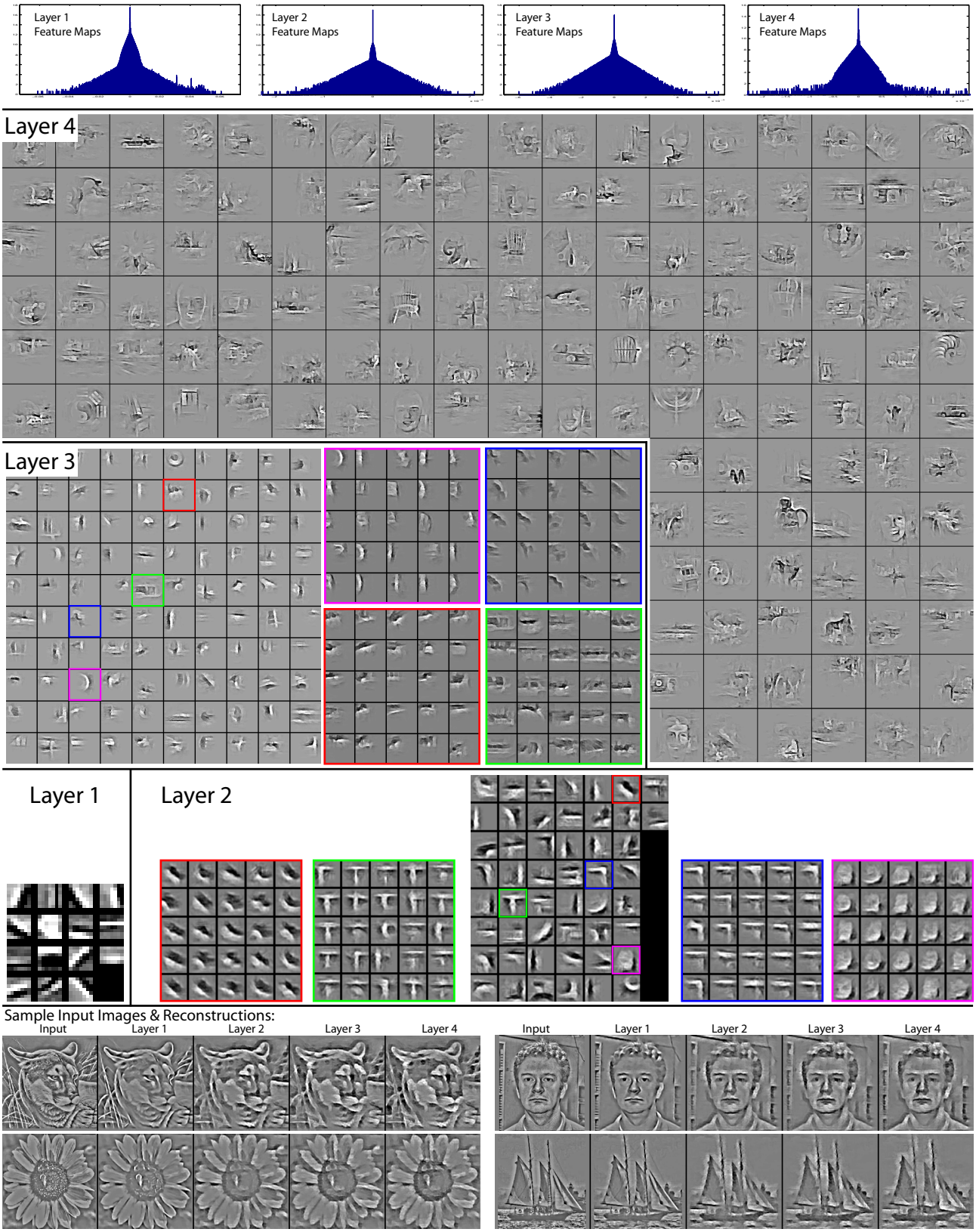

Figure 2. A visualization of the filters learned by our model, as well as image reconstructions and feature map histograms for each layer. See Section 4.1 for explanation. This figure is best viewed in electronic form.

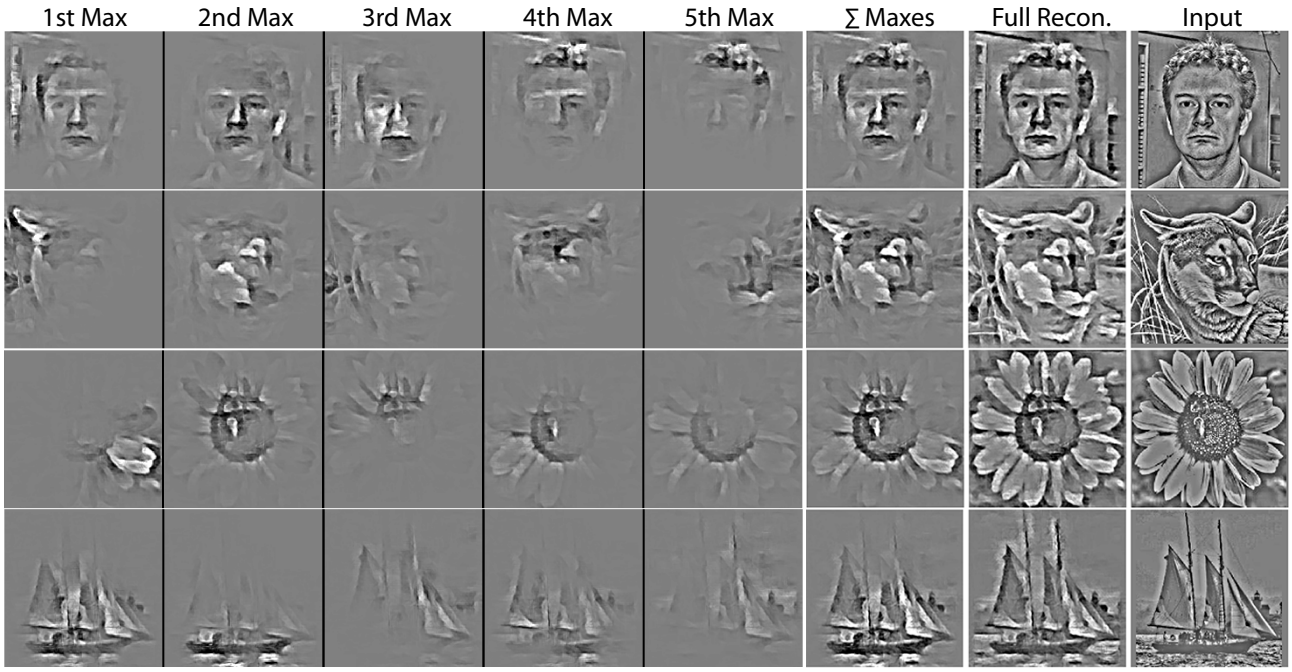

Figure 3. Columns 1–5: The largest 5 absolute activations in the 4th layer projected down to the input pixel space, for 4 different examples. Note how distinct structures within the image are reconstructed, despite the model being entirely unsupervised. See Section 3 for details on their use for recognition. Columns 6–8: sum of first 5 columns; reconstruction using all 4th layer activations; original input image.

geometric transformations offered by the switches below. The 4th layer filters show fairly complete reconstructions of entire objects. In many cases, the background has been suppressed and only consistent structures are captured. Fig. 2 also shows reconstructions from each layer of the model for 4 example input images. Note that unlike related models, such as Lee *et al.* [11], the sharp image edges are preserved in the reconstructions, even from the 4th layer. Finally, at the top of Fig. 2, log-histograms of the feature map activations are shown for each layer. As we ascend the model, the distribution becomes increasingly heavy-tailed, reflecting increased sparsity at higher layers. Further plots, showing the raw feature maps, filters and videos of inference taking place can be found in the supplementary material.

In Fig. 3 we show the pixel space reconstructions of the top $M = 5$ single feature map activations inferred from 4 different images (see Section 3 for more details). Note how a single element reconstructs complex groupings in the input image, thereby providing a soft decomposition. For example, the 2nd max for the human face reconstructs the left eye, nose, ear and mouth of the man, but little else. Conversely, the 5th max only focuses on reconstructing the hair. Similarly, different maxes within the schooner image (bottom row) pick out the hull and sails. The structures within each max reconstruction consist of textured regions (e.g. center of flower), as well as edge structures. They also tend to reconstruct the object better than the background. For comparison, Fig. 3 also shows the summation of the 5 max images ($\sum_{m=1}^{5} \hat{y}_4^{i,m}$); the full reconstruction using all elements of layer 4 ($\hat{y}_4^i$) and the original input image y^i.

4.2. Evaluation on Caltech 101

We use $M = 50$ decompositions from our model to produce input for training the Spatial Pyramid Match (SPM) classifier of Lazebnik *et al.* [9]. The classification results on the Caltech 101 test set are shown in Table 2.

Applying the SPM classifier to layer 1 features z_1 from

Our model - layer 1	66.5%
Our model - layer 4	70.3%
Our model - layer 1 + 4	71.1%
Chen *et al.* [2] layer-1+2	$65.7 \pm 0.7\%$
Kavukcuoglu *et al.* [8]	$65.7 \pm 0.7\%$
Zeiler *et al.* [18] layer-1+2	$66.9 \pm 1.1\%$
Boureau *et al.* [1] (Hard)	$70.9 \pm 1.0\%$
Jarrett *et al.* [7]	$65.6 \pm 1.0\%$
Lazebnik *et al.* [9]	$64.6 \pm 0.7\%$
Lee *et al.* [11] layer-1+2	$65.4 \pm 0.5\%$
Boureau *et al.* [1] (Soft)	$75.7 \pm 1.1\%$
Yang *et al.* [17] (Soft)	$73.2 \pm 0.5\%$

Table 2. Recognition performance on Caltech-101 of our model and a range of other approaches, grouped by similarity. Group 1: our approach; group 2: related convolutional sparse coding methods combined with SPM classifier; group 3: other methods using SPM classifier; group 4: methods using soft quantization in place of hard k-means in SPM classifier.

our model produces similar results (66.5%) to many other approaches, including those using convolutional sparse coding (2nd group of rows in Table 2). However, using the max activations from layer 4 in the SPM classifier, as detailed in Section 3, we obtain a significant performance improvement of around 4%, surpassing the majority of hierarchical and sparse coding approaches that also use the same SPM classifier (middle two groups in Table 2). Summing the SVM kernels resulting from the max activations from layer 4 and the layer 1 features, we achieve 71.1%. The only approach based on the SPM with comparable performance is that of Boureau *et al.* [1], based on Macrofeatures. Current state-of-the-art techniques [1, 17][4] use soft quantization of the descriptors, in place of the hard k-means quantization used in the SPM. We expect that using Macrofeatures and soft quantization would also boost our performance.

4.3. Comparison to layer-wise reconstruction

In Fig. 4 we show the results of training a model which only attempts to minimize the reconstruction of the layer below, i.e. $C_l = \|\hat{z}_{l-1} - z_{l-1}\|_2^2 + |z_l|_1$. Even in high layers, the filter projections are simple and are unable to give good reconstructions.

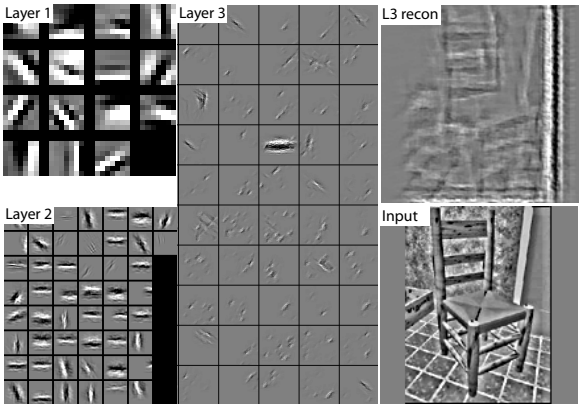

Figure 4. A visualization of the filters from a model trained to reconstruct just the output of the layer below. Without each layer being connected directly back to the input, complex filters cannot be obtained.

5. Discussion

The novel methods introduced in this paper allows us to learn rich image models with many layers. Existing sparse coding variants have not demonstrated the ability to learn more than 2 layers of representation. Our model is able to accurately reconstruct input images, even from layers 4. The decompositions produced by our model can be combined with standard classifiers to give excellent classification rates on Caltech 101. Matlab code for our algorithm is available at www://xyz.abc.

[4] In Table 2, we only consider approaches based on a single feature type. Approaches that combine hundreds of different features with multiple kernel learning methods outperform the methods listed in Table 2.

References

[1] Y. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *CVPR*. IEEE, 2010. 7, 8

[2] B. Chen, G. Sapiro, D. Dunson, and L. Carin. Deep learning with hierarichal convolutional factor analysis. *JMLR*, page Submitted, 2010. 2, 4, 5, 7

[3] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008. 1

[4] S. Fidler, M. Boben, and A. Leonardis. Similarity-based cross-layered hierarchical representation for object categorization. In *CVPR*, 2008. 2

[5] C. E. Guo, S. C. Zhu, and Y. N. Wu. Primal sketch: Integrating texture and structure. *CVIU*, 106:5–19, 2007. 2

[6] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, 2006. 1, 4

[7] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009. 4, 7

[8] K. Kavukcuoglu, P. Sermanet, Y. Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning convolutional feature hierachies for visual recognition. In *NIPS*, 2010. 2, 4, 5, 7

[9] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 5, 7

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *IEEE*, 86(11):2278–24, 1998. 1, 4

[11] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, pages 609–616, 2009. 1, 4, 5, 7

[12] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *ICML*, 2009. 1

[13] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, 1997. 1

[14] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999. 2

[15] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *CVPR*, 2005. 2

[16] Z. W. Tu and S. C. Zhu. Parsing images into regions, curves, and curve groups. *IJCV*, 69(2):223–249, August 2006. 2

[17] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009. 7, 8

[18] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus. Deconvolutional networks. In *CVPR*, 2010. 2, 3, 4, 7

[19] L. Zhu, Y. Chen, and A. L. Yuille. Learning a hierarchical deformable template for rapid deformable object parsing. *PAMI*, March 2009. 2

[20] S. Zhu and D. Mumford. A stochastic grammar of images. *Foundations and Trends in Comp. Graphics and Vision*, 2(4):259–362, 2006. 2