Convolutional Neural Fabrics

Shreyas Saxena, Jakob Verbeek

▶ **To cite this version:**

Shreyas Saxena, Jakob Verbeek. Convolutional Neural Fabrics. NIPS - Advances in Neural Information Processing Systems, Dec 2016, Barcelona, Spain. pp.1-9. hal-01359150v3

HAL Id: hal-01359150
https://inria.hal.science/hal-01359150v3

Submitted on 30 Jan 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Convolutional Neural Fabrics

Shreyas Saxena **Jakob Verbeek**
INRIA Grenoble – Laboratoire Jean Kuntzmann

Abstract

Despite the success of CNNs, selecting the optimal architecture for a given task remains an open problem. Instead of aiming to select a single optimal architecture, we propose a "fabric" that embeds an exponentially large number of architectures. The fabric consists of a 3D trellis that connects response maps at different layers, scales, and channels with a sparse homogeneous local connectivity pattern. The only hyper-parameters of a fabric are the number of channels and layers. While individual architectures can be recovered as paths, the fabric can in addition ensemble all embedded architectures together, sharing their weights where their paths overlap. Parameters can be learned using standard methods based on back-propagation, at a cost that scales linearly in the fabric size. We present benchmark results competitive with the state of the art for image classification on MNIST and CIFAR10, and for semantic segmentation on the Part Labels dataset.

1 Introduction

Convolutional neural networks (CNNs) [15] have proven extremely successful for a wide range of computer vision problems and other applications. In particular, the results of Krizhevsky *et al*. [13] have caused a major paradigm shift in computer vision from models relying in part on hand-crafted features, to end-to-end trainable systems from the pixels upwards. One of the main problems that holds back further progress using CNNs, as well as deconvolutional variants [24, 26] used for semantic segmentation, is the lack of efficient systematic ways to explore the discrete and exponentially large architecture space. To appreciate the number of possible architectures, consider a standard chain-structured CNN architecture for image classification. The architecture is determined by the following hyper-parameters: (i) number of layers, (ii) number of channels per layer, (iii) filter size per layer, (iv) stride per layer, (v) number of pooling *vs*. convolutional layers, (vi) type of pooling operator per layer, (vii) size of the pooling regions, (viii) ordering of pooling and convolutional layers, (ix) channel connectivity pattern between layers, and (x) type of activation, *e.g*. ReLU or MaxOut, per layer. The number of resulting architectures clearly does not allow for (near) exhaustive exploration.

We show that all network architectures that can be obtained for various choices of the above ten hyper-parameters are embedded in a "fabric" of convolution and pooling operators. Concretely, the fabric is a three-dimensional trellis of response maps of various resolutions, with only local connections across neighboring layers, scales, and channels. See Figure 1 for a schematic illustration of how fabrics embed different architectures. Each activation in a fabric is computed as a linear function followed by a non-linearity from a multi-dimensional neighborhood (spatial/temporal input dimensions, a scale dimension and a channel dimension) in the previous layer. Setting the only two hyper-parameters, number of layers and channels, is not ciritical as long as they are large enough. We also consider two variants, one in which the channels are fully connected instead of sparsely, and another in which the number of channels doubles if we move to a coarser scale. The latter allows for one to two orders of magnitude more channels, while increasing memory requirements by only 50%.

All chain-structured network architectures embedded in the fabric can be recovered by appropriately setting certain connections to zero, so that only a single processing path is active between input and output. General, non-path, weight settings correspond to ensembling many architectures together,

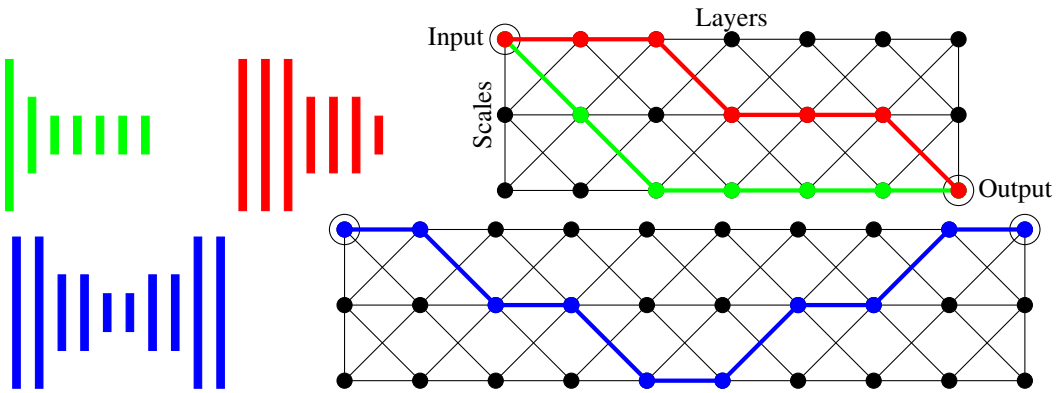

Figure 1: Fabrics embedding two seven-layer CNNs (red, green) and a ten-layer deconvolutional network (blue). Feature map size of the CNN layers are given by height. Fabric nodes receiving input and producing output are encircled. All edges are oriented to the right, down in the first layer, and towards the output in the last layer. The channel dimension of the 3D fabric is omitted for clarity.

which share parameters where the paths overlap. The acyclic trellis structure allows for learning using standard error back-propagation methods. Learning can thus efficiently configure the fabric to implement each one of exponentially many embedded architectures, as well as ensembles of them. Experimental results competitive with the state of the art validate the effectiveness of our approach.

The contributions of our work are: (1) Fabrics allow by and large to sidestep the CNN model architecture selection problem. Avoiding explicitly training and evaluating individual architectures using, *e.g.*, local-search strategies [2]. (2) While scaling linearly in terms of computation and memory requirements, our approach leverages exponentially many chain-structured architectures in parallel by massively sharing weights among them. (3) Since our fabric is multi-scale by construction, it can naturally generate output at multiple resolutions, *e.g.* for image classification and semantic segmentation or multi-scale object detection, within a single non-branching network structure.

2 Related work

Several chain-structured CNN architectures, including Alex-net [13] and the VGG-16 and VGG-19 networks [27], are widely used for image classification and related tasks. Although very effective, it is not clear that these architectures are the best ones given their computational and memory requirements. Their widespread adoption is in large part due to the lack of more effective methods to find good architectures than trying them one-by-one, possibly initializing parameters from related ones [2].

CNN architectures for semantic segmentation, as well as other structured prediction tasks such as human pose estimation [25], are often derived from ones developed for image classification, see *e.g.* [20, 24, 31, 33]. Up-sampling operators are used to increase the resolution of the output, compensating for pooling operators used in earlier layers of the network [24]. Ronneberger *et al.* [26] present a network with additional links that couple layers with the same resolution near the input and output. Other architectures, see *e.g.* [3, 7], process the input in parallel across several resolutions, and then fuse all streams by re-sampling to the output resolution. Such architectures induce networks with multiple parallel paths from input to output. We will show that nearly all such networks are embedded in our fabrics, either as paths or other simple sub-graphs.

While multi-dimensional networks have been proposed in the past, *e.g.* to process non-sequential data with recurrent nets [5, 11], to the best of our knowledge they have not been explored as a "basis" to span large classes of convolutional neural networks. Misra *et al.* [23] propose related cross-stitch networks that exchange information across corresponding layers of two copies of the same architecture that produces two different outputs. Their approach is based on Alex-net [13], and does not address the network architecture selection problem. In related work Zhou *et al.* [34] interlink CNNs that take input from re-scaled versions of the input image. The structure of their network is related to our fabric, but lacks a sparse connectivity pattern across channels. They consider their networks for semantic segmentation, and set the filter sizes per node manually, and

use strided max-pooling for down-sampling and nearest neighbor interpolation for up-sampling. The contribution of our work is to show that a similar network structure suffice to span a vast class of network architectures for both dense prediction and classification tasks.

Springenberg *et al*. [29] experimentally observed that the use of max-pooling in CNN architectures is not always beneficial as opposed to using strided convolutions. In our work we go one step further and show that ReLU units and strided convolutions suffice to implement max-pooling operators in our fabrics. Their work, similar to ours, also strives to simplify architecture design. Our results, however, reach much further than only removing pooling operators from the architectural elements. Lee *et al*. [17] generalize the max and average pooling operators by computing both max and average pooling, and then fusing the result in a possibly data-driven manner. Our fabrics also generalize max and average pooling, but instead of adding elementary operators, we show that settings weights in a network with fewer elementary operators is enough for this generalization.

Kulkarni *et al*. [14] use ℓ_1 regularization to automatically select the number of units in "fully-connected" layers of CNN architectures for classification. Although their approach does not directly extend to determine more general architectural design choices, it might be possible to use such regularization techniques to select the number of channels and/or layers of our fabrics.

Dropout [30] and swapout [28] are stochastic training methods related to our work. They can be understood as approximately averaging over an exponential number of variations of a given architecture. Our approach, on the other hand, allows to leverage an exponentially large class of architectures (ordering of pooling and convolutional layers, type of pooling operator, *etc*.) by means of continuous optimization. Note that these approaches are orthogonal and can be applied to fabrics.

3 The fabric of convolutional neural networks

In this section we give a precise definition of convolutional neural fabrics, and show in Section 3.2 that most architectural network design choices become irrelevant for sufficiently large fabrics. Finally, we analyze the number of response maps, parameters, and activations of fabrics in Section 3.3.

3.1 Weaving the convolutional neural fabric

Each node in the fabric represents one response map with the same dimension D as the input signal ($D = 1$ for audio, $D = 2$ for images, $D = 3$ for video). The fabric over the nodes is spanned by three axes. A **layer axis** along which all edges advance, which rules out any cycles, and which is analogous to the depth axis of a CNN. A **scale axis** along which response maps of different resolutions are organized from fine to coarse, neighboring resolutions are separated by a factor two. A **channel axis** along which different response maps of the same scale and layer are organized. We use $S = 1 + \log_2 N$ scales when we process inputs of size N^D, *e.g*. for 32×32 images we use six scales, so as to obtain a scale pyramid from the full input resolution to the coarsest 1×1 response maps.

We now define a sparse and homogeneous edge structure. Each node is connected to a 3×3 scale–channel neighborhood in the previous layer, *i.e*. activations at channel c, scale s, and layer l are computed as $a(s, c, l) = \sum_{i,j\in\{-1,0,1\}} \text{conv}\big(a(c + i, s + j, l - 1), w^{ij}_{scl}\big)$. Input from a finer scale is obtained via strided convolution, and input from a coarser scale by convolution after upsampling by padding zeros around the activations at the coarser level. All convolutions use kernel size 3. Activations are thus a linear function over multi-dimensional neighborhoods, *i.e*. a four dimensional 3×3×3×3 neighborhood when processing 2D images. The propagation is, however, only convolutional across the input dimensions, and not across the scale and layer axes. The "fully connected" layers of a CNN correspond to nodes along the coarsest 1×1 scale of the fabric. Rectified linear units (ReLUs) are used at all nodes. Figure 1 illustrates the connectivity pattern in 2D, omitting the channel dimension for clarity. The supplementary material contains an illustration of the 3D fabric structure.

All channels in the first layer at the input resolution are connected to all channels of the input signal. The first layer contains additional edges to distribute the signal across coarser scales, see the vertical edges in Figure 1. More precisely, within the first layer, channel c at scale s receives input from channels $c + \{-1, 0, 1\}$ from scale $s - 1$. Similarly, edges within the last layer collect the signal towards the output. Note that these additional edges do not create any cycles, and that the edge-structure within the first and last layer is reminiscent of the 2D trellis in Figure 1.

3.2 Stitching convolutional neural networks on the fabric

We now demonstrate how various architectural choices can be "implemented" in fabrics, demonstrating they subsume an exponentially large class of network architectures. Learning will configure a fabric to behave as one architecture or another, but more generally as an ensemble of many of them. For all but the last of the following paragraphs, it is sufficient to consider a 2D trellis, as in Figure 1, where each node contains the response maps of C channels with dense connectivity among channels.

Re-sampling operators. A variety of re-sampling operators is available in fabrics, here we discuss ones with small receptive fields, larger ones are obtained by repetition. Stride-two convolutions are used in fabrics on fine-to-coarse edges, larger strides are obtained by repetition. *Average pooling* is obtained in fabrics by striding a uniform filter. Coarse-to-fine edges in fabrics up-sample by padding zeros around the coarse activations and then applying convolution. For factor-2 *bilinear interpolation* we use a filter that has 1 in the center, $1/4$ on corners, and $1/2$ elsewhere. *Nearest neighbor interpolation* is obtained using a filter that is 1 in the four top-left entries and zero elsewhere.

For *max-pooling* over a 2×2 region, let a and b represent the values of two vertically neighboring pixels. Use one layer and three channels to compute $(a+b)/2$, $(a-b)/2$, and $(b-a)/2$. After ReLU, a second layer can compute the sum of the three terms, which equals $\max(a, b)$. Each pixel now contains the maximum of its value and that of its vertical neighbor. Repeating the same in the horizontal direction, and sub-sampling by a factor two, gives the output of 2×2 max-pooling. The same process can also be used to show that a network of *MaxOut units* [4] can be implemented in a network of ReLU units. Although ReLU and MaxOut are thus equivalent in terms of the functions they can implement, for training efficiency it may be more advantageous to use MaxOut networks.

Filter sizes. To implement a 5×5 filter we first compute nine intermediate channels to obtain a vectorized version of the 3×3 neighborhood at each pixel, using filters that contain a single 1, and are zero elsewhere. A second 3×3 convolution can then aggregate values across the original 5×5 patch, and output the desired convolution. Any 5×5 filter can be implemented exactly in this way, not only approximated by factorization, *c.f.* [27]. Repetition allows to obtain filters of any desired size.

Ordering convolution and re-sampling. As shown in Figure 1, chain-structured networks correspond to paths in our fabrics. If weights on edges outside a path are set to zero, a chain-structured network with a particular sequencing of convolutions and re-sampling operators is obtained. A trellis that spans $S+1$ scales and $L+1$ layers contains more than $\binom{L}{S}$ chain-structured CNNs, since this corresponds to the number of ways to spread S sub-sampling operators across the L steps to go from the first to the last layer. More CNNs are embedded, *e.g.* by exploiting edges within the first and last layer, or by including intermediate up-sampling operators. Networks beyond chain-structured ones, see *e.g.* [3, 20, 26], are also embedded in the trellis, by activating a larger subset of edges than a single path, *e.g.* a tree structure for the multi-scale net of [3].

Channel connectivity pattern. Although most networks in the literature use dense connectivity across channels between successive layers, this is not a necessity. Krizhevsky *et al.* [13], for example, use a network that is partially split across two independent processing streams.

In Figure 2 we demonstrate that a fabric which is sparsely connected along the channel axis, suffices to emulate densely connected convolutional layers. This is achieved by copying channels, convolving them, and then locally aggregating them. Both the copy and sum process are based on local channel interactions and convolutions with filters that are either entirely zero, or identity filters which are all zero except for a single 1 in the center. While more efficient constructions exist to represent the densely connected layer in our trellis, the one presented here is simple to understand and suffices to demonstrate feasibility. Note that in practice learning automatically configures the trellis.

Both the copy and sum process generally require more than one layer to execute. In the copying process, intermediate ReLUs do not affect the result since the copied values themselves are non-negative outputs of ReLUs. In the convolve-and-sum process care has to be taken since one convolution might give negative outputs, even if the sum of convolutions is positive. To handle this correctly, it suffices to shift the activations by subtracting from the bias of every convolution i the minimum possible corresponding output $a_i^{\min}$ (which always exists for a bounded input domain). Using the adjusted bias, the output of the convolution is now guaranteed to be non-negative, and to propagate properly in the copy and sum process. In the last step of summing the convolved channels, we can add back $\sum_i a_i^{\min}$ to shift the activations back to recover the desired sum of convolved channels.

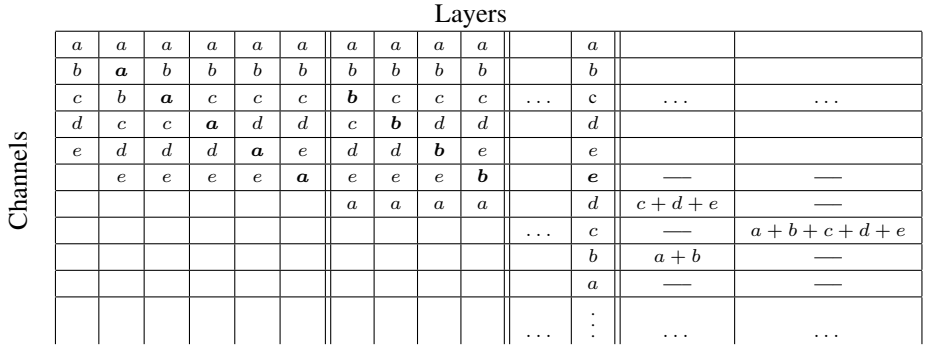

Figure 2: Representation of a dense-channel-connect layer in a fabric with sparse channel connections using copy and swap operations. The five input channels $a, \ldots, e$ are first copied; more copies are generated by repetition. Channels are then convolved and locally aggregated in the last two layers to compute the desired output. Channels in rows, layers in columns, scales are ignored for simplicity.

Table 1: Analysis of fabrics with L layers, S scales, C channels. Number of activations given for $D = 2$ dim. inputs of size $N \times N$ pixels. Channel doubling across scales used in the bottom row.

# chan. / scale	# resp. maps	# parameters (sparse)	# parameters (dense)	# activations
constant	$C \cdot L \cdot S$	$C \cdot L \cdot 3^{D+1} \cdot 3 \cdot S$	$C \cdot L \cdot 3^{D+1} \cdot C \cdot S$	$C \cdot L \cdot N^2 \cdot \frac{4}{3}$
doubling	$C \cdot L \cdot 2^S$	$C \cdot L \cdot 3^{D+1} \cdot 3 \cdot 2^S$	$C \cdot L \cdot 3^{D+1} \cdot C \cdot 4^S \cdot \frac{7}{18}$	$C \cdot L \cdot N^2 \cdot 2$

3.3 Analysis of the number of parameters and activations

For our analysis we ignore border effects, and consider every node to be an internal one. In the top row of Table 1 we state the total number of response maps throughout the fabric, and the number of parameters when channels are sparsely or densely connected. We also state the number of activations, which determines the memory usage of back-propagation during learning.

While embedding an exponential number of architectures in the number of layers L and channels C, the number of activations and thus the memory cost during learning grows only linearly in C and L. Since each scale reduces the number of elements by a factor 2^D, the total number of elements across scales is bounded by $2^D/(2^D - 1)$ times the number of elements N^D at the input resolution.

The number of parameters is linear in the number of layers L, and number of scales S. For sparsely connected channels, the number of parameters grows also linearly with the number of channels C , while it grows quadratically with C in case of dense connectivity.

As an example, the largest models we trained for 32×32 input have $L = 16$ layers and $C = 256$ channels, resulting in 2M parameters (170M for dense), and 6M activations. For 256×256 input we used upto $L = 16$ layers and $C = 64$ channels, resulting in 0.7 M parameters (16M for dense), and 89M activations. For reference, the VGG-19 model has 144M parameters and 14M activations.

Channel-doubling fabrics. Doubling the number of channels when moving to coarser scales is used in many well-known architectures, see *e.g.* [26, 27]. In the second row of Table 1 we analyze fabrics with channel-doubling instead of a constant number of channels per scale. This results in $C2^S$ channels throughout the scale pyramid in each layer, instead of CS when using a constant number of channels per scale, where we use C to denote the number of "base channels" at the finest resolution. For 32×32 input images the total number of channels is roughly 11× larger, while for 256×256 images we get roughly 57× more channels. The last column of Table 1 shows that the number of activations, however, grows only by 50% due to the coarsening of the maps.

With dense channel connections and 2D data, the amount of computation per node is constant, as at a coarser resolution there are 4× less activations, but interactions among 2×2 more channels. Therefore, in such fabrics the amount of computation grows linearly in the number of scales as compared to a single embedded CNN. For sparse channel connections, we adapt the local connectivity pattern between nodes to accommodate for the varying number channels per scale, see Figure 3 for an illustration. Each node still connects to nine other nodes at the previous layer: two inputs from scale $s - 1$, three from scale s, and four from scale $s + 1$. The computational cost thus also grows only

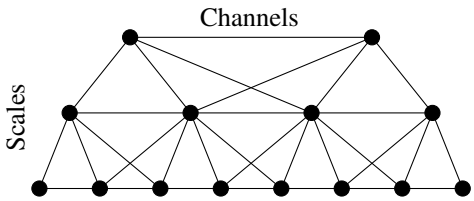

Figure 3: Diagram of sparse channel connectivity from one layer to another in a channel-doubling fabric. Channels are laid out horizontally and scales vertically. Each internal node, *i.e*. response map, is connected to nine nodes at the previous layer: four channels at a coarser resolution, two at a finer resolution, and to itself and neighboring channels at the same resolution.

by 50% as compared to using a constant number of channels per scale. In this case, the number of parameters grows by the same factor $2^S/S$ as the number of channels. In case of dense connections, however, the number of parameters explodes with a factor $\frac{7}{18}4^S/S$. That is, roughly a factor 265 for 32×32 input, and 11,327 for 256×256 input. Therefore, channel-doubling fabrics appear most useful with sparse channel connectivity. Experiments with channel-doubling fabrics are left for future work.

4 Experimental evaluation results

In this section we first present the datasets used in our experiments, followed by evaluation results.

4.1 Datasets and experimental protocol

Part Labels dataset. This dataset [10] consists of 2,927 face images from the LFW dataset [8], with pixel-level annotations into the classes *hair*, *skin*, and *background*. We use the standard evaluation protocol which specifies training, validation and test sets of 1,500, 500 and 927 images, respectively. We report accuracy at pixel-level and superpixel-level. For superpixel we average the class probabilities over the contained pixels. We used horizontal flipping for data augmentation.

MNIST. This dataset [16] consists of 28×28 pixel images of the handwritten digits $0, \ldots, 9$. We use the standard split of the dataset into 50k training samples, 10k validation samples and 10k test samples. Pixel values are normalized to $[0, 1]$ by dividing them by 255. We augment the train data by randomly positioning the original image on a 32×32 pixel canvas.

CIFAR10. The CIFAR-10 dataset (http://www.cs.toronto.edu/~kriz/cifar.html) consists of 50k 32×32 training images and 10k testing images in 10 classes. We hold out 5k training images as validation set, and use the remaining 45k as the training set. To augment the data, we follow common practice, see *e.g*. [4, 18], and pad the images with zeros to a 40×40 image and then take a random 32×32 crop, in addition we add horizontally flipped versions of these images.

Training. We train our fabrics using SGD with momentum of 0.9. After each node in the trellis we apply batch normalization [9], and regularize the model with weight decay of 10^{-4}, but did not apply dropout [30]. We use the validation set to determine the optimal number of training epochs, and then train a final model from the train and validation data and report performance on the test set. We release our Caffe-based implementation at http://thoth.inrialpes.fr/~verbeek/fabrics.

4.2 Experimental results

For all three datasets we trained sparse and dense fabrics with various numbers of channels and layers. In all cases we used a constant number of channels per scale. The results across all these settings can be found in Appendix A.3, here we report only the best results from these. On all three datasets, larger trellises perform comparable or better than smaller ones. So in practice the choice of these only two hyper-parameters of our model is not critical, as long as a large enough trellis is used.

Part Labels. On this data set we obtained a super-pixel accuracy of 95.6% using both sparse and dense trellises. In Figure 4 we show two examples of predicted segmentation maps. Table 2 compares our results with the state of the art, both in terms of accuracy and the number of parameters. Our results are slightly worse than [31, 33], but the latter are based on the VGG-16 network. That network has roughly $4,000\times$ more parameters than our sparse trellis, and has been trained from over 1M ImageNet images. We trained our model from scratch using only 2,000 images. Moreover, [10, 19, 31] also include CRF and/or RBM models to encode spatial shape priors. In contrast, our results with convolutional neural fabrics (CNF) are obtained by predicting all pixels independently.

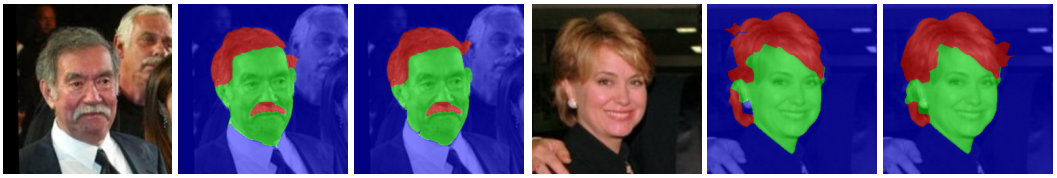

Figure 4: Examples form the Part Labels test set: input image (left), ground-truth labels (middle), and superpixel-level labels from our sparse CNF model with 8 layers and 16 channels (right).

Table 2: Comparison of our results with the state of the art on Part Labels.

	Year	# Params.	SP Acccur.	P Accur.
Tsogkas *et al*. [31]	2015	>414 M	96.97	—
Zheng *et al*. [33]	2015	>138 M	96.59	—
Liu *et al*. [19]	2015	>33 M	—	95.24
Kae *et al*. [10]	2013	0.7 M	94.95	—
Ours: CNF-sparse ($L = 8, C = 16$)		0.1 M	95.58	94.60
Ours: CNF-dense ($L = 8, C = 64$)		8.0 M	95.63	94.82

MNIST. We obtain error rates of 0.48% and 0.33% with sparse and dense fabrics respectively. In Table 3 we compare our results to a selection of recent state-of-the-art work. We excluded several more accurate results reported in the literature, since they are based on significantly more elaborate data augmentation methods. Our result with a densely connected fabric is comparable to those of [32], which use similar data augmentation. Our sparse model, which has 20× less parameters than the dense variant, yields an error of 0.48% which is slightly higher.

CIFAR10. In Table 4 we compare our results to the state of the art. Our error rate of 7.43% with a dense fabric is comparable to that reported with MaxOut networks [4]. On this dataset the error of the sparse model, 18.89%, is significantly worse than the dense model. This is either due to a lack of capacity in the sparse model, or due to difficulties in optimization. The best error of 5.84% [22] was obtained using residual connections, without residual connections they report an error of 6.06%.

Visualization. In Figure 5 we visualize the connection strengths of learned fabrics with dense channel connectivity. We observe qualitative differences between learned fabrics. The semantic segmentation model (left) immediately distributes the signal across the scale pyramid (first layer/column), and then progressively aggregates the multi-scale signal towards the output. In the CIFAR10 classification model the signal is progressively downsampled, exploiting multiple scales in each layer. The figure shows the result of heuristically pruning (by thresholding) the weakest connections to find a smaller sub-network with good performance. We pruned 67% of the connections while increasing the error only from 7.4% to 8.1% after fine-tuning the fabric with the remaining connections. Notice that all up-sampling connections are deactivated after pruning.

Table 3: Comparison of our results with the state of the art on MNIST. Data augmentation with translation and flipping is denoted by T and F respectively, N denotes no data augmentation.

	Year	Augmentation	# Params.	Error (%)
Chang *et al*. [1]	2015	N	447K	0.24
Lee *et al*. [17]	2015	N		0.31
Wan *et al*. (Dropconnect) [32]	2013	T	379K	0.32
CKN [21]	2014	N	43 K	0.39
Goodfellow *et al*. (MaxOut) [4]	2013	N	420 K	0.45
Lin *et al*. (Network in Network) [18]	2013	N		0.47
Ours: CNF-sparse ($L = 16, C = 32$)		T	249 K	0.48
Ours: CNF-dense ($L = 8, C = 64$)		T	5.3 M	0.33

Table 4: Comparison of our results with the state of the art on CIFAR10. Data augmentation with translation, flipping, scaling and rotation are denoted by T, F, S and R respectively.

	Year	Augmentation	# Params.	Error (%)
Mishkin & Matas [22]	2016	T+F	2.5M	5.84
Lee *et al*. [17]	2015	T+F	1.8M	6.05
Chang *et al*. [1]	2015	T+F	1.6M	6.75
Springenberg *et al*. (All Convolutional Net) [29]	2015	T+F	1.3 M	7.25
Lin *et al*. (Network in Network) [18]	2013	T+F	1 M	8.81
Wan *et al*. (Dropconnect) [32]	2013	T+F+S+R	19M	9.32
Goodfellow *et al*. (MaxOut) [4]	2013	T+F	>6 M	9.38
Ours: CNF-sparse ($L = 16, C = 64$)		T+F	2M	18.89
Ours: CNF-dense ($L = 8, C = 128$)		T+F	21.2M	7.43

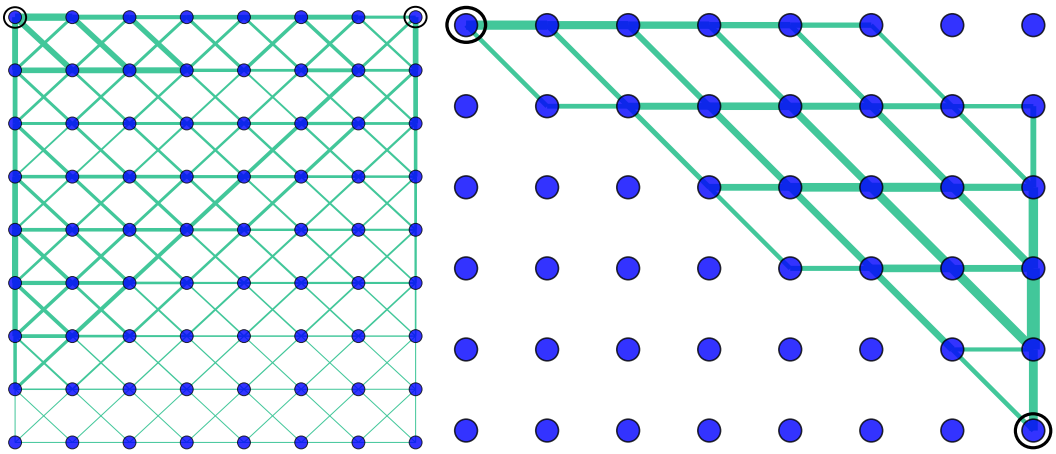

Figure 5: Visualization of mean-squared filter weights in fabrics learned for Part Labels (left) and CIFAR10 (right, pruned network connections). Layers are laid out horizontally, and scales vertically.

5 Conclusion

We presented convolutional neural fabrics: homogeneous and locally connected trellises over response maps. Fabrics subsume a large class of convolutional networks. They allow to sidestep the tedious process of specifying, training, and testing individual network architectures in order to find the best ones. While fabrics use more parameters, memory and computation than needed for each of the individual architectures embedded in them, this is far less costly than the resources required to test all embedded architectures one-by-one. Fabrics have only two main hyper-parameters: the number of layers and the number of channels. In practice their setting is not critical: we just need a large enough fabric with enough capacity. We propose variants with dense channel connectivity, and with channel-doubling over scales. The latter strikes a very attractive capacity/memory trade-off.

In our experiments we study performance of fabrics for image classification on MNIST and CIFAR10, and of semantic segmentation on Part Labels. We obtain excellent results that are close to the best reported results in the literature on all three datasets. These results suggest that fabrics are competitive with the best hand-crafted CNN architectures, be it using a larger number of parameters in some cases (but much fewer on Part Labels). We expect that results can be further improved by using better optimization schemes such as Adam [12], using dropout [30] or dropconect [32] regularization, and using MaxOut units [4] or residual units [6] to facilitate training of deep fabrics with many channels.

In ongoing work we experiment with channel-doubling fabrics, and fabrics for joint image classification, object detection, and segmentation. We also explore channel connectivity patterns in between the sparse and dense options used here. Finally, we work on variants that are convolutional along the scale-axis so as to obtain a scale invariant processing that generalizes better across scales.

Acknowledgment. We would like to thank NVIDIA for the donation of GPUs used in this research. This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

References

[1] J.-R. Chang and Y.-S. Chen. Batch-normalized maxout network in network. Arxiv preprint, 2015.
[2] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. In *ICLR*, 2016.
[3] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *PAMI*, 35(8):1915–1929, 2013.
[4] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013.
[5] A. Graves, S. Fernández, and J. Schmidhuber. Multi-dimensional recurrent neural networks. In *ICANN*, 2007.
[6] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.
[7] S. Honari, J. Yosinski, P. Vincent, and C. Pal. Recombinator networks: Learning coarse-to-fine feature aggregation. In *CVPR*, 2016.
[8] G. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: a database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, 2007.
[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.
[10] A. Kae, K. Sohn, H. Lee, and E. Learned-Miller. Augmenting CRFs with Boltzmann machine shape priors for image labeling. In *CVPR*, 2013.
[11] N. Kalchbrenner, I. Danihelka, and A. Graves. Grid long short-term memory. In *ICLR*, 2016.
[12] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
[13] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.
[14] P. Kulkarni, J. Zepeda, F. Jurie, P. Pérez, and L. Chevallier. Learning the structure of deep architectures using ℓ_1 regularization. In *BMVC*, 2015.
[15] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1989.
[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, pages 2278–2324, 1998.
[17] C.-Y. Lee, P. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *AISTATS*, 2016.
[18] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.
[19] S. Liu, J. Yang, C. Huang, , and M.-H. Yang. Multi-objective convolutional learning for face labeling. In *CVPR*, 2015.
[20] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.
[21] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *NIPS*, 2014.
[22] D. Mishkin and J. Matas. All you need is a good init. In *ICLR*, 2016.
[23] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stich networks for multi-task learning. In *CVPR*, 2016.
[24] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015.
[25] T. Pfister, J. Charles, and A. Zisserman. Flowing ConvNets for human pose estimation in videos. In *CVPR*, 2015.
[26] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, 2015.
[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.
[28] S. Singh, D. Hoiem, and D. Forsyth. Swapout: learning an ensemble of deep architectures. In *NIPS*, 2016.
[29] J. Springenberg, A. Dosovitskiy andT. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR*, 2015.
[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.
[31] S. Tsogkas, I. Kokkinos, G. Papandreou, and A. Vedaldi. Deep learning for semantic part segmentation with high-level guidance. Arxiv preprint, 2015.
[32] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using DropConnect. In *ICML*, 2013.
[33] H. Zheng, Y. Liu, M. Ji, F. Wu, and L. Fang. Learning high-level prior with convolutional neural networks for semantic segmentation. Arxiv preprint, 2015.
[34] Y. Zhou, X. Hu, and B. Zhang. Interlinked convolutional neural networks for face parsing. In *International Symposium on Neural Networks*, 2015.

A Supplementary Material

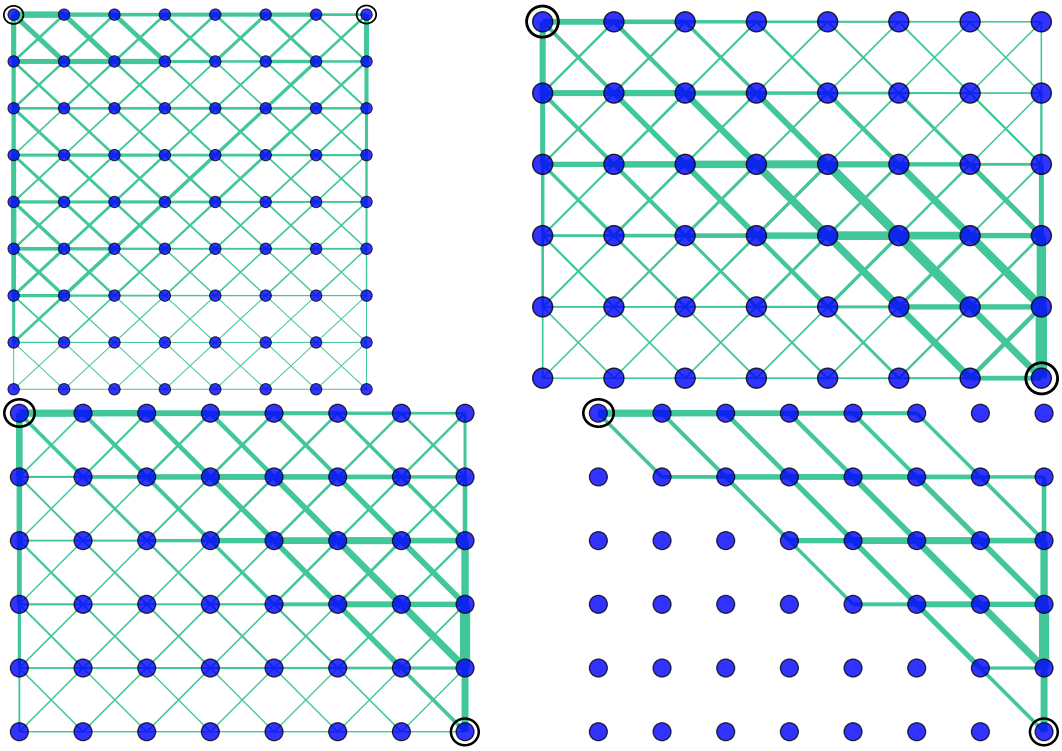

Figure 6: Visualization of connection strengths in fabrics learned for Part Labels (top left), MNIST (top right), and CIFAR10 (bottom left and right). Layers are laid out horizontally, and scales vertically.

A.1 Fabric visualizations

In Figure 6 we show fabrics learned on the three datasets used in the main paper. The connection strengths are visualized by setting edge widths proportional to the mean squared filter weights related to that connection (mean across the 3×3 weights and across all channels). The fabrics used for these visualization use dense channel connectivity.

Clearly, the learned connection patterns differ between the segmentation and the classification datasets, since output needs to be produced at different resolutions for these tasks. Interestingly, the pattern is also different between the two classification datasets. For the task of image classification, we have two models, CIFAR10 (bottom-left) and MNIST (top-right). In the case of MNIST, signal in the fabric is propagated in a band-diagonal pattern, exploiting multiple scales in each layer. With CIFAR10 we observe a similar pattern, but with a small difference. The signal is first processed at the finest scale for a few layers, and then propagated down. These results demonstrate that even for the same task, the fabric is able to configure itself in order to accommodate nuances of the dataset.

For the CIFAR-10 dataset we also show the pruned network, obtained by (arbitrarily) pruning all connections below the mean connection strength. After fine-tuning the pruned network, the classification error is 8.1% as compared to 7.4% for the full fabric, while removing 67% of the connections in the fabric.

A.2 Fabric structure

In Figure 7 we illustrate how the multiple inputs to a node in a fabric are combined. Input from a finer scale is processed by strided convolution, from the same scale by normal convolution, and from a coarser scale by first upsampling the signal by zero padding and then convolving the signal. The three input signals are then added, and then a ReLU activation function is applied to the result.

In Figure 8 we illustrate the 3D fabric structure when using sparse channel connectivity. In this case, each internal node corresponds to a single response map at a given resolution. Nodes receive input from 3×3 nodes from the scale-channel plane of the previous layer. Note that each edge represents a 3×3 spatial convolution. Therefore, the input that projects into any "neuron" in the fabric is a 4D tensor of size $3 \times 3 \times 3 \times 3$ across two spatial dimensions, a scale and a channel dimension.

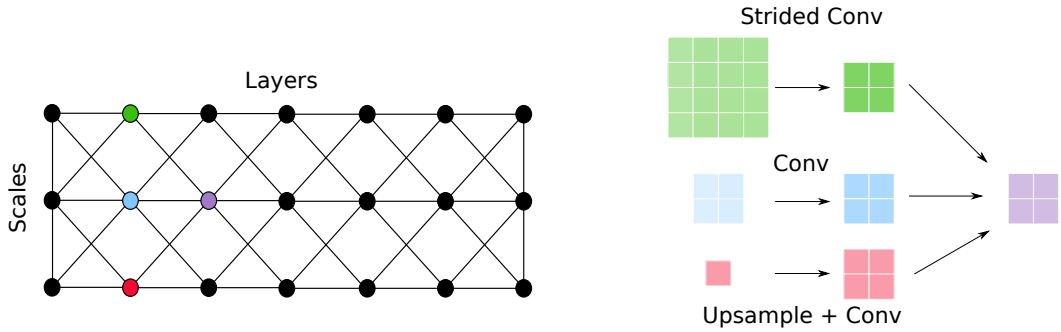

Figure 7: Visualization of the multi-scale signal propagation in fabrics. Layers are laid out horizontally, and scales vertically. Strided convolution and convolution after zero-padding up-sampling are used to adapt finer and coarser input signals to the node resolution.

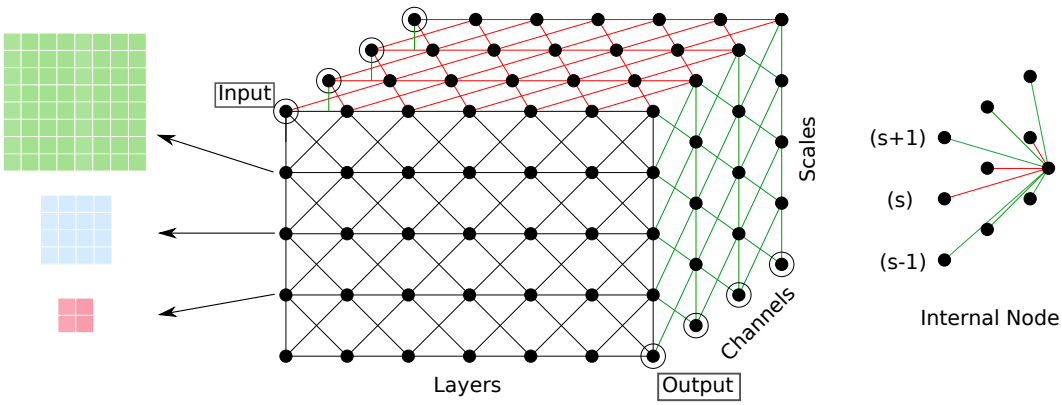

Figure 8: Visualization of the 3D fabric with sparse channel connectivity. Layers are laid out horizontally, and scales vertically, channels along the third dimension.

A.3 Additional experimental results

In Figure 9 we provide additional segmentation results on the Part Labels data set, both at superpixel and pixel level. In tables 5–12 we give detailed experimental results obtained using sparse and dense fabrics of different sizes, and give the corresponding number of parameters for each model.

The results reported here are measured on the test set, using models trained on the train set only. In the main paper results are reported on the test set using models trained from both the train and validation data. In both cases the number of training epochs has been selected to maximize performance on the validation data, when training from the train data only.

On all three datasets larger fabrics generally give better or comparable results as compared to smaller ones, despite relatively simple regularization by means of weight decay and early stopping. These results show that the number of channels and layers are not critical parameters of our fabrics, it simply suffices to take them large enough.

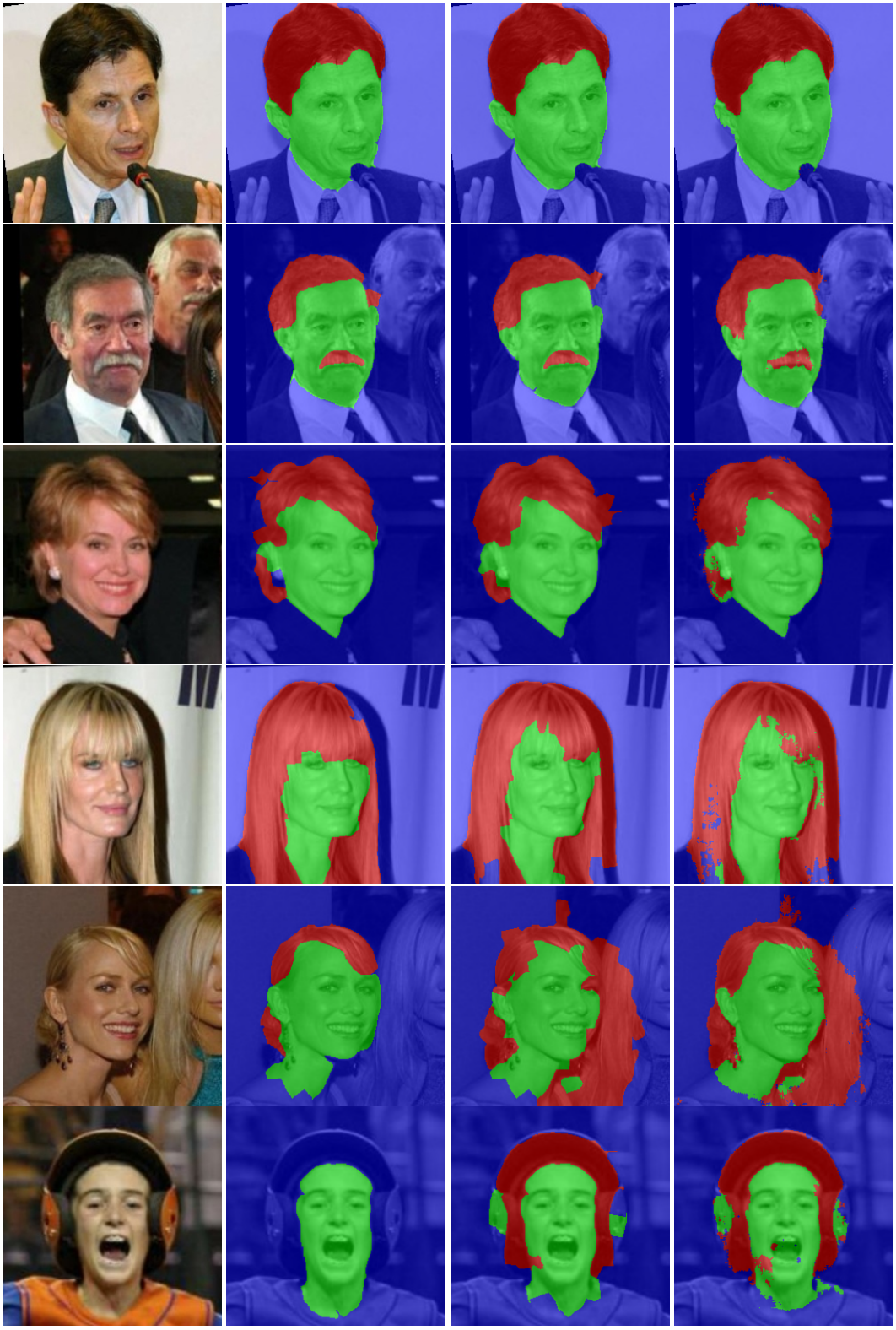

Figure 9: Example segmentations on the Part Labels test set, from left to right: input image, ground-truth labels, superpixel-level predictions, and pixel-level prediction from our sparse CNF model with 8 layers and 16 channels. Successful segmentation in the top three rows, failure cases in the bottom three rows. All predictions are made independently across pixels, *i.e*. no CRF models have been used.

Table 5: Accuracy on Part Labels for CNF-sparse. Number of parameters given in parentheses.

Layers / Channels	4		16		64	
2	91.95	*(6K)*	94.76	*(23K)*	95.14	*(93K)*
4	93.94	*(12K)*	95.02	*(47K)*	95.34	*(187K)*
8	94.87	*(23K)*	**95.48**	*(93K)*	95.46	*(373K)*
16	95.15	*(47K)*	95.38	*(187K)*	95.26	*(746K)*

Table 6: Accuracy on Part Labels for CNF-dense. Number of parameters given in parentheses.

Layers / Channels	4		16		64	
2	93.57	*(8K)*	95.26	*(124K)*	95.33	*(2M)*
4	93.67	*(16K)*	95.05	*(249K)*	95.20	*(4M)*
8	95.09	*(31K)*	95.22	*(498K)*	**95.39**	*(8M)*
16	94.92	*(62K)*	95.29	*(995K)*	95.34	*(16M)*

Table 7: Error rate on MNIST for CNF-sparse. Number of parameters given in parentheses.

Layers / Channels	4		8		16		32		64		128	
2	4.94	*(4K)*	2.32	*(8K)*	1.68	*(16K)*	1.40	*(31K)*	0.97	*(62K)*	1.00	*(124K)*
4	2.96	*(8K)*	1.69	*(16K)*	1.14	*(31K)*	0.96	*(62K)*	0.98	*(124K)*	0.78	*(249K)*
8	1.94	*(16K)*	1.12	*(31K)*	0.87	*(62K)*	0.69	*(124K)*	0.79	*(249K)*	0.57	*(498K)*
16	1.13	*(31K)*	0.91	*(62K)*	0.71	*(124K)*	**0.56**	*(249K)*	0.68	*(498K)*	0.70	*(1M)*
32	1.37	*(62K)*	0.88	*(124K)*	0.67	*(249K)*	0.61	*(498K)*	0.77	*(1M)*	0.73	*(2M)*

Table 8: Error rate on MNIST for CNF-dense. Number of parameters given in parentheses.

Layers / Channels	4		8		16		32		64		128	
2	3.47	*(5K)*	1.46	*(21K)*	0.73	*(83K)*	0.65	*(331K)*	0.59	*(1M)*	0.59	*(5M)*
4	2.74	*(10K)*	0.88	*(42K)*	0.67	*(166K)*	0.54	*(663K)*	0.59	*(3M)*	0.51	*(10M)*
8	1.65	*(21K)*	0.70	*(83K)*	0.60	*(332K)*	0.52	*(1M)*	**0.39**	*(5M)*	0.46	*(21M)*
16	1.04	*(41K)*	0.60	*(166K)*	0.50	*(663K)*	0.55	*(2M)*	0.39	*(10M)*	0.47	*(42M)*
32	1.29	*(83K)*	0.92	*(332K)*	0.64	*(1M)*	0.57	*(5M)*	0.61	*(21M)*	0.56	*(85M)*

Table 9: Error rate on CIFAR10 for CNF-sparse. Number of parameters given in Table 11.

Layers / Channels	2	4	8	16	32	64	128	256
2	68.70	49.65	48.95	34.48	31.48	28.82	27.67	25.56
4	62.34	43.69	34.28	30.07	26.18	25.14	22.96	22.60
8	58.26	40.02	28.10	24.44	22.12	22.20	20.66	21.38
16	50.31	32.28	25.70	22.65	19.74	19.07	19.05	**18.89**

Table 10: Error rate on CIFAR10 for CNF-dense. Number of parameters given in Table 12.

Layers / Channels	2	4	8	16	32	64	128	256
2	68.70	50.96	33.66	23.92	18.83	15.72	13.79	13.11
4	62.34	41.92	27.76	19.22	14.65	13.38	12.09	10.06
8	58.26	35.12	22.53	15.57	13.05	10.88	9.42	**9.31**
16	50.31	28.27	19.03	13.57	10.95	9.65	10.63	14.27

Table 11: Number of parameters for CIFAR10, CNF-sparse.

Layers / Channels	2	4	8	16	32	64	128	256
2	*(1K)*	*(4K)*	*(8K)*	*(16K)*	*(31K)*	*(62K)*	*(124K)*	*(249K)*
4	*(3K)*	*(8K)*	*(16K)*	*(31K)*	*(62K)*	*(124K)*	*(249K)*	*(498K)*
8	*(5K)*	*(16K)*	*(31K)*	*(62K)*	*(124K)*	*(249K)*	*(498K)*	*(1M)*
16	*(10K)*	*(31K)*	*(62K)*	*(124K)*	*(249K)*	*(498K)*	*(1M)*	*(2M)*

Table 12: Number of parameters for CIFAR10, CNF-dense.

Layers / Channels	2	4	8	16	32	64	128	256
2	*(1K)*	*(5K)*	*(21K)*	*(83K)*	*(331K)*	*(1M)*	*(5M)*	*(21M)*
4	*(3K)*	*(10K)*	*(42K)*	*(166K)*	*(663K)*	*(3M)*	*(10M)*	*(42M)*
8	*(5K)*	*(21K)*	*(83K)*	*(332K)*	*(1M)*	*(5M)*	*(21M)*	*(85M)*
16	*(10K)*	*(41K)*	*(166K)*	*(663K)*	*(2M)*	*(10M)*	*(42M)*	*(170M)*