Convolutional Neural Fabrics

Shreyas Saxena **Jakob Verbeek**
INRIA Grenoble, Laboratoire Jean Kuntzmann
Grenoble, France
firstname.lastname@inria.fr

Abstract

Despite the success of convolutional neural networks, 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 CNN 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 the model (nr. of channels and layers) are not critical for performance. While individual CNN architectures can be recovered as paths in the trellis, the trellis can in addition ensemble all embedded architectures together, sharing their weights where their paths overlap. The trellis 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. The results of Krizhevsky *et al*. [14] in particular 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, (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 CNN and deconvolutional 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 the trellis embeds different architectures. Each activation in the trellis 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, nr. 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%.

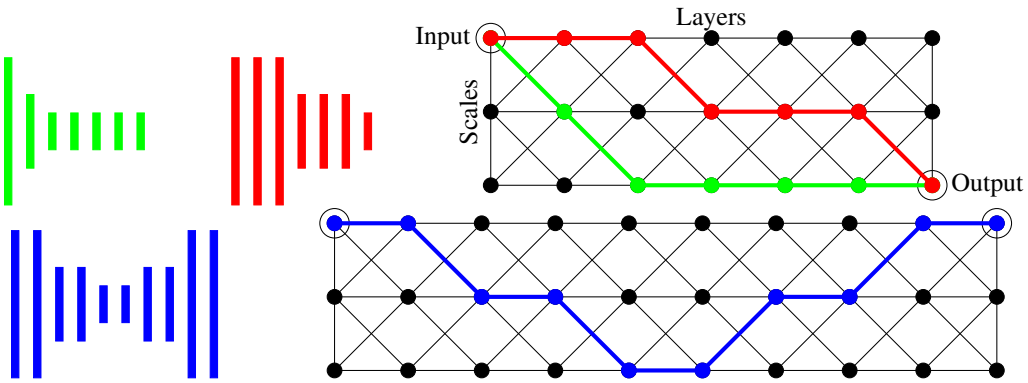

Figure 1: Trellis embedding of two seven-layer CNNs (red, green) and a ten-layer deconvolutional network (blue). Feature map size of the CNN layers are given by height. Trellis nodes receiving the input and producing output are encircled. All edges are oriented to the right, down in the first layer,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

All chain-structured (de)convolutional architectures embedded in the fabric can be recovered appropriately setting certain weights to zero, so that only a single activation path is non-zero along the scale and layer axes. General non-path weight settings correspond to ensembling many architectures together, which share parameters where edges 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 architectures and, more generally, mixtures of all of them. Experimental results competitive with the state of the art validate the effectiveness of our approach.

Our contributions are: (1) Our fabric allows to by and large sidestep the CNN model architecture selection problem. Avoiding explicitly training and evaluating individual architectures using, *e.g.*, greedy local-search strategies [3]. (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 between 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, by connecting a prediction layer to nodes at different scales in the last layer.

Below, we first discuss related work, and then present our trellis model in Section 3. We present a selection of our experimental results in Section 4 (more in Appendix A), and conclude in Section 5.

2 Related work

The chain-structured CNN architecture used by Krizhevsky *et al*. [14] for ImageNet classification is widely used for many other vision tasks. Other widely adopted architectures are the VGG-16 and VGG-19 networks of Simonyan and Zisserman [27]. Although effective for many tasks, it is by no means clear that these architectures are among the best ones given their computational and memory requirements. Their widespread adoption is at least in part due to the lack of more effective techniques to find good architectures than extremely costly brute-force exhaustive or local search [3].

Many of the (de)convolutional neural networks used for semantic segmentation, as well as other structured prediction tasks such as pose estimation [25], are often based on the CNN architectures developed for image classification of [14, 27], see *e.g.* [2, 8, 18, 21, 24, 30, 32]. In addition to convolution and pooling operators, deconvolutional networks also involve up-sampling operators to increase the resolution of the predictions in the output layer [8, 24]. Ronneberger *et al*. [26] present a deconvolutional network with additional links that couple distant convolutional and deconvolutional layers of the same resolution. Multi-scale architectures are commonly used for semantic segmentation [2, 4, 18]. They re-sample the input image to several resolutions, process them independently for several layers, and then fuse all streams by up-sampling to the maximum resolution. The streams are then fused, and passed through several more layers to produce the final output. Such architectures can be thought of as a tree that has leafs in different re-scaled input images, and the root at the output. In the next section we show that nearly all of the (de)convolutional networks discussed above are embedded in our fabric, either as paths or other simple sub-graphs.

The work of Zhou *et al*. [33] is closely related to ours. They interlink CNN models that take input from re-scaled versions of the input image. The structure of their model is related to our trellis, but lacks a sparse connectivity pattern across the channel dimension. Moreover, they set the number of channels and filter size per layer and scale in an ad-hoc manner. They explored their networks only for semantic segmentation, not for image classification. Finally, they did not observe that structures similar to theirs suffice to span a vast class of (de)convolutional networks, which is our main result.

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 the architecture of [14], and does not address the network architecture selection problem.

Springenberg *et al*. [28] experimentally observed that the use of max-pooling in CNN architectures is not always beneficial as opposed to only using strided convolutions. In our work we go one step further and show that ReLU units and (strided) convolutions suffice implement max-pooling operators in our trellis. 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.

While multi-dimensional networks have been proposed in the past, *e.g*. to process non-sequential data with recurrent nets [6, 12], to the best of our knowledge they have not been explored as a "basis" to span large classes of (de)convolutional neural networks.

3 The fabric of convolutional neural networks

We now give a precise definition of our trellis model, and show in Section 3.2 that most architectural design choices of (de)convolutional nets become irrelevant for sufficiently large trellises. Finally, we analyze the number of response maps, parameters, and activations of our trellises in Section 3.3.

3.1 Weaving the convolutional neural fabric

Each node in the trellis represents a response map with the same dimension as the input signal (1D for audio, 2D for images, 3D for video). The structure of the trellis 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. In practice we use S scales when we process inputs of size 2^{S-1}, *e.g*. for 32×32 images we use six scales, so as to obtain a full scale pyramid from the input resolution all the way to the coarsest 1×1 activations.

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*. channel c at scale s receives input from channels $\{c-1, c, c+1\}$ at scales $\{s-1, s, s+1\}$ in the previous layer. 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. Activations in the trellis are thus a linear function over multi-dimensional neighborhoods, *i.e*. a four dimensional $3\times3\times3\times3$ neighborhood when using 2D input images. The propagation is, however, only convolutional across the input dimensions, and not across the scale and layer axes. Note that "fully connected" layers of a CNN correspond to nodes and edges along the coarsest 1×1 scale of the trellis. Rectified linear units (ReLUs) are used at all nodes. Figure 1 illustrates the connectivity pattern in 2D, omitting the channel dimension for clarity.

All channels in the first layer at finest resolution are connected to all channels in 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 $\{c-1, c, c+1\}$ from the previous 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 in the trellis, and that the edge-structure within the first and last layer is reminiscent of the 2D trellis in Figure 1.

3.2 Stitching chain-structured networks on the fabric

We now explicitly show how various architectures are embedded in the trellis, demonstrating it subsumes essentially all (de)conv. models discussed above. In practice, learning configures the trellis to behave as one architecture or another, but generally as an ensemble of all embedded architectures.

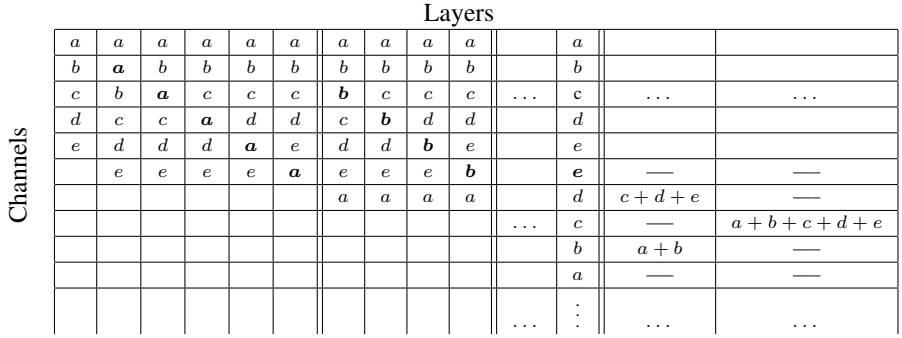

Figure 2: Schematic representation of a dense-channel-connect layer in our sparse trellis using local copy and swap operations. The five input channels $a, \ldots, e$ are first copied; more copies are generated by repetition. Channels are then convolved and locally aggregated in the last two layers to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

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. Several operators are used in (de)convolutional networks to change the resolution of response maps, *strided convolution* is the most basic one. Stride-two convolutions are used in the trellis on all fine-to-coarse edges, larger strides can be obtained by following multiple such edges. *Average pooling* across small regions is also strided convolution with uniform filter weights, and thus available in the trellis. See the next paragraph for averaging pooling over larger areas.

Bi-linear interpolation is commonly used in deconvolutional networks. Factor-two interpolation can be represented in our trellis on coarse-to-fine edges by using a filter that has 1 in the center, $1/4$ on corners, and $1/2$ elsewhere. Interpolation by larger powers of two can be obtained by repetition.

Consider *max-pooling* over a 2×2 region, larger sizes are obtained by repetition. 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. This process can also be adapted to show that a network of *MaxOut units* [5] can be implemented in a trellis of ReLU units.

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 one, 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 in this way, not only factorized approximations, *c.f.* [27]. Repetition allows to implement every filter of any desired size.

Ordering convolution and re-sampling. As shown in Figure 1, chain-structured (de)convolutional nets correspond to paths in the trellis. If weights on edges outside a path are set to zero, a chain-structured (de)convolutional net 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.* [4, 21, 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 [4].

Channel connectivity pattern. Although most networks in the literature use dense connectivity across channels between successive layers, this is not a necessity. For example, Krizhevsky *et al.* [14] used a network that is partially split across two independent processing streams.

In Figure 2 we demonstrate that our trellis, 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

Table 1: Number of response maps, parameters, and activations for a trellis with L layers, S scales, C channels, and D dimensional input. Number of channels is doubling on coarser scales in bottom row.

# chan. / scale	# resp. maps	# parameters (sparse)	# parameters (dense)	# activations
constant	$C \cdot L \cdot S$	$C \cdot L \cdot 3^{D+1} \cdot 3 \cdot S$	$C \cdot L \cdot 3^{D+1} \cdot C \cdot S$	$C \cdot L \cdot 2^{D(S-1)} \cdot \frac{4}{3}$
doubling	$C \cdot L \cdot 2^S$	$C \cdot L \cdot 3^{D+1} \cdot 3 \cdot 2^S$	$C \cdot L \cdot 3^{D+1} \cdot C \cdot 4^S \cdot \frac{4}{9}$	$C \cdot L \cdot 2^{D(S-1)} \cdot 2$

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 trellis 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 exists for any 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.

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 trellis, and the number of parameters when channels are sparsely or densely connected. We also state the number of activations in the trellis, 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.

The number of parameters is linear in the 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. The number of activations grows exponentially with the number of scales, since each scale layer doubles the resolution response maps. In other words, the number of scales is logarithmic in the input size, *e.g.* six for 32×32 images, and nine for 256×256 images.

As an example, the largest models we trained for 32×32 input have $L = 16$ layers and $C = 256$ channels, resulting in 2M parameters (255M 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 (15M for dense), and 89M activations. For reference, the VGG-19 model has 144M parameters and 14M activations.

In addition to the trellis structure defined above, we analyze a second trellis in the second row of Table 1; here the number of channels doubles when moving one scale coarser instead of being constant. In the case of sparsely connected channels, 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. Whereas before a node at scale s took input from three channels from scales $\{s-1, s, s+1\}$, it now takes two inputs from scale $s-1$, three from scale s, and four from scale $s+1$.

This results in a number of channels throughout the scale pyramid that is exponential in the number of scales S for a given number of "base channels" at the finest resolution C. For 32×32 input images the total number of channels is roughly $11\times$ 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%. Since each node still takes nine inputs from the previous layer, the computational cost grows also only by 50%. In case of sparse channel connection, the number of parameters grows by the same factor $2^S/S$. In case of dense connections, however, the number of parameters explodes with a factor $\frac{4}{9}4^S/S$. That is, roughly a factor 303 for 32×32 input, and 12,945 for 256×256 input. Therefore, the channel-doubling variant of our trellis is very attractive, provided that sparse channel connectivity is used. Similar channel-doubling is also used in the well-known VGG-16/19 architectures [27].

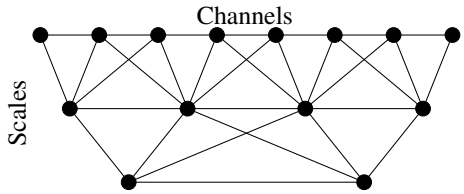

Figure 3: Diagram of sparse channel connectivity from one layer to another in the channel-doubling trellis. Channels are laid out horizontally, scales vertically. Each internal node, response map, is connected to nine other nodes at the previous layer: four channels at a coarser resolution, two at a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

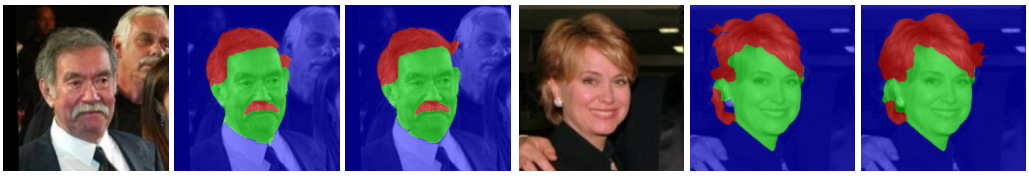

Figure 4: Examples form the Part Labels test set: input image (left), ground-truth labels (middle), and superpixel-level labels from our sparse CNF model with 8 layers and 16 channels (right).

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 [11] consists of 2,927 face images from the LFW dataset [9], 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 performance in terms of pixel-level and superpixel-level accuracy. For superpixel we average the class probabilities over the contained pixels. We did not use 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.* [19, 5], and pad the images with zeros to a 40×40 image and then take a random 32×32 crop, in addition we add flipped versions of these images.

Training. We train our trellises using SGD[1] with momentum of 0.9. After each node in the trellis we apply batch normalization [10], and regularize the model with weight decay of 10^{-4}, but did not apply dropout [29]. 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.

4.2 Experimental results

For all three datasets we trained sparse and dense trellises with various numbers of channels and layers. In all cases we used a constant number of channels per scale, experiments with channel-doubling are ongoing and will be included in the camera ready version. The results across all these settings can be found in Appendix A, 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 [30, 32], but the latter are based on the VGG-16 network. That network

[1] Upon publication we release our implementation based on Caffe (`http://caffe.berkeleyvision.org`).

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

	Year	# Params.	SP Acccur.	P Accur.
Tsogkas *et al*. [30]	2015	>414 M	96.97	—
Zheng *et al*. [32]	2015	>138 M	96.59	—
Liu *et al*. [20]	2015	>33 M	—	95.24
Kae *et al*. [11]	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

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
Grid LSTM [12]	2016	T		0.32
Dropconnect [31]	2013	T+F	379K	0.32
CKN [22]	2014	N	43 K	0.39
Maxout [5]	2013	N	420 K	0.45
Network in Network [19]	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

has $1380\times$ 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, [30] also includes CRF and RBM for spatial regularization. Our sparse convolutional neural fabric (CNF) results improve over [11] which also employ CRF and RBM shape models. In contrast, we predict all pixels independently using a model with about seven times less parameters.

MNIST. We obtain error rates of 0.48% and 0.33% with sparse and dense trellises 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 trellis is comparable to those of [12, 31], which use similar data augmentation. Our sparse model, which has $20\times$ 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 trellis is comparable to that reported with MaxOut networks [5]. On this dataset the error of the sparse model, 19.6%, is significantly worse. We are investigating the reason for this exception.

Visualization. In Figure 5 we visualize the weight strengths of learned trellis models. We observe qualitative differences between these models. 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. The classification model the signal is propagated in a band-diagonal pattern, exploiting multiple scales in each layer.

5 Conclusion

We presented convolutional neural fabrics: homogeneous and sparsely connected three-dimensional trellises over response maps. The fabric subsumes a large class of (de)convolutional networks. It sidesteps the tedious process of specifying, training, and testing individual networks in order to find good architectures. The fabric has only two main design 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 a variant with dense channel connectivity, and one with channel-doubling over scales. The latter strikes a very attractive capacity/memory trade-off.

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 (%)
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) [28]	2015	T+F	1.3 M	7.25
Lin *et al*. (Network in Network) [19]	2013	T+F	1 M	8.81
Wan *et al*. (Dropconnect) [31]	2013	T+F+S+R	19M	9.32
Goodfellow *et al*. (MaxOut) [5]	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	32M	7.43

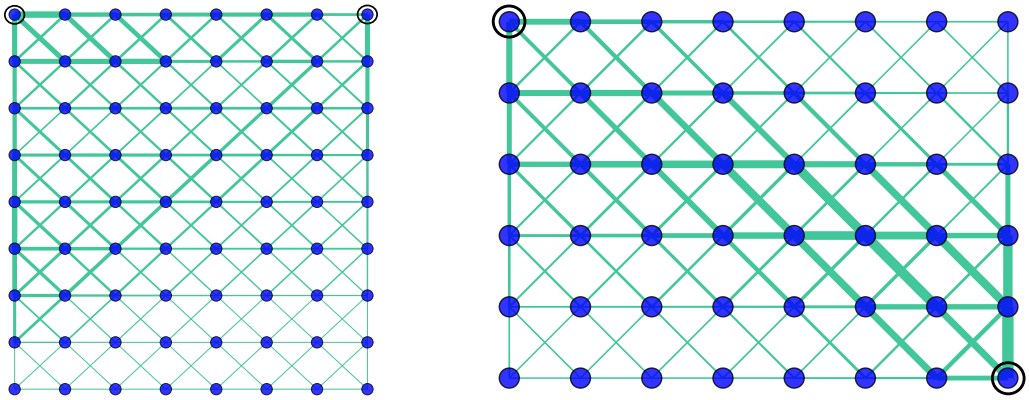

Figure 5: Visualization of mean-squared filter weights (mean along weights and channels) in models learned for Part Labels (left) and MNIST (right). Layers are laid out horizontally, and scales vertically.

In our experiments we study performance of 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 demonstrate that our generic fabric approach is competitive with the best hand-crafted CNN architectures. We expect that these results can be further improved by using better optimization schemes such as Adam[13], using dropout [29] or dropconect [31] regularization, and using MaxOut units [5] and/or residual units [7] to facilitate training of deep fabrics with many channels.

In ongoing work we analyze our model in terms of the frequencies and sampling rates of signals flowing through the trellis. We also conduct experiment with channel-doubling fabrics, and joint classification and semantic segmentation models using PASCAL VOC, MS COCO, and ImageNet.

References

[1] J.-R. Chang and Y.-S. Chen. Batch-normalized maxout network in network. Arxiv preprint, 2015.

[2] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *ICLR*, 2015.

[3] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. In *ICLR*, 2016.

[4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *PAMI*, 35(8):1915–1929, 2013.

[5] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013.

[6] A. Graves, S. Fernández, and J. Schmidhuber. Multi-dimensional recurrent neural networks. In *ICANN*, 2007.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. Arxiv preprint.

[8] S. Hong, H. Noh, , and B. Han. Decoupled deep neural network for semi-supervised semantic segmentation. In *NIPS*, 2015.

[9] 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.

[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[11] A. Kae, K. Sohn, H. Lee, and E. Learned-Miller. Augmenting CRFs with Boltzmann machine shape priors for image labeling. In *CVPR*, 2013.

[12] N. Kalchbrenner, I. Danihelka, and A. Graves. Grid long short-term memory. In *ICLR*, 2016.

[13] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[14] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[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*, 2015.

[18] G. Lin, C. Shen, A. van den Hengel, and I. Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *CVPR*, 2016.

[19] M. Lin, Q. Chen, and S. Yan. Network in network. Arxiv preprint, 2013.

[20] S. Liu, J. Yang, C. Huang, , and M.-H. Yang. Multi-objective convolutional learning for face labeling. In *CVPR*, 2015.

[21] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[22] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *NIPS*, 2014.

[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] J. Springenberg, A. Dosovitskiy andT. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR*, 2015.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[30] S. Tsogkas, I. Kokkinos, G. Papandreou, and A. Vedaldi. Deep learning for semantic part segmentation with high-level guidance. Arxiv preprint, 2015.

[31] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using DropConnect. In *ICML*, 2013.

[32] 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.

[33] Y. Zhou, X. Hu, and B. Zhang. Interlinked convolutional neural networks for face parsing. In *International Symposium on Neural Networks*, 2015.

A Additional Experimental results

Here we give detailed experimental results obtained using sparse and dense trellises 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 trellises 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 trellis model, it simply suffices to take them large enough.

A.1 Detailed experimental results on Part Labels

Two configurations missing in Table 5 and Table 6, we were not able to complete these experiments in time, we will include them in the camera ready.

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)*	—	

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)*	—	

A.2 Detailed experimental results on MNIST

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	*(8K)*	1.46	*(31K)*	0.73	*(124K)*	0.65	*(498K)*	0.59	*(2M)*	0.59	*(8M)*
4	2.74	*(16K)*	0.88	*(62K)*	0.67	*(249K)*	0.54	*(1M)*	0.59	*(4M)*	0.51	*(16M)*
8	1.65	*(31K)*	0.70	*(124K)*	0.60	*(498K)*	0.52	*(2M)*	**0.39**	*(8M)*	0.46	*(32M)*
16	1.04	*(62K)*	0.60	*(249K)*	0.50	*(1M)*	0.55	*(4M)*	0.39	*(16M)*	0.47	*(64M)*
32	1.29	*(124K)*	0.92	*(498K)*	0.64	*(2M)*	0.57	*(8M)*	0.61	*(32M)*	0.56	*(128M)*

A.3 Detailed experimental results on CIFAR10

Table 9: Error rate on CIFAR10 for CNF-sparse. Number of parameters given in table 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 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	*(2K)*	*(4K)*	*(8K)*	*(16K)*	*(31K)*	*(62K)*	*(124K)*	*(249K)*
4	*(4K)*	*(8K)*	*(16K)*	*(31K)*	*(62K)*	*(124K)*	*(249K)*	*(498K)*
8	*(8K)*	*(16K)*	*(31K)*	*(62K)*	*(124K)*	*(249K)*	*(498K)*	*(1M)*
16	*(16K)*	*(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	*(2K)*	*(8K)*	*(31K)*	*(124K)*	*(498K)*	*(2M)*	*(8M)*	*(32M)*
4	*(4K)*	*(16K)*	*(62K)*	*(249K)*	*(1M)*	*(4M)*	*(16M)*	*(64M)*
8	*(8K)*	*(31K)*	*(124K)*	*(498K)*	*(2M)*	*(8M)*	*(32M)*	*(127M)*
16	*(16K)*	*(62K)*	*(249K)*	*(1M)*	*(4M)*	*(16M)*	*(64M)*	*(255M)*