Published in PNAS on December 26, 2017*

A mixed-scale dense convolutional neural network for image analysis

Daniël M. Pelt[1] and James A. Sethian[1,2]

***Abstract*— Deep convolutional neural networks have been successfully applied to many image processing problems in recent works. Popular network architectures often add additional operations and connections to the standard architecture to enable training deeper networks. To achieve accurate results in practice, a large number of trainable parameters is often required. Here, we introduce a network architecture based on using dilated convolutions to capture features at different image scales, and densely connecting all feature maps with each other. The resulting architecture is able to achieve accurate results with relatively few parameters and consists of a single set of operations, making it easier to implement, train, and apply in practice, and automatically adapts to different problems. We compare results of the proposed network architecture with popular existing architectures for several segmentation problems, showing that the proposed architecture is able to achieve accurate results with fewer parameters, with a reduced risk of overfitting the training data.**

Introduction

Machine learning is successful in many imaging applications, such as image classification [1]–[3] and semantic segmentation [4]–[6]. Many applications of machine learning to imaging problems use *deep convolutional neural networks* (DCNNs), in which the input image and intermediate images are convolved with learned kernels in a large number of successive layers, allowing the network to learn highly nonlinear features. The popularity of machine learning has grown significantly due to (a) recent developments that allow for effective training of deeper networks, e.g. the introduction of rectified linear units [7] and dropout layers [8]; (b) the public availability of highly optimized software to both train and apply deep networks, e.g. TensorFlow [9] and Caffe [10]; and (c) the public availability of large pretrained networks and large training data sets, e.g. VGG [2] and ImageNet [11], and will continue to be an active research area [12].

To achieve accurate results for difficult image processing problems, DCNNs typically rely on combinations of additional operations and connections including, for example, downscaling and upscaling operations to capture features at various image scales [4], [5]. To train deeper and more powerful networks, additional layer types [8], [13] and connections [14], [15] are often required. Finally, DCNNs typically use a large number of intermediate images and trainable parameters (e.g. more than 100 million [2]) to achieve results for difficult problems.

The large size and complicated nature of many DCNNs bring significant challenges. For example, the chosen combination of layers and connections can significantly influence the accuracy of trained networks. Determining which combination is best for a given problem is difficult to predict a priori. Consequently, a network that works well for one problem is not guaranteed to work well for a different problem, and can require significant changes to achieve accurate results. Furthermore, the large number of parameters to learn during training requires careful choices of hyperparameters (e.g. learning rates and initialization values) to avoid problems such as overfitting [8] and vanishing gradients [13] that result in inaccurate trained networks. As a result, image analysis often relies on problem-specific traditional methods instead.

Here, we introduce a network architecture specifically designed to be easy to implement, train, and use. All layers of the network use the same set of operations and are connected to each other in the same way, removing the need to choose which operations and connections to use for each specific problem. Our proposed network architecture achieves accurate results with relatively few intermediate images and parameters, eliminating both the need to tune hyperparameters and additional layers or connections to enable training. The network uses *dilated* convolutions instead of scaling operations to capture features at various image scales, employing multiple scales within a single layer, and densely connecting *all* intermediate images with each other. During training, the network learns which combinations of dilations to use for the given problem, allowing the same network to be applied to different problems.

This paper is structured as follows. We first introduce notation and discuss the general structure of existing deep convolutional networks. We then introduce the proposed network architecture. We explain the experiments we performed to investigate the performance of the architecture, comparing with popular existing architectures, and discuss their results. Finally, we conclude with a summary and final remarks.

Notation and concepts

Problem definition

In this paper, we apply our approach to real-valued two-dimensional (2D) images. We define an image as a set of pixels $\boldsymbol{x} \in \mathbb{R}^{m \times n \times c}$ with m rows, n columns, and c *channels*. We denote the image corresponding to a single channel j of $\boldsymbol{x}$ as $\boldsymbol{x}^j$. Many image processing problems can be written as the problem of finding a function f that takes a certain image $\boldsymbol{x}$ and produces an output image $\boldsymbol{y}$, i.e. $f : \mathbb{R}^{m \times n \times c} \to \mathbb{R}^{m' \times n' \times c'}$. Note that the dimensions of the output image can be different from those of the input

*Available at PNAS Online: https://doi.org/10.1073/pnas.1715832114

[1]Center for Applied Mathematics for Energy Research Applications (CAMERA), Lawrence Berkeley National Laboratory, Berkeley, CA 94720

[2]Department of Mathematics, University of California, Berkeley, CA 94720

image. In image classification problems, for example, the output image consists of a single probability value for each of the c' possible classifications, i.e. $m' = n' = 1$. In the rest of this paper, however, we will focus on problems with *dense* outputs, i.e. with the number of rows and columns of the output image identical to those of the input image: $m' = m$ and $n' = n$, similar to "pixel to pixel" architectures [16].

Convolutional neural networks

Convolutional neural networks (CNNs) model the unknown function f by using several *layers* that are connected to each other in succession. Each layer i produces an output image $\boldsymbol{z}_i \in \mathbb{R}^{m_i \times n_i \times c_i}$, called a *feature map*, using output of the previous layer $\boldsymbol{z}_{i-1}$ as input. The dimensions of the layer output $\boldsymbol{z}_i$ can be different from the layer input $\boldsymbol{z}_{i-1}$. The input image $\boldsymbol{x}$ is taken as the first layer $\boldsymbol{z}_0$, and the final layer produces the output image $\boldsymbol{y}$.

Each individual layer can consist of multiple operations. A common layer architecture first convolves each channel of the input feature map with a different filter, then sums the resulting convolved images pixel-by-pixel, adds a constant value (the *bias*) to the resulting image, and finally applies a nonlinear operation to each pixel. These operations can be repeated using different filters and biases to produce multiple channels for the output feature map. Thus, the output $\boldsymbol{z}_i^j$ of a single channel j of such a *convolutional* layer is given by

$$\boldsymbol{z}_i^j = \sigma\left(g_{ij}(\boldsymbol{z}_{i-1}) + b_{ij}\right) \tag{1}$$

Here, $\sigma : \mathbb{R}^{m_i \times n_i} \to \mathbb{R}^{m_i \times n_i}$ is a nonlinear operation such as the popular sigmoid function or rectified linear unit (ReLU) [7], $b_{ij} \in \mathbb{R}$ is the bias, and $g_{ij} : \mathbb{R}^{m_{i-1} \times n_{i-1} \times c_{i-1}} \to \mathbb{R}^{m_i \times n_i}$ convolves each channel of the input feature map with a different filter and sums the resulting images pixel-by-pixel:

$$g_{ij}(\boldsymbol{z}_{i-1}) = \sum_{k=0}^{c_{i-1}} C_{\boldsymbol{h}_{ijk}} \boldsymbol{z}_{i-1}^k \tag{2}$$

where $C_{\boldsymbol{g}}\boldsymbol{a}$ is a 2D convolution of image $\boldsymbol{a}$ with filter $\boldsymbol{g}$. Different ways of handling the boundaries of the image during convolution are possible: here, we use reflective boundaries. Often, the filters $\boldsymbol{h}_{ijk}$ are relatively small (e.g. 3×3 pixels), enabling faster computation of network outputs and making the network easier to train. The architecture of the final layer can differ from other layers, and can depend on the application: common choices include using a fully connected layer instead of a convolutional one [2], or a softmax function as the nonlinear operation for classification problems [5]. A schematic of a two-layer CNN architecture is shown in Fig. 1.

The goal of *training* a CNN is to find filters $\boldsymbol{h}_{ijk}$, biases b_{ij}, and potential other parameters, such that the CNN performs the task that is required. In *supervised learning*, training is achieved by using a set of N_t representative inputs $\boldsymbol{X} = \{\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_{N_t}\}$ with corresponding correct outputs $\boldsymbol{Y} = \{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{N_t}\}$ and iteratively minimizing a chosen error metric between $\boldsymbol{Y}$ and the CNN output for $\boldsymbol{X}$. Because

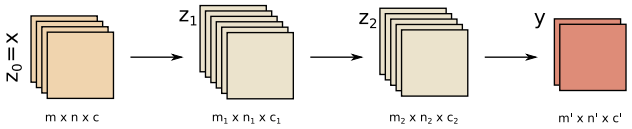

Fig. 1. A schematic representation of a two-layer CNN with input $\boldsymbol{x}$, output $\boldsymbol{y}$, and feature maps $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$. Arrows represent convolutions with nonlinear activation.

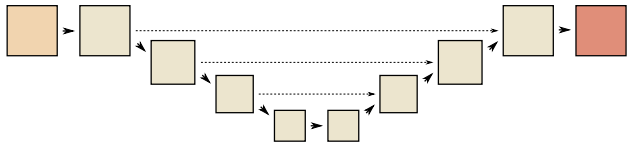

Fig. 2. A schematic representation of a common DCNN architecture with scaling operations. Downward arrows represent downscaling operations, upward arrows represent upscaling operations, and dashed arrows represent skip connections.

of the specific architecture of CNNs, partial gradients of the error with respect to the filters and biases can be computed accurately and efficiently through backpropagation for several popular error metrics, enabling the use of efficient gradient-based optimization algorithms [17].

Deep convolutional neural networks

Deep convolutional neural networks (DCNNs) use a network architecture similar to standard CNNs, but consist of a larger number of layers, which enables them to model more complicated functions. In addition, DCNNs often include downscaling and upscaling operations between layers, decreasing and increasing the dimensions of feature maps to capture features at different image scales. Many DCNNs incrementally downscale feature maps in the first half of the layers, called the *encoder* part of the network, and subsequently upscale in the second half, called the *decoder* part. Skip connections are often included between feature maps of the decoder and encoder at identical scales [5]. A schematic representation of a common encoder-decoder DCNN architecture is shown in Fig. 2.

In general, the increased depth of DCNNs compared with shallow CNNs makes training more difficult. The increased depth often makes it more likely that training gets stuck in a local minimum of the error function, and can result in gradients that become either too large or too small [13]. Furthermore, DCNNs typically consist of many parameters (e.g. filters and biases), often several million or more, that have to be learned during training. The large parameter space can make training more difficult, by increasing both training time [18], and the likelihood of overfitting the network to the training data [8], thereby forcing large training sets. Several additions to standard DCNN architectures have been proposed, including Batch Normalization layers [13], which rescale feature maps between layers to improve the scaling of gradients during training, highway connections [14], residual connections [15], and fractal networks [19], which allow information to flow more easily through deep networks by skipping layers, and Dropout layers [8], in which feature

maps are randomly removed from the network during training, reducing the problem of overfitting large networks.

Although these additions have advanced image processing in several fields [12], they can be difficult to routinely apply in areas such as biomedical imaging and materials science. Instead, traditional imaging algorithms are used, such as the Hough transform [20] and template matching [21], or manual processing, (e.g. biological image segmentation [22]).

Theory and Algorithms

Our goal is to enable easier application of DCNNs to many imaging problems by introducing a less complicated network architecture with significantly fewer parameters to learn and which is able to automatically adapt to different problems. To do so, we introduce "the Mixed-Scale Dense (MS-D)" network architecture, which (a) mixes scales within each layer and (b) densely connects all feature maps.

Mixing scales

Instead of using downscaling and upscaling operations to capture features at different scales, the MS-D architecture uses *dilated convolutions*. A dilated convolution $D_{\boldsymbol{h},s}$ with dilation $s \in \mathbb{Z}^+$ uses a dilated filter $\boldsymbol{h}$ that is only nonzero at distances that are a multiple of s pixels from the center.[1] Recently, it was shown that dilated convolutions are able to capture additional features in DCNNs that use the traditional scaling approach [23]. Furthermore, instead of having each layer operate at a certain scale as in existing DCNNs, in the Mixed-Scale approach each individual *channel* of a feature map within a single layer operates at different scale. Specifically, we associate the convolution operations for each channel of the output image of a certain layer with a different dilation:

$$g_{ij}(\boldsymbol{z}_{i-1}) = \sum_{k=0}^{c_{i-1}} D_{\boldsymbol{h}_{ijk},s_{ij}} \boldsymbol{z}_{i-1}^k \tag{3}$$

The proposed Mixed-Scale approach alleviates many of the disadvantages of the standard downscaling and upscaling approach. First, large-scale information about the image quickly becomes available in early layers of the network through relatively large dilations, making it possible to use this information to improve the results of deeper layers. Furthermore, information at a certain scale can be used directly to inform decisions at other scales without having to pass through layers at intermediate scales. Similar advantages were recently found when training large multigrid architectures [24]. No additional parameters have to be learned during training, since the Mixed-Scale approach does not include learned upscaling operations. This results in smaller networks that are easier to train. Finally, although dilations s_{ij} must be chosen in advance, the network can learn *which* combinations of dilations to use during training, making identical Mixed-Scale DCNNs applicable across different problems (see experiments below).

[1] Alternatively, dilated convolutions can be defined without using dilated filters by changing the convolution operation itself; see [23] for a detailed explanation.

Dense connections

When using convolutions with reflective boundaries, the Mixed-Scale approach has an additional advantage compared with standard scaling: all network feature maps have the same number of rows and columns as the input and output image, i.e. $m_i = m$ and $n_i = n$ for all layers i and hence, when computing a feature map for a specific layer, we are not restricted to using only the output of the previous layer. Instead, all previously computed feature maps $\{\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{i-1}\}$, including the input image $\boldsymbol{x}$, can be used to compute the layer output $\boldsymbol{z}_i$. Thus, we change the channel image computation (Eq. 1) and the convolutional operation (Eq. 3) to:

$$\begin{aligned} \boldsymbol{z}_i^j &= \sigma\left(g_{ij}(\{\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{i-1}\}) + b_{ij}\right) \\ g_{ij}(\{\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{i-1}\}) &= \sum_{l=0}^{i-1} \sum_{k=0}^{c_{l-1}} D_{\boldsymbol{h}_{ijkl},s_{ij}} \boldsymbol{z}_l^k \end{aligned} \tag{4}$$

Similarly, to produce the final output image $\boldsymbol{y}$, *all* feature maps can be used instead of only those of the last layer. We call this approach of using all previously computed feature maps *densely connecting* a network.

In a densely connected network, all feature maps are maximally (re-)used: if a certain useful feature is detected in a feature map, it does not have to be replicated in other layers to be used deeper in the network, as in other DCNN architectures. As a result, significantly fewer feature maps and trainable parameters are required to achieve the same accuracy in densely connected networks compared with standard networks. The smaller number of maps and parameters makes it easier to train densely connected networks, reducing the risk of overfitting and enabling effective training with relatively small training sets. Recently, a similar dense connection architecture was proposed which relied on a relatively small number of parameters [25], however, in [25] the dense connections were only used within small sets of layers at a single scale, with traditional downscaling and upscaling operations to acquire information at different scales. Here, we combine dense connections with the mixed-scale approach, enabling dense connections between the feature maps of the *entire* network, resulting in more efficient use of all feature maps, and an even larger reduction of the number of required parameters.

Mixed-Scale Dense neural networks

By combining mixed-scale dilated convolutions and dense connections, we can define a DCNN architecture that we call the Mixed-Scale Dense (MS-D) network architecture. Similar to existing architectures, an MS-D network consists of several layers of feature maps. Each feature map is the result of applying the same set of operations given by Eq. 4 to all previous feature maps: dilated convolutions with 3×3 pixel filters and a channel-specific dilation, summing resulting images pixel-by-pixel, adding a constant bias to each pixel, and finally applying a ReLU activation function. The final network output is computed with the same set of operations applied to *all* feature maps, using 1×1 pixel filters instead of 3×3 pixel filters. In other words,

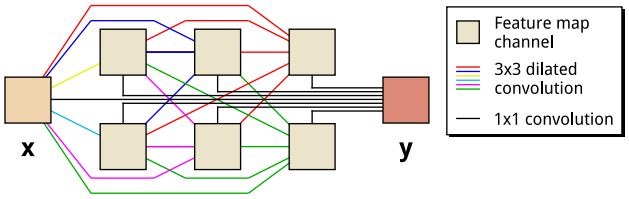

Fig. 3. Schematic representation of an MS-D network with $w = 2$ and $d = 3$. Colored lines represent 3×3 dilated convolutions, with each color representing a different dilation. Note that all feature maps are used for the final output computation.

channels of the final output image are computed by taking linear combinations of all channels of all feature maps, and applying an application-specific activation function to the result:

$$\boldsymbol{y}^k = \sigma' \left(\sum_{i,j} w_{ijk} \boldsymbol{z}_i^j + b'_k \right) \tag{5}$$

Different ways of choosing the number of channels per layer are possible. Here, we use a simple approach with each layer having the same number of channels, denoted by the network width w, and the number of non-input and non-output layers of the network denoted by the network depth d. A graphical representation of an MS-D network with $w = 2$ and $d = 3$ is shown in Fig. 3. The parameters that have to be learned during training are the convolution filters $\boldsymbol{h}_{ijkl}$ and biases b_{ij} of Eq. 4 and the weights w_{ijk} and biases b'_k of Eq. 5. Given a network depth d and width w, and number of input channels c_{in} and output channels c_{out}, the number of trainable parameters $N_{par} = N_{flts} + N_{wgts} + N_{bias}$ is given by $N_{flts} = 9 \sum_{i=0}^{d-1} w(iw + c_{in})$, $N_{wgts} = (wd + c_{in})c_{out}$, and $N_{bias} = wd + c_{out}$.

Compared with existing DCNN architectures, the MS-D network architecture has several advantages. Due to the mixing of scales through dilated convolutions and dense connections, MS-D networks can produce accurate results with relatively few feature maps and trainable parameters. Furthermore, an MS-D network learns which combination of dilations to use during training, allowing the same network to be effectively applied to a wide variety of problems. Finally, all layers are connected to each other in the same way and computed using the same set of standard operations, making MS-D networks easier to implement, train, and use in practice. MS-D networks do not include learned scaling operations or advanced layer types to facilitate training, and do not require architecture changes when being applied to different problems. These advantages can make MS-D networks applicable beyond semantic segmentation, with potential value in classification, detection, instance segmentation, and adversarial networks [16].

Experiments

Setup

We implemented the MS-D architecture in Python, using PyCUDA [26] to enable GPU acceleration of computationally expensive parts such as convolutional operations. We note that existing frameworks such as TensorFlow [9] or Caffe [10] typically do not support the proposed mixed-scale approach well, since they assume that all channels of a certain feature map are computed in the same way. Furthermore, existing frameworks are mostly optimized for processing large numbers of relatively small images by efficiently implementing convolutions using large matrix multiplications [27]. To allow the application of MS-D networks to problems with large images, we implemented the architecture using direct convolutions. Computations were performed on two workstations, with an NVidia GeForce GTX 1080 GPU and four NVidia Tesla K80 GPUs, respectively, all running CUDA 8.0.

In general, deeper networks tend to produce more accurate results than shallower network [2]. Because of the dense connections in MS-D networks, it is possible to effectively use networks that have many layers and few channels per layer, resulting in very deep networks with relatively few channels. Such very deep networks might be more difficult to train than shallower networks, as explained above. However, we did not observe such problems, and were able to use the extreme case of each layer consisting of only one channel ($w = 1$), and the number of layers d controlling the number of trainable parameters. We initialize all convolution filter parameters based on the same considerations as [3] by sampling random values from a zero-mean normal distribution with a standard deviation of $\sqrt{2/n_c}$, where n_c is the number of incoming and outgoing connections of a feature map channel: $n_c = 9(c_{in} + w(d-1)) + c_{out}$. All other trainable parameters are initialized to zero. Finally, in most experiments we use equally distributed dilations $s_{ij} \in [1, 10]$ by setting the dilation of channel j of layer i equal to $s_{ij} = ((iw + j) \bmod 10) + 1$.

In segmentation problems with L labels, we represent correct outputs by images with L channels, with channel j set to 1 for pixels that are assigned to label j and set to 0 for other pixels. We use the soft-max activation function in the final output layer, and use the ADAM optimization method [17] during training to minimize the *cross-entropy* between correct outputs and network outputs [5]. To compare results of MS-D networks with existing architectures for segmentation problems, we use the global accuracy metric [4], defined as the percentage of correctly labeled pixels in the network output, and the class accuracy metric [4], computed by taking the average of the true positive rates for each individual label.

Simulated data

In a first experiment, network input consist of 512×512 pixel single-channel images of objects with two shapes (circles and squares), 3 different sizes, and 6 possible textures, with added Gaussian noise. Out of all 36 combinations of shape, size, and texture, we train networks to detect six specific combinations, e.g. large squares with a horizontal texture, small circles with a diagonal texture, etc. We chose this segmentation problem because it requires DCNNs to combine features at small scales (pixel intensity and texture)

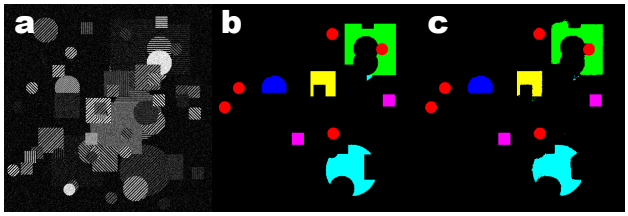

Fig. 4. Example of the segmentation problem of the simulated dataset, with (a) the single-channel input image, (b) the correct segmentation, with labels indicated by color, and (c) the output of a trained MS-D network with 200 layers.

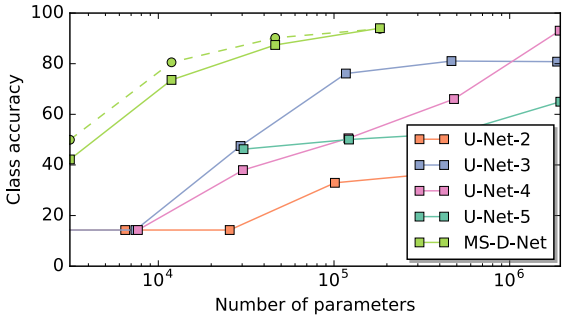

Fig. 5. The class accuracy of a set of 100 simulated images (Fig. 4) as a function of the number of trainable parameters for the proposed MS-D network architecture and the popular U-Net architecture. For each U-Net network (U-Net-q), q indicates the number of scaling operations used. For the MS-D architecture, results are shown for dilations $s_{ij} \in [1, 10]$ (solid line) and $s_{ij} \in \{1, 2, 4, 8, 16\}$ (dashed line).

with features at larger scales (size and shape) to produce accurate results. An example input is shown in Fig. 4a, with colors indicating the six combinations that have to be detected in Fig. 4b. We compare segmentation results of trained MS-D networks with those of the popular U-Net architecture [5]: We use a TensorFlow implementation [28]. U-Net architectures are similar to that shown in Fig. 2. Two main parameters influence performance: the number of downscaling (and subsequent upscaling) operations, and the number of channels per feature map. We train each network with the same set of 10^5 randomly generated images, using a batch size of one image, and stop training once global accuracy for a different set of 100 *validation* images has not improved for 10^5 iterations.

In Fig. 5, class accuracy for an independent test set of 100 images is shown as a function of the number of trainable parameters for MS-D networks with $w = 1$ and $d \in \{25, 50, 100, 200\}$ layers, and U-Net networks with 2, 3, 4, and 5 scaling operations and various numbers of channels. The performance of the U-Net networks depends significantly on the chosen number of scaling operations. Networks with 3 scaling operations are able to achieve around 80% accuracy with relatively few parameters, but do not improve significantly when using more channels per feature map, while networks with 4 scaling operations are able to achieve around 95% accuracy, but require a large number of parameters to do so. For a given number of parameters, MS-D networks are able to achieve significantly higher accuracies than all tested U-Net architectures, especially with relatively few parameters, and the performance of MS-D networks is similar for different choices of dilations.

CamVid dataset

Next, we compare results for the CamVid dataset [29], using 367 training, 101 validation, and 233 testing color images of road scenes with 360×480 pixels [4]. The goal is to segment 11 classes such as cars, roads, sidewalks, and pedestrians. We train MS-D networks and U-Net networks with local contrast normalized images [30] until no improvement in global accuracy of the validation set, using minibatches of 10 images for MS-D networks and smaller minibatches of 3 images for U-Net networks due to memory constraints. We also report results for the SegNet architecture [4], showing the two best global accuracy results from Table 1 of [4], and two traditional segmentation methods [31], [32], showing the two best results from Table 2 of [4]. For U-Net networks, the number of feature map channels was chosen such that the number of parameters was similar to that of the SegNet.

Method	Pars (M)	GA	CA
MS-D-Net (100 layers)	**0.048**	85.1	56.8
MS-D-Net (200 layers)	0.187	**87.0**	**63.9**
U-Net (3 scaling operations) [5]	1.863	83.2	50.4
U-Net (4 scaling operations) [5]	1.926	85.5	48.4
SegNet-Basic-EncoderAddition [4]	1.425	84.2	56.5
SegNet-Basic [4]	1.425	84.0	54.6
Boosting+Detectors+CRF [31]		83.8	62.5
Super Parsing [32]		83.3	51.2

Table I. The number of trainable parameters (Pars) in millions (M), global accuracy (GA), and class accuracy (CA) for the CamVid test set. The highest global accuracy, highest local accuracy, and smallest number of parameters out of all tested methods are shown in bold.

Table. I shows global and class accuracies. MS-D segments with highest global and class accuracy, while using roughly 10 times fewer parameters. Furthermore, an MS-D network with 100 layers achieves similar accuracies to other network architectures while using 30 to 40 times fewer parameters.[2] Fig. 6 shows global accuracy during training for validation and training sets, for both U-Net network and an MS-D network. Lack of improvement for the U-Net network in validation set accuracy, and its difference with training set accuracy, indicate overfitting of the chosen training set. Due to the smaller number of trainable parameters, the MS-D network improves validation set accuracy for more training iterations, with significantly smaller difference with training set accuracy, showing reduced risk of overfitting of MS-D networks, and the ability to accurately train with relatively small training sets. In addition, MS-D networks are able to achieve accurate results without pretraining additional large datasets, e.g. ImageNet [11], or relying on large pretrained networks, e.g. VGG [2].

[2]The authors of [4] report improved results for the SegNet architecture with 90.4% global accuracy by training with a significantly larger set of around 3500 images. However, since this larger set is not publicly available, we cannot directly compare this result with the MS-D network architecture.

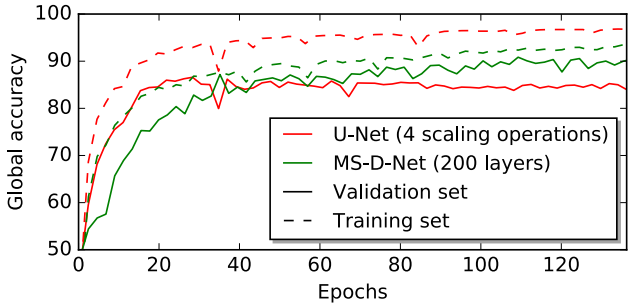

Fig. 6. The global accuracy of a U-Net network and an MS-D network as a function of the training epoch for the CamVid dataset. Given are the accuracies for the validation set (solid lines) and the training set (dashed lines).

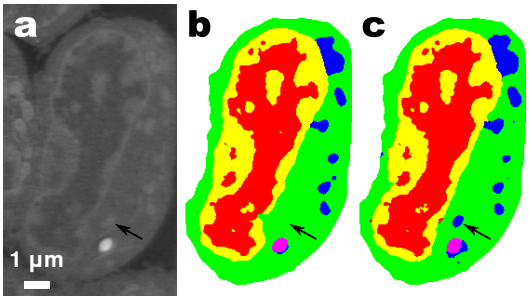

Fig. 7. A tomographic slice of the test cell (a), with the corresponding manual segmentation (b) and output of an MS-D network with 100 layers (c).

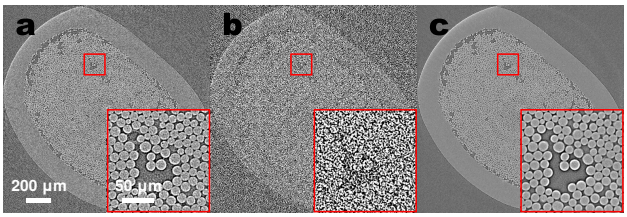

Fig. 8. Tomographic images of a fiber-reinforced mini-composite, reconstructed using 1024 projections (a) and 128 projections (b). In (c), the output of an MS-D network with image (b) as input is shown. A small region indicated by a red square is shown enlarged in the bottom-right corner of each image.

Segmenting biomedical images

To test whether an MS-D network can be easily applied to a new problem without adjustments, we use the same network parameters as above, with $w = 1$, $d = 100$, and dilations $s_{ij} \in [1, 10]$ applied to segmenting cell structures. We use eight manual segmentations of $512 \times 512 \times 512$ tomographic reconstructions of (mouse) lympoblastoid cells, consisting of five labels: nuclear envelope, euchromatin, heterochromatin, mitochondria and lipid drops. A sample tomographic slice and corresponding manual segmentation are shown in Fig. 7a and Fig. 7b. The labeling of cell structures depends on multiple factors at different image scales, such as the position of the structure relative to other structures, and the pixel intensity differences between two structures can be relatively small, making it difficult to use traditional methods to perform automatic labeling. Instead, researchers rely on time-consuming manual segmentation.

To learn limited 3D features, we use five channels in the input image of the MS-D network: the current slice to be segmented and four adjacent slices. Out of eight manual cell segmentations, we randomly chose six for training, one for validation, and report results for the remaining cell. During training, we used a batch size of 10 images, and stopped after no improvements in global accuracy for the validation cell, yielding network parameters with the best global accuracy. Fig. 7c shows network output for the slice of Fig. 7a, showing high similarity to manual segmentation. Remaining differences between network output and manual segmentation, indicated by an arrow in Fig 7, typically represent ambiguous cell structure (see Figs. S1 and S2 for additional results). Final global accuracy and class accuracy of the trained network for the test cell are 94.1% and 93.1%, indicating that identical MS-D networks can be trained for different problems. Results for two other challenging problems are given in Figs. S3 and S4.

Denoising large tomographic images

Finally, we use the above architecture, only changing the nonlinear function of the final layer from the soft-max function to the identity, and train on the different task of denoising tomographic reconstructions of a fiber-reinforced mini-composite. 2160 images of 2560^2 pixels were reconstructed using 1024 acquired X-ray projections to obtain images with relatively low amounts of noise (Fig. 8a). Noisy images of the same object were obtained by reconstructing using 128 projections (Fig. 8b). The input is a noisy image, with corresponding noiseless image used as target output during training. From the sample top, 500 images were used for training, and 100 images were used for validation. Fig. 8c shows output for a tested image near the sample bottom, computed in 2.05 seconds using a GTX 1080 GPU (see Fig. S5 for additional timings). The MS-D network accurately denoises highly noisy images by learning image features from the training set, and identical MS-D networks can be easily applied to different problems with minimal changes.

Conclusions

We have presented a deep convolutional "Mixed-Scale Dense (MS-D)" network architecture for image processing problems, using dilated convolutions instead of traditional scaling operations to learn features at different scales, employing multiple scales in each layer, and computing the feature map of each layer using all feature maps of earlier layers, resulting in a densely connected network. By combining dilated convolutions and dense connections, the MS-D network architecture can achieve accurate results with significantly fewer feature maps and trainable parameters than existing architectures, enabling accurate training with relatively small training sets. MS-D networks are able to automatically adapt by learning which combination of dilations to use, allowing

identical MS-D networks to be applied to a wide range of different problems.

ACKNOWLEDGMENTS

Supporting contributions were performed by A. Ekman, C. Larabell [supported by the National Institutes of Health-National Institute for Drug Abuse (NIH-NIDA) Grant U01DA040582], S. Mo, O. Jain, D. Parkinson, A. MacDowell, and D. Ushizima [Center for Advanced Mathematics for Energy Research Applications (CAMERA)]. Computer calculations were performed at the Lawrence Berkeley National Laboratory under Contract DE0AC02-5CH11231. Tomographic images of fiber-reinforced minicomposite were provided by N. Larson and collected at the US Department of Energy's (DOE) Advanced Light Source (ALS) Beamline 8.3.2. This work was supported by CAMERA, jointly funded by The Office of Advanced Scientific Research (ASCR) and the Office of Basic Energy Sciences (BES) within the DOE's Office of Science. Soft X-ray tomography data were collected and segmented at the National Center for X-ray Tomography, supported by the National Institutes of Health-National Institute of General Medical Sciences (NIH-NIGMS) Grant P41GM103445 and the Department of Energy, Office of Biological and Environmental Research, Grant DE0AC02-5CH11231.

REFERENCES

[1] P. Agrawal, R. Girschick, and J. Malik, "Analyzing the performance of multilayer neural networks for object recognition," in *ECCV 2014, Part VII, LNCS*, 2014, pp. 329–344.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[4] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *arXiv:1511.00561v3*, 2016.

[5] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[6] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 4, pp. 640–651, 2017.

[7] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th Int. Conf. on machine learning (ICML-10)*, 2010, pp. 807–814.

[8] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *J. Mach. Learning Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. of 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI). Georgia, USA*, 2016.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv:1408.5093*, 2014.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[14] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in neural information processing systems*, 2015, pp. 2377–2385.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[16] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *arXiv preprint arXiv:1611.07004*, 2016.

[17] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[19] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *arXiv:1605.07648*, 2016.

[20] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[21] Y. Lee, T. Hara, H. Fujita, S. Itoh, and T. Ishigaki, "Automated detection of pulmonary nodules in helical ct images based on an improved template-matching technique," *IEEE Transactions on medical imaging*, vol. 20, no. 7, pp. 595–604, 2001.

[22] G. Gerig, M. Jomier, and M. Chakos, "Valmet: A new validation tool for assessing and improving 3d object segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2001*. Springer, 2001, pp. 516–523.

[23] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *International Conference on Learning Representations (ICLR)*, 2016.

[24] T.-W. Ke, M. Maire, and S. X. Yu, "Multigrid neural architectures," in *The IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[25] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[26] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation," *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.

[27] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv:1410.0759*, 2014.

[28] J. Akeret, C. Chang, A. Lucchi, and A. Refregier, "Radio frequency interference mitigation using deep convolutional neural networks," *Astronomy and Computing*, vol. 18, pp. 35–39, 2017.

[29] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[30] K. Jarrett, K. Kavukcuoglu, Y. LeCun *et al.*, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th Int. Conf. on Computer Vision*. IEEE, 2009, pp. 2146–2153.

[31] L. Ladickỳ, P. Sturgess, K. Alahari, C. Russell, and P. Torr, "What, Where and How Many? Combining object detectors and CRFs," *Computer vision–ECCV 2010*, pp. 424–437, 2010.

[32] J. Tighe and S. Lazebnik, "Superparsing," *Int. Jour. Computer Vision*, vol. 101, no. 2, pp. 329–349, 2013.

Supplementary Information

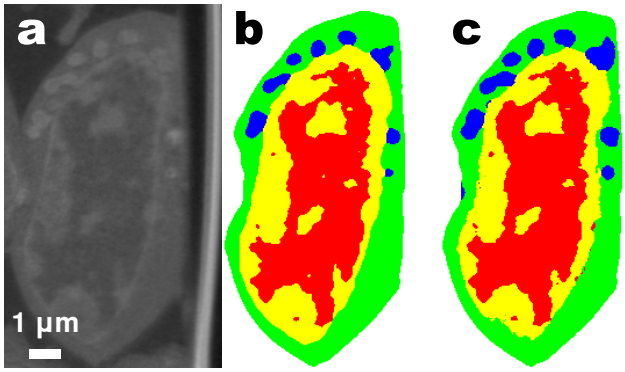

Fig. S1. A tomographic slice of a lympoblastoid cell (a), with the corresponding manual segmentation (b) and output of an MS-D network with 100 layers (c).

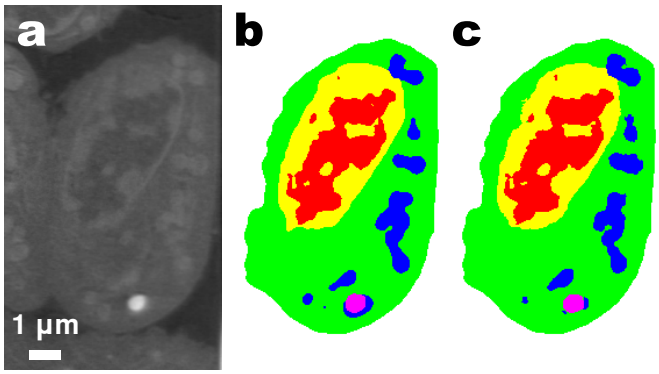

Fig. S2. A tomographic slice of a lympoblastoid cell (a), with the corresponding manual segmentation (b) and output of an MS-D network with 100 layers (c).

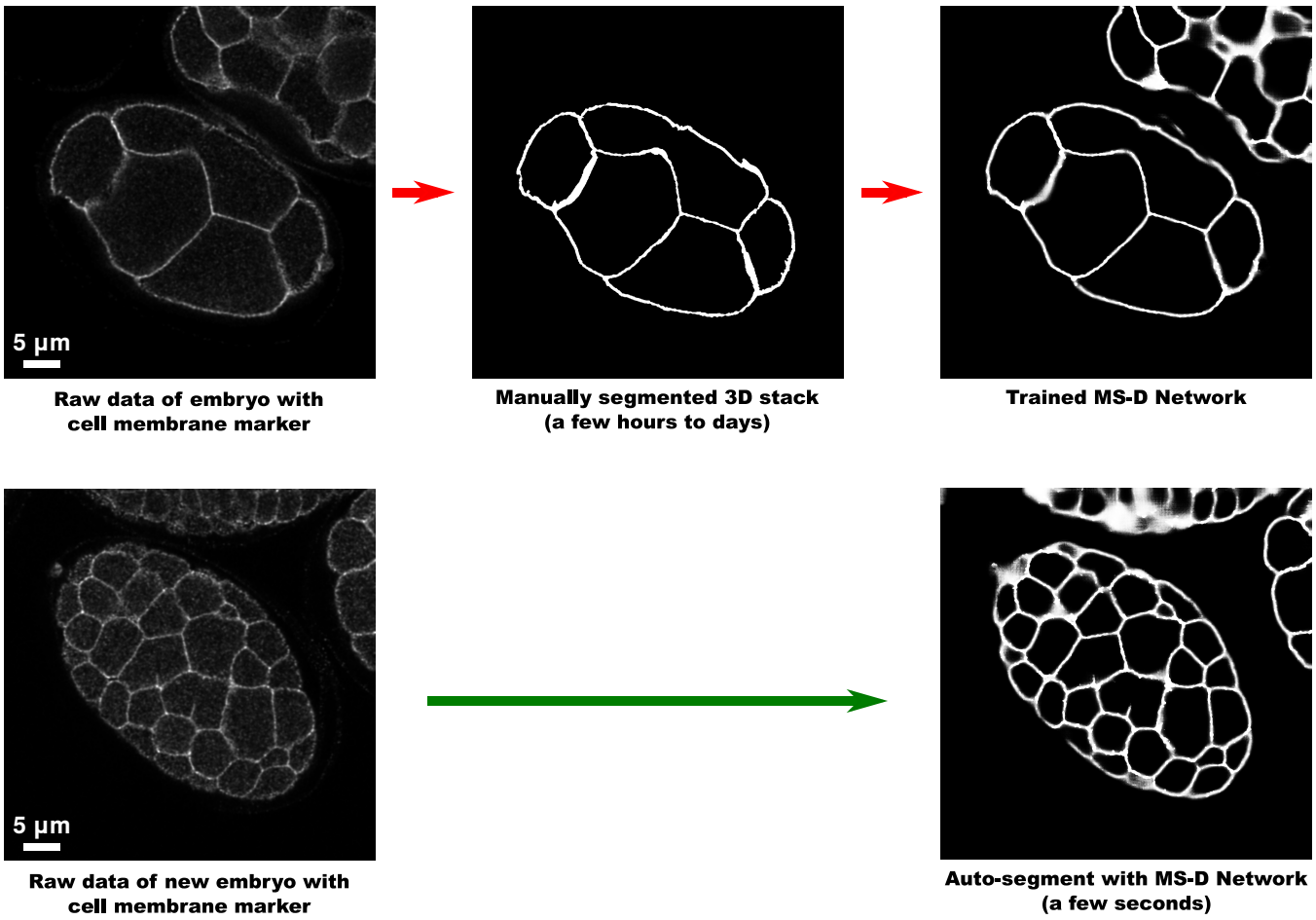

Fig. S3. Example of how an MS-D network can improve the analysis of C. elegans worm embryos by eliminating time-consuming manual segmentation of the cell membranes. Here, we used an MS-D network with 100 layers and trained using a single manually segmented stack of 190 images from a single C. elegans embryo that was engineered to express a GPF-tagged cell membrane protein. (Top Row:) To generate the training material for the machine learning neural network, a stack of 190 images from a 7-cell embryo was first thresholded. The speckles were then removed from each image, and the gaps in the membranes were filled in manually to reflect the actual cell membrane structure. These 190 manually segmented images were then trained to generate an algorithm for labeling cell boundaries. (Bottom Row:) The algorithm generated by machine learning was then directly applied to the raw images of an 86-cell C. elegans embryo to label boundaries for all the cells within the embryo. Attribution for embryo contribution: Uzawa, S., Bian, Q., Meyer, B.J., Howard Hughes Medical Institute and Department of Molecular and Cell Biology at U.C. Berkeley

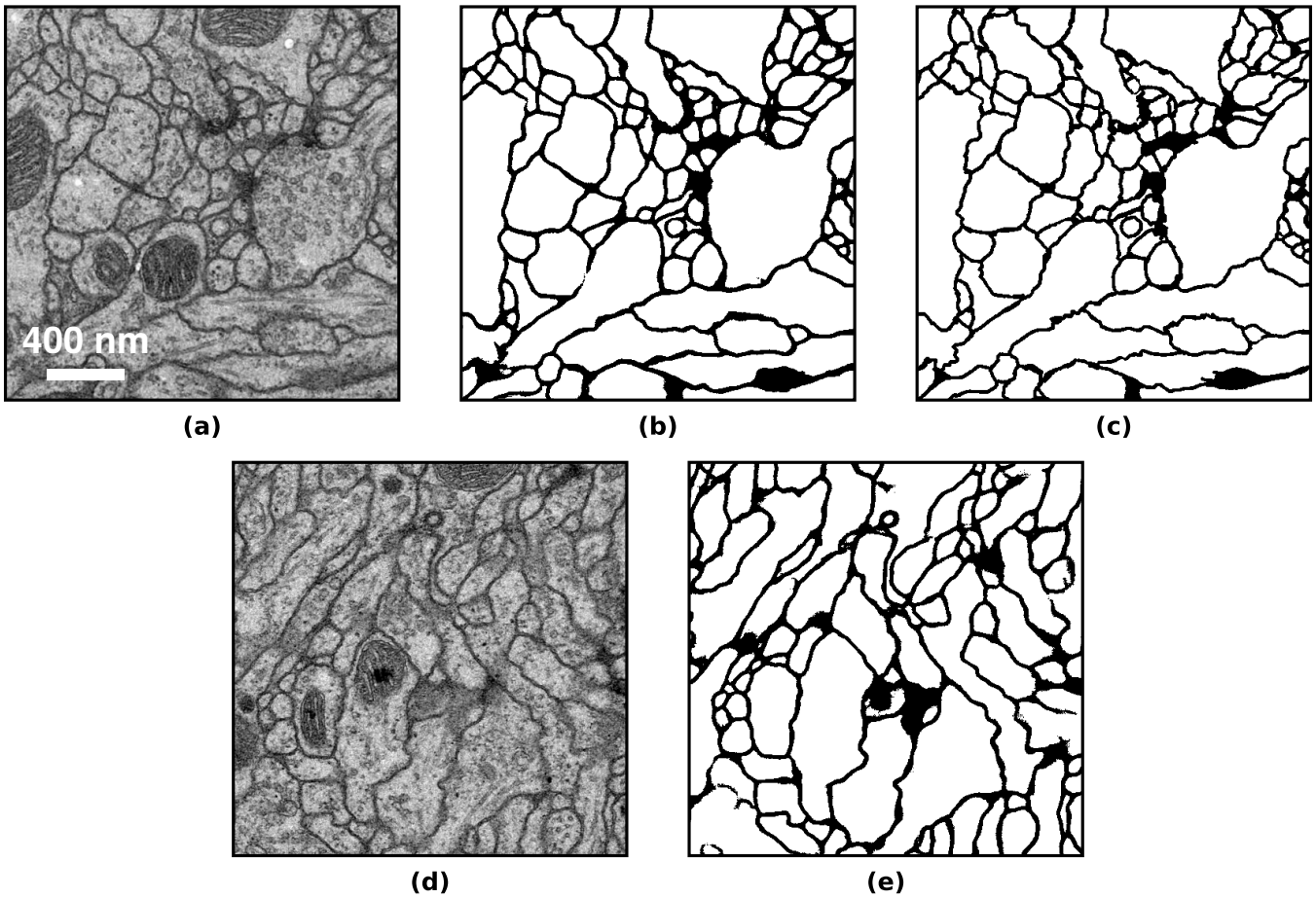

Fig. S4. Example of applying an MS-D network with $w = 1$ and $d = 100$ to the "ISBI Challenge: Segmentation of neuronal structures in EM stacks". Input images consist of 512×512 pixel serial section Transmission Electron Microscopy (ssTEM) images of the Drosophila first instar larva ventral nerve cord (VNC). The goal is to segment neural structures in each image. The challenge dataset consists of 30 training images for which a manual labeling is provided, and 30 testing images for which the manual labeling is withheld. For training the MS-D network, the 30 available images were augmented by rotation, reflection, and elastic deformation, and training was stopped after no improvement was observed in the global accuracy for the original, not augmented, images. In (a), an input image from the training set is shown, with the corresponding output of the MS-D network (b), and manual labeling (c). An input image from the test set is shown in (d), with the corresponding MS-D network output shown in (e). For more information about the challenge, see: Arganda-Carreras I, et al. (2015) Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in neuroanatomy* 9:142.

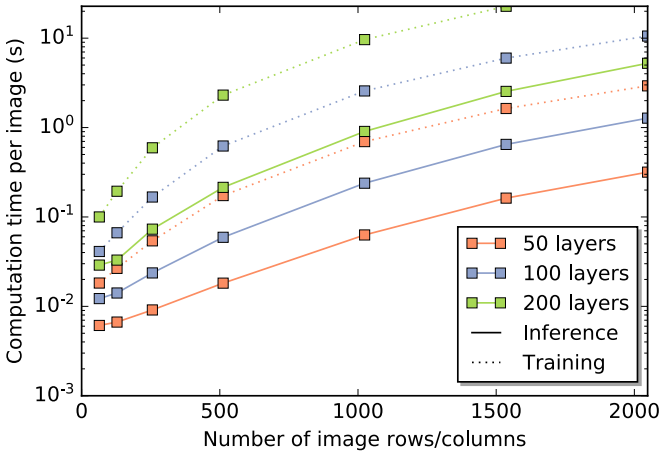

Fig. S5. The computation time of processing a single image with an MS-D network as a function of the number of rows and columns, for the application shown in Fig. 8. Results are shown for both inference (i.e. a forward pass) and training (i.e. backpropagation and gradient computation), for various numbers of layers. Computation times were measured by taking the average time of 500 computations using a single GTX 1080 GPU and a batch size of one. Note that, in our implementation, the required computation time for larger batches scales linearly with the number of images.