Incremental Convolutional Neural Network Training

Yuan Liu, Yanlin Qian, Ke Chen, Joni-Kristian Kämäräinen, Heikki Huttunen
Department of Signal Processing
Tampere University of Technology
Tampere 33720, Finland

Lixin Fan, Jukka Saarinen
Nokia Research Center
Tampere, Finland

***Abstract*—Experimenting novel ideas on deep convolutional neural networks (DCNNs) with big datasets is hampered by the fact that network training requires huge computational resources in the terms of CPU and GPU power and hours. One option is to downscale the problem, e.g., less classes and less samples, but this is undesirable with DCNNs whose performance is largely data-dependent. In this work, we take an alternative route and downscale the networks and input images. For example, the ImageNet problem of 1,000 classes and 1,2M training images can be solved in hours on a commodity laptop without GPU by downscaling images and the network to the resolution of 8×8. We attempt to provide the solution to transfer the knowledge (parameters) of a trained DCNN with lower resolution to improve the efficiency of training a DCNN with higher resolution, and continue training incrementally until the full resolution is achieved. In our experiments, this approach achieves clear boost in computing time without the loss of performance.**

I. Introduction

Deep Convolutional Neural Network (DCNN) based methods have become the dominant approach with state-of-the-art results in many pattern recognition applications, for example, speech recognition [1], [2], visual class detection [3], [4], [5], face recognition [6], [7], [8], and video categorization [9], [10]. In visual class detection, the seminal work of Krizhevsky *et al.* [3] has been significantly improved by ever deeper networks extending the original eight layers to 19 of VGGNet [11], 22 layers of GoogLeNet [12] and recently 152 layers of the network by He *et al.* [13]; with experiments using over 1200-layer network.

The current trend is thus to increase network complexity by adding more layers, which are almost exclusively convolutional [12], [13]. The most significant bottleneck in increasing the model complexity is the required computational time: Training the state-of-the-art 100+ layer network may take weeks on a high-end GPU. Although the problem may partially be circumvented by running several network candidates in parallel, the design is nevertheless somewhat iterative sequential process and slow training renders quick testing of novel ideas in the spirit of agile programming practically impossible.

The straightforward solution to decrease the DCNN training times is to use a smaller number of training samples (images) and a smaller number of classes, but still low level GPUs may not have enough memory for convolutions, performance may start to suffer from over-fitting and the results may seem overly optimistic for the original large scale problem. Motivated by the aforementioned observation, we propose an alternative solution where the original problem is "tinyfied" by downscaling the input images, decreasing network layers and downscaling the network filters and layers. As the starting point, we adopt the original network structure of Krizhevsky *et al.* [3] and downscale it for smaller size images. We then propose incremental learning from 8×8 images step by step to the full scale (Fig. 1). With the popular and large ImageNet [14] and Places205 [15] datasets for image classification, we demonstrate that our incremental procedure can achieve the original performance with significantly boosted training times. Interestingly, the smallest networks can be trained within hours on commodity laptop hardware without GPU and therefore novel ideas can be conveniently tested and incrementally scaled upwards.

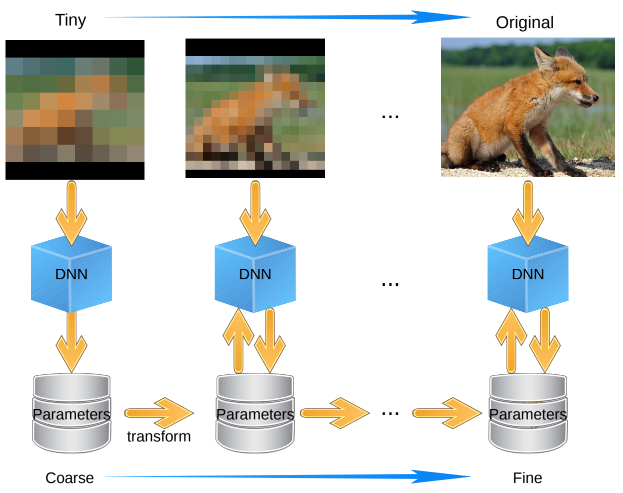

Fig. 1. Workflow of our incremental Deep Convolutional Neural Network (DCNN) learning.

II. Related Work

Since the seminal work of Krizhevsky *et al.* [3] for large scale image classification, their results have readily been superseded by introducing deeper networks [11], [12], [13]. He *et al.* [13] also propose references to the layer inputs and shortcuts that provide lower complexity than original networks of the same depth. Girshick *et al.* [16], [17] introduce the R-CNN approach where detection is based on region proposals to reduce the size of images fed into deep model for recognition, but they use the original DCNN training. Improved performance can also be achieved by replacing the last decision layer with the margin maximization target function [18] or using random forests on the top of deep features [19].

Ba and Caruna [20] propose a model compression approach that is used to "compress" a full DCNN model to a traditional shallow network. Sindhwani *et al.* [21] introduce special transforms that make DCNNs computationally lighter. More recently, *network transformations* between small and large networks have been studied, including the work of Chen *et al.* [22], which proposes two strategies to transfer the learned knowledge between light and heavy network structures, termed `Net2WiderNet` and `Net2DeeperNet`. The former strategy transforms any network into an equivalent *wider* network (with more nodes at each layer) and the latter transforms into an equivalent *deeper* net by adding new layers to the existing topology. Although the study proposes a method for extending both convolutional and deep layers, it only considers adding identity mappings at the same resolution. Here, we extend the approach by also considering transformations across different resolution: *How to increase the resolution of both the input and each of the feature maps of the convolutional layers?*

Contributions – To the authors' best knowledge our work is for the first time to specifically address the problem of incremental coarse-to-fine training of deep convolutional neural networks (`Coarse2FineNet`). This can be achieved by

- a method to upscale network's convolution filters for higher resolution images; and
- a method to add and initialize new convolution filters and neurons.

With the help of the above we experimentally verify that DCNN training can be performed incrementally from small images to full scale images and full scale network without loss of accuracy and with faster training time.

III. Incremental Training of DCNN

The incremental DCNN training basically boils down to the two fundamental research questions: 1) how can the network's convolution filters be up-scaled while preserving its input-output mapping? 2) how can convolution filters and network layers be added while preserving its input-output mapping? More specifically, we wish to initialize the parameters $\boldsymbol{\theta}'$ of a large (fine resolution) student network $g(\cdot)$ using the trained parameters $\boldsymbol{\theta}$ of a small (coarse resolution) teacher network $f(\cdot)$ such that

$$\forall \mathbf{x} : g(\mathbf{x}; \boldsymbol{\theta}') = f(\mathbf{x}; \boldsymbol{\theta}). \tag{1}$$

It turns out that if we can upscale the convolution filters and add new filters, then we can also add new layers as the problem is essentially the same. For simplification, we first illustrate the proposed method with inputs (images) of the size 8×8 (CNN$_8$) that are upscaled to the size 16×16 input (CNN$_{16}$).

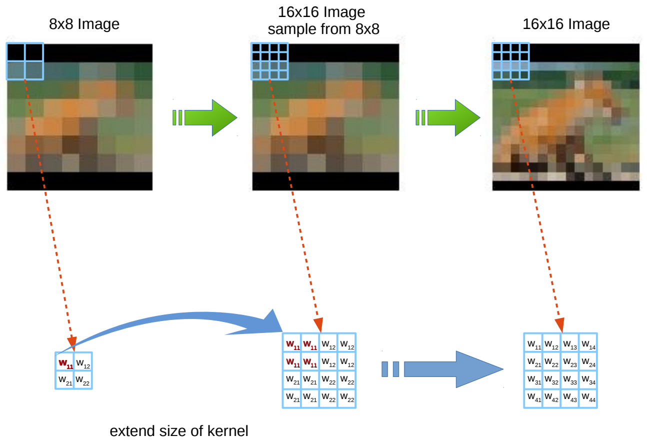

Fig. 2. Convolution filter upsampling.

A. Convolution layer filter upscaling

The feature maps of a convolutional layer are computed by convolving the previous layer with a convolution kernel, adding a bias, and mapping through a nonlinear function. Denote the k^{th} output feature map as $\boldsymbol{Y}^k$, the convolutional filter by its weights $\boldsymbol{W}^k$ and the bias by b^k. Then the k^{th} feature map at location (i, j) is given by

$$\boldsymbol{Y}^k(i,j) = y_{ij}^k = \varphi((\boldsymbol{W}^k \mathbf{x})_{ij} - b^k) \tag{2}$$

where φ is the nonlinearity, such as the ReLU or sigmoid function. For simplification, only the k^{th} output feature map is discussed next and the label k is omitted, and a numeric label is added to distinguish the different size of images.

Firstly, in the convolutional layer of CNN$_8$, a 8×8 matrix $\boldsymbol{X}^{(8)}$ is the input, a matrix $\boldsymbol{Y}^{(8)}$ is the output feature map, a matrix $\boldsymbol{W}^{(8)}$ of 2×2 and $b^{(8)}$ are the weights and the bias of the filter. Based on Equation (2), the output can be written as

$$\begin{aligned} y_{11}^{(8)} &= \varphi((\boldsymbol{W}^{(8)}\mathbf{x})_{11} - b^{(8)}) \\ &\vdots \\ y_{88}^{(8)} &= \varphi((\boldsymbol{W}^{(8)}\mathbf{x})_{88} - b^{(8)}) \end{aligned} \tag{3}$$

In a similar manner, we can write the output of the convolutional layer of CNN$_{16}$, as

$$\begin{aligned} y_{11}^{(16)} &= \varphi((\boldsymbol{W}^{(16)}\mathbf{x})_{11} - b^{(16)}) \\ &\vdots \\ y_{16,16}^{(16)} &= \varphi((\boldsymbol{W}^{(16)}\mathbf{x})_{16,16} - b^{(16)}) \end{aligned} \tag{4}$$

From a pre-trained model CNN$_8$, the parameters $\boldsymbol{W}^{(8)}$ and $b^{(8)}$ are extracted and tentatively used in $\boldsymbol{W}^{(16)}$ and $b^{(16)}$, while enforcing the outputs of CNN$_8$ and CNN$_{16}$ be identical. Note that, in our experiments, the dimension of $\boldsymbol{W}^{(16)}$ is twice that of $\boldsymbol{W}^{(8)}$.

Transformation of the bias term — The transformation of the bias from coarse CNN$_8$ to the fine CNN$_{16}$ is straightforward copy operation: $b^{(16)} = b^{(8)}$. This is obvious since they have the same effect to the output, which is independent of the spatial domain.

Transformation of the convolution kernel — For transforming the weight matrices, $\boldsymbol{W}^{(8)} \mapsto \boldsymbol{W}^{(16)}$, we have to consider their different size and define how to extend from $\mathbb{R}^{8\times8} \mapsto \mathbb{R}^{16\times16}$. To this aim, consider the sample images shown in Fig. 2. In this case, the images are upsampled by replicating each pixel, which copies each pixel of the 8×8 image $\boldsymbol{I}^{(8)}$ on the left into four equally valued pixels in the center image $\boldsymbol{I}^{(16s)}$ in Fig. 2. Thus, the relationship between $\boldsymbol{I}^{(8)}$ and $\boldsymbol{I}^{(16s)}$ can be written as

$$\left.\begin{array}{l} \boldsymbol{I}^{(16s)}_{2j-1,2k-1} \\ \boldsymbol{I}^{(16s)}_{2j,2k-1} \\ \boldsymbol{I}^{(16s)}_{2j-1,2k} \\ \boldsymbol{I}^{(16s)}_{2j,2k} \end{array}\right\} := \boldsymbol{I}^{(8)}_{j,k} \qquad (5)$$

with $j,k \in \{1,2,\ldots,8\}$. Based on Equation (2), the output $\boldsymbol{Y}^{(16)}$ of $\boldsymbol{I}^{(16s)}$ can be similarly written as

$$\begin{aligned} y^{(16)}_{11} &= \varphi((\boldsymbol{W}^{(16s)}\mathbf{x})_{11} - b^{(16s)}) \\ &\vdots \\ y^{(16)}_{16,16} &= \varphi((\boldsymbol{W}^{(16s)}\mathbf{x})_{16,16} - b^{(16s)}) \end{aligned} \qquad (6)$$

To achieve exactly the same (although upsampled) output for both $\boldsymbol{Y}^{(8)}$ and $\boldsymbol{Y}^{(16s)}$, the weights are copied as

$$\left.\begin{array}{l} w^{(16s)}_{2j-1,2k-1} \\ w^{(16s)}_{2j,2k-1} \\ w^{(16s)}_{2j-1,2k} \\ w^{(16s)}_{2j,2k} \end{array}\right\} := \frac{1}{4} w^{(8)}_{j,k} \qquad (7)$$

for $j,k \in \{1,2\}$. The reason for the scale factor $1/4$ in Equation (7) is that one pixel in $\boldsymbol{I}^{(8)}$ is repeated four times in $\boldsymbol{I}^{(16s)}$. As the result of the upsampled convolution, the following identity holds:

$$y^{(16s)}_{2j-1,2k-1} = y^{(8)}_{j,k}, \qquad (8)$$

for $j,k \in \{1,\ldots,8\}$. Moreover, the even-indexed values $y^{(16s)}_{2j,2k}$ can be ignored in the next layer.

B. Inner Product Layer Upscaling

The fully connected layers treat inputs as a simple vectors and take all neurons to produce an output. Again for simplicity, we describe the mapping between $\boldsymbol{Y}^{(8)}$ and $\boldsymbol{Y}^{(16s)}$, where the input is the output of the previous convolution layer.

Firstly, each inner product layer of CNN$_8$ produces the output as a vector $\mathbf{v}^{(8)}$, and computes the matrix product with the weights $\boldsymbol{W}^{(8)}$, as

$$\mathbf{y}^{(8)} = \boldsymbol{W}^{(8)}\mathbf{v}^{(8)} - \mathbf{b}^{(8)} \qquad (9)$$

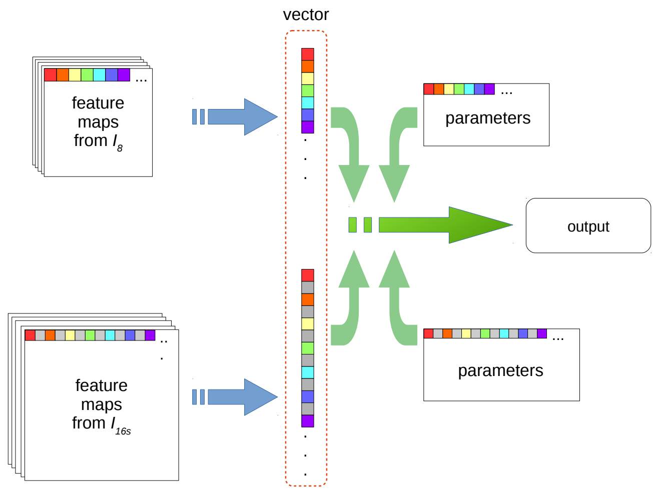

Fig. 3. Adding convolution filters during training.

where $\boldsymbol{W}^{(8)}$ and $\mathbf{b}^{(8)}$ denote the weight and the bias in the inner product layer, respectively. The same inner product in CNN$_{16}$ is

$$\mathbf{y}^{(16)} = \boldsymbol{W}^{(16)}\mathbf{v}^{(16)} - \mathbf{b}^{(16)} \qquad (10)$$

The parameters of $\boldsymbol{W}^{(8)}$ and $\mathbf{b}^{(8)}$ of the pre-trained model can be extracted and transformed into $\boldsymbol{W}^{(16)}$ and $\mathbf{b}^{(16)}$. For the bias, the solution is the same as before, direct copy, but $\boldsymbol{W}^{(8)}$ and $\boldsymbol{W}^{(16)}$ do not have the same dimensions, which causes the problem on the transformation between them. We solve this problem in a similar manner as in Equation (8) for the convolutional layers. More specifically, the relationship between $\boldsymbol{Y}^{(8)}$ and $\boldsymbol{Y}^{(16)}$ of convolutional layer is shown in Fig. 3 where the gray weights denote new parameters that did not exist in the smaller network. In the larger network they are randomly initialized to small gaussian noise and otherwise the weights are copied from the smaller network. In other words, we can add gaussian noise between two original values of $\boldsymbol{W}^{(8)}$ in the both horizontal and vertical directions to generate $\boldsymbol{W}^{(16)}$.

C. Adding New Feature Maps

To improve the generalisation capability, adding new feature maps produced by additional convolution kernels is different from the above since they represent elements that did not exist at all on the lower level networks. In general, the main problem with additional filter elements is how to set their initial values and we introduce three methods which are experimentally tested.

Random Gaussian Noise – The original approach in the Caffe implementation of [3] can be utilized. The random Gaussian values are used for the weights and constants are set to the bias. We experiment with different magnitude of the Gaussian noise to test this initialization approach.

Flipping Existing Filters After training a DCNN model, the convolutional filters can be learned and visualized. In the paper

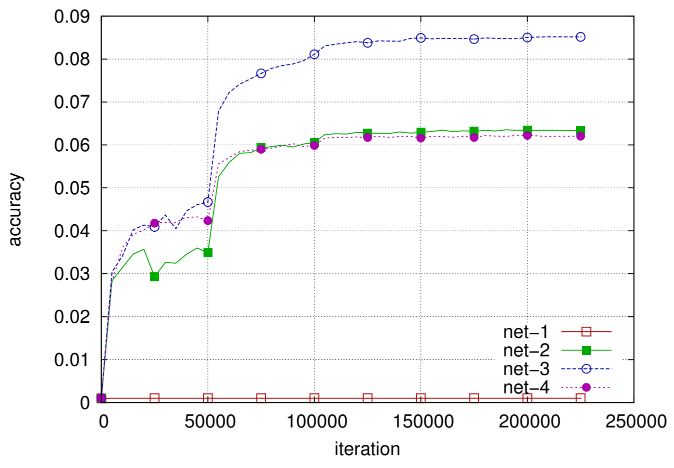

Fig. 4. Results of training and testing with 8×8 downscaled ImageNet images and the small networks in Table I.

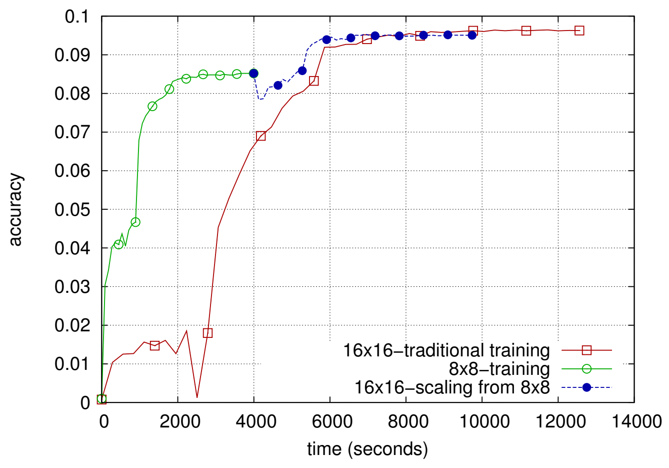

Fig. 5. Incremental training vs. conventional training using upscaled convolution filters (ImageNet dataset).

of Krizhevsky et al [3], there are 96 convolutional filters of size $11\times11\times3$ learned by the first convolutional layer (conv1) that show a variety of frequency- and orientation-selective filters, as well as various coloured blobs. Even though the number of filters implemented in our experiments is less, they still can learn parts of them. Thus, we attempt to find a way to initialize the new added filters as supplementary filters. Theoretically, if the supplementary filters are necessary in the network, they can boost training.

IV. Experiments

A. Datasets

All our experiments are performed with the ImageNet Large Scale Visual Recognition Challenge 2012 dataset [23] of 1,000 classes, more than 1.2 million training images and 50,000 validation images (50 per each class) and the Place205 benchmark including 2.5 millions of images to cover 205 scene classes [24]. According to the standard practice we train with the training set and report performance for the validation set.

B. Implementation details

In Table I are listed the network structures that we used in our experiments. They are scaled versions of the original network by Krizhevsky *et al.* [3]. The accuracy of the different networks for 8×8 downscaled ImageNet images is shown in Fig. 4. For all experiments we report the wall time and all experiments were run on a single thread.

C. Experiment 1: Upscaling the First Convolution Layer

The first experiment was performed using $net-2$ in Table I where the network was otherwise fixed, but the network was first trained with 8×8 images and then the input filter size doubled and the weights mapped to a larger network. For comparison, the same network was trained with the same initialization for 16×16 images from scratch. The results are in Fig. 5 where it is obvious that smaller network achieves better accuracy much faster and the re-mapping maintains almost the same accuracy. The small down dip in the accuracy can be explained by the fact that the higher resolution images contain details that are not visible in the smaller images. The dip does not occur if the 16×16 images are generated from 8×8 images but the performance numbers are exactly the same.

The results is even more dominant if we add more upscaling steps. The steps $8 \times 8 \rightarrow 16 \times 16 \rightarrow 32 \times 32$ where applied for ImageNet in Fig. 6 and for Places205 in Fig. 7 where in the both cases the incremental learning achieves higher performance clearly faster than using the 32×32 images from the scratch. However, the dip between re-mapping also becomes more evident therefore leaving an open research question how to alter the network training parameters between the mapping stages. In these experiments we used the Caffe default parameters.

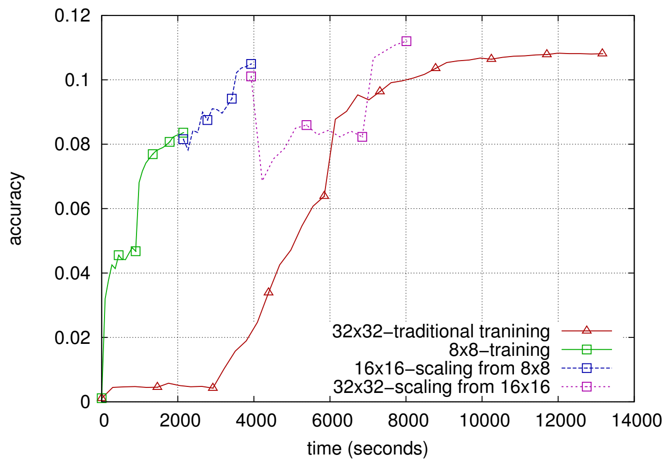

Fig. 6. Incremental training vs. conventional training using upscaled convolution filters - double upscaling (ImageNet dataset).

TABLE I
CAFFE DCNN PARAMETER USED IN OUR EXPERIMENTS.

	Network parameters				
	$L1(conv)$	$L2(conv)$	$L3(conv)$	$L6(full)$	$L8(full)$
orig [3]	$96(11)P3/2$	$256(5)P3/2$	$384(3)P$-/-	4096	1000
2^5-scaled (8×8)	$3(0.3)P0/0$	$8(1)P1/1$	$12(3)P$-/-	128	1000
2^4-scaled (16×16)	$6(0.7)P1/0$	$16(3)P2/1$	$24(3)P$-/-	256	1000
2^3-scaled (32×32)	$12(1.3)P2/1$	$32(4)P3/2$	$48(3)P$-/-	512	1000
net-1	$3(2)P1/1$	-	-	128	1000
net-2	$6(2)P1/1$	-	-	256	1000
net-3	$12(2)P1/1$	-	-	512	1000
net-4	$6(2)P2/1$	$16(3)P3/2$	-	256	1000

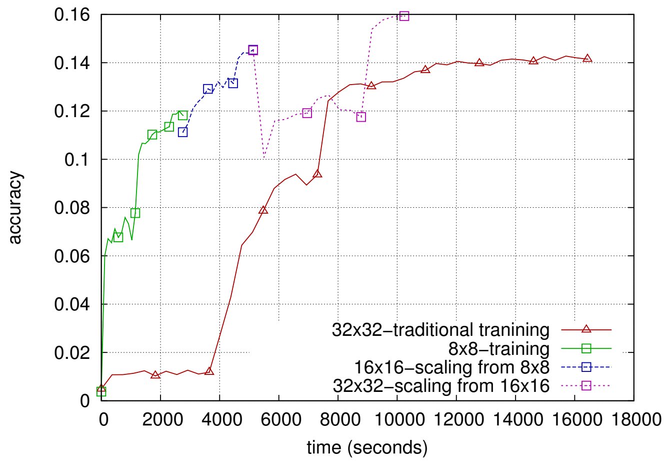

Fig. 7. Incremental training vs. conventional training using upscaled convolution filters - double upscaling (Places205 dataset).

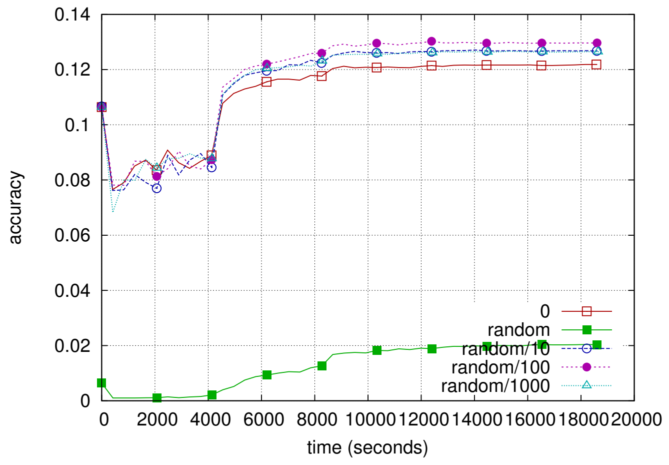

Fig. 8. Adding and initializing new convolution filters: initialization to 0 and Gaussian noise $N(1, 0)$ divided by 1, 10, 100 and 1000 (ImageNet dataset).

D. Experiment 2: Adding and Initializing New Convolution Filters

In this work we experimented between the mapping from $net-2$ to $net-3$ where new filters need to be added. The results for the Gaussian initialization are shown in Fig. 8 and for filter flipping in Fig. 10. All flipping methods provided good initialization, but since small Gaussian noise or initialization to 0 also provided almost the same results we prefer Gaussian noise $N(0, 1/100)$ due to its simplicity.

Finally to verify the both mappings between $net-2$ and $net-3$ we repeated the $8\times8 \rightarrow 16\times16 \rightarrow 32\times32$ experiment and the results in Fig. 10 (cf. Fig. 6) verify that the incremental steps using both scaled and added filters perform well.

E. Experiment 3: Adding Layers to the Full Size

To test our incremental training from the smallest 8×8 to the full size 256×256 images we made small networks for which we incrementally added filters and layers (all layers initialized to have no effect similar to Section III-C). The networks were manually crafted without optimization to start from a three layer network to full eight layer network of Krizhevsky et al. [3]. The results are given in Fig. 11 where it is clear that our icremental learning can achieve the full accuracy ([3] report 59.3 top-1 accuracy, but they further utilize data augmentation that was not used in our work).

V. CONCLUSION

In this work, we investigated the novel approach of training deep convolutional neural networks (DCNNs) incrementally and in coarse-to-fine manner. Our approach enables quick and agile experiments on novel ideas using the full data but its “tinyfied” version. In addition, the learned network knowledge can be tranformed to the larger networks and higher resolution by the proposed upscaling steps (Section III-A and Section III-B). In our experiments, incremental training achieved the same accuracy as full large scale network training, but progress was much faster indicating that coarse structures are the same but can be learned using less feature details. During the course of work the two open research questions were identified: What are the incremental steps in the terms of the DCNN topology? How the training parameters (e.g., the learning rate and momentum) should be adjusted between the

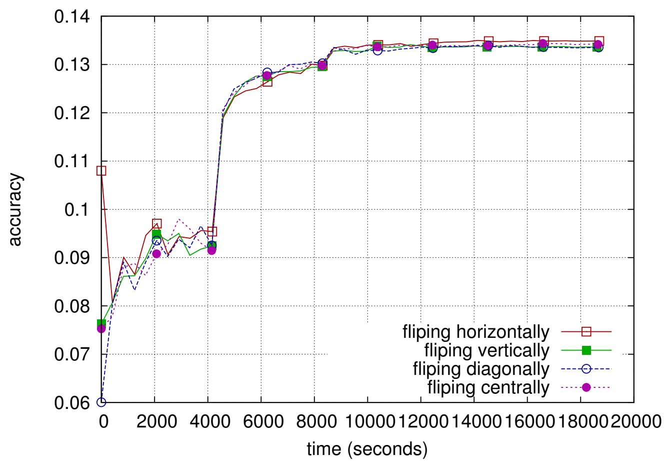

Fig. 9. Results on initialising new added filters as flipping horizontally, vertically, diagonally and centrally on the ImageNet dataset.

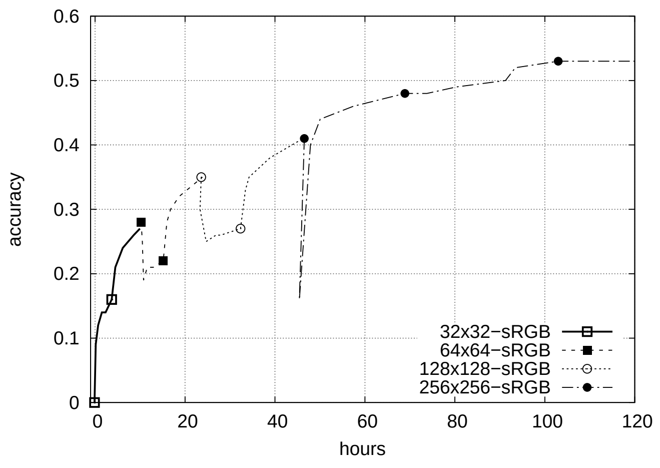

Fig. 11. Results for incrementally training a full-sized 256 × 256 network that achieves the same performance as the original.

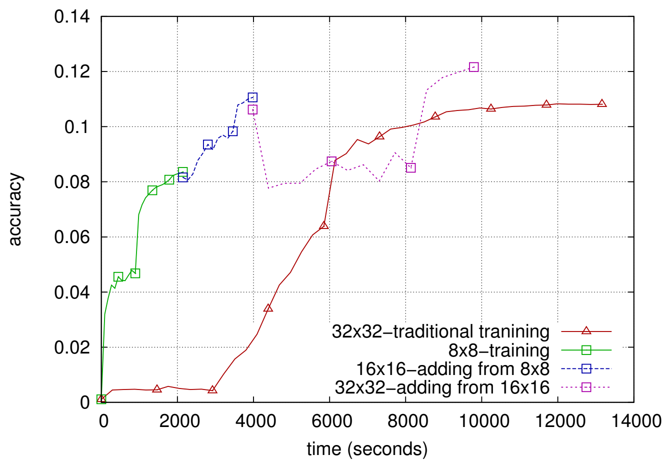

Fig. 10. Results of the traditional method and incremental learning with added and scaled filters for the ImageNet dataset.

incremental steps? Our future work will address these open questions to make incremental DCNN training unsupervised.

Acknowledgment

This work is funded by the Academy of Finland Grant No. 267581 “Learning 10,000 Visual Classes and Some Stuff” and the Finnish Funding Agency for Innovation (Tekes) project Data to Intelligence (D2I).

References

[1] A. Graves and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *CASSP*, 2012.
[2] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-r. Mohamed, G. Dahl, and B. Ramabhadran, “Deep convolutional neural networks for large-scale speech tasks,” *Neural Networks*, 2015.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *NIPS*, 2012.
[4] H. Huttunen, S. Yancheshmeh, and K. Chen, “Car type recognition with deep neural networks,” in *IV*, 2016.
[5] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *TPAMI*, 2015.
[6] Y. Sun, X. Wang, and X. Tang, “Deep learning face representation from predicting 10,000 classes,” in *CVPR*, 2014.
[7] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, “A convolutional neural network cascade for face detection,” in *CVPR*, 2015.
[8] O. M. Parkhi, A. Vedaldi, and A. Zisserman, “Deep face recognition,” in *BMVC*, 2015.
[9] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *CVPR*, 2014.
[10] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, “Beyond short snippets: Deep networks for video classification,” in *CVPR*, 2015.
[11] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *ICLR*, 2015.
[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *CVPR*, 2015.
[13] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016.
[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *CVPR*, 2009, pp. 248–255.
[15] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, “Learning deep features for scene recognition using Places database,” in *NIPS*, 2014, pp. 487–495.
[16] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *CVPR*, 2014.
[17] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *NIPS*, 2015.
[18] Y. Tang, “Deep learning using support vector machines,” *CoRR*, vol. abs/1306.0239, 2013. [Online]. Available: http://arxiv.org/abs/1306.0239
[19] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulo, “Deep neural decision forests,” in *ICCV*, 2015.
[20] L. Ba and R. Caruana, “Do deep nets really need to be deep?” in *NIPS*, 2014.
[21] V. Sindhwani, T. Sainath, and S. Kumar, “Structured transforms for small-footprint deep learning,” in *NIPS*, 2015.
[22] T. Chen, I. J. Goodfellow, and J. Shlens, “Net2net: Accelerating learning via knowledge transfer,” *CoRR*, vol. abs/1511.05641, 2015. [Online]. Available: http://arxiv.org/abs/1511.05641
[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision (IJCV)*, pp. 1–42, April 2014.
[24] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, “Learning deep features for scene recognition using places database,” in *NIPS*, 2014.