

# Model Parallel Training and Transfer Learning for Convolutional Neural Networks via Domain Decomposition

Axel Klawonn<sup>[0000-0003-4765-7387]</sup>, Martin Lanser<sup>[0000-0002-4232-9395]</sup>, and Janine Weber<sup>[0000-0002-6692-2230]</sup>

## 1 Introduction

Convolutional neural networks (CNNs) [9] have been shown to be tremendously successful in processing image data or, more general, data with a grid-like structure. However, with increasing numbers of model parameters and increasing availability of large amounts of training data, parallelization approaches for a time- and memory-efficient training process have become increasingly important; see also [1] for an overview. In general, most parallelization approaches can be categorized into model or data parallel methods [1]. In data parallel approaches, different cores or processors of a parallel machine obtain local copies of the underlying deep learning model which are trained with local subsets of training data points. Usually, the locally trained models are then aggregated once or iteratively after a fixed number of epochs to obtain a final, global model. In model parallel approaches, not the training data but the neural network model itself is distributed to different cores or processors of a CPU or, typically, a GPU. Depending on the decomposition of the network architecture, the total global model then needs to be composed from the locally trained network parameters either once, at the end of the training, or frequently, given that in neural networks, one layer usually needs the output of the previous layer.

Generally speaking, many model parallel training approaches can be interpreted as domain decomposition methods (DDMs) [14]; see [5] for a survey of existing approaches based on the combination of machine learning and DDMs. In [7], a novel model parallel training strategy for CNNs applied to different image classification problems has been presented. This training strategy is based on a decomposition of the input images into smaller subimages and hence, proportionally smaller CNNs op-

---

Axel Klawonn, Martin Lanser, Janine Weber  
Department of Mathematics and Computer Science, University of Cologne, Weyertal 86-90,  
50931 Köln, Germany, <https://www.numerik.uni-koeln.de>  
Center for Data and Simulation Science, University of Cologne, 50923 Köln, Germany,  
<https://www.cds.uni-koeln.de>  
e-mail: {axel.klawonn,martin.lanser,janine.weber}@uni-koeln.de

erating exclusively on the subimages are trained in parallel. In particular, the training of the local CNNs does not require any communication between the different local models. Subsequently, a dense feedforward neural network (DNN) is trained that evaluates the resulting local classification probability distributions into a final global decision. Due to the divide-and-conquer character as well as the implementation of a global coupling between the different local CNN models, the described method can be loosely interpreted as a domain decomposition approach.

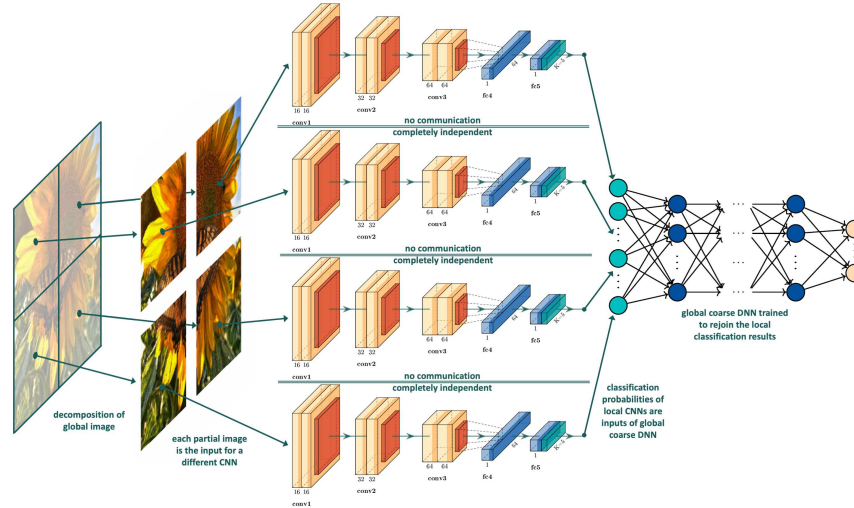
In this paper, we extend our previous work from [7] by several comparisons. First, we provide further comparative results of the CNN-DNN model from [7] with computationally less costly alternatives to combine the local CNN classifications into a final, global decision. Second, we present classification accuracies for training the CNN-DNN model from [7] as one cohesive model architecture. Finally, we additionally consider the idea of transfer learning such that the network parameters of the locally trained CNN models are used as initial values for a subsequently trained global coherent CNN-DNN model.

## 2 Training strategies

In this section, we briefly describe the parallel CNN-DNN model architecture as introduced in [7] as well as its extended variants and modifications for transfer learning which are considered in this paper for the first time. Let us note that the idea of decomposing a CNN into smaller subnetworks within the context of preconditioning and transfer learning has also been considered in [2]. However, the globally trained CNN model in [2] is different from our globally trained network architecture.

### 2.1 Parallel CNN-DNN model architecture

As presented in [7], we consider a hybrid CNN-DNN neural network architecture which naturally supports a model parallel training strategy. As a starting point, we assume that we have a classic CNN model that takes as input data a two-dimensional pixel image with  $H \times W$  pixels and outputs a probability distribution with respect to  $K \in \mathbb{N}$  classes. In order to define our CNN-DNN model, we now decompose the input data in form of images into a finite number of  $N \in \mathbb{N}$  smaller subimages. Note that for colored input images with 3 channels of  $H \times W$  pixels, we exclusively decompose the images in the first two dimensions, the height and the width, but not in the third dimension. Hence, each image is decomposed into  $N$  subimages with height  $H_i$  and width  $W_i$ ,  $i = 1, \dots, N$ . Then, for each of these subimages, we construct corresponding subnetworks, that is, local CNNs that only operate on certain subimages of all input images. Let us note that, in this paper, due to space limitations, we exclusively consider decompositions of the input images into



**Fig. 1** Visualization of the CNN-DNN network architecture. **Left:** The original image is decomposed into  $N = 4$  nonoverlapping subimages. **Middle:** The  $N = 4$  subimages are used as input data for  $N$  independent, *local* CNNs. **Right:** The probability values of the local CNNs are used as input data for a DNN. The DNN is trained to make a final classification for the decomposed image by weighting the local probability distributions. Figure taken from [7, Fig. 4].

rectangular subimages without overlap. We refer to this type of decomposition as type A decomposition; refer also to [7, Sect. 3.1] for more general and overlapping decompositions of the input images. Analogously, the described decomposition of the input data can also be generalized to three-dimensional data, that is, voxel data, for example, in form of computed tomography (CT) scans. In that case, the considered CNN model uses three-dimensional instead of two-dimensional convolutional and pooling operations. The local CNN models are defined such that they always have the same general structure as the original global CNN but differ in the number of channels of the feature maps, the dimension of the feature maps resulting from smaller subimages, the number of neurons within the fully connected layers, as well as in the number of input nodes. All of the listed layers are proportionally smaller than for the respective global CNN. In particular, each local CNN is trained with input data that correspond to a local part of the original pixel image but has access to all training data points. Consequently, the described approach is a model parallel training method. As output data for each of the local CNNs, which can be trained completely in parallel and independently of each other, we obtain a set of  $N$  local probability distributions with respect to the  $K$  classes, where each of the local probability distributions corresponds to a local decision exclusively based on information extracted from the local subimages.

With the aim of generating a final, global decision in form of a global probability distribution with respect to the  $K$ -class classification problem, we subsequently train a DNN that aggregates the local CNN decisions. More precisely, the DNN uses as

input data a vector containing the  $K * N$  local probability values of all  $N$  local CNNs. The DNN model is then trained to map this input vector to the correct classification labels of the original input images corresponding to the  $K$  classes of the considered image classification problem. Let us note that, as observed in detail in [7, Table 3-10], the local CNNs usually perform worse than one large global CNN due to the reduced image size of their respective training data. Hence, it is essentially important that the DNN is trained effectively, such that it is able to accumulate the probability distributions of the local CNNs into the correct global image classification. In Fig. 1, we show an exemplary visualization of the described CNN-DNN model architecture for a global CNN of VGG3 type [12]. The definition and training of the local CNNs is based on the decomposition of the input images into  $N = 4$  subimages and hence,  $N = 4$  local CNNs are trained in parallel for this case. Additionally, a DNN is trained to obtain the final, global classification.

**Comparison with computationally less costly alternatives** Besides evaluating the training time and accuracy values of our presented CNN-DNN model, we additionally compare its performance in terms of classification accuracy with two computationally less expensive methods to combine the local classifications of the local CNNs into a final global classification. As a first alternative, we consider the computation of an average probability distribution among the outputs of the local CNNs and assign each input with the label that shows the highest average probability. In Section 3, we refer to this variant as *average probability* (avg. prob.). Second, we additionally consider a simple majority voting, that is, we assign each image with the label that most of the local CNNs assign their respective subimages to. Let us note that this classification is not necessarily unique since two or more classes may exist which share the majority of the votes. In such cases, we additionally consider the probability values for the respective classes and choose the class among the majority candidates with the highest assigned probability value. In Section 3, we refer to this variant as *majority voting* (maj. vot.).

**Training the CNN-DNN as one model** Even though the main objective in [7] is to provide a network architecture that is well-suited for a model parallel training procedure, additionally, we carefully investigate the classification accuracies of the proposed CNN-DNN model to ensure that the enhanced parallelization is not of the cost of drastically reduced classification performance. Hence, in Section 3, we always compare the accuracy of our CNN-DNN model with a global benchmark CNN which has the same structure and architecture as the local CNNs but with proportionally more parameters and which operates on the entire images as input data. Additionally, for the first time, we also compare the CNN-DNN, where the local CNNs are trained in parallel as described above, with a CNN-DNN that is sequentially trained as one coherent model. That means that we implement the CNN-DNN architecture as shown in Fig. 1 as one model using the functional API of TensorFlow and train it within one sequential training loop. For the remainder of this paper, we refer to this approach as *coherent CNN-DNN* (CNN-DNN-coherent).

## 2.2 Transfer Learning

To provide a broader performance test of our proposed network architecture, we further use the concept of transfer learning for the CNN-DNN trained as one model. In this case, we first train proportionally smaller CNNs operating on separate subimages as described in Section 2.1 for 150 epochs and subsequently use the obtained network parameters as initializations for the respective weights and bias values of the coherent CNN-DNN model. The coherent CNN-DNN model with this initialization is then further trained for additional 50 epochs with respect to the global classification labels of the underlying images. For both, that is, the local learning and the transfer learning, the training is started with a learning rate of 0.001 while the learning rate is adaptively reduced by a factor of 0.5 on plateaus with a patience of 20 epochs. Regarding the loose analogy of the CNN-DNN training approach to DDMs, the concrete implementation of transfer learning based on locally pre-trained smaller networks can also be interpreted as a preconditioning strategy within an iterative solver or optimization method, respectively; see also [2] for a closely related approach for a different global neural network architecture. In the following, we refer to this approach as *CNN-DNN with transfer learning* (CNN-DNN-transfer).

## 3 Experiments

In this section, we present some experiments with respect to the described approaches and compare the classification accuracies for different image recognition problems. All experiments have been carried out on a workstation with 8 NVIDIA Tesla V100 32GB GPUs using the TensorFlow library.

**Network architectures and datasets** To evaluate the performance of the described training strategies from Section 2, we consider two different network architectures and three different image classification datasets. First, we test our approach for a CNN with nine blocks of stacks of convolutional layers and a fixed kernel size of  $3 \times 3$  pixels, in case of two-dimensional image data, or  $3 \times 3 \times 3$  voxels, for three-dimensional image data, respectively. We refer to this network architecture as VGG9 for the remainder of this paper and refer to [12] for more implementational details of this network model. Second, we apply all training strategies to a residual neural network (ResNet) [3] with 20 blocks of convolutional layers where we additionally implement skip connections between each block and its third subsequent block; see also [3] for more technical details. We refer to this network architecture as ResNet20. All networks are trained using the Adam (Adaptive moments) optimizer [4] and the cross-entropy loss function.

We test both network models for the CIFAR-10 data [8], the TF-Flowers dataset [13], and a three-dimensional dataset of chest CT scans [11]. The CIFAR-10 dataset [8] consists of 50 000 training and 10 000 validation images of  $32 \times 32$  pixels which are categorized in  $K = 10$  different classes; see also Fig. 2 (left). Given that



**Fig. 2** **Left:** Exemplary images of the CIFAR-10 dataset [8]. **Middle:** Exemplary images of the TF-Flowers dataset [13]. **Right:** Exemplary slices for one chest CT scan taken from the MosMedData dataset [11].

these images are relatively small, we only decompose the images into  $N = 4$  subimages. The TF-Flowers dataset [13] consists of 3 670 images which we split into 80% training and 20% validation data. All these images have  $180 \times 180$  pixels and are classified into  $K = 5$  different classes of flowers; cf. also Fig. 2 (middle). As the last dataset, we consider the three-dimensional image set of chest CT scans [11] which consists of CT scans with and without signs of COVID-19 related pneumonia, that is, we have  $K = 2$ . For an exemplary visualization of CT slices for one exemplary datapoint, see Fig. 2 (right). Each of these CT scans consists of  $128 \times 128 \times 64$  voxels and hence, here, we train CNN models using three-dimensional filters and pooling layers. For all datasets, the DNN model consists of four hidden layers; see [7] for more details.

**Results** In Table 1, we compare the classification accuracies for the validation and training data for the CNN-DNN approach for a VGG9 model with a majority voting and an average probability distribution to combine the local CNN classifications into a global decision. As we can observe, for all tested datasets, the CNN-DNN approach results in higher validation accuracies than both, the average probability distribution and the majority voting, for all tested decompositions. This shows that it is not a trivial task to combine the local classifications obtained from the local CNNs into a final, global classification and that it seems to be helpful to train a small DNN to make this evaluation automatically for us.

When observing the results in Table 2, two major observations can be made. First, with respect to the CNN-DNN-coherent model, we see that for the VGG9 model, the coherent model trained in one sequential training loop results in lower validation accuracies than the CNN-DNN model for all three considered datasets and all tested decompositions. However, for the ResNet20 model, the quantitative behavior is reversed, that is, the CNN-DNN-coherent networks result in higher classification accuracies with respect to the validation data. A possible explanation for this could be as follows. The CNN-DNN-coherent model which is implemented as one connected model architecture might have a more complex loss function and thus, loss surface than the locally trained smaller CNNs as well as the relatively small DNN. Hence, optimizing the respective parameters of the VGG9 model all at once might be more difficult than optimizing first the parameters of the local CNNs in

**Table 1** Classification accuracies for the validation and training data (in brackets) for the CNN-DNN approach for a VGG9 model and computationally less costly alternatives to combine the classifications of the local CNNs. In particular, we show the obtained accuracy values for an average probability distribution (avg. prob.) and a majority voting (maj. vot.).

Decomp.	avg. prob.	maj. vot.	CNN-DNN
<b>CIFAR-10</b>			
type A	0.6745	0.6237	<b>0.7669</b>
$2 \times 2, \delta = 0$	(0.7081)	(0.6546)	(0.8071)
<b>TF-Flowers</b>			
type A	0.6162	0.5974	<b>0.6938</b>
$2 \times 2, \delta = 0$	(0.6498)	(0.6026)	(0.7552)
type A	0.7565	0.7022	<b>0.8471</b>
$4 \times 4, \delta = 0$	(0.7745)	(0.7238)	(0.8593)
<b>Chest CT scans</b>			
type A	0.8038	0.7761	<b>0.9143</b>
$2 \times 2 \times 1, \delta = 0$	(0.8279)	(0.7997)	(0.9357)
type A	0.8024	0.7453	<b>0.8988</b>
$4 \times 4 \times 2, \delta = 0$	(0.8409)	(0.7999)	(0.9493)

parallel and subsequently, the parameters of the DNN. However, when considering the ResNet20 model, the optimization of the CNN-DNN-coherent model might be easier given that the introduction of skip connections usually results in smoother loss surfaces for deep neural networks and enhanced training properties; see also [3, 10]. This could explain that for the ResNet20 model, the CNN-DNN-coherent shows an improved classification accuracy. A detailed investigation of the resulting loss surfaces and their complexity for the tested models is a potential topic for future research.

Second, when considering the transfer learning strategy, we observe higher classification accuracies for both, the VGG9 network model and the ResNet20 model for all tested datasets compared to the training strategies without transfer learning. Hence, using DDM for means of preconditioning and transfer learning can help to further increase the accuracy of image classification models; cf. also [2]. A detailed investigation of the required training times of the transfer learning strategy for our proposed model architecture is a further topic for future research. First results with regards to the training times of the transfer learning approach and the TF-Flowers dataset are already published in [6, Fig. 4].

## References

1. Ben-Nun, T., Hoefler, T.: Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)* **52**(4), 1–43 (2019)
2. Gu, L., Zhang, W., Liu, J., Cai, X.C.: Decomposition and composition of deep convolutional neural networks and training acceleration via sub-network transfer learning. *Electronic Transactions on Numerical Analysis* **56**, 157–186 (2022)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (2016)

**Table 2** Classification accuracies for the validation and training data (in brackets) for a global CNN benchmark model (VGG9 or ResNet20), the CNN-DNN approach as introduced in [7], the CNN-DNN model trained as one coherent model (CNN-DNN-coherent), and a coherent CNN-DNN model trained with a transfer learning approach (CNN-DNN-transfer).

Decomp.	global CNN	CNN-DNN	CNN-DNN-coherent	CNN-DNN-transfer
<b>CIFAR-10, VGG9</b>				
type A	0.7585	<b>0.7999</b>	0.7515	<b>0.8462</b>
$2 \times 2, \delta = 0$	(0.8487)	(0.8663)	(0.7902)	(0.8889)
<b>CIFAR-10, ResNet20</b>				
type A	0.8622	0.8784	<b>0.8998</b>	<b>0.9117</b>
$2 \times 2, \delta = 0$	(0.9343)	(0.9467)	(0.9558)	(0.9664)
<b>TF-Flowers, VGG9</b>				
type A	0.7887	<b>0.8154</b>	0.7808	<b>0.8378</b>
$2 \times 2, \delta = 0$	(0.9321)	(0.8827)	(0.8667)	(0.8999)
type A	0.7887	<b>0.8589</b>	0.7676	<b>0.8608</b>
$4 \times 4, \delta = 0$	(0.9321)	(0.8872)	(0.7995)	(0.8806)
<b>TF-Flowers, ResNet20</b>				
type A	0.8227	0.8475	<b>0.8776</b>	<b>0.8997</b>
$2 \times 2, \delta = 0$	(0.9178)	(0.9454)	(0.9603)	(0.9702)
type A	0.8227	0.8068	<b>0.8406</b>	<b>0.8654</b>
$4 \times 4, \delta = 0$	(0.9178)	(0.8892)	(0.9002)	(0.9244)
<b>Chest CT scans, VGG9</b>				
type A	0.7667	<b>0.9143</b>	0.8889	<b>0.9304</b>
$2 \times 2 \times 1, \delta = 0$	(0.8214)	(0.9357)	(0.9097)	(0.9577)
type A	0.7667	<b>0.8988</b>	0.8774	<b>0.9025</b>
$4 \times 4 \times 2, \delta = 0$	(0.8214)	(0.9493)	(0.9305)	(0.9488)

4. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
5. Klawonn, A., Lanser, M., Weber, J.: Machine learning and domain decomposition methods—a survey. arXiv preprint arXiv:2312.14050 (2023)
6. Klawonn, A., Lanser, M., Weber, J.: Domain-decomposed image classification algorithms using linear discriminant analysis and convolutional neural networks. arXiv preprint arXiv:2410.23359 (2024)
7. Klawonn, A., Lanser, M., Weber, J.: A domain decomposition-based CNN-DNN architecture for model parallel training applied to image recognition problems. *SIAM Journal on Scientific Computing* **46**(5), C557–C582 (2024)
8. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009). Technical report
9. LeCun, Y.: Generalization and network design strategies. *Connectionism in perspective* **19**, 143–155 (1989)
10. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. *Advances in neural information processing systems* **31** (2018)
11. Morozov, S., Andreychenko, A., Pavlov, N., Vladzmyrskyy, A., Ledikhova, N., Gombolevskiy, V., Blokhin, I., Gelezhe, P., Gonchar, A., Chernina, V.: MosMedData: Chest CT Scans with COVID-19 Related Findings Dataset. medRxiv (2020). DOI 10.1101/2020.05.20.20100362
12. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
13. The TensorFlow Team: Flowers dataset (2019)
14. Toselli, A., Widlund, O.: *Domain Decomposition Methods—Algorithms and Theory*, Springer Series in Computational Mathematics, vol. 34. Springer-Verlag, Berlin (2005)