JMLR: Workshop and Conference Proceedings 44 (2015) 160-172 NIPS 2015

The 1st International Workshop "Feature Extraction: Modern Questions and Challenges"

Deep Clustered Convolutional Kernels

Minyoung Kim MINYOUNG.KIM@US.PANASONIC.COM
Panasonic Silicon Valley Laboratory
10900 N. Tantau Ave, STE 200
Cupertino, CA 95014, USA

Luca Rigazio LUCA.RIGAZIO@US.PANASONIC.COM
Panasonic Silicon Valley Laboratory
10900 N. Tantau Ave, STE 200
Cupertino, CA 95014, USA

Editor: Afshin Rostamizadeh

Abstract

Deep neural networks have recently achieved state of the art performance thanks to new training algorithms for rapid parameter estimation and new regularizations to reduce overfitting. However, in practice the network architecture has to be manually set by domain experts, generally by a costly trial and error procedure, which often accounts for a large portion of the final system performance. We view this as a limitation and propose a novel training algorithm that automatically optimizes network architecture, by progressively increasing model complexity and then eliminating model redundancy by selectively removing parameters at training time. For convolutional neural networks, our method relies on iterative split/merge clustering of convolutional kernels interleaved by stochastic gradient descent. We present a training algorithm and experimental results on three different vision tasks, showing improved performance compared to similarly sized hand-crafted architectures.

Keywords: Deep Learning, Convolutional Neural Networks, Convolutional Kernels

1. Introduction

Recently, deep neural networks (DNNs) have led to significant improvement in several machine learning domains, from speech recognition (Dahl et al., 2012) to computer vision (Krizhevsky et al., 2012; Taigman et al., 2013) and machine translation (Sutskever et al., 2014). DNNs have reached state of the art performance thanks to their theoretically proven modeling and generalization capabilities (Hornik et al., 1989; Hornik, 1991; Kůrková, 1992), and practically driven by improvements in training algorithms for rapid parameter estimation (Martens, 2010; Sutskever et al., 2013), novel regularization methods to reduce overfitting (Srivastava et al., 2014) as well as ever increasing data-sets (Deng et al., 2009) and powerful new computing platforms (Chetlur et al., 2014). However, before parameter estimation (so called training) can begin the DNN's structure (also called model architecture) is usually manually defined by domain experts (Lin et al., 2013), and can often account for a substantial portion of the final system performance (Szegedy et al., 2014). We view this step as a bottleneck in the current deep learning pipeline, one that relies on a trial and error human expert in the loop approach which is, to say the least, rather alchemic in nature. We want to address this basic scalability issue of the deep learning development pipeline with training methods that automatically search for DNN architectures while jointly estimating model parameters.

While structural optimization is a notoriously difficult combinatorial task, successful strategies were adopted in the past for (shallow) models that motivated our approach. For instance, for Hidden Markov Models with Gaussian mixture kernels, split/merge algorithms were used to independently vary model complexity for each HMM state, resulting in improved accuracy for large vocabulary speech recognition (Sankar, 1998). Information theoretic methods, such as the minimum description length criterion, were also applied to the problem of structural optimization (Barron et al., 1998), resulting in improved performance in speech recognition (Shinoda and Watanabe, 2000) and as well as training algorithms for auto-encoders (Hinton and Zemel, 1994). However, to the best of our knowledge, there is little published work on structural optimization in the deep learning community, with the notable exception of work based on empirical evaluation (Bergstra and Bengio, 2012) and random search strategies (Bergstra and Bengio, 2012). Although, recently Bayesian optimization of hyper-parameters have been introduced (Snoek et al., 2012).

While these works are interesting, hyper-parameters are only one aspect of the DNN structure, albeit one which is closely related to the performance of the training algorithm. However, there are several other structural parameters that strongly affect DNN's performance which are usually set by experimental trial and error, such as network depth and for convolutional models the number of convolutional filters and kernel size for each layer. In our work, we aim to optimize model architecture, specifically targeting convolutional neural networks (CNNs), and optimizing complexity for each layer. Therefore, in our approach, the model architecture is not maintained constant during training, instead the model complexity is continuously optimized throughout the training step (parameter estimation by stochastic gradient descent), resulting, we believe, in a more scalable approach to the training of deep neural networks. In Section 2, we describe the general approach we are taking for problem of structure optimization of convolutional neural networks. In Section 2.1, we describe the theoretical foundations of our approach. In Section 3, we discuss data-sets and experimental results and in Section 4 we discuss about limitations and possible future improvements.

2. Deep Clustered Convolutional Kernels

The basic idea for our Deep Clustered Convolutional Kernels (DCCKs) it a convolutional model architecture and associated structural training algorithm. We adopt a split/merge outer-loop to the training process that first increases model capacity to model new factors of variability seen in the data, then estimates new parameters for this larger model by stochastic gradient descent (SGD), and finally reduces model capacity to minimize model-space redundancy. Our approach takes inspiration by previous work in the area of Gaussian kernel HMMs (Sankar, 1998; Rigazio et al., 2000; Bocchieri and Mak, 2001; Lee et al., 2001), and is philosophically based on Occam's razor principle whereby a smaller model with similar performance on a given data-set is likely to have better generalization capabilities to new unseen data.

An alternative view of work may be in the context of recent developments in DNN's compression: (Ba and Caruana, 2014) shows that a (shallow) DNN can approach the performance of a substantially larger DNN when trained to mimic the logit output of the larger model. Similarly, (Hinton et al., 2014) shows that logit-mimic training (referred to as "Dark Knowledge") results in orders of magnitude smaller models, compared to the initial complex ensemble models, yet provides competitive performance when tested on both small tasks (MNIST) as well as large scale industrial tasks (large vocabulary speech recognition). It is important to notice that for both these works the authors acknowledge that, while such smaller high performance models can be obtained by logit mimic training from a more complex model set, thus showing that there is an optimal point in the parameter space with high performance, there is currently no known training procedure to directly achieve such optimal point in the smaller model. In this view finding such an elusive point in parameter space by systematically optimizing DNN's structure to eliminate redundancy and minimizing number of parameters, while at the same time estimating the model parameters under the given loss function. The main contribution of our work is a training methodology to iteratively optimize the number of convolutional kernels while estimating the convolutional filter parameters.

2.1 Training algorithm

Conceptually our training procedure is rather straightforward: starting from an initial network architecture, we first train the model by SGD until performance tops out on a validation set. Next, we increase the model complexity of selected convolutional layers by splitting the convolutional kernels. Splitting has the purpose of creating new plausible convolutional filters given the current set of filters and can be done by applying image pre-processing techniques to the kernels, as well as adding jittering and noise to create enough variation. After splitting, the model is again trained by SGD and possibly split again until performance tops out. At that point model is merged to reduce redundancy in the parameter space and again trained by SGD. Notice that the split/merge procedure can start at any layer but then it has to propagate upwards to change the number of kernels of the connecting layers (fan-out). In our setup, given by input data x, forward propagation f is done by:

$$f(x) = g(Wx + B) \tag{1}$$

where g is ReLU activation function with $g(x) = max(0, x)$, W is the weight parameters of the convolutional layer, and B is the biases, each with the following dimensions:

$$W_l = N_l \times \overbrace{d_l \times k_l \times k_l}^{P} \tag{2}$$

$$B_l = 1 \times 1 \times 1 \times N_l \tag{3}$$

where l is a convolutional layer with $l \in \{1, ..., L\}$, N_l is the number of outputs of l, d_l is number of channels of l, and k_l is size of kernel used for l. We use square convolutional kernels, so kernel dimensions are $k_l \times k_l$. For simplicity, we define sub-dimension of W as

P, shown in Equation 2. For the first convolutional layer we have $d_1 = 3$ for RGB images and $d_1 = 1$ for gray-scale images. In the following convolutional layers, d is the output of the previous convolutional layer thus, P would be the size of the feature vectors. This implies that, when we perform the split/merge steps for level l, we need to update both W_l and B_l as well as W_{l+1}. Biases for the following convolutional layer are independent. An important caveat is that the order of the optimal split/merge operation depends on the specific data-set and the filter parameters. For instance, if the initial filters are sparse it is beneficial to do merge first. Otherwise, it is best to perform split first especially on smaller data-sets when the initial filters are already compact and discriminative.

2.2 Splitting Kernels

With splitting, we want to increase model complexity by creating new convolutional kernels from the set of existing well-trained kernels. Therefore, we create new kernels by selectively choosing from a fixed set of transformations. The possible set of transformations to play with is vast and includes the six isometries of the plane, angular rotation, change in contrast (negative "reversing") and many others. In our experiments, we focus on two transformations that seemed to provide a consistent improvement:

- **Rotation** creates new kernels by rotating existing kernels in random directions.
- **Noise perturbation** creates new kernels by adding Gaussian noise to the existing kernels.

One important aspect we verified in our experiments is that rotating kernels has a lower computational cost at training time than rotating training images to create augmented training set. Moreover, we observed that rotating the filters can help improve robustness for highly tilted objects outlets, which would be otherwise hard to correctly classify (see Figure 1). Adding random Gaussian noise, on the other hand, has the obvious benefit of creating diversity and helping with the SGD, like previously reported by (Srivastava et al., 2014). Regarding the splitting strategy, currently we took the simplest approach and split every kernel by a fixed amount. This is bound to be locally unoptimal, and surely a better splitting strategy that tries to maximize some diversity or discrimination criteria could be devised, instead of indiscriminately splitting every single kernel. However, for the most part, we observe that wasteful parameters created by this simple splitting strategy will be eliminated during the final merging step; therefore, aside from a potential sub-optimality in the CPU/Memory usage, we speculate the final model accuracy might not be very affected by this uniform splitting strategy.

2.3 Merging Kernels

After the splitting step, the model might have too much capacity and thus part of the model might become over-parameterized, possibly resulting in over-fitting and lower generalization power. Therefore, the merging step has the purpose of removing model space redundancies and reducing model size, while maintaining the overall model accuracy. In our algorithm we use k-means clustering to merge kernels since, naturally, k-means cluster distortion under the defined distortion measure (we employ $L2$ norm to compute cluster distortion). We

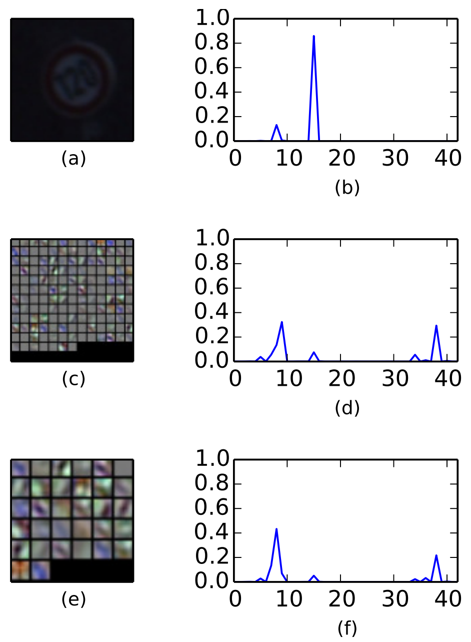

Figure 1: (a) Highly tilted, misclassified test image (b) Soft-max output of original baseline model resulting in miss-classification (c) Baseline model convolutional kernels: notice high proportion of redundant kernels (d) Soft-max at DCCKs intermediate training stage, after split and fine-tuning (e) Final DCCK convolutional kernels, after merge and fine-tuning, showing reduced redundancy (f) Final DCCK soft-max output, correctly classifying the image

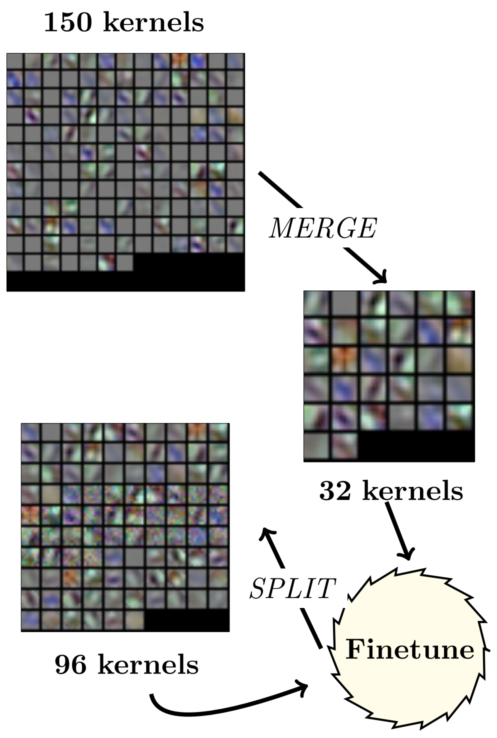

Figure 2: DCCK training example: starting from a large GTSRB model 150 convolutional kernels for the first layer, the algorithm first merges it to 32 kernels. After fine-tuning, kernels are split by adding noise and rotating, then fine-tuned one more time.

empirically observe that k-means clustering to merge filter maps is effective in reducing kernel's redundancy (see filters in Figure 2). Then, we train the network and get weight and bias matrices from each convolutional layer to then choose the filters that are nearest to each centroid. We update W_l and B_l, with W_l' and B_l' using k-means clustering to get centroids C as:

$$C = \arg\min_{P} \sum_{j=1}^{\mathcal{C}} \sum_{p \in P} ||p - \mu_j||^2 \tag{4}$$

$$W_l' = \begin{cases} [P_1', ..., P_i', ..., P_{\mathcal{C}}'], \ \ or \\ [C_1, ..., C_i, ..., C_{\mathcal{C}}] \end{cases} \tag{5}$$

where

$$P_i' = \arg\min_{P'} \ ||P' - C_i \ ||^2 \quad , \quad i = \{1, ..., \mathcal{C}\} \tag{6}$$

and finally,

$$B_l' = \begin{cases} [B_1', ..., B_i', ..., B_{\mathcal{C}}'], \ \ or \\ [\beta_1, ..., \beta_i, ..., \beta_{\mathcal{C}}] \end{cases} \tag{7}$$

where B_i' is P_i''s matched biases matrix, and β_i is

$$\beta_i' = \frac{\sum_i b_n}{\eta_i} \quad , \quad n = \{1, ...N_l\} \tag{8}$$

where η_i is number of p in group C_i.

As shown in Equation 5 and 8, we explored two different methods to update W and B. The first method consists in choosing the Pi that is closer to each centroid C_i. In this case, we use the correspondent bias vector B_i to the corresponding P_i selected. The other way is to use the centroid C_i itself as filter parameters and update B_i with average bias from each cluster. An important detail to choose the right value of k: if we choose k too small then average cluster distortion will be too high to appropriately represent the model parameters, possibly resulting in ineffective features maps. On the other hand, if we choose k too big, not enough kernels will be merged.

3. Experimental results

Our experimental results are based on three different data-sets: MNIST, German Traffic Sign Recognition Benchmark (GTSRB), and CIFAR-10. To make our experiments significant and to validate our approach, we started from hand-tuned model architectures that were as close as possible to the state of the art, in an effort to prove that our split/merge training procedure can still improve model architecture even when starting from a very highly tuned architecture. Our baseline performance on each dataset is 0.82% (MNIST), 2.44% (GTSRB1), 1.24% (GTSRB-3DNN), and 10.4% (CIFAR-10). For all experiments, we used the BVLC Caffe C++ package (Jia et al., 2014). We started our experiments from MNIST since the quick training time allowed to quickly determine reasonable range of hyper-parameters such as the number of centroids k, number of kernels for the split/merge procedure. Next, we move to a more realistic task such as GTSRB for which we started from an initial model, extremely close to the state of the art and finally confirm the portability of our findings on the harder CIFAR-10 data-set. We report the details of each data-set experiments in the following sections.

Table 1: MNIST baseline architecture

Layer	**# of maps**	**Kernel size**
Input	3	
Convolutional	100	5x5
Max Pooling	100	2x2
Convolutional	50	5x5
Max Pooling	50	2x2
Fully connected	100	1x1
Fully connected	10	1x1

Table 2: GTSRB-3DNN architecture

Layer	**# of maps**	**Kernel size**
Input	3	
Convolutional	150	3x3, 3x3, 3x3
Max Pooling	150	2x2, 2x2, 2x2
Convolutional	150	4x4, 4x4, 2x2
Max Pooling	150	2x2, 2x2, 2x2
Convolutional	250	4x4 4x4, 2x2
Max Pooling	250	2x2 2x2, 2x2
Fully Connected	500	1x1 1x1, 1x1
Fully Connected	43	1x1 1x1, 1x1

Table 3: MNIST Error Rate after fine-tuning

No.	Stage	conv1	conv2	Err(%)
1	original	100	50	0.82
2	original	200	50	0.78
3	original	300	50	0.75
4	split from [1]	**200**	50	0.58
5	merge from [4]	**100**	50	**0.59**

3.1 MNIST results

The MNIST data-set contains 60,000 training and 10,000 testing images of hand-written digits of size 28x28. The baseline model is composed of two convolutional layers and two fully-connected layers, as shown in Table 1, with ReLU and pooling following each convolutional layer. This baseline model achieves 0.82% error rate with the simple network. DCCKs training algorithm begins by splitting the first convolutional layer from 100 to 200 kernels; after the subsequent fine-tuning the model achieved 0.59% error rate, which is almost 30% relative improvement from the original model. This compared favorably to a 200 kernel model trained from scratch, which achieves 0.78%, and even a 300 kernel model trained from scratch, which achieves 0.75%. This verifies that splitting filters has the potential to help the following SGD based fine-tuning to achieve an optimal point which generalizes better. Also, more importantly after following merging step, back to 100 kernels, the performance dropped only 0.01% to an error rate of 0.59%.

3.2 GTSRB results

Table 4: Results for GTSRB1

No.	Stage	conv1	conv2	Err(%)
1	original	150	150	2.44
2	merge [1]	**32**	150	2.34
3	merge [2]	32	**32**	2.7
4	merge [2]	32	**64**	2.36
5	split [2]	**64**	150	2.5
6N	split [3]	32	**64**	2.25
7R	split [3]	32	**64**	**2.15**
8	split [1]	**300**	150	2.24
9	merge [1]	**40**	150	2.31
10	split [1]	150	**300**	2.27

Table 5: Results for GTSRB-3DNN

No.	Stage	conv1	conv2	Err(%)
1	original	150	150	1.24
2	original	16	150	1.67
3	merge [1]	**32**	150	**1.18**
4	merge [1]	**16**	150	1.25
5	split [1]	**300**	150	1.21
6	split [3]	**64**	150	**1.15**

The GTSRB data-set contains 39,209 training images and 12630 testing images with various size, with 43 different classes consisting of standard traffic signs from Germany (Houben et al., 2013). First, we resized all images to 48x48 and then we applied pre-processing

Table 6: Speed comparisons for GTSRB models

Model	**Stage**	**conv1**	**conv2**	**Speed(ms)**
1. simple	original	150	150	14.8
2. simple	merge [1]	32	150	14.1
3. simple	merge [2]	32	64	12.6
4. 3-DNNs	original	150	150	27.9
5. 3-DNNs	merge [4]	32	150	19.4

techniques such as histogram equalization, adaptive histogram equalization, and contrast normalization. For this task, we have two sets of initial networks: a single model baseline GTSRB1, consisting of three convolutional and two fully connected reaching 2.44% error rate, and larger state of the art ensemble model GTSRB-3DNN (Table 2), inspired by MCDNN(Ciresan et al., 2012), and reaching 1.24% error rate, which is within 0.2% from the best published result. We remark the ensemble models use different input size of 48x48 pixels, 38x48 pixels and 28x48 pixels: because of this, we expected a high degree of redundancy on the GTSRB-3DNN kernels which may be successfully exploited by the DCCKs merging step. Indeed, by visually inspecting the lower convolutional layers we could easily identify an abundant amount of redundancy (see Figure 2). Because of this highly redundant structure in the initial model, we inverted the sequence of our training procedure to first merge kernels instead of splitting, which maintains the accuracy and provides significantly faster training. Table 6 shows speed comparisons for GTSRB models and corresponding DCCK trained models. Test time of forward-pass with minibatches of 10 48x48 images was measured using Nvidia GeForce GTX 770.

Furthermore, the specific structure of the traffic signs provided for some peculiar behaviors on this database: for instance, kernel rotation especially helped improving performance. A detailed inspection of the recognition errors highlighted that several traffic signs were misclassified by the baseline model were highly tilted; such instances were mostly recovered and correctly recognized after DCCKs training (see Figure 1 for one example of such instance). We also remark that using centroids as new kernels resulted in better gains on this data set.

Table 4 and 5 shows the experimental results. In Table 4, 'R' denotes 'Rotation' and 'N' denotes 'Noise perturbation'. We remark that in all the experiments, in almost all cases, we either achieve significantly better performance or similar performance with significantly reduced model size. (e.g. [7R] in Table 4, which splits both the first and second convolutional layer followed by merge of the second layer, achieved the best performance). One exception we observed during the experiment is that, it shows the worst performance of all experiments when we merged the last convolutional layer which is fully connected to the first fully connected layer of this network architecture. We speculate this issue is due to the fact that is notoriously hard to optimize parameters of fully connected layers, splitting a convolutional layer which fans-out into a fully connected layer has the potential to harm the parameter structure to a point where SGD cannot easily recover.

3.3 CIFAR10 results

Table 7: Results for CIFAR-10

No.	Stage	conv1	conv2	conv3	Err(%)
1	original	192	192	192	10.4
2	split [1]	**384**	192	192	10.29
3	split [1]	**576**	192	192	10.25
4	merge [3]	**192**	192	192	**10.2**
5	split [1]	192	192	**384**	10.04
6	split [1]	192	**384**	192	10.04
7	merge [6]	192	**192**	192	10.28

The CIFAR-10 data-set consists of 50,000 training and 10,000 testing images. Each image is 32x32 pixels and represent a class of natural occurring objects. To develop the CIFAR-10 baseline we used the same techniques discussed in (Goodfellow et al., 2013) and the Network-In-Network (Lin et al., 2013) model which achieves a baseline 10.4% error rate, which is within reasonable distance from to the state of the art. When we apply DCCKs training on the CIFAR-10 data-set, the increased performance is not as large as on the previous data-sets but it is still significant and consistent. We believe that this is because the highly successful highly (manually) optimized Network-In-Network architecture makes it harder for the automatically devised DCCKs training to provide a large improvement. Therefore these results should demonstrate that DCCKs may still provide some improvement even when applied on top of more complex highly tuned architectures, while keeping the number of parameters under control. Additionally we show that by splitting layers and doubling the number of parameters we could achieve an additional 0.5% average error rate improvement.

4. Discussion

In this work, we introduced the concept of DCCKs and introduced a training procedure whereby convolutional kernels learned by SGD can be effectively split and merged. Experimental results confirmed this process results in gradually improving performance, while the training algorithm jointly optimizes structure as well as model's parameters. Results show that DCCKs can make parsimonious use of model capacity by converging towards the minimal number of parameters that gives the best performance, even when starting with highly manually optimizing network architecture. Figure 3 and 4 shows validation data accuracy and loss over fine-tune epochs; the "original" and the "merge" curves refer to training and generalization loss for models having the same number of parameters; notice how the "merge" curve is consistently above the "original" curve, apparently providing an upper-bound to the loss, and thus empirically confirming that the DCCKs architecture was indeed an improved by the training algorithm. Moreover, in some experiments, DCCKs resulted in significantly higher performance with smaller number of parameters than the original model. On the other hand, DCCKs showed bigger gains on simpler databases, such as MNIST and GTSRB, than on more complex CIFAR-10 data-set and to the more complex Network-In-Network model architecture. This is however to be expected, especially because the NIN architecture is extremely well tuned and very high performance to begin with, so it is natural to expect smaller gains by our automatic structure optimization procedure. Beside the obvious advantage of automatic structure optimization, a side benefit of DCCKs training is that manipulating kernels takes less computations than pre-processing training data, which makes DCCKs optimization more efficient.

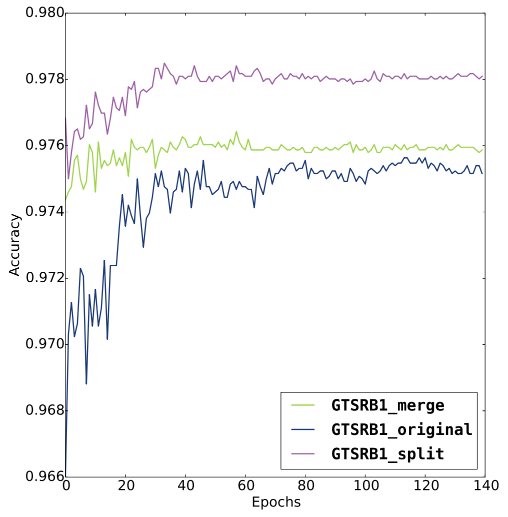

Figure 3: Test-set accuracy of GTSRB1 (simple) network during fine-tuning. Notice that GTSRB1_merge and GTSRB1_merge have the same number of parameters, but the optimized DCCK architecture shows better accuracy throughout epochs.

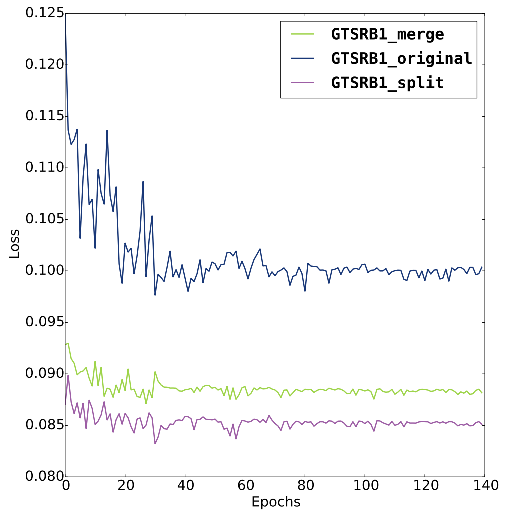

Figure 4: Test-set loss of GTSRB1 (simple) network during fine-tuning. Notice that GTSRB1_merge and GTSRB1_merge have the same number of parameters, but the optimized DCCK architecture shows better accuracy throughout epochs.

To conclude, we believe there are several aspects of DCCKs training algorithm that could be improved. As we mentioned in Section 2.2, currently all kernels are split by the same amount. However, one could argue that some kernels might be better than others and should be replicated first, possibly based on the ability provide new discriminative features. If we could determine such kernels, we could potentially improve training speed, though, final accuracy after the merge step might not be much impacted as much. Finding a more extensive set of kernel transformations to achieve a highly selective split step would also be an appropriate next step, as well as comparison and combination with logit-mimic training and model compression techniques (Ba and Caruana, 2014; Hinton et al., 2014). Ultimately, like for any new methodology in the deep learning sector, it would be very important to test how well DCCKs scale higher dimensional larger problems, such as ImageNet and to different non-vision tasks such as speech recognition or language modeling.

References

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662, 2014.

Andrew Barron, Jorma Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *Information Theory, IEEE Transactions on*, 44(6):2743–2760, 1998.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

Enrico Bocchieri and BK-W Mak. Subspace distribution clustering hidden markov model. *Speech and Audio Processing, IEEE Transactions on*, 9(3):264–275, 2001.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. URL `http://arxiv.org/abs/1410.0759`.

Dan Ciresan, Ueli Meier, and Jrgen Schmidhuber. Multi-column deep neural networks for image classification. In *IN PROCEEDINGS OF THE 25TH IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR 2012*, pages 3642–3649, 2012.

George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *ICML*, 2013.

Geoffrey Hinton, Oriol Vinyalsm, and Jeff Dean. Dark knowledge. 2014. URL `http://www.ttic.edu/dl/dark14.pdf`.

Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, pages 3–3, 1994.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4, 2012.

Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural networks*, 5 (3):501–506, 1992.

Jay J Lee, Jahwan Kim, and Jin H Kim. Data-driven design of hmm topology for online handwriting recognition. *International journal of pattern recognition and artificial intelligence*, 15(01):107–121, 2001.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013. URL `http://arxiv.org/abs/1312.4400`.

James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, 2010.

Luca Rigazio, Brice Tsakam, and Jean-Claude Junqua. An optimal bhattacharyya centroid algorithm for gaussian clustering with applications in automatic speech recognition. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1599–1602. IEEE, 2000.

Ananth Sankar. Experiments with a gaussian merging-splitting algorithm for hmm training for speech recognition. In *Proceedings of DARPA Speech Recognition Workshop*, pages 99–104. Citeseer, 1998.

Koichi Shinoda and Takao Watanabe. Mdl-based context-dependent subword modeling for speech recognition. *The Journal of the Acoustical Society of Japan (E)*, 21(2):79–86, 2000.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013.

Ilya Sutskever, Oriol Vinyals, and Quoc V V Le. Sequence to Sequence Learning with Neural Networks. In Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL `http://arxiv.org/abs/1409.4842`.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2013.

Appendix A.

In this appendix we introduce the following algorithm from Section 2.1:

Algorithm 1 Deep Clustered Convolutional Kernels training algorithm

```
Input: Initial network architecture $net$ with parameters $\lambda$, noise variance $\sigma_n$ and jitter
angle $\sigma_\alpha$, stopping conditions $\delta_{0,1,2}$ and mini-batch size $M$
while $\Delta$ Validation Accuracy $> \delta_0$ do
  while $\Delta$ Validation Accuracy $> \delta_1$ do
    // SPLIT
    $n_k$ = gaussianNoise($\sigma_n$)
    $\alpha_k$ = gaussianNoise($\sigma_\alpha$)
    $\lambda_1 = concat(\lambda, \lambda + n_k)$
    $\lambda = concat(\lambda_1, rotate(kernels(\lambda), \alpha_k))$
    // FINETUNE
    while $\Delta$ Validation Accuracy $> \delta_2$ do
      runSGD($M$ mini-batches)
    end while
  end while
  // MERGE
  $centroid = Kmeans(kernels(\lambda))$
  $\lambda = nearest(kernels(\lambda), centroid)$
  while $\Delta$ Validation Accuracy $> \delta_2$ do
    runSGD($M$ mini-batches)
  end while
end while
```