Deep Columnar Convolutional Neural Network

Somshubra Majumdar
D. J. Sanghvi College of Engineering
Mumbai, India

Ishaan Jain
D. J. Sanghvi College of Engineering
Mumbai, India

ABSTRACT

Recent developments in the field of deep learning have shown that convolutional networks with several layers can approach human level accuracy in tasks such as handwritten digit classification and object recognition. It is observed that the state-of-the-art performance is obtained from model ensembles, where several models are trained on the same data and their predictions probabilities are averaged or voted on. Here, the proposed model is a single deep and wide neural network architecture that offers near state-of-the-art performance on various image classification challenges, such as the MNIST dataset and the CIFAR-10 and CIFAR-100 datasets. On the competitive MNIST handwritten image classification challenge, the proposed model approaches the near state-of-the-art 35 model ensemble in terms of accuracy. On testing the model on the CIFAR datasets, it is found that the proposed model approaches the performance of the top two ensemble models. The architecture is also analyzed on the SVHN dataset.

Keywords

Neural Networks, Convolutional Neural Network, Computer Vision

1. INTRODUCTION

Recent publications suggest that deep neural networks achieve impressive results on various challenging tasks such as handwritten number recognition obtained from the MNIST dataset [1] and general image classification from the CIFAR 10 dataset [2]. Advances in graphic processing units have reduced the learning time for deep neural networks from several weeks or months to a few hours or days [3].

Focus is primarily on one such deep learning architecture, called as Deep Convolutional Neural Network (DCNN), which was first introduced by K. Fukushima [4], later improved by Y. Lecun [1] and further simplified by [5, 6]. Digit classification of the famous MNIST dataset is approaching near human level performance, with Ciresan [7] approaching 0.23% error and L.Wan [8] approaching 0.21% error. DCNNs show their true potential when they are wide (have several maps per layer) and deep (have several layers). Training such deep and wide DCNN requires several weeks to months on just CPUs. Even multi-threaded CPUs are not able to scale performance to adequate levels. Therefore, fast highly parallel code for GPUs has been used to overcome this limitation of CPUs. The highly parallel execution of GPU code can offer higher performance, up to two orders of magnitude greater than just using CPUs [9, 10]. The training algorithm for such deep networks are fully online and adaptive [11], such that weight updates occur after each back-propagation step. Thus the Adadelta learning algorithm has been utilized throughout, as it offers suitable performance for each iteration and does not get stuck at local optima.

It is to be noted that a deep and wide DCNN does not need unsupervised pre training or fine-tuned initialization in order to achieve near state of the art performance, though it is useful when there are fewer samples of each class available to learn. A deep columnar architecture for DCNNs is introduced, which is forked and merged at several layers in order to create very wide layers upon merging the forked layers. This approach is similar to the architecture used by Ciresan [7], termed as Multi-column Deep Neural Networks, however it also incorporates merging of the layers to create wider DCNNs for each additional learning stage.

It is seen that the proposed model approaches near state of the art performance on the MNIST data set using only a single model, and obtains slightly better results than single models which perform exceedingly well on CIFAR-10 and CIFAR-100 data sets.

2. ARCHITECTURE

The architecture has been termed as Deep Columnar Convolutional Neural Network (DCCNN) due to the fact that like MCDNN [7], the preprocessed input is also forked to connect divergent DCNNs on either same input or apply different preprocessing steps. The difference is that the forks are merged after each level using one of 2 merge operations, which significantly impacts the learning capability of the network. Initially all the weights are randomly initialized using Glorot uniform initialization [12]. Then each layer is trained using a set of data termed the training set, and validated on a set of unseen data termed as the test set. There are a few important techniques incorporated to train the network:

1. Wide Architecture: While a few models use only a few maps per layer (possess a shallow architecture) as seen in the LeNet 7 [13], the proposed model utilizes a large number of maps per layer, stacked in both horizontal (fork) and vertical (merge) layers. Fork layers usually comprise of several maps of same size or of different sizes.

2. Deep Architecture: the proposed model utilizes a deep architecture of several layers, and further compounds the depth with a large number of forked layers which are merged into a single large layer. Upon merging, the merged layer becomes a very wide layer. The deeper merge layers may have several million parameters and this increases the model performance.

3. GPU Processing: As shown by S. Hochreiter [14] and D. E. Rumelhart [15], multi-layer deep neural networks are difficult to train via standard gradient descent. However, due to improvements in modern processing power and the utilization of GPUs has greatly reduced this problem. Optimized code for massively parallel Graphic Processing Unit (GPU)

code allow for huge gains in training speed over CPU code. Given a large enough sample size to train the network the network does not require any additional pre training such as described by Ranzato [16] or the use of Restricted Boltzmann Machines [17].

4. Pooling via Convolution Subsampling: In some cases, to preserve information even during pooling operations, the max pooling layer is replaced by a convolution layer and the output of the previous layer is subsampled, an approach used by Springenberg [18].This is termed as convolutional pooling, and requires that the map size is the same size as the map size of the prior layer. As seen in [18], it provides an improvement in convergence speed and increases the size of the network.
5. Variable Kernel Size: All forked convolutional layers which accept a preprocessed image possess a 5x5 kernel. This improves the convergence speed and the accuracy of the network. All middle tier fork layers utilize either a 4x4 or 3x3 kernel, while simultaneously increasing the number of maps. The final tier fork and merge layer use a 3x3 kernel to improve performance, and often have the largest map size.
6. Fork Layers: The fork layer, also termed as a DCNN column, accepts a single input and may contain more than one convolutional layer stacked sequentially. There may be several fork layers at any given level, and the initial fork layer may accept either the same input or input which has been preprocessed by different methods. The latter is preferred, since it offers each fork a different view at the same data. Fork layers may also have different kernel size and different map size at the same level if the outputs are being concatenated at the merge layer.
7. Merge Layers: The merge layer accepts k different fork layers from the previous level and merges them using one of 2 merge operations- average and concatenate. Each of these operations merge the forked layers into a single layer which increases the width of the network. The optimal operation used during testing was found to be the concatenation operation. The merge operation can be given as:

$$merge^{n}(map_n, \quad kernel_{row}, \quad kernel_{col}) = \sum_{i=1}^{k} fork_i^{n-1}(map_i, \quad kernel_{row}, \quad kernal_{col})$$

Where n represents the nth level of the architecture, k represents the number of forked layers, map$_i$ represents the number of maps in the ith fork layer, kernel$_{row}$ is the size of kernel row and kernel$_{col}$ is the size of kernel column. The basic architecture can be seen in Figure 1.

It is to be noted that merge^n layer is the concatenation or average of all the maps of all previous fork layer. If one represents the number of maps of merge^n as map$_n$, and map$_i$ as the number of maps of the ith fork layer, then concatenation operation can be represented as:

$$map_n = \sum_{i=1}^{k} map_i$$

On the other hand, the averaging operation can be performed as the average of the maps of the k fork layers at the previous level. It can be represented as:

$$map_n = \frac{1}{k}\sum_{i=1}^{k} map_i$$

As can be seen, in the case of concatenation, the number of maps at the nth layer increase due to concatenation of each of the maps of the previous layers. Due to this, there is a marked improvement in how much information is learned from the previous level, but it also drastically increases the computation time. Convolution pooling may be applied to speed up the training process without losing the information from direct max pooling.

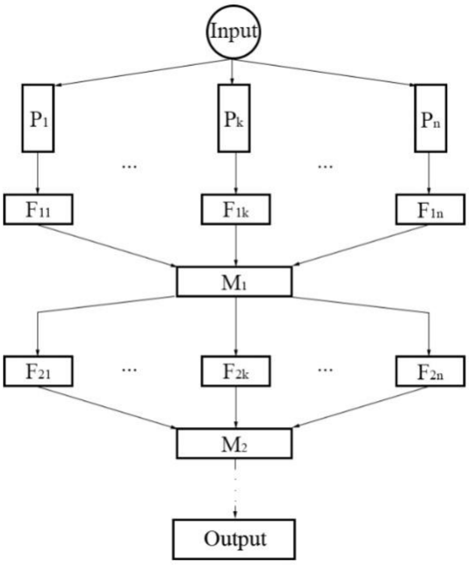

Figure 1. Basic architecture of DCCNN. P$_1$ to P$_n$ represent the various preprocessing steps that can be performed prior to the first fork layer. F$_{ij}$ represents the jth Fork layer at the ith level of the architecture. M$_1$ to M$_n$ represent the Merge layers.

3. EXPERIMENTS

The DCCNN architecture is tested on various image classification and object recognition benchmarks, and it is seen that the model approaches state of the art performance without the use of model ensembles. A basic architecture I described, which can be replicated multiple times in order to create deeper and wider networks as required for the image classification problem.

The basic model can be given as:

$$Input - \left[(32nC5)^{k1} - (M_1^{k1*})\right] - \left[(32nC4)^{ki} - (M_i^{ki*})\right]^{i} - \left[(32nC3)^{kn} - (M_n^{kn*})\right] - yD$$

Where:

1. 32nCf refers to a convolutional layer possessing 32n maps and a kernel size of fxf, where f may be any value greater than 0.

2. k^1 is the number of forks connected to the preprocessed input (same or different preprocessing steps can be applied to each fork). Common values of n at this stage are 2 - 4.

3. k^i is the number of intermediate forks which is connected to the i^{th} merge layer (M_i), and these intermediate blocks can be replicated with varying values of n, ranging from 4 - 12.

4. k^n is the number of forks of the final level which is connected to a final fully connected dense layer yD where y is the number of class labels. Optimal values of n at this stage are 12 - 16

5. The merge layer connected to the fork layers is denoted by M, subscripted with level number and superscripted k* value, which denotes the value of the merging functions. k* therefore represents the value of map_n, and therefore can be the sum or average of each map_i.

6. Convolution pooling can be applied in between each block after the merge layer, which can be represented as (32nC4 (2x2)) which indicted a convolutional layer with 2x2 subsampling and 32n maps with a 4x4 kernel size [18]. N should be selected so that the value of 32n matches the number of maps from the prior merge layer.

7. After each merge layer, Batch Normalization is used. This avoids the vanishing gradient problem and improves convergence.

8. Dropout is always applied just before the final output layer with dropout probability (0.5)

The Relu activation function was used in all layers other than the final dense layer, which used the softmax activation function. Leaky Relu activation function was tested for the CIFAR-10 and CIFAR-100 dataset and seems to offer substantial performance improvements according to Xu [19]. Therefore, the Leaky Relu activation can be utilized for those two data sets. The datasets were all trained using the Adadelta online learning algorithm [11] with initial learning rate set as 1.0 for all tests, rho value of 0.95 and epsilon value of 10^{-8}. In case Max Pooling is used instead of convolutional pooling, then a linear activation function is used for the max pool layer. The max pooling layer must also have a stride of 2x2, and can be represented by MP2.

During training steps, each mini batch was preprocessed using several transformations such as translations, scaling and rotated for CIFAR data sets, whereas original images are used for validation. Training ends when the validation loss is zero or when validation loss increases repeatedly for 50 epochs. Weights are initialized using the glorot uniform initialization method [12]. In all experiments, the merge method utilized was the concatenation operation, since it offered slightly better results as compared to the averaging operation.

3.1 MNIST

The MNIST dataset [1] is normalized by dividing the pixel value by 255.0 for the grayscale channel. The entire training data set is utilized and simple transformations are applied to each batch randomly. Each mini batch can be randomly rotated (maximum of 20 degrees), width shifted (at most 20% of original width) and height shifted (at most 20% of the original height). The architecture selected for this data set is as follows:

$$1x28x28 - \left[(64C5)^2 - (M_1^{128})\ \right] - MP2 - \left[(128C4)^2 - (M_2^{256})\right] - MP2 - \left[(256C3)^6 -\ (M_3^{1536})\right] - MP2 -\ 10D$$

The architecture for the MNIST dataset can be seen in figure 2.1. This network architecture possesses approximately 4.3 million parameters and is trained for 400 epochs. Training time for this data set is roughly 98 seconds per epoch on a NVIDIA 980M GPU and total training time is approximately 11 hours. It is observed that accuracy does not improve over 350 epochs.

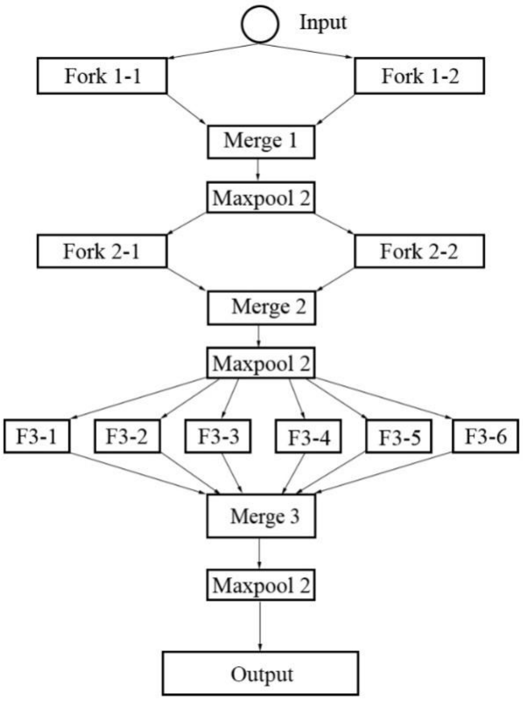

Figure 2.1 DCCNN architecture for MNIST dataset. It has a 3 level architecture, with Fork – Merge – Pool blocks.

The resultant accuracy of this network is compared to various models in Table 1. DCCNN trained with initial preprocessing steps being the same for level 1 fork layer accuracy is found to be equivalent to DCCNN trained with different preprocessing steps for the level 1 fork layer, however convergence is slightly faster (453 epochs vs 439 epochs) for different preprocessing. DCCNN has a very low error rate of 0.23%, which is equivalent to MCDNN [7], and slightly more than the state of the art 0.21% achieved by DropConnect [8], both of which utilize ensembles of 35 and 5 independent networks respectively and then vote on their predictions.

Thus the single model performance of the DCCNN rivals two large ensemble models. In addition, the DCCNN architecture can also be used in an ensemble of DCCNN models to achieve even higher accuracy, perhaps reaching the human level accuracy of less than 0.2 % error rate on the MNIST dataset.

Table 1. Results on MNIST Dataset

Method	Error Rate %	Paper
CNN	0.40	[6]
CNN	0.32	[21]
CNN Committee	0.27 ± 0.02	[22]

DCCNN	**0.23**	This
MCDNN (35 Column Ensemble)	**0.23**	[7]
DropConnect (5 Model Ensemble)	**0.21**	[8]

Many of the wrongly identified digits contain random strokes or have wrong labels associated with them. The 23 error cases are associated with 23 correct second guesses, improving over the MCDNN [7] with only 20 correct second guesses. The 23 images which were not classified correctly are shown in Figure 2.2, with the prediction of the network on the bottom left and the true label of the picture on the bottom right of each image. It can be seen that several images have been wrongly classified by humans, but correctly classified by the network.

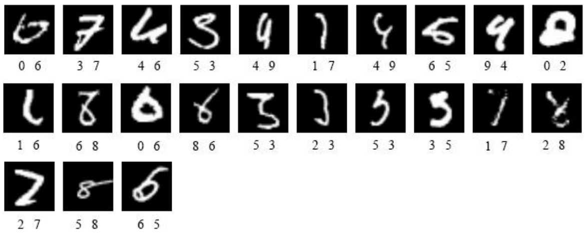

Figure 2.2 Digits which are incorrectly classified by DCCNN. The bottom left number of each image represents the predicted value by the model. The bottom right of each image number represents the class label assigned to this image.

To ensure that performance was due to the DCCNN architecture and not purely due to preprocessing steps, the model was tested on the original data set as well. On the original data set, error rose slightly to 0.24%. Therefore, it is certain that the single model performance is attributed to the architecture itself, and is augmented by preprocessing steps as described above. It is possible that this error can further be minimized by using an ensemble of DCCNN but this has not been tested yet. Thus, a conclusion can be drawn that DCCNN outperforms MCDNN in terms of correct second guesses, and performs well in comparison to other large ensemble models such as DropConnect and CNN committees.

3.2 CIFAR-10

The CIFAR-10 dataset [2] is a collection of 32x32 pixel size images which belong to one of 10 distinct classes. Each class contains 5000 training samples and another 1000 samples for testing. Even within the same class, images vary greatly from one another. In some cases they may not be centered or may contain only part of the object to recognize and usually have different backgrounds for the same object. The objects may vary in size and color or texture of the objects vary greatly as well.

For a complex and challenging image classification problem like the CIFAR-10 dataset, a deeper architecture than the simple MNIST DCCNN architecture as well as image preprocessing is utilized to improve the score. Like in the MNIST dataset, the pixels of each channel are divide by 255.0 to normalize the image over RGB channels. Simple transformations are also applied to each mini batch randomly, such that the images can be rotated (up to 15 degrees), width and height shifted (up to 15 % of original width/height), sheared (up to 1 radian) and scaled (up to 20% of original scale). The architecture selected for CIFAR-10 is as follows.

$$3x32x32 - \left[(96C5 - 96C3)^2 - (M_1^{192})\ \right] - 96C5\ (2x2) - \left[(192C3 - 192C3)^3 - \left(M_2^{576}\right)\right] - 192C3\ (2x2) - \left[([256C1],[256C3],[256C5])^1 - \ (M_3^{768})\right] - 256C3\ (2x2)\ -\ 10D$$

The DCCNN architecture for the can be seen in Figure 3.

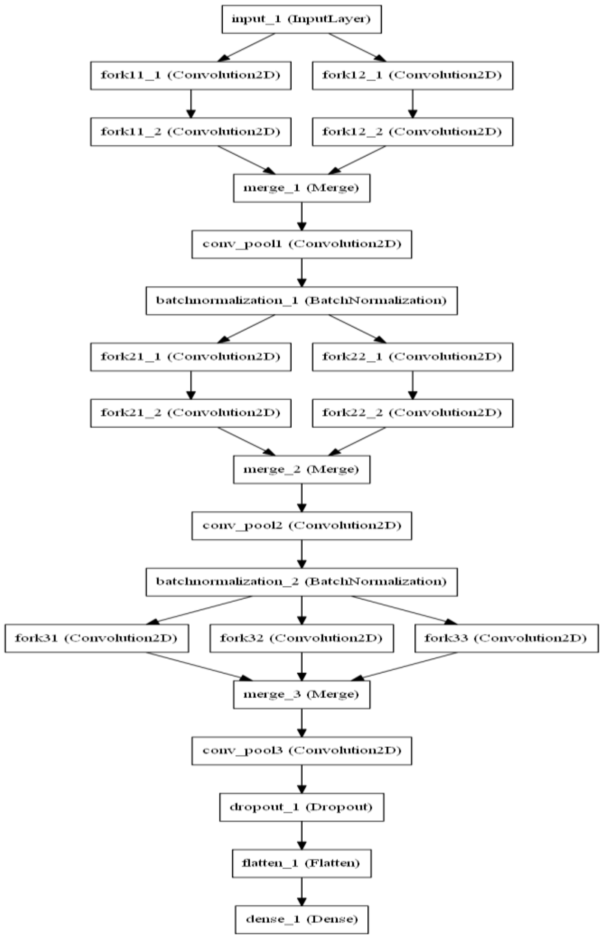

Figure 3. DCCNN architecture for CIFAR-10 dataset. Convolutional Pooling is used instead of Max Pooling.

The network is a modification of the ALL-CNN-C architecture proposed by Springenberg [18]. It utilizes the convolution pooling technique via subsampling in order to maximize the information that the network can learn by avoiding max pooling. Unlike their architecture which stacks multiple convolution layers sequentially, the DCCNN stacks the same layers horizontally and then vertically. Adding an additional level with large number of forks and large filters will outperform the ALL-CNN-C architecture, at the cost of nearly 8 times the number of parameters.

This network architecture possesses approximately 5.5 million parameters and is trained for 600 epochs. Training time for this dataset is approximately 234 seconds per epoch on a NVIDIA 980M GPU, and total training time is approximately 39 hours. It is observed that after 550 iterations there was no further improvement. The resultant accuracy of this model is compared to various models in Table 2.

Table 2. Results on CIFAR-10 Dataset

Method	Error Rate %	Paper
MCDNN	11.21	[7]
Maxout	9.38	[23]
DropConnect	9.32	[8]
Network in Network	8.81	[24]
ALL-CNN-C (Small augmentation)	7.25	[18]
DCCNN	**6.90**	This
Exponential Linear Unit CNN	**6.25**	[25]
Large-ALL-CNN (Large augmentation)	**4.41**	[18]
Fractional Max Pooling	**3.47**	[21]

For this dataset, presenting different preprocessed image to each initial fork offered no additional benefit to validation accuracy, however it did improve the convergence time slightly (maximum validation accuracy score occurred at the 557th epoch for same preprocessed images to all initial layer, in contrast to the 538th epoch for different preprocessed images to each initial layer).

DCCNN has a low error rate of just 6.90% compared to the ALL-CNN-C architecture (7.25%, [18]) with small data augmentation, but is higher than the state of the art Fractional Max Pooling (3.47%, [21]). It must be noted that the fractional max pooling architecture is an ensemble model with over 50 million parameters. The Large-ALL-CNN-C architecture however achieves a very high accuracy (4.41%, [18]) by using very large data augmentation and larger image sizes (128x128) to allow for more depth in the network and larger number of parameters as compared to the CIFAR-10 DCCNN.

Considering only small data augmentations, the proposed network performs better than the comparable ALL-CNN-C architecture. Out of the 690 wrong predictions, 501 second guess predictions were correct. Therefore, the top-2 prediction error rate is 1.89%

3.3 CIFAR-100

The CIFAR-100 dataset [2] is a collection of 32x32 pixel size images which belong to one of 10 distinct classes, and also belong to one of 100 distinct subclasses. Each class contains 5000 training samples and another 1000 samples for testing. Therefore there are 500 training sample and 100 testing samples of each subclass. Even within the same subclass, images vary greatly from one another. In some cases they may not be centered or may contain only part of the object to recognize and usually have different backgrounds for the same object. The objects may vary in size and color or texture of the objects vary greatly as well.

For a complex and challenging image classification problem like the CIFAR-100 dataset, a deeper architecture than the CIFAR-10 DCCNN architecture as well as image preprocessing is utilized to improve the score. Like in the CIFAR-10 dataset, pixels of each channel are divided by 255.0 to normalize the image over RGB channels. One can also apply simple transformations to each mini batch randomly, such that the images can be rotated (up to 10 degrees), width and height shifted (up to 15 % of original width/height), sheared (up to 2 radian) and scaled (up to 20% of original scale). The architecture selected for CIFAR-100 is as follows:

$$3x32x32 - [(96C5 - 96C5)^{2} - (M_{1}^{192})\ ] - 96C5\ (2x2) - 192C3{-}192C32{-}M2384{-}192C4 \quad 2x2{-} 256C1x3{-}256C3x1{,}256C3x1{-}256C1x3,\ 256C1,\ 256C3,\ 256C51{-}M31280{-}\ 192C3\ (2x2) - 100D$$

The network described above is different from the earlier described models since it uses a deeper architecture. The first two fork layers are composed of stacked convolution layers, and the final fork layer consists of 2 stacks of 1x3 and 3x1 kernel size convolutional layers, along with 3 other convolutions layers of different kernel sizes (1x1, 3x3 and 5x5). The network is a modification of the Large-ALL-CNN architecture proposed by Springenberg [18]. It also utilizes the convolution pooling via subsampling as was used for the CIFAR-10 dataset. additional levels are added to increase the depth of the network further, with more number of forks and larger maps. It is possible to create an even deeper model, however the training time required increases sharply since the number of parameters also increases. An attempt was made to create a deep architecture which balances the depth with the number of parameters and attempts to maximize the performance while reducing the training time.

This network architecture possesses approximately 7.6 million parameters and is trained for 600 epochs. Training time for this dataset is approximately 334 seconds per epoch on a NVIDIA 980M GPU, and total training time is approximately 56 hours. It is observed that after 570 iterations there was no further improvement. It is to be noted that due to use of Adadelta learning algorithm it is not guaranteed to produce exactly same results, but it should approach similar results.

,

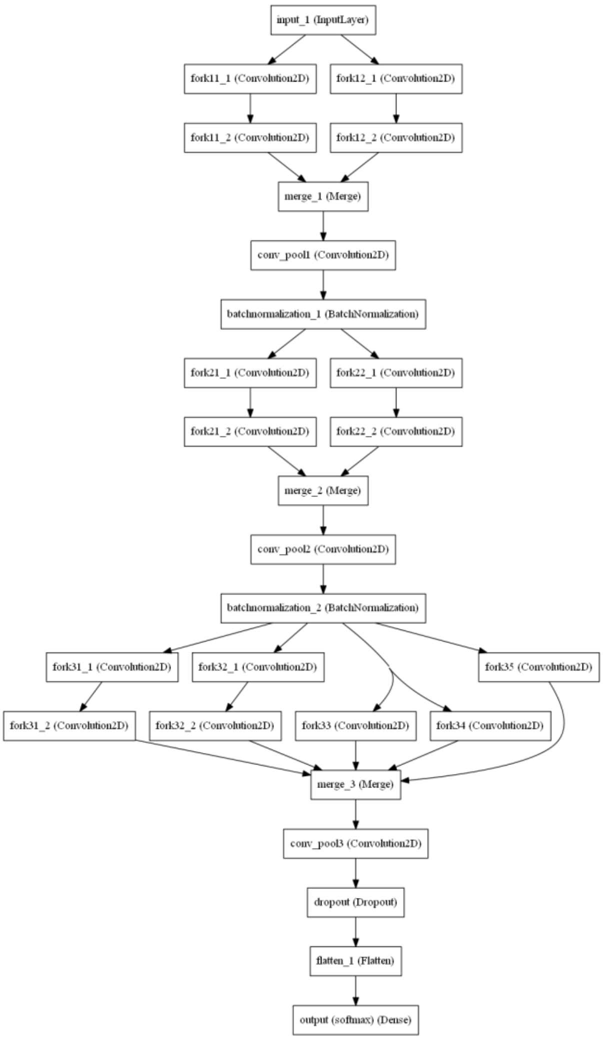

Figure 4. DCCNN architecture for the CIFAR-100 dataset. Similar to CIFAR-10 architecture, Convolutional Pooling is used instead of Max Pooling.

As can be seen in the Figure 4, which represents the DCCNN architecture for the CIFAR-100 Dataset, 6 forks at the last layer of the CIFAR-10 architecture have been replaced with 5 forks. Each fork layer also contains stacked convolutional layers. Due to this architecture, DCCNN is able to perform slightly better than the simpler CIFAR-10 DCCNN on this dataset. While the overall number of filters at the last layer may be less than that of the CIFAR-10 architecture, the performance is better due to the additional depth of the network.

The resultant accuracy of this model is compared to various models in Table 3. DCCNN has a low error rate of just 28.63% compared to the ALL-CNN-C architecture (33.71%, [18]), but is higher than the state of the art Fractional Max Pooling (26.39%, [21]). It must be noted that the fractional max pooling architecture is an ensemble model with over 50 million parameters. Considering only small data augmentations, the proposed network performs better than the comparable ALL-CNN-C architecture, and approaches similar performance to the ensemble of Fractional Max Pooling networks. However, the performance of DCCNN is poor compared to the 18 convolutional layer ELU architecture [25] which does not use model ensemble, but possesses roughly 39.4 million parameters and requires roughly 165,000 iterations to achieve such high performance. An attempt has been made to create a deeper network to see if DCCNN performance can come close to the ELU model, but considering the relatively small size of the network and shorter training time, it performs adequately well. It is seen that out of the 2873 wrong predictions, 1124 second guess predictions were correct. Therefore top-2 predictions error rate is 17.39%.

Table 3. Results of CIFAR-100 Dataset

Method	Error Rate %	Paper
Network in Network	35.68	[24]
Maxout	34.58	[23]
ALL-CNN-C	33.71	[18]
DCCNN	**28.63**	This
Fractional Max Pooling	**26.39**	[21]
Exponential Linear Unit CNN	**24.28**	[25]

3.4 SVHN

The Street View House Numbers (SVHN) Dataset [26] is a real-world image dataset, which is similar to the MNIST dataset. However, it incorporates an order of magnitude more label data (over 600,000 images). It is also more complex to train, given the fact that this is a real world problem, since the images need to be recognized in different backgrounds and in different natural scenes. The SVHN dataset it obtained from house numbers in Google Street View Images. A few examples of the SVHN dataset can be seen in Figure 5.

Figure 5. Image data of the SVHN dataset [26]

The SVHN dataset comes in two formats: original images with character level bounded boxes and 32x32 images centered on a single character (with most images having some distracting elements near the true number). Each of these datasets have 73257 training samples and 26032 testing samples, along with a further 531131 extra significantly easier extra training samples. The 32x32 cropped image dataset is used, and two tests are run: first training-testing on the smaller data set, and then adding the extra samples and training-testing again.

The dataset is normalized by dividing the pixel value by 255.0 for the RGB channel. The entire training dataset is utilized and simple transformations are applied to each batch randomly. Each mini batch can be randomly rotated (maximum of 20 degrees), width shifted (at most 20% of original width) and height shifted (at most 20% of the original height). The architecture selected for this data set is as follows:

$$1x28x28 - \left[(64C5)^2 - (M_1^{128})\ \right] - 64C5\ (2x2) - \left[(128C4)^2 - \left(M_2^{256}\right)\right] - 128C4\ (2x2) - \left[(256C3)^6 - \left(M_3^{1536}\right)\right] - 256C3\ (2x2) -\ 10D$$

Since the dataset is comparable to the MNIST dataset, the MNIST DCCNN architecture as described above in Figure 2.1 is utilized. However, The Max Pooling layer is replaced with a Convolutional Pooling layer. The network contains approximately 6.3 million parameters. This network is trained with 200 epochs for the smaller dataset, and then for 200 epochs with the full dataset. This is done to understand the model performance on the small amount of complex samples and then on the large amount of simpler samples. Training time for the full dataset is approximately 800 seconds per epoch, and total training time is approximately 44 hours.

The resultant accuracy of both tests is shown in Table 4. The DCCNN architecture is able to perform well on both tests, but does not reach near state-of-the-art performance on either test. This may be attributed to the simpler architecture of the MNIST dataset which uses fewer fork layers and smaller number of maps than the CIFAR-100 DCCNN model. Using a deeper model was not possible due to GPU memory constraints, but it would help improve the score significantly.

Table 4. Results of SVHN Dataset

Method	Error Rate %	Paper
DCCNN (small training set)	5.92	This
DCCNN (full dataset)	**1.92**	This
Deeply-Supervised Net	1.92	[27]
Maxout NIN	1.81	[28]
Recurrent Convolutional Network	1.77	[29]
Competitive Multi-Scale Convolution	1.76 ± 0.07	[30]
Tree + Max – Avg Pooling	**1.69**	[31]

4. CONCLUSION

Deep neural networks are one of the most powerful methods to accomplish tasks such as image classification and object recognition. There are currently a few models whose performance rivals that of even human beings. Often, these models comprise of committees of networks and thus require vast amounts of computation time. Here, a single model is shown which performs as well as or only slightly worse than such committees. It is also shown that DCCNNs have similar performance to state of the art convolutional network architectures in image classification challenges such as MNIST, CIFAR-10, CIFAR-100 and SVHN with the use of a single model.

This paper highlights the possibility of replacing model ensembles with a single similar performing model. This principle can be extended to using said single model in a, more complex, model ensemble. To put it into perspective, this would be similar to an ensemble of ensembles, which has a great scope for future development.

5. REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner." Gradient based learning applied to document recognition". Proceedings of the IEEE, 86(11):2278–2324, November 1998.

[2] A. Krizhevsky." Learning multiple layers of features from tiny images". Master's thesis, Computer Science Department, University of Toronto, 2009. 1

[3] Raina, Rajat, Anand Madhavan, and Andrew Y. Ng. "Large-scale Deep Unsupervised Learning Using Graphics Processors." Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09 (2009). Print.

[4] Fukushima, Kunihiko. "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position."Biol. Cybernetics Biological Cybernetics 36.4 (1980): 193-202. Print.

[5] Behnke, Sven. "Hierarchical Neural Networks for Image Interpretation."Lecture Notes in Computer Science(2003). Print.

[6] Simard, P.y., D. Steinkraus, and J.c. Platt. "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis." Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings. Print.

[7] Ciresan, D., U. Meier, and J. Schmidhuber. "Multi-column Deep Neural Networks for Image Classification." 2012 IEEE Conference on Computer Vision and Pattern Recognition (2012). Print.

[8] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus. "Regularization of Neural Network using DropConnect". International Conference on Machine Learning 2013

[9] Strigl, Daniel, Klaus Kofler, and Stefan Podlipnig. "Performance and Scalability of GPU-Based Convolutional Neural Networks."2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (2010). Print.

[10] Uetz, Rafael, and Sven Behnke. "Large-scale Object Recognition with CUDA-accelerated Hierarchical Neural Networks." 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems (2009). Print.

[11] Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012).

[12] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International conference on artificial intelligence and statistics. 2010.

[13] Lecun, Y., Fu Jie Huang, and L. Bottou. "Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting."Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.Print.

[14] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies”. In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning internal representations by error propagation”. In Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations, pages 318–362. MIT Press, Cambridge, MA, USA, 1986

[16] Ranzato, Marc'aurelio, Fu Jie Huang, Y-Lan Boureau, and Yann Lecun. "Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition."2007 IEEE Conference on Computer Vision and Pattern Recognition (2007). Print.

[17] Erhan, Dumitru, et al. "Why does unsupervised pre-training help deep learning?." The Journal of Machine Learning Research 11 (2010): 625-660.

[18] Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." arXiv preprint arXiv:1412.6806 (2014).

[19] Xu, Bing, et al. "Empirical evaluation of rectified activations in convolutional network." arXiv preprint arXiv:1505.00853 (2015).

[20] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. “Deep, big, simple neural nets for handwritten digit recognition”. Neural Computation, 22(12):3207–3220, 2010.

[21] Graham, Benjamin. "Fractional max-pooling." arXiv preprint arXiv:1412.6071(2014).

[22] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. In International Conference on Document Analysis and Recognition, pages 1250–1254, 2011.

[23] Goodfellow, Ian J., et al. "Maxout networks." arXiv preprint arXiv:1302.4389(2013).

[24] Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." arXiv preprint arXiv:1312.4400 (2013).

[25] Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." arXiv preprint arXiv:1511.07289 (2015).

[26] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng. “Reading Digits in Natural Images with Unsupervised Feature Learning.” NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.

[27] Lee, Chen-Yu, et al. "Deeply-supervised nets." arXiv preprint arXiv:1409.5185 (2014).

[28] Chang, Jia-Ren, and Yong-Sheng Chen. "Batch-normalized Maxout Network in Network." arXiv preprint arXiv:1511.02583 (2015).

[29] Liang, Ming, and Xiaolin Hu. "Recurrent convolutional neural network for object recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[30] Liao, Zhibin, and Gustavo Carneiro. "Competitive Multi-scale Convolution." arXiv preprint arXiv:1511.05635 (2015).

[31] Lee, Chen-Yu, Patrick W. Gallagher, and Zhuowen Tu. "Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree." arXiv preprint arXiv:1509.08985 (2015).