AN ENSEMBLE OF SIMPLE CONVOLUTIONAL NEURAL NETWORK MODELS FOR MNIST DIGIT RECOGNITION

Sanghyeon An **Minjun Lee** **Sanglee Park** **Heerin Yang** **Jungmin So**

Department of Computer Science and Engineering
Sogang University

ABSTRACT

We report that a very high accuracy on the MNIST test set can be achieved by using simple convolutional neural network (CNN) models. We use three different models with 3×3, 5×5, and 7×7 kernel size in the convolution layers. Each model consists of a set of convolution layers followed by a single fully connected layer. Every convolution layer uses batch normalization and ReLU activation, while pooling is not used. Rotation and translation is used to augment training data, which is a technique frequently used in most image classification tasks. A majority voting using the three models independently trained on the training set can achieve up to 99.87% accuracy on the test set, which is one of the state-of-the-art results. A two-layer ensemble, a heterogeneous ensemble of three homogeneous ensemble networks, can achieve up to 99.91% test accuracy. The results can be reproduced by using the code at https://github.com/ansh941/MnistSimpleCNN.

***K*eywords** image classification · MNIST

1 Introduction

MNIST handwritten digit recognition data set (Figure 1, [1]) is one of the most basic data sets used to test performance of neural network models and learning techniques. Using 60,000 images as the training set, a 97%-98% accuracy could easily be achieved on the test set of 10,000 images, with learning methods such as k-nearest neighbors (KNN), random forests, support vector machines (SVM) and simple neural network models. Convolutional neural networks (CNN) improve this accuracy to over 99% with less than 100 misclassified images in the test set.

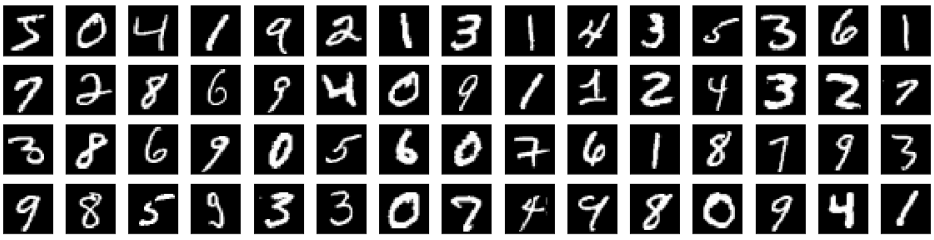


Figure 1: Images from the MNIST training set.

The final 100 images are more difficult to classify correctly. In order to improve accuracy after 99%, we need more complex models, careful tuning of hyperparameters such as learning rate and batch size, regularization techniques such as batch normalization and dropout, and augmentation of training data. The highest accuracy achieved on the MNIST test set are approximately 99.7% to 99.84%, as reported in the papers [2, 3, 4, 5, 6].

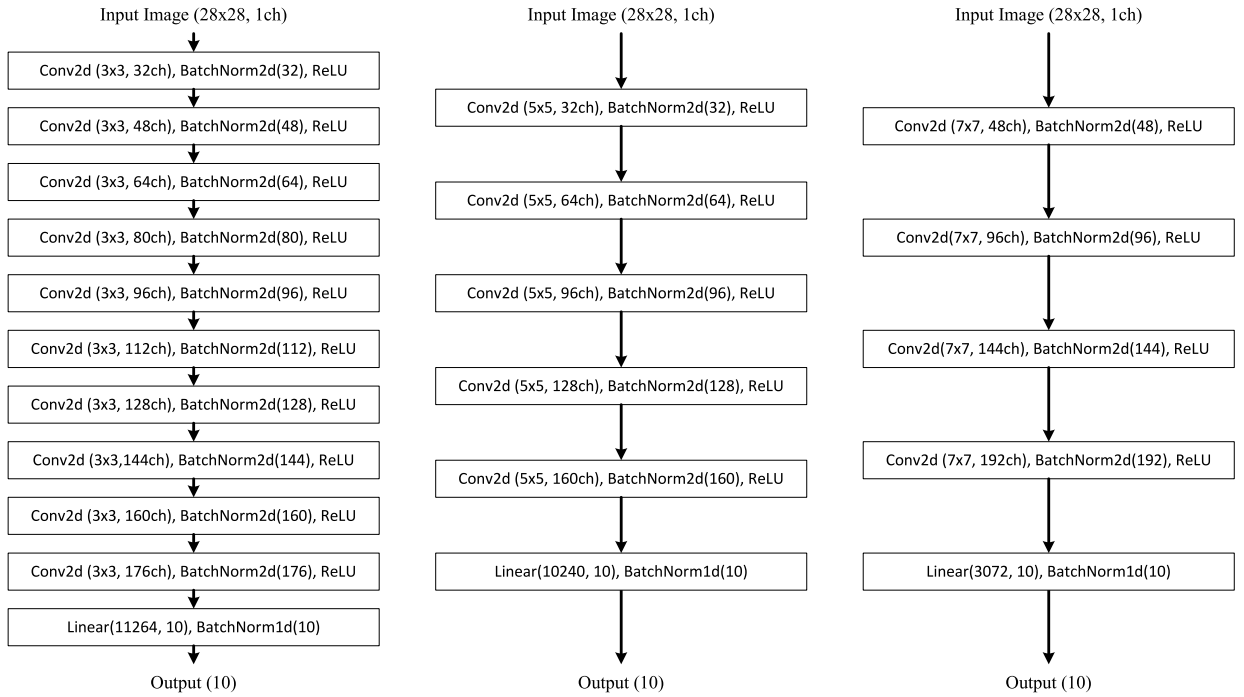
In this paper, we report a model that can achieve a very high accuracy on the MNIST test set without complex structural aspects or learning techniques. The model uses a set of convolution layers followed by a fully connected layer at the end, which is one of the commonly used model architectures. We use basic data augmentation schemes, translation and rotation. We train three models with similar architectures, and use majority voting between the models to obtain the

final prediction. The three models have similar architectures, but have different kernel sizes in the convolution layers. Experiments show that combining models with different kernel sizes achieves better accuracy than combining models with the same kernel size.

2 Network Design and Training

Our network models consist of multiple convolution layers and a fully connected layer at the end. In each convolution layer, a 2D convolution is performed, followed by a 2D batch normalization and ReLU activation. Max pooling or average pooling is not used after convolution. Instead, the size of feature map is reduced after each convolution because padding is not used. For example, if we use a 3×3 kernel, the width and height of the image is reduced by two after each convolution layer. Similar approach is taken in other networks [6, 2]. The number of channels is increased after each layer in order to account for reduction in feature map size. Once the feature map size becomes small enough, a fully-connected layer connects the feature map to the final output. A 1D batch normalization is used at the fully-connected layer, while dropout is not used.

We use three different networks and combine the results from these networks. The networks differ only in the kernel sizes of the convolution layers: 3×3, 5×5, and 7×7. Because different kernel size lead to different size reduction in feature maps, the number of layers is different for each network. The first network, M_3, uses 10 convolution layers with $16(i+1)$ channels in ith convolution layer. The feature map becomes 8×8 with 176 channels after the 10th layer. The second network, M_5, uses 5 convolution layers with $32i$ channels in ith convolution layer. The feature map becomes 8×8 with 160 channels after the 5th layer. The third network, M_7, uses 4 convolution layers with $48i$ channels in ith convolution layer. The feature map becomes 4×4 with 192 channels after the 4th layer. The structure of the three networks are shown in Figure 2.



(a) M_3 (b) M_5 (c) M_7

Figure 2: Network models used for MNIST digit classification.

When training, we apply transformation on data that consist of random translation and random rotation. For random translation, an image is randomly shifted horizontally and vertically, up to 20% of the image size in each direction. For random rotation, the image is rotated up to 20 degrees in either clockwise or counterclockwise direction. The amount of transformation varies for each image and each epoch, so the network gets to see various versions of an image in the training set (Figure 3). For training and evaluation, the input vectors which are typically integers in [0, 255] are converted to floating point values in [-1.0, 1.0].

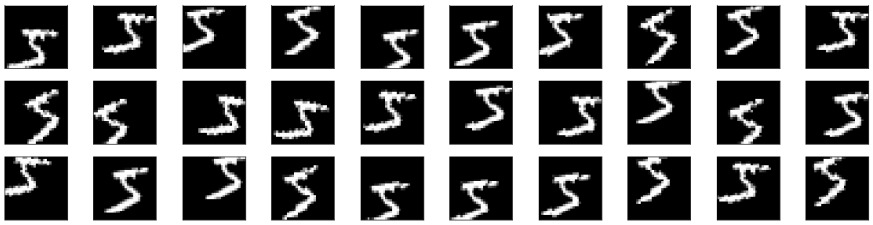


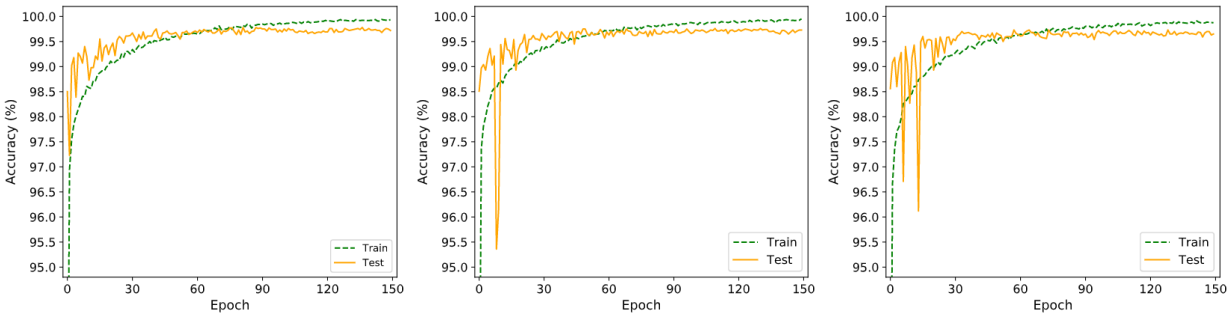
Figure 3: Random translation and random rotation applied to a training image.

The network parameters are initialized using default initialization methods in PyTorch [7]. For parameter optimization, we use the Adam optimizer with cross-entropy loss function. Learning rate starts at 0.001, and exponentially decays with decaying factor γ=0.98. The batch size is 120, and so 500 parameter updates occur in an epoch. We use exponential moving average of weights for evaluation, which may lead to better generalization [8]. The exponential decay used for computing the moving average is 0.999.

3 Experiments

3.1 Results for Individual Networks and Ensembles

For each type of network, we have trained 30 networks with different initial parameters. Each network was trained for 150 epochs, since the test accuracy hardly improved after that point. Figure 4 shows the change in the training accuracy and the test accuracy while training. In terms of test accuracy, networks with larger kernels show some instability at early epochs, but the patterns of all networks become similar after 50 epochs. Table 1 shows the minimum, average, maximum accuracy of 30 networks between 50 and 150 epochs, in the 95% confidence range. The accuracy of M_3 is slightly higher followed by M_5 and M_7, but the difference is not too significant (less than 0.02%). Between 50 and 150 epochs of 30 networks, the highest test accuracy observed from M_3, M_5, M_7 was 99.82, 99.80, and 99.79 respectively.



(a) M_3 (b) M_5 (c) M_7

Figure 4: Train accuracy and test accuracy of M_3, M_5, and M_7 during training.

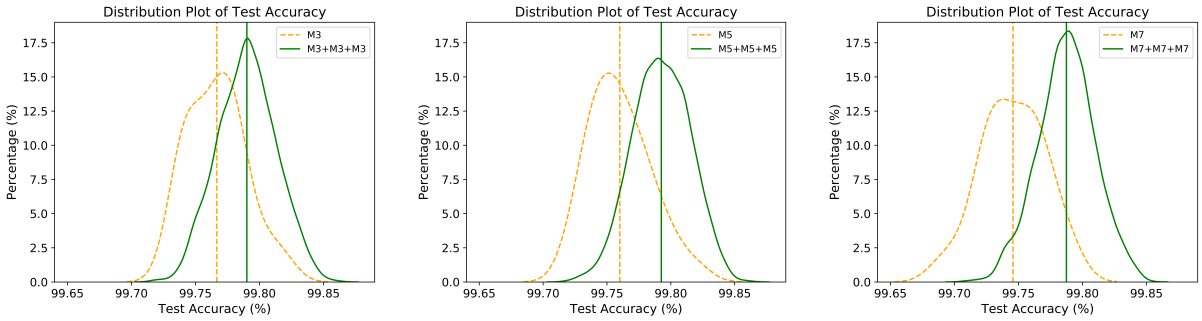
Table 1: Test accuracy of networks measured between 50 epoch and 150 epoch in training.

model	test accuracy			
	min	avg	max	best
M_3	99.5930 ± 0.0136	99.6949 ± 0.0058	99.7667 ± 0.0084	99.82
M_5	99.5863 ± 0.0115	99.6835 ± 0.0074	99.7583 ± 0.0081	99.80
M_7	99.5470 ± 0.0288	99.6711 ± 0.0089	99.7450 ± 0.0093	99.79

It is known that using ensemble of networks can improve generalization and achieve higher test accuracy [9, 10, 11, 12]. To test the performance of ensemble networks on the MNIST data set, we trained 30 networks each of M_3, M_5, and M_7, and tested four different ensemble strategies. In the first three strategies, we randomly select three networks from

the same type of networks (M_3, M_5, or M_7). In the fourth strategy, we select one network from each type. The final result is obtained by using majority voting. That is, if two networks agree that an image belongs to a particular class, that class is selected. If the three networks vote on different class, one class is randomly selected among the three. For each strategy, we tested 1000 ensemble networks and plotted the histogram for the test accuracy.

Figure 5 shows the benefit of using ensemble of homogeneous networks. For M_3, M_5, and M_7, higher test accuracy could be achieved by combining results from three networks. (The line moves to the right.) Figure 6 shows the test accuracy of the four ensemble methods discussed above, and Table 2 shows the 95% confidence range of test accuracy for the four methods. It can be observed that while the average test accuracy of homogeneous ensemble methods are similar, the ensemble method where one network is selected from each type of networks achieves higher accuracy.



(a) M_3 (b) M_5 (c) M_7

Figure 5: Distribution of test accuracy for individual networks and homogeneous ensemble networks.

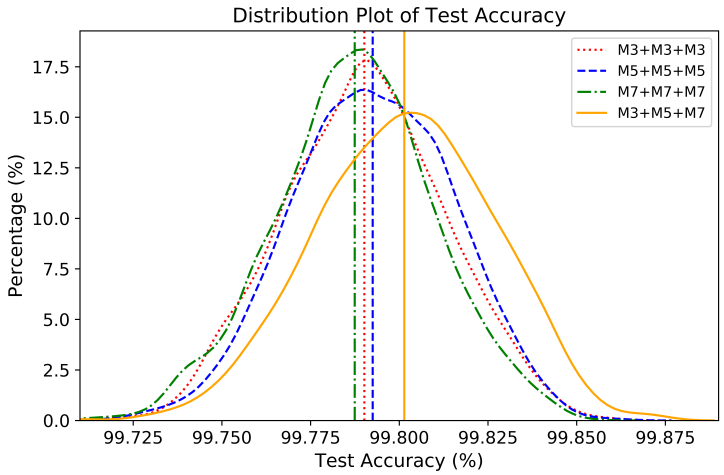


Figure 6: Distribution of test accuracy for homogeneous and heterogeneous ensemble networks.

From Figure 5 we can see that using an ensemble of three homogeneous networks could improve the test accuracy. Also, it is shown in Figure 6 that combining results from heterogeneous networks could help boost the accuracy as well. We have tested a two-level ensemble method, where we first combine results from three homogeneous networks, and then combine results from three homogeneous ensemble networks. For this study, we trained 3 groups of 10 networks for each type of network, M_3, M_5, and M_7. For each network, we trained for 150 epochs and saved the best model in terms of test accuracy. Then, we randomly chose 3 networks from M_3 and combined their results using majority voting. Similarly, we combined results of three networks for M_5 and M_7. After that, we used majority voting for the three ensemble networks. Figure 7 shows the distribution of test accuracy for 1000 ensemble of individual networks (M_3+M_5+M_7) and 1000 ensemble of ensemble networks ((M_3+M_3+M_3)+(M_5+M_5+M_5)+(M_7+M_7+M_7)). The graph shows that using ensemble of ensemble networks improves the test accuracy in average. Table 3 shows the 95% confidence range and the best accuracy observed for ensemble of individual and ensemble of ensemble networks.

Table 2: 95% confidence range of test accuracy for homogeneous and heterogeneous ensemble networks.

configuration	test accuracy	
	95% confidence range	best accuracy
$M_3 + M_3 + M_3$	99.7901 ± 0.0014	99.86
$M_5 + M_5 + M_5$	99.7925 ± 0.0014	99.86
$M_7 + M_7 + M_7$	99.7874 ± 0.0014	99.85
$M_3 + M_5 + M_7$	$\mathbf{99.8014 \pm 0.0015}$	**99.87**

In addition to random selection, we also show the best case in order to see what is the best accuracy we can achieve. For the best case, we picked 10 homogeneous ensemble networks from M_3, M_5, and M_7 that shows the best test accuracy. Then, we chose one network from each type and combined their results. The best accuracy achieved was 99.91%.

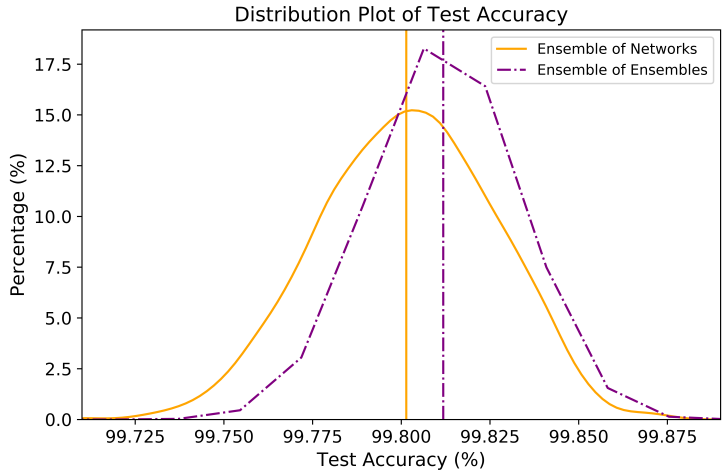


Figure 7: Distribution of test accuracy for ensemble of individual networks and ensemble of ensemble networks.

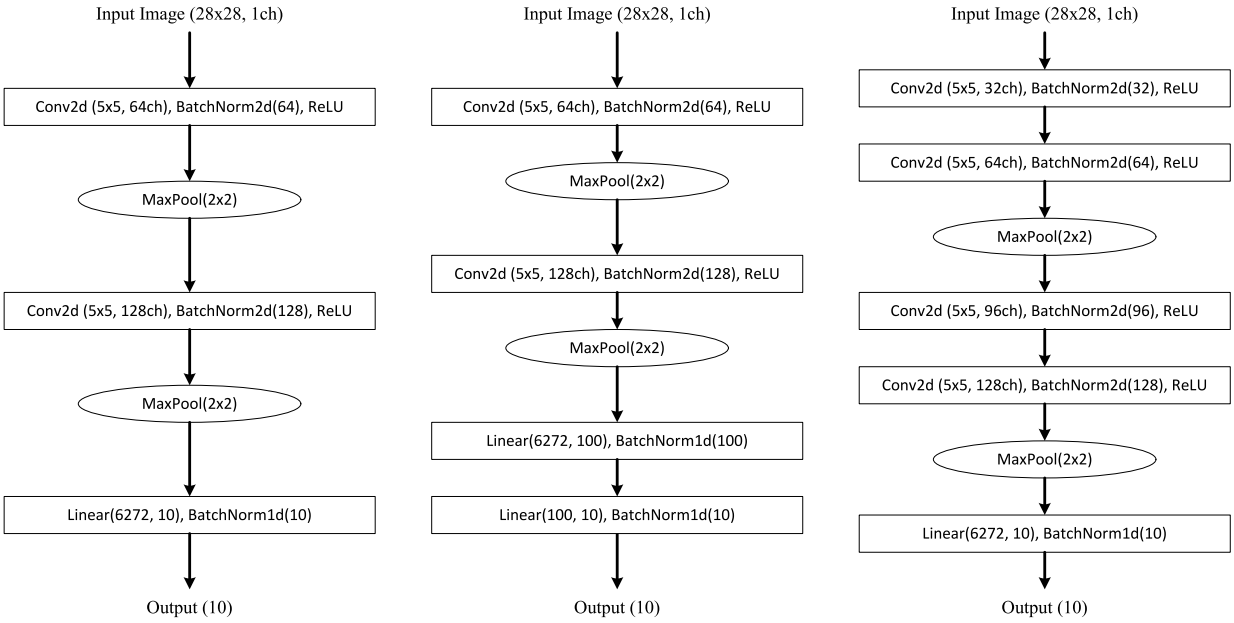
Table 3: 95% confidence range and best-case test accuracy for ensemble of individual networks and ensemble of ensemble networks.

configuration	test accuracy	
	95% confidence range	best accuracy
ensemble of individual networks	99.8014 ± 0.0015	99.87
ensemble of ensembles (random)	99.8118 ± 0.0002	99.89
ensemble of ensembles (best)	99.8646 ± 0.0008	99.91

3.2 Impact of network architecture

When building a CNN, a common practice is to use pooling, such as max pooling or average pooling [13]. Pooling is used to obtain translation invariance and also reduce dimension of the feature maps. A commonly used CNN model consists of a set of convolution layers where each convolution layer is followed by a pooling layer, and one or multiple fully connected layers at the end. Some networks have two convolution layers before the pooling layer. Figure 8 show some of the commonly used CNN structures, and we name the three networks C_1, C_2, and C_3.

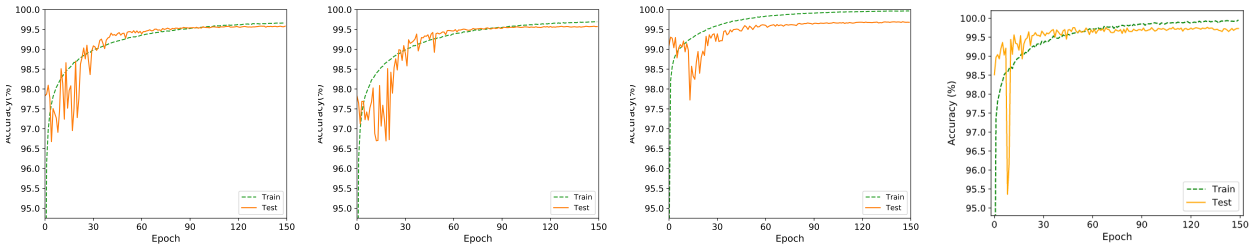
Figure 9 shows the change in training and test accuracy during training. It can be observed that for networks using max pooling, the test accuracy goes through oscillations in the early stage of training. On the other hand, the test accuracy of M_5 increases in a more stable manner. Table 4 shows the test accuracy of 30 networks between 50 epoch and 150



(a) C_1 (b) C_2 (c) C_3

Figure 8: Commonly used CNN structures with max pooling.

epoch of training. The average test accuracy of C_3 and M_5 is better than that of C_1 and C_2, which means using more convolution layers could result in better feature learning. Having more fully connected layers at the end did not help, as can be seen from the accuracy of C_1 and C_2. Between C_3 and M_5, M_5 achieves higher accuracy in general, and also can reach higher accuracy in the best case.



(a) C_1 (b) C_2 (c) C_3 (d) M_5

Figure 9: Train accuracy and test accuracy of C_1, C_2, C_3, and M_5 during training.

Table 4: Test accuracy of networks measured between 50 epoch and 150 epoch in training.

model	test accuracy			
	min	avg	max	best
C_1	99.3052 ± 0.0865	99.5293 ± 0.0105	99.6419 ± 0.0059	99.70
C_2	99.3594 ± 0.0442	99.5316 ± 0.0090	99.6337 ± 0.0051	99.68
C_3	99.4720 ± 0.04268	99.6448 ± 0.0078	99.7372 ± 0.0033	99.78
M_5	**99.5863 ± 0.0115**	**99.6835 ± 0.0074**	**99.7583 ± 0.0081**	**99.80**

Figure 10 shows the distribution plot of 30 networks for C_1, C_2, C_3, and M_5. For this graph, each network is trained for 150 epochs, and the network with the highest test accuracy is saved. It can be shown that M_5 achieves better test

accuracy than other networks in general. Table 5 shows the 95% confidence range of test accuracy for each network models.

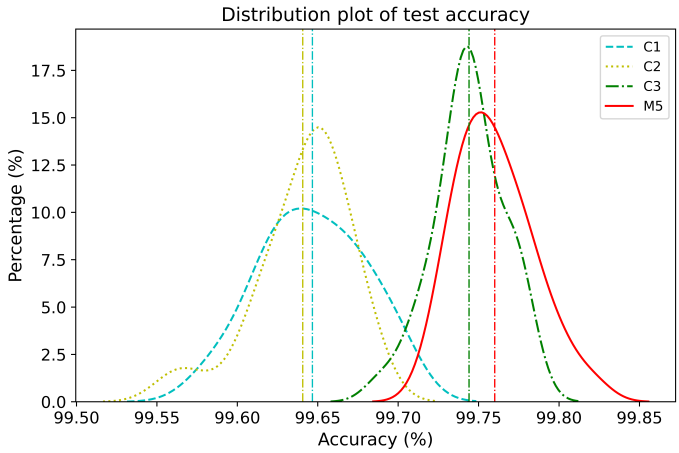


Figure 10: Distribution of test accuracy for networks with different architectures.

Table 5: 95% confidence range of test accuracy for networks with different architectures.

model	test accuracy
C_1	99.6466 ± 0.0121
C_2	99.6406 ± 0.0108
C_3	99.7440 ± 0.0080
M_5	$\mathbf{99.7600 \pm 0.0089}$

3.3 Impact of data augmentation

Data augmentation is a technique to increase the diversity of training data without actually collecting data and labeling them. It is an essential technique for supervised learning in which a large data set is required for the network model to achieve high performance [14, 15, 16, 17, 18]. When training the proposed network, we used two schemes for data generation: random rotation and random translation. There are many other schemes such as cropping, flipping, and resizing, and the best augmentation schemes depend on the data. In this section, we study whether data augmentation actually helps improving the network performance. We compared performance of four M_5 networks with different combinations of augmentation schemes applied. Figure 11 shows the distribution plot of 30 networks for four different augmentation strategies. It can be observed that data augmentation is helpful in general. For the MNIST data set, applying random rotation has slightly higher contribution than random translation, but both schemes are needed to achieve the best accuracy. Table 6 shows the 95% confidence range of test accuracy for the four augmentation strategies.

Table 6: 95% confidence range of test accuracy for networks trained with different augmentation schemes.

augmentation scheme		test accuracy
translation	rotation	
✗	✗	99.6783 ± 0.0086
✓	✗	99.7203 ± 0.0074
✗	✓	99.7327 ± 0.0077
✓	✓	$\mathbf{99.7600 \pm 0.0089}$

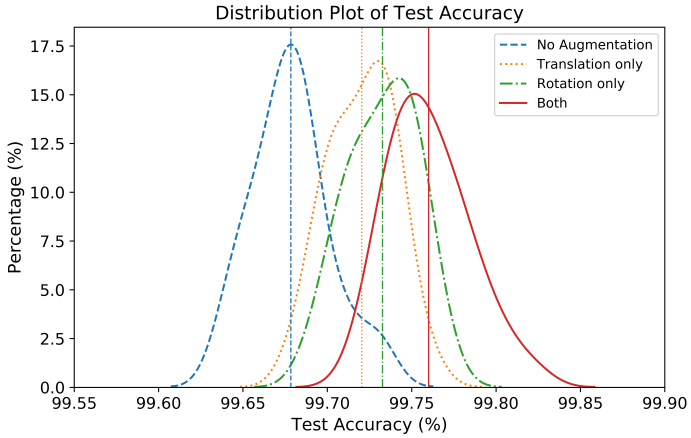


Figure 11: Distribution of test accuracy for networks trained with different augmentation schemes.

3.4 Impact of batch normalization

Batch normalization is a well known technique to improve performance of the network as well as stability and speed of training [19]. It has been reported that most neural network models benefit from using batch normalization [20, 21]. In this section we study the impact of batch normalization on the performance of the network model M_5. We compared three configurations: the first model uses no batch normalization at all, the second model uses batch normalization only at the fully connected layer, and the third model uses batch normalization at all layers. Figure 12 shows the distribution plot of 30 networks for each configuration, and Table 7 shows the 95% confidence range of test accuracy for each configuration. It is evident that using batch normalization helps improve the performance of neural network models. The best performance is achieved when batch normalization is used at each convolution and fully connected layer.

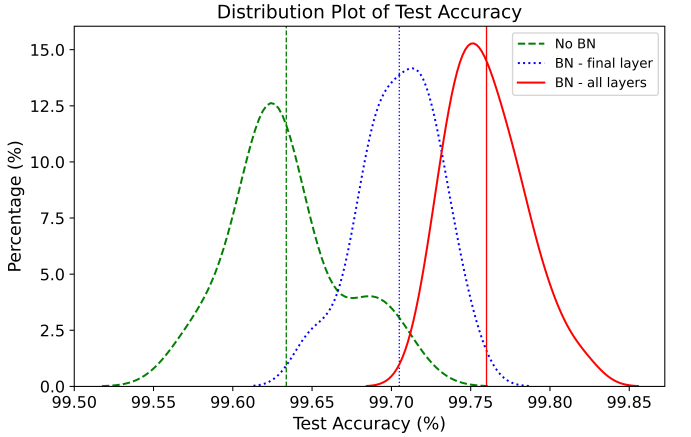


Figure 12: Distribution of test accuracy for networks trained with different batch normalization schemes.

Table 7: 95% confidence range of test accuracy for networks trained with different batch normalization schemes.

configuration	test accuracy
no batch normalization	99.6337 ± 0.0131
batch normalization at the final layer	99.7050 ± 0.0092
batch normalization at all layers	$\mathbf{99.7600 \pm 0.0089}$

4 Conclusion

The MNIST handwritten digit data set is often used as an entry-level data set for training and testing neural networks. While achieving 99% accuracy on the test set is rather easy, correctly classifying the last 1% of the images is challenging. People have tried many different network models and techniques to increase test accuracy, and the best accuracy reported reaches approximately 99.8%. In this paper we showed that a simple CNN model with batch normalization and data augmentation could reach the best accuracy. Using an ensemble of homogeneous and heterogeneous network models could boost the performance, up to 99.91% test accuracy which is one of the state-of-the-art performance. Studies with various different configurations show that the high performance is not achieved by a single technique or model architecture, but is contributed by multiple techniques such as batch normalization, data augmentation, and ensemble methods.

References

[1] Y. Lecun, C. Cortes, and C. J. Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* (2010)

[2] A. Byerly, T. Kalganova, and I. Dear, "A branching and merging convolutional network with homogeneous filter capsules," *arXiv preprint arXiv:2001.09136* (2020)

[3] K. Kowsari, M. Heidarysafa, D. E. Brown, K. J. Meimandi, and L. E. Barnes, "RMDL: Random multimodel deep learning for classification," *in Proc. 2nd Int. Conf. Inf. Syst. Data Mining (ICISDM)*, pp. 19-28 (2018).

[4] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of Neural Networks using DropConnect," *in Proc. International Conference on Machine Learning (ICML)* (2013)

[5] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012)

[6] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *in Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3859-3869 (2017)

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N, Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *in Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[8] P. Izmilov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, "Averaging Weights Leads to Wider Optima and Better Generalization," *in Proc. Conference on Uncertainty in Artificial Intelligence (UAI)* (2018)

[9] L. Breiman, "Bagging Predictors," *in Machine Learning*, 24(2):123-140 (1996)

[10] Y. Freund and R. E. Shapire, "Discussion of additive logistic regression: A statistical view of boosting," *in Annals of Statistics*, 28:337-374 (2000)

[11] J. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, 1189–1232 (2001)

[12] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T. Liu, "LightGBM: A highly efficient gradient boosting decision tree," *in Advances in Neural Information Processing Systems*, 3149-3157 (2017)

[13] M. Ranzato, F. J. Huang, Y. L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," *in Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1-8 (2007)

[14] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning augmentation policies from data," *in Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 113-123 (2019)

[15] E. D. Cubuk, B. Zoph, J. Shlens, Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," *arXiv preprint arXiv:1909.13719* (2019)

[16] Z. Zhong, L. Zheng, G. Kang, S. Li, Y. Yang, "Random erasing data augmentation," *arXiv preprint arXiv:1708.04896* (2017)

[17] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim, "Fast autoaugment," *arXiv preprint arXiv:1905.00397* (2019)

[18] P. Bachman, R. Hjelm, and W. Buchwalter, "Learning representations by maximizing mutual information across views," *across views. arXiv:1906.00910* (2019)

[19] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167* (2015)

[20] J. Bjorck, C. Gomes, B. Selman, K. Q. Weinberger, “Understanding Batch Normalization," *in Advances in Neural Information Processing Systems* (2018)

[21] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?,” *in Advances in Neural Information Processing Systems* (2018)