Handwritten Digit Recognition on Modified MNIST

Alexa Hernandez
McGill University
260743067

Jacob Dube
McGill University
260741967

Saifullah ElSayed
McGill University
260733168

***Abstract*—Given an image that may contain various digits that differ in size, this paper attempts to use committees of convolutional neural networks to classify the digit occupying the most space. We focus on three different methods in constructing the committee; namely, average committee, majority vote committee, and median committee. We explore how the number of CNNs used to form each type of committee affects performance. To do so, we test and compare the performance of each type of committee using all possible subsets of 5 individual CNNs. We find that, in general, the performance of each type of committee increases with the number of CNNs used. Moreover, we find that the vast majority of tested committees outperform even our best-performing individual CNN.**

I. Introduction

Handwritten digit recognition is a task of great interest in both academia and industry due to its widespread applicability. The uses of handwritten digit recognition include, but are not limited to, signature verification, postal-code recognition for mail sorting, online check processing for banks, and analysis of scanned documents.

The MNIST dataset, which contains images of isolated handwritten digits from 0-9, is the most widely used dataset for training and testing classifiers for handwritten digit recognition [1]. We perform our experiments on a modified version of the MNIST dataset. The modified MNIST dataset consists of 50,000 images, each of which may contain multiple digits that differ in size. The task is to identify the digit that occupies the most physical space within a given image.

For this task, we implement committees of convolutional neural networks (CNNs). We employ three different methods in constructing the committees; namely, average committee, majority vote committee, and median committee. Throughout the process we explore the effectiveness of two different regularization approaches on the performance of our base CNN. The two approaches compared are dropout and a combination of dropout and batch normalization. We also compare the performance of individual CNNs to the performance of committees of CNNs. Lastly, we investigate how the performance of each type of committee is affected by the number of CNNs used to build the committee. To do so, we test and compare the performance of each type of committee on all possible subsets of 5 CNNs. We now summarize our key findings.

First off, we find that a combination of dropout and batch normalization is a more effective regularization technique than just dropout for our base CNN. Next, we find that the overwhelming majority of tested committees outperform even our best-performing individual CNN. Lastly, we find that, in general, increasing the number of CNNs used to build each type of committee improves performance.

II. Related Work

Handwritten digit recognition is a relatively well-studied task in computer vision. Past researchers have trained, tested, and optimized several different classifiers in attempts to achieve the lowest test error rate on the MNIST digit recognition benchmark.

In 1998, the creators of the MNIST database compared the performance of the following models: Linear Classifiers, K-Nearest Neighbors, Support Vector Machines (SVMs), Neural Networks, and CNNs [2]. While SVMs performed surprisingly well, CNNs proved to be especially well-suited for handwritten digit recognition, yielding the lowest error rate of 0.7%. The superior performance of CNNs in handwritten digit recognition is further confirmed by [3], [4], [6].

The performance of CNNs on MNIST has since been improved by employing elastic transformations to expand the training set [3], and pre-training each hidden CNN layer individually in an unsupervised manner [9]. The optimal performance on MNIST to date was achieved using an average committee of CNNs whose errors on various parts of the validation set differed as much as possible [4]. This was achieved by training 35 identical CNNs on input pre-processed/normalized in different ways. The resulting committee yielded a record test error rate 0.23%.

The consistently superior performance of CNNs is not unique to task of recognizing handwritten digits. CNNs have produced state-of-the-art results on a plethora of image analysis tasks such as object recognition [5], handwritten character recognition [6], and traffic sign recognition [7].

III. Data and Setup

A. The Modified MNIST Dataset

We performed experiments on a modified version of the MNIST dataset. This modified dataset consists of 40,000 training images and 10,000 test images. Each image may contain multiple digits that differ in size. Each digit within a given image may have undergone a transformation (rotation, translation, etc). Moreover, noise was injected into the background of each image. Images are represented as 64 × 64 matrices of pixel intensity values; that is, the images are in grey-scale. Every training image has an associated label indicating the digit that occupies the most physical space in the image. Five example training images from the modified MNIST dataset are shown in Figure 1. The task is to identify the digit occupying the most space within a given image.

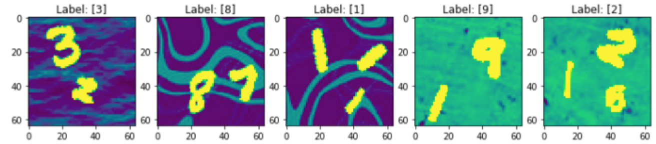

Figure. 1. Five example training images sampled from the modified MNIST dataset and their associated labels. Note that while the above images are presented in color for improved readability, the actual images are in grey-scale.

B. Preprocessing

CNNs have attained record performance with minimal preprocessing [5]. As such, we decided not to perform any preprocessing. Thus, we trained our models on the raw pixel intensity values.

IV. Proposed Approach

A committee system produces predictions by combining the outputs of a collection of individuals classifiers that have been trained on the same training set. The motivation is that each individual classifier may offer complementary information which can be exploited by the committee to achieve better results [10]. We decided to implement committees of CNNs as they have outperformed all other techniques in both handwritten digit recognition [6], [4], and traffic sign recognition [7]. Although [4] constructs a committee using 35 CNNs, given the lengthy training times that CNNs require and our limited hardware (gtx 1060 6gb gpu), we restrict ourselves to using 5 CNNs.

Our proposed approach is as follows: first we design and train a base CNN. Next we compare the affect of 2 different regularization approaches on the base CNN with the intention of gaining insight on how to improve the base CNN's performance. Using the regularization approach that produced superior results, we proceed to construct 5 distinct CNNs to be used in a committee system. We then implement three different committee systems; namely, average committee, majority vote committee, and median vote committee. Lastly, we test and compare the performance of each type of committee on all possible subsets of the 5 CNNs. The following sections present a detailed description of each stage of the proposed approach.

A. The Base CNN

In this section we describe the base CNN we designed.

Compared to other models, CNNs have an extensive list of model hyperparameters. Choices must be made concerning the activation function, the number of convolutional layers, the convolutional kernel size, the number of fully connected layers, the number of maximum pooling layers, whether or not to use dropout and/or batch normalization, etc. Accordingly, we decided to first design and train a base CNN. The architecture of the base CNN is as follows: there are 5 convolutional layers, each of which are followed by a maximum pooling layer. The exact details of the convolutional and maximum pooling layers are described below. Furthermore, we use dropout after some of the maximum pooling layers to help combat overfitting. Dropout consists of setting the output of each hidden neuron to 0 with a pre-specified probability [14].

1) Convolutional Layer: Each convolutional layer has a kernel size of 3 × 3. Moreover, each input is padded such that the output has the same length as the original input. All convolutional layers are followed by the Rectified Linear Unit (ReLU) activation function, as CNNs with ReLUs exhibit substantially decreased training times compared with their equivalents with tanh or sigmoid units [5]. Lastly, the 5 convolutional layers have filter sizes 32, 64, 128, 256, and 256, respectively. The filter size indicates the dimensionality of the output space.

2) Maximum Pooling Layer: The use of maximum pooling layers reduce dimensionality thereby leading to faster convergence. They can also improve generalization by reducing overfitting [11]. The output of a maximum pooling layer is the neuron with the maximum activation over a specified non-overlapping rectangle region termed the pool. Each maximum pooling layer has a pool size of 2×2.

The last two layers are fully connected layers. The first fully connected layer has ReLU activation function and a dropout rate of 0.5. The second fully connected layer has a softmax activation function which produces a number between 0 and 1, inclusive. See Figure 2 for a visual summary of the base CNN architecture.

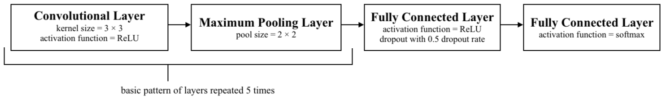
Convolutional Layer
kernel size = 3 × 3
activation function = ReLU
Maximum Pooling Layer
pool size = 2 × 2
Fully Connected Layer
activation function = ReLU
dropout with 0.5 dropout rate
Fully Connected Layer
activation function = softmax
basic pattern of layers repeated 5 times

Figure. 2. Summary of the base CNN architecture. Each convolutional layers has a filter size of 32, 64, 128, 256, and 256, respectively. Dropout is applied after some of the maximum pooling layers.

B. Training and Optimizing the Base CNN

Unfortunately, our limited computing power rendered the use of k-fold cross validation infeasible. Instead, each time we trained a model we randomly selected 15% of the 40,000 training images to form the validation set. During training, the base CNN was optimized with respect to the cross entropy loss function using Adam optimizer [12], [13]. We initialized the Adam optimizer hypermaraters as follows: learning rate = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = None, decay = 0.0. The base CNN was trained for 400 epochs using a batch size of 512. We retrained the base CNN various times, each time manually adjusting the placement of dropout and the corresponding dropout rates in attempt to improve performance. The base CNN with the dropout placement and corresponding dropout rates that yielded the best results was then used as the base CNN. Please refer to Appendix A Figure 1 for the specific architecture of the final base CNN implemented using Keras [13].

C. The Five Individual CNNs

In this section we describe our approach in constructing and selecting 5 CNNs to be used in a committee system.

First we tested the effect of two different regularization approaches on the performance of the base CNN with the intention of gaining insight on how to further improve the base CNN. More specifically, we compared the performance of the base CNN using dropout versus a combination of both batch normalization and dropout. Batch normalization normalizes all input across a mini-batch region [15].

We then proceeded in constructing 5 distinct CNNs using the regularization approach that achieved superior performance in the foregoing experiment. Each of the 5 CNNs maintained the basic architecture presented in Figure 2. However, for each model we varied the number of repetitions of the basic pattern of layers of Figure 2 and the filter size of each convolutional layer. Moreover, we trained each model using the same training methodology described for the base CNN. However, we optimized the hyperparameters of the optimal regularization approach. Please refer to Appendix B Figures 1 and 2 for the specific architecture of the resulting 5 CNNs implemented using Keras [13].

D. Constructing a Committee

After having trained 5 CNNs that improved upon the base CNN, we sought to achieve increased accuracy through the use of committee systems. The three different methods we used to build the committee systems are [10]:

1) Average Committee: compute the average class probabilities from the *n* classifiers and output the class with the highest average class probability.
2) Majority Vote Committee: outputs the class with the most votes from the *n* classifiers. If two classes have the same number of votes, we take the class that corresponds to the smaller digit.
3) Median Committee: determines the median of the class probabilities from the *n* classifiers and outputs the class with the highest median class probability.

Here *n* is the number of CNNs used in the committee. We tested and compared the performance of each type of committee on all possible subsets of the 5 CNNs. The optimal committee, as per the validation accuracy, was then selected as our final model. We then re-trained our final model using all 40,000 training images prior to submitting our predictions to Kaggle.

V. Results

For the results, we retrain and test the base CNN and the five individual CNNs on the same validation and training sets such that we randomly select 15% of the 40,000 training images to form the validation set.

A. Performance of the Base CNN using Different Regularization Approaches

In table 1 we report our results of the comparison of the effect of different regularization techniques on the base CNN's performance. The exact architecture of the base CNN as implemented in keras can be found in Appendix A Figure 1. Only dropout is applied to the base CNN. CNN1 has the exact same architecture as the base CNN, except a combination of batch normalization and dropout is applied instead of just dropout.

	Training Accuracy	Validation Accuracy	Training Time (s)
Base CNN	0.9998	0.9147	2885± 20
CNN1	0.9999	0.9555	4117± 20

Table. 1. Training accuracy, validation accuracy, and training time of the base CNN and CNN1. Only dropout is applied to the base CNN, while a combination of dropout and batch normalization is applied to CNN1.

B. Performance of 5 Individual CNNs

In table 2 we report the training accuracy, validation accuracy, and training time of the 5 CNNs, denoted CNN1, CNN2, CNN3, CNN4, and CNN5. The exact architecture of the 5 CNNs as implemented in keras can be found in Appendix B Figures 1-2.

	Training Accuracy	Validation Accuracy	Training Time (s)
CNN1	0.9999	**0.9555**	4117±20
CNN2	0.9999	0.9500	3295±20
CNN3	0.9991	0.9538	3298±20
CNN4	0.9999	0.9528	3700±20
CNN5	0.9998	0.9507	3296±20

Table. 2. Training accuracy, validation accuracy, and training time of the 5 individual CNNs.

C. Performance of the Committee Systems

We observed the effect of the number of CNNs used to form each type of committee system. To do so, we compared the performance of each type of committee system using all possible subsets of the 5 CNNs. In Table 2 we report the highest validation accuracy achieved by each type of committee using 1, 2, 3, 4, and 5 CNNs. In this case, the row with "Number of CNNs" = 1 simply reports the validation accuracy obtained by CNN1 (the best-performing of the 5 CNNs), and is included to facilitate the comparison of the performance of the individual CNNs and the committees of CNNs.

Number of CNNs	Average Committee	Majority Voting Committee	Median Committee
1	0.9555	0.9555	0.9555
2	0.9613	0.9550	0.9613
3	0.9623	0.9612	0.9628
4	0.9632	0.9603	0.9625
5	**0.9633**	0.9625	0.9630

Table. 3. Highest validation accuracy achieved by each type of committee using 1, 2, 3, 4, and 5 CNNs.

The highest accuracy is achieved by the average committee that uses all 5 CNNs. Accordingly, we decided to use an average committee of the 5 CNNs as our final model. We re-trained the 5 CNNs using all 40,000 training images and used the re-trained 5 CNNs to form an average committee. This model attained a test accuracy of 0.9670 on 30% of the test images on Kaggle.

VI. Discussion and Conclusion

This work attempts to classify the handwritten digit occupying the most space within a given image using a committee of CNNs. In the process we explore the effect of different regularization approaches on the performance of our base CNN, the performance of individual CNNs compared to the performance of committees, and the effect of the number of CNNs used on the performance of three different types of committees.

First, we explore the effectiveness of 2 different regularization approaches in reducing the likelihood of overfitting and thereby increasing performance. The two approaches being compared are dropout and a combination of dropout and batch normalization. As seen in Table 1, the validation accuracy of the base CNN, to which only dropout is applied, is significantly lower than its training accuracy. We thus presume that, despite the use of dropout, the base CNN is likely to suffer from overfitting. While there still exists a discrepancy between the validation and training accuracy of CNN1, to which a combination of batch normalization and dropout is applied, it is notably smaller. Hence, we conclude that CNN1 is less likely to overfit. As such, we conclude that a combination of batch normalization and dropout is more effective in preventing overfitting on our base CNN. Accordingly, a combination of batch normalization and dropout was applied when constructing CNN2, CNN3, CNN4, and CNN5.

Next, we compare the performance of the individual CNNs to the committees of CNNs. The best-performing individual CNN, as indicated in Table 2, is CNN1 which yields a training accuracy of 0.9555. All tested committees, except for the majority voting committee with 2 CNNs, outperform CNN1. Please refer to Table 3 for more details. Moreover, the best-performing average committee, majority voting committee, and median committee achieve validation accuracies that are 0.0078, 0.0070, and 0.0075 higher than that of CNN1, respectively. These results are consistent with [10], whose results indicate that individual CNNs' handwritten digit recognition rates can be considerably improved by using committees.

We also investigate how the performance of all three types of committees is impacted by the number of CNNs used. Table 3 reveals that, in general, the accuracy of all three types of committees increases with the number of CNNs used. Figure 3 presents a visualization of this general trend. We presume that this increase in performance occurs because each individual CNN offers complementary information which can be harnessed by the committee system to produce more accurate predictions. Thus, including more CNNs provides the committee with more complementary information to be exploited. We recognize that including more than 5 individual CNNs in the committees probably would have yielded tangible improvements in performance. However, as seen in Table 2, each individual CNN took roughly one hour to train on our hardware. Thus, given better hardware we could have included more CNNs in the committees in hopes of improving accuracy.

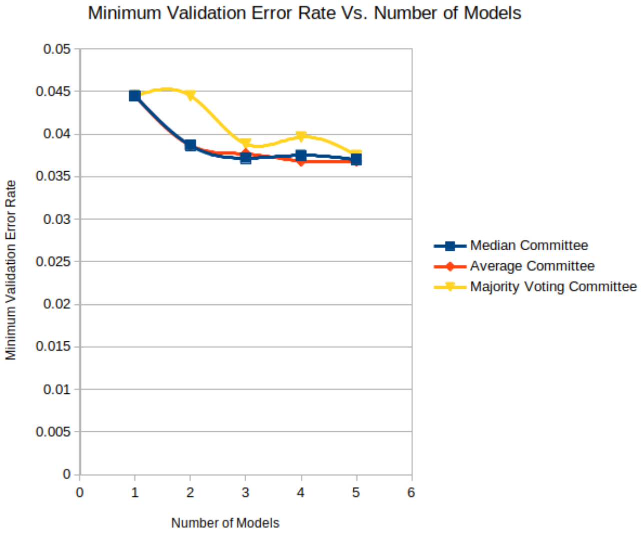

Figure. 3. This figure plots the minimum validation error rate obtained by each type of committee using 1, 2, 3, 4, and 5 CNNs.

Lastly, we compare the performance of each of the three types of committees. We find that while the average committee performs marginally better, all three types of committees exhibit fairly similar performance. Please refer to Table 3 for more details. In the end, the best-performing committee is the average committee with all 5 CNNs and is thus selected as our final model. This model achieved an accuracy of 0.9670 on 30% of the test images on Kaggle.

For future work, we would focus more on how to construct an optimal collection of CNNs to be used in a committee. That is, a group of CNNs such that the committee can harness as much complementary information as possible from each individual CNN. Past research suggests that a committee is most effective when it utilizes a collection of classifiers whose errors made on the training set differ as much as possible [6]. Furthermore, it is suggested that for isolated handwriting digit recognition on MNIST, this can be achieved by training identical classifiers on training data that has been pre-processed/normalization in different ways [6]. [4]. We would like to further investigate the foregoing methodology and observe whether or not it yields the same state-of-the-art results on the modified MNIST dataset as on the MNIST dataset.

VII. Statement of Contribution of Work

All members of the group contributed equally. We all researched past work related to handwritten digit recognition and brainstormed an appropriate approach. Jacob's main task was implementation, training, and testing. Alexa's task was writing the report. Saifullah's helped equally with both of the foregoing tasks. All parties were in constant communication to ensure cohesion throughout.

References

[1] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[2] 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

[3] P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition*, volume 2, pages 958-962, 2003.

[4] D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. Arxiv preprint arXiv:1202.2745, 2012.

[5] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

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

[7] D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. A committee of neural networks for traffic sign classification. In *International Joint Conference on Neural Networks*, pages 1918-1921, 2011.

[8] D. Strigl, K. Kofler, and S. Podlipnig. Performance and scalability of gpu-based convolutional neural networks. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference* on, 0:317-324, 2010.

[9] Y.-l. Boureau, Y. L. Cun, et al. Sparse feature learning for deep belief networks. In *Advances in neural information processing systems*, pages 1185-1192, 2008.

[10] U. Meier, D. C. Ciresan, L. M. Gambardella, and J. Schmidhuber, Better digit recognition with a committee of simple neural nets, in *International Conference on Document Analysis and Recognition*, pages 1135-1139, 2011.

[11] Dominik Scherer, Adreas Muller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, 2010.

[12] Kingma, Diederik P and Ba, Jimmy Lei. Adam: A method for stochastic optimization. *arXiv preprint arXiv*:1412.6980, 2014

[13] Chollet, F. (2015) keras, GitHub. https://github.com/fchollet/keras

[14] Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. effeout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929-1958, January 2014.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

APPENDIX

A. Base CNN Architecture

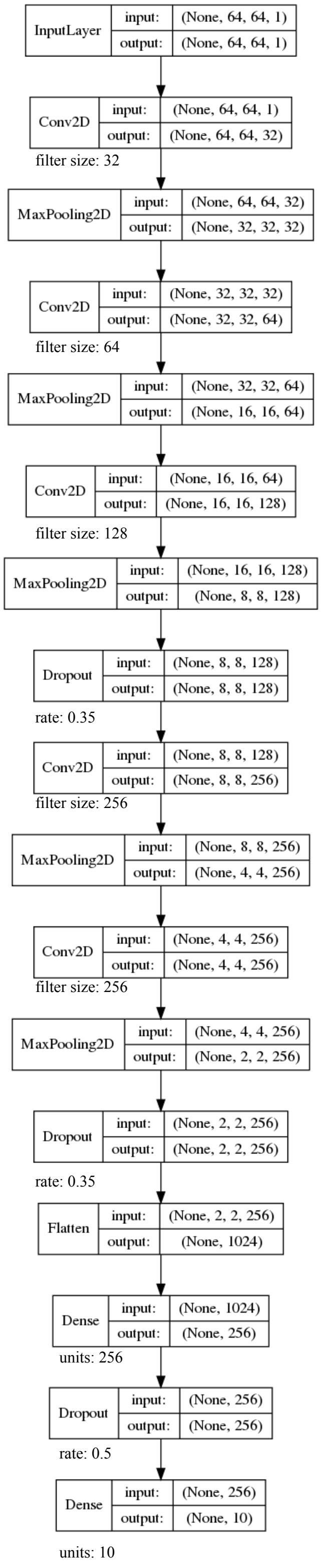

Figure A.1: a summary representation of the fine-tune base CNN's implementation using Keras. Conv2D, MaxPooling2D, Dropout, etc. correspond to Keras methods.

B. Architecture of the Final Five CNNs

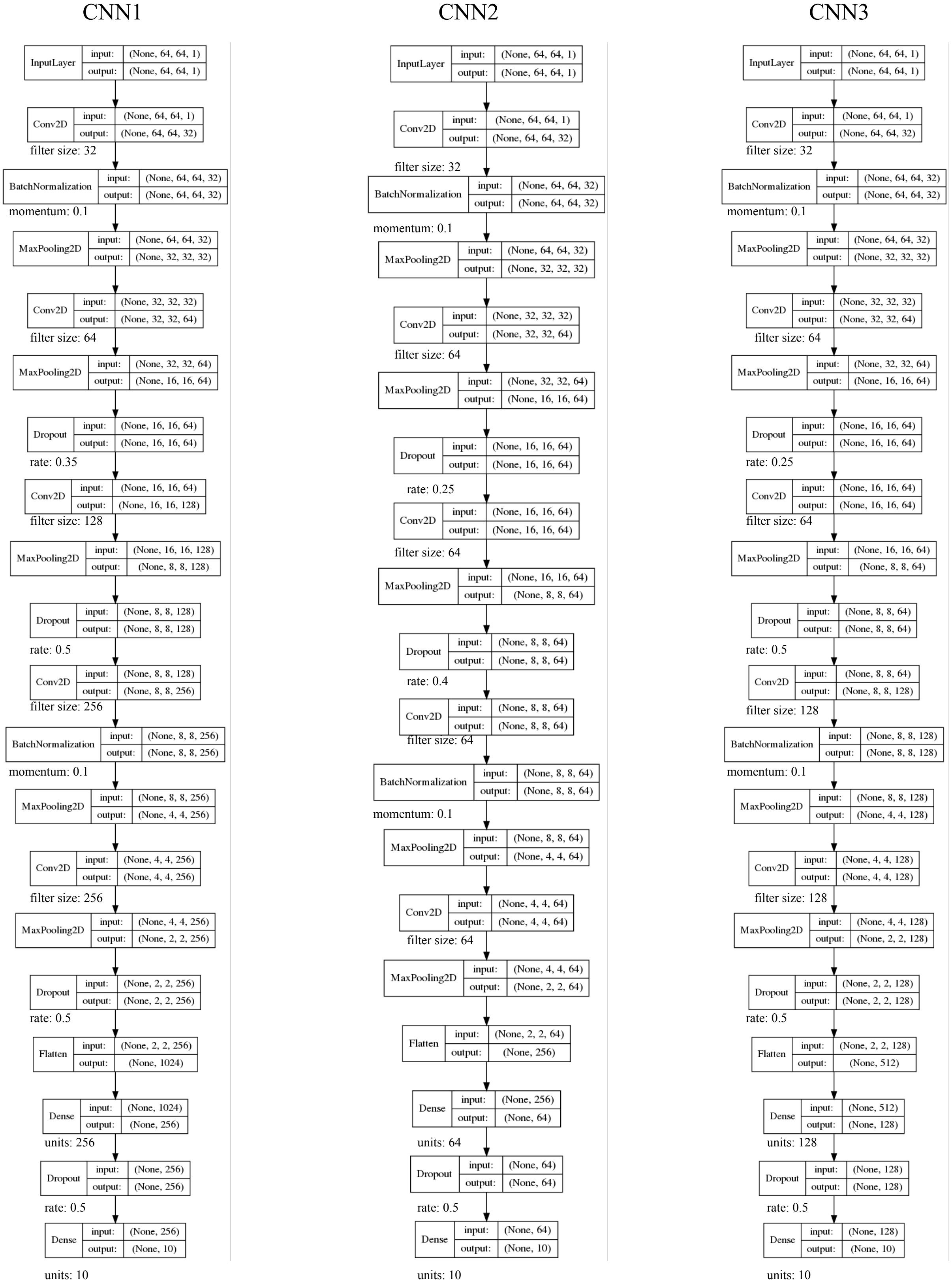
CNN1
InputLayer input: (None, 64, 64, 1) output: (None, 64, 64, 1)
Conv2D input: (None, 64, 64, 1) output: (None, 64, 64, 32)
filter size: 32
BatchNormalization input: (None, 64, 64, 32) output: (None, 64, 64, 32)
momentum: 0.1
MaxPooling2D input: (None, 64, 64, 32) output: (None, 32, 32, 32)
Conv2D input: (None, 32, 32, 32) output: (None, 32, 32, 64)
filter size: 64
MaxPooling2D input: (None, 32, 32, 64) output: (None, 16, 16, 64)
Dropout input: (None, 16, 16, 64) output: (None, 16, 16, 64)
rate: 0.35
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 128)
filter size: 128
MaxPooling2D input: (None, 16, 16, 128) output: (None, 8, 8, 128)
Dropout input: (None, 8, 8, 128) output: (None, 8, 8, 128)
rate: 0.5
Conv2D input: (None, 8, 8, 128) output: (None, 8, 8, 256)
filter size: 256
BatchNormalization input: (None, 8, 8, 256) output: (None, 8, 8, 256)
momentum: 0.1
MaxPooling2D input: (None, 8, 8, 256) output: (None, 4, 4, 256)
Conv2D input: (None, 4, 4, 256) output: (None, 4, 4, 256)
filter size: 256
MaxPooling2D input: (None, 4, 4, 256) output: (None, 2, 2, 256)
Dropout input: (None, 2, 2, 256) output: (None, 2, 2, 256)
rate: 0.5
Flatten input: (None, 2, 2, 256) output: (None, 1024)
Dense input: (None, 1024) output: (None, 256)
units: 256
Dropout input: (None, 256) output: (None, 256)
rate: 0.5
Dense input: (None, 256) output: (None, 10)
units: 10
CNN2
InputLayer input: (None, 64, 64, 1) output: (None, 64, 64, 1)
Conv2D input: (None, 64, 64, 1) output: (None, 64, 64, 32)
filter size: 32
BatchNormalization input: (None, 64, 64, 32) output: (None, 64, 64, 32)
momentum: 0.1
MaxPooling2D input: (None, 64, 64, 32) output: (None, 32, 32, 32)
Conv2D input: (None, 32, 32, 32) output: (None, 32, 32, 64)
filter size: 64
MaxPooling2D input: (None, 32, 32, 64) output: (None, 16, 16, 64)
Dropout input: (None, 16, 16, 64) output: (None, 16, 16, 64)
rate: 0.25
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 64)
filter size: 64
MaxPooling2D input: (None, 16, 16, 64) output: (None, 8, 8, 64)
Dropout input: (None, 8, 8, 64) output: (None, 8, 8, 64)
rate: 0.4
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 64)
filter size: 64
BatchNormalization input: (None, 8, 8, 64) output: (None, 8, 8, 64)
momentum: 0.1
MaxPooling2D input: (None, 8, 8, 64) output: (None, 4, 4, 64)
Conv2D input: (None, 4, 4, 64) output: (None, 4, 4, 64)
filter size: 64
MaxPooling2D input: (None, 4, 4, 64) output: (None, 2, 2, 64)
Flatten input: (None, 2, 2, 64) output: (None, 256)
Dense input: (None, 256) output: (None, 64)
units: 64
Dropout input: (None, 64) output: (None, 64)
rate: 0.5
Dense input: (None, 64) output: (None, 10)
units: 10
CNN3
InputLayer input: (None, 64, 64, 1) output: (None, 64, 64, 1)
Conv2D input: (None, 64, 64, 1) output: (None, 64, 64, 32)
filter size: 32
BatchNormalization input: (None, 64, 64, 32) output: (None, 64, 64, 32)
momentum: 0.1
MaxPooling2D input: (None, 64, 64, 32) output: (None, 32, 32, 32)
Conv2D input: (None, 32, 32, 32) output: (None, 32, 32, 64)
filter size: 64
MaxPooling2D input: (None, 32, 32, 64) output: (None, 16, 16, 64)
Dropout input: (None, 16, 16, 64) output: (None, 16, 16, 64)
rate: 0.25
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 64)
filter size: 64
MaxPooling2D input: (None, 16, 16, 64) output: (None, 8, 8, 64)
Dropout input: (None, 8, 8, 64) output: (None, 8, 8, 64)
rate: 0.5
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 128)
filter size: 128
BatchNormalization input: (None, 8, 8, 128) output: (None, 8, 8, 128)
momentum: 0.1
MaxPooling2D input: (None, 8, 8, 128) output: (None, 4, 4, 128)
Conv2D input: (None, 4, 4, 128) output: (None, 4, 4, 128)
filter size: 128
MaxPooling2D input: (None, 4, 4, 128) output: (None, 2, 2, 128)
Dropout input: (None, 2, 2, 128) output: (None, 2, 2, 128)
rate: 0.5
Flatten input: (None, 2, 2, 128) output: (None, 512)
Dense input: (None, 512) output: (None, 128)
units: 128
Dropout input: (None, 128) output: (None, 128)
rate: 0.5
Dense input: (None, 128) output: (None, 10)
units: 10

Figure B.1: a summary representation of the implementation of CNN1, CNN2, and CNN3 using Keras. Conv2D, MaxPooling2D, Dropout, etc. correspond to Keras methods.

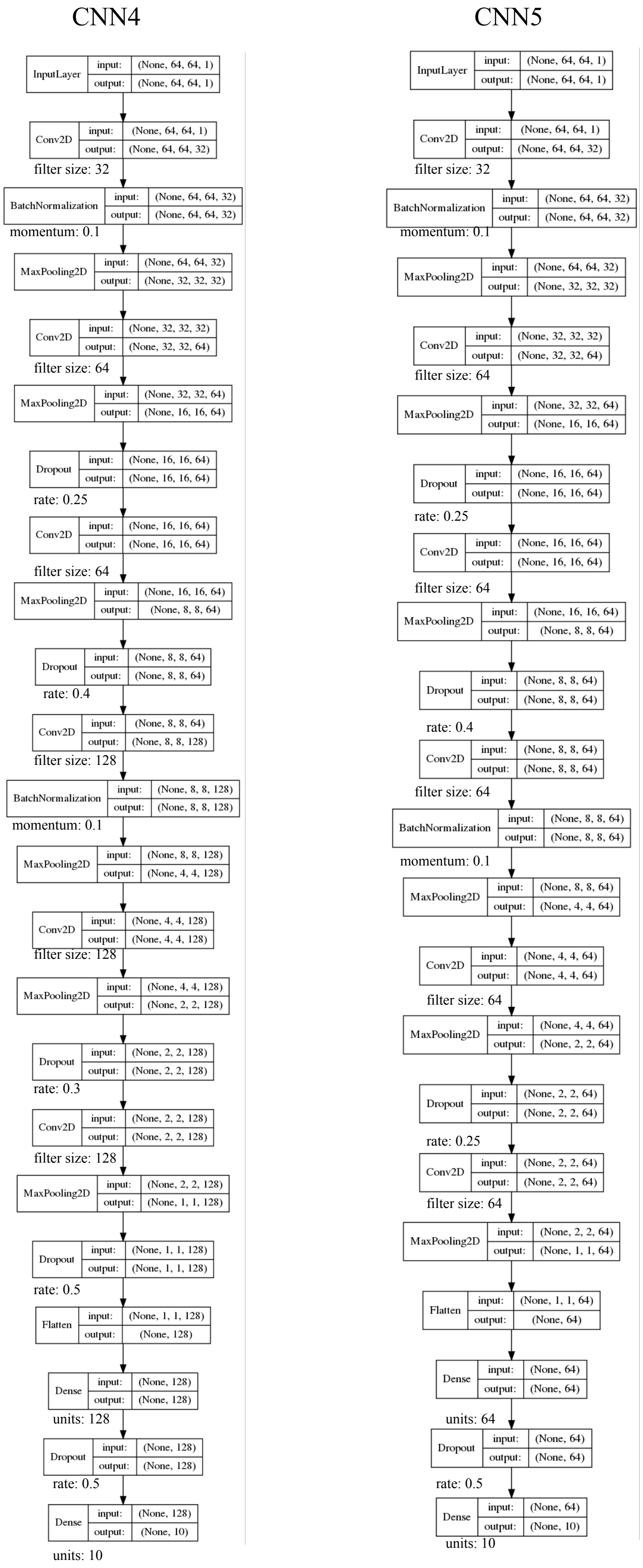
CNN4
InputLayer input: (None, 64, 64, 1) output: (None, 64, 64, 1)
Conv2D input: (None, 64, 64, 1) output: (None, 64, 64, 32)
filter size: 32
BatchNormalization input: (None, 64, 64, 32) output: (None, 64, 64, 32)
momentum: 0.1
MaxPooling2D input: (None, 64, 64, 32) output: (None, 32, 32, 32)
Conv2D input: (None, 32, 32, 32) output: (None, 32, 32, 64)
filter size: 64
MaxPooling2D input: (None, 32, 32, 64) output: (None, 16, 16, 64)
Dropout input: (None, 16, 16, 64) output: (None, 16, 16, 64)
rate: 0.25
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 64)
filter size: 64
MaxPooling2D input: (None, 16, 16, 64) output: (None, 8, 8, 64)
Dropout input: (None, 8, 8, 64) output: (None, 8, 8, 64)
rate: 0.4
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 128)
filter size: 128
BatchNormalization input: (None, 8, 8, 128) output: (None, 8, 8, 128)
momentum: 0.1
MaxPooling2D input: (None, 8, 8, 128) output: (None, 4, 4, 128)
Conv2D input: (None, 4, 4, 128) output: (None, 4, 4, 128)
filter size: 128
MaxPooling2D input: (None, 4, 4, 128) output: (None, 2, 2, 128)
Dropout input: (None, 2, 2, 128) output: (None, 2, 2, 128)
rate: 0.3
Conv2D input: (None, 2, 2, 128) output: (None, 2, 2, 128)
filter size: 128
MaxPooling2D input: (None, 2, 2, 128) output: (None, 1, 1, 128)
Dropout input: (None, 1, 1, 128) output: (None, 1, 1, 128)
rate: 0.5
Flatten input: (None, 1, 1, 128) output: (None, 128)
Dense input: (None, 128) output: (None, 128)
units: 128
Dropout input: (None, 128) output: (None, 128)
rate: 0.5
Dense input: (None, 128) output: (None, 10)
units: 10
CNN5
InputLayer input: (None, 64, 64, 1) output: (None, 64, 64, 1)
Conv2D input: (None, 64, 64, 1) output: (None, 64, 64, 32)
filter size: 32
BatchNormalization input: (None, 64, 64, 32) output: (None, 64, 64, 32)
momentum: 0.1
MaxPooling2D input: (None, 64, 64, 32) output: (None, 32, 32, 32)
Conv2D input: (None, 32, 32, 32) output: (None, 32, 32, 64)
filter size: 64
MaxPooling2D input: (None, 32, 32, 64) output: (None, 16, 16, 64)
Dropout input: (None, 16, 16, 64) output: (None, 16, 16, 64)
rate: 0.25
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 64)
filter size: 64
MaxPooling2D input: (None, 16, 16, 64) output: (None, 8, 8, 64)
Dropout input: (None, 8, 8, 64) output: (None, 8, 8, 64)
rate: 0.4
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 64)
filter size: 64
BatchNormalization input: (None, 8, 8, 64) output: (None, 8, 8, 64)
momentum: 0.1
MaxPooling2D input: (None, 8, 8, 64) output: (None, 4, 4, 64)
Conv2D input: (None, 4, 4, 64) output: (None, 4, 4, 64)
filter size: 64
MaxPooling2D input: (None, 4, 4, 64) output: (None, 2, 2, 64)
Dropout input: (None, 2, 2, 64) output: (None, 2, 2, 64)
rate: 0.25
Conv2D input: (None, 2, 2, 64) output: (None, 2, 2, 64)
filter size: 64
MaxPooling2D input: (None, 2, 2, 64) output: (None, 1, 1, 64)
Flatten input: (None, 1, 1, 64) output: (None, 64)
Dense input: (None, 64) output: (None, 64)
units: 64
Dropout input: (None, 64) output: (None, 64)
rate: 0.5
Dense input: (None, 64) output: (None, 10)
units: 10

Figure B.2: a summary representation of the implementation of CNN4 and CNN4 using Keras. Conv2D, MaxPooling2D, Dropout, etc. correspond to Keras methods.