**A STATE-OF-THE-ART METHOD FOR IMAGE CLASSIFICATION ON MNIST DATASET**

**-------------------------------------------------------------------------------------------------**

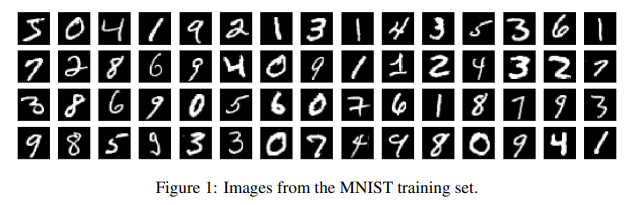
1. **INTRODUCTION**
   1. **WHAT IS THIS PROJECT ABOUT?**

In this report, we would present a state-of-the-art method for image classification on MNIST dataset and compare the best results with benchmark results of this dataset.

The programming language is Python 3. We use GPU device for strong computing power and boost the computing speed.

* 1. **MNIST DATASET**

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.



The data is stored in a very simple file format designed for storing vectors and multidimensional matrices. General info on this format is given at the end of this page, but you don't need to read that to use the data files.

All the integers in the files are stored in the MSB first (high endian) format used by most non-Intel processors. Users of Intel processors and other low-endian machines must flip the bytes of the header.

There are 4 files:

1. train-images-idx3-ubyte: training set images
2. train-labels-idx1-ubyte: training set labels
3. t10k-images-idx3-ubyte: test set images
4. t10k-labels-idx1-ubyte: test set labels

The training set contains 60000 examples, and the test set 10000 examples.

The first 5000 examples of the test set are taken from the original NIST training set. The last 5000 are taken from the original NIST test set. The first 5000 are cleaner and easier than the last 5000.

* 1. **TASK**

This task is a survey of state-of-the-art methods for image classification on Mnist dataset. We do some surveys on newest models for image classification, implement the best one on MNIST dataset and compare obtained results with benchmark results in following link : http://yann.lecun.com/exdb/mnist/.

* 1. **WHY DO WE DO THIS TASK?**

Image classification has always been a hot research direction in the world, and the emergence of deep learning has promoted the development of this field. Convolutional neural networks (CNNs) have gradually become the mainstream algorithm for image classification since 2012, and the CNN architecture applied to other visual recognition tasks (such as object detection, object localization, and semantic segmentation) is generally derived from the network architecture in image classification. In this report, which focuses on the CNN-based methods for image classification task, we cover recent state-of-the-art (SOTA) network architectures.

1. **SOME BASIC TERMS IN DEEP LEARNING**
   1. **CLASSIC CNN ARCHITECTURE**

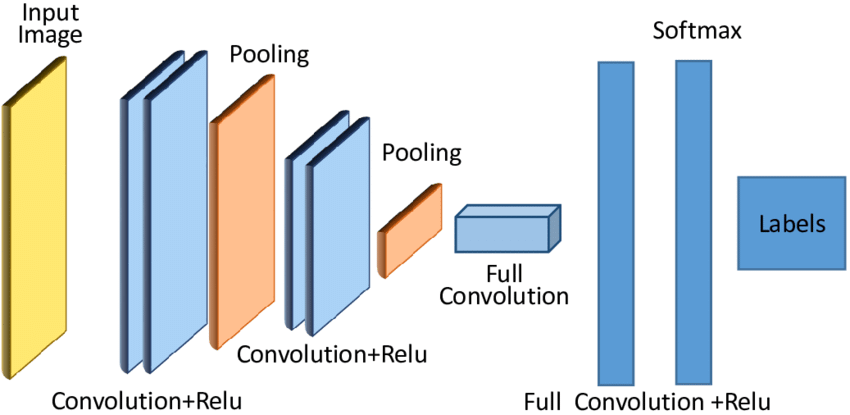


Figure 2: The basic cnn architecture

* 1. **CROSS-ENTROPY LOSS**

Cross-entropy is commonly used in machine learning as a loss function. Cross-entropy is a measure from the field of information theory, building upon [entropy](https://machinelearningmastery.com/what-is-information-entropy/) and generally calculating the difference between two probability distributions. Cross-entropy loss is used when adjusting model weights during training. The aim is to minimize the loss, i.e, the smaller the loss the better the model. A perfect model has a cross-entropy loss of 0. Normally its serves for multi-class and multi-label classifications.

\* *How to compute CE:*

The cross-entropy between two probability distributions, such as Q from P, can be stated formally as:

* H(P, Q)

Where H() is the cross-entropy function,

P may be the target distribution and

Q is the approximation of the target distribution.

Cross-entropy can be calculated using the probabilities of the events from P and Q, as follows:

H(P, Q) = — sum x in X P(x) \* log(Q(x))

* 1. **BATCHNORM**

Batch Norm is a neural network layer that is now commonly used in many architectures. It often gets added as part of a Linear or Convolutional block and helps to stabilize the network during training.

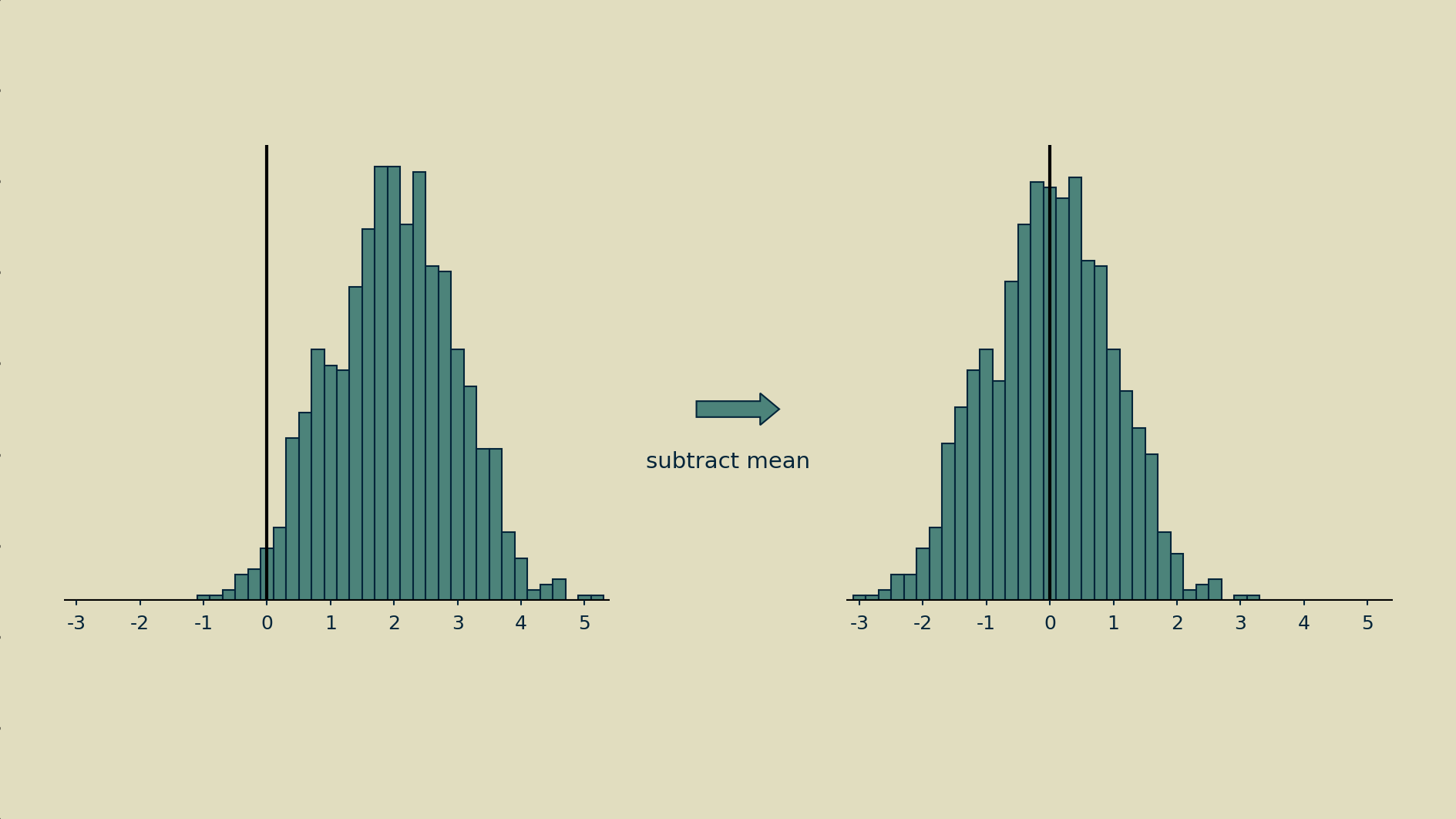


Figure 3.1: Step one is to subtract the mean to shift the distribution.

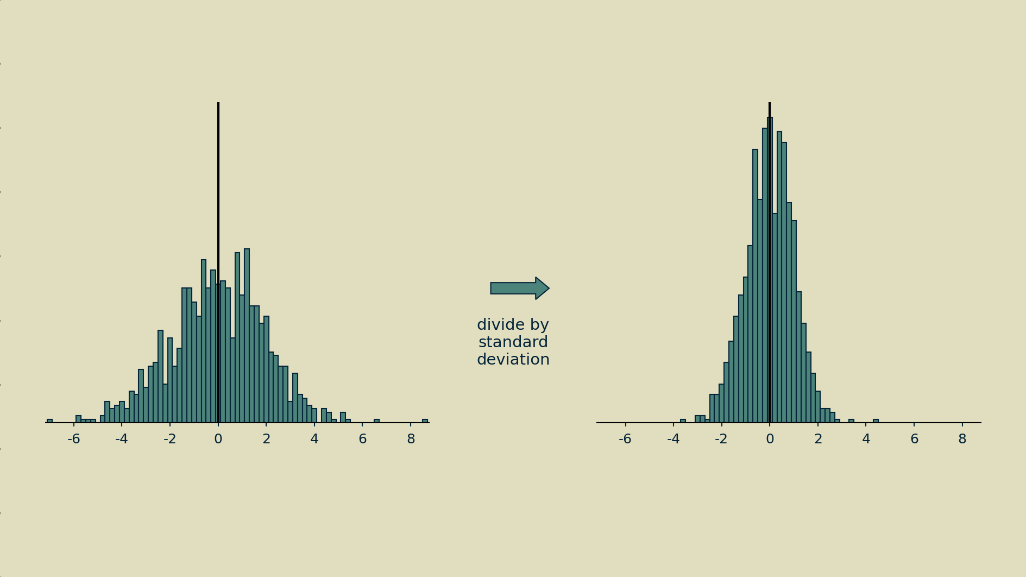


Figure 3.2 Step two is to divide all the shifted values by their standard deviation (the square root of the variance).

* 1. **RELU ACTIVATION**

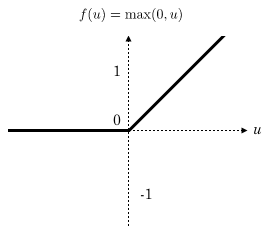


Figure 4: ReLU activation

The rectifier or ReLU (Rectified Linear Unit) activation function is an [activation function](https://en.wikipedia.org/wiki/Activation_function) defined as the positive part of its argument:

{\displaystyle f(x)=x^{+}=\max(0,x)}*y*=*max*(0,*x*)

where *x* is the input to a neuron

Rectified linear units, compared to [sigmoid function](https://en.wikipedia.org/wiki/Sigmoid_function) or similar activation functions, allow faster and effective training of deep neural architectures on large and complex datasets.

* 1. **EXPONENTIAL MOVING AVERAGE**

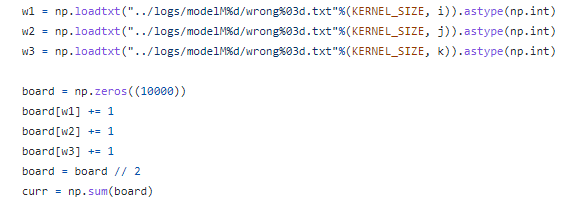
An exponential moving average (EMA) is a type of moving average (MA) that places a greater weight and significance on the most recent data points. The exponential moving average is also referred to as the exponentially weighted moving average.

* 1. **MAJORITY VOTING**

A voting ensemble (or a “*majority voting ensemble*“) is an ensemble machine learning model that combines the predictions from multiple other models. It is a technique that may be used to improve model performance, ideally achieving better performance than any single model used in the ensemble.

A voting ensemble works by combining the predictions from multiple models. It can be used for classification or regression. In the case of classification, the predictions for each label are summed and the label with the majority vote is predicted.

\* *Code implement*



**3. AN ENSEMBLE OF SIMPLE CONVOLUTIONAL NEURA NETWORK MODELS**

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. 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, M3, 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, M5, 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, M7, 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

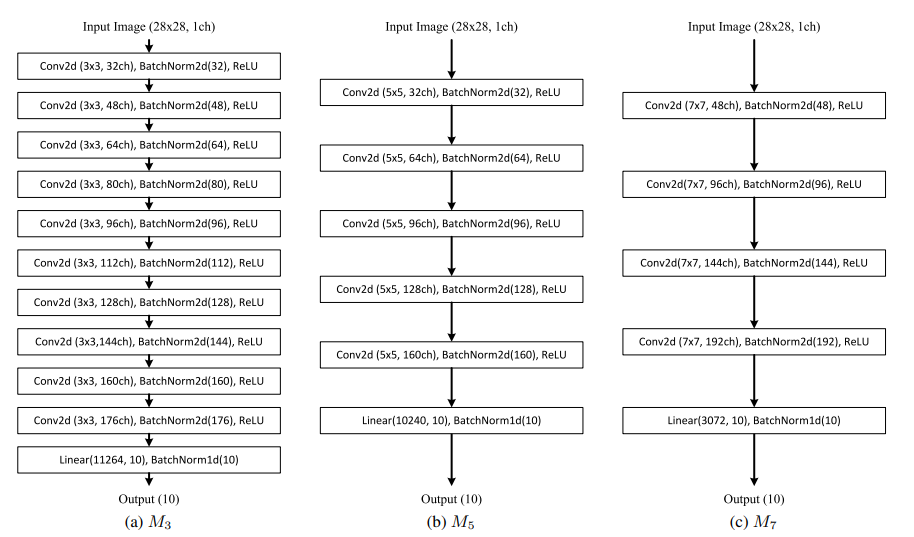


Figure 5: Network models used for MNIST digit classification.

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].

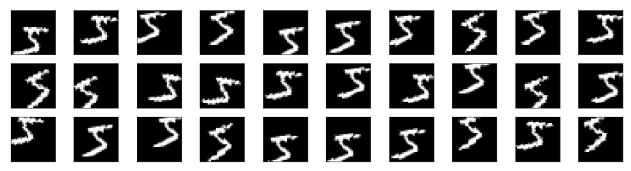
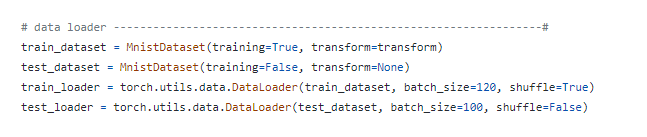


Figure 6: Random translation and random rotation applied to a training image

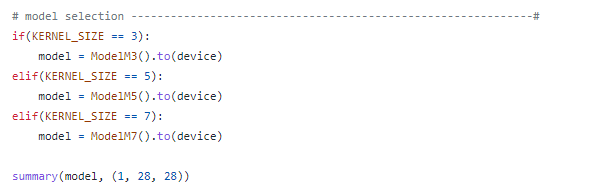
The network parameters are initialized using default initialization methods in PyTorch For parameter optimization, we use the Adam optimizer with cross-entropy loss function. Learning rate starts at 0.001, and exponentially decays with decaying factor γ=0.98. The batch size is 120, and so 500 parameter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

**4. IMPLEMENT ON MNIST DATASET**

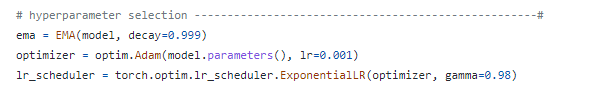
**- Load data**



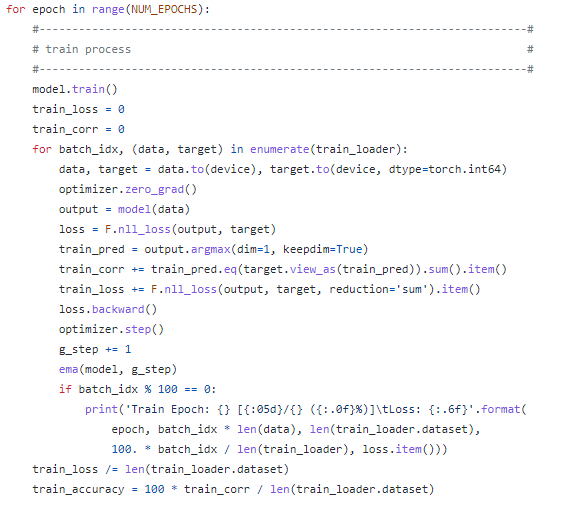
**- Select model**



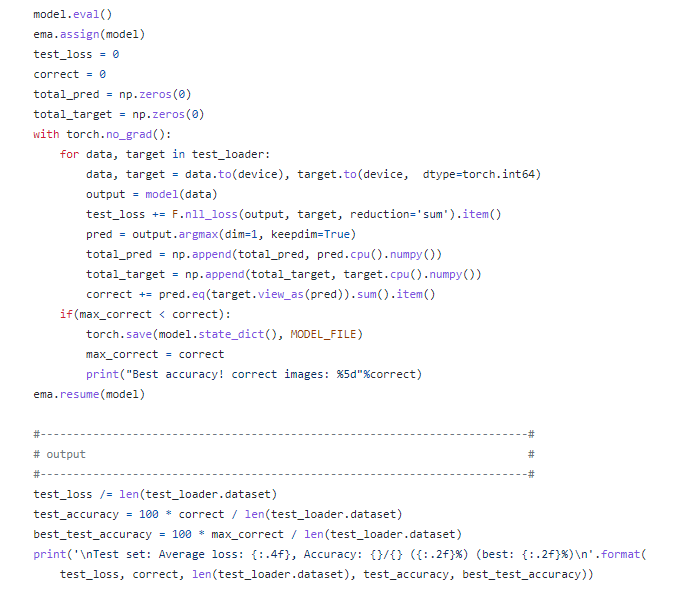
**- Choose hyperparameter**



**- Training**



**- Evaluate**



**5. COMPARE OBTAINED RESULTS**

**4.1 BENMARK RESULTS**

|  |  |  |
| --- | --- | --- |
| Architecture | Preprocessing | Error rate |
| Convolutional net LeNet-1 | subsampling to 16x16 pixels | 1.7 |
| Convolutional net LeNet-4 | none | 1.1 |
| Convolutional net LeNet-4 with K-NN instead of last layer | none | 1.1 |
| Convolutional net LeNet-4 with local learning instead of last layer | none | 1.1 |
| Convolutional net LeNet-5, [no distortions] | none | 0.95 |
| Convolutional net LeNet-5, [huge distortions] | none | 0.85 |
| Convolutional net LeNet-5, [distortions] | none | 0.8 |
| Convolutional net Boosted LeNet-4, [distortions] | none | 0.7 |
| Trainable feature extractor + SVMs [no distortions] | none | 0.83 |
| Trainable feature extractor + SVMs [elastic distortions] | none | 0.56 |
| Trainable feature extractor + SVMs [affine distortions] | none | 0.54 |
| unsupervised sparse features + SVM, [no distortions] | none | 0.59 |
| Convolutional net, cross-entropy [affine distortions] | none | 0.6 |
| Convolutional net, cross-entropy [elastic distortions] | none | 0.4 |
| large conv. net, random features [no distortions] | none | 0.89 |
| large conv. net, unsup features [no distortions] | none | 0.62 |
| large conv. net, unsup pretraining [no distortions] | none | 0.60 |
| large conv. net, unsup pretraining [elastic distortions] | none | 0.39 |
| large conv. net, unsup pretraining [no distortions] | none | 0.53 |
| large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions] | none | 0.35 |
| committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions] | width normalization | 0.27 +/- 0.02 |
| committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions] | width normalization | 0.23 |

Table 1: Benchmark Results For Mnist Classification. The 1st Column Is Classifiers Based On Cnn, The 2nd Column Is About Preprocessing Techniques And The Last One Is The Percent Of Test Error Rate

**4.2 OUR RESULTS ON TEST SET**

Table 2: Accuracy of individual networks

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Min Acc** | **Max Acc** | **Average Acc** |
| **M3** | 98.44% | 99.79% | 99.61% |
| **M5** | 98.68% | 99.81% | 99.61% |
| **M7** | 98.30% | 99.78% | 99.56% |

Table 2 shows the minimum, average, maximum accuracy of individual networks with the kernels are 3, 5, and 7, respectively. As we can see, the model with kernel size = 5 achieve the best accuracy ( about 99.81%) and the third model is the worst. The average accuracy of M3 and M5 is slightly higher followed by *M*7, but the difference is not too significant (less than 0.05%).

Table 3: Error Rate of individual networks

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Max Error Rate** | **Min Error Rate** | **Avg Error Rate** |
| **M3** | 1.56% | 0.21% | 0.39% |
| **M5** | 1.32% | 0.19% | 0.39% |
| **M7** | 1.7% | 0.22% | 0.44% |

It is known that using ensemble of networks can improve generalization and achieve higher test accuracy. 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.

Table 4: Accuracy of ensemble of individual networks

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Min Acc** | **Max Acc** | **Average Acc** |
| **M3+M3+M3** | 99.73% | 99.85% | 99.79% |
| **M5+M3+M3** | 99.75% | 99.85% | 99.80% |
| **M7+M3+M3** | 99.74% | 99.85% | 99.80% |
| **M3+M5+M7** | 99.74% | 99.86% | 99.80% |

Table 4 and 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. Table 4 shows the test accuracy of the four ensemble methods discussed above. The accuracy of test set using ensemble of homogeneous networks is better than individual ones.

Table 5: Error Rate of ensemble of individual networks

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Max Error Rate** | **Min Error Rate** | **Avg Error Rate** |
| **M3+M3+M3** | 0.27% | 0.15% | 0.21% |
| **M5+M3+M3** | 0.25% | 0.15% | 0.20% |
| **M7+M3+M3** | 0.26% | 0.15% | 0.20% |
| **M3+M5+M7** | 0.26% | 0.14% | 0.20% |

It can be observed from Table 5 that while the maximum test error rate of homogeneous ensemble methods are similar, the ensemble method where one network is selected from each type of networks achieves lower error rate. And the most remarkable point is that our model got lowest error rate ever. The best error rate in benchmark results for Mnist classification is 0.23, and our error rate is only 0.2!

**6. 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 99.77%. In this report following the SOTA method, 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 99.80% 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.