CNN Based Handwriting Recognition with 0.0305 FOM

Columbia University

Professor
Dr. Jinjun Xiong

Group
Data Laundry

Group Members
Yuqing Yao, Zhijuan Pan, Mengyang Ma

April 30 2017

CNN Based Handwriting Recognition with 0.0305 FOM

I. Introduction

In the traditional model of pattern recognition, a hand-designed feature extractor gathers relevant information from the input and eliminates irrelevant variabilities (LeCun & Bengio, 2016). Over the last several years, image recognition and classification has become a popular task which has been widely tested using various machine learning techniques. These techniques have played an increasingly important role in the design of handwritten recognition.

The goal of this project is to build a 10-class classifier to recognize the handwriting digits accurately. The main task of this project is to explore various techniques related to neural network and deep learning to solve the 10-class classification problem using the well-known MNIST dataset.

1.1 Dataset

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image.

The training and the testing sets are separated in the original database. In the process of training our Neural Network model, we targeted at using the least number of training data to attain the best possible testing data accuracy and achieve the best network structure.

1.2 Neural Network Methodologies

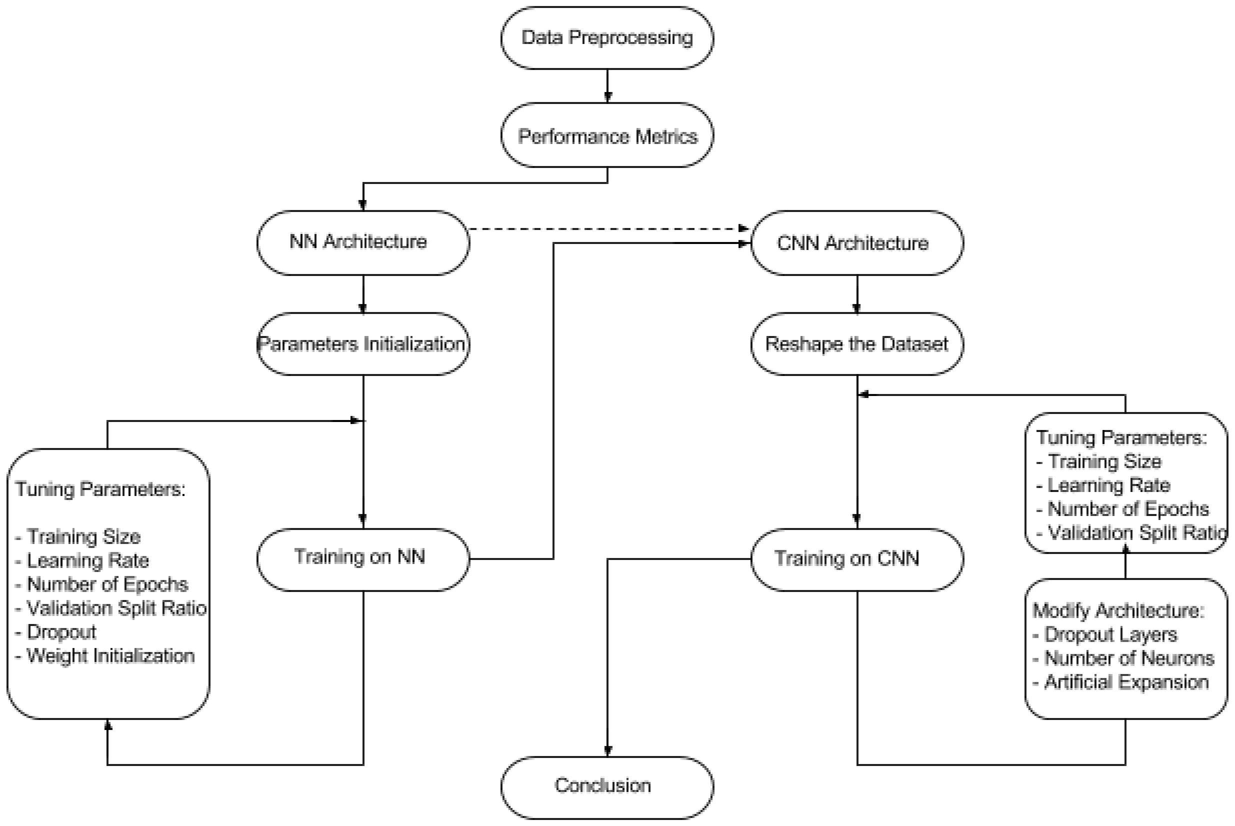
There are many types of neural networks in modern machine learning models, such as feedforward neural network, radial basis function (RBF) network, recurrent neural network and modular neural networks.

In this image recognition task, we first applied a simple neural network model which contains one input, one output and two hidden layers. We then trained a convolutional neural network (CNN), a type of feed-forward artificial neural network which has been used for image recognition and classification widely. Since CNN has been proven very effective in tasks such as identifying faces, objects and traffic signs, we determined to train a well-structured CNN for better image recognition results.

1.3 Technology Roadmap

We plan to start from the plain BP Neural Network, train on the optimal values of parameters, and move on to advanced model structures, then we'll tune the parameters again. For the Convolutional Neural Network, we will discuss the optimization by two perspectives: model topology and hyperparameters.

Please see the graph below for the roadmap of this project.



1.4 Work Division

Zhijuan Pan: Basic NN Architecture, Weight Initialization, Training on Dropout, Artificial Expansion, Performance Metrics Functions

Yuqing Yao: CNN Architecture, Batch Normalization, Number of Neurons, Data Preprocessing, Training Size, Artificial Expansion, Model integration

Mengyang Ma: Batch Size, Validation Ratio, Learning Rate, Number of Epochs, Visualization, Report Drafting

II. Experimental Design

2.1 Data Preprocessing

After loading the database and necessary TensorFlow packages into python, we prepared the MNIST dataset mainly in three steps: normalizing the dataset, reshaping the dataset and One-Hot Encoding.

Normalization
The MNIST dataset has pixel values in a range from 0 to 255. Thus, we started our data preprocessing with simple rescaling by shifting the data into a range from 0 to 1.

Reshaping
The MNIST digits are provided with 2 dimensions per example representing a greyscale image in 28*28 pixels. In our basic neural network training model, the 2-dimensioanl digits were transformed into vectors of 784. In the following CNN model, the layer design required the images to be explicitly expressed in 1-channel images. The Convolution2D layers in Keras package required the number of samples, number of channels, width and heights of the images. Therefore, we reshaped the vectors of 784 from the basic NN model into 28*28 images again when we trained the CNN model.

One-Hot Encoding
This is an approach which converts the class label integers into a so-called one-hot array, where every unique label is represented as a column in the new array (Raschka, 2017). The MNIST digits are from 0 to 9, and we encoded these digits into a “one-hot” format of vectors of 10*10.

2.2 Parameter Selection

There are many ways to train and tune a neural network model through changing the model parameters, hyper-parameters, activation function and cost function, etc. Due to the limited time, we determined to train our NN model based on training size, varied number of layers, validation split ratio, learning rate, number of epochs, batch size, initializing weights, and the callbacks in early stopping.

We maintained the activation function (hidden layer-rectified linear unit, output-softmax), optimization method (SGD), and the cost function (cross-entropy) unchanged.

2.3 Performance Metrics

Three metrics were employed to evaluate the training model performance: test loss, test accuracy and “FOM”.

Test Loss

When we trained the neural network with Tensorflow, we incorporated the “evaluate” function in Keras package to compute the loss on the testing data, batch by batch. Loss interprets how well the model is doing for errors made for each example in the testing set. The objective, therefore, is to reduce or minimize the loss function’s value with respect to different model parameters.

Test Accuracy

Similarly, the Keras package reports the test accuracy in the “evaluate” function. We thus extracted this metric and used it as an evaluating metric for our NN models.

FOM

FOM = P1/2 + (1-P2), where P1 is how much the used training samples take account of the entire training samples (60K). P2 is the test accuracy. The objective, therefore, is to attain a FOM as small as possible. This can be achieved by using as small number of training data while achieving as small test error as possible. The scaling factor of 2 for P1 is to give more weight to the test error than training data size. □

2.4 Controlled Experiments

Machine learning algorithms have been investigated through performing systematic research into the algorithms behavior, and this is achieved by designing and executing controlled experiments (Brownlee, 2014).

Applying similar insights, we trained our neural network models in a controlled experiment manner: we changed only one parameter in each training session and tuned the model on an iterative base. In this approach could we assess the influence of each changed model parameter instead of confounding the effects of several varied parameters in the training process.

III. Basic NN Model

3.1 Model Architecture

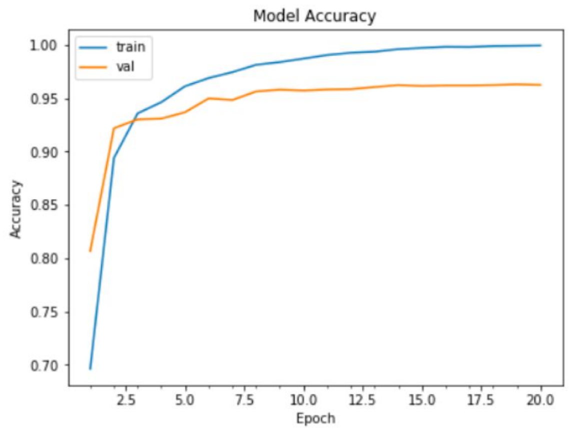
We constructed our basic NN model based on approaches employed in literatures with relevant topics, and the model architecture for our basic NN can be described in four aspects.

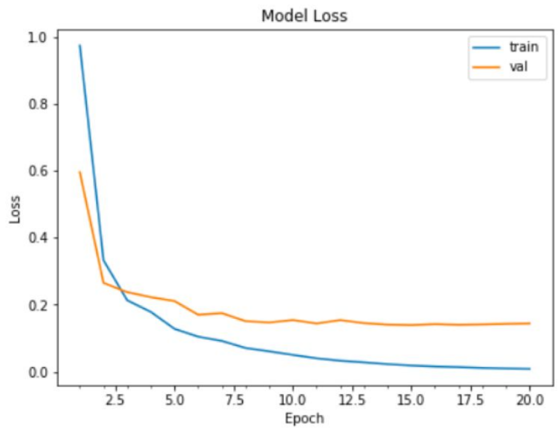
There are three layers in our basic NN model: Input—2 hidden layers—output. There are 784 neurons in the Input layer, 300 neurons in the 1st hidden layer, 100 neurons in the 2nd layer, and 10 neurons in the output layer. The activation function for the hidden layers is Relu (Rectified linear unit) and the activation function for the output layer is Softmax. The cost function employed is Categorical Cross Entropy.

3.2 Parameter Selection

We initially used a training size of 20,000, a validation ratio of 0.3, a learning rate of 0.5, number of epochs of 20, and a batch size of 300 for our basic NN model. The selections were based on either the common used values or the values applied in literatures with relevant tasks. The weight initialization method was according to Keras default selection, named glorot_uniform. It draws samples from a uniform distribution within a limit, where limit is sqrt(6 / (fan_in + fan_out)), where fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor.

3.3 Performance





```
 9440/10000 [==========================>..] - ETA: 0sTest Loss:  0.115154625503
Test Accuracy:  0.9664
FOM:  0.200266666667
```

These graphs demonstrate how well the basic NN model performs on the training and validation sets. As we can see, accuracy grows and loss drops dramatically at the beginning. The four curves start to become flat after 5 epochs, and eventually remain stable.

The gap on the accuracy and loss between the validation and training sets illustrates an overfitting issue of our basic NN model, which indicates that proper approaches regarding fixing overfitting issue such as regularization and dropout should be employed in the following tuning process.

In addition, the reported test accuracy is around 0.9664, which is not as high as expected. The reported test loss is around 0.115, and the FOM is around 0.2. These two metrics are not as low as expected. All metric statistics suggest that further tuning and modification methods should be applied to our CNN model in the following training steps.

IV. Model Optimization

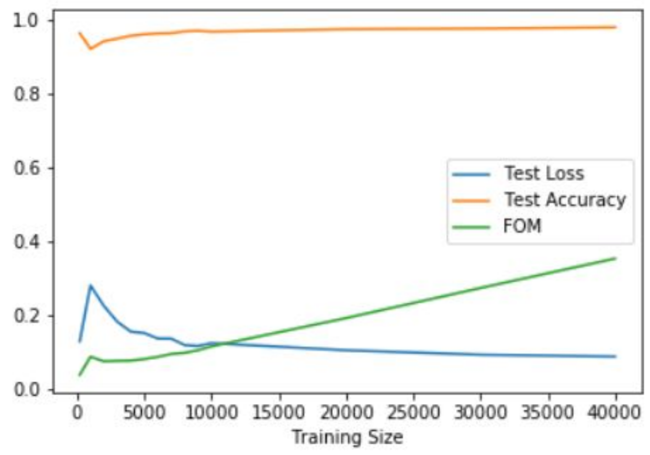
We choose to sequentially discuss the influence of different parameters each at a time, find the optimal value, then plug in the optimal values to the basic model, and finally search around each of the optimal values to get the optimal result.

4.1 Training Size

It is a rule of thumb in the machine learning field that increasing the size of training set tends to boost the model performance. Foody et al. (1995) found that the accuracy of sensor image data increases significantly as the training size increases. However, there usually exist saturation points for sample size in model training process (Lawrence et al., 1996).

We created a list of numbers ranging from 200 to 40,000 for training size to experiment. Each of the numbers are plugged into the process of sampling from the 60,000 training data. Then we use the subsampled training set to fit the basic model.

Ceteris paribus, we get the following graph of Test Loss, Test Accuracy, and FOM versus the changing training size.



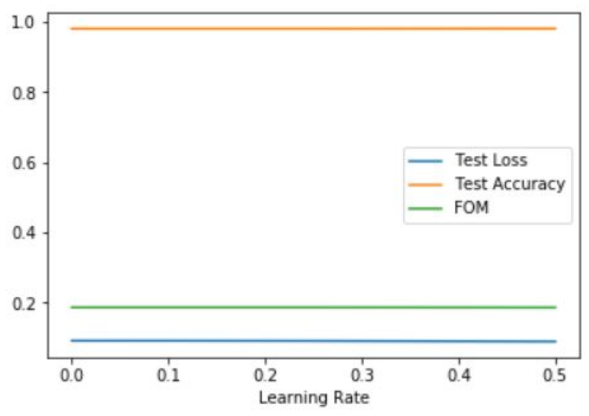
The graph illustrates that the performance metrics of the basic NN model are improved when we increase the training size. The Test Loss is decreased significantly at the early stage. However, the marginal difference decreases as the training size increases. The Test Accuracy also increases over the training process, but the increment slows down as the training size increases. The FOM almost linearly increases when we increase the training size. Therefore, we should keep the training size at a reasonably low level.

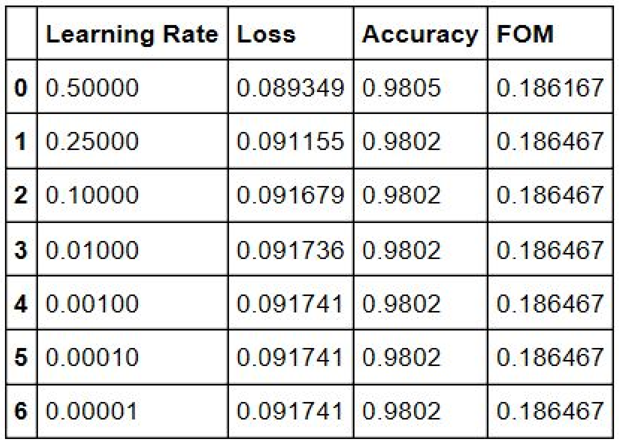
From the graph above, we can see that when the training size stays within the 1,000 to 3,000 zone, FOM is relatively stable and even achieves the lowest at around 2,000. Therefore, we chose 2,000 as the optimal training size.

4.2 Learning Rate

Low values of learning rate η lead to slow learning (Kandil et al., 1993) while lower values decrease the fluctuation of loss or accuracy during the training process and lead to more stable results.

We start from 0.5 and gradually decrease the learning rate to 1×10^{-5}. The following graph and chart shows that decreasing the learning rate does not significantly improve the model performance.





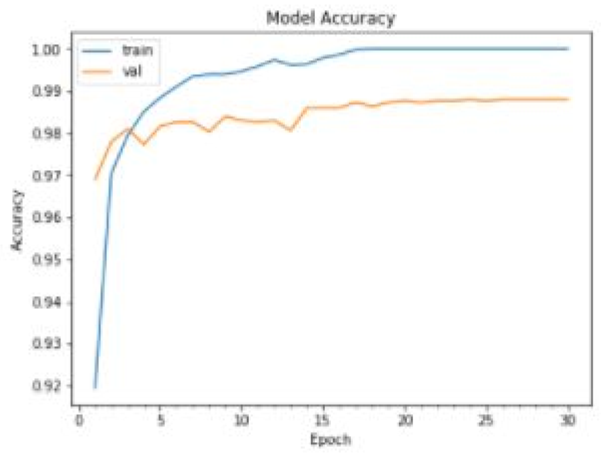
	Learning Rate	Loss	Accuracy	FOM
0	0.50000	0.089349	0.9805	0.186167
1	0.25000	0.091155	0.9802	0.186467
2	0.10000	0.091679	0.9802	0.186467
3	0.01000	0.091736	0.9802	0.186467
4	0.00100	0.091741	0.9802	0.186467
5	0.00010	0.091741	0.9802	0.186467
6	0.00001	0.091741	0.9802	0.186467

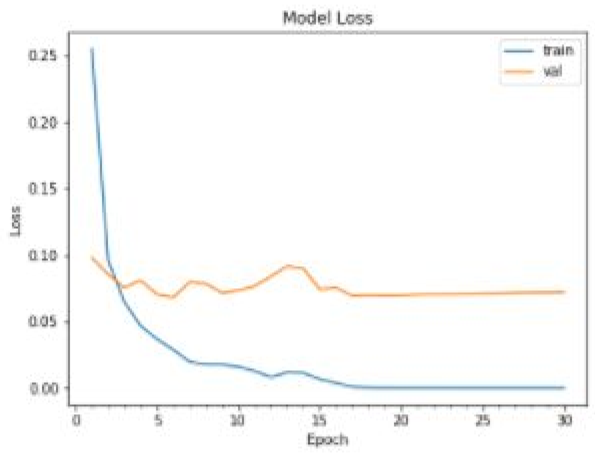
Therefore we will keep the learning rate at 0.5 for the initial optimization.

However, the result of unchanged performance doesn’t mean that learning rate is not an important parameter in the training process because we only plotted the end result, not the whole process of each iteration. As we’ll discuss later in the CNN optimization part, a small learning rate indeed lead to a more stable convergence process.

4.3 Epochs

We plotted as below the the Accuracy and Loss of training and validation set for models trained through 1 to 30 epochs.



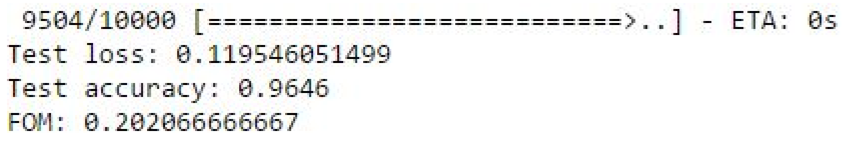


The accuracy increases as the number of epoch per session increases but plateaued at around 20. The Loss also start to yield tiny decrease at the epoch number of 20. a Therefore, we use 20 as the number of epochs for other parts of the training.

4.4 Validation Split Ratio

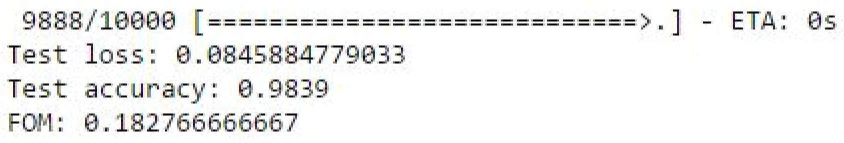
Initially, we used 0.3 as the default value of validation split. It means for each round of training, 30% from the training set are drawn as validation set.

When the validation split ratio equals 0.3, the basic model performance is as follows.



```
 9504/10000 [===========================>..] - ETA: 0s
Test loss: 0.119546051499
Test accuracy: 0.9646
FOM: 0.202066666667
```

We referred to some literature that discuss the validation process and expected that when validation ratio decreases to a reasonably small level, the performance may become better. Therefore, we changed it to 0.05 and got the following result.



```
 9888/10000 [============================>.] - ETA: 0s
Test loss: 0.0845884779033
Test accuracy: 0.9839
FOM: 0.182766666667
```

The Accuracy slightly increased and the Loss significantly decreased from 0.1195 to 0.0846, resulting in the 0.02 decrease of FOM. Hence, we set the validation proportion to 0.05.

4.5 Dropout

The dropout technique is supposed to tackle the overfitting issue. We change the dropout rate from 0.2 to 0.3, then 0.4. The model performances is as follows.

Dropout Rate	0.2	0.3	0.4
Test Loss	0.1058	0.1090	0.1126
Test Accuracy	0.9673	0.9682	0.9670
FOM	0.1994	0.1985	0.1997

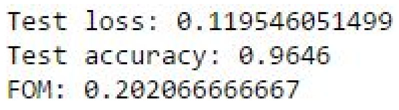
Therefore, we use 0.3 as the dropout rate.

4.6 Weight Initialization

Some literatures suggest that the ways of initializing weights to neurons lead to different levels of performance of Neural Network models. “A small miscalibration of the initial weights leads to vanishing or exploding gradients, as well as poor convergence properties.” (Krähenbühl et al., 2016).

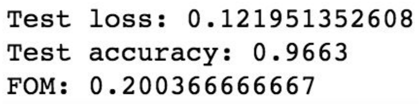
Therefore, we compared the Glorot uniform with random uniform as initialization method.

For Glorot uniform:



```
Test loss: 0.119546051499
Test accuracy: 0.9646
FOM: 0.202066666667
```

For random uniform:



```
Test loss: 0.121951352608
Test accuracy: 0.9663
FOM: 0.200366666667
```

We regard random uniform as a better choice for this model.

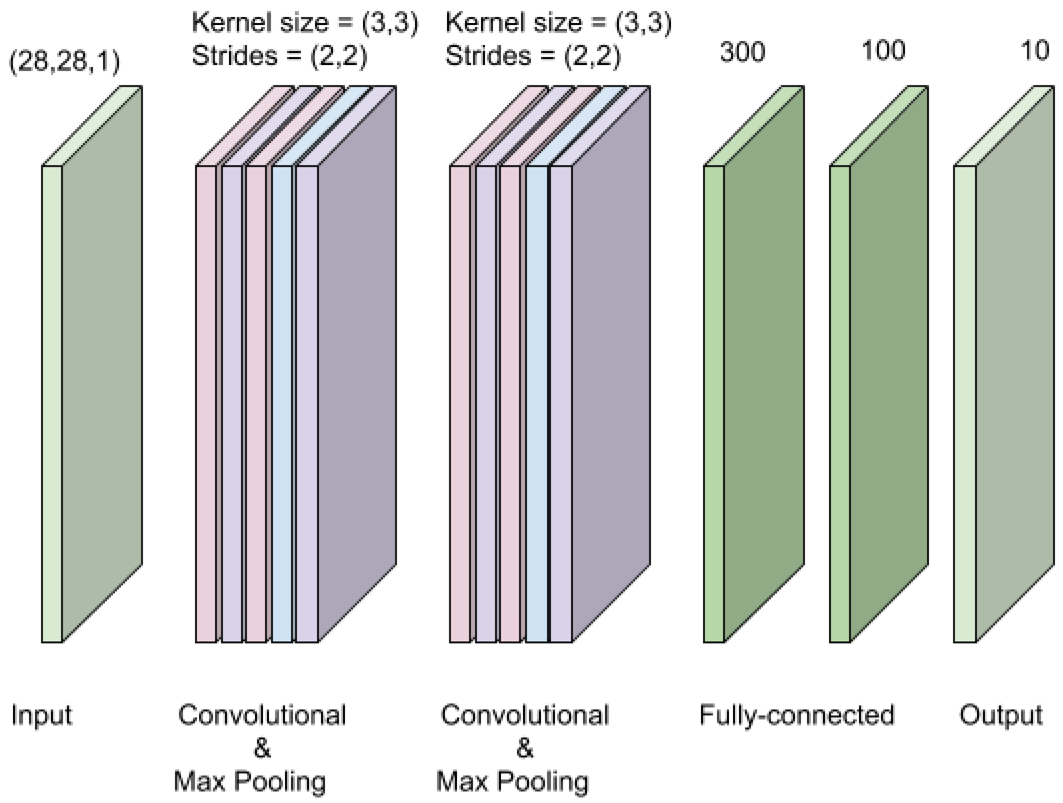
4.7 Network Architecture: CNN

LeCun et al. (1998) officially published the Convolutional Neural Network and it soon became one the of most successful models in image recognition field of studies.

We introduce the structure of convolutional layers into our model in hope that it will improve the performance of the model.

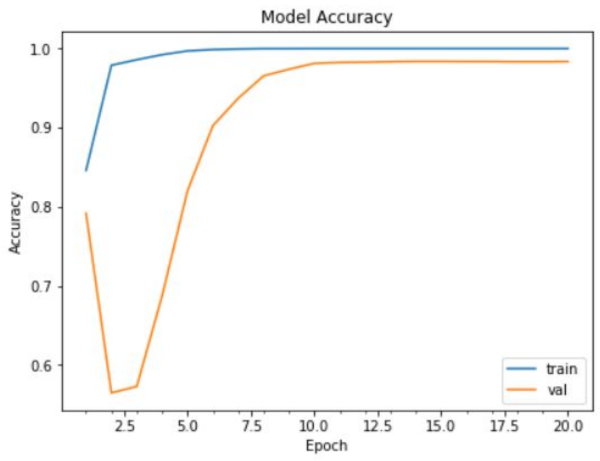
We start from 1 input layer, 2 convolutional layers, 2 max pooling layers, 4 batch normalization layers, then 2 fully connected hidden layers, and finally the output layer.

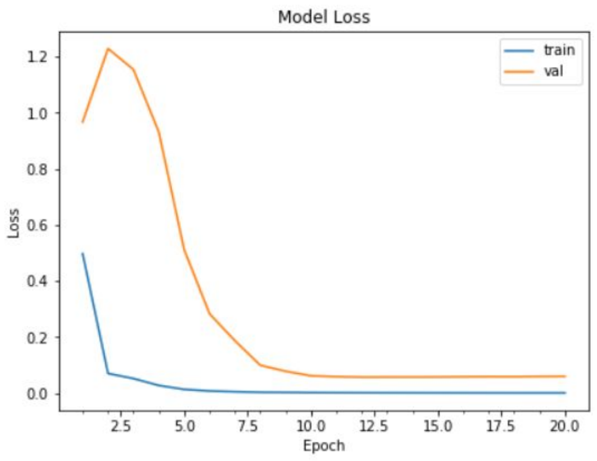
Please see the detailed CNN architecture as below.



We used Rectified Linear Unit as activation function and Categorical Cross Entropy as cost function.

For 20,000 training samples, the CNN model yields the following results.





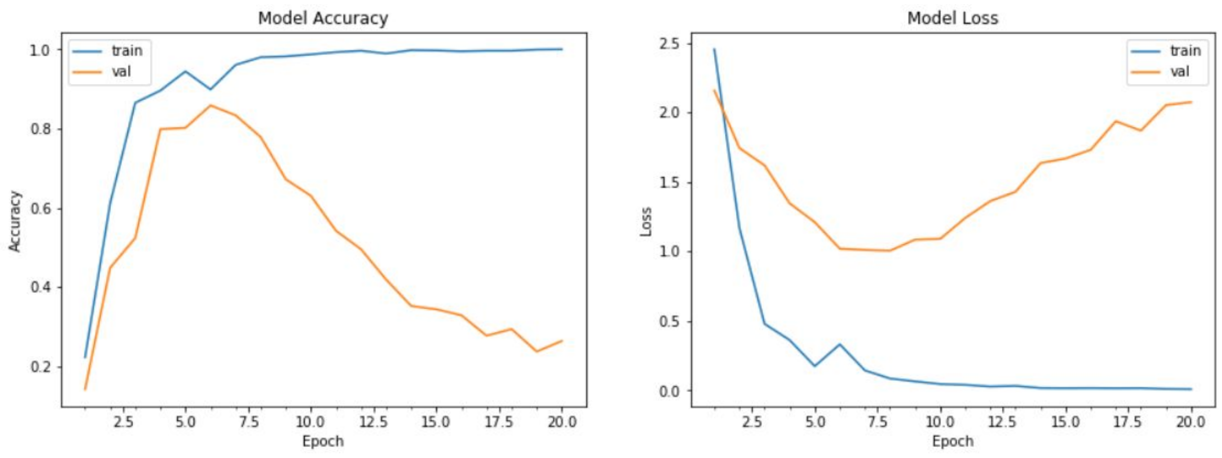
```
10000/10000 [==============================] - 13s

Test Loss:  0.0470036370637
Test Accuracy:  0.9872
FOM:  0.179466666667
```

We find that FOM is still very high and the validation performance is highly dispersed from the training result. Hence, we try to refine the dropout layers to see if model performance can be improved.

4.8 Refining Dropout Layers

We add 2 dropout layers, each after the convolutional layers. However, the dropout seems make overfitting more severe.



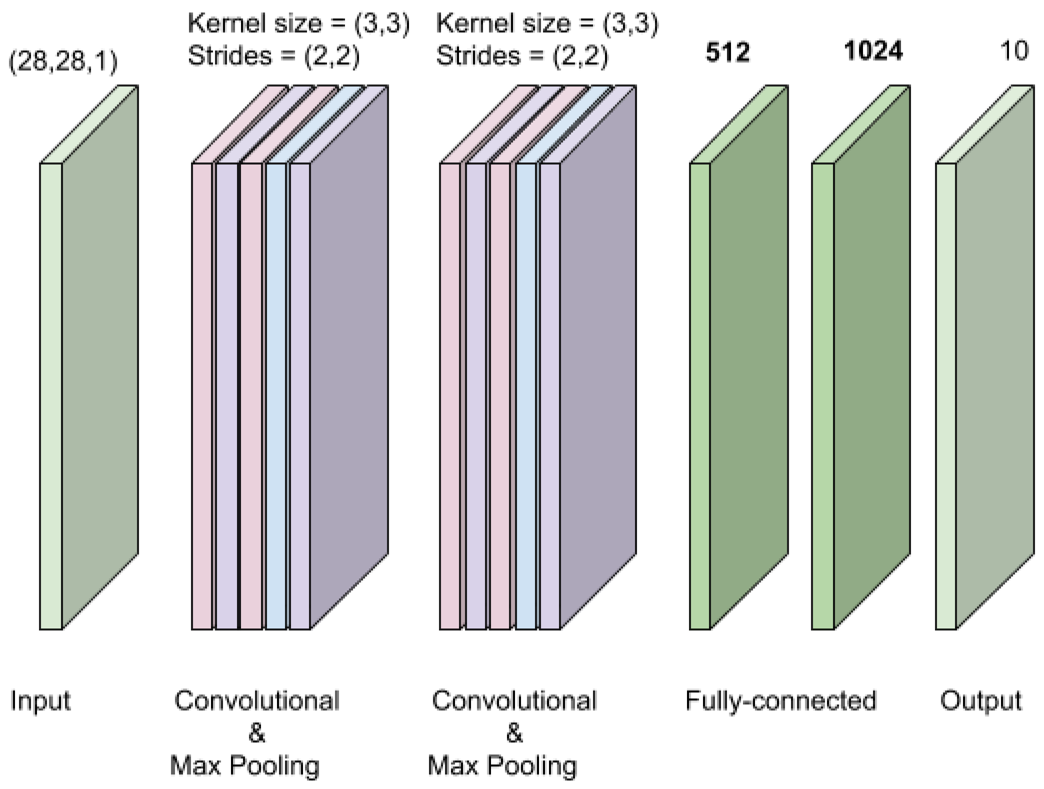
```
10000/10000 [==============================] - 14s

Test Loss:  1.95319421539
Test Accuracy:  0.297
FOM:  0.719666666667
```

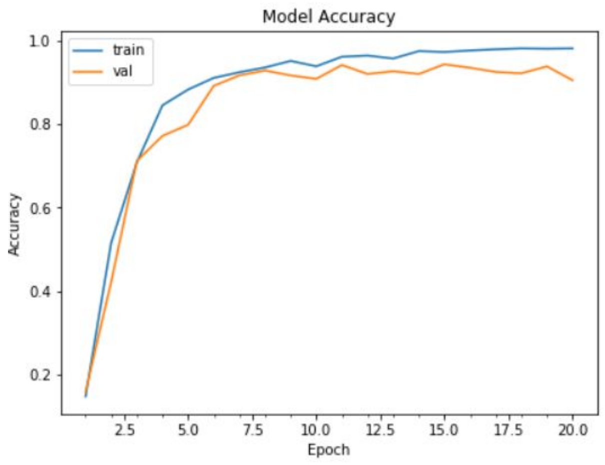
From the graph above, we can see that the validation performance (orange line) firstly improves, then revert at a certain point. Therefore, we tune the dropout by decreasing the ratio and got a more stable result.

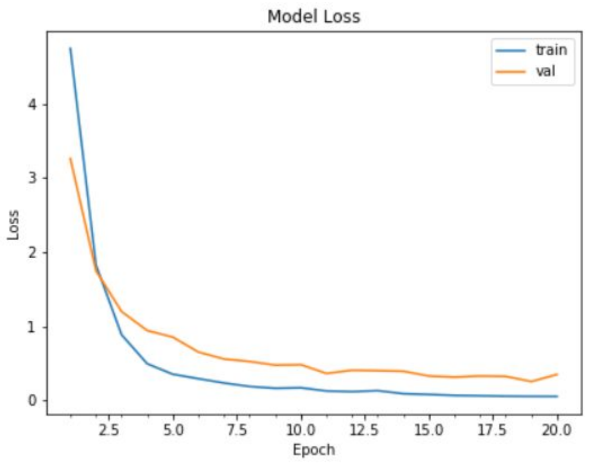
4.9 Number of Neurons

Usually the more neurons a neural network has, the more features it can capture so we tried on increasing the number of neurons on the hidden layers. 512 and 1024 are the magic numbers usually used in practice of Neural Network training. Thus we increase the number of neurons of our hidden layers to 512 and 1024 respectively.



We train the model again and found out that increasing the number of neurons indeed improve the performance. At the same time, the curve of validation performance also became closer the the training performance curve. This means the robustness of the CNN also largely increased.





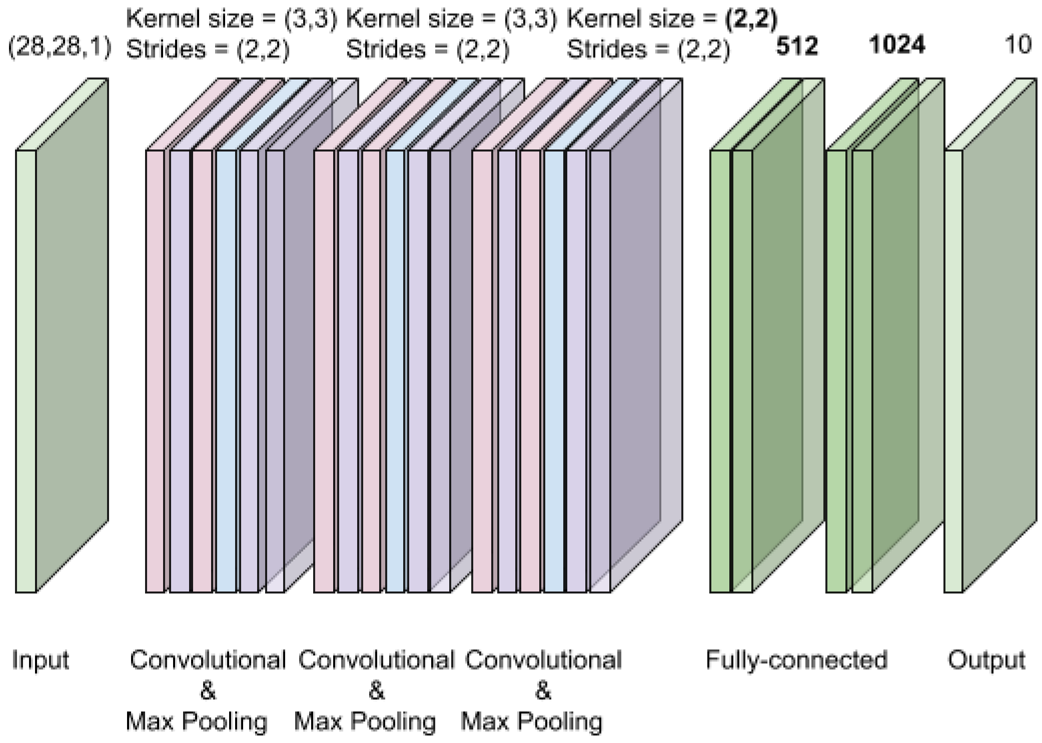
```
 9984/10000 [============================>.] - ETA: 0sTest Loss:  0.283481245667
Test Accuracy:  0.9205
FOM:  0.0961666666667
```

As a consequence, we keep the structure of (512, 1024) neurons for hidden layers.

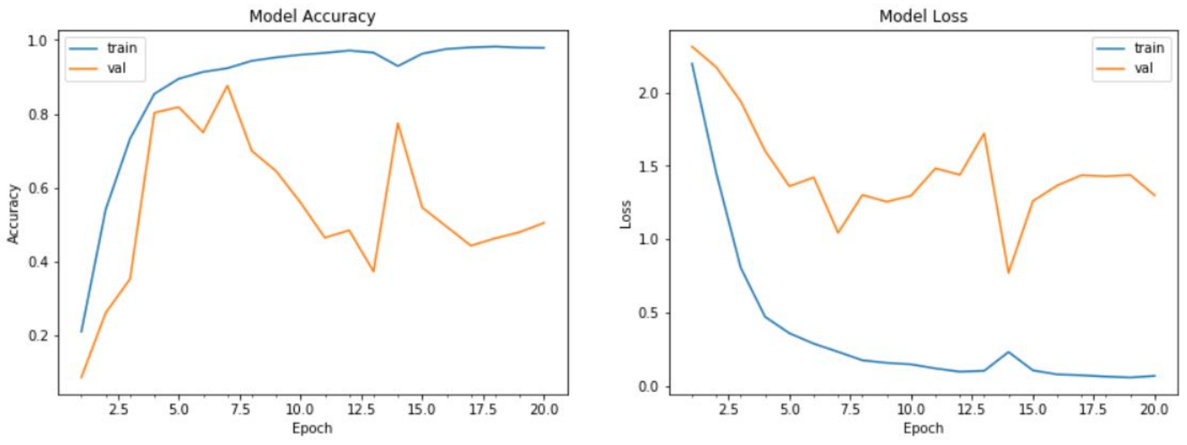
4.10 Number of Convolutional Layers

It is commonly acknowledged that deeper Neural Networks tend to have more explanatory and predictive power. We suggest that it worth a try to add convolutional layers to improve the model performance.

We added a convolutional layer with corresponding batch normalization layer and max pooling layer to the structure. To be clear, please refer to the following graph for details.


(28,28,1)
Kernel size = (3,3) Strides = (2,2)
Kernel size = (3,3) Strides = (2,2)
Kernel size = (2,2) Strides = (2,2)
512
1024
10
Input
Convolutional & Max Pooling
Convolutional & Max Pooling
Convolutional & Max Pooling
Fully-connected
Output

However, the convolutional layer seems to make the validation performance fluctuate violently. At the same time, the training performance seems to have nice curves. This kind of pattern indicates overfitting of model.



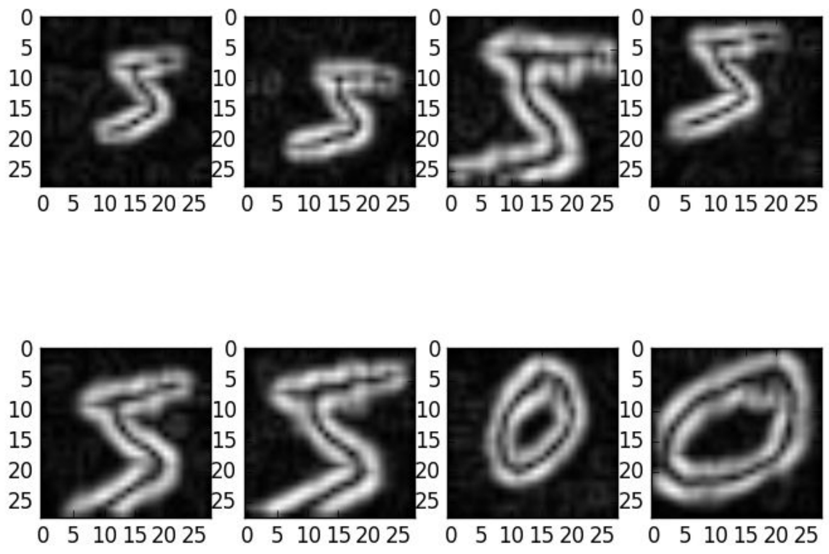
```
 9984/10000 [============================>.] - ETA: 0sTest Loss:  1.20280556631
Test Accuracy:  0.5416
FOM:  0.475066666667
```

We tried tuning on the 3 layer convolutional Neural Network but the interactions of different parameters and the model components appear to have complicated patterns and we didn’t find a direction to work on. Due to limited time and resources we have, for this project we chose to go back to 2 layer CNN model.

4.11 Artificial Image Augmentation

One of the ways to address the overfitting issue is to increase the training samples. However we want to keep the number of samples used as low as possible. Here we borrow the technique of Artificial Image Augmentation to kill two birds with one stone.

Image Augmentation is proposed by Krizhevsky et al. (2012) to artificially generate new samples from existing training samples by tweaking the original data.

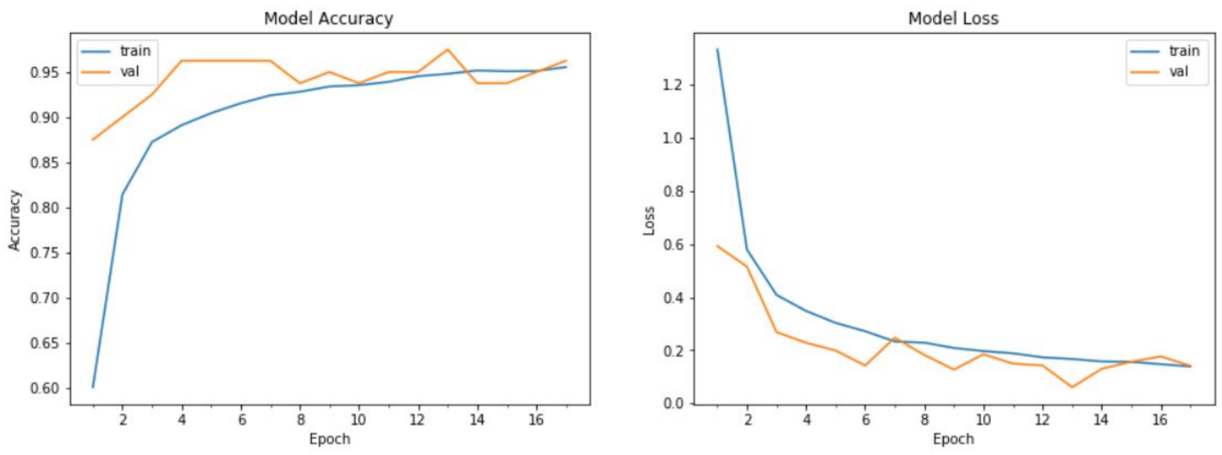


Source: JdeRobot Official Website[1]

We used zooming (range of 0.2), height shifting (range of 0.2), width shifting (range of 0.2), and rotation range of 20 degrees.

For each batch of data, we can randomly generate transformed training samples from the batch.

We firstly set the steps per epoch to 50 and got the following result.



[1] David Pascual Hernández, Deep Learning on RGBD sensors. Retrieved from: http://jderobot.org/Dpascual-tfg#Real-time_data_augmentation

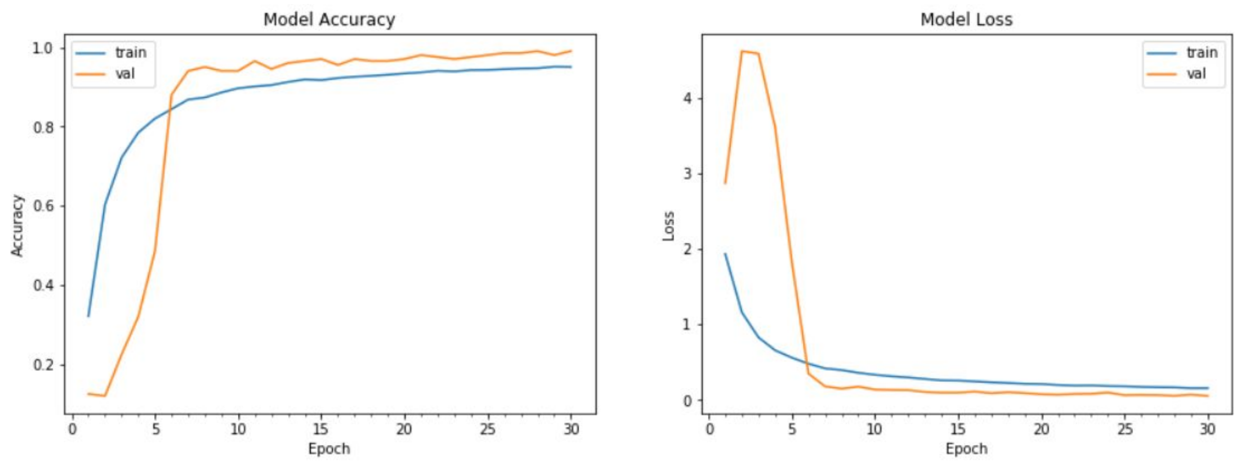
```
 9984/10000 [============================>.] - ETA: 0s
Test Loss:  0.0492120004125
Test Accuracy:  0.9854
FOM:  0.0312666666667
```

The training performance curves seems nice but the validation performance curves still fluctuate along the way.

This may due to the learning rate is too large for this stage, causing the optimization process hard to converge. Therefore, we decreased the learning rate to 0.05.

Moreover, when we use the validation split of 0.05, the validation set is small is we use 50 as step size. For this consideration, we increased the validation size to 200.

After the changes, we got the following results.

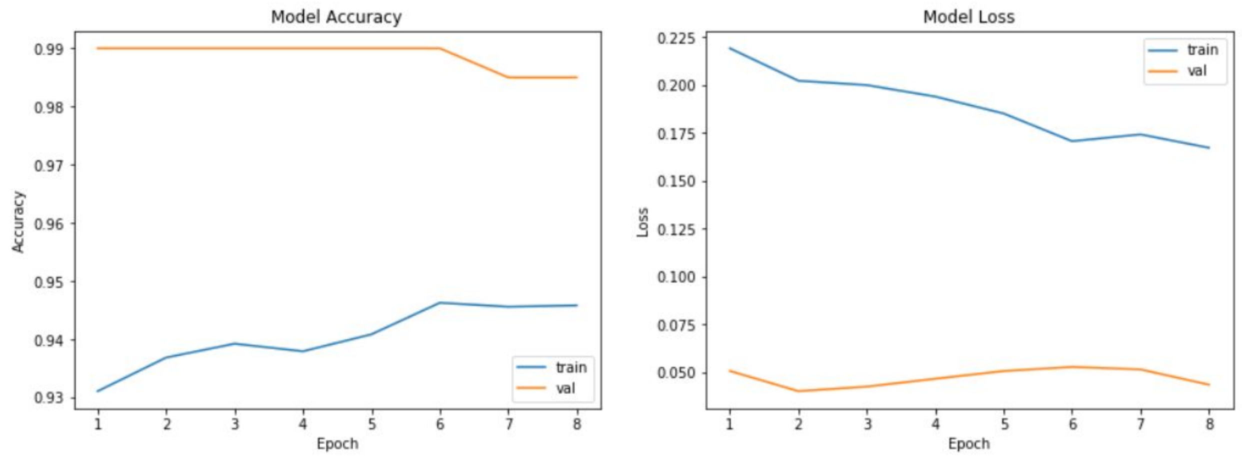


```
 9984/10000 [============================>.] - ETA: 0s
Test Loss:  0.0441268049651
Test Accuracy:  0.985
FOM:  0.0316666666667
```

The FOM decreased to 0.0317, which is reasonably good. However, we want to give a final try on decreasing the training size to give the FOM a decrease.

We then used 1,000 as the training size to sample from the original 60,000 training dataset. To compensate for the decreased training size, we increased the steps per epoch to 100.

The final optimized model has the following performance.



```
 9984/10000 [============================>.] - ETA: 0s
Test Loss:  0.0418975893818
Test Accuracy:  0.9862
FOM:  0.0304666666667
```

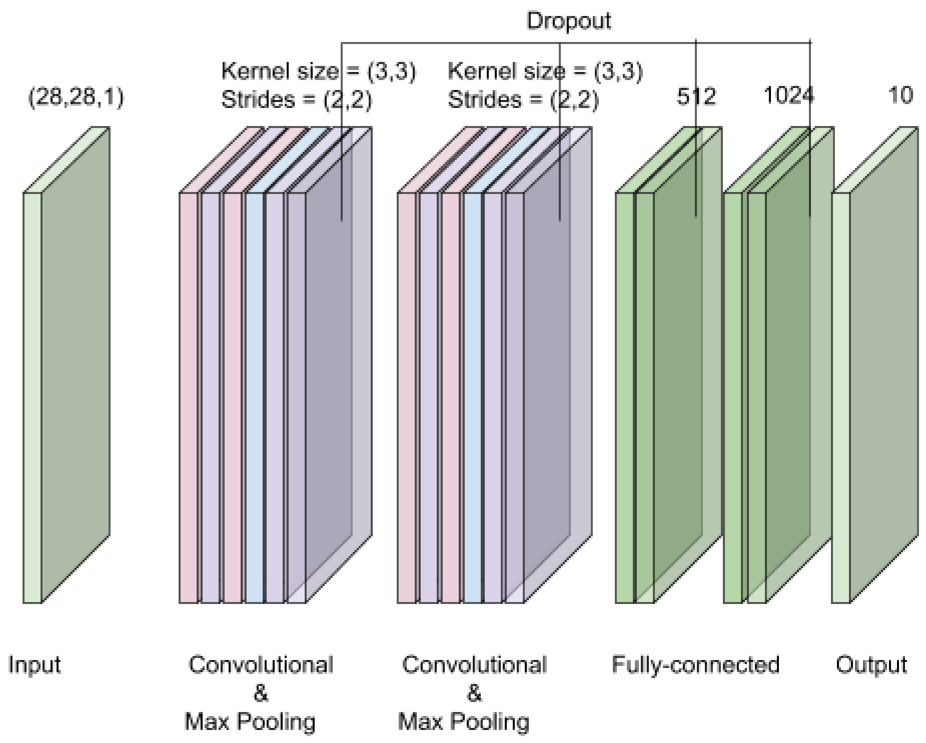
Because we adopted the early stopping method, the training process stopped at the 8th epoch but the overall curve shapes seem acceptable. At the same time, the model achieved 0.9862 Test Accuracy and lowered the FOM to 0.0304667. We consider this refined CNN our optimal model.

V. Conclusions

We started from a basic Neural Network, tuned the parameters, then moved forward to Convolutional Neural Network, and adopted Image Augmentation to artificially expand the training size. We finally yield a 0.0305 FOM. There are several conclusions for our model.

The Final Network Topology

We used 2 layers of convolutional networks, corresponding 2 max pooling layers with kernel size (3,3) and stride step (2,2), 2 fully connected hidden layers of 512 and 1024 neurons respectively to construct the model. Please see the graph below for more intuitive illustration of model topology.



Number of Training Data Used

Finally, we are able to use **1K** training samples out of the original 60K to get get reasonably good result. The percentage of training samples to the 60K is:

P1 = 1,000/6,000 = 0.01667

Test Accuracy on Trained Network

Using the 10K testing set, we got the test accuracy of:

P2 = 0.9862

Parameters Tuning

For the basic 2 hidden layer NN model, we found that when training size = 2,000 (out of 6,000), learning rate = 0.5, number of epochs = 25, validation split = 0.05, and dropout rate = 0.3 would deliver the optimal result.

However, when we change the architecture by adding convolutional layers, the original optimal parameters are no longer the optimal ones for the new network.

For the final CNN model, we used training size = 1,000, learning rate = 0.05, number of epochs = 30, validation split = 0.1, dropout rate = 0.25 to get the optimal result.

VI. Lessons Learned

From this project, we have learned that one of the key successful elements in teamwork is about delegation. A team that works well together understands the strengths and weaknesses of each team member. One of the benefits of our strong teamwork was that team members became proficient at dividing up tasks, and therefore the tasks were accomplished by the most qualified people in the shortest time.

We also developed deeper understanding regarding the project management skills. For example, we learned the importance of defining a plan and a schedule, enforcing and encouraging strong teamwork, maximizing resources and integration, and managing change and quality.

From the project itself, we learned that the dynamics of model training is more complicated than controlling the experiment. The various difficulties constantly occurred in the model training process requires promising attitudes and problem solving capabilities in training machine learning models.

Lastly, we found that the computational capability limitation could influence the process of training models. Since we only used CPU to run the models instead of GPU, when we added another layer to the convolutional neural network model, we could not run on all the optimized parameters due the running speed issue.

Reference

LeCun, Y., Cortes, C., Burges, C. The MNIST Database for Handwritten Digits. http://yann.lecun.com/exdb/mnist/

Foody, G. M., McCulloch, M. B., & Yates, W. B., The effect of training set size and composition on artificial neural network classification. *International Journal of Remote Sensing*. 1995 (16).

Lawrance, S., Giles, C. L., Tsoi, A. C., What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation. *Technical Reports from UMIACS*. Retrieved from: http://hdl.handle.net/1903/809.

Kandil, N., Khorasani, K., Patel, R.V., Sood, V.K., Optimum learning rate for backpropagation neural networks. Canadian Conference on Electrical and Computer Engineering, 1993.

Krähenbühl, P., Doersch, C., Donahue, J., Data-dependent Initializations of Convolutional Neural Networks. *Computer Vision and Pattern Recognition*. 2016

Sebastian Raschka. One-Hot Encoding Documentation. Github.com. Retrieved from: https://rasbt.github.io/mlxtend/user_guide/preprocessing/one-hot_encoding/

LeCun, Y., Bengio, Y., Convolutional Networks for Images, Speech, and Time-Series. Retrieved from: http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf

Rethinking the inception architecture for computer vision (2016), C. Szegedy et al.

C. Szegedy et al. Inception-v4, inception-resnet and the impact of residual connections on learning (2016)

K. He et al. Identity Mappings in Deep Residual Networks (2016)

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition (2014)

A. Krizhevsky et al. ImageNet classification with deep convolutional neural networks (2012)