Deep Learning Miniproject 1

Xiao Zhou 294916
xiao.zhou @epfl.ch

***Abstract*—The objective of this project is to test different architectures to compare two digits visible in a two-channel image in MNIST data set, and the final target is a binary classification. I have tried several basic neural network models and methods for improvement. Later I combine the model with features from actual digit class. After several attempts, the best method can reach an accuracy of over 96% in test set after 50 epochs.**

I. Introduction of Project

The project is based on MNIST data set with corresponding labels for each image of digit. The training input contain 2000 pairs of 2*14*14 tensor, each of which includes 2 digit grayscale images of size 14*14. During training, 1000 pairs form a training set while the other form the testing set. The goal of the project is to implement different neural network architectures to predict whether the first digit of the pair is no larger than the second one. The target contain 2000 binary labels for the series of pairs. The binary labels for training set are unbiased since 52.6% are labeled as positive and the other negative. Besides, the input class of each pair of digits are also provided. Therefore, the problem can be treated as a binary classification.

II. Data Preparation

To estimate the variance and test the robustness of each model, I randomize the order of data set each time before training. In case of ill-conditioned distribution, I implement standardization of each 2*14*14 digit image pair with the mean and standard deviation of the training set. It's assumed that the distribution of each pair conforms to a reasonable distribution and helps to improve model performance. All the training and testing data later have already been implemented with data normalization.

III. Implementation of Different Basic Model Architectures

A. Multi-Layer Perception

Multi-layer perception (MLP) models are used as a baseline. After simple parameter tuning, I set the number of hidden layer neurons to 100, which proves to reach an acceptable performance without taking a long time for training. I try 2 such models, 1-layer classifier and 3-layer MLP. When increasing the number of hidden layers of MLP, the overall performance has slightly improved without large fluctuation. There will not be an obvious increase if the layer number exceeds 3 and a 3-layer MLP takes less parameters and training time. The accuracy at testing set can reach 77% after 50 epochs.

B. Simple Convolution Neural Network

In this section, I try 2 similar convolution neural network (CNN) models, with reception field of one model shrinking fast while the other without rapid shrinking. They both contain 2 convolution layer of a kernel size of 3, followed by 2 linear fully connected layers. There are 8 and 16 channels after these 2 convolution operations. Relu is chosen as the main activation function. The first CNN model has no padding operation in convolution layer and a max-pooling layer after each convolution layer. According to the architecture, the receptive field for next layer is shrinking fast. The accuracy at testing set is around 76% .

In the second CNN layer, there is no max-pooling layer after the first convolution layer and there exists a padding operation of size 1, which prevents the receptive field from shrinking fast and helps keep more feature information. As a result, this help improve the performance on the basis of previous model and the accuracy can get to 78%, highest of the basic models.

C. Implementation of Basic Models

1) Optimizer Choice: After comparisons of 2 typical optimizers (especially Adam and SGD) and corresponding hyperparamters such as learning rate and momentum. Adam has a better convergence speed and final accuracy in most cases. In comparison of basic models and basic optimization methods in III, SGD is used and the emphasis is on the exploration of different basic structures. In Section IV and Section V, the model parameters are updated by Adam for the accuracy improvement driven by different learning strategies. All experiments are implemented in a mini-batch fashion.

2) Weight Initialization: In all of the experiments, I used normal distribution initialization for trainable parameters instead of default method in Pytorch, so that they can conform to a Gaussian distribution.

D. Results of Basic Models

According to Figure 1, the second CNN model has the best performance among all basic models, with least cross entropy and highest accuracy in both training and testing sets. The later optimization of architectures is based on this model.

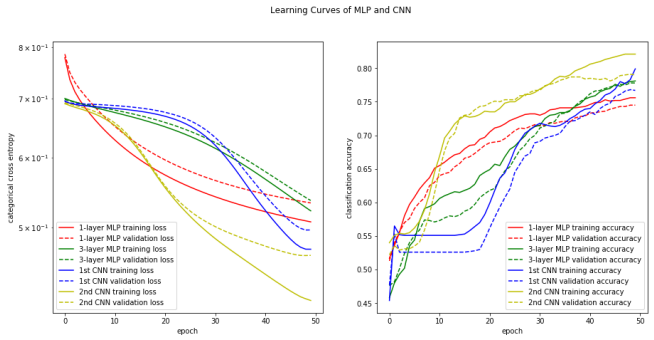

Figure 1: Learning Curves of Basic Models **(validation = testing)**

IV. Optimization of Model

A. L1 and L2 Regularization

It can be observed through Figure 1 that there exists a huge gap between training set and testing set, and the loss of testing set begins to slightly increase after around 20 epochs with accuracy not increasing anymore. It's assumed as a phenomenon of overfitting and thus I tried some regularization tricks. However, an auxiliary loss of L1 and L2 regularization does not bring an obvious improvement in performance. Thus, below I will try other methods, including dropout and batch normalization.

B. Dropout: DP-model

A dropout layer helps increase independence between units and distributes the representation. In general, it prevents co-adaptation by making the presence of other hidden units unreliable and improves the performance suffering from overfitting. I add dropout with probability of 0.4 after 2 convolution layers and the gap between training and testing set decreases.

C. Batch Normalization: BN-model

Batch normalization shifts and rescales according to the mean and variance estimated on the batch on a layer,thus making the layer learning independently from other layers. Sometimes it can also replace dropout layer. Batch normalization is added after 2 convolution layers in the model. As a result, it alleviates overfitting and the accuracy of testing set increases by 2%.

D. Change of Activation Function

The activation function of models is defaulted as Relu since it can reach a tradeoff between low computational cost and low gradient vanishing problem. In this section, I try to change the activation to sigmoid. As a result, the testing accuracy stays at 52.6% (the percentage of positive labels). I assume it's due to the property of sigmoid that the gradient will be close to zero at two sides of the function, which means the model cannot learn any more. By adding batch normalization, the problem can be alleviated and the performance has improved a lot, but it cannot reach the level of Relu activation in the case. In theory, Tanh, Leaky ReLU and ELU also have their merits in learning process, but they won't lead to significant difference in the project. Thus, ReLU will be used in next Section.

E. Going Deeper with Residual Block: Deep Res-model

Intuitively, a deeper network helps to extract high-level features and improves training performance. However, it may also face the problem of gradient vanishing. I try to add 6 convolution layers with kernel size of 3, padding of 1 and channel kernels of 16 as well as batch normalization before fully connected layers in the model. It turns out to have a phenomenon of gradient vanishing without other operations.

A residual block helps realize learning of deep net by introducing dierence between the value before the block and the one needed after. In my model, I add a residual block among every 2 convolution blocks. It turns out to help improve gradient vanishing problem and bring an increase in overall performance.

F. Results of Different Optimization Methods

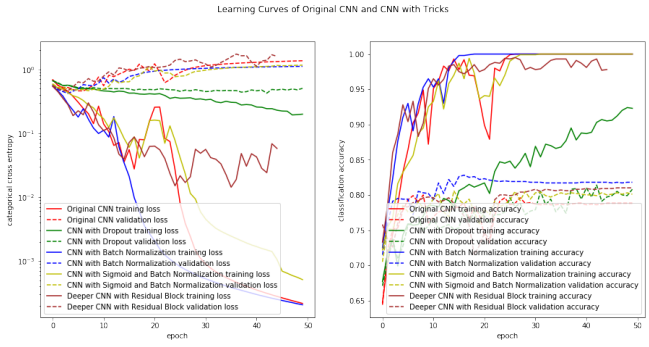

Figure 2: Learning Curves of Original CNN and CNN with Different Optimization Methods **(validation = testing)**

Among the tricks I tried, the batch normalization serves as the most apparent way to improve model performance. With combination of these tricks, the test accuracy can rise to over 80% in binary classification. However, I seek new way to improve the performance by adding the information of digit class in each pair. It is worth noting that the deeper model with residual block is stopped early due to the limited memory of virtual machine, thus it's assumed to achieve better performance with more epochs.

V. Adding the Information of Digit Class

A. Weight Sharing and Auxiliary Loss

The digit class information can help extract valuable features , which are ignored in previous simple binary classification. Thus, I implement the same convolution layers and fully connected layers as well as activation function, max pooling batch normalization and dropout, to extract information of 2 digit in each pair through weight sharing.

Loss for each digit classification is available, and then concatenate the 2 feature maps to finish to final binary classification. The brief architecture of the improved model is as follows. (I do not show the layer of activation, pooling, dropout and batch normalization for simplicity, which exist in my model)

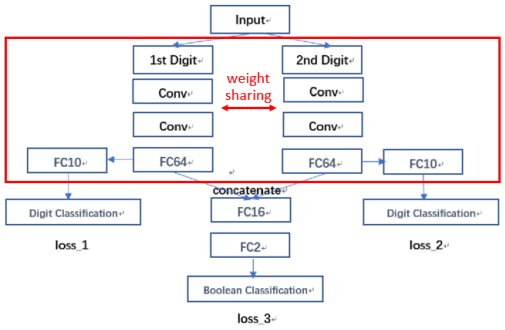

Figure 3: Structure of Weight-sharing CNN Model

The new loss is defined as follows, where L is the new total loss, L_3 is the binary cross entropy loss and L_1 and L_2 are the loss for digit classification in each pair.

$$L = \alpha \times L_3 + \beta_t \times \gamma \times (L_1 + L_2)$$

When I only consider the binary cross entropy loss without auxiliary loss of digits (i.e. set γ to 0), there is a small improvement. However, when I take into account the digit loss, the model of weight sharing can make a bigger difference. Also, since the final target is a binary classification, I use a weight decay of β_t with respect to epochs on auxiliary loss. In the case, after I set initial α to 1 and γ to 5, the test accuracy can rise by another 3%.

B. Training Directly on Digit Class: Direct-digit model

Digit class information has proved to improve the model performance. With the same optimization methods before, I make another attempt, where I ignore the binary classification at training and only focus on the digit classification in each pair. In other words, I set γ in the equation above far larger than α, and there is no weight decay for γ. The final binary classification is realized by directly comparing 2 predicted digits in each pair. This method has the best performance in the project, reaching around 95%.

C. Result of Combination with Digit Class

From the figure, the model with weight sharing and auxiliary loss still shows a trend of increase after 30 epochs but stopped due to the limited memory of virtual machine and it's assumed to achieve better performance with more training time. It can be concluded that the combination with digit class by weight sharing and auxiliary can help improve the model performance, and the training directly on digit class get the best performance in the project.

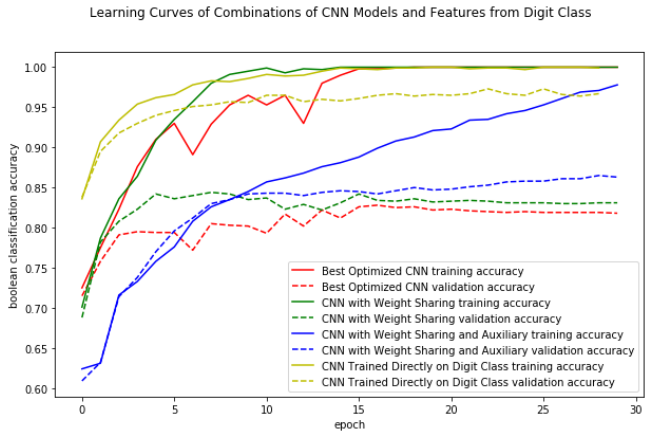

Figure 4: Learning Curves of Accuracy of Combinations with Digit Class **(validation = testing)**

VI. Summary of Different Models

Here I summarize the typical model mentioned above. To evaluate the robustness of the models, I run each model 10 rounds with randomized data and weights initialization.

Model (Optimizer)	Average Testing Accuracy	Standard Deviation
1-layer Perception (SGD)	0.742	0.027
3-layer MLP (SGD)	0.771	0.016
1st CNN (SGD)	0.765	0.014
2nd CNN (SGD)	0.789	0.010
2nd CNN (Adam)	0.804	0.012
DP-model (Adam)	0.815	0.016
BN-model (Adam)	0.822	0.011
Deep Res-model (Adam)	0.818	0.020
WS-model (Adam)	0.842	0.015
WS-AL-model (Adam)	0.866	0.023
Direct-digit model (Adam)	0.965	0.011

VII. Conclusion

The objective of the project is to explore different architecture changes that can help improve model performance. From several neural networks model I have implemented, I can conclude that proper simple CNN model is a good basic model. Regularization method like dropout and batch normalization can help alleviate overfitting and improve model performance. Residual block can help with gradient vanishing problem in deeper net. Finally, in the project, combination with digit class can make a difference in model performance. My best model is the network trained directly on digit class, which can reach around 96% of accuracy.

The main emphasize is put on the exploration of model structures and most hyperparameters are set as default values. Thus, there is still much that can be done to improve my model. In the future, I can implement data augumentation , try other tricks like early stopping and do more precise hyperparameter tuning, especially learning rate, weight decay and loss ratio in weight sharing model.