Deep Learning Project1 Report

Chenkai Wang
323452
chenkai.wang@epfl.ch

Siran Li
321825
siran.li@epfl.ch

Shijian Xu
327448
shijian.xu@epfl.ch

***Abstract*—In this project, we aim to test different architectures to compare two digits visible in a two-channel image. We use two main architectures: Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN). For each type of the models, the effect of the detailed structure choices, like activation types, dropout rates, etc., are investigated. Besides, we also assess the performance improvement that can be achieved by incorporating weight sharing and auxiliary loss.**

I. Introduction

This project aims to implement a deep network that can be used to predict for a two-channel digits image that if the first digit is lesser or equal to the second. More specifically, the data are generated from MNIST by first downsampling the image size and then concatenate two digit images together to form a $2 \times 14 \times 14$ tensor. The label is 0 or 1, indicating whether the first digit is larger, which results in a binary classification task.

We choose two types of deep networks: MLP and CNN. For each model, we first search for the best architecture via extensive experiments and determine the best architecture configurations. Then, based on the selected best architecture, we evaluate the effect of different activation functions, dropout rates, and batch normalization on the final performance. We then make decisions whether to apply these tricks to the final models based on the evaluation results. After that, we test the performance improvements achieved by using weight sharing and/or auxiliary loss.

II. Experiment Design

A. The Basic Structure

To simplify the architecture searching space, we pre-define that the MLP feature extraction part consists of 4 linear layers. The classifier parts of the MLP are composed of 1 linear layer. Activation layers are added at appropriate positions. For CNN, we pre-define that the ConvNet feature extraction part consists of 3 convolutional layers. The classifier parts of the ConvNet are composed of an MLP with 1 hidden layer. Max pooling layers are used in the feature extractor. The detailed structures are listed in Table I.

Module	MLP	ConvNet
Feature Extractor	Linear ReLU Linear ReLU Linear ReLU Linear ReLU	Conv2d ReLU MaxPool2d Conv2d ReLU MaxPool2d Conv2d ReLU MaxPool2d
Binary Classifier	Linear Sigmoid	Linear ReLU Linear Sigmoid
Digit Classifier	Linear	Linear ReLU Linear

TABLE I: Basic Structures of the models.

B. Experiment Protocol

We conduct the experiments following the below experiment protocol. First, we define the vanilla MLP and vanilla ConvNet. The vanilla MLP uses *ReLU* activation function. No *dropout* layer is used. Besides, we do not incorporate the weight sharing and auxiliary loss into this model. Similar to the vanilla MLP, the vanilla ConvNet uses *ReLU* activation function. No *BatchNorm* layer is used. No weight sharing and auxiliary loss. The illustration of these vanilla models are shown in Fig. 2.

Based on the vanilla models, the first experiment is searching for the best architectures. Since we have pre-defined the depth of these models, what we need to search are **hidden layer dimensions** for MLP and **channel numbers** for ConvNet. We determine the best architecture based on the test accuracy. Second, we conduct experiments to determine the best activation function. The candidate activation functionsa are *ReLU*, *Tanh*, *Sigmoid* and *SELU*. After determining the activation function, we then evaluate the effects of different dropout rates for MLP. For ConvNet, we evaluate the effect of adding *BatchNorm*.

After determining the network structure via the above experiments, we assess the performance achievements by using weight sharing and auxiliary loss. The network structure with weight sharing and auxiliary loss is shown in Fig. 3.

III. Experiment Results

A. Searching for the best architecture

In this part, we optimize the network parameters to obtain the possible optimal architecture. We achieve a list of all possible combinations from the values in [16, 32, 64, 128, 256, 512], which yields the number of trainable parameters between 65000 and 75000. For each of these sets of parameters, we train the corresponding network (MLP and CNN) 10 times. During the training, the learning rate is fixed to 0.001 and the batch size is 100. The searching results are shown in Fig. 1.

From the results, we can see that the combination [128, 64, 32, 256] achieves the best performance for MLP and the

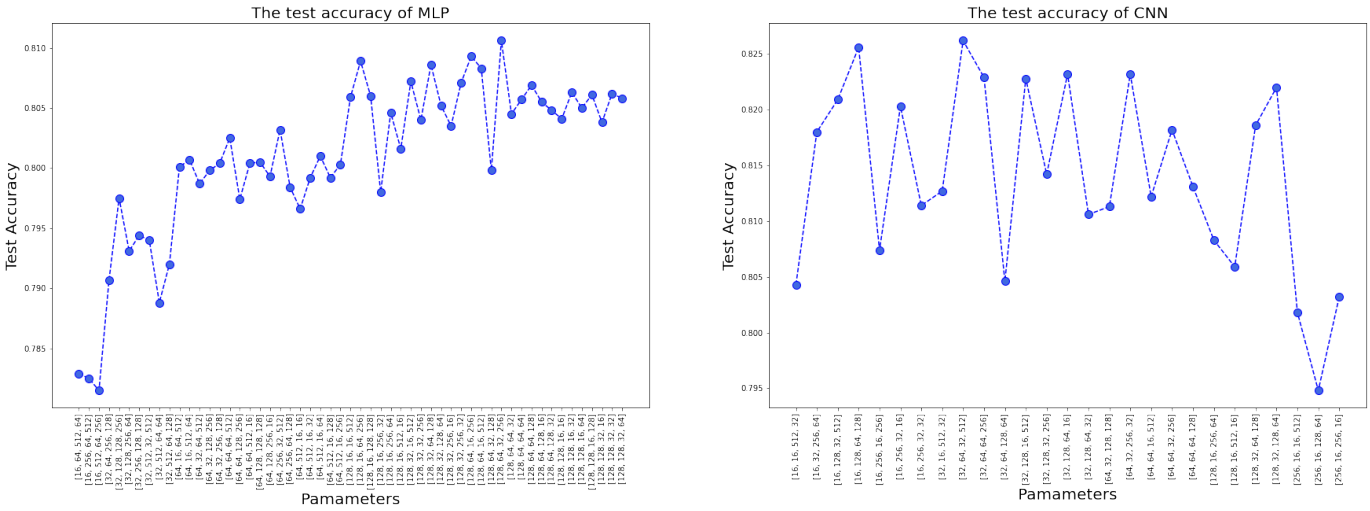


Fig. 1: The test accuracy of MLP and CNN under different parameters.

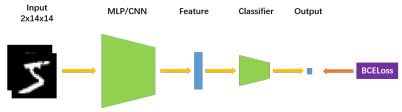


Fig. 2: The vanilla network structure.

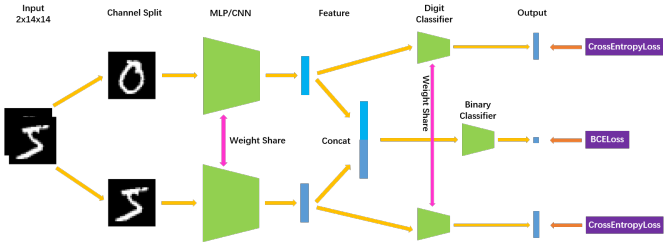
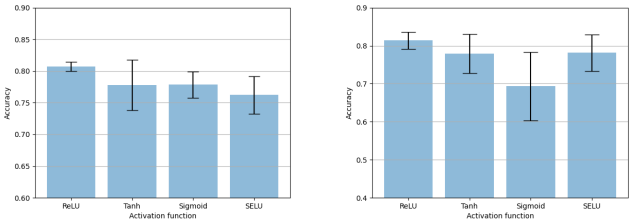


Fig. 3: The network structure with weight sharing and auxiliary losses.

combination [32, 64, 32, 512] achieves the best performance for ConvNet.



(a) MLP (b) ConvNet

Fig. 4: The accuracy of choosing different activation functions.

B. Searching for the best activation function

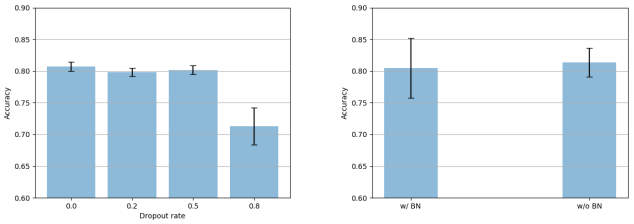
Based on the results in III-A, we continue to search for the best activation function. The candidate activation functions are *ReLU*, *Tanh*, *Sigmoid* and *SELU*. For MLP, the hidden layer unit numbers are set as: 128, 64, 32, 256. For ConvNet, the channel numbers for each conv layer are set as: 32, 64, 32, and the hidden dimension for the classifier is 512. We test for 10 rounds for each activation function choice, and the results are shown in Fig. 4. From the results we can see that for both MLP and ConvNet, the *ReLU* activation function always achieves the best performance. During the experiments, we also found that training model with *Sigmoid* activation function requires much larger learning rate than other activation functions ($lr_{sigmoid} = 0.01, lr_{others} = 0.001$). This might be because *Sigmoid* activation function is much easier to be saturated and the returned gradient will be very small. Hence, to be effectively trained, a larger learning rate is needed. In other words, *ReLU* activation function alleviates the gradient vanishing problem [1] in *Sigmoid* and boosts the performance. In the following experiments, we will stick to use *ReLU*.

C. Effect of Dropout and BatchNorm

Dropout is usually applied on fully-connected layers, but seldom be used on convolutional layers. It is widely accepted that dropout has limited benefits when applied to convolutional layers and the possible explanation is that activation units in convolutional layers are spatially correlated and information can still flow through convolutional networks despite dropout [2]. So, in this section, we only evaluate the effect of adding dropout layers to MLP. For ConvNet, we evaluate another trick, which is Batch Normalization.

To evaluate the effect of dropout layer, we test with different dropout rates, which are 0, 0.2, 0.5 and 0.8. Each choice is evaluated for 10 rounds. The results are shown in Fig. 5.

From the results, we can see that adding Dropout to MLP doesn't improve the performance. Instead, the MLP model performs worse compared with not adding Dropout. Ideally, Dropout can be used to prevent overfitting [3]. We hypothesize the explanation is that the shallow MLP model has low capacity, and adding dropout will further reduce its learning



(a) MLP-Dropout (b) ConvNet-BatchNorm

Fig. 5: The accuracy of applying Dropout to MLP and BatchNorm to ConvNet.

ability. Usually, BatchNorm is believed to mitigate the problem of internal covariate shift and improve the final performance [4]. However, we can also observe that adding BatchNorm doesn't boost the performance in our case. There must be many factors leading to that result. We hypothesize that this is because our model is too simple and does not benefit from it.

D. Effect of Weight sharing and Auxiliary loss

From the above experiments, we have the conclusion that we should use ReLU activation and should not apply Dropout or BatchNorm to these models. Based on these observations, we will evaluate the effect of weight sharing and auxiliary loss.

Weight sharing simply splits the input 2-channel image into two 1-channel images and then processes them separately with the same feature extractor. The extracted two features are then concatenated together to pass through the classifier, as shown in Fig. 3. This structure is similar to the Siamese network, which was proposed by Yann LeCun *et al.* in 1993 [5]. The difference is, originally Siamese network is used to compare the feature similarity between two different inputs. In our case, we just concatenate the features of two different images.

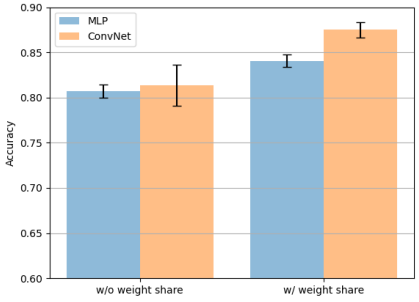


Fig. 6: The effect of weight sharing for MLP and ConvNet.

The results are shown in Fig. 6. From this figure, we can see that weight sharing improves the performance greatly for both MLP and ConvNet. We hypothesize that decoupling the input 2-channel image reduce the complexity of the problem, making the network more easy to learn.

From the above experiments, we get that weight sharing is very useful. Then we want to evaluate the effect of auxiliary loss based on weight sharing. The idea is that the extracted two features are passed through the digit classifier separately to predict the corresponding digit labels (0-9), and the concatenated features are passed through the binary classifier to predict the binary label (0/1). In total it will produce 3 losses and we add these losses together for backward propagation. Apart from naively adding 3 losses together, we also evaluate the trade-off between the binary classification loss and the digit prediction loss. So the final loss is: $loss = loss_{digit1} + loss_{digit2} + \lambda loss_{binary}$, where λ ranges from 0.5 to 1.5, with step size 0.1. The results are shown in Fig. 7.

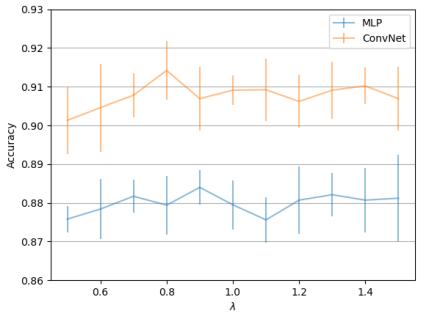


Fig. 7: The effect of auxiliary loss for MLP and ConvNet.

From this figure, we can see that auxiliary loss further improves the performance. For MLP, the best performance is achieved when $\lambda = 0.9$. For ConvNet, the best performance is achieved when $\lambda = 0.8$.

In fact, this is a kind of multi-task learning [6], in which multiple learning tasks are learned at the same time. The commonalities and differences across tasks can be efficiently exploited to improve the final performance. In our case, the two different tasks are digits classification and number value comparison (binary classification).

IV. Conclusion

In this project, we compare different model architectures and assess the performance improvement achieved by weight sharing and auxiliary loss. Now we have a better perspective of the influence of different parameters on neural networks' performances.

In summary, we find that ConvNet with ReLU activation can achieve the best performance for this binary classification problem. And weight sharing and auxiliary loss have a positive effect on both MLP and ConvNet. As said above, we choose the combination of parameters [32, 64, 32, 512] and use ReLU as our activation function. Finally, we achieve a CNN model with an accuracy of 91.36% (the mean of 10-time training with a standard deviation of 0.006275), using weight sharing and auxiliary loss.

REFERENCES

[1] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[2] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Dropblock: A regularization method for convolutional networks," *arXiv preprint arXiv:1810.12890*, 2018.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[5] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," *Advances in neural information processing systems*, vol. 6, pp. 737–744, 1993.

[6] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.