Effect of weight sharing and auxiliary loss on different architectures

Jiaan ZHU
jiaan.zhu@epfl.ch

Menghe JIN
menghe.jin@epfl.ch

Yueqing SHEN
yueqing.shen@epfl.ch

***Abstract*—The implementation of weight sharing and the adaptation of auxiliary loss are widely regarded as powerful tools in facilitating the optimization of deep learning models. In the project, three architectures are selected for comparison, key characteristics such as accuracy and time consumption are studied in altogether twelve models with or without weight sharing and auxiliary loss to obtain an insightful understanding of their effects. The optimizing processes of individual models are also disclosed to ensure a fair comparison among their performances.**

I. Introduction

The goal of this project is to compare different model architectures in recognizing which of two handwritten digits is greater, and to assess the effect of weight sharing [1] and auxiliary loss [2]. Both the given training set and test set consists of 1000 pairs of handwritten digits of 2*14*14 size. Besides the boolean values of train_target, the classes of the two digits in each pair are also provided to facilitate the implementation of auxiliary loss.

In the project, we test the performance of twelve models based on three architectures and try to explain the results. Section II introduces the architecture of models and the details of training and parameter optimization. Section III provides details of our results and discussion. Section V concludes our work as well as possible future steps.

II. Methods

A. model architectures

Since our principal intention is to investigate the effect from the implementation of weight sharing and auxiliary loss, three relatively simple architectures are selected: (1)a three layers multilayer perceptron, two hidden fully connected layers each followed by a Relu and an output layer followed by a Sigmoid; (2)a five layers convolutional neural network, two convolutional layers and two fully connected layers each followed by Relu, and an output layer followed by Sigmoid; (3) a ResNet with 4 'Resnet blocks', the structure of ResNet is reused from the Practical Session 6, except for the average pooling step where a smaller parameter is used due to shrinking size of our inputs. For comparison, each model has similar number of parameters(around 70,000).

As is illustrated by **Fig.** 1, models are designed and tested according to the following "formula": architecture, architecture + weight sharing, architecture + auxiliary loss, architecture + weight sharing + auxiliary loss. To simplify

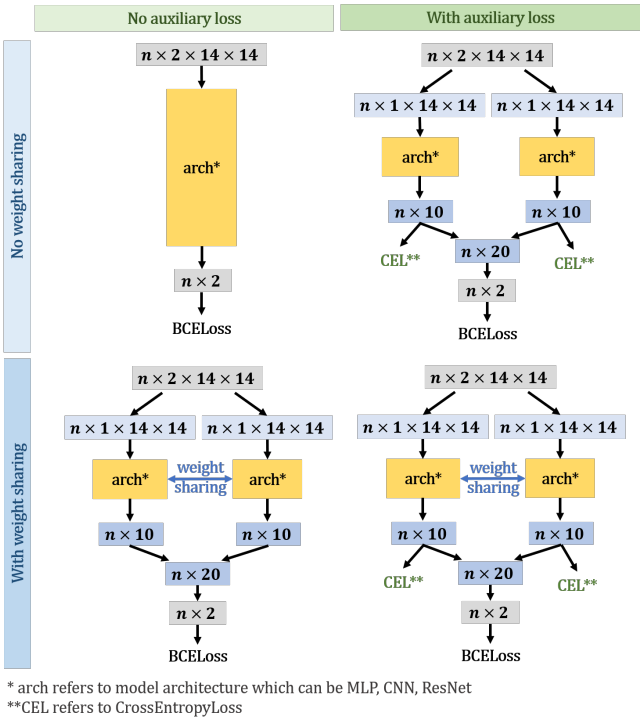


Fig. 1: Model design

the notation, 'MLP' stands multilayer perceptron, 'BaseNet' stands for convolutional neural network, 'ResNet' stands for resnet. The prefix 'Aux' and 'Siamese' stand for auxiliary loss and weight sharing respectively. For models with auxiliary loss, the second last layer is followed by a Softmax in each path, and then compared with train_classes to produce an auxiliary output. The input size and output size of each model are the same($n * 2 * 14 * 14$ and $n * 2$ separately).

B. training

All the models are implemented in pytorch 1.7.1. During training, we use Adam optimizer and 25 epochs. Since the size of train set is pretty small, 25 epochs are enough for convergence. The hyper-parameters like learning rate and batch size are optimized by grid search(**??**). Binary Cross Entropy(BCE) loss is used for calculating the main loss and for models with auxiliary loss and Cross Entropy(CE) loss is used to calculate the auxiliary loss. For models with auxiliary loss, the total loss is calculated as $BCEloss + \alpha * CEloss$, where

TABLE I: Optimization results

model	learning rate	batch size	α	number of parameters
MLP	$5 * 10^{-4}$	8	-	73314
SiameseMLP	$5 * 10^{-3}$	8	-	33172
AuxMLP	$5 * 10^{-3}$	16	0.9	66302
AuxsiameseMLP	$5 * 10^{-3}$	8	0.7	33172
BaseNet	$5 * 10^{-4}$	16	-	72536
SiameseBaseNet	$1 * 10^{-3}$	16	-	72268
AuxBaseNet	$5 * 10^{-4}$	8	1.0	134766
AuxsiameseBaseNet	$5 * 10^{-3}$	32	0.6	72268
ResNet	$1 * 10^{-3}$	32	-	75746
SiameseResNet	$5 * 10^{-3}$	32	-	76034
AuxResNet	$5 * 10^{-3}$	32	0.6	153804
AuxsiameseResNet	$5 * 10^{-3}$	32	0.6	78924

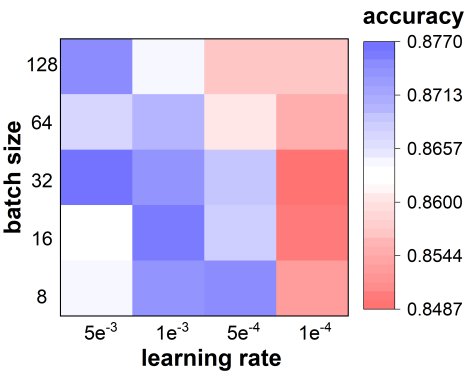
α is the coefficient of auxiliary loss. Each model is trained 10 times with randomized data and weight initiation and the mean accuracy and its standard derivation are recorded.

The training of models is done on a single GTX 1080 Ti GPU and the final results are validated in virtual machine provided in the course.

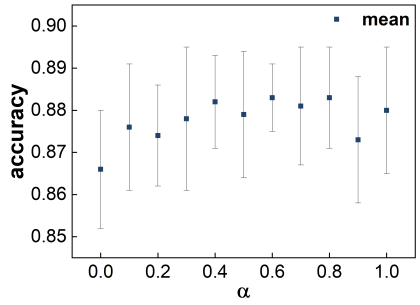
C. hyper-parameter optimization

In order to compare the model performances fairly without biases, parameters of each model are carefully optimized by altering hyper-parameters such as learning rate and batch size to achieve the best possible accuracy while models with auxiliary loss undergo an additional adjustment in the coefficient of auxiliary loss. Global optimal solution could be achieved by grid search, however, to leverage time and computing power, local optimal was obtained following a learning rate&batch size - α consecutive optimization order. Take the AuxsiameseResNet architecture as an example. **Fig 2** presents the process of optimizing this model. First, the optimal learning rate and batch size are selected based on the accuracy of test set by grid search as shown in figure 2(a). Then the optimal α is selected as shown in figure 2(b), and finally the change of loss against the epoch is plotted to check the convergence(figure 2(c)). In this way, the best hyper-parameters of this model are obtained as follows: learning rate: $5 * 10^{-3}$, batch size: 32, and α: 0.6.

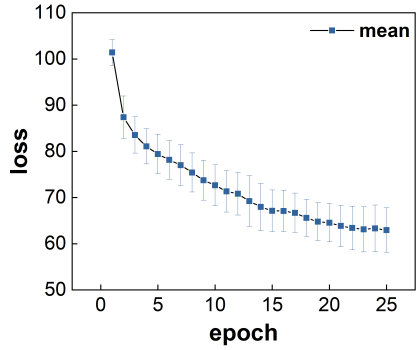
The optimization results and the number of parameters in each model are shown in table1. These parameters are then selected to test the performance of each model.



(a) Tuning learning rate & batch size



(b) Tuning alpha



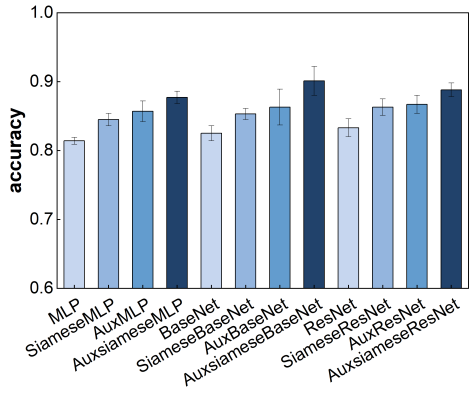
(c) Checking convergence

Fig. 2: An example of optimization process for (a)best batch size and learning rate; (b)auxiliary coefficient in AuxsiameseResNet. model; (c)check the convergence of the training

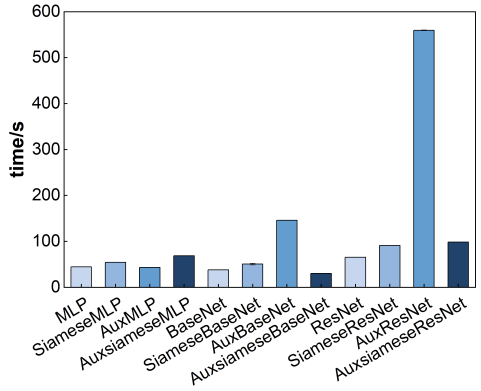
III. Results and Discussion

Each model is trained 10 times and the corresponding mean accuracy and the mean time consumption are recorded. The CNN with both weight sharing and auxiliary loss achieves the highest accuracy(0.901 ± 0.021). As shown in the figure3, the accuracy of models without weight sharing and auxiliary loss are all below 0.85. However, as one of weight sharing and auxiliary loss is added, the accuracy excels 0.85. For each architecture, the models with both weight sharing and auxiliary loss achieves the highest accuracy, which shows the effect of these two modifications are addable. Most of models only need fewer than 100 seconds for training while AuxResnet takes the longest time. It's surprising that our best model, the AuxsaimeseBaseNet, is with the shortest training time(30.11 ± 0.261). We also notice that the training time on GPU is slower than on CPU, maybe due to the relatively small dataset and simple models we use in this project.

Another interesting discovery is that the performance of AuxsiameseResNet is worse than AuxsiameseBaseNet. In principle the ResNet is more powerful than the simple convolutional neural network even with the similar number of parameters. It may be due to the small size of our training set, and we propose that if we reduce the layers of ResNet, the



(a) Mean accuracy of each model



(b) Mean time consumption of each model

Fig. 3: Results: (a) mean accuracy of each model; (b) mean time consumption of each model

performance of ResNet could be better.

The process of tuning the hyper-parameters remains to be improved. As mentioned in the previous section, we don't use intact grid search due to limited computational power. In the step of tuning the learning rate and batch size, we only evaluate them by focusing on the accuracy of test set and it's also important to look at the standard derivation for checking the stability of each condition.

IV. Conclusion

According to the testing results presented in this report, the highest accuracy achieved on the test set is 0.901 by CNN with weight sharing and auxiliary loss. Though data augmentation could significantly improve the accuracy [3], we skip this pre-process step as the main aim of this project is to study the effect of weight sharing and auxiliary loss. As shown in the results[Ref], both weight sharing and auxiliary loss could improve the performance of models and their effects are addable.

References

[1] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural Computation*, vol. 4, no. 4, pp. 473–493, 07 1992.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[3] T. M. K. Connor Shorten, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, July 2019.