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.

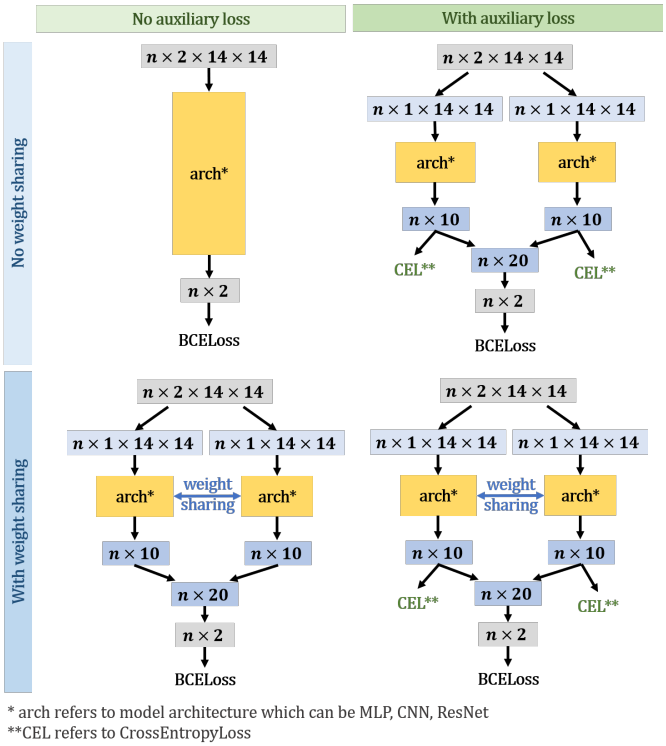


Fig. 1: Model design

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 parameter of average pooling. We set the pooling parameter to 4 here because if the parameter is larger than 4, more information will be lost and if the parameter is smaller than 4, noises will be introduced. 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 the notation, 'MLP' stands multilayer perceptron, 'CNN' 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

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
CNN	$5 * 10^{-4}$	16	-	72536
SiameseCNN	$1 * 10^{-3}$	16	-	72268
AuxCNN	$5 * 10^{-4}$	8	1.0	144494
AuxsiameseCNN	$5 * 10^{-3}$	32	0.6	72268
ResNet	$1 * 10^{-3}$	16	-	75746
SiameseResNet	$1 * 10^{-3}$	16	-	77812
AuxResNet	$1 * 10^{-3}$	64	0.7	152692
AuxsiameseResNet	$1 * 10^{-3}$	16	0.8	77812

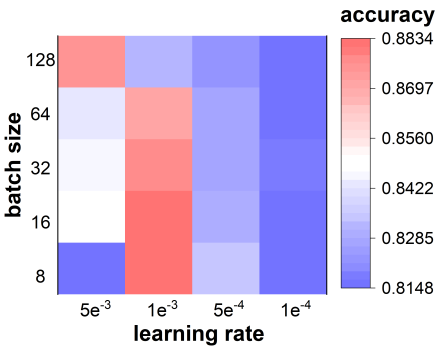
auxiliary loss and Cross Entropy(CE) loss is used to calculate the auxiliary loss. For models with auxiliary loss, the total loss is calculated as $BCEloss + \alpha * CEloss$, where α is the coefficient of auxiliary loss. Each model is trained 10 times with randomized data and weight initiation and the mean accuracy and its standard derivation are recorded.

The training of models is done on CPU and the final results are validated in virtual machine provided in the course.

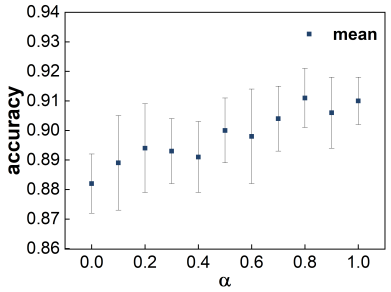
C. hyper-parameter optimization

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 for all hyper-parameter combinations. However, to leverage time and computing power, a two-step grid search strategy is used. The learning rate and batch size are optimized by first-round grid search, followed by another grid search of the α(coefficient of auxiliary loss) for the models with auxiliary loss. 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 **Fig.** 2a. Then the optimal α is selected as shown in figure **Fig.** 2b, and finally the change of loss against the epoch is plotted to check the convergence(**Fig.** 2c). In this way, the best hyper-parameters of this model are obtained as follows: learning rate: $1 * 10^{-3}$, batch size: 16, and α: 0.8.

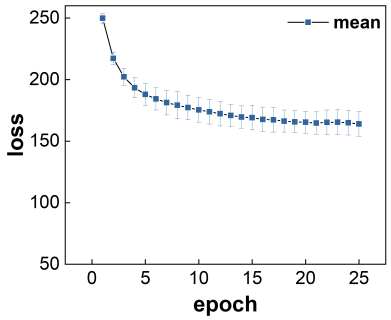
The optimization results and the number of parameters in each model are shown in **Table.** I. These parameters are then selected to test the performance of each model.



(a) Tuning learning rate & batch size



(b) Tuning alpha



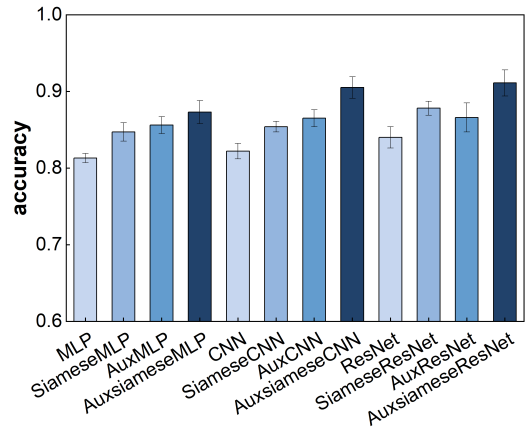
(c) Checking convergence

Fig. 2: Optimization process for models, take AuxsiameseResNet as an example: (a)best batch size and learning rate are found by grid search step 1; (b)best auxiliary coefficient is found by grid search step 2; (c)check the convergence of the training: 25 epochs are already enough for convergence

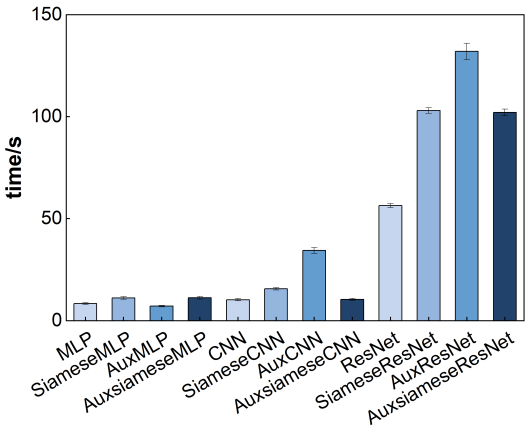
III. Results and Discussion

Each model is trained 10 times and the corresponding mean accuracy and the mean time consumption are recorded. The ResNet with both weight sharing and auxiliary loss achieves the highest accuracy(0.911 ± 0.017). As shown in the **Fig.** 3a, 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 exceeds 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(**Fig.** 3b). We also notice that the training on a GTX 1080 Ti GPU is slower than on CPU, maybe due to the relatively small dataset and simple models we use in this project, that's why we finally choose to test and train our models on CPU.

One interesting discovery is that the performance of AuxsiameseResNet is not significantly better than AuxsiameseCNN.



(a) Mean accuracy of each model



(b) Mean time consumption of each model

Fig. 3: Results: (a) mean accuracy of each model, from left to right: $0.813 \pm 0.006, 0.847 \pm 0.012, 0.856 \pm 0.011, 0.873 \pm 0.015, 0.822 \pm 0.010, 0.854 \pm 0.007, 0.865 \pm 0.011, 0.905 \pm 0.014, 0.840 \pm 0.014, 0.878 \pm 0.009, 0.866 \pm 0.019, 0.911 \pm 0.017$, ; (b) mean time consumption of each model, from left to right(unit: second): $8.386 \pm 0.395, 11.090 \pm 0.619, 7.304 \pm 0.377, 11.222 \pm 0.601, 10.222 \pm 0.533, 15.610 \pm 0.626, 34.437 \pm 1.419, 10.415 \pm 0.513, 56.378 \pm 0.971, 102.985 \pm 1.366, 131.947 \pm 4.006, 102.064 \pm 1.599$.

In principle the ResNet is more powerful than the simple convolutional neural network even with the similar number of parameters. It may due to the small size of our training set. It's also surprising for us that AuxsiameseCNN reaches a comparable accuracy with AuxsiameseResNet while it cost much less training time(10.415 ± 0.513)s than the latter(102.064 ± 1.599)s.

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 addition, in the step of tuning the learning rate and batch size, we only evaluate them by focusing on the accuracy of test set. Although we have noticed for most of our tests, the standard derivation of accuracy in each condition is relatively small, it may also necessary 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 0.911 ± 0.017 by ResNet 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, 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.