Compare two digits with siamese network

FENG Wentao, SUN Zhaodong, WANG Yunbei
{wentao.feng, zhaodong.sun, yunbei.wang}@epfl.ch

***Abstract*—In this project, we first build a convolution neural network to recognize hand-written digits. Then, based on this model, we have created a siamese convolution neural network to compare two visible numbers. We also apply weight sharing and auxiliary loss to achieve the objective. Finally, we discuss the influence of those techniques.**

I. Introduction

In this project, we have 1000 pairs of 14×14 images of hand-written digits as the train set($1000 \times 2 \times 14 \times 14$ tensors) and another test set of the same size. All the images are transformed from the MNIST database. Every pair of images has two kinds of labels. The preliminary label is boolean format: 1 means the first digits is less or equal than the second one and 0 means the other situations. Auxiliary label is a list with two elements. Each element is the true digit corresponding to the image.

The main goal of this project is to build a deep net to compare two digits. Namely, it should judge whether the first digit is less or equal than the second one. The deep net should be trained with the given train set.

II. Single channel convolution network

We first implement a single channel convolution network as the baseline. The net is based on the classic model LeNet-5[1]. We tune this model to achieve good performance.

A. Modification on filters

The original LeNet-5 is designated to recognize the hand-written image in 32×32. In our case, the resolution of the image is set to 14×14. Therefore, we need to modify the convolution filter to fit this model into our case. We first increase the number of filters in the first convolution layer, because this layer get the original image as the input and extra filters can help to extract crucial information as much as possible. As a result, the number of the filter in the second convolution layer is increased to 64.

Then, we reduce the size of a single filter from 5×5 to 3×3. The small filter works better than the large filter in this project as the following reasons:

- Slowing the reduction of image dimension. From the definition of 2D convolution, the 5×5 filter will reduce the height and width of an image by 4 and the 3×3 filter only reduce the shape by 2. The resolution of our image is low. Hence the small filter is preferable, and thus we can increase the depth of the net.
- Having fewer parameters. We assume that the depth of the first convolution layer is 8. The number of parameter in that is $5 \times 5 \times 8 = 200$ for the filter with size 5 and $3 \times 3 \times 10 = 90$ for the filter with 3. If this layer is much deep, the gap will be large. Fewer parameters mean faster learning. Therefore, we decide to use a small filter to compensate for the additional training cost brought by increasing the filter quantity.
- Extracting local information. The large filter is used to extract the general information of input and small filter focus on local information. Our task is based on hand-written digits recognition. The edge information is important in this kind of task. The main idea of edge detection is to find the variation of pixel value, which is the local information of pixels on the border of the foreground and background.
- We skip all even-sized filter to avoid aliasing. Through convolution with filter, we encode the adjacent information of a pixel (people call this pixel the anchor point). Normally, we take the same number of pixels in 4 directions from anchor point into consideration. So, the filter size will be $2n+1$ (n is the radius from anchor point). If we use even sized filter, we cannot find an anchor point.

B. Choice on activation function

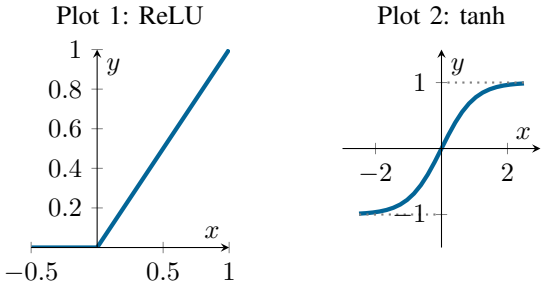
There are many popular non-linear activation functions such as *ReLU*, *sigmoid*, *tanh*. We firstly choose *ReLU* as activation function between hidden layers, because it has many nice properties. Specifically, unlike *sigmoid* and *tanh*, *ReLU* is much less likely to cause vanish gradient when the input becomes big. Secondly, *ReLU* encourage sparsity which is an important attribute to prevent over-fitting.

Then, we try to use this version to perform our task. However, it fails our expectation. From the train log, we find three points that worth attention:

1) The value of loss function decreases slowly or even no descending after an epoch.
2) The accuracy rate keeps constant around 10%, which means the network guesses results randomly.
3) The softmax output of the network is close to one-hot encoding style, and the hotkey rarely changes.

It is quite possible that our network suffers from the *dying ReLU*[2] problem. From the plot 1 in II-B, the *ReLU* becomes 0 when the input is non-positive. In our case, 90% of output neurons die, and they never come back to live

afterward. To address this problem, we have tried *Leaky ReLU*, *ELU* and to decrease learning rate. However, the most efficient way we tried to escape that is changing to *tanh*. As our network is not deep, the drawback of *tanh* is not influential, and, from the plot 2 in II-B, it can also add non-linearity as good as *sigmoid*. Therefore, *tanh* perfectly fits our task.



C. Other setup

As this is a classification problem, we apply the softmax function after the output layer and use the cross-entropy as the loss function. For the optimizer, we choose *Adam* with learning rate equals to 0.01.

D. Result

Figure 1 demonstrates the final structure of the single channel convolution network. We conduct tests on the original net and tuned net with data generated by `dlc_practical_prologue.py`. From the results in Table I, our convolution network performs better than the original one, and it can be further used in the next section.

Metric	Original net	Tuned net
train loss	2.38	1.54
test loss	0.02	0.02
test accuracy	0.10	0.89

Table I: The results after 5 epochs training

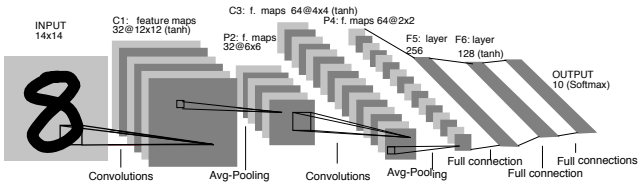


Figure 1: Our network structure (Note: for saving space, please refer the annotations for true setup for each layer)

III. Siamese Convolution Network

Siamese network normally contains two sub-networks. Here, the convolution network we obtain in the last section works as sub-network. At the final stage of the siamese network, a fully connected layer with softmax function, in Figure 2, gets the outputs of two sub-networks and yields two values that represent the probability of 1(first digits is less or equal than the second one) and 0(otherwise).

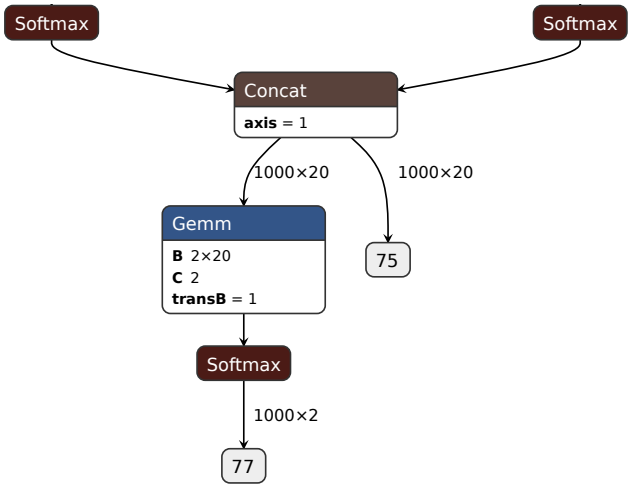


Figure 2: Combination layer and output layer

Besides adding the combination and output layers, We also set two parameters for users to customize model:

- `w_sharing` is a boolean variable to control the activation of weight sharing.
- `aux_labels` is a boolean variable to control the status of auxiliary loss.

To avoid over-fitting due to insufficient training samples, we add drop-out layers after each convolution layers. After hyper-parameter tuning, the drop-out ratio is 30%. Meanwhile, since the cross-entropy cannot be used with multiple targets problem, we encode all the labels with one-hot style and turn to binary cross entropy. The final architecture of siamese network is in Figure 7 in appendix.

IV. Results and Discussion

We test 20 rounds for all four combinations of two parameters and obtain the following results. The accuracy is measured by the percentage of right predictions and loss stands for binary cross entropy loss.

A. Disable weight sharing and disable auxiliary loss

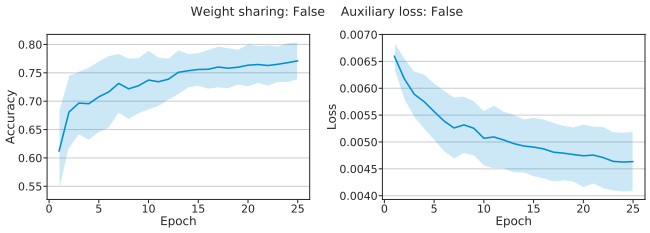


Figure 3: Test accuracy and test loss

The first setting is to disable both adjustable options. From the Figure 3, we can only get a test accuracy of

$78.26\% \pm 3.26\%$. Moreover, the loss curve behaves like non-monotony. Both accuracy and loss have a relatively large variation. Overall, without weight sharing and auxiliary loss, the model cannot achieve our requirement that the accuracy should above 85% and large variation also affects the reliability of the model.

B. Disable weight sharing and enable auxiliary loss

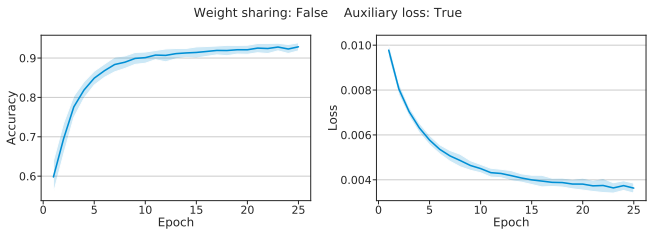


Figure 4: Test accuracy and test loss

When we add auxiliary loss, the most intuitive improvement in Figure 4 is that the variation reduces significantly. Meanwhile, the accuracy of predictions also becomes much better than the previous one. In this setting, we finally get a accuracy of $93.55\% \pm 0.81\%$, which is beyond our minimum requirement.

C. Enable weight sharing and disable auxiliary loss

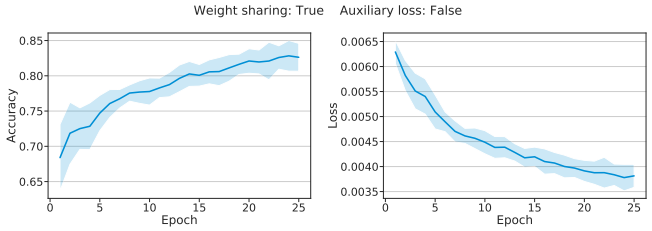


Figure 5: Test accuracy and test loss

In Figure 5, we find that weight sharing can also decrease some variance and improve prediction, but its influence is not as much as the auxiliary loss. In this test, the accuracy is $83.95\% \pm 1.51\%$.

D. Enable weight sharing and Enable auxiliary loss

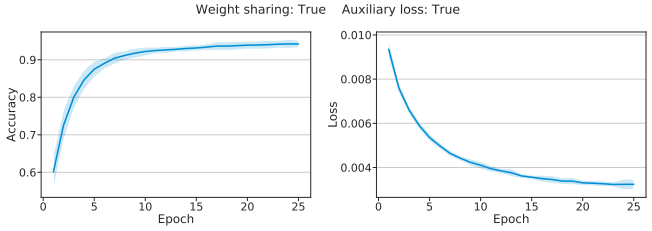


Figure 6: Test accuracy and test loss

When both weight sharing and auxiliary loss are activated, our model can absolutely achieve the best performance. The best accuracy is $94.84\% \pm 0.80\%$.

V. Summary

We choose the accuracy of no weight sharing and no auxiliary loss as the reference and calculate the relative performance in Table II by dividing the reference. From the comparisons in Table II, our model can benefit from both techniques. Since our training data are 1000 pairs of images, the model, whose weights shared between sub-networks, can be trained with two times the number of samples compared to that without sharing. More training samples means our model can generalize better and thus has a better prediction. The reason behind the improvement brought by auxiliary loss is intuitive. Additional labels increase the complexity of the model. Without digits labels, we can consider the task into a binary classification problem, which means our model is simple and under-fits the task. With the availability of the classes of the two digits, our task becomes a classification problem with $10 \times 10 \times 2 = 200$ classes. The complexity model rises greatly, but it does not reach the level of over-fitting. Therefore, the accuracy of prediction is improved significantly. Overall, siamese convolution neural network can perform our task well and yield excellent results.

Settings	Accuracy	Relative performance
None	78.26%	1
Auxiliary loss	93.55%	1.20
Weight sharing	83.95%	1.07
Both	94.84%	1.21

Table II: Performance comparisons

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[2] Lu, Shin, and G. Em, "Dying relu and initialization: Theory and numerical examples," Mar 2019.

APPENDIX

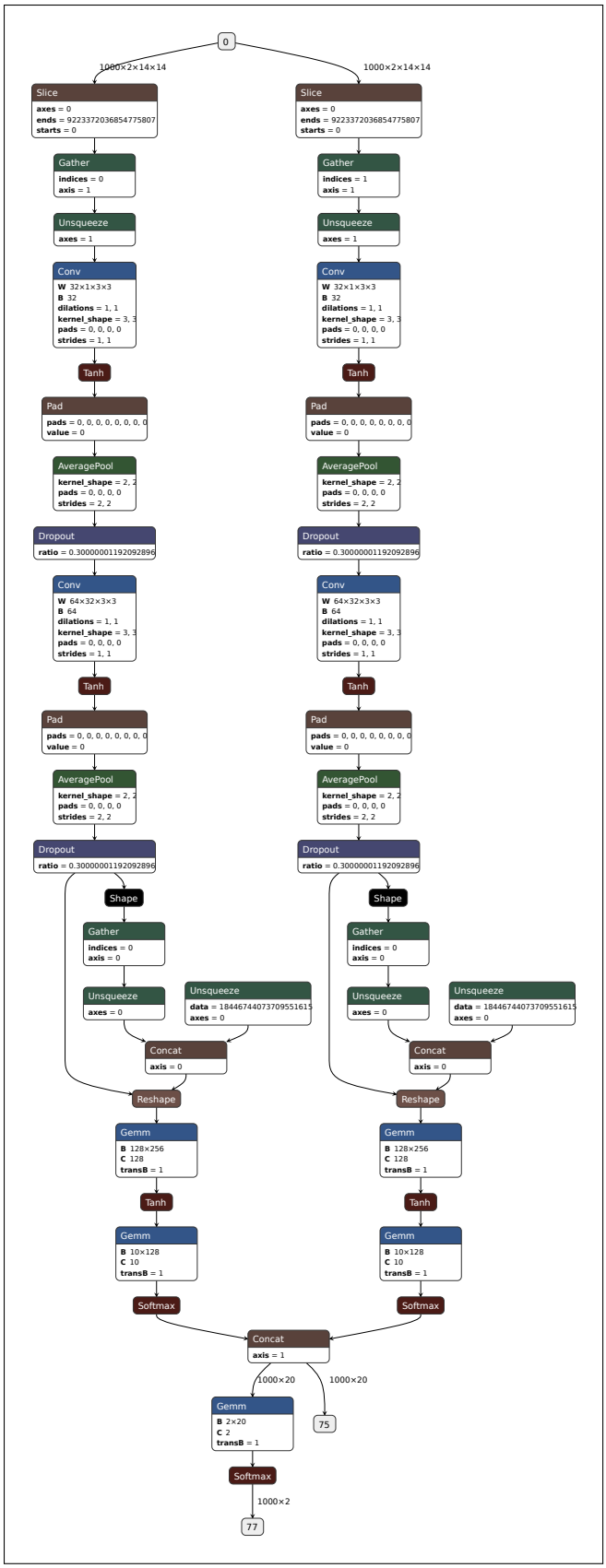


Figure 7: The structure of siamese network