Deep Learning Mid-term Project on MNIST

马立傲
22300130110@m.fudan.edu.cn

1 Introduction

In this problem we will investigate handwritten digit classification. MNIST (Modified National Institute of Standards and Technology database) is a large database of handwritten digits commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by “re-mixing” the samples from NIST’ s original datasets. The dataset contains 60,000 training images and 10,000 testing images. Each image is a 28x28 pixel grayscale image and is labeled with the correct digit(0-9) it represents.

Here in this project and experiment, I will show how to implement one or more neural network to recognize the handwritten digit including the MLP and CNN model, some of which may have some modifications according to the requirement of the project. I will show how I conducted the experiment to test the model and conclude the ability of the model. The implemented several modifications, which can reply the questions in the 1.2 in the requests of our project, is accumulated as the questions’ sequence in the report to tested whether the model acts better. If you want to do modification on the model and check other combination of modification, you can change the relevant parameters in the test_train.py or mindspore_CNN.py. The Github repository link is here, which include all the codes, MNIST data and model weights: https://github.com/Azazel-Malaria/deeplearning_midterm_22300130110. If you want to check the data and model weights on the drive, I also saved the data on Google Drive. Here is the link: https://drive.google.com/drive/folders/18Bmi-3s6W8_zp

9F4JQt2x60A3_W4_cLj?usp=drive_link.

2 Experiment

2.1 Primal MLP

For primal MLP model, without any other augment method to strengthen the ability of the model, the metric score of the last epoch on the train and validation dataset is respectively at 0.125 and 0.117, which turns out to be far from effective in classifying the MNIST set. It can be concluded that the MLP failed to get the detailed feature of the image data on slide level.

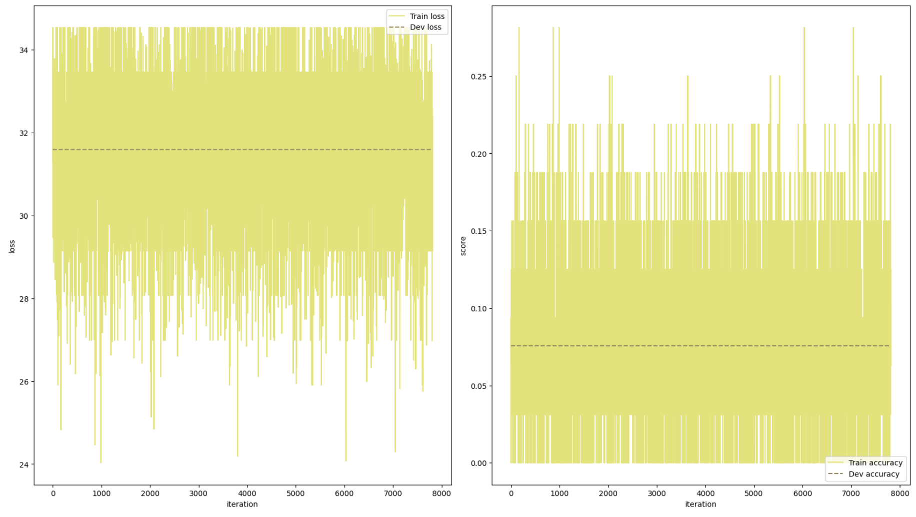

图 1: MLP loss and accuracy

The parameter in the MLP experiment follows the primal setting downloaded from elearning:

weight decay=[1e-4, 1e-4]

learning rate=0.06

milestones=[800, 2400, 4000]

gamma=0.5

number of epochs=5

log of iters=100

The weight of the primal MLP shows as the following:

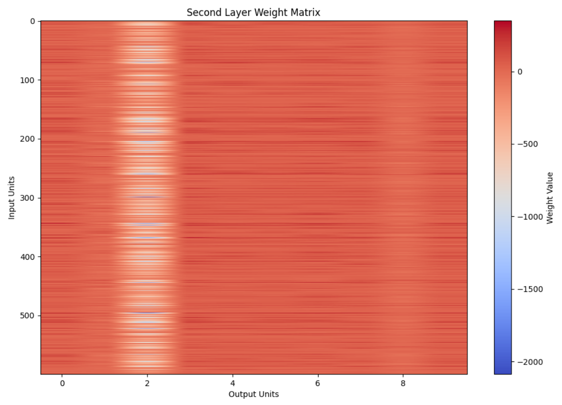

图 2: Primal MLP layer weight

The primal method, of course, is far from effective in training. The loss curve on the training set waves severely, while the dev loss failed to converge. The accuracy curve is the same, showing the typical overfitting phenomenon.

2.2 Modify the number of hidden units of MLP

Change the number of the hidden units to [512, 256, 128, 10], the weight of the parameter of the units changes and the accuracy increased little, which commits to our ”commonsense”, as in the image processing, simply deepen the neural network is not quite effective. The weight of the layer, however, changed greatly through visualization, denoting the weight changes in the new neural layer.

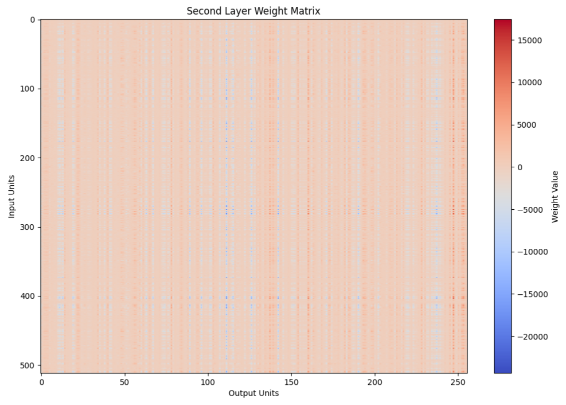

图 3: Hidden layer added MLP weight

2.3 Momentum update method

The momentum update rule is given by:

$$w_{t+1} = w_t - \alpha_t \nabla f(w_t) + \beta_t (w_t - w_{t-1})$$

where:

- w_t is the parameter vector at step t,
- α_t is the learning rate (step size),
- $\nabla f(w_t)$ is the gradient of the objective function at w_t,
- β_t is the momentum coefficient (commonly set to 0.9),
- $w_t - w_{t-1}$ is the momentum term (previous update direction).

So, we changed the code in the optimizer.py and added the class of MomentGD. The curve of the accuracy and loss differs from the former model. Though the maximum accuracy is still around 0.1, the loss curve of the model is more steady than SGD counterpart. It shows the great convergence ability of Momentum update method, as the loss curve descents quickly and almost plunges as a line. The dev accuracy also changes, showing small waves here.

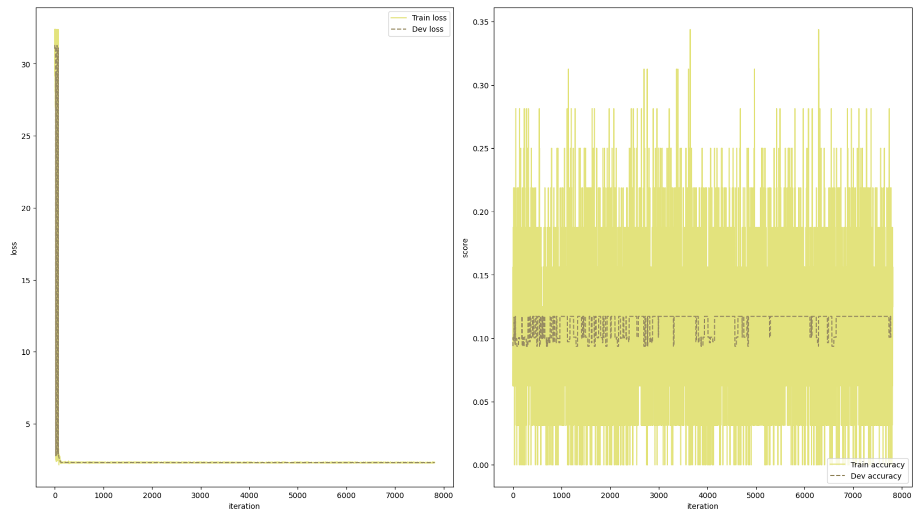

图 4: Accuracy of MomentGD MLP

2.4 Regulation methods

2.4.1 L2 regularization

The L2 regularization adds a penalty term to the loss function based on the squared L2 norm of the weights:

$$J_{reg}(\mathbf{w}) = J(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

where:

- $\mathbf{w}$ is the weight vector,
- λ is the regularization strength,
- $\|\mathbf{w}\|_2^2 = \sum_i w_i^2$ is the squared L2 norm of the weights.

The gradient of the regularized loss function includes an additional term:

$$\nabla J_{reg}(\mathbf{w}) = \nabla J(\mathbf{w}) + \lambda \mathbf{w}$$

In gradient descent, the weights are updated as:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \left(\nabla J(\mathbf{w}_t) + \lambda \mathbf{w}_t \right)$$

where η is the learning rate.

Here in the experiment, I set every l2 penalty regularization strength as 0.01. However, the l2 method still doesn't change the accuracy or weight obviously, which means that the MLP algorithm has almost reached its upper limitation. But the loss curve become even smother(in the curve, though it is not quite obvious, you still can find that the descending line is thinner, which means that the descending process is smother, with less fluctuation), which will lead to less probability to be overfitting.

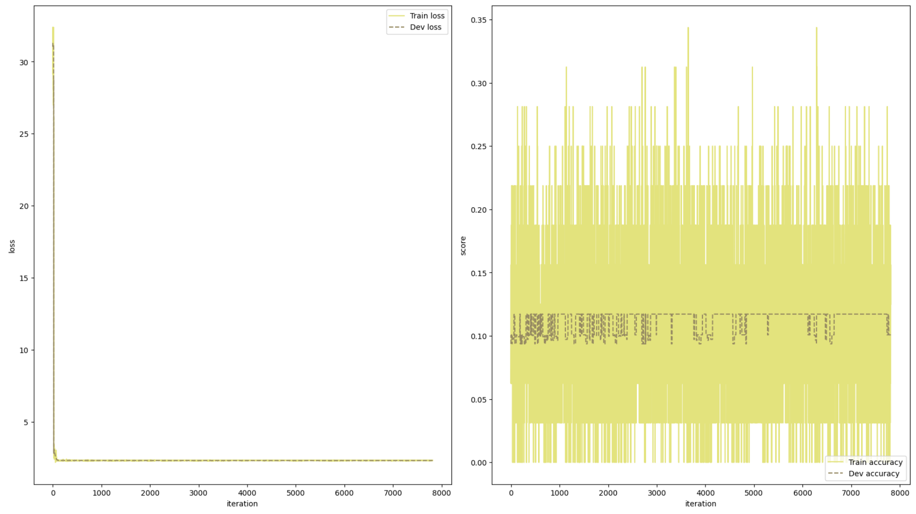

图 5: MLP with L2 regulation

2.4.2 Early Stopping

For the early stopping method, it will stop the training if the model is going to be overfit. In MLP, as you can seen in the accuracy curve before added early stopping, especially in validation accuracy curve, the charts show the typical overfitting phenomenon. After added early stopping the training process soon stopped and the accuracy curve is shortened obviously.

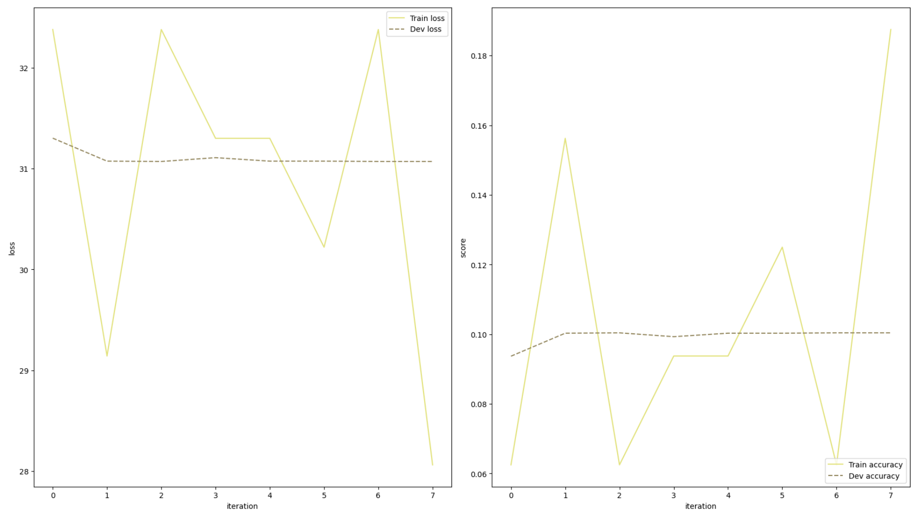

图 6: Early stopping MLP accuracy curve

2.4.3 Dropout

If without early stopping, the dropout method turns out to be also similar to l2 method. However, if without l2 loss, as you can see, it is less efficient in smoothe the loss curve.

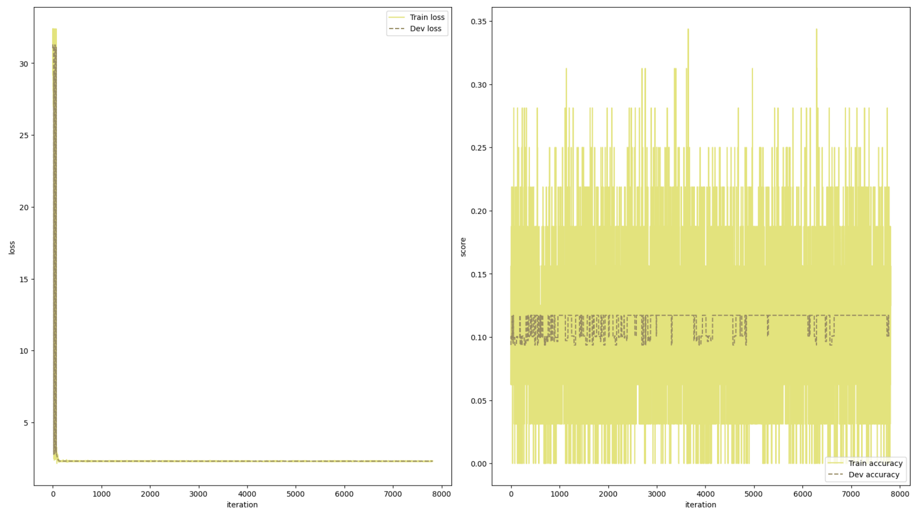

图 7: MLP with dropout(without early stopping and l2 penalty)

2.5 Cross Entropy Loss and Softmax Layer

In the op.py, here we realized the softmax function, saved in 'op.py', and also defined the class of "CrossEntropyLossWithSoftmax". But if we use the method, it is hard to find it makes any obvious enhancement to the performance of the model through the curve, no matter whether use any regularization method or not.

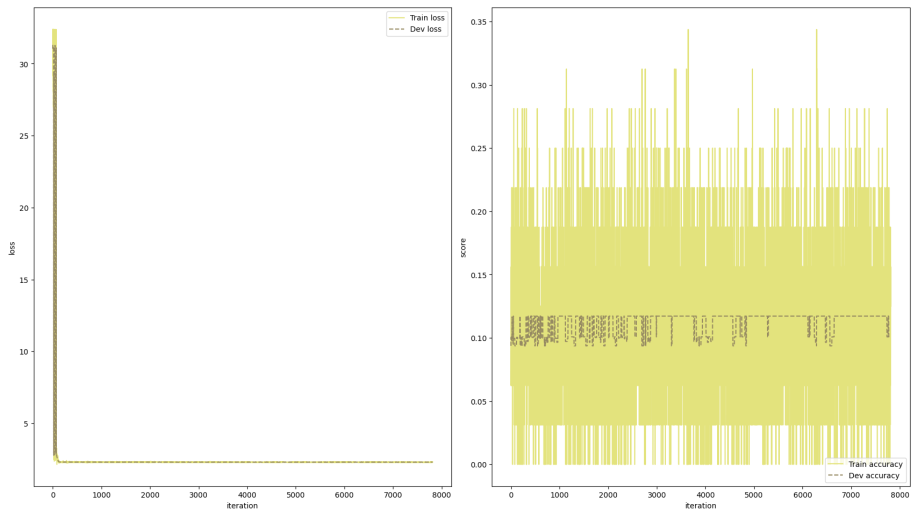

图 8: Softmax MLP accuracy

2.6 Transformation on images

The original images set was enlarged through the transformation like shifting, rotating and zooming. The shifting scale and angle in the transformation was set randomly in uniform distribution under the parameter: [max shift=2, max rotate=15, max zoom=0.1]

Through transformation, however, the accuracy score on MLP model didn't increase obiviously, remaining at around 0.1 in the last epoch. Considering the severe overfitting and the limitation of MLP, here is another experiment made on the CNN(LeNet) model. However, the accuracy score even dropped if compared with the accuracy curve in the next part.

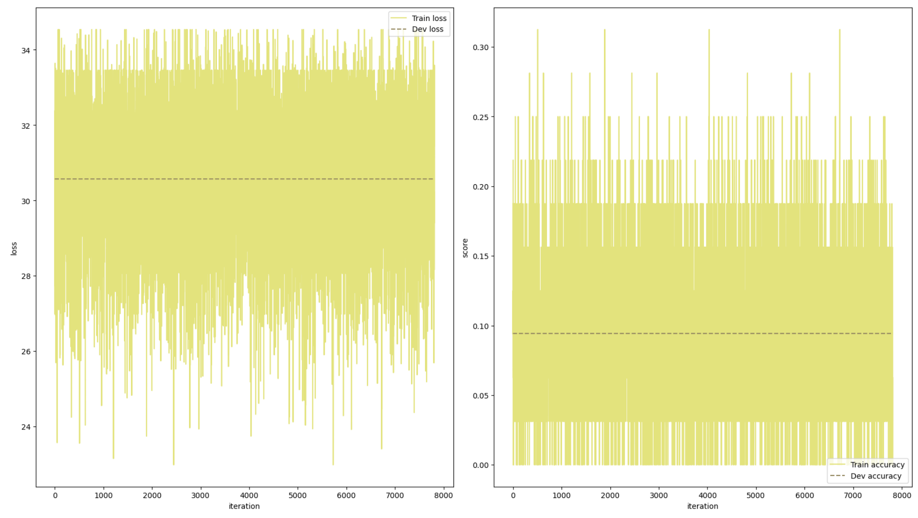

图 9: MLP with image transformation

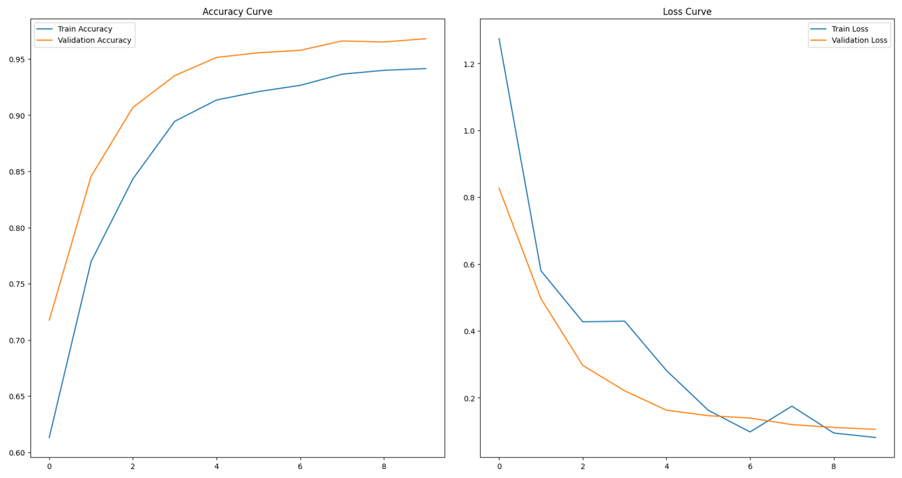

图 10: LeNet with image transformation

2.7 CNN (LeNet and Adam)

As it shows above, the MLP method has reached almost to its maximum accuracy. So here we tried the CNN model — actually not only one CNN model. So before showing the CNN, I must mention that the primal deep learning set is too slow to implement the convolution operation (but it can run, you can still check it in the ”test_train.py”). So here we directly show the result of the Mindspore framework version.

First, the CNN layer was constructed as:

Input → Conv2d(1, 6, kernel size=5) → ReLU() → MaxPool2d(kernel size=2, stride=2) → Conv2d(6, 16, kernel size=5) → ReLU() → MaxPool2d(kernel size=2, stride=2) → Flatten →Dense(16*4*4, 120) → ReLU() → Dense(120, 84) → ReLU→ Dense(84, 10)

However, the accuracy is quite low, as you can find in the curve, showing amost the same as MLP, and also quite strange.

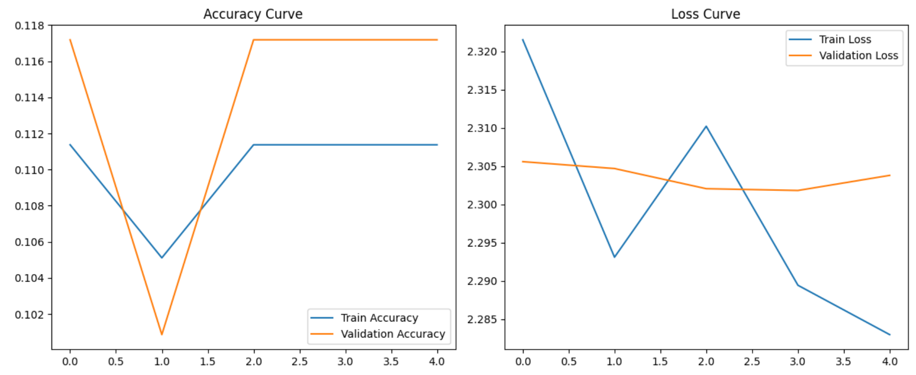

图 11: Primal CNN accuracy

So, here we changed the learning rate(which is too big if it set 0.06, leading to the strong oscillation in the loss curve) to 0.001 and also changed the optimizer to the Adam to get the best model, which is more comprehensive compared to simply use SGD or Moment method, though the request in the project didn't mention this. The method is noted as:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

The accuracy surges to over 0.95. Other settings are: normal weight initialization = 0.02, without early stopping or other regulation methods(for no overfitting), without image transition.

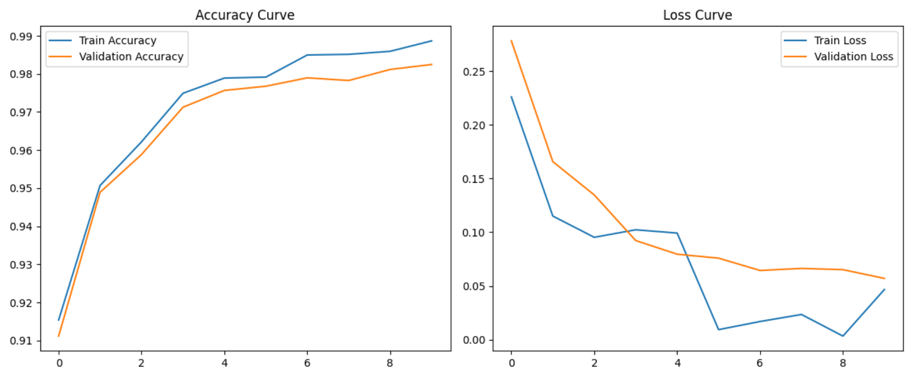

图 12: CNN with Adam

Here, we can also try to recurrent the classical LeNet, also under lr = 0.001 and Adam optimizer(other settings are also the same).

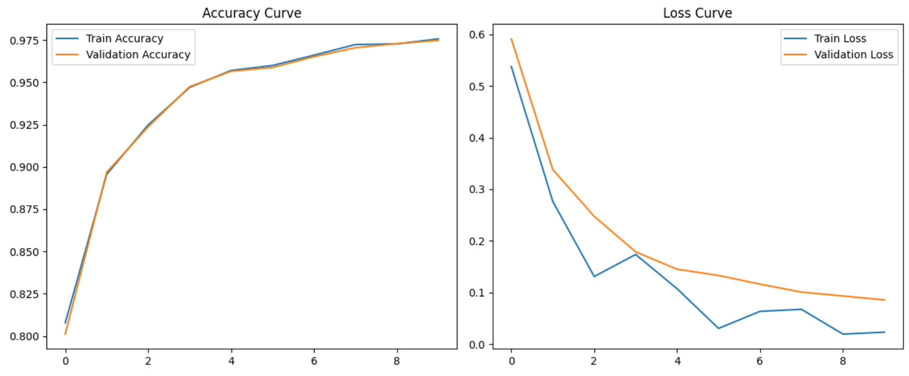

图 13: LeNet

Though the accuracy of LeNet on validation set is lower then the primal

CNN, its gap between train and validation set is trivial, showing it is still not overfit.

So, here we use the transition again, hoping to get better result through the augment in the dataset.

2.8 Weight Visualization

In the fold of the ”saved_models”, I saved the weight of the parameters and the history accuracy. So here, let’s check the visualization of the weight.

For MLP model, the visualization is quite easy to read. as you can see, the primal MLP layers shows the weight of the layer fitting to the shape of MNIST number, though some are not accurate enough.

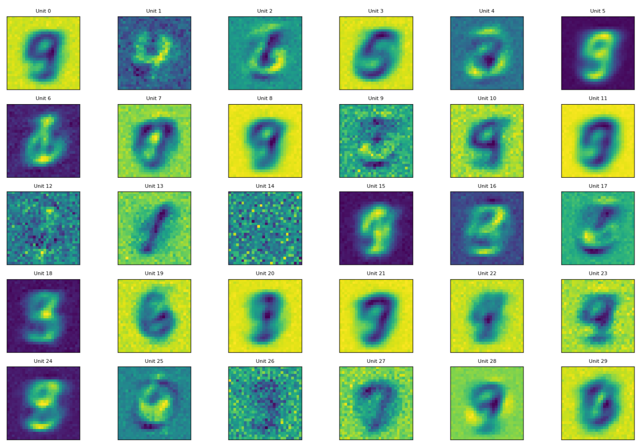

图 14: Primal MLP weight

For CNN, however, the visualization of convolution kernels is different and more abstract.

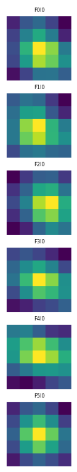

图 15: LeNet Conv Kernel 1

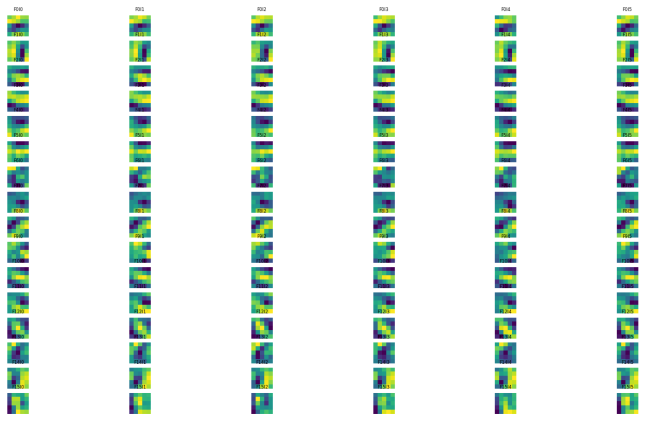

图 16: LeNet Conv Kernel 2

For the first convolution layer, the visualized kernels turned out to be nearly centrosymmetric, which implies that these kernels primarily function as low-level feature detectors, sensitive to edges, strokes, and basic geometric patterns (e.g., horizontal, vertical, or diagonal lines) in the input digits. While the second convolution layers seems to show the more detailed changes in the images. These kernels likely combine outputs from the first layer to detect mid-level structures, such as curve intersections, stroke endpoints, or localized shapes.

2.9 Conclusion

In conclusion, on the MNIST dataset, the operations, including adding hidden layer, moment method, regularization methods, CEL, softmax and Adam, most are effective in enhancing the model but not all are efficient. The moment method and l2 penalty are the best ways to smoothen loss curve, while in the CNN, the Adam method can greatly increase accuracy score. Through the visualization we can find that CNN models can capture the detailed and geometric features in the images.