COMP3314_2C Machine Learning Programming Assignment 2: Convolutional Neural Networks

Final Report

By Guo Xiaole, Zhao Wenzhe

Table of contents

1. The best testing accuracy and settings

1.1 The Overall Testing Accuracy

Our model achieves a testing accuracy of 94%.

1.2 The Accuracy of Each Class

Accuracy of class 0: 94.6%
Accuracy of class 1: 93.6%
Accuracy of class 2: 95.8%
Accuracy of class 3: 91.8%
Accuracy of class 4: 95.4%
Accuracy of class 5: 92.8%
Accuracy of class 6: 93.4%
Accuracy of class 7: 96.2%
Accuracy of class 8: 93.6%
Accuracy of class 9: 94.6%

1.3 Model Settings

1.3.1 CNN architecture:

A summary of the architecture is as follows:
[3*32*32] INPUT
[64*32*32] CONV1: 64 3x3 filters with stride 1, pad 1 - # of Weights: 1728, # of Biases: 64
[64*32*32] RELU activation layer
[64*32*32] Normalization layer
[64*32*32] CONV2: 64 3x3 filters with stride 1, pad 1 - # of Weights: 36864, # of Biases: 64
[64*32*32] RELU activation layer
[64*32*32] Normalization layer
[64*16*16] MAX POOL: 2x2 filter at stride 2
[64*16*16] Dropout layer
[128*16*16] CONV3: 128 3x3 filters with stride 1, pad 1 - # of Weights: 73728, # of Biases: 128
[128*16*16] RELU activation layer
[128*16*16] Normalization layer
[128*16*16] CONV4: 128 3x3 filters with stride 1, pad 1 - # of Weights: 147456, # of Biases: 128
[128*16*16] RELU activation layer
[128*16*16] Normalization layer
[128*8*8] MAX POOL: 2x2 filter at stride 2
[128*8*8] Dropout layer

[256*8*8] CONV5: 256 3x3 filters with stride 1, pad 1 - # of Weights: 294912, # of Biases: 256
[256*8*8] RELU activation layer
[256*8*8] Normalization layer
[256*8*8] CONV6: 256 3x3 filters with stride 1, pad 1 - # of Weights: 589824, # of Biases: 256
[256*8*8] RELU activation layer
[256*8*8] Normalization layer
[256*4*4] MAX POOL: 2x2 filter at stride 2
[256*4*4] Dropout layer
[256*4*4] Flattened to [4096]
[512] FC1: 512 neurons - # of Weights: 2097152, # of Biases: 512
[512] RELU activation layer
[512] Normalization layer
[512] Dropout layer
[10] FC2: 10 neurons - # of Weights: 5120, # of Biases: 10

The entire details of the architecture are shown in section 2.1 Network architecture.

1.3.2 Epoch

The maximum epoch is 17.

1.3.3 Initial learning rate

lr=5e-4=0.0005

1.3.4 Decaying strategy of the learning rate

Step_size = 15, gamma=0.7

1.3.5 Data augmentation strategy

Same as the one in the template.

1.4 Computer configuration and running time

Computer configuration:
The CPU is Intel(R) Core(™) i9-10900 CPU @ 2.80GHz.
The time required is 2647.25s.

2. Experimental settings and corresponding accuracies

2.1. Network architecture

At first, we tried using the existing CNN architecture, LeNet-5. It turned out that the training speed of LeNet is much faster, and it only takes around 590s to run 15 epochs. However, the accuracy of LeNet is not satisfactory, with the training accuracy of 78.1% and the testing accuracy of 82%. We infer it's because the LeNet-5 model is not complex enough to capture the feature of the data. Thus we discarded the LeNet-5 model. Then we tried to make the model more complex by adding additional layers so that it can learn the complex features of the images better.

The first improvement of the network architecture is that our group added Batch Normalization layers after each ReLU activation in the convolutional layers, which standardizes the output from the last layer, and helps to stabilize and speed up the training process.

Next, we think there are too few convolutional layers in the template network, so we added additional sets of convolutional layers, allowing the network to learn more abstract and higher-level representations. We also increased the number of the initial convolutional filters from 32 to 64 to allow the network to capture more complex patterns in the input images.

Then, we started to train the model with the above architecture, and all the other hyperparameters were the initial values except for the epoch which was 17. We obtained the following results:

```
Epoch:[17], training accuracy: 97.1, training loss: 0.091
Finished Training
Accuracy of the network on test images: 93 %
Accuracy of class 0: 94.0%
Accuracy of class 1: 94.2%
Accuracy of class 2: 94.6%
Accuracy of class 3: 90.0%
Accuracy of class 4: 96.2%
Accuracy of class 5: 92.8%
Accuracy of class 6: 93.8%
Accuracy of class 7: 95.6%
Accuracy of class 8: 91.8%
Accuracy of class 9: 95.0%
```

There is a relatively huge gap between training accuracy and testing accuracy. It means overfitting might have happened.

We inferred that overfitting might be caused by two reasons. Firstly, the architecture might be too complex and have too many layers. Secondly, we found that the SVHM dataset has the feature that the images contain more than one digit number, and the digit on the sides of the image might be a big noise. For example, for the image that has the label ‘3’, there are other digits like ‘5’ on the side of the picture. That is a sign that the data has a lot of noise. In order to avoid the noise being learned by the model and causing overfitting, we decided to increase the dropout rate from 0.2 to 0.3 to reduce overfitting. And it indeed turned out to reduce overfitting with the following result:

```
Epoch:[17], training accuracy: 96.3, training loss: 0.118
Finished Training
Accuracy of the network on test images: 94 %
Accuracy of class 0: 95.0%
Accuracy of class 1: 93.2%
Accuracy of class 2: 94.6%
Accuracy of class 3: 90.8%
Accuracy of class 4: 96.6%
Accuracy of class 5: 94.0%
Accuracy of class 6: 95.0%
Accuracy of class 7: 96.4%
Accuracy of class 8: 95.0%
Accuracy of class 9: 93.8%
```

When we test this new network architecture with the same hyper-parameters and data augmentation as the template. The testing accuracy of the new network architecture is 93% with epoch 15 and 94% with epoch 17, while the accuracy is 92% for the initial architecture. So, we assume the new network architecture is better than the template network architecture.

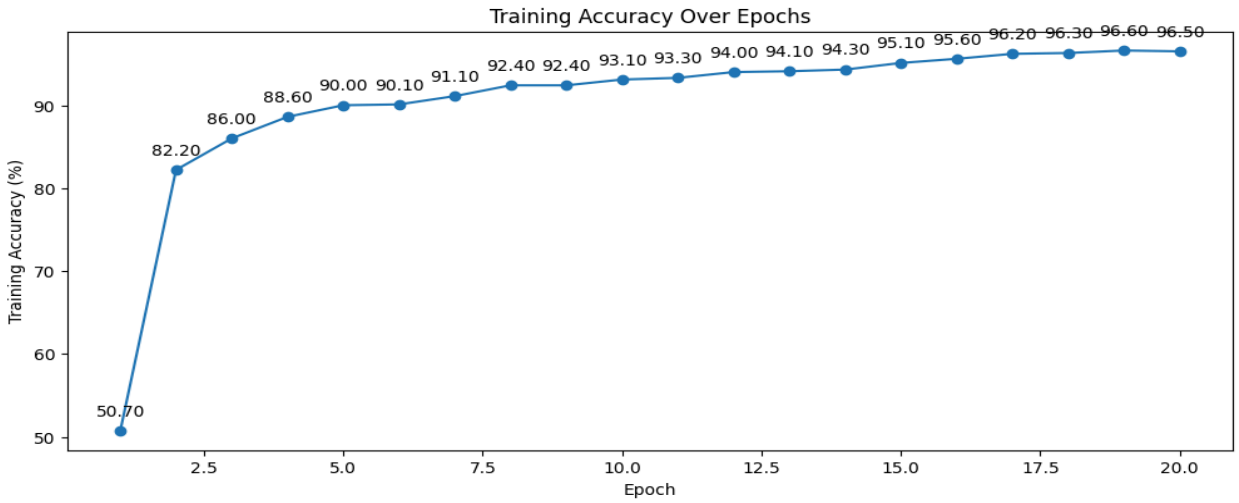
All our tests on the hyper-parameters below are based on this new network architecture.

Here are the details about the new network architecture:

```
(0): Conv2d(3, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU(inplace=True)
(2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
(3): Conv2d(64, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU(inplace=True)
(5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
(6): MaxPool2d(kernel size=2, stride=2, padding=0, dilation=1, ceil mode=False)
(7): Dropout(p=0.3, inplace=False)
(8): Conv2d(64, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
(9): ReLU(inplace=True)
(10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
(11): Conv2d(128, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
(12): ReLU(inplace=True)
(13): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
(14): MaxPool2d(kernel size=2, stride=2, padding=0, dilation=1, ceil mode=False)
(15): Dropout(p=0.3, inplace=False)
(16): Conv2d(128, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
(17): ReLU(inplace=True)
(18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
(19): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
(20): ReLU(inplace=True)
(21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
(22): MaxPool2d(kernel size=2, stride=2, padding=0, dilation=1, ceil mode=False)
(23): Dropout(p=0.3, inplace=False)
(24): Flatten(start dim=1, end dim=-1)
(25): Linear(in features=4096, out features=512, bias=True)
(26): ReLU(inplace=True)
(27): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
(28): Dropout(p=0.3, inplace=False)
(29): Linear(in features=512, out features=10, bias=True)
```

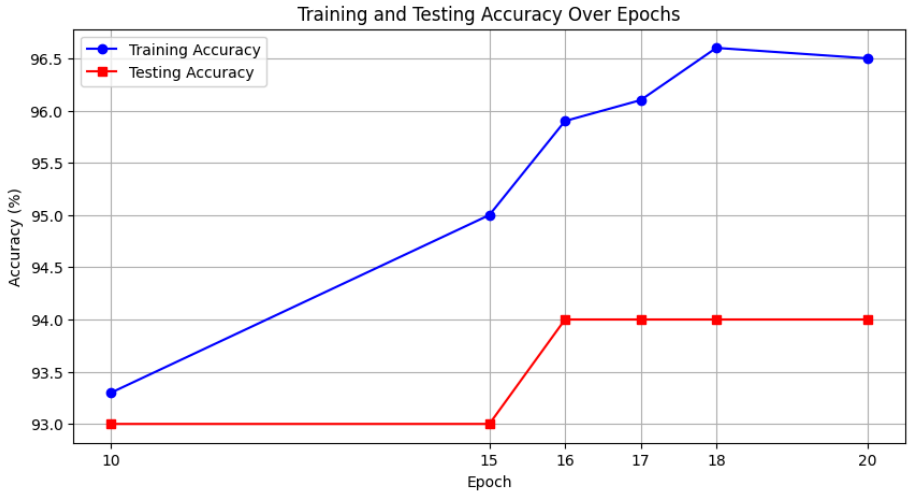
2.2. Test of epoch

The first hyper-parameter that we decided to test is the epoch. We first let the model to train for 20 epochs. The relationship between training accuracy epochs is shown in the diagram below. It shows that starting from epoch 10, the training accuracy reaches 93%. Also, the rate of increase of the training accuracy becomes small. Thus, we plan



to choose the epoch from the range (10,20). We set the epoch to be 10, 15, 16, 17, 18, 20 and other hyper-parameters to be fixed.

The result shows that for epoch=10 and 15, the test accuracy is 93%, and for epoch=16, 17, 18, and 20, the test accuracy is 94%. However, after some repeated tests by our group, we found that the accuracy of epoch<=16 is not stable, which is sometimes 93% and sometimes 94%. Also, we noticed when the epoch is greater or



equal to 18, the overall running time becomes more than 2700 seconds (45 minutes). As a result, we decided to use epoch=17 as our best value.

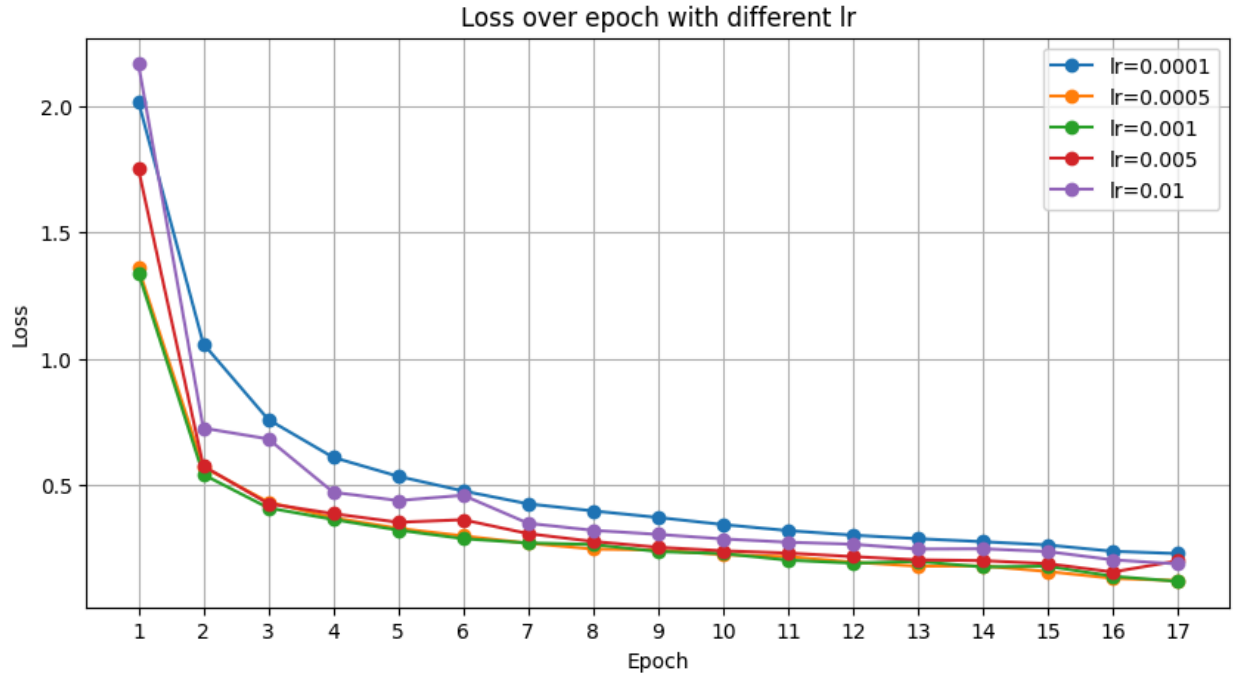
2.3. Test of initial learning rate

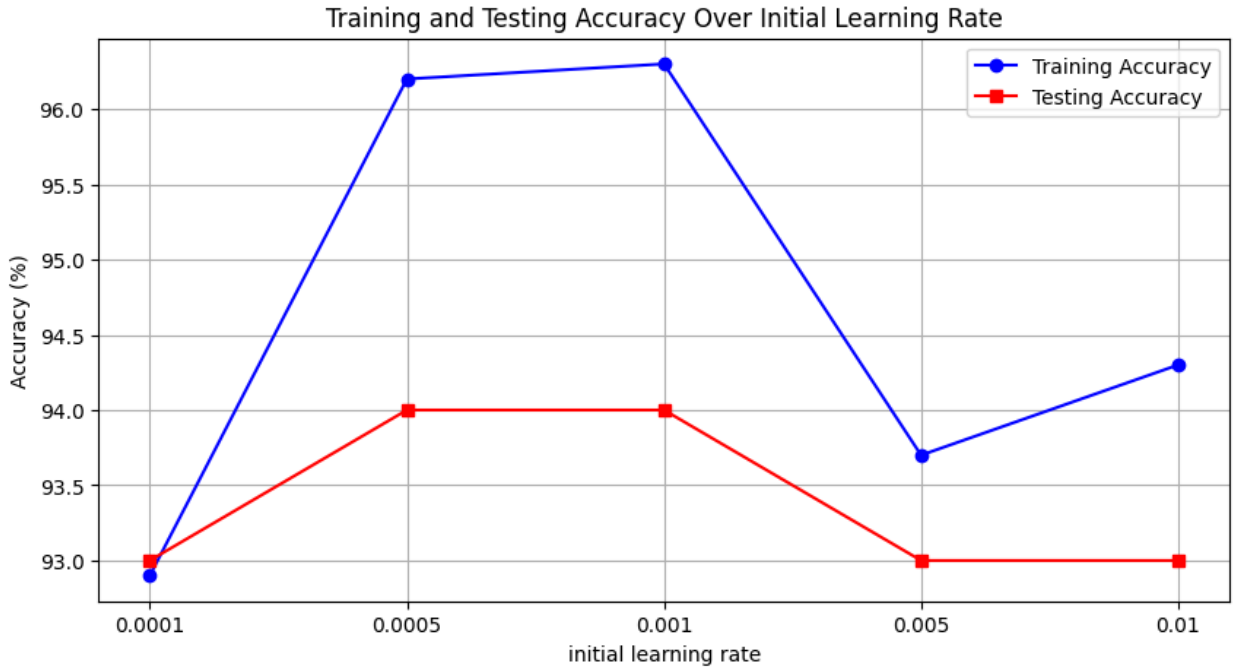
We tested with the following values [0.0001,0.0005,0.001,0.005,0.01] with epoch=17. The training accuracy and testing accuracy achieved is shown in the diagram below. For lr = 0.0001, it's so small that the weights are updated too slowly. As a result, even at epoch 17, the training accuracy is only 93%, and the loss is the highest among all the initial learning rate choices.

By contrast, for lr = 0.005 and 0.01, they seem to be so large that the optimal solution is likely to be overshot. That's why even if they update weights much quicker but at epoch 17, they still can't achieve training accuracy above 95%. Also, the loss over the epoch seems to be a bit volatile and not steadily decreasing.

Lr = 0.0005 and lr = 0.001 performs equally well. From the diagram, we can see the loss is steadily decreasing. Although lr=0.001 seems to have a higher training accuracy than lr=0.0005, the difference is not big, and they have the same testing accuracy. In this

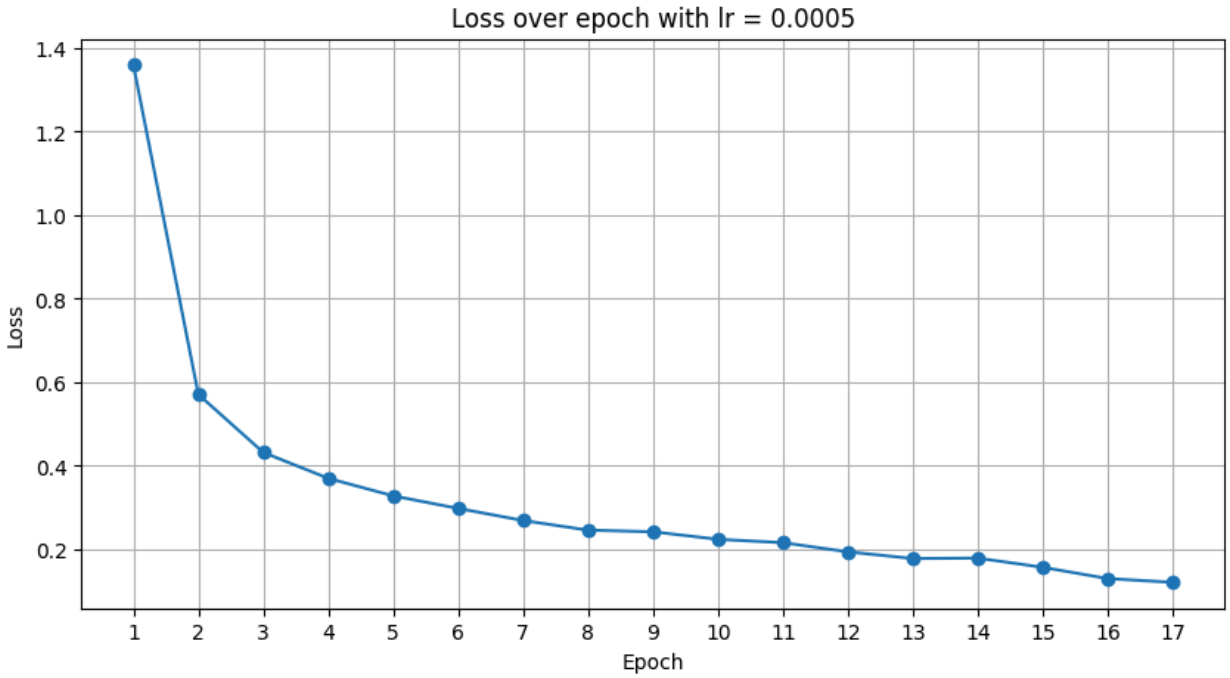
case, we want to be more conservative and update the weight more slowly to avoid overshooting. As a result, we chose lr=0.0005.





2.4. Test of decay strategy

Our group chose a set of gamma and step size values to test for the decay strategy. Here are our tested values of (gamma, step size): (0.8, 15), (0.7, 15),(0.5, 15),(0.8, 10),(0.7, 10),(0.5, 10).
According to the rule, we should decay the learning rate when the loss decreases slowly. However, from the diagram below, we can tell that actually, the loss curve doesn't encounter an obvious plateau. We infer that that's why the above combination of decaying strategies doesn't turn out to make a big difference.



We decided to set the step_size to be the larger value 15. Because we noticed that if the step_size is 10, then after epoch 10, the update of weights seems to be a little bit too slow. Also, since the initial learning rate (0.0005) is already a bit small, we think it's safe to set the step_size to be larger, meanwhile ensuring that overshooting the optimal solution will not happen. We finally chose (step_size = 15, gamma=0.7).

2.5. Test of data augmentation

As mentioned earlier, the SVHM data set has a feature that besides the digit in the center, there are other digits on the sides that become noises. So we try to apply center crop to the images to let the model only focus on the center of the image. However, the result achieved with the crop doesn't improve.

```
Epoch:[17], training accuracy: 94.9, training loss: 0.164
Finished Training
```

```
Accuracy of the network on test images: 93 %
Accuracy of class 0: 94.4%
Accuracy of class 1: 93.8%
Accuracy of class 2: 93.6%
Accuracy of class 3: 91.4%
Accuracy of class 4: 96.2%
Accuracy of class 5: 92.6%
Accuracy of class 6: 95.0%
Accuracy of class 7: 95.6%
Accuracy of class 8: 92.2%
Accuracy of class 9: 95.0%
```

We came up with two reasons why it doesn't work as expected. Firstly, although there are noises, they are randomized with such a large training dataset, so it may not affect the generalization ability severely. Secondly, since some of the images don't have the noises on the sides, instead, the digit that we want the model to focus on occupies the whole image, then cropping the image might cut out some important features of the digit. Consequently, we decided not to apply the cropping.

2.6 Test of Optimizer

We tested other optimizers, such as SGD and RMSprop. SGD performs poorly, it achieves a training accuracy of less than 70% and a testing accuracy of less than 80% at epoch 17. RMSprop seems to have no improvement in the model's performance. So we decided to stick with the Adam optimizer.

3. Conclusion and analysis

To summarize, we improved the model performance by making the following adjustments to the template:

1. We found the initial CNN architecture was too simple. Thus, we changed the architecture with more convolution and normalization layers, which evidently improved model performance;
2. We Increased the epoch to 17 to learn more from the data;
3. We set the initial learning rate to 0.0005 to ensure both speedy and stable weight updates;
4. We found the decay strategy doesn't make a huge impact on the model performance here. We set the decay strategy to be (step_size = 15, gamma=0.7)
5. We tried cropping the image to deal with a feature of the SVHM dataset. However, it didn't make a huge difference because the noises seemed well randomized, and we had already incorporated measures to prevent overfitting in the architecture design.

6. Other optimizers failed to improve the model performance, so we stuck with the Adam optimizer.

Overall, our model achieves a testing accuracy of 94% at epoch 17 within 45 minutes.