MNIST Handwritten Digits Recognition

Weixuan Liu

12/10/2021

In this report, we are implementing a modified version of ResNet34 (He et al., 2016) to the MNIST dataset to effectively classify the handwritten digits to their own category. In our design, the architecture of original ResNet34 is modified in a way that adapted best to our specific problem, and we demonstrate how well the performance is compared with the baseline method, which is the CNN model we built in the class.

1. Introduction

In this dataset, we have 60000 training images of size 28x28 with only 1 single channel representing the handwritten digits starting from 0 to 9 (10 classes), with 10000 testing images to evaluate the performance. The objective of this task is to correctly recognize handwritten images in the testing set. The dataset we are using has already been decoded and thus no decoding step is required to extract the data.

Before building up the model, we first perform the data augmentation on the dataset. After train-validation splitting, we use 85% of the data as the training set and the rest of the data as the testing set. In the training set, the data is augmented with a rotation of up to 10 degrees, random zoom range of up to 0.1 (from 0.9 to 1.1), width shift range of 0.1 (from -0.1 to 1.1), and a height shift range of up to 0.1 (from -0.1 to 0.1). In this way, we randomly generate 1,200,000 more images and use them to train the model in order to further increase the prediction accuracy.

2. Model Architecture and Configuration

In our model architecture, our novelty is that for the first layer, instead of using a filter size of 7x7 in the first convolutional layer, we use a filter size of 5x5 instead, with a stride of 1 (originally it was 2) and a padding of 1 (orginally it was 3). This is because ResNet is originally designed for ta larger image, but in the MNIST dataset, only 28x28 images are used, and thus we need to reduce the filter size. Our extensive experiment indicates that when filter size equals 5x5, the model performs the best. In addition to this, some other changes are made to the model as well, which includes: (1) in layer (see table 1 for detail), the max pooling layer uses a stride of 1 instead of 2 in original ResNet34, and without the dilation, (2) for each of the layer blocks 1,2,3 and 4, instead of only apply the downsampling once in the first skipping layer, we apply downsampling to all the skipping layers, and it yields a better performance in this way, (3) in the fully connected layer, we first have a drop out of rate 0.5 after the average pooling, then connect 512 neurons to 10 output neurons and use the softmax activation function.

In the model configuration, we choose a batch size of 100, with stochastic gradient descent (learning rate of 0.001) and categorical cross-entropy loss function, a total of 10 epochs are evaluated. GPU will be used to train the model in order to speed up the model training process.

Table 1: Overall ResNet 34 architecture, where the detail of layer 1,2,3 and 4 can be found in the subsequent table 2.

Layer.Name	Operation	Connection
Conv1	conv2d, 5x5, 64, stride 1, padding 1	Input Data
Pool1	pooling, 3x3, maxpooling, stride 1	Conv1
Stacked Layer 1	stacking 3 'layer 1'	Pool1
Stacked Layer 2	stacking 4 'layer 2' (stride = 2 in 2a, 2e of the first block)	Stacked Layer 1
Stacked Layer 3	stacking 6 'layer 3' (stride = 2 in 3a, 3e of the first block)	Stacked Lyaer 2
Stacked Layer 4	stacking 3 'layer 4' (stride = 2 in 4a, 4e of the first block)	Stacked Lyaer 3
Pool2	avg pooling	Stacked Layer 4
Flatten	flatten	Pool2
Dropout	dropout(0.5)	Flatten
Linear	linear(512,10)	Dropout
Activation	softmax(1)	Linear

Table 2: The detailed architecture of layer 1,2,3 and 4.

Layer.Name	Operation.Index	Operation	Connection
layer 1	1a	conv2d,3x3,64,stride 1,padding 1	previous layer
layer 1	1b	batchnorm, relu	1a
layer 1	1c	conv2d,3x3,64,stride 1,padding 1	1b
layer 1	1d	batchnorm	1c
layer 1	1e	conv2d,1x1,64,stride 1,batchnorm	previous layer
layer 1	1f	1d + 1e	1d,1e
layer 2	2a	conv2d,3x3,128,stride 1,padding 1	previous layer
layer 2	2b	batchnorm, relu	2a
layer 2	2c	conv2d,3x3,128,stride 1,padding 1	2b
layer 2	2d	batchnorm	2c
layer 2	2e	conv2d,1x1,128,stride 1,batchnorm	previous layer
layer 2	2f	2d + 2e	2d,2e
layer 3	3a	conv2d,3x3,256,stride 1,padding 1	previous layer
layer 3	3b	batchnorm, relu	3a
layer 3	3c	conv2d,3x3,256,stride 1,padding 1	3b
layer 3	3d	batchnorm	3c
layer 3	3e	conv2d,1x1,256,stride 1,batchnorm	previous layer
layer 3	3f	3d + 3e	3d,3e
layer 4	4a	conv2d,3x3,512,stride 1,padding 1	previous layer
layer 4	4b	batchnorm, relu	4a
layer 4	4c	conv2d,3x3,512,stride 1,padding 1	4b
layer 4	4d	batchnorm	4c
layer 4	4e	conv2d,1x1,512,stride 1,batchnorm	previous layer
layer 4	4f	4d + 4e	4d,4e

3. Experiment Result

3.1. Baseline CNN

The first set of result comes from the baseline convolutional neural network structure, which is simply as follow (figure 1):

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 10, 28, 28]             100
              ReLU-2          [-1, 10, 28, 28]               0
         MaxPool2d-3          [-1, 10, 14, 14]               0
            Conv2d-4          [-1, 20, 14, 14]           1,820
              ReLU-5          [-1, 20, 14, 14]               0
         MaxPool2d-6            [-1, 20, 7, 7]               0
            Conv2d-7            [-1, 40, 7, 7]           7,240
              ReLU-8            [-1, 40, 7, 7]               0
         MaxPool2d-9            [-1, 40, 4, 4]               0
           Linear-10                 [-1, 128]          82,048
             ReLU-11                 [-1, 128]               0
           Linear-12                  [-1, 10]           1,290
          Softmax-13                  [-1, 10]               0
================================================================
Total params: 92,498
Trainable params: 92,498
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.24
Params size (MB): 0.35
Estimated Total Size (MB): 0.59
```

Figure 1: baseline CNN architecture.

The baseline method will use 100 as the batch size with 20 epochs, and other setup remains the same as the ResNet34 design. The training and validation loss is given by the following graphs (figure 2):

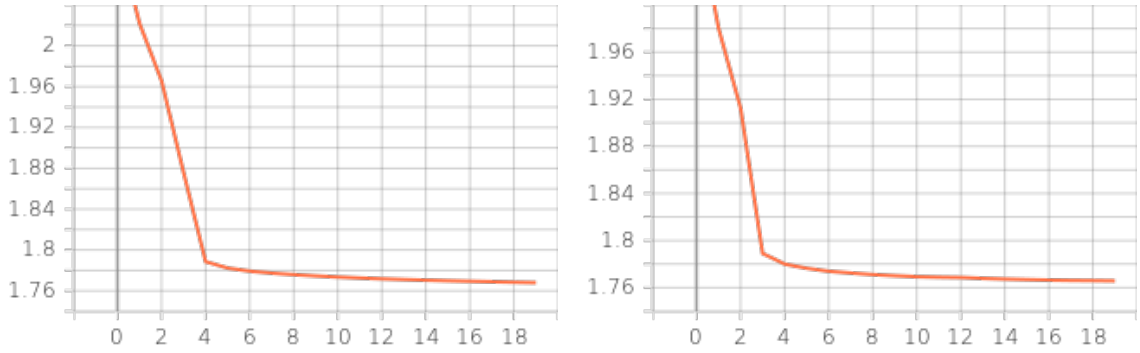

Figure 2: training and validation loss for the baseline cnn method with respect to epoch (x-axis) using categorical cross-entropy loss.

We see that the performance is consistently poor across all the epochs. When evaluating it on the testing set, the testing accuracy is 69.4%, which indicates a relatively poor performance, the training and validation accuracy with respect to epochs is given by the following (figure 3):

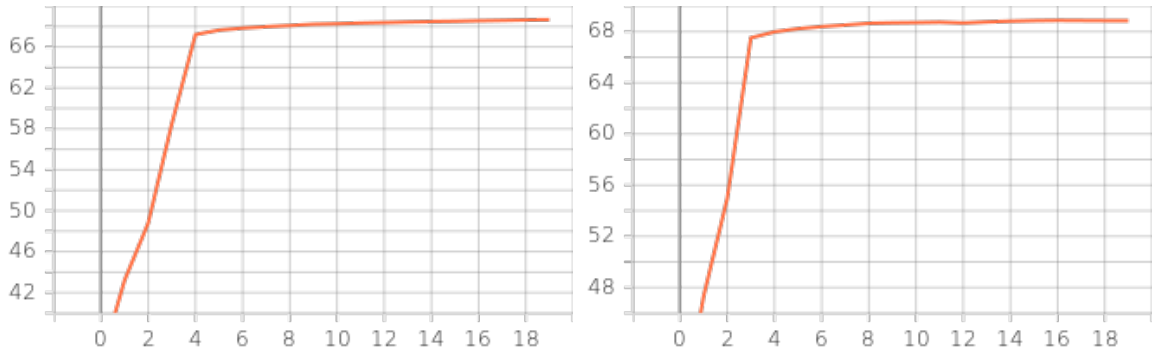

Figure 3: training and validation loss for the baseline cnn method with respect to epoch (x-axis) using categorical cross-entropy loss.

3.2. ResNet34

The second set of result is from our ResNet34 design, with training and validation loss given by the following graphs (figure 4):

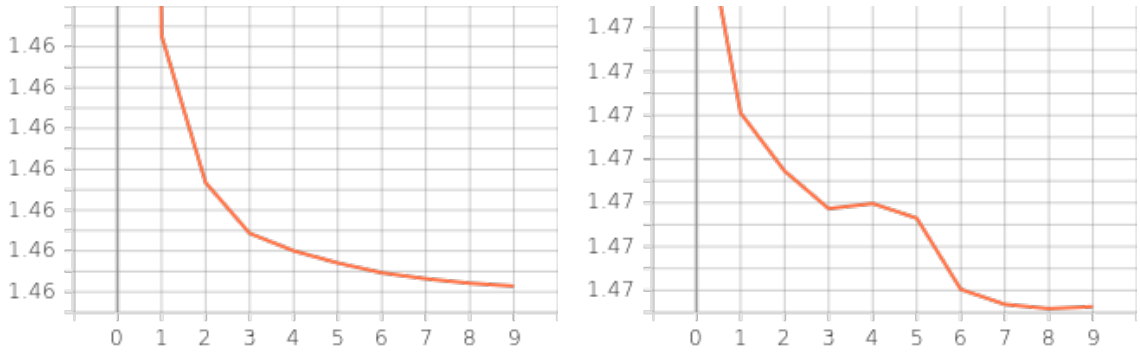

Figure 4: training and validation loss for the ResNet34 method with respect to epoch (x-axis) using categorical cross-entropy loss.

We see that the performance is consistently excellent across all the epochs. When evaluating it on the testing set, the testing accuracy is: **99.45%**, and the training and validation accuracy with respect to epochs is given by the following (figure 5):

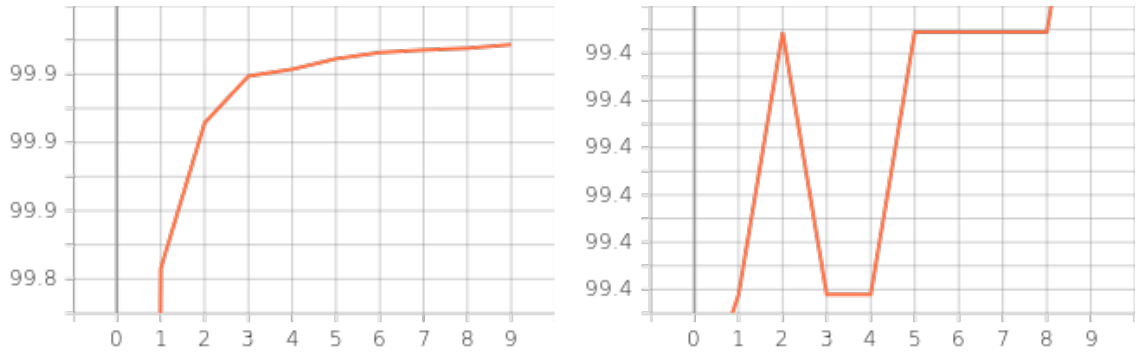

Figure 5: training and validation loss for the ResNet34 method with respect to epoch (x-axis) using categorical cross-entropy loss.

3.3. ROC Curve Comparisons

Finally, we will compare the performance of two models by examining their ROC-curve, which is given as below (figure 6):

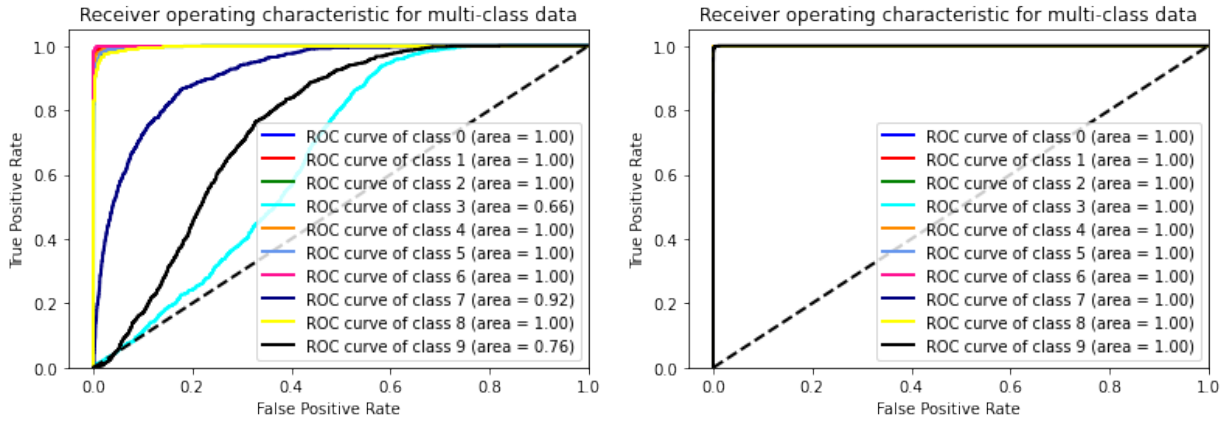

Figure 6: ROC curve when evaluating the performance on the test data for the both methods, where the left graph is the ROC curve for the baseline method, and the graph on the right is the ROC curve for our ResNet34 model.

The result indicates that the ResNet34 has a way better result than the baseline CNN method we built in the class, and the ROC-AUC score is consistently 1 when, but the baseline method only have a portion of classes with ROC-AUC score to be 1.

4. Discussion

In this report, we demonstrate how our modified ResNet design achieves 99.45% prediction accuracy in recognizing the handwritten digits, which outperforms the baseline CNN method. In the future, more advanced architecture such as DenseNet and EfficientNet can be attempted to achieve an even better accuracy. In addition, We can ensemble multiple well-known architectures to further improve the model. In the data augmentation step, we can further increase the number of images that are used for the training purpose.

Another direction to go with is to check whether for those wrong predictions in the validation set, the reason it is wrong is because the model is not really sure about whether to classify it to certain class. If

that's the case, we can implement some state-of-the-art semi-supervised learning method to better make use of validation as well as testing images to achieve a better performance, the simplest case would be psedu-labeling such as self-training.

Reference

- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).