COMP551 Mini-Project3

Dawei Zhou, Ruohan Wang, Xinyi Zhu

November 2021

1 Abstract

In this project, our objective is to correctly identify the handwritten letter and digit in the $56 * 56$ grey-scaled images from a modified version of the MNIST dataset. We tackled the problem by investigating a convolutional neural network(CNN) model. By exploring different design choices, along with regularization and data augmentation techniques, our best performing model achieved an accuracy of 93% on the validation set and 93.65% on the Kaggle test set. We also trialed with other architectures, namely *VGG16* and *RESNEXT*.

2 Introduction

The handwritten digit recognition is the ability of a machine to recognize human handwritten digits. Convolutional neural networks (CNN) are complex feed-forward neural networks that are widely used for image classification and recognition. In this project, our objective was to identify the characters in the images from the combo MNIST dataset.

This dataset consists of 60,000 training images, half of which are labeled. Each sample is a 56x56 grey-scaled image that consists of two characters: one letter from the English alphabet (a-z) and one digit (0-9). These characters can be of different size, location and orientation. Each label is in the form of a size-36 binary vector, which corresponds to the correct output of the image.

For the data preprocessing, we tried the following operations: enlarging the size of image, adding Gaussian noise to the image, normalizing the image, denoising the image, conducting Principle Components Analysis(PCA).

The regularization techniques used in our models are ReLU, striding, padding, Batch-Normalization, Max pooling and dropout.

3 Experiments and results

3.1 Phase 1: simple CNN model

We started with a simple CNN model(Appendix figure 4). After training with the original data, the model reached its maximum accuracy of 67% after 25 epochs with training loss around 1.6.

3.2 Phase 2: data augmentation

To improve the performance, we decided to preprocess the data. we first tried to enlarge the size of images, but the model converged with a training loss around 4.5 and both the training and the validation accuracy decreased. Then we added the Gaussian noise and normalization to the images. In both cases, the model converged with a validation accuracy around 65% after 50 epochs. We also notice that there are some impulse noises in the image. We tried to remove the noises with the *fastNlMeansDenoising* function. The image looks much clearer and the the model converged with a 3% higher accuracy. Finally, we applied PCA to the images and train the model with 5, 10 and 15 principle components and we found that using 10 principle components increased the validation accuracy by 2%. On the basis of PCA, we rotated the image by the degrees of 7, 15, 30, -15 and -30. Our training data became 6 times bigger. And our model reach the accuracy of 84% with the augmented data.

3.3 Phase 3: other regularizations

We found that our model always have more than 97% training accuracy after reaching the maximum validation accuracy. We then tried different regularization techniques to avoid overfitting. We added more ReLU, striding, padding, batch-normalization, Maxpooling and dropout. We observed that the gap between validation accuracy and training accuracy has reduced to less than 5% with more regularization. After several trials with different model designs, we ended up with a model with 93% validation accuracy and 99% training accuracy with a training loss of 0.45 (figure 1 and 2).

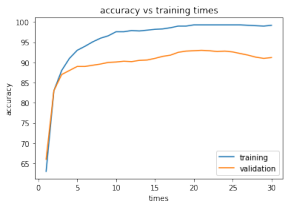

Figure 1: Accuracy vs epoch

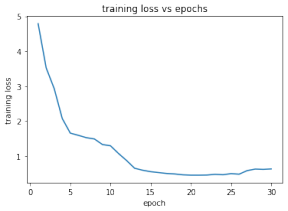

Figure 2: training loss vs times

3.4 Phase 4: modified architecture

We added two more convolutional layers and got our final model (figure 3). With two more convolutional layers, the validation accuracy did not change, but this model converged much faster.

We also tried to reduce the learning rate after 3 epochs by 0.6 and 0.5. But the converged training loss and validation accuracy stayed the same.

In addition, we explored other classic CNN architectures: *VGG16* (architecture in Appendix Figure 5) [1] and *Resnext* [2]. The model converged with a 91% validation accuracy and 99% training accuracy. The Resnext model converged with 92% validation accuracy and 99% training accuracy.

ConvNet Configuration	
	input (56 x 56 grayscale image)
layer1	conv2d-32 conv2d-32
	MaxPool
layer2	conv2d-64 conv2d-64
	MaxPool
layer3	conv2d-128 conv2d-128
	MaxPool
layer4	conv2d-256 conv2d-256
	MaxPool
layer5	conv2d-512 conv2d-512
	MaxPool
	FC-500
	FC-72
	FC-36
	softmax

Figure 3: Final model

4 Discussion and Conclusion

During our trials with different preprocessing techniques, we found the normalization made the image slightly more clear, but the model trained with the processed images had worse performance. The reason might be that the pixel values in the processed image are more close to each other and that made it more difficult for model to recognize the patterns.

By introducing more regularizations, the variance of the model was reduced. But too much regularization stops the model from learning more information from the data.

By adding more convolutional layers to the model, we generally get better performance. This result affirms the previous research in neural networks. One explanation might be that with more convolutional layers, the model is abstracting with more different levels and that usually lead to better generality.

We also attempted to reduce the converging loss by decreasing the learning rate. However, the overall performance of our model did not changed. For future investigation, we believe the remaining loss should be further reduced by improving our model architecture.

5 Statement of Contribution

Dawei Zhou: PCA, model design and tuning, tried other architecture, report writeup

Ruohan Wang: Data Preprocessing, Data Agumentation, model design and tuning

Xinyi Zhu: initial pipeline design, data visualization, tried other architecture, report writeup

Reference

[1] *VGG-NETS from* `https://pytorch.org/hub/pytorch_vision_vgg/`
[2] *RESNEXT from* `https://pytorch.org/hub/pytorch_vision_resnext/`

Appendix

```
CNN(
  (cnn_layers): Sequential(
    (0): Conv2d(1, 6, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU()
    (10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU()
    (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
    (13): ReLU()
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (linear_layers): Sequential(
    (0): Linear(in_features=6400, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=500, bias=True)
    (3): ReLU()
    (4): Linear(in_features=500, out_features=72, bias=True)
    (5): ReLU()
    (6): Linear(in_features=72, out_features=36, bias=True)
  )
)
```

Figure 4: The CNN model we started with

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 5: The Architecture of VGG16