

Handwriting Recognition Using Convolutional Neural Network

Group 17

ChangRunze, JiangYihang, ZhouDecheng, CaoKun, ZhengYu

Abstract—Using Convolution Neural Network(CNN) to implement the recognition of handwritten letters.

Index Terms—Character recognition, CNN.

I. INTRODUCTION

THIS doc will introduce the details of our implementation about our method about Handwritten Letters recognition. In this report, I will detail the selection of our dataset, the Neural Network structure we built and the training result.

A. Dataset

We use the Mixed National Institute of Standards and Technology database(MNIST) to train our network. And we only use the letters part of this dataset. The training set and test set contain 124,800 and 20,800 images, respectively, and the image size is 28*28 pixels.

1) MNIST pre-processing

Parse the *.idx3 -ubyte* format and get the images.

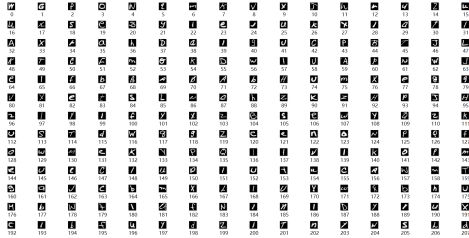


Fig. 1. Part of the images in the Train set.

23,7,16,15,23,17,13,11,22,24,10,14,18,21,26,21,21,24,19,5,2,25,9,5,10,21,11,24,12,1,17,9,1,24,18,1,
0,22,5,1,9,5,22,15,10,16,4,6,11,10,26,25,3,17,18,13,6,3,21,9,26,7,12,21,20,17,5,17,13,6,11,23,11,2,
8,3,16,7,4,24,5,14,1,4,20,13,15,14,8,8,6,8,4,6,23,18,4,20,19,12,21,6,4,14,25,6,9,3,16,21,23,13,3,25,1,
20,20,15,23,10,2,17,19,1,18,24,1,23,9,9,12,10,22,22,23,2,11,8,21,22,17,1,22,5,13,3,8,25,17,5,19,16,
23,23,4,3,17,22,2,19,9,10,5,20,25,5,2,4,16,15,7,18,10,20,6,10,3,12,12,18,21,5,20,24,17,15,23,9,17,11,
4,19,5,17,6,18,13,4,14,6,3,19,22,22,20,5,9,11,22,2,14,10,10,17,26,20,21,25,12,6,25,23,14,24,14,19,1,
8,21,7,12,9,19,15,18,23,24,14,11,17,14,20,20,13,22,3,4,24,2,11,20,2,25,25,5,7,4,4,1,23,23,25,11,5,
8,22,22,21,16,2,9,7,9,9,4,3,6,17,21,18,16,24,12,10,7,10,3,21,1,20,5,8,18,4,19,7,10,24,24,19,22,3,15,1,
6,26,4,5,24,3,7,19,22,18,23,13,8,10,1,18,20,4,14,11,10,26,26,18,5,10,16,26,5,2,21,26,11,22,14,8,9,2,
3,26,15,10,12,7,26,24,16,21,26,8,22,9,16,12,25,14,16,1,9,11,22,15,5,22,4,3,6,20,12,21,19,11,20,24,6,
3,13,17,3,17,15,23,6,18,21,16,5,23,6,5,3,21,16,22,11,16,22,10,23,25,3,13,21,23,1,9,3,3,16,3,14,9,10,
6,17,17,18,21,20,18,17,1,16,16,21,24,1,5,7,10,17,21,19,9,9,7,12,2,14,16,6,19,9,14,6,1,3,21,22,16,
7,20,11,4,24,24,8,17,11,10,15,13,4,20,15,20,24,4,12,15,24,1,22,19,19,2,14,23,15,13,13,2,10,26,19,2,
25,19,8,9,20,2,21,21,20,10,7,18,5,11,4,15,15,21,20,19,26,9,10,21,14,13,18,8,19,9,24,17,19,10,26,8,1,
24,26,15,23,10,10,10,10,24,9,24,11,24,21,7,6,26,16,13,9,22,21,15,22,21,9,2,3,6,16,22,11,26,3,11,22

Fig. 2. Part of the images' label in the Train set.

2) Dataset classification

Although we now have images and their corresponding labels, this is not convenient for us to process and train them. We further classify the data, use the folder name as the label of the images in this folder. Then use the *ImageFolder* class in *pytorch* to read the data to reduce the number of times we read the label file.

3) Load the Dataset

The dataset is to be read in as a folder.

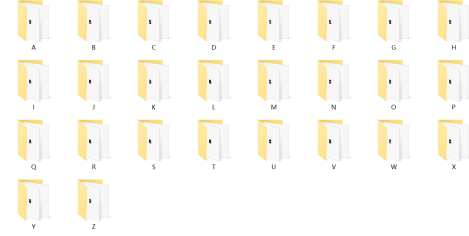


Fig. 3. Use the labels of the images to categorize the images.

```
train_data = ImageFolder(root="./Train_png", transform=transform)
train_loader = torch.utils.data.DataLoader(dataset = train_data, batch_size=BATCH_SIZE, shuffle=True)
test_data = ImageFolder(root="./Test_png", transform=transform)
test_loader = torch.utils.data.DataLoader(dataset = test_data, batch_size=test_data.__len__())
```

B. CNN Building

The network used in this training letter recognition is similar to the structure of LeNet-5: a two-layer convolutional network (convolution + pooling) + a three-layer fully connected layer:

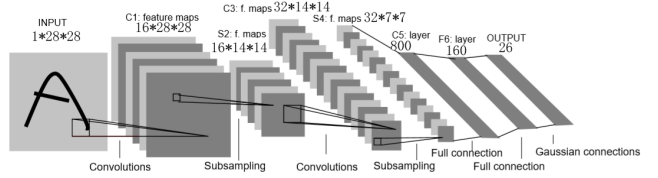


Fig. 4. CNN structure.

We build the CNN as Fig. 4. The CNN structure is as follow.

```
CNN(
    (Conv1): Sequential(
      (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```

(2): ReLU(inplace=True)
(3): MaxPool2d(kernel_size=2, stride
=2, padding=0, dilation=1,
ceil_mode=False)
)
(Conv2): Sequential(
(0): Conv2d(16, 32, kernel_size=(5,
5), stride=(1, 1), padding=(2, 2))
(1): Dropout(p=0.2, inplace=False)
(2): BatchNorm2d(32, eps=1e-05,
momentum=0.1, affine=True,
track_running_stats=True)
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride
=2, padding=0, dilation=1,
ceil_mode=False)
)
(Linear): Sequential(
(0): Linear(in_features=1568,
out_features=800, bias=True)
(1): Dropout(p=0.5, inplace=False)
(2): ReLU(inplace=True)
(3): Linear(in_features=800,
out_features=160, bias=True)
(4): Dropout(p=0.5, inplace=False)
(5): ReLU(inplace=True)
(6): Linear(in_features=160,
out_features=26, bias=True)
)
)

```

Optimizer and Loss function:

```

optimizer = torch.optim.Adam(cnn.
parameters(), lr = LR)
loss_func = nn.CrossEntropyLoss()

```

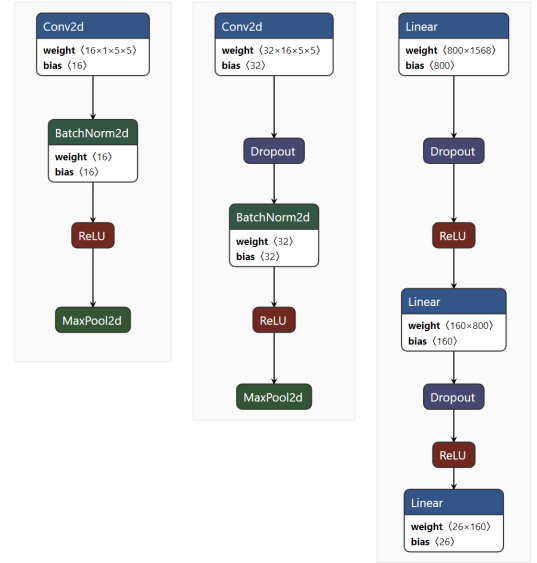


Fig. 5. Model of our CNN.

C. Processing of Convolution

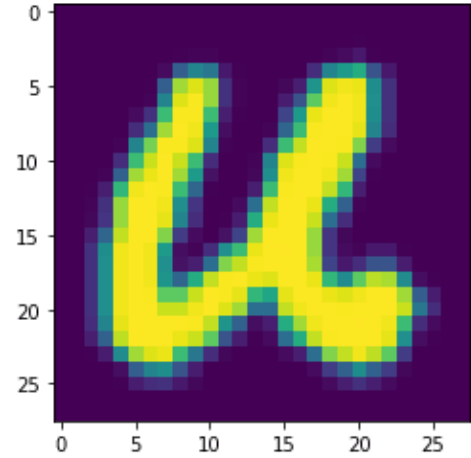


Fig. 6. The original image Input: Letter 'u'.

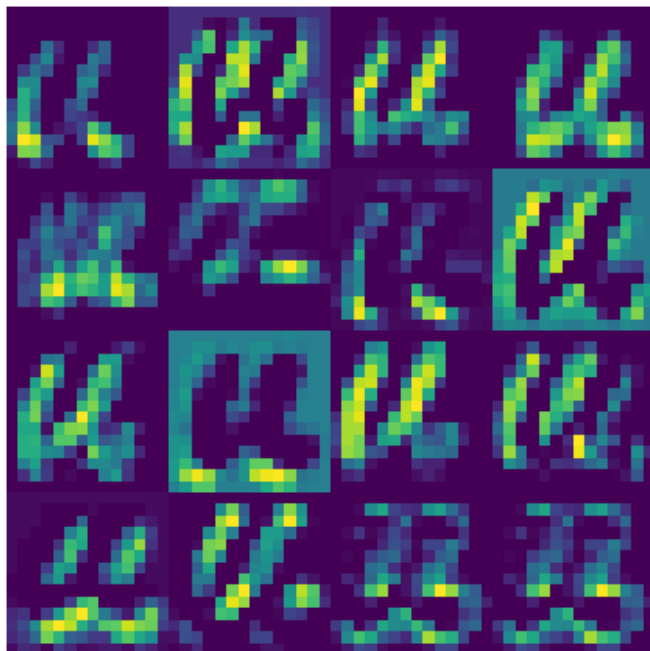


Fig. 7. Letter u after the first convolution.

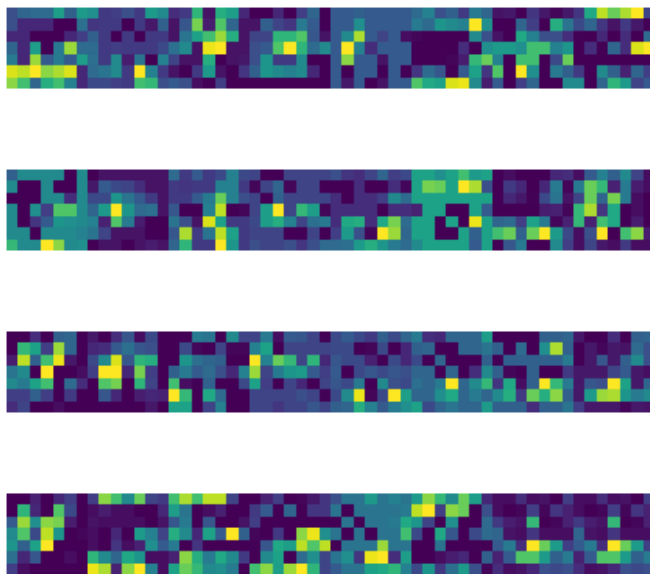


Fig. 8. Letter u after the second convolution.

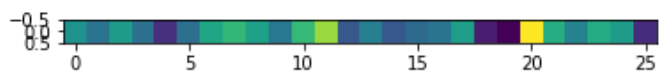


Fig. 9. Fully Connected Layer

II. RESULT

Eopch: 4	train loss: 0.287322	test accuracy:0.90168	step: 0
Eopch: 4	train loss: 0.233646	test accuracy:0.90543	step: 100
Eopch: 4	train loss: 0.247977	test accuracy:0.90240	step: 200
Eopch: 4	train loss: 0.251770	test accuracy:0.90259	step: 300
Eopch: 4	train loss: 0.286806	test accuracy:0.90331	step: 400
Eopch: 4	train loss: 0.348613	test accuracy:0.90976	step: 500
Eopch: 4	train loss: 0.260653	test accuracy:0.90649	step: 600
Eopch: 4	train loss: 0.206683	test accuracy:0.90879	step: 700
Eopch: 4	train loss: 0.296226	test accuracy:0.90682	step: 800
Eopch: 4	train loss: 0.144547	test accuracy:0.91235	step: 900

Fig. 10. Train loss and test accuracy.

$$Accuracy = 0.91$$

Loss and Accuracy :

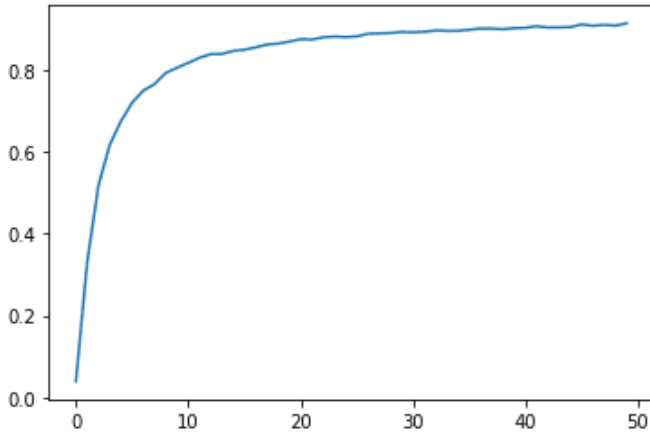


Fig. 11. Accuracy.

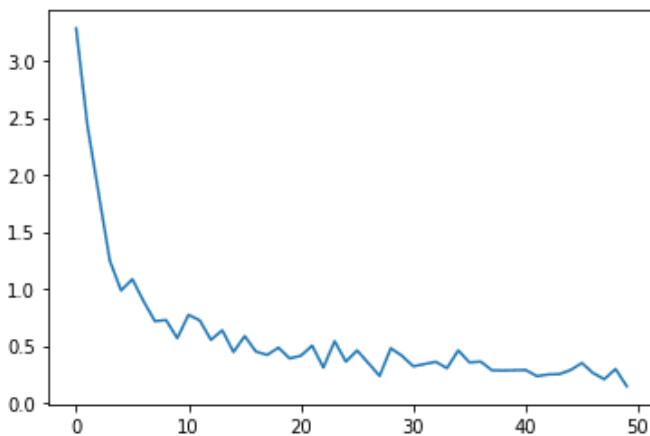


Fig. 12. Loss.

III. REFERENCE

A. Tools

Netron : Model visualization.

B. Libs

Torch : Main Lib to build CNN Model.

PIL.Image : Images' pre - processing.

matplotlib.pyplot : Plot.

cv2 : Classification of Dataset.

C. Code

<https://blog.csdn.net/SESESsss/article/details/111570525>

EMNIST dataset pre-processing.

<https://blog.csdn.net/qq41845951/article/details/118814447>

Visualization and CNN frame.