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 out dataset, the Neural Network structure we built and the training result.

A. Dataset

We using Mixed Nation 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.

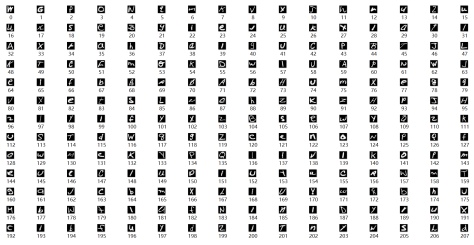

Fig. 1. Part of the images in the Train set.

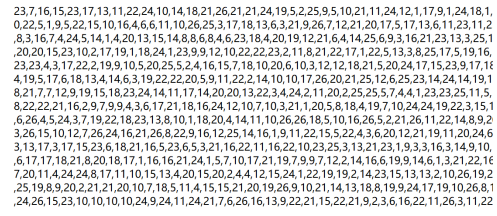

Fig. 2. Part of the images' label in the Train set.

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.

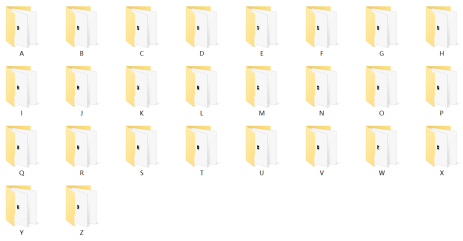

Fig. 3. Use the labels of the images to categorize the images.

```
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:

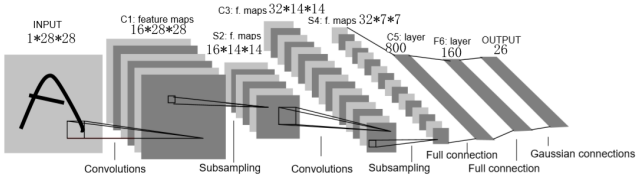

Fig. 4. CNN structure.

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()
```

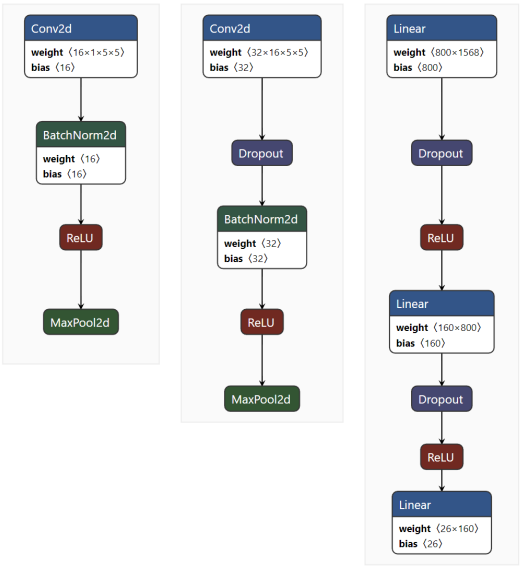

Fig. 5. Model of our CNN.

C. Processing of Convolution

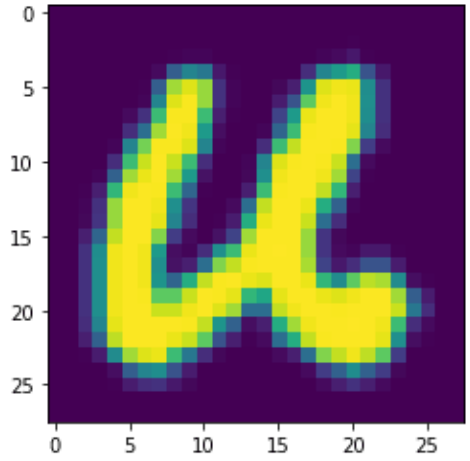

Fig. 6. The original image Input: Letter 'u'.

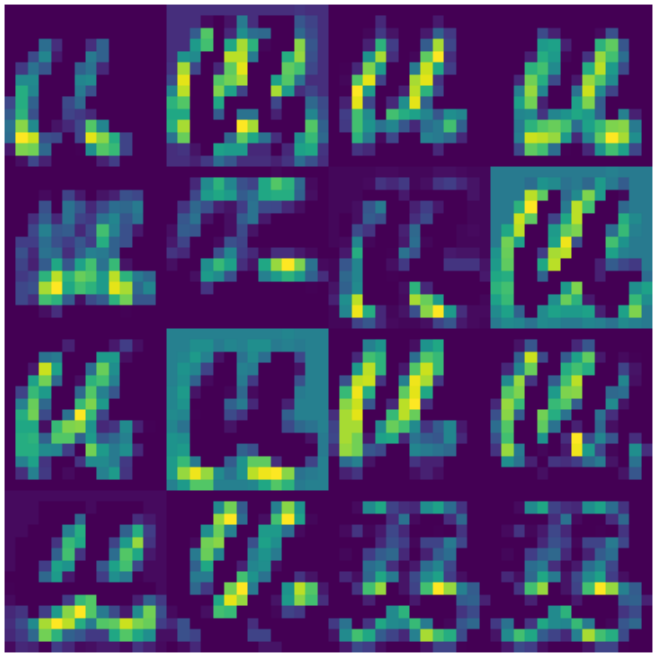

Fig. 7. Letter u after the first convolution.

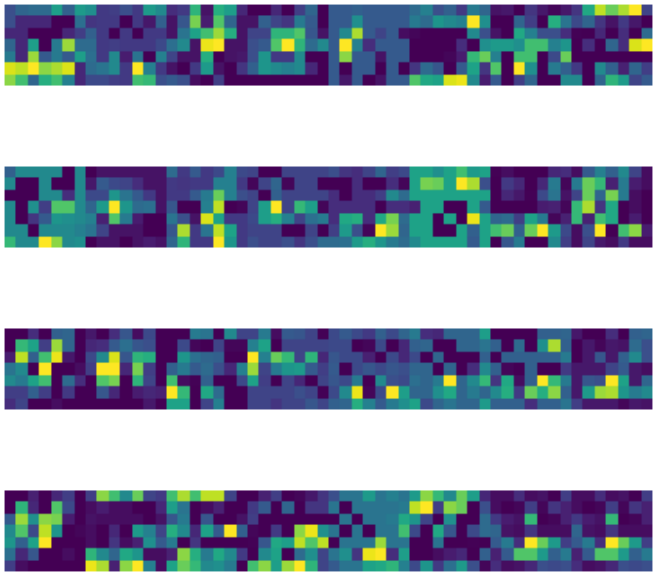

Fig. 8. Letter u after the second convolution.

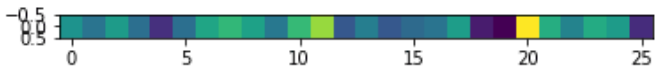

Fig. 9. Fully Connected Layer

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.

$$Accuracy = 0.91$$

$$Loss \quad and \quad Accuracy:$$

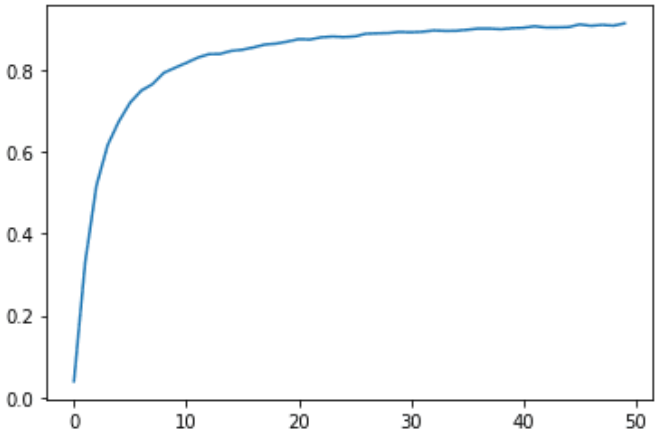

Fig. 11. Accuracy.

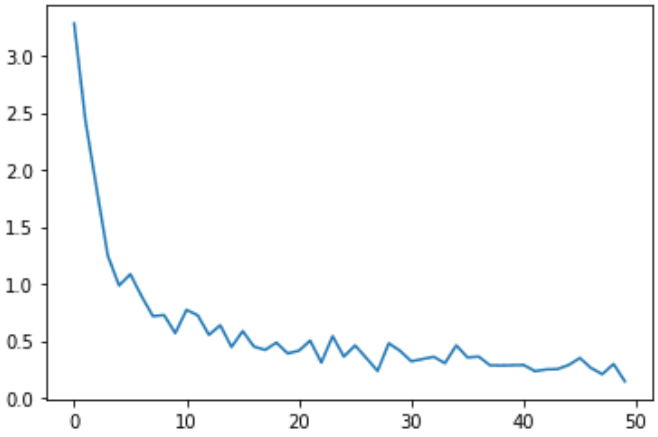

Fig. 12. Loss.

III. Reference

A. Tools

$\boldsymbol{Netron}: Model \quad visualization.$

B. Libs

$\boldsymbol{Torch}: Main \quad Lib \quad to \quad build \quad CNN \quad Model.$
$\boldsymbol{PIL.Image}: Images' \quad pre-processing.$
$\boldsymbol{matplotlib.pyplot}: Plot.$
$\boldsymbol{cv2}: Classification \quad of \quad Dataset.$

C. Code

$https://blog.csdn.net/SESESssss/article/details/111570525$
EMNIST dataset pre-processing.
$https://blog.csdn.net/qq_41845951/article/details/118814447$
Visualization and CNN frame.