Project Report

Xiaofei Hou, Zirun Lin, Zhiwen Tan, Yikun Zhang

Abstract—**Handwritten digit recognition is a classical problem of Machine Learning. In the past few years, several methods were created and applied such as Support Vector Machine(SVM) , Naive Bayes and Artificial Neural Network. Our project will focus on one of the deep learning methods, Convolutional Neural Networks(CNN), to implement handwritten digit recognition. In our experiment, CNN has a better performance than leaner classification and MLP, the overall accuracy is about 92% on the given data set. In this case, CNN can be considered as an effective method to recognize handwritten digits.**

Index Terms— **Machine Learning; CNN; Image Preprocessing; Handwritten Digit Recognition**

I. INTRODUCTION

WITH the growing popularity of handwritten character recognition systems, several methods are developed but there are still many misjudgments occur in some smart devices system.

There are many algorithms to recognize handwritten digits, such as BP, SVM, KNN neural networks. In 1990, The group of Yann Lecun[1] used Back-Propagation Neural Network (BP_NN) to achieve handwritten digit recognition. The targeted data set is the zip code of the United States Postal System.The result showed accuracy is about 90%, misrecognition rate is 1% and rejection rate is 9%. In 1998, the group of M.A. Hearst proposed the Support Vector Machine (SVM) algorithm[3]. This algorithm is an excellent classification and regression algorithm based on statistics. In 2003, A. Bellili proposed a hybrid MLP-SVM algorithm[2] for unconstrained handwritten digit recognition. This algorithm uses a support vector machine (SVM) to improve the locality around the separation surface between each pair of digits in the input pattern space. Regional multi-layer perceptron (MLP) performance. In 2012, the group of Xiao-Xiao Niu proposed the combination of CNN and SVM to improve the accuracy of recognizing handwritten digits[4]. Many researches shows that CNN had great potential and prospect in the image classification field[5,6].

In this project, we develop two .py files. train.py shows the process that trains the model by using the provided data. And test.py is the runnable file for testing.

In the train.py file, our goal is to train our model to distinguish among handwritten digits by using CNN. First, we create our datasets by taking photos of handwritten digits 0-9. To avoid the influence of noise, we implement image preprocessing to do image preprocessing[7,8]. By finding the four edges of digits in the images, we can extract the main part of digits. Therefore, we can also reduce the sum of image pixels which reduce the complexity of the training process. Then, we resize all the extracted digits to the same size. After that we construct model by using keras library[9,10]. We train the model by inputting all the datasets and save the model into the digit_recognition.json file. And the parameters of weights are saved in weights_records.h5 file.

In the test.py file, the program loads the model and weights from digit_recognition.json and weights_records.h5 files. Then the test data will be processed by image matting and resizing. After that, the program would start to test the input data and output the accuracy and the classification results which will be saved in results.npy.

II. IMPLEMENTATION

A. *Image Preprocessing*

The goal of image preprocessing is to reduce the dimensionality of the feature matrix. We import the resize function from the cv2 package. However, if we resize the image matrix directly, the performance will be not as good as we expect. Although the sizes of each image are consistent in the dataset, the sizes of each digit are different. Some digits occupy almost the whole region of the images while some digits occupy only a small region. Then if we resize all of them into small sizes, the useful information might be lost for the small written digits. In order to solve this problem, we design an algorithm to extract the region of digits from the images and crop the blank region before resizing.

The structure of the preprocessing is as following:

For each image matrix:

(1) normalize the pixel values to [0.0, 1.0]
(2) find the upper edge of the digital region
(3) find the lower edge of the digital region
(4) find the left edge of the digital region
(5) find the right edge of the digital region
(6) cut off blank region and resize the image to small size

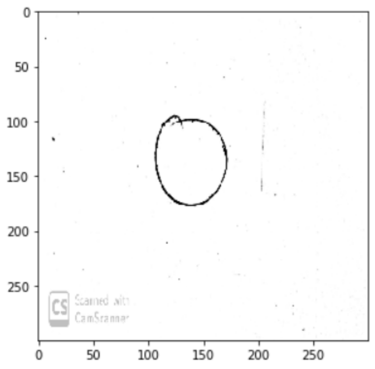

Figure 1. The image sample

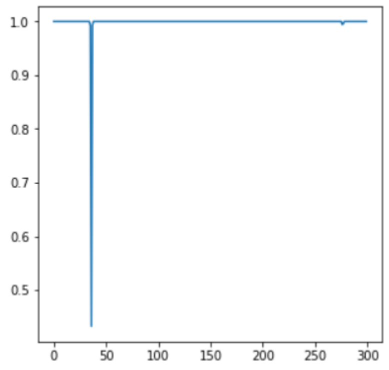
Figure 2. The first row of the image sample

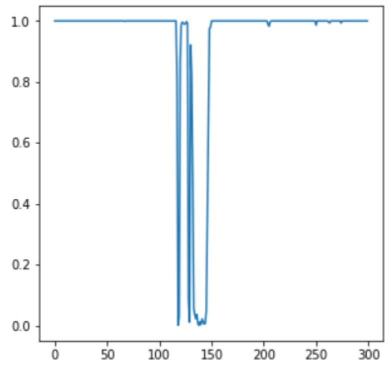
Figure 3. The 97th row of the image sample

We use the method of scanning lines and looking for gaps of pixel values to find edges. Because we normalized the pixel values to [0.0,1.0], we can set a standard threshold to identify digit and background for all images. As shown in figure 1, this is an image sample of digit 0. For the example of finding the upper edge, we scan each row from the first row to the last row. Firstly, we calculate the difference between minimum value and maximum value, if it is larger than the threshold, then this row may have overlap from the region of digit. But it also has potential noise points which result in the gap, as shown in figure 2. In order to rule out the possibility of noise, we add two constraints to the process. The first constraint is that there must be more than two consecutive points who have a gap with the neighboring points, as shown in figure 3. The second constraint is that the pass of the first constraint must occur consecutively for several times, because the pixels of digits are continuous but the noise is separate. We set a parameter "cnt_up", which means least count of success. In addition, in order to obtain the whole digit, we reserve several rows up to the upper edge. We set a parameter "init" to represent the number of reserved rows. Once we find the edge of the region of digit, we will return the combination of the edge and "init", otherwise we return 0 which means that we reach the first row. For the four finding edge functions, they all have four parameters: image matrix X, threshold, cnt_up, init and one return: the index of the edge. The four parameters for each function will be obtained by experiment.

B. *Structure of Convolutional Neural Networks*

To construct the CNN model, we import the keras sequential model which allows us to add layers sequentially. As we all know, CNN has the property of preserving spatial information. Like the MLP, we usually apply a convolutional layer with a relatively small kernel size (3*3 or 5*5) to draw the spatial features, so, we add the first layer with the parameters: filters = 16, kernel_size=(3, 3), padding='same', input_shape=(32, 32, 1), activation='relu'. Then, in our project, we add another convolutional layer with a larger kernel size to describe more general features, it has the parameters: filters=36, kernel_size=(8, 8), padding='same', activation='relu'. The attribute 'filter' represents the dimensionality of the layer's output space. In order to make the output has the same size of input, we need to pad the input data and set 'padding' to 'same'. When the convolutional layer is input layer, we have to assign input size which, in our project, is 32*32*1. Also, we set 'activation' to 'relu' which is very popular now, relu can make the convergence faster and has a relatively low computing complexity. Then, we try to reduce the dimensionality of feature space in order to avoid overfitting while the main features should be preserved. In this case, we insert two max pooling layers and set the size to 2*2.

Before we add the dense layer (128 neurons with 'relu') which is the transition between convolutional layer and output layer, we should flatten the output of convolutional layer. Then, we add our output layer which has 10 neuron and 'softmax' activation function. Softmax can make our output layer output a vector and it is very popular in classification network. What's more, to reduce the potential of overfitting in dense layer, we added dropout and set size to 0.5 like we did in MLP model.

The structure of our CNN is shown in figure 4.

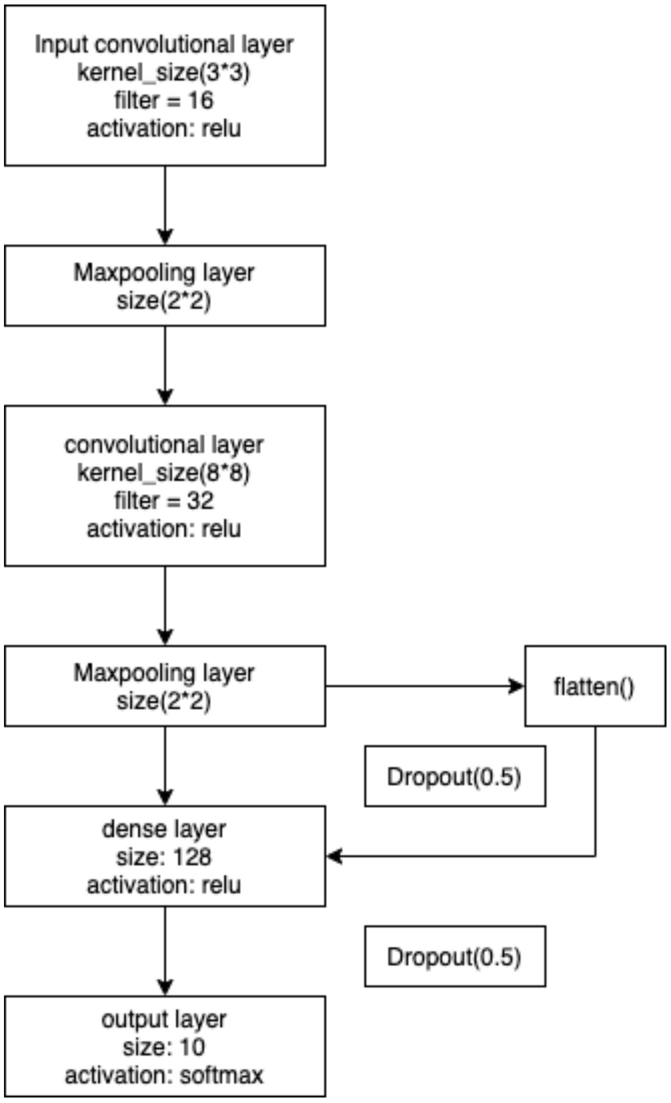

Figure 4. The Structure of CNN

III. EXPERIMENTS

A. Parameters Determination of Image Preprocessing

In order to determine image preprocessing parameters, we use loop and imshow() function to test the performance of the preprocessing algorithm. If the digit region can be extracted completely, the parameters will be approved. Initially, set the threshold to 0.5, cnt_up to 5, and init to 0.

The first experiment: change the value of threshold and find optimal threshold. We found that if the threshold is large, such as 0.5, some written digit edges with light color cannot be found. The extracted digit would be incomplete. If the threshold is too small, such as 0.2, some noise could not be ignored. The extracted region would cover some blank regions with noise. The result is that 0.3 is optimal for this dataset.

The second experiment: change the value of "cnt_up" and find optimal "cnt_up". We found that if cnt_up is large, such as 10, the noise could be ignored for all training data. However, some thin digits can not be recognized. For example, the digit "1" is too thin. If the cnt_up is larger than 2, the left edge and right edge cannot be found. And for another example, the upper part of the digit "7" is too thin, some written digits are in light color and broken shape. If the cnt_up is larger than 2, the upper edge of "7" might be incorrect. The digit will be incomplete. In the end, we set cnt_up to 2 for upper, left and right edge finding function. And we set cnt_up to 4 for lower edge finding function. This allows us to remove most of the noise.

The Third experiment: find optimal "init". Because the digit "1" is too thin, if we do not preserve extra columns, the resize process will result in terrible deformation. As a result, we set "init" to 20 for left and right edge finding functions. And setting "init" to 5 is enough for upper and lower edge finding functions.

Besides the above three tests, we also compared the performance of image preprocessing with PCA. For image preprocessing, we resize the image from 300x300 to 32x32. For PCA, we import the PCA function from sklearn, and set n_components to 0.98. The result showed that the accuracy of classification with PCA is much lower than that of image preprocessing which we designed.

B. Model Training

For the number of layers we set, we need to make sure model has a relatively less complex structure to avoid heavy computing task and a better performance on test data set. What's more, we set max pooling layer and dropout to avoid overfitting because max pooling can reduce the output dimension of convolutional layers and dropout will reduce the amount of wights. Finally, we get the parameters of model which is shown above.

Before we fit our network, we should compile the model first. In our model, we choose categorical_crossentropy as our loss function, the convergence standard of cross_entropy is the similarity between the model's output and data labels. Then, we set 'adam' as the optimizer and the metrics will be 'accuracy'.

After we process the training data, we split the data into training data set and test data set. During the training process, we also set 20% of training data as validation data set. Then we start to train the model we construct. (Once the structure and weights are determined, we need to shuffle all the training set before final training.)

Two crucial factors need to be determined, the epochs and batch size. For epochs, if the epochs is too small, our model can be underfitting. Once it is too large, network can experience the risk of overfitting. Both situations would lead to a terrible classification accuracy on the unknown test data set. We set the epochs from 20 to 200 and found that 50 epochs have the best performance.

For batch size, inappropriate batch size will lead to slow convergence and make the loss function unstable. We set the batch_size from 12 to 240, then we found that when batch size reaches 120, model have the best performance on test data.

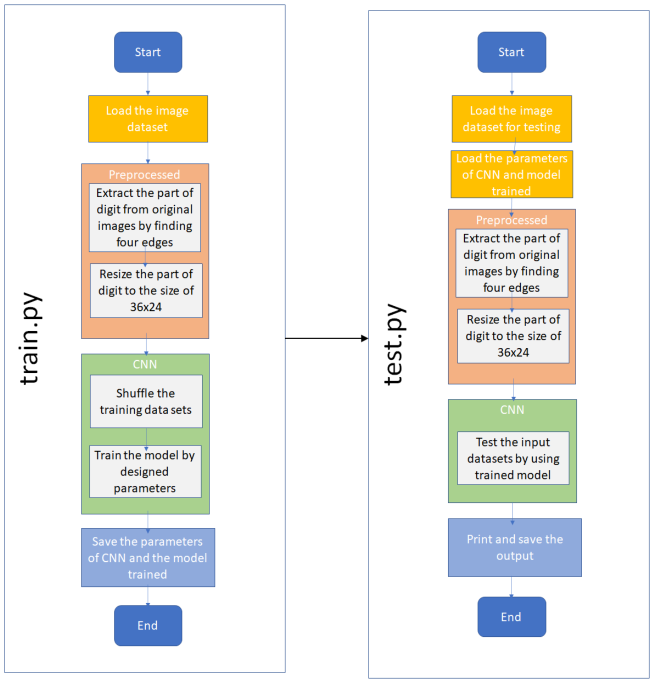
train.py
Start
Load the image dataset
Preprocessed
Extract the part of digit from original images by finding four edges
Resize the part of digit to the size of 36x24
CNN
Shuffle the training data sets
Train the model by designed parameters
Save the parameters of CNN and the model trained
End
test.py
Start
Load the image dataset for testing
Load the parameters of CNN and model trained
Preprocessed
Extract the part of digit from original images by finding four edges
Resize the part of digit to the size of 36x24
CNN
Test the input datasets by using trained model
Print and save the output
End

Figure 5. The flowchart of train.py and test.py

IV. CONCLUSIONS

Compared with PCA, the image preprocessing algorithm we proposed is more appropriate for feature reduction in written digit recognition. The number of samples is small and the difference among the sizes of digits in these images are very large. Special preprocessing is necessary to maintain the similarities among data with the same labels. However, the image preprocessing method we proposed needs precise parameters. These parameters are obtained by experiment based on training data. So, the shortcoming is that it can only identify similar data with high accuracy. If the data has too much noise, the accuracy will drop.

In order to evaluate the accuracy of our experiment, we also perform linear regression algorithm and multiple layer perceptron. The classification accuracy rate of linear regression is 42%. Obviously, regression method is not good for classification assignment. The target labels for linear regression will be a continuous value between zero and nine, if the loss function is $\varepsilon = y - t$, where y is predicted class, t is the target label and the errors samples are assumed independent. With a mean and a variance independent from each other, this will be a signal independent problem.

The classification accuracy rate for multiple layer perceptron is 71%. Although the accuracy rate is higher than linear regression method, it is also not a good algorithm for this project. It is because that the flatten operation will make the images loss spatial information and the training set is not enough.

So, according to our experiment results, CNN has the best classification performance and its accuracy reaches 92% on the test data set (20% of the given data).

REFERENCES

[1] LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD. Backpropagation applied to handwritten zip code recognition. Neural computation. 1989 Dec;1(4):541-51.
[2] Bellili A, Gilloux M, Gallinari P. An MLP-SVM combination architecture for offline handwritten digit recognition. Document Analysis and Recognition. 2003 Jul 1;5(4):244-52.
[3] Hearst MA, Dumais ST, Osuna E, Platt J, Scholkopf B. Support vector machines. IEEE Intelligent Systems and their applications. 1998 Jul;13(4):18-28.
[4] Niu XX, Suen CY. A novel hybrid CNN–SVM classifier for recognizing handwritten digits. Pattern Recognition. 2012 Apr 1;45(4):1318-25.
[5] He K, Zhang X, Ren S, Sun J. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence. 2015 Jan 9;37(9):1904-16.
[6] He K, Sun J. Convolutional neural networks at constrained time cost. InProceedings of the IEEE conference on computer vision and pattern recognition 2015 (pp. 5353-5360).
[7] S. A. Azeem, M. E. Mesentery and H. Ahmed, "Online Arabic Handwritten Digits Recognition," 2012 International Conference on Frontiers in Handwriting Recognition, Bari, 2012, pp. 135-140.
[8] Caiyun Ma and Hong Zhang, "Effective handwritten digit recognition based on multi-feature extraction and deep analysis," 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Zhangjiajie, 2015, pp. 297-301.
[9] A. Pourmohammad and S. M. Ahadi, "Using Single-Layer neural network for recognition of isolated handwritten Persian digits," 2009 7th International Conference on Information, Communications and Signal Processing (ICICS), Macau, 2009, pp. 1-4.
[10] M. M. Abu Ghosh and A. Y. Maghari, "A Comparative Study on Handwriting Digit Recognition Using Neural Networks," 2017 International Conference on Promising Electronic Technologies (ICPET), Deir El-Balah, 2017, pp. 77-81.