Convolutional Neural Network for Handwriting Character Recognition

Zezhou Zhang, Pengyu Zhang. University of Florida

***Abstract*—In this report, we propose a deep neural network to classify handwritten characters. As a step toward this goal, the proposed neural network model is convolutional neural network (CNN) model, which is optimized for multiple features abstraction. Multiple features from the image of a handwritten character can be extracted through different convolutional layers for classification. We demonstrate that an efficient CNN model can be accomplished by choosing appropriate hyperparameters and the number of neurons in each layer. We show that optimization of the model can be up to expectation by modifying the training dataset, such as adjusting all the images to the same size and removing the images from training datasets that are impossible to determine the letters. We also demonstrate that the network can be improved efficiently and prevent overfitting by a method called dropout, which is randomly discarding units to prevent them from co-adapting too much. The empirical study on various test dataset demonstrates the efficacy of the proposed model on its high accuracy of classifying the handwritten characters. This result has shown the potential of CNN for image processing to solve other identification problems.**

I. Introduction

THE convolutional neural network is a class of deep neural networks, and it has been commonly implemented in image processing. In this project, we will build a convolutional neural network to recognize 8 different characters and classify those characters into the right clusters at high accuracy.

II. IMPLEMENTATION

The model we choose to recognize handwritten characters is a convolutional neural network mainly based on the library provided by Pytorch in the environment of Python 3.7. We preprocess the original training data to satisfy the format of input of our model and then build our convolutional neural network. We completed the process of forwarding and backpropagation and choose loss function and optimizer to train our model.

A. *Convolutional neural network*

The first thing we need to think about is the format of the dataset. Basically, find the dimension of each data which is the number of pixels of row and column. Some data from the given dataset do not satisfy the dimension of 50-by-50 pixels. Some data are not even good enough to import into training process. So we resize each image to 48-by-48 pixels to satisfy out later implementation. And we mutually remove some images with very bad performance. For example, half of the images are totally dark and some images do not contain a character.

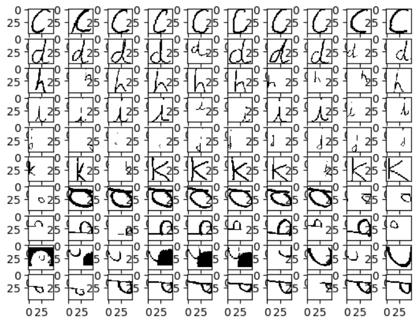


Fig. 1. Unclearly images

We choose a convolutional neural network as the training model because with so many pixels to analyze (48*48=2034), we need to collect massive kinds of information and at the same time, we want to avoid running into gradient vanishing problem. We will also find that by using fully connected networks, the number of parameters is growing very fast which will cause decreasing in training speed or implementing impossible. Under this circumstance, we need to choose a training model that can find the correlation between adjacent features in one image and finding a method to extract those features after which will reduce the unnecessary nodes in traditional neural networks. That is the reason why we choose the convolutional neural network as a very appropriate mode.

The first thing on which we should focus is how to extract features from our raw dataset. In the convolutional neural network, the first layer named the convolutional layer is implemented to mapping features from raw data by using multiple filters. From the definition of convolution and the size of our raw data, each 2D filter will produce their 2D output with a specific stride. In this project, we choose 5 for kernel size and 1 for stride. For each filter, there will be an output. Therefore we need multiple filters whose number is 64 to extract enough features from original data. The elements in every 5-by-5 filter window can be treated as weights which need to be trained in backpropagation

As we mentioned above, after passing raw data through the convolution layer, for every input data we will get 64 different

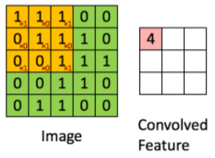


Fig. 2. Convolution of a 3-by-3 filter in 5-by-5 image. [1]

outputs whose format is 44-by-44(48-5+1). Since in the process of forwarding pass, we just simply multiplying inputs by weights and adding bias, we can only get linear combinations as a result. But in neural networks, there always exists a non-linear relationship that can be approximated by introducing a non-linear function named activation function. Therefore, the next step needed to do is implementing activation function. In this handwriting recognition project, we choose rectified linear units (ReLU) as the activation function

One important part of the convolutional neural network is a process named "pooling". Pooling has a similar function with convolution, however, the basic difference between pooling and convolution is that we do not need to train elements in pooling filters as we need to do to convolution filters. Pooling process help to reduce the number of parameters in the neural network as well as making features to be more robust to rotation and scales changes of input images. In our project, we choose 128 pooling filters to be 2-by-2 window with stride 2 in both x and y-direction. In this situation, we use max-pooling which means we take the maximum value of every 4 elements as the new value of one element in the new image. Another important thing we should mention is that if we use 2 as the value of stride, there will absolutely no overlapping during the process of pooling. So we need to add extra rows and columns in the image before implementing pooling to ensure the 2-by-2 pooling window can operate correctly with a stride of [2,2]. The process of adding rows and columns is called "Padding". The output size of any dimension from convolution or pooling can be computed by the following function

After completing the process of convolution and pooling, we have already obtained 128 different images whose dimension is 12-by-12 for one single input. Then we need to import our 128*12*12 pixels into a fully connected layer after flatting every 12-by-12 dimension data to 144 rows. As we talked above, the convolutional layer and pooling layer focus on extracting multiple features from one image Then a fully connected layer works as an interpretation based on the rich-information data and will produce a classification result. How to implement a fully connected neural network depends on how many layers we want and how many neurons we need for each layer. In this project, we choose three-layer fully connected neural networks and 1000 neurons for the hidden layer. Because we want the outputs the classify each input into one cluster, it's reasonable to choose 8 neurons in the output layer.

$$W_{out} = \frac{(W_{in} - F + 2P)}{S} + 1$$

Where W_{out} and W_{in} are the widths of the output and input data. P is the value of padding, S is the value of stride and F is the size of the filter. In order to keep the width of input data and the width of output data the same value, we need to set value of padding to be 2. [2]

B. *Build the model*

Our model is generated by a deep learning framework named "PyTorch". PyTorch is an open-source machine learning library based on the Torch library [3] used for applications such as computer vision and natural language processing. It is free and open-souces software released under the Modified BSD license. More concretely, we use "nn.Module" as the basic PyTorch class to build our model which can be seen in the diagram below. In our model, we import a dropout layer which will randomly drop out neurons during the training process. Adding a dropout layer can essentially help to avoid overfitting [4]. Before starting training our model, we choose Cross Entropy [5] Loss as the loss function defined as "criterion" and Adam optimizer [6] defined as "optimizer". In the training loop, we pass the true image labels as well as model outputs to Cross Entropy function firstly, then performing backpropagation and optimizing the parameters in our model by using optimizer function. During the training loop, we split the original dataset into small batched to implement Mini-Batch Gradient Descent which is a very common method used in the deep learning field.

After finishing the training loop, we used testing dates which are randomly split from the whole dates to test our model. The output neuron with the highest value will be the prediction of the model. Then we print the accuracy of our prediction We choose 25 as the value of epoch and 0.0005 as the value of learning rate. We also print the loss for every 5 batches in one epoch in order to clearly show the relationship between loss and number of epoch. Our result will be in the following figure and table.

III. EXPERIMENTS

In our experiment, we mainly discuss the situation that how the accuracy will be influenced when changing learning rate, the value of epoch and different methods of initialization. All experiments are based on Mini-Batch Gradient Descent.

A. *Initialization*

The default initialization in class "nn.Module" is based on "kaiming.uniform," [7] which is also called "He initialization". PyTorch also provides "kaimin.normal" for choosing. In our experiments, we compare these two different initialization and found that the accuracy is much higher if we choose "kaiming.uniform" when making the value of epoch and value of learning rate fixed.

B. *Value of epoch*

We choose 6 as the value of epoch with learning rate as 0.001 and "kaiming.uniform" initialization. Then increase the value of epoch by 1 for every experiment. The loss of every prediction decrease as the value of epoch increase until the value of epoch increase to 25. Therefore, it is reasonable to set the value of epoch to be 25

C. *Value of learning rate*

After ensuring the epoch and method of initialization, we design multiple experiments to find what value of learning rate is appropriate to find the local optimum. From the experiments of changing the value of epoch, we found that the value of loss fluctuates especially in the middle of every training process which means the learning rate is not appropriate. Then we take more experiments while decreasing the value of learning rate from 0.001 to 0.0005 at the step of 0.0001. The value of loss remains to decrease steadily and only fluctuates very small scale at the end of every training process.

Table I				
Epoch	Learning Rate	Initialization	Loss	Accuracy
10	0.0005	Uniform	0.5439	86.25%
10	0.0001	Uniform	0.5847	85.78%
15	0.0005	Uniform	03518	89.06%
15	0.0005	Normal	0.6430	69.80%
20	0.0005	Uniform	0.3329	91.03%
25	0.0005	Uniform	0.1843	96.25%

Table. 1. Parameters from experiments

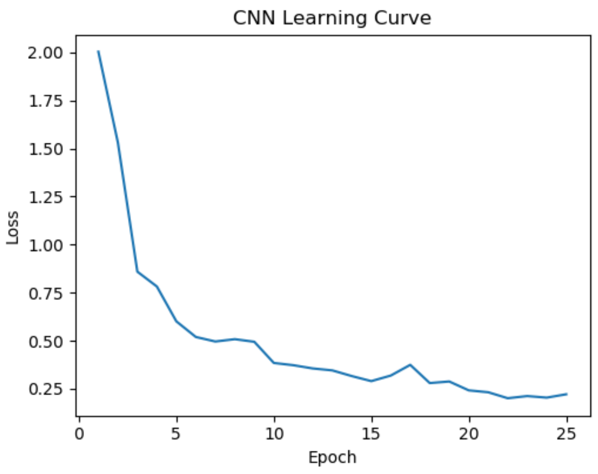


Fig. 3. Learning Curve of CNN model from 25 Epoches Trained

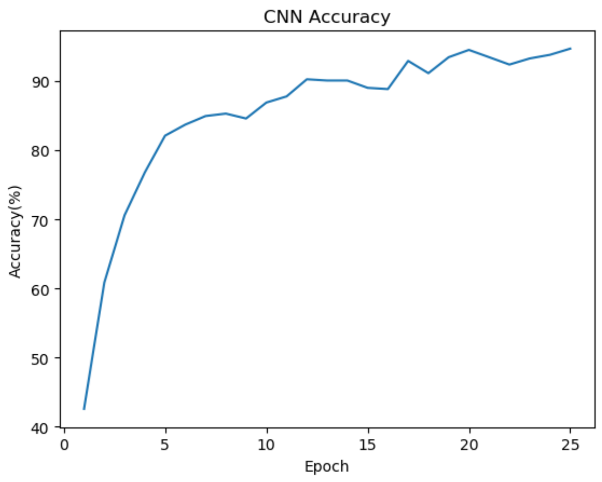
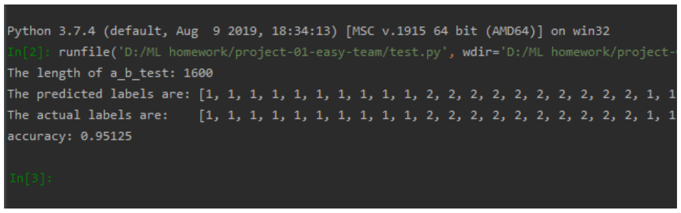


Fig. 4. Test Accuracy of CNN model from 25 Epoches Trained

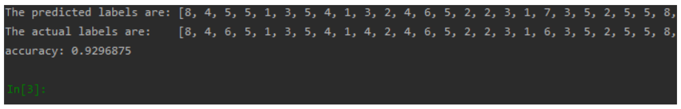
IV. CONCLUSION



```
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] on win32
In[2]: runfile('D:/ML homework/project-01-easy-team/test.py', wdir='D:/ML homework/project-
The length of a_b_test: 1600
The predicted labels are: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1
The actual labels are:    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1
accuracy: 0.95125

In[3]:
```

Fig. 5. The Predicted Labels and Actual Labels of Only ‘a’ and ‘b’ Test Data



```
The predicted labels are: [8, 4, 5, 5, 1, 3, 5, 4, 1, 3, 2, 4, 6, 5, 2, 2, 3, 1, 7, 3, 5, 2, 5, 5, 8,
The actual labels are:    [8, 4, 6, 5, 1, 3, 5, 4, 1, 4, 2, 4, 6, 5, 2, 2, 3, 1, 6, 3, 5, 2, 5, 5, 8,
accuracy: 0.9296875

In[3]:
```

Fig. 6. The Predicted Labels and Actual Labels of All Test Data

In this paper, we proposed a CNN architecture to accomplish the feature extraction of handwritten character images. After several experiments and learning curve to find the appropriate hyperparameters and numbers of neurons in each layer, we make an efficient neural network model. In Fig. 5 and Fig.6, we can demonstrate that hard and easy test datasets can get high accuracy on classifying letters from the trained CNN model, so it is concluded that CNN can provide great performance on image processing for classification.

REFERENCES

[1] N. Papernot, “Convolutional Neural Nets in PyTorch,” *ALORITHMIA*, Apr 2018. Accessed on: Dec. 1, 2019. [Online]. Available: https://algorithmia.com/blog/convolutional-neural-nets-in-pytorch

[2] N. Kumar, “Visualizing Convolution Neural Networks using Pytorch,” *Towards Data Science*, Oct 2019. Accessed on: Dec.1, 2019. [Online]. Available: https://towardsdatascience.com/visualizing-convolution-neural-networks-using-pytorch-3dfa8443e74e

[3] Wikipedia contributors, "PyTorch," *Wikipedia, The Free Encyclopedia*. [Online]. Available: https://en.wikipedia.org/wiki/PyTorch

[4] S. Nitish *et al*., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*., vol.15, pp.1929-1958, Jun 2015.

[5] D.Boer *et al.*, “A Tutorial on the Cross-Entropy Method,” Ann Oper Res, vol.134, pp.19-67, Feb 2005. DOI: 10.1007/s10479-005-5724-z

[6] P.Kingma *et al.*, “Adam: A Method for Stochastic Optimization,” *Cornell University*., 2014. Accessed on: Dec. 2, 2019, [Online] Available: https://arxiv.org/abs/1412.6980

[7] X.Zhang *et al*., “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” *Computer Vision and Pattern Recognition*., 2015 Accessed on: Dec. 2, 2019, [Online] Available: https://arxiv.org/abs/1502.01852