Final Project Report for CS 175, Spring 2018

Project Title: Hand Written Digits Recognition
Project Number: Team 67

Student Name(s)
Shen Cheng, 12367802, shenc8@uci.edu

General Instructions:

- Your report should be 5 to 7 pages long in PDF

- If you want to add more details (e.g., additional graphs, examples of your system's output, etc) beyond the 7-page limit, feel free to add an Appendix to your report for additional results

- **What the Team submits to Canvas (one student submits the items below on behalf of the team)**
    - Your report entitled **FinalReport.pdf**
    - A zip file called **project.zip** that contains the following in 1 directory called project/
        - A README file that contains a 1 line description of each file in project/
        - A Jupyter notebook (called **project.ipynb**) that can be run directly and that demonstrates your project. Your notebook can import a sample of the data that you used, import 1 or more models that you built, and generate examples of the types of predictions or simulations your model can make. The notebook should not take any longer than 1 minute to run in total (if you have models that require a lot of training time, train them offline and just upload the models and some sample data to illustrate them). Feel free to generate examples of your model(s) in action, e.g., for reviews you could generate examples of reviews where the models work well and reviews where the models work poorly.
        - Also save a .html version of your notebook called project.html, showing the outputs of all the cells in the notebook.
        - Upload any data files needed to run project.ipynb – keep your data sets to 5MB in total or or less.
        - Also include a subdirectory called src (within the zipped project/ directory) with all of the individual code (scripts, modules) for Python (or equivalent for other languages) that your team wrote or adapted– these don't need to be called by the project.ipynb notebook but need to be in the src/ directory
        - Note that we don't necessarily plan to run all your code, but may want to look and run parts of it.

    - **Individual Contribution**

Each team member needs to submit a ½ page of additional text as an individual, titled "**IndividualContribution_ID.pdf**" that provides an honest assessment of which parts of the project you contributed to and which parts were worked on jointly. This should be written individually – you may wish to discuss the plan of what you will write with your project partner, but the page you write should be generated separately by each individual.

1. Introduction and Problem Statement (1 or 2 paragraphs)

My project aims to recognize handwritten digits using CNN model based on MNIST data. There's a more advanced technique called OCR (Optical character recognition). It aims to recognize typed, handwritten characters taken in real life through Artificial Intelligence. There are many fields that utilize the technique such as surveillance cameras to recognize license plate. Because of the data I found is MNIST Data which only contains digit images, I only deal with recognize handwritten digits.

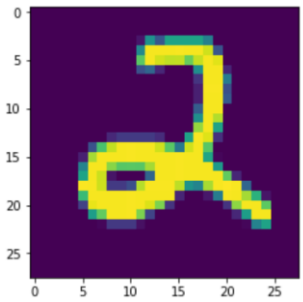
Basically, it contains three parts in this project: character segmentation and train a CNN based on MNIST data then convert segmented images to MNIST data for input use. For the character segmentation, it aims to use OpenCV to cut character based on contours. Right now, it works to recognize digits from 0-9 written on any size of paper (it can ignore some small noise but cannot recognize something close to digits like characters). For training model, it aims to reach a CNN model which reaches close to 100% validation accuracy based on Kaggle MNIST datasets. Right now, my CNN model reaches 98.45% accuracy in validation data. For the last part, it aims to convert image to a numpy array with 784-pixels values which should be the same to MNIST data and recognize them used trained model. Right now, it can be successfully converted and digits can be recognized by trained model except some minor mistakes like distinguishing between 5 and 6.

2. Related Work: (1 or 2 paragraphs)

During this quarter, the course mainly used the image classification to teach us things like different methods in multi-class classification (SVM, SoftMax, KNN and so on), different layers and methods in a Convolutional Network (Batch Normalization, Dropout layer, SGD and so on). Also, how to use PyTorch or Tensorflow. Because I used PyTorch to train the model, I used partial work in assignment 4 to build PyTorch functions like checkaccuracy (), train (). And I asked in the Piazza and this is allowed. Also, when I build the model, I follow one of suggestion in the instruction of assignment 4 which all parts have been covered in the course to build the model. I evaluate the performance and it reaches a very high result. So, I didn't build a new algorithm.

3. Data Sets

The database I used for single digit recognition is called MNIST Data (Modified National Institute of Standards and Technology database). Each example image is 28 pixels in height and 28 pixels in width. In total, there are 764 pixels (28x28). Each of pixel is assigned with a value between 0-255. For the blank part (background), it's pixel value 0. So, one sample data is an numpy array with 784 pixels values as features and 1 value as target (range from 0 to 9), here's one example shown in matplot.



In total, there are 42000 samples in train dataset. 90% of them for train use and 10% of them for validation use. Also 32000 samples in test dataset (can be submit in Kaggle to evaluate). The URL for the dataset is https://www.kaggle.com/c/digit-recognizer/data.

For input image used in image segmentation and later test, the dataset consists of 20 images with handwritten digits on blank background. The number is random and each image is named with its real number in the paper (e.g. 123.jpg stands for 123 written in the picture). Most of the images are written by myself and put in folder called 'real'. Here's one example of input image.

4. Description of Technical Approach [at least 1 page]

Training CNN model – Data preparation

Train Test Data Split: convert training data to 90% for train, the rest for validation (train_test_split function in sklearn library)

Data Normalization: divide the training set by 255 (a maximum pixel value), a common and simple way to standardize and centralized the pixel value dataset.

Greyscale: reduce colored part of an image. Reduce the complexity of features in data points

PCA: try to use PCA to reduce dimensionality of data. But instead, getting a lower accuracy (around 92%) so give up this method

Data Loader: a method provides by Pytorch to process dataset by doing things like shuffle, divide into batches

Resize the image data: resize the data into 1 x 28 x 28, 1 refers to the color channel (since only one color), 28 refers to width or height of the image which is consistent with the data provided.

Train CNN model – build and train model

Convolutional layer: core part of CNN model which builds the neuron structure of CNN model

ReLu layer: activation function use function max (0, x);

Batch Normalization layer: normalization on data, which allow more flexibility in initialization and allows higher learning rate.

Drop Out layer: reduce features to train, which helps avoid overfitting.

Cross Entropy Loss: use SoftMax method as the output layer. SoftMax loss is interpreted as the distribution of values. It asks the max class value to more proportional than the others to be taken.

Adams: SGD method

Graph Segmentation

Resize: use CV2 resize to control the size the image to control the proportion of taken digits

Convert to white digits and black ground image, use CV2 apativeThreshold to sets digits as pixel value 255 (white), also adjust (increase) the block size to ignore noise in the image

Find Contours: use CV2 find contours function to locate the parts with contours (from my understanding, any parts which has large contrast to background) under a black background

Bounding Rectangle: use CV2 bounding rectangle function to locate digit based on results from find contours.

Convert to MNIST data

Resize: use CV2 resize to adjust the digit to 20x20 size

Build background: create a new Image object with 28x28 size, 'L' (white) color

Paste image: use image paste method to paste the digit into the background, also adjust by setting box parameter as 4x4;

Get pixel values, Image.fromarray to get pixel values of the image

5. Software [at least ½ a page]

a. Pytorch Model: Use provided Pytorch library to build the CNN model

Training Data Conversion: By observing how CIFAR10 is loaded by Pytorch in assignment (need to build a class to handle the dataset) to allow Pytorch functions, write my own python class which did jobs like __getitem__, process the dataset like normalization.

Graph Segmentation: Learn Graph Segmentation techniques from instructions online (https://www.pyimagesearch.com/2014/04/21/building-pokedex-python-finding-game-boy-screen-step-4-6/), stackflow or other places since it's not taught in the class. But situation varies a lot between my case and online case. Write my own functions which adjusts to the program.

Convert to MNIST dataset: Learn the method from https://www.youtube.com/watch?v=yi_dDsRqvK0 since it's not covered in the class. But situation varies a lot between my case and online case. Write my own functions which adjusts to the program.

b. Pytorch training, check accuracy method. Follow the code from assignment 4 to build basic use of Pytorch functions for training the model

Use Jupiter notebook as we did usually, also consider to Google Cloud but since the accuracy is extremely closed to perfect and I used up Google Cloud free trial in another course so give it up.

6. Experiments and Evaluation [at least 1 page]

Training CNN model

(a). To build a CNN model, I followed my observations from assignment 4 about how to build an effective, complex CNN model, how parameters in each layer influence the performance and which optimizer works better for large batch size. Here are following things I did to build the model

Convolutional layer: the larger output channel is, usually have a better performance. However, it creates a larger workload for GPU. Sometime, my GPU will shut down because of overloading. Set 64 in the first one and 80 in the second one

Batch Normalization layer: allow more flexibility in initialization and allows higher learning rate.

Drop out layer: drop features to avoid overfitting. Try to do without dropout layer. Although very low entropy loss I have got on training data, it gets the performance worse because of overfitting.

Cross Entropy Loss: SoftMax calculate the loss using log-probability. It asks the max class value to more proportional than the others to be taken.

Adams: in assignment 4, I get the conclusion by comparing performances between different optimizers. It's true that Adams performs the best as the batch size gets large.

PCA: tried PCA but gets a lower score (92%). Can't understand why so comment out

b) Training loss during 5 epochs.

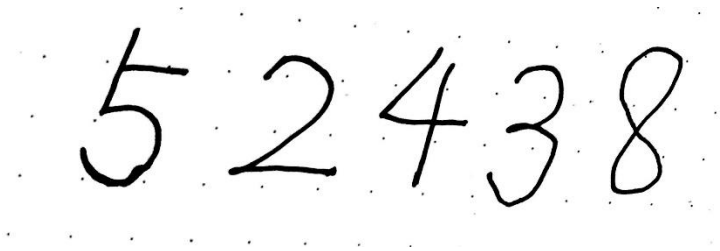
```
Starting epoch 1 / 5
t = 1000, loss = 0.2340
t = 2000, loss = 0.3046
t = 3000, loss = 0.8837
t = 4000, loss = 0.0689
t = 5000, loss = 0.0608
t = 6000, loss = 0.0585
Starting epoch 2 / 5
t = 1000, loss = 0.0245
t = 2000, loss = 0.0220
t = 3000, loss = 0.3745
t = 4000, loss = 0.0338
t = 5000, loss = 0.1487
t = 6000, loss = 0.0049
Starting epoch 3 / 5
t = 1000, loss = 0.0383
t = 2000, loss = 0.0989
t = 3000, loss = 0.0730
t = 4000, loss = 0.0097
t = 5000, loss = 0.0480
t = 6000, loss = 0.2690
Starting epoch 4 / 5
t = 1000, loss = 0.0069
t = 2000, loss = 0.2254
t = 3000, loss = 0.0007
t = 4000, loss = 0.0107
t = 5000, loss = 0.0052
t = 6000, loss = 0.0197
Starting epoch 5 / 5
t = 1000, loss = 0.0033
t = 2000, loss = 0.0196
t = 3000, loss = 0.0085
t = 4000, loss = 0.0003
t = 5000, loss = 0.0047
t = 6000, loss = 0.0031
```

Final Validation result

```
Got 4159 / 4200 correct (99.02)
```

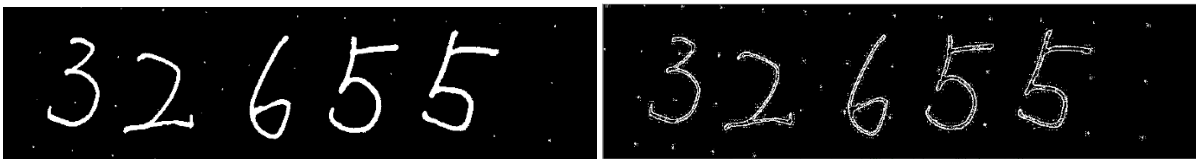
Graph Segmentation

(a) Biggest problem met is the noise in the images. Because it didn't put the image under a real situation, it basically determines by size to exclude noise (not a classification model). Here' s example noise we can find in the image: the dot, and multiple contours in digit 8



The later noise results from CV2 find contours function is not a classification method to distinguish which one digit or which one is noise. It can only find all parts with contours, either the dot points or two circle parts in digit 8. So, it's needed to exclude these noises by image processing

By doing so, first is to adjust block window size parameter in CV2 adaptiveThreshold function. So, let's compare a threshold image with high block window size and smaller one.



Obviously if the block window size is large, it ignores minor noise in the image and emphasize digit part. So I choose (41, 40) as parameter.

Second is to ignore contours with noise, I set only width greater than 20 or height greater than 20 (originally use end but find not fit for digit 1) can be accepted. I also adjust the image size to 870 x 225 (The image size I used firstly to adjust parameters) to be proportional to the size of segmented digit.

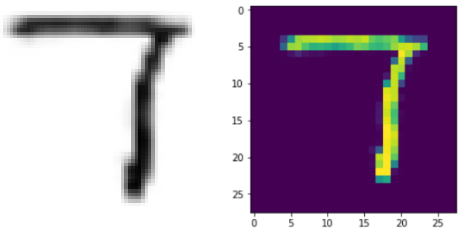
Finally, I make a sorted list which ordered by the x position of segmented digits to keep it in an accurate sequence.

(b) Results will be shown as well as the next part

Convert to MNIST data

(a). After the digit is segmented from the image, it needed to be converted to 28x28 MNIST data form. The first job to do is to shrink the size digit. Because I only know the size of MNIST image but don't know the size of digit in the center part. I tried like 15, 20, 25. And size 20x20 of digit size gives me the best performance. Then I paste it in 28x28 white background. By observing the numpy array of MNIST data, it's with black background (0-pixel value), Don't know this thing at first since using imshow to present MNIST data. So, I use 255 subtract current pixel values to get new ones which contain the similar pattern.

(b). Here's the example of segmented digit (after shrink) and imshow of converted MNIST data



Digits recognition

(a). Make 10 images of handwritten digits (three types) and put into a folder called ‘real’. Each of them is named as the actual value in the image (convenient for compare). Write a predict function. Read images in the folder and print out predicted digits.

(b). Here are results of comparison between real images and predicted digits. There’s a mistake that 5 and 6 get wrong. I think it’s less likely related to CNN model because not much thing I can do to improve accuracy. Instead, it maybe because my handwritten habit (which is not qualified to the pattern in training data) or the image process is not doing well.

```
2 3
real\23.jpg

2 7 3 8 4
real\27384.jpg

3 2 5 5 5
real\32655.jpg

3 4 0 0 2 2 4
real\3400224.jpg

4 3 3 1 2
real\43312.jpg

5 2 4 3 8
real\52438.jpg

6 8 8 7 2
real\68872.jpg

7 5 4 3 2
real\76432.jpg

8 3
real\83.jpg

9 8 7 5 5 2
real\987662.jpg
```

6. Discussion and Conclusion [at least ½ a page]

The project is closely related to the knowledge I learnt this quarter. It helps me get a more deepening understanding about CNN, Pytorch and anything else. I think it’s not quite hard for me in this part. One reason is that I distinguish the digit based on a special circumstance and just use basic computer vision technique rather than using a trained model to classify. I think I get an almost good result in this project.

However, it's just an extremely simplified project. For the real practice of OCR, there's a more complicated real circumstance. How to deal with this problem will be more important for me if I want to make more progress. Right now, as I knew from online OCR technique is much more sophisticated. I think I can work not only configure out digits but characters and even foreign language (quite different from English letters). And combine with outside device like the camera to make it more practical.