Receipt Scanner with Convolutional Recurrent Neural Network

Cai Ting, Chen Ke, Wu Zefeng, Yang Zhuohan, Zou Yutong

National University of Singapore
A0162536E, A0148045H, A0147971U, A0147995H, A0162510W

Abstract

In Singapore, many university students or young people who just step into society choose to share accommodation with roommates in order to ease the rent pressure. However, this always brings the trouble of splitting the bill after dining or grocery shopping together, since the data have to be manually keyed in and computed for individuals' share of the expenses. In this paper, we are to provide a solution that is intentionally designed for receipt Optical Character Recognition. This application should have the following properties (1) be of relatively high accuracy in terms of character recognition. (2) fast processing speed. (3) interpret recognized results in a more user-friendly way. Here we will first have a short introduction on existing solutions, which is followed by application specifications. Then, the particular Machine Learning algorithms and other technologies utilized will be introduced. After that, experiment results and corresponding analysis will be discussed. In the end, we will be providing reflections and suggestions for further developments and explorations.

Introduction

The receipt scanner is a typical Optical Character Recognition problem and numerous solutions have been provided. One of the most common approaches should be Neural Network, which is a popular machine learning model. After research and experiments, we decide to use a variance of CNN, which is Convolutional Recurrent Neural Network (CRNN) for character recognition.

The Advantage of Neural Network

The main reason why Neural Network dominates in this area is that it is good at pattern recognition and is robust to noises. In application, it should be able to recognize characters under various conditions: receipts may be distorted or not clearly printed; different fonts might be used in different stores; the image resolution could be low; the photo for recognition may be blurred or taken under deem light. In order to handle these, a feasible solution is to learn patterns of different characters and then recognize characters based on the patterns.

As mentioned in Conv Nets: A Modular Perspective (Olah, 2014), CNN is a special kind of Neural Network that can express computationally large models while keeping the number of parameters small. Since recognizing characters needs to identify numerous features, CNN is absolutely an ideal choice to train models and classify characters in images.

Qualitative Advantages of the CRNN

There are many available Neural Network models for character recognition. One of them is Recurrent Neural Network (RNN) which works surprisingly well when the sequence of character strokes is known. It can increase its confidence of classification as the drawing of the character proceeds, which is similar to Google Quick Draw model. However, this model does not suit our application because we are trying to recognize machine printed text without drawing sequence. In such case, RNN seems to be inferior than other models. Thus, pure CNN may be a better solution to the OCR problem. However, CRNN might even fit the application better because it is a combination of CNN and RNN and is context dependent. When the model is trying to classify characters, it is not recognizing single characters but item names and item prices, which are usually combinations of characters. In such case, CRNN should be more likely to reduce misclassifications due to the context dependency. In addition, since CRNN is recognizing a sequence of characters at a time, it is more efficient than CNN.

Specific Requirements Satisfactory

From our perspective, CRNN could satisfy the first two requirements of the application. Firstly, since it is context dependent, given some prior knowledge, CRNN may achieve better accuracy. For instance, "q" and "9" are hard to be differentiated. However, given the knowledge that no combination of English characters and integers would ap-

pear in a single word, which can be specified when training the data, the misclassification is likely to be eliminated with the help of RNN. Secondly, since CRNN is trying to detect a sequence of characters instead of processing trough deep convolutions individually for each character, the processing speed would be significantly improved.

The last requirement, which is user-friendly display, might not be satisfied. The application should be able to expand abbreviations to readable words. Our intention was to contact specific stores to fetch the full item name datasets. However, this is not a general solution and actually no store accepted our cooperation application. As far as we are concerned, Natural Language Processing (NLP) might be a considerable solution. However, due to time constraint, it is left for further explorations.

Applications of the CRNN Model

The input of our CRNN model should be an image of a receipt and the output should be a correctly classified and formatted text file. The developer of the application can feed in receipt images and filter out item-price pairs in output text for display. Ideally NLP can be used to generate user-friendly item names.

Figure 1 Application Illustration

As for users, they can take a picture of the receipt, wait for the application to process the image, select items, correct misclassified characters and finally split the money accordingly. Figure 1 is an illustration.

The CRNN Model

CRNN is introduced in the paper *An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition* (Shi, 2015). The main concern addressed in the paper is that in real world text comes with sequence. On this basis, the advantage of RNN can be utilized to recognize text sequences on top of the normal CNN and possibly achieve better results.

The Proposed CRNN Architecture

The structure of CRNN contains three main components

- Convolutional Layers
- Recurrent Layers
- Transcription Layer

Convolutional Layers:

The configuration of convolutional layers is rather standard. It is trying to use convolutions with deep feature maps to figure out characteristics of the image. Between convolutional layers, max pooling is used to progressively reduce the spatial size of feature representation. At the end of the convolutional layer, the usual full connected layer is removed. Instead, feature map is divided into single-pixel-wide feature sequences. Then the feature sequence is fed into recurrent layers.

Recurrent Layers:

In this section, bidirectional Long Short Term Memory (LSTM) is used to analyze the feature map provided by CNN and distribute classifications. Traditional RNN actually experiences the vanishing gradient problem (Bengio, 1994). More specifically, as recurrent process proceeds, important features from previous inputs may be dominated by later inputs. As such image-based problems may deal with long sequences, LSTM is introduced. Intuitively, it tries to keep important features in the network cells and pass them to later processes.

Furthermore, for images, contexts from both directions are valuable. As LSTM is directional, analysis from both directions should be considered. Hence, feature sequences are processed into forward sequence and backward sequence. Both sequences are fed into LSTM for classification.

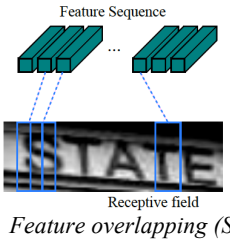

Figure 2 Feature overlapping (Shi, 2015)

Transcription Layer:

Because the feature sequence is generated from CNN, it is possible that adjacent feature units overlap on receptive fields. As a result, duplicated classifications might be produced. Furthermore, it is also possible for a feature unit to

capture redundant information. Hence, outputs of the form "--ss-tate" could be produced. To retrieve actual text, a transcription layer is needed. In our case, we simply remove dashes and join adjacent duplicated characters.

Technical Insight of CRNN

After several rounds of experiments, we have found some interesting aspects of the CRNN model. It is rather powerful but still have some limitations.

Ideally the model is extendable for all languages
After reading the code and training a new model, we realized that the training process does not concern about the meaning of the text. The real output of the model is a sequence of indices, which are mapped to corresponding characters. This would be extremely helpful for our application since it enables different language supports and can be used by people from different countries.

Not good at interpreting sentences
This is one of the main defects of the model. When it is not confident to distribute classification, it will give empty result, which is the same as interpretations for white spaces. As a result, when dealing with sentences it would be hard to find word breaks. One possible solution would be ignoring white spaces for the first place and use language models to add back white spaces. However, this method may increase the computational complexity of the model and extend the training and classification time. Fortunately, for our application, sentence detection is not necessary because we are focusing on only the item names and prices, which are generally short texts.

Misclassification of characters
Interestingly, the actual experiment results show that the model does not work well on enhancing accuracy as expected. For instance, it still mistook "o" for "0" in the word "amount". From our perspective, this is due to lack of information for LSTM to capture the important features. Taking the previous case as an example, "o" is in between of two short English character sequences and hence the model might not be confident to deliver English characters rather than numbers. If there is a long sequence of English characters before or after it, the result might be better. In addition, since the plain texts on training images are randomly generated, the LSTM does not capture the spelling of English words as well. If we could possibly have the data of real grocery item names and use that to generate training data, the classification should be much better.

Make CRNN Fit

In fact, the CRNN model does not fit our application perfectly. This is because CRNN is not able to recognize text from a complicated image. Other than that, when the image size is large (recent cellphones are capturing images with high resolutions), it would not be possible to directly apply models like CNN to analyze those images with large sizes. The process would be computationally complex, which is beyond the capability of a cellphone. We proposed to break down the receipt into lines and further down to individual words. After that we use CRNN to recognize words and format them back to the original positions. In addition, the original model was trained using scene images, which are different from machine printed texts. Hence, model should be retrained. The details will be discussed in the next section.

The Novels

Though OCR is a hot topic, many of the existing projects are focusing on text recognition for scene images. For our application, the scenario is different since the texts are generally smaller and the background of the text is not complex. As a result, we could not find a model that fits our application exactly. We found a CRNN model that analyzes short wordings. Then, we used image processing to break down the receipt into words so that the model can be used. Furthermore, new training data were generated to re-train the model. Originally the model only supports 26 English characters. We further modified it to support more characters such as "?", "*" and digital numbers.

Image processing

We detected and extracted the area of receipts out of the background and segmented the receipts into lines and letters.

Receipt scanning:
With a generated or real receipt image, we first scanned the image to extract the receipt area. We used methods in OpenCV package to first detect the contours of image parts with clear edges, sort them by contour area, and choose the ones with largest areas. Then we looped through them from the largest to the smallest and found the one with 4 corners when approximated as a simpler polygon. Then we use four-point transformation to convert this contoured area into a rectangle.

If the algorithm failed to detect such 4-cornered area, scanning would fail. In the actual testing of this functionality, we found that this algorithm only worked in situations where receipts are in a high contrast monochrome colored background, and the shape of the receipt in the image had

to be almost quadrilateral. Therefore, the scanning did not work well for most real-world receipt images.

Line segmentation:
We processed scanned receipt image to find out the lines in it. After black and white binarization, we found out the contours, and dilated these contours with values of 100 in horizontal direction and 3 in vertical direction. Hence each detected letter (or part of letter) would be stretched horizontally and slightly vertically so that they could intersect with each other. We then regarded these united areas as lines.

However, this was not enough. Usually lines would be correctly detected, but sometimes we found two lines (where one is above the other) vertically joined as well. Because in certain fonts, dilated characters like 'y' in the upper line and characters like 'h' in the lower line would intersect with each other. In order to solve this problem, we removed the outliers in the height values of the lines, took the average of the remaining heights as assumed line height, and forced to separate the line boxes with approximately *n* times the assumed line height into *n* vertical lines.

This is not a perfect solution, because sometimes the receipt contains lines of different font sizes or font types. But this algorithm actually brought some improvements in line segmentation.

Word Segmentation:
For CRNN model, we processed the scanned receipt image to first detect the words in it. Similar to line segmentation, we found the contours of the letters, dilated them with values of 3 in the vertical direction and 15 in horizontal direction, so letters in a word would most likely be joined. We took the bounding boxes of these united areas as boxes of words. We slightly increased the size of these boxes to include some parts of the context. This introduced much noise, but we preferred noise to having cropped letters in the images. Since the training data and testing data are generated with noise included, noise does not affect the performance of the model.

Character segmentation (used for experiment on CNN):
With line outputs from previous steps, we separated each line into individual characters. Similar to line segmentation, we detected contours of character parts, and joined them in vertical direction in order to combine parts of characters like 'i'. Then the bounding boxes of these joined paths were outputted as letter boxes. Some adjacent characters were joined with each other as well due to the font or the blurring. We applied the same method in segmenting wrongly joined line.

Dataset generation

We generated large amount of training and testing data, i.e. the simulated photos of receipts.

Full receipt data generation:
In order to create enough amount of receipt data with labelled text, we used python PIL library to generate simulated photos of receipts.

We drew fake receipt text on a rectangle with random selected paper texture (image A) and added it to a colorful background. After this we blurred (image B) and distorted (image C) the image. Image A, B, C, the original text and the JSON file with generation configuration and fake shopping items were saved in the output directory.

One-line data generation:
We also created a set of training data and testing data consisting of 32 x 100 images of one-line short text for CRNN training and testing.

Text formats include: single word, word followed by ':', word in brackets, integer string, float number string, price text ($12.34, 12.34$), number followed by '%'.

To make these data more natural and harder to recognize, these images are cropped from a context, which means there are noise surrounding the text to be detected. After noise is added, the images are then distorted and blurred. In our model, any image input will be stretched to 100x32 before processing. In order to fit conditions where words consisting of few (one or two) letters are being inputted into the model, such short texts are also generated with paddings so that they will not be stretched into a strange manner when being processed.

The data generation went on smoothly. But we could not perfectly create images similar to real word receipts, which are generally curled, shaded, or sometimes with corner(s) missing. Also, real world receipts are sometimes not clearly printed due to lack of ink.

Model Modification

We have tried both CNN and CRNN models. The CNN model was originally used to classify Hangul characters. Modifying the CNN model to support English letters and other characters was simple, which only required modifying the configuration file. However, the language support for CRNN was hard coded into models. Hence, we manually modified the parameters and added corresponding support for character decoding.

The Experiments, Results and Analysis

We made use of the default structure of the Neural Network models with self- generated data. We have found that larger amount, more complex and noisier data led to better

results. However, the model itself might still need some improvements.

The Configurations and Experiment Steps

We did not do any modification on the existing model structure since it is claimed to be efficient. Our focus was on delivering sufficient training data and investigating whether we could gain good results.

Type	Configurations
Transcription	-
Bidirectional-LSTM	#hidden units:256
Bidirectional-LSTM	#hidden units:256
Map-to-Sequence	-
Convolution	#maps:512, k:2 × 2, s:1, p:0
MaxPooling	Window:1 × 2, s:2
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
MaxPooling	Window:1 × 2, s:2
Convolution	#maps:256, k:3 × 3, s:1, p:1
Convolution	#maps:256, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:128, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:64, k:3 × 3, s:1, p:1
Input	W × 32 gray-scale image

Figure 3 CRNN Structure (Shi, 2015)

Data-driven experiments

Since the training process is time-consuming and due to time constraint, we managed to finish three experiments. Firstly, we used white background, standard texts and randomly chosen 6 characters per image. We generated 30000 images for training while 10000 for testing. We ran 4000 epochs. The result quickly converged at around 900 epochs. As a result, the model could identify some of the characters from a nicely generated receipt; most of the characters are replaced by empty spaces. In addition, nearly no character on a real receipt could be identified.

Then, we tried to formalize texts of different types and different lengths. We made use of 7 different monospace fonts. English words and numbers were generated separately. 240000 training images were generated while 80000 images were generated for testing. The results were much better, however, we realized that some of the words were misclassified due to the noises. Because of the imperfect line segmentation, parts of the characters from the previous line or the next line might be cropped into the image. In addition, sometimes the receipt could be distorted and the characters may not be perfectly standard.

Figure 4 Example Images

Finally, we added noises, distortion and blur effects into the image data. We still generated 240000 images for training while 80000 for testing. We ran 10000 epochs. The cost (loss) verses epochs graph is shown in Figure 5.

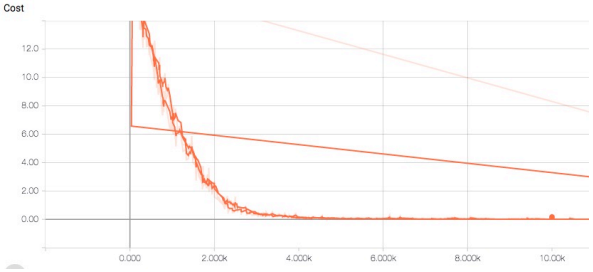

Figure 5 Cost to Epochs Graph

The Comparison with CNN

Since the training targets were relaxed to individual characters, the training process seemed to be simplified and required less time. We were focusing on 52 (Lower and Upper cases) English characters, 10 digits and 9 special characters. For each character, we generated 180 images with distortion, noises and blurred effect for training and 60 for testing. The cost converged at around the 200th epochs. Actually, CNN worked well when the receipt could be nicely broken down. But when lines were close to each other, the result would not be satisfying.

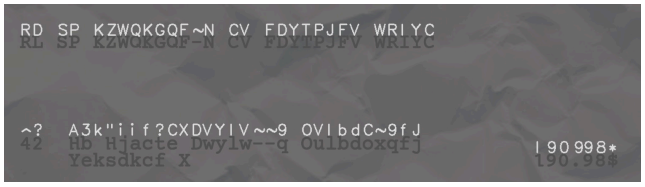

Figure 6 CNN Result Sample

However, the model processed one letter at a time, and each took around 1 to 3 seconds. The outcome that disappointed us most was processing time.

The Training Results

We were glad to achieve many satisfying results while the trained model could still be improved.

High accuracy on small image with machine printed words

We tried to crop small images nicely from a real receipt image. Then we used our trained model to recognize. The result was good; out of 10 images, 8 are recognized correctly. One of the examples is shown in Figure 7.

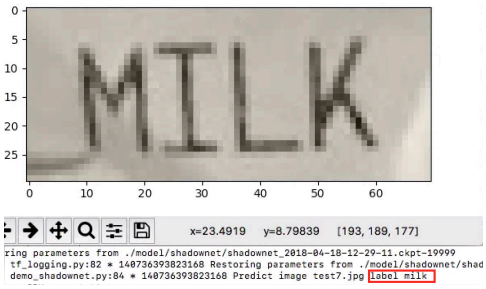

Figure 7 CRNN Result Example

Better image processing is required for receipt recognition and model need modification

In fact, if the receipt is nicely broken down, the result would be good, as shown in Figure 8.

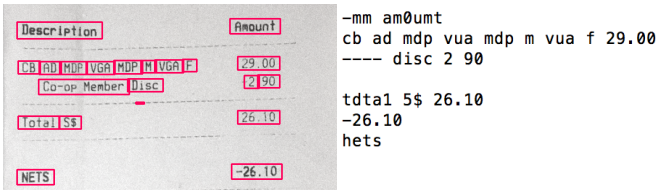

Figure 8 Sample Input and Output for CRNN

As we could observe, short texts are generally well-recognized. However, when the receipt is badly distorted, or when the texts are faded, the result would be bad. In addition, long texts usually lead to problems and this is due to the fixed length of training images. To deal with misclassifications for long texts, one way is to cut them down and recognize long texts separately. This may cause a problem of cutting a character in the middle. Hence a better way might be modifying the model to deal with images with different length.

Conclusions and Team Reflections

The whole process of training data and doing modification was tedious but the results were rewarding. And we realized that even for recognizing single characters, the model might be surprisingly complex. We will have a short discussion on the reflections below.

Real and good datasets are hard to find

Most of the time, datasets need data and corresponding labels. Data can to be mined, for instance, the receipt images can be easily download from websites, but the corresponding text is hard to be generated. Therefore, most of the time people choose to use programs to generate data but one important fact is that when generating data, various features of the data should be considered. Otherwise, overfitting could be easily encountered. From our opinion, at the beginning state, models can be trained with fake data. However, the model should be improved along with the real input from users of an application. For instance, our application should allow users to modify misrecognized characters and use the data to train the model so that it could be gradually improved and approach the real target function.

CNN is good, but use if efficiently

Convolutional Neural Network was a breakthrough for dealing with images efficiently. However, without good hardware support, the training and even classification require time. Hence, we should try to use CNN efficiently. For instance, try methods that recognize words instead of individual character.

Language models and error correction should be helpful

The model cannot be perfect, there should always be misclassifications. Hence, it would be important to find ways that could correct those errors. One way is for users to manually do correction, but natural language processing might be very helpful. One simple example would be with the database of all possible item names, use the edit distance to find possible corrections of a certain output from CRNN.

Team Contributions

- Cai Ting & Zou Yutong: Image processing & Line segmentation (Math students and are good at dealing with mathematical concepts)
- Wu Zefeng: Line segmentation & dataset generation (CS student, able to code and have done data mining project before)
- Chen Ke & Yang Zhuohan: Neural Network model constructions & codifications (One from statistics and one from CS. Collaborate to construct ML model)

References

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

Olah, C. (2014, July 8). Conv Nets: A Modular Perspective. Retrieved from https://colah.github.io/posts/2014-07-Conv-Nets-Modular/