Optical Character Recognition using Convolution Neural Networks
MIT 6.819 Final Project Fall 2017

Ekin Karasan
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
ekarasan@mit.edu

Dimitris Koutentakis
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
dkout@mit.edu

Abstract

Optical Character Recognition is a very important task for automatic data entry, making scanned text researchable, and ease of access. In this report we analyze the implementation of a simple Optical Character Recognition (OCR) algorithm using Convolutional Neural Networks (CNNs). CNNs are a powerful tool for learning features in various different types of images and have been used in tasks such as image recognition, object detection and scene recognition. We evaluate the performance of our implementations and discuss modifications and improvements that help the accuracy of OCR.

1. Introduction

As the use of computers and automated data analysis increases, there is an increasing need to convert images of text into machine-encoded text. More and more people want to keep a digital archive of their documents, convert old documents to searchable text, and perform data analysis on documents. For these reasons, it is very useful to be able to convert printed text into digital computer documents.

Optical Character Recognition (OCR) is a process by which handwritten or printed text is converted to a format which is understandable by machines. In this paper, we implement an OCR algorithm that heavily relies on Convolutional Neural Networks. OCR algorithms can be extremely complex, especially to achieve high accuracy with handwritten texts. We implement a simplified OCR algorithm aimed to achieve high accuracy in converting printed text that satisfies certain assumptions.

2. Related Work

Various architectures for OCR have previously been developed and are used widely. One example is the open-source engine developed by Google called Tesseract [6]. As in most OCR architectures, Tesseract starts by preprocessing the image firstly by doing a connected component analysis, in which outlines of the components are determined. These outlines are gathered together to form objects called *Blobs* and broken into different words. This method of preprocessing is very accurate, however, also very computationally expensive. Once the text is broken down into different words, the words are recognized using a two-pass adaptive recognition process. In the first pass, the algorithm attempts to recognize each of the words. Each word that is classified correctly is then added to the training set to improve the accuracy. In the second pass, the words that are not classified well are re-classified. Cuneiform [5] is another one of the earlier OCR architectures developed which also incorporates adaptive character recognition, similar to Tesseract. Many newer OCR software are able to achieve much higher accuracy compared to Cuneiform, however, have significantly more complex architectures. Some of these software include OCRopus [2], Abbyy [3] and Ocrad [4].

In the past 5 years, the success of CNNs in object recognition have been demonstrated many times. One example of this demonstration is AlexNet [1]. Using their deep convolutional neural network, they were able to win the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge), a very highly-reputed computer vision challenge. AlexNet not only utilized new techniques such as dropout, local response normalization and data augmentation, but also led to a focus in the field that established CNNs to be the leading class of models for image recognition. Figure 1 shows an overview of AlexNet's architecture.

We will explore using a modified version of the Alexnet architecture for character recognition, combined with our algorithm for preprocessing which will be described in the following section.

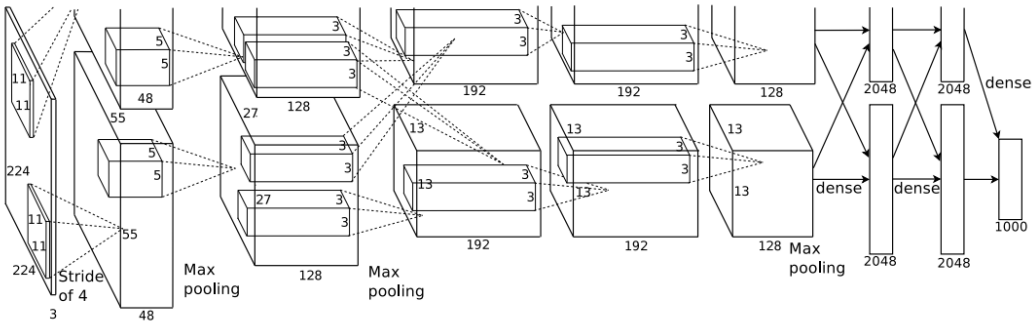

Figure 1. AlexNet architecture. [1]

3. Approach

We will start this section by giving an overview of the problem we wanted to solve, including some of the simplifications and assumptions we made. Afterwards, we will give a detailed explanation of the two main modules in our system: the preprocessing module and the character recognition module.

3.1. Problem Space and Assumptions

In this paper, we discuss the implementation of OCR for a small subset of images. Here are a list of assumptions we used while defining our problem space:

- Images are of computer generated printed documents.
- Lines in document are horizontal (straight).
- Characters in the same line are of the same size.
- Possible characters include all letters, both uppercase and lowercase, and numbers.
- Special characters are not included except for period.

3.2. Preprocessing

The input of the preprocessing module was an image of the text and the outputs were images of each of the letters extracted from the text.

The first step of preprocessing involved converting the input image into black and white. Once this was accomplished, horizontal lines that only contained white pixels were identified to separate the text into lines. Each line was processed in a similar way, identifying vertical columns with only white pixels, to separate each line into characters. The coordinates of each character in that line were stored.

Once the coordinates for every character are determined, spaces between words are identified by comparing the space between every character to the maximum character width in that line. If this spacing exceeds a certain threshold, a space between two words is detected. Periods are also detected manually, by looking at the ratio of black pixels in a character and the dimensions of the character.

After spaces and periods are determined, a square image of the other characters (excluding periods) are created by padding the image with white pixels. This square image is then passed onto the character recognition module.

3.3. Character Recognition

The character recognition module takes the square image of characters as input and determines the characters corresponding to the images. The recognition was done on a character-by-character basis rather than a word-by-word basis, like the one in Tesseract, because a word-by-word recognition would require a very large training dataset and many hours of training. In order to do that we trained a slightly modified version of the network we used in the Miniplaces challenge (a variation of the AlexNet), adapting the network to the OCR task.

The dataset used consisted of images 62 characters (26 uppercase, 26 lowercase and 10 digits) in 1016 different fonts. 85% of this dataset was used for training and 15% of this dataset was used for validation. This dataset was later reduced to 816 different fonts for each of the characters by eliminating uncommon fonts or fonts that have the same character for lower and upper case letters.

Our Miniplaces network was trained with the mentioned training and validation sets. The resulting weights of the network were used to classify the images of each of the letters. The results of our experiments with the network and datasets are discussed in Section 4.

3.4. Text Reconstruction

Finally, the algorithm takes the sequence of letters, periods, lines and spaces specified by the preprocessing stage and generates text by either writing the letter recognized by the neural network, a period, a space or a new line. The final output document is generated and saved in a `.txt` format.

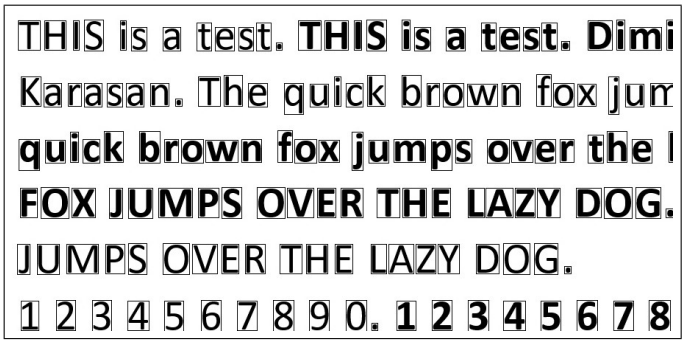

Figure 2. Preprocessing Algorithm Output

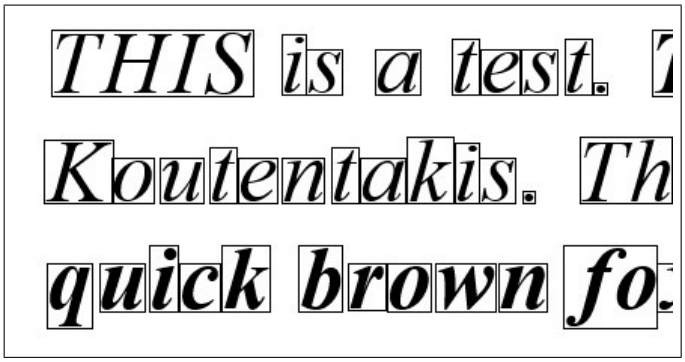

Figure 3. Preprocessing Algorithm failure on italicized serif font

Figure 4. Preprocessing Algorithm success on italicized font

4. Experiments & Results

4.1. Preprocessing

The pre-processing algorithm worked very well in almost all types of scenarios. As seen in figure 2, the preprocessing part of our algorithm correctly identifies all the characters in the image (denoted by the box around them).

However as seen in figure 3, in certain situations, such as fonts with large serifs, when italicized, the preprocessing algorithm merged letters close together and thought entire words were letters. This underlines some of the limitations of our preprocessing algorithm and is the main reason we assume that all inputted text is non-italicized even though the classifier can correctly classify italicized letters.

It seems like the preprocessing algorithm only has a problem when the italicized fonts are serif fonts as in other cases it works correctly (such as in figure 4).

4.2. Character Recognition

We tested the accuracy of our system with 4 different variations of the neural network and datasets. These four variations can be summarized as follows:

1. The first iteration of our network was trained on the full dataset with image flips and very little data augmentation.

2. The second iteration of our network was also trained on the full dataset with the removal of image flips, as letters always have the same orientation.

3. For the third iteration, we increased the data augmentation and removed uncommon fonts from the dataset in order to train more closely to more commonly used fonts.

4. In the fourth iteration we increased the crops and rotations of the image augmentation while reducing blur and shearing. We also decreased the learning rate and increased the percentage of the data that is included in the training set.

Our results can be viewed in tables 1 through 4.

First CNN Accuracy		
Font	Case-Sensitive	Case-Insensitive
Calibri	88.93%	96.54%
Times New Roman	86.85%	91.70%
Candara	77.85%	86.51%
Overall	**83.27%**	**90.00%**

Table 1. Results for accuracy of first CNN.

Second CNN Accuracy		
Font	Case-Sensitive	Case-Insensitive
Calibri	91.70%	98.62%
Times New Roman	82.35%	87.54%
Candara	78.55%	86.85%
Overall	**84.20%**	**91.00%**

Table 2. Results for accuracy of second CNN.

4.3. Discussion

As we can see from the results included above, each modification we made resulted in a more robust and better-performing algorithm. It is also important to note that the output of the character recognition algorithm depends quite

Third CNN Accuracy		
Font	Case-Sensitive	Case-Insensitive
Calibri	94.81%	98.26%
Times New Roman	80.62%	86.85%
Candara	79.58%	87.20%
Overall	**85.01%**	**90.77%**

Table 3. Results for accuracy of third CNN.

Fourth CNN Accuracy		
Font	Case-Sensitive	Case-Insensitive
Calibri	95.5%	98.96%
Times New Roman	83.74%	85.58%
Candara	80.62%	87.19%
Overall	**86.62%**	**91.58%**

Table 4. Results for accuracy of fourth CNN.

a bit on the font used. For example, fonts that have lower and uppercase letters that are very similar and/or fonts that are different than most common fonts (on which our CNN was trained) will have lower accuracies. In addition, letters such as uppercase "i" and lowercase "L" were easily confused because in sans-serif fonts they look very similar: I, l. Other examples of characters that got confused were capital "o" and the number zero as well as lowercase "I" and the number one. One solution to this problem could be to decide on the letter after looking at its size comparative to other characters nearby (larger characters are likely to be capital letters or numbers than smaller characters).

Finally it is interesting to note that even though our algorithm worked with most fonts (even italicized), the preprocessing algorithm failed as it could not correctly separate characters. In order to solve this we could use a more computationally intense "blob detection" algorithm, similar to that of Tesseract, that looks at all the connected black pixels in order to separate the documents into letters.

In order to extend this algorithm even more, we could replace the horizontal line scanning with an algorithm that finds points in the first line and then performs least-mean-squares to fit an optimal line and reads the rest of the document parallel to that. Furthermore, given more training time and a large enough MNIST-like handwritten dataset for all characters, it is very possible to extend the character recognition to handwritten documents. Adding special characters in our dataset would also allow for recognition of characters other than letters or digits.

5. Individual Contribution

For this project, I focused on the character recognition. In specific, I worked on the training of the Neural Network and the reconstruction of the document. After trying several variations of our MiniPlaces CNN, which is based on the Alexnet architecture, I reached the conclusion that what works best for this project is a relatively low learning rate and image augmentation that mostly depends on scaling and rotations, some cropping, very little blurring and no flips (since the letters will always face the correct way). I also found that reducing the dataset a bit in order to not include very uncommon fonts, fonts that have greek characters, or fonts that have uppercase characters for lowercase letters (and vice versa) also helped.

Finally in order to reconstruct the document I used the output of the forward pass on the trained Convolutional Neural Netork for each image of a letter that was outputted from the pre-processing algorithm. This output was then written that into a `.txt` document, based on the sequence of letters, spaces, periods and lines provided from the pre-processing module.

6. Conclusion

In conclusion, we have demonstrated a robust algorithm for converting images of text into machine-readable and searchable documents, using a preprocessing and a character recognition stage. We are very satisfied with the overall results of our algorithm, as we can mostly correctly convert images of documents into text, sometimes exceeding accuracies of 98%, depending on the font and goal of the conversion.

Even though the results are satisfactory, there are potential solutions to the problems we have faced as well as steps to extend this algorithm to wider use cases, which are detailed in Section 4.3.

References

[1] I. S. A. Krizhevsky and G. E. Hinton. Imagenet classification with deep convolutional neural networks, 2012.

[2] T. Breuel. The ocropus open source ocr system. *Document Recognition and Retrieval XV*, 2008.

[3] http://finereader.abbyy.com/.

[4] http://www.gnu.org/software/ocrad/.

[5] F. Shafait. Document image analysis with ocropus. *IEEE 13th International Multitopic Conference*, 2009.

[6] R. Smith. An overview of the tesseract ocr engine, 2007.