# **Project Report**

|  |  |
| --- | --- |
| Yuchen Tong  1003534669  Yuchen.tong@mail.utoronto.ca | Junming Zhang  1003988982  Junmingpeter.zhang@mail.utoronto.ca |

**Abstract**

*We introduce an OCR pipeline which takes input as a document type image scanned or captured by cameras and outputs the text in the picture as text format file. We first discuss our pipeline in detail and analyze possible pros and cons of our design in variance conditions. Second, we provide some sample outputs from our pipeline and some related benchmarks for performance measuring. Third, we explain some success and failure cases we encountered during our implementation and how we make decisions and how they affect our pipeline. Finally, we show some our experiments to support some of our thoughts and concerns towards this pipeline by comparing our pipeline with existing official OCR tools – tesseract.*

# Introduction

Optical Character Recognition (OCR) is the electronic or mechanical conversion of images of typed, handwritten, or printed text into machine-encoded text from any source [1].

What we try to solve here can be considered as the sub-problem of OCR, where we mainly focus on recognizing printed text from any document type image captured or scanned by cameras. Traditional text detection methods tend to involve multiple processing step, e.g. character/word candidate generation [2]. To avoid some complexities, we assume the input images are well-conditioned, such as, directly captured picture of a page of a book, digital screen, and relatively flattened documents. There are two main approaches for text recognition:

* *Character-based*: Individual characters are first detected and then grouped into words [2]. Such as the model proposed by (Neumann and Matas 2012; Pan, Hou, and Liu 2011; Yao et al. 2012; Huang, Qiao, and Tang 2014) [3]
* *Word-based*: Words are directly hit with the similar manner of general object detection [2].
* *Text-line-based:* Text lines are detected and then broken into words. [2]

## Intuition and Main Approach

Our intuition is that since there exist only ten digits and twenty-six English characters. It is easier to train a CNN with character image patches instead of word or sentences due to the label size, because we do not need to consider all the combinations of characters(words). Since the characters are basically the smallest elements to form a document, once we can segment them properly and train a model that can successfully detect single character patches properly, the result might be decent under our assumption stated in the previous part.

To build a *Character-based* text recognition pipeline, our main approach is that:

1. Pre-process the raw image to preserve a well-conditioned document image.
2. Extract feature patches (character image patch) from pre-processed image.
3. Train a CNN model that can successfully classify digits and alphabets.
4. Combine all the classified character patches and output the text file.

## pros

* Reduce the complexity of model training. Because the feature patch size is significantly smaller than considering words and sentences directly.
* Consider only printed document can potentially reduce the variance of the input. Because considering scene text reading is much more challenging, due to the large variations in both foreground text and background objects, as well as uncontrollable lighting conditions, etc. [2].
* Consider only well-conditioned picture makes the pre-process stage simpler. Since, in the worst-case scenario, it is very challenging to cover all the edge case in general. In order to achieve good results in general, it might be very reasonable considering applying some complex networks to solve this problem, such as *DewarpNet* by (Sagnik and Ke et al. 2019) [4]

## cons

* Lose the valuable relational information between characters if we break words into characters and consider them separately.
* Considering only printed document leads to train model with different font. Early versions needed to be trained with images of each character and worked on one font at a time. Advanced systems capable of producing a high degree of recognition accuracy for most fonts are now common, and with support for a variety of digital image file format inputs [5].
* Once the condition is not good enough, the pipeline will be extremely sensitive to noise.

# Methodology

* Pre-process the raw image. (I use Matt Zucker’s approach from his online article.)

1. Split the text into lines [6].
2. Find a coordinate transformation that makes the lines parallel and horizontal [6].

Consider the page that we take picture from is a 3D transformed plane with a rotation vector r and a transformation matrix T. Since the page is always impossible to be perfectly flat, it might be bumpy and curl. So, the model Matt describes is basically model each line of text (horizontal cut lines) using cubic polynomial with x, y indicates the pixels’ horizontal and vertical displacement. There is an extra z value which indicates the spatial displacement (the bump height of the page)

Assume that the left edge of the document page is at (0, y) and the right edge is at (1, y), we get the new cubic polynomial:

Let:

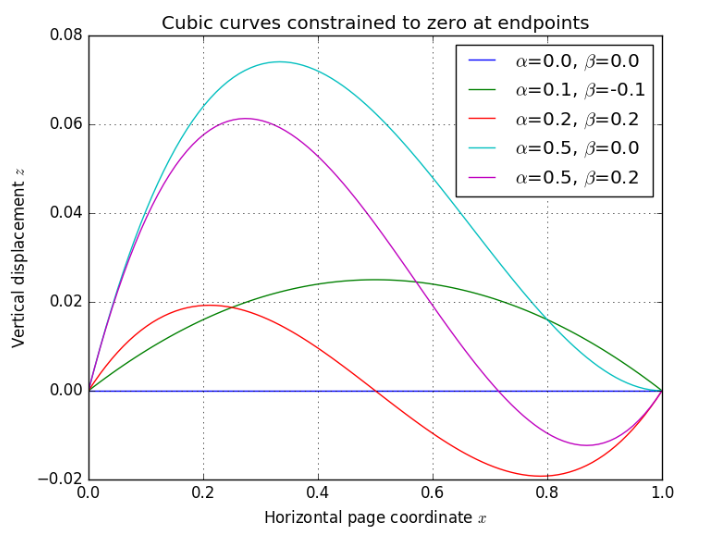
So, given value of α and β, the cubic polynomial is uniquely determined:

Figure 1: From Matt’s article [6]

Combining transformation matrix T and rotation vector r, we can get the 3D special span of the page:

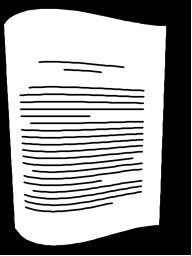
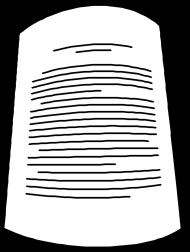
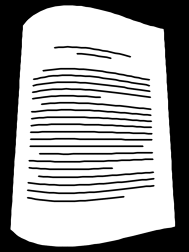
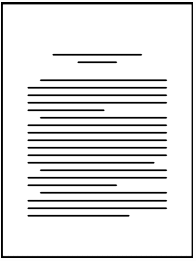


Figure 2: From Matt’s article [6]

Here is the pipeline about how Matt’s algorithm processes the input image and apply the model above [6]:

* 1. Obtain the optimized page boundary where all the text data is contained [6].
  2. Detect the contour of the contour of all the text and applying morphological dilation to thicker, fill and connect text horizontally. Then apply morphological erosion to thinner white areas to remove single white noise [6].
  3. Using connected component analysis to connect and threshold all the text like lines and apply PCA to approximate the best fitting line [6].
  4. Combine all of the contours corresponding to a single horizontal span on the page [6].
  5. Choosing some discrete samples from all the spans to form a sample span line. Each one of the key points is selected among about 20 samples [6].
  6. Use PCA again to estimate the mean orientation of all spans. The mean orientation which selected will be used to establish initial entry for the optimization step [6].
  7. Solving the optimal T, t, α, β…parameters by minimizing the fit error (squared distances between fitting points location and its original location) [6].
  8. Remap the text information that fitted with optimal cubic span approximation to a blank background with the optimal dimension. Applying adaptive threshold to the image to get the final output [6].
* Extract feature patches. (Our own implementation)
  1. Invert the color of the binary image
  2. Apply morphological dilation with 2 by 2 box step by step to gradually thicker the characters and make them connect as a word. The steps keep going until it hits the steepest descent and hits a threshold (Once all the characters connect as a word).
  3. Find word contours and corresponding bounding boxes for each contour. Draw all the bounding boxes white to make a word contour mask to avoid intersected contours. Find the contours of the contour mask to obtain a new list of bounding boxes.
  4. Threshold the boxes by the information area (how many intensity values in the area of the patch) to get a list of rectangle represents the word patches.
  5. Sort all the rectangle as lines based first on the vertical position and second the horizontal position. Rectangles consider in the same line if the difference their vertical position is within a certain threshold (we use 20 pixels).
  6. For each word patch we find character contours and the bounding box using the same pipeline as finding word contours.
  7. For each rectangle which represents the character patches, cut out the real image patch from the page image. Resize the patch using fixed ratio resize and put it at the center of of a background patch with dimension (28, 28). The reason we use 28 by 28 is that our CNN is trained with EMNIST handwritten digit dataset. The dataset using 28 by 28 as the standard.
* Convolutional neural network (CNN, learn to generate CNN model from [7], and the idea of CNN is inspired by [2])

1. Load data from the byclass dataset (to obtain the dataset, please visit [9])

Byclass dataset (including training, testing and map set), the byclass dataset is chosen because it has abundance of training data with 10 digits (0~9), 26 uppercase letters (A~Z) and 26 lowercase letters (a~z), based on [10], and a table from [10] is attached below:

A screenshot of a cell phone screen with text

Description automatically generated

After reading train, test and map from the byclass dataset, there is a couple of conversions on the dataset, including splitting data and labels, transferring the labels to one-hot encoding (categorial data), reshape the training dataset to a CNN feedable dataset and split validation data and labels

1. Build the CNN model

4 choices have been taken to build such a model

1. A neural network model from skitlearn. The result is not ideal since the accuracy can only achieve 0.49 after tuning the hyperparameters to optimize the performance
2. A CNN with 2 convolutional layers, 2 nonmax-suppression (maxpooling) and 2 dense layers (dot product) (this is originally based on the work of [7]), a summary of model is below:

A screenshot of a cell phone

Description automatically generated

And charts about accuracy and loss for each epoch is attached for this model below:

A screenshot of a cell phone

Description automatically generatedA picture containing screenshot

Description automatically generated

And the accuracy on the test set achieves about 0.86.

3. A CNN model with 6 convolutional layers, 6 max-suppression (maxpooling) and 2 dense layers (dot product) (this model is designed based on the structure on [2]). The model has filter sizes 512 -> 1024 -> 1024 -> 512 -> 256 ->256 ->256.

Although this design achieves high accuracy on the first epoch, but still not used because of high training time (the device does not support CUDA).

4. An alternative version of choice 3 (still based on [2] but downscale the filter sizes). A summary of the model is below:

A screenshot of a cell phone

Description automatically generated

And charts about accuracy and loss for each epoch is attached for this model below:

A close up of a map

Description automatically generatedA picture containing screenshot

Description automatically generated

This model achieves 0.87 accuracy on the test data. Therefore, the model improves the behavior, but not a lot. The reason might be the filter size is still too small and many details from the paper are neglected or simplified due to problem on training.

c. Train the model by the training dataset, and also do validation by the validation dataset. And also save the model. The way to save the model is adapted from [9]. (The model is in .json format and the weights are saved in .h5 format)

d. An interface to load the model is provided, the way to load the model is adapted from [9].

## Results

…

## Benchmarks

…

# Experiments

…

# Conclusion

…

# Authors’ Contributions

…

# References

1. OnDemand, HPE Haven. “OCR Document”. Archived from the original, 2016.
2. [Minghui, Baoguang et al 2016] Minghui, L., and Baoguang, S. 2016. TextBoxes: A Fast Text Detector with a Single Deep Neural Network.
3. [Neumann and Matas 2012] Neumann, L., and Matas, J. 2012. Real-time scene text localization and recognition. In Proc. CVPR, 3538–3545.
4. [Sagnik, Ke, Zhixin et al 2019] Sagnik D., and Ke, M. 2019. DewarpNet: Single-Image Document Unwarping with Stacked 3D and 2D Regression Networks.
5. OnDemand, HPE Haven. “undefined”. Archived from the original, 2016
6. Zucker, Matt. (2016, August 15), Page dewarping, <https://mzucker.github.io/2016/08/15/page-dewarping.html>
7. Ashwani Jindal (2019), EMNIST using Keras CNN, <https://www.kaggle.com/ashwani07/emnist-using-keras-cnn>
8. Chris Crawford (2017), EMNIST (Extended MNIST) An extended variant of the full NIST dataset, <https://www.kaggle.com/crawford/emnist>

# [9] Jason Brownlee (2019), How to Save and Load Your Keras Deep Learning Model, <https://machinelearningmastery.com/save-load-keras-deep-learning-models/>

[10] [Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre ́ van Schaik at 2017], Gregory C., Saeed A., Jonathan T., and Andre S. 2017. EMNIST: an extension of MNIST to handwritten letters