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.

1. 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]

1.1. 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:

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

1.2. 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]

1.3. 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.

2. Methodology

- Pre-process the raw image. (I use Matt Zucker's approach from his online article.) The reason we choose this algorithm instead of other more complicated Dewarping pipeline such as DewarpNet [4] is that we have added some constraints about our inputs. Since we assume that the input image and the documents are well-conditoned. Under our assumption, a more procedural type of approach such as Zucker's cubic polynomial modeling is enough to satisfy our needs.

  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)

$$f(x) = ax^3 + bx^2 + cx + d$$

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:

$$f(x) = ax^3 + bx^2 - (a+b)x$$
$$f'(x) = 3ax^2 + 2bx - (a+b)$$

Let:

$$\alpha = f'(0) = -a - b$$
$$\beta = f'(1) = 2a + b$$

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

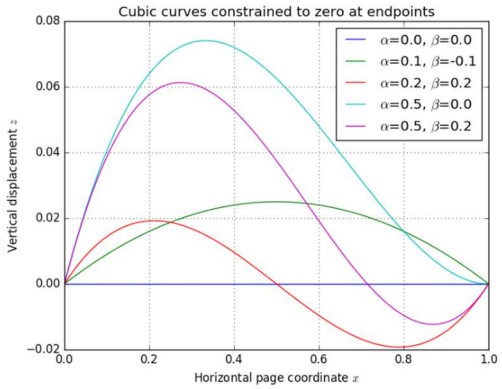


Figure 1: From Matt's article [6]

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

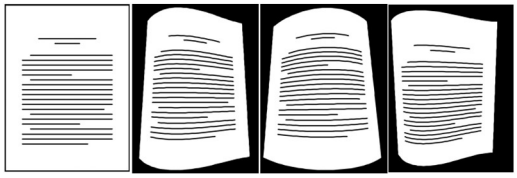


Figure 2: From Matt's article [6]

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

a. Obtain the optimized page boundary where all the text data is contained [6].
b. 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].
c. Using connected component analysis to connect and threshold all the text like lines and apply PCA to approximate the best fitting line [6].
d. Combine all of the contours corresponding to a single horizontal span on the page [6].
e. 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].

f. 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].
g. Solving the optimal T, t, α, β…parameters by minimizing the fit error (squared distances between fitting points location and its original location) [6].
h. 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)
    a. Invert the color of the binary image
    b. 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).
    c. 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.
    d. 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.
    e. 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).
    f. For each word patch we find character contours and the bounding box using the same pipeline as finding word contours.
    g. 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])
    a. 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:

TABLE I
BREAKDOWN OF THE NUMBER OF AVAILABLE TRAINING AND TESTING SAMPLES IN THE NIST SPECIAL DATABASE 19 USING THE ORIGINAL TRAINING AND TESTING SPLITS.

	Type	No. Classes	Training	Testing	Total
By Class	Digits	10	344,307	58,646	402,953
	Uppercase	26	208,363	11,941	220,304
	Lowercase	26	178,998	12,000	190,998
	Total	**62**	**731,668**	**82,587**	**814,255**
By Merge	Digits	10	344,307	58,646	402,953
	Letters	37	387,361	23,941	411,302
	Total	**47**	**731,668**	**82,587**	**814,255**
MNIST [1]	Digits	10	60,000	10,000	70,000

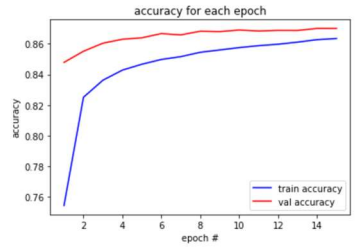
After reading train, test and map from the byclass dataset, there is a couple of conversions on the dataset, including splitting data and labels, transferring the labels to one-hot encoding (categorial data), reshape the training dataset to a CNN feedable dataset and split validation data and labels

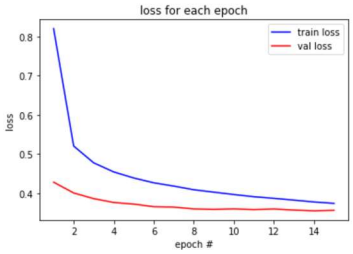
b. Build the CNN model
4 choices have been taken to build such a model

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:

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 28, 28, 128)	3328
max_pooling2d_2 (MaxPooling2	(None, 14, 14, 128)	0
conv2d_3 (Conv2D)	(None, 14, 14, 64)	73792
max_pooling2d_3 (MaxPooling2	(None, 7, 7, 64)	0
flatten_1 (Flatten)	(None, 3136)	0
dense_2 (Dense)	(None, 128)	401536
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 62)	7998

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





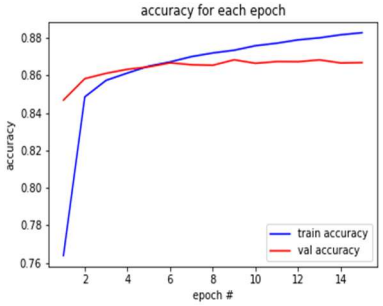
And the accuracy on the test set achieves about 0.86.

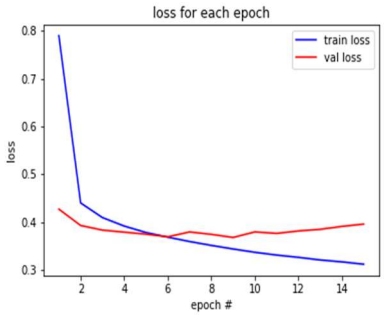
3. A CNN model with 6 convolutional layers, 6 max-suppression (maxpooling) and 2 dense layers (dot product) (this model is designed based on the structure on [2]). The model has filter sizes 512 -> 1024 -> 1024 -> 512 -> 256 ->256 ->256. Although this design achieves high accuracy on the first epoch,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

Layer (type)	Output Shape	Param #
conv2d_24 (Conv2D)	(None, 28, 28, 64)	1664
max_pooling2d_21 (MaxPooling	(None, 14, 14, 64)	0
conv2d_25 (Conv2D)	(None, 14, 14, 128)	204928
max_pooling2d_22 (MaxPooling	(None, 7, 7, 128)	0
conv2d_26 (Conv2D)	(None, 7, 7, 128)	409728
max_pooling2d_23 (MaxPooling	(None, 3, 3, 128)	0
conv2d_27 (Conv2D)	(None, 3, 3, 64)	204864
max_pooling2d_24 (MaxPooling	(None, 1, 1, 64)	0
conv2d_28 (Conv2D)	(None, 1, 1, 32)	18464
conv2d_29 (Conv2D)	(None, 1, 1, 32)	9248
flatten_2 (Flatten)	(None, 32)	0
dense_4 (Dense)	(None, 128)	4224
dropout_2 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 62)	7998

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





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].

2.1. Results

Sample outputs from preprocess (image dewarping):

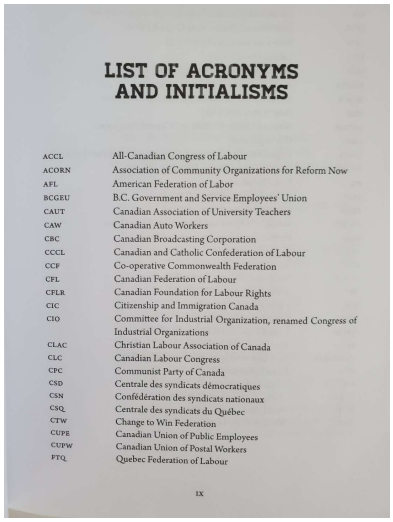


Figure 3: A well-conditioned image captured from a book [7]

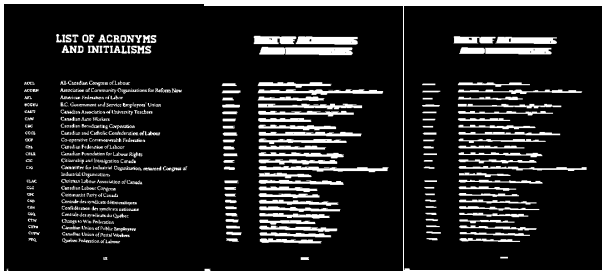


Figure 4: Text Contour, Dilation to highlight lines, Erosion to avoid over exaggeration

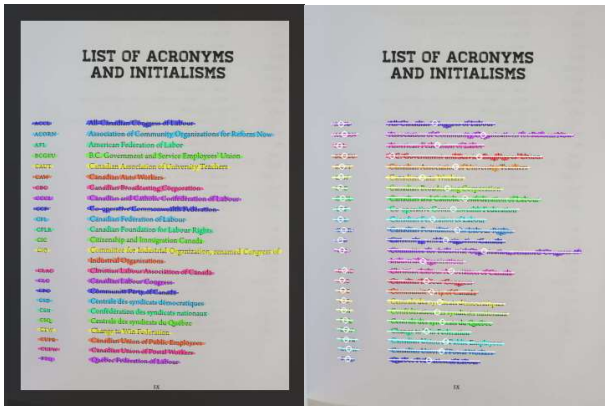


Figure 5: Text line like area, and approximated lines after applying connected component analysis and PCA

Figure 6: Sample keypoints from span, and the location difference after optimization (Red dot represents the original location, Blue dots represents the new location modeled using the cubic polynomial)

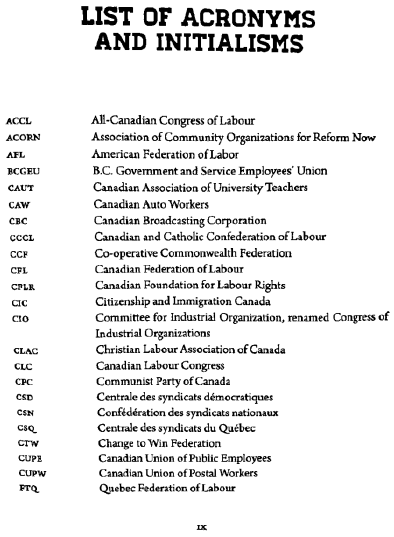


Figure 7: The binary remapped (cubic polynomial fitting) image from original image.

Sample outputs from Feature patch extraction:

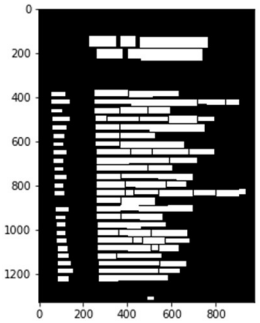


Figure 8: Binary word contour mask

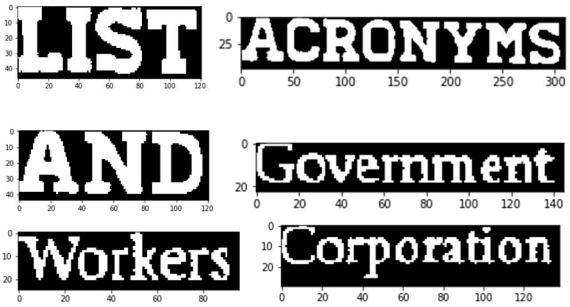


Figure 9: some sample of word patch detected using word patch mask.

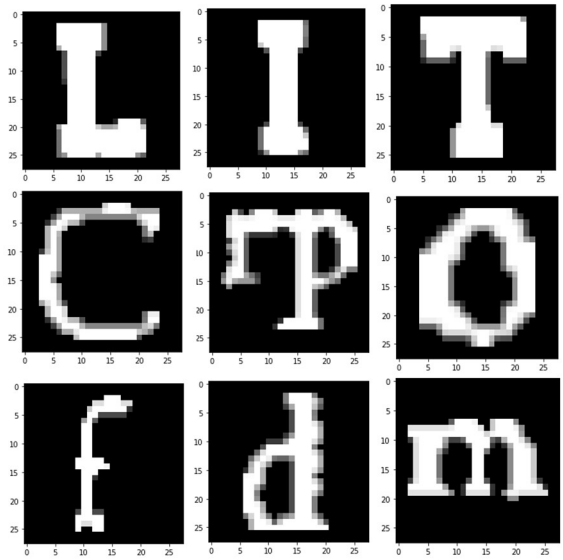


Figure 10: some sample of 28 by 28 standardized character patches detected using character patch mask

We originally think that we can avoid training the model with different fonts by training the handwritten dataset directly. Since most printed fonts are more formal, standardized and has lower variance than handwritten characters. That is why we think that if the model can predict handwritten characters well, it will be automatically easy for the model to identify printed characters. It turns out that it is not the case, we have trained several models on EMINIST dataset. No matter how high the training and testing accuracy is, when we pass our dataset to the model, it performs poorly just as shown in the example.

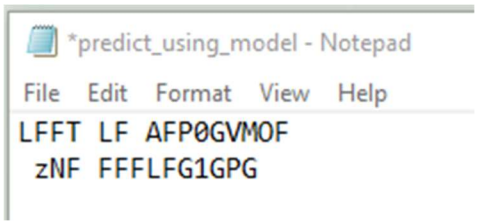


Figure 11: Sample output of the text region predicted using CNN trained with EMNIST dataset (handwritten characters and digits)

Although the prediction for the word “LIST” is “LFFT”, “OF” is “LF”, and “AND” is “zNF”, which are the only identifiable successful signs.

Look at the feature extraction stage and model training stage separately. Both of the preprocess and CNN training stages perform decent individually, but once we feed the data extracted from real world image to the model, it performs quite poor. So here is the reason we think that cause this situation:

- The image we mainly focus on is the document type image with printed characters. However, the dataset we use to train the model is EMNIST handwritten dataset, which is inconsistent.

  So, we go back and assembled a word patch using characters from EMNIST dataset and use it as a document to feed back to the start of our pipeline. Therefore, the characters are extracted by the feature extraction stage to form a 28 by 28 feature patch. This time since the characters are from the same dataset which we used to train the model. The result turns out to be quite reasonable. The model successfully identified most of the characters right. The outputs are shown in figure 18 and 19. This process we just demonstrated proofs that the connection between our two stage is quite reasonable. One main approach which can increase the accuracy is that we need to make the dataset persistent. That is,
  a. We build our own dataset by scanning enough document in real life using our pre-processing pipeline and feed the CNN to train it.
  b. We scan and preprocess other datasets but in font by font order to train our CNN.

  Both options require to build a huge dataset using our preprocess pipeline, which exceeds our ability to further improve the model. That is the main limitation of our design.

Figure 18: The random word patch generated from EMNIST dataset.

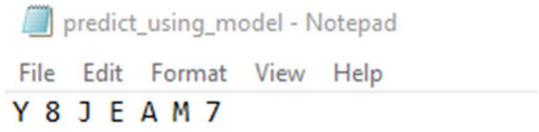


Figure 19: The output image from reading the document type image generated from EMNIST

2.2. Benchmarks

Stage	Complexity	Time (varies in picture dimension)
Dewarping	O(n^2), n parameters	average 5.3 seconds among test images
Character patch extraction	O(n), n characters	average 15 seconds among test image

3. Experiments

3.1. Some experiments relate to the page dewarping model from Matt Zucker [6]:

Since the model is using cubic polynomial fitting for the page's 3D shape along x axis (image's horizontal direction). The limitation is that it cannot fit the shape which is represented using polynomials that has higher orders than cubic (When the page has more than 3 bumps horizontally). So, we did the following experiments:

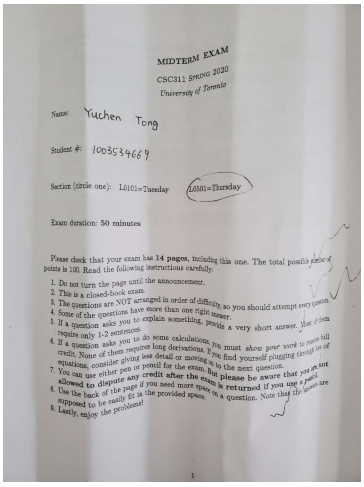


Figure 12: A image a very bumpy document

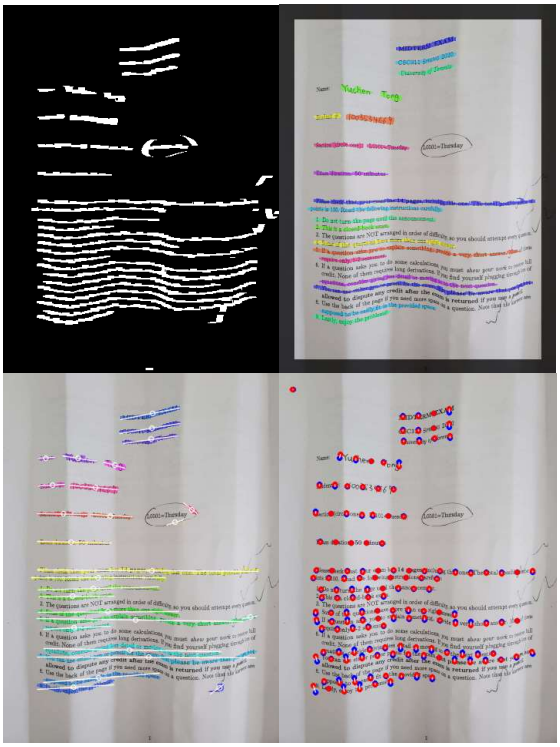


Figure 13: Connected component analysis and cubit polynomial fitting.

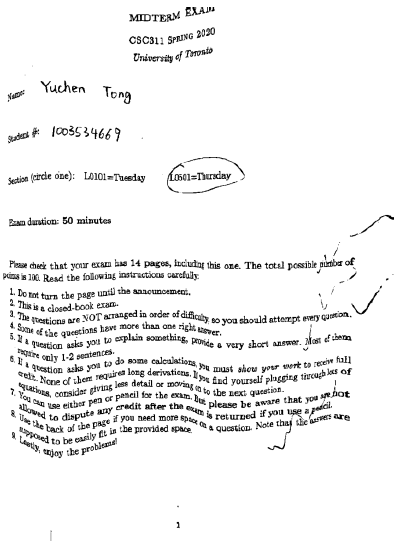
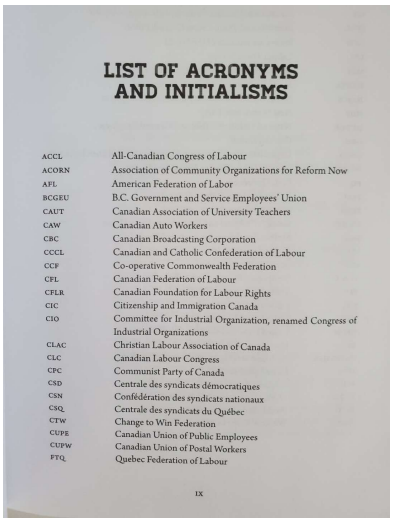

MIDTERM EXAM
CSC311 SPRING 2020
University of Toronto
Name: Yuchen Tong
Student #: 1003534669
Section (circle one): L0101=Tuesday L0601=Thursday
Exam duration: 50 minutes
Please check that your exam has 14 pages, including this one. The total possible number of points is 100. Read the following instructions carefully.
1. Do not turn the page until the announcement.
2. This is a closed-book exam.
3. The questions are NOT arranged in order of difficulty so you should attempt every question.
4. Some of the questions have more than one right answer.
5. If a question asks you to explain something, provide a very short answer. Most of them require only 1-2 sentences.
6. If a question asks you to do some calculations, you must show your work to receive full credit. None of them requires long derivations. If you find yourself plugging through lots of equations, consider giving less detail or moving on to the next question.
7. You can use either pen or pencil for the exam, but please be aware that you are not allowed to dispute any credit after the exam is returned if you use a pencil.
8. Use the back of the page if you need more space on a question. Note that the answers are supposed to be easily fit in the provided space.
9. Lastly, enjoy the problems!
1

Figure 14: The binary image after applying page dewarping pipeline.

Since the horizontal shape cannot using cubic polynomial to correctly fit. It fails to dewarp the testing image with bad condition. However, we are under the assumption that out input image will be well-conditioned, so the 3D space that Matt's model spans can satisfy most our cases. Consider a page from an opened book, the natural shape of the page can very likely be modeled using cubic polynomial.

3.2. Some experiments with tesseract OCR:

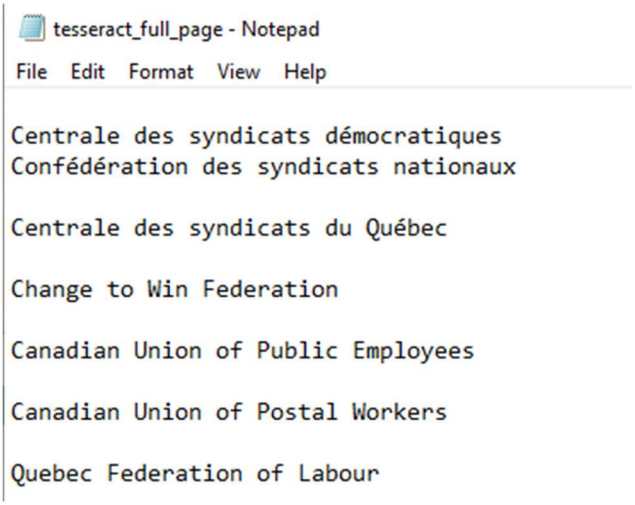
Firstly, since our *character-based* text recognition with CNN trained under EMNIST dataset does not perform as well as our anticipated. Secondly, training the model with our own labeled data is a bit too task-specific, it will be hard to generalize to real life example unless we keep collecting and labeling to build a huge dataset for training. So, we decided to do some experiments using existing OCR tool.


LIST OF ACRONYMS AND INITIALISMS

ACCL	All-Canadian Congress of Labour
ACORN	Association of Community Organizations for Reform Now
AFL	American Federation of Labor
BCGEU	B.C. Government and Service Employees' Union
CAUT	Canadian Association of University Teachers
CAW	Canadian Auto Workers
CBC	Canadian Broadcasting Corporation
CCCL	Canadian and Catholic Confederation of Labour
CCF	Co-operative Commonwealth Federation
CFL	Canadian Federation of Labour
CFLR	Canadian Foundation for Labour Rights
CIC	Citizenship and Immigration Canada
CIO	Committee for Industrial Organization, renamed Congress of Industrial Organizations
CLAC	Christian Labour Association of Canada
CLC	Canadian Labour Congress
CPC	Communist Party of Canada
CSD	Centrale des syndicats démocratiques
CSN	Confédération des syndicats nationaux
CSQ	Centrale des syndicats du Québec
CTW	Change to Win Federation
CUPE	Canadian Union of Public Employees
CUPW	Canadian Union of Postal Workers
FTQ	Quebec Federation of Labour

IX

Figure 15: Input image for our pipeline and tesseract.


tesseract_full_page - Notepad
File Edit Format View Help

```
Centrale des syndicats démocratiques
Confédération des syndicats nationaux

Centrale des syndicats du Québec

Change to Win Federation

Canadian Union of Public Employees

Canadian Union of Postal Workers

Quebec Federation of Labour
```

Figure 16: Some of the sample lines in the output file from tesseract.

The result is very accurate consider it even successfully "é" in "Québec".

Next, we process the input image and produce a list of character image patch for tesseract to recognize one by one and outputs the file. We set the tesseract flag to "psm 10" which represents single character recognition. The sample output is shown in figure 17.


tesseract_single_char - Notepad
File Edit Format View Help

```
R CC™LatiZen 5h l®ePPOam—-undd-at=1FONLmsfoF t=LEBLObnCOUMraRdsatsS

CO=mm | Mtt—ee—folindUMS tr ilmaOl=] BZatleOn sren~e dCO-Of
iPSS t ile LOadb oun imZsartim§sSoonas PaionofCcmada

ccC=n€onmatendmd dle tener d |MLSStaSmPrOaerusyrdlLECOcsfOeC=ds[momEsFsdOacratm) bees

es
D
```

Figure 17: the output from tesseract by recognize single character one by one.

The result is shockingly terrible. When we ask tesseract to identify each single character, it often identifies the character as empty string or some random characters or symbols. So, we did some research of this and found that tesseract is *text-line-based*. It trained with entire line of text instead of just single character.

4. Conclusion

Our pipeline involves image pre-processing, feature extraction, training and predicting using CNN. In general, the performance and results are not as good as what we anticipated. Mainly because the dataset we are using cannot be easily generalize to real life situation. Secondly, the design of our pipeline is over-simplified compare to those advanced OCR tools like tesseract. Thirdly, in order to further improve the accuracy of our pipeline, we have to put huge amount of work focusing on building and enlarging our own dataset, which is beyond our reach. After doing some researches online and experiments with existing OCR tools, we can conclude that our design is too naïve and it bottlenecks the performance. However, despite all the failures we experienced from implementing this project, we still did a lot of experimental study and learned a lot from our failures.

5. Authors' Contributions

Yuchen Tong:

- Dewarping stage (Code and intuition are from online sourse) [6]
- Character patch extraction stage (original idea)

Junming Zhang:

- Convolutional neural network (CNN, learn to generate CNN model from [7], trained by the data set from [8], save and load the model with the functions adapted from [9] and the idea of CNN is inspired by [2])
- Output text based on models' prediction.

6. 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