Khmer OCR fine tune engine for Unicode and legacy fonts using Tesseract 4.0 with Deep Neural Network

Kim Sokphyrum **Suos Samak**

Researchers at Open Institute, Phnom Penh, Cambodia

{phyrum, csamak}@open.org.kh

Supervised By: **Mr. Javier Sola**

Program Director at Open Institute, Phnom Penh, Cambodia

javier@open.org.kh

Abstract

The ISRI Analytic Tools were used to measure the accuracy of version 4.0 of the Tesseract Optical Character Recognition (OCR) engine and other five fine tune engines for Khmer Unicode and legacy Limon Khmer fonts. Version 4.0 of Tesseract integrates a deep neural network (DNN) approach as a text line recognizer. Character and cluster level accuracy were measured and evaluated by comparing the OCR generated files from all engines with the corrected or ground truth text files. The system were tested with 14 different Khmer Unicode fonts as well as 20 pre-unicode "Limon" legacy fonts (at size 12pt). The test was done with the text that included 11,272 clusters (19,618 characters). Khmer pre-trained engine of Tesseract 4.0 provided a **87.49%** average accuracy for Unicode fonts at the character level (CHL) and **89.43%** at the clusters level (CLL). But for Limon fonts, it produced **62.27%** median accuracy at the CHL and **60.08%** at the CLL. By default Tesseract 4.0 has been pre-trained with 16 Khmer Unicode fonts and it has been able to recognise fonts which were not trained on. However, the system is expected to perform much better in Limon fonts when trained to do so by implementing fine tune process for those new fonts. The (Fine tune) Engine for Limon S1 Unicode (by converting the Limon S1 into Unicode font) bring out a better accuracy result for Limon fonts with the accuracy 70.66% (CHL) and 69.55% (CLL). The accuracy for Engine for Limon S1F1 Unicode was 71.58% (CHL) and 70.50% (CLL). The Engine for Limon S1S2F1 Unicode remarkably generates higher accuracy result with 72.35% (CHL) and 71.35% (CLL).

Keywords: Tesseract 4.0, Deep Neural Network (DNN), Fine tune Khmer OCR engine, Khmer OCR, ISRI accuracy measurement tool.

I. Introduction

While OCR can be applied to any printed document, defining priority targets helps research and development aim first at specific results. In the case of Cambodia, the main need for OCR seems to be accessible to the law and legal documents.

The Cambodian government does not produce machine-readable versions of laws, regulations or any other legal document. All these documents are only made available in printed form. In the past they were printed in fonts such as Limon, and at this time they are mostly typed in Unicode, as required by the Council of Ministers.

While efforts are being made to retype laws and other legal documents, their number is massive, and only automatic re-typing (through OCR) is a viable option. Many of the documents that require OCR are published in the Official Gazette of the Kingdom of

Cambodia, which are published weekly by the Office of the Council of Ministers (modern versions of the Gazette are printed using unicode fonts, while old issues were printed using Limon fonts). Other legal documents take the shape of Prakas, letters or memos.

Another important target for OCR is research, a fundamental activity for a country's social and economic development. Advancing in research requires access to previous research reports, thesis, data, and to old documents, most of which only exist in printed form; they are not accessible in editable unicode text, and therefore not searchable. Providing access to these documents through search would be a huge step forward for Cambodian research.

Until now Khmer OCR is still infancy, especially for Khmer Limon legacy fonts, while machine learning and deep learning are significantly advancing in the fields of Computer Vision and pattern recognition. However, there were some literary research papers on Khmer OCR developments which were discussing and debating on various techniques to be used efficiently for Khmer OCR system. Those output results are comparative based on the methodologies used as mentioned in Table 1.

Fortunately, Tesseract 4.0 comes up with Deep Neural Networks as an black-box training tool [1] which works well for complex scripts that are lack of training dataset, for instance Khmer, Laos, Burmeses,...etc. Additionally, It also provided pre-trained recognition engines for 101 scripts [2], among them, Khmer script is also included. By default, OCR recognition engine for Khmer is only able to perform well on Khmer Unicode image but not good on Khmer Limon image.

That is why fine-tuning process is a significant step for improving the performance of pre-trained Tesseract 4.0 Khmer OCR engine in which new fonts were repeatedly learned with the constraint iterations from all similar shapes of Khmer Limon fonts that were converted into Unicode format by reorganizing the Unicode code-point with its correlate Limon glyph using FontForge tool [3].

Table 1: Prior Khmer OCR Development

Author	Accuracy	Technique
Chanoeurn CHEY et al. (2003) [4]	**92%** (10 Khmer characters)	Lagedre/Euclidean
Chanoeurn CHEY et al. (2006) [5]	**92.85%** (22pt); **91.66%** (18pt); **89.27%** (12pt)	Wavelet Descriptor
Mr. ING LENG IENG (2009) [6]	**98.88%** for Limon R1 (22pt); **92.48%** for Limon S1 (22pt)	Discrete Cosine Transform, HMM
Mr. KRUY VANNA (2011) [7]	**97.9%** for 19 types of Limon fonts	Vertical Component extr., Connected Component Analyse, & Scale Invariant Feature Transform extr.
Mr. En Sovann (2011) [8]	**96.1%** (Improve the work of Mr. Kruy Vanna by adding new fonts)	Vertical Component extr., Connected Component Analyse, & Scale Invariant Feature Transform extr.
Hao Jeudi (2011) [9]	**96.1%** (produce UI for the work of Mr. En Sovann)	improve UI on the work of Mr. En Sovann
Iech Setha (2012) [10]	**99%** for Khmer OS Content (36pt)	Edge detection & template matching
Sok Pongsametrey (2013) [11]	**98.54%** for Khmer OS content (36pt)	Support Vector Machine (SVM)
Hann Meng et al. (2014) [12]		Research on theory of ANN only
Mr. Vichet Chea. (2015) [13]	**92.74%** for 6 Khmer Unicode fonts	Tesseract 3
Mr. PHAN Neth (2016) [14]	**92%** (12pt) for 12 fonts	LSTM base

This paper demonstrates OCR training tool and technique to advance Khmer OCR system. It discusses about Tesseract 4.0 with Deep Neural Networks in section II, the how-to implement fine tune OCR engine for Khmer script in section III, ISRI Analytic Tools used for accuracy measurement in section IV, Testing dataset in section V, Accuracy comparison in section VI, and then it comes to conclusion in section VII.

II. TESSERACT 4.0 WITH DNN

Version 4.0 of Tesseract integrates a deep neural network (DNN) approach as a text line recognizer. DNN consists of multi-hidden

layers in between one input layer and one output layer. Each layer has interconnections of neurons or perceptrons with others in another network layer, constructing a big mass of neural network just looks like a network of millions cells inside human's brain. Technically, while the training process, DNN performs automatic feature extraction itself on huge unlabeled training data set [15], guided by a given mandatory network specification (Net Specs) to Tesseract 4.0 training tool by using Variable-size Graph Specification Language (VGSL) [16] including significant features, for instance variable size images as the input, convolutions, and LSTMs,...etc.

III. FINE TUNE OCR ENGINE FOR KHMER SCRIPT

Fine tuning is the process of combining existing training model, i.e Tesseract 4.0 pre-trained model, with a new training model using either the same or different neural network specification [17]. The following were the five steps for creating a Khmer OCR fine tune training model engine:

1. Training data for Khmer language was downloaded from Tesseract github [26], called langdata. It consisted of 7,200 lines of training text, 4,526 lines of numbers, 808 lines of punctuation, 102,416 lines of training text bigram freqs, 3,635 lines of training text unigram freqs, 2,461 lines of word bigrams and 151,860 lines of wordlist. Thus in total the training data was 272,906 lines. Notice that only training text file was required while the rest of the files were optional which were used for improving the training quality. In this stage, two important files i.e *.tiff and *.box were created [18],[19].
2. Starter traineddata was the process of creating initial training model for the process of training data from the scratch using tesseract command "tesstrain.sh" [19],[20]. In this command, we can define any specific font name that we want to train or specify the font directory instead of font name to train many fonts. For the experimental test, Limon S1, Limon S2 and Limon F1 unicode fonts were trained and an important flag, *"linedata_only"*, was set to tell the trainer to use LSTM Neural Network method. The flag *"lang"* was also stated to instruct the trainer to train engine for a specific language i.e khmer language (khm). As the result, three files were created i.e *.unicharset, *.traineddata, and *.lstmf [20].
3. Next step was the training step which used command “lstmtraining”. Checkpoint files were periodically written by the trainer during training process. Any checkpoint can be used to create a full traineddata for recognition model [21],[23].
4. Combine output step was the procedure of converting a checkpoint file into a full traineddata engine for recognition model using the same command “lstmtraining” but a few flags were required. The flag “*stop_training”* was used to tell the trainer to convert checkpoint file into a complete traineddata model. And the flag “*continue_from”* was to instruct the trainer to continue from either any existing checkpoint file or LSTM model file, which was extracted from any existing *.traineddata file using command “combine_tessdata -e” [17],[22].
5. Since a few Limon Unicode fonts were trained in the earlier process, i.e step 3, the accuracy of recognition model would be low. Thus the fine tune step was required. LSTM model was extracted from the Tesseract 4.0 pre-defined traineddata recognition model and combined with new trained model to create a full fine tune recognition model for all the trained fonts [17].

A complete process of how to implement a fine tune Khmer OCR recognition model was illustrated in Figure 1.

In this paper, five experimental fine tune recognition models and Tesseract pre-defined model were tested and measured their accuracies. The models were named as the following:

1. Tesseract 4.0 pre-trained engine
2. Engine for Limon S1 Unicode
3. Engine for Limon S1F1 Unicode

4. Engine for Limon S1S2F1 Unicode
5. Engine for Limon S1S2 Unicode
6. Engine for Limon S1S2 Unicode (new Nets)

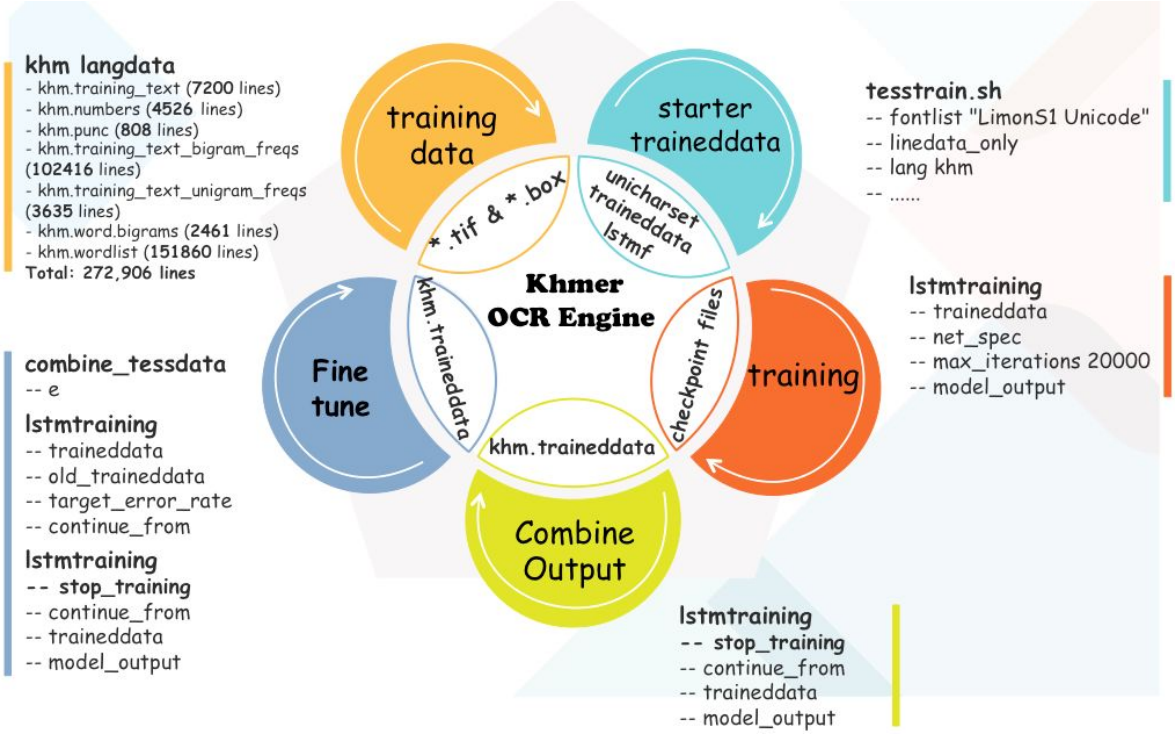

Figure 1: Summary Steps of how to create a fine tune Khmer OCR recognition model

The term ***"Limon S1/S2/F1 Unicode"*** was referred to all alike shapes of Limon S1/S2/F1 which was transformed into Unicode format by rearranging the Unicode code-point with its correlate Limon glyph.

While ***"Engine for Limon S1 Unicode"*** was a fine tune recognition model for Limon S1 Unicode font using the same neural network specification as pre-defined Tesseract 4.0 engine. The similar term also applied for ***"Engine for Limon S1F1/S1S2/S1S2F1 Unicode"***, only the number of fonts to be trained were different. However, ***"Engine for Limon S1S2 Unicode (new Nets)"*** was a fine tune engine for Limon S1S2 fonts using different neural network specification from Tesseract 4.0.

The detail of neural network specification of Tesseract 4.0 pre-defined and new NetSpecs were shown in Figure 2 and Figure 3.

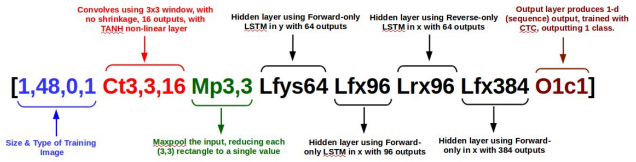

Figure 2: Network specification used by Tesseract 4.0 for pre- trained Khmer OCR engine

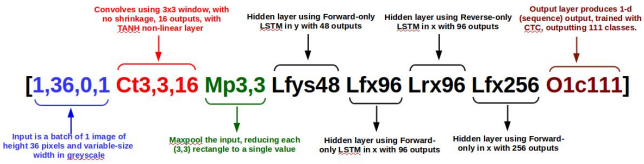

Figure 3: Network specification used for experimental training Khmer Limon S1S2 (new Nets) OCR engine

IV. ISRI Analytic Tool

The ISRI (Information Science Research Institute) Analytic Tool is a tool which was developed by the University of Nevada, Las Vegas (UNLV) for conducting Annual Tests of Optical Character Recognition (OCR) Accuracy from 1992 to 1996 [24].

This toolkit is command-line based application and consists of many measurements tools such as character accuracy, word accuracy, marked character efficiency, non-stopword accuracy, phrase accuracy and the cost of correcting automatic zoning errors [25]. This tool is not able to produce word accuracy for some complex scripts, i.e Khmer, because of no word segmentation tool for Khmer embedded in the ISRI Analytic Tool. This issue brings only character and cluster level accuracy for Khmer script.

V. Testing Dataset

Two types of images, images without noise and with noise, which contained Khmer texts with both 14 disparate Unicode fonts and 20 Legacy Limon fonts, were chosen as testing dataset. The former was an open testing text which was collected from internet on a few various domains such as political, news about pop stars and tourism place. It consisted of 11,272 clusters which were equal to 19,618 characters. And then the image file was automatically generated from the text using **text2image** command line. The latter was the publication of National Gazettes with Limont fonts. Those image datasets were tested with pre-trained engine and other 5 fine tune engines.

VI. Accuracy Comparison

6.1 Image without noise

Each image was generated for all Khmer Unicode and Limon fonts. The image was in *.tiff format with the size of 3600x4800px and the resolution of 300 dpi.

The Figure 4 and 5 showed the detail of the CHL and CLL accuracy of 6 recognition engines for individual font and the Figure 6 and 7 displayed the error rate results of those engines. The pre-trained Tesseract 4.0 engine produced better result for Khmer unicode fonts which ranged from 76.53% to 97.32% (CHL) and 67.28% to 98.80% (CLL) and the average accuracy was 87.49% (CHL) and 89.43% (CLL). However, the accuracy result for pre-unicode Limon fonts was not so good. In pre-defined Tesseract 4.0 engine, the accuracy of Limon F4 was only 12.93% (CHL) and 8.45% (CLL) while that of Limon S1, S2, S3, S5, and S7 were range from 85.41% (CHL) to 88.39% (CHL) and 86.80% (CLL) to 88.93% (CLL). In average, the accuracy was only 62.27% (CHL) and 60.08% (CLL).

The ***Engine for Limon S1 Unicode*** significantly produced good accuracy result for Limon fonts especially Limon S1 and S2 (S2 has similar shape as S1). The CHL accuracy of Limon S1 increased from 86.22% (Tesseract 4.0 pre-defined engine) to 93.30% (*Engine for Limon S1 Unicode*). For Limon fonts, the accuracy ranged from 20.15% to 93.30% (CHL) and from 13.45% to 94.39% (CLL). The average accuracy grew up to 70.66% (CHL) and 69.55% (CLL).

The ***Engine for Limon S1F1 Unicode*** gave much higher accuracy result for Limon S1, S2 and Limon F1. CHL accuracy of Limon F1 was from 41.59% (Tesseract pre-defined engine) to 88.74% (*Engine for Limon S1F1 Unicode*). The average accuracy was 71.58% (CHL) and 70.50% (CLL).

The ***Engine for Limon S1S2F1 Unicode*** remarkably produced accuracy result with the average of 72.35% (CHL) and 71.35% (CLL). However, the average accuracy of Khmer Unicode fonts in the fine tune engines were slightly decreased by 1% to 2% if compared to Tesseract pre-defined engine. This would need more investigation in the future to know what the real cause is.

Other experiments on ***Engine for Limon S1S2 Unicode*** vs ***Engine for Limon S1S2 Unicode (new Nets)*** found out that using different NetSpecs for the fine tuning process would help to produce a better accuracy result for both Unicode and Limon Fonts than the fine tuning engine using the same NetSpecs as

Tesseract pre-defined engine. Figure 8 illusted the detailed of what had been mentioned.

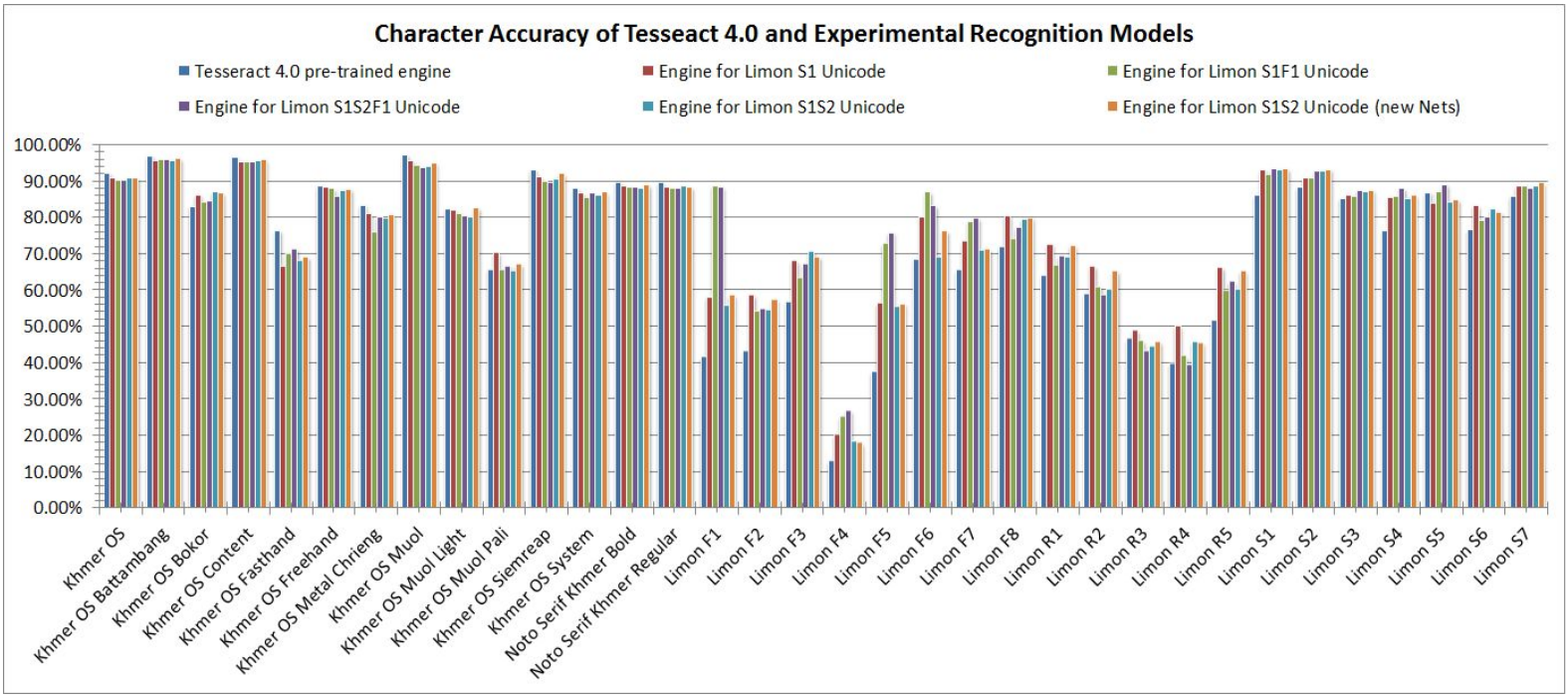

Figure 4 : The CHL Accuracy of six recognition models for Khmer Unicode and Limon Legacy fonts

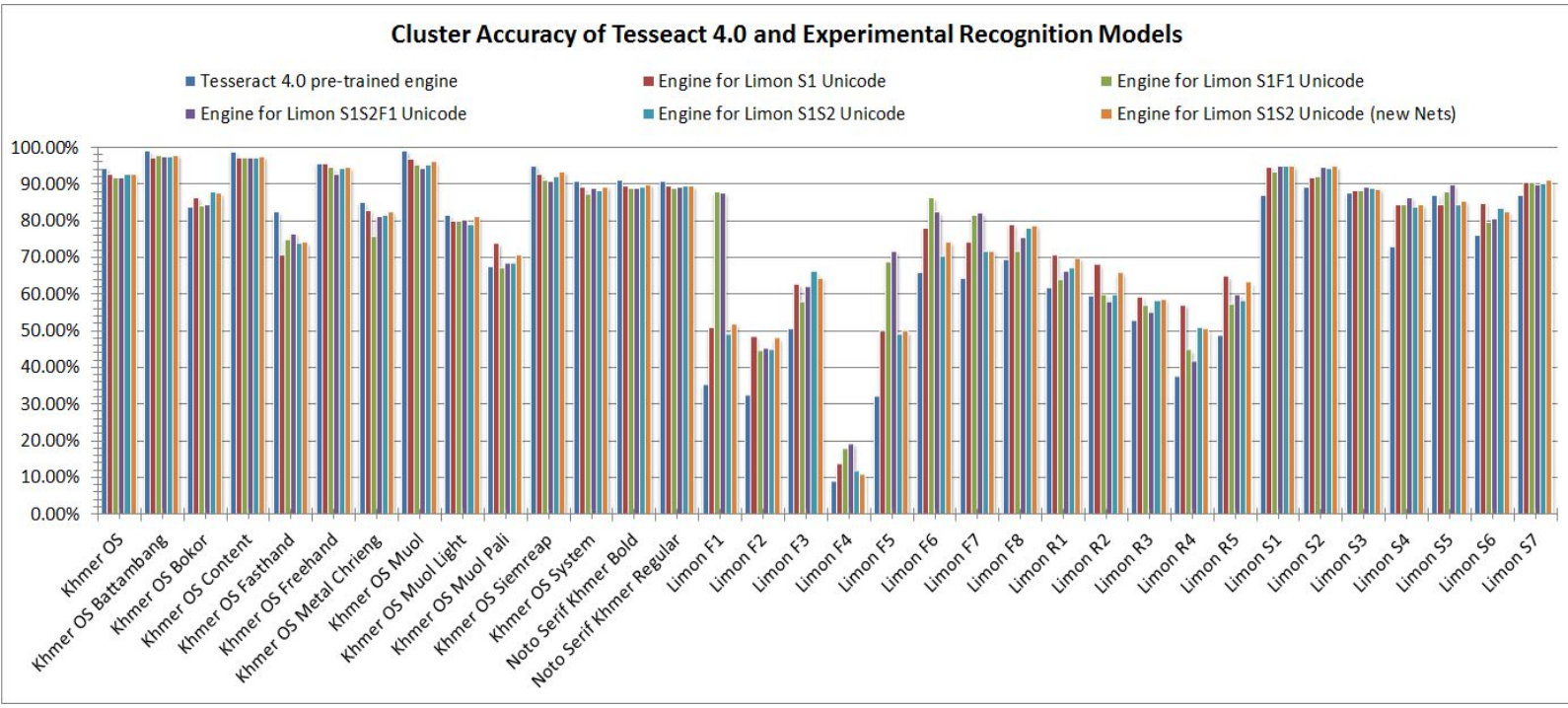

Figure 5: The CLL Accuracy of six recognition models for Khmer Unicode and Limon Legacy font

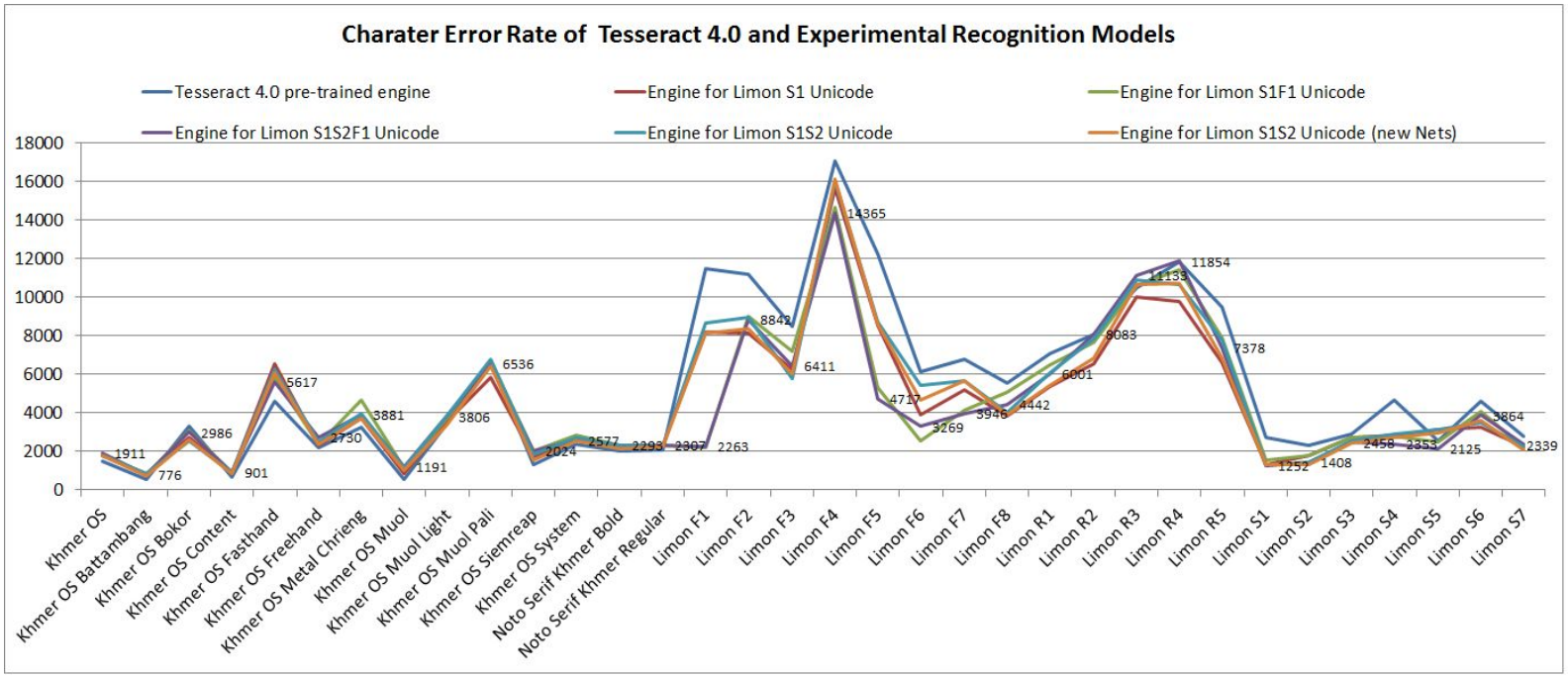

Figure 6: The Error rate of CHL Accuracy of six recognition models for Khmer Unicode and Limon fonts

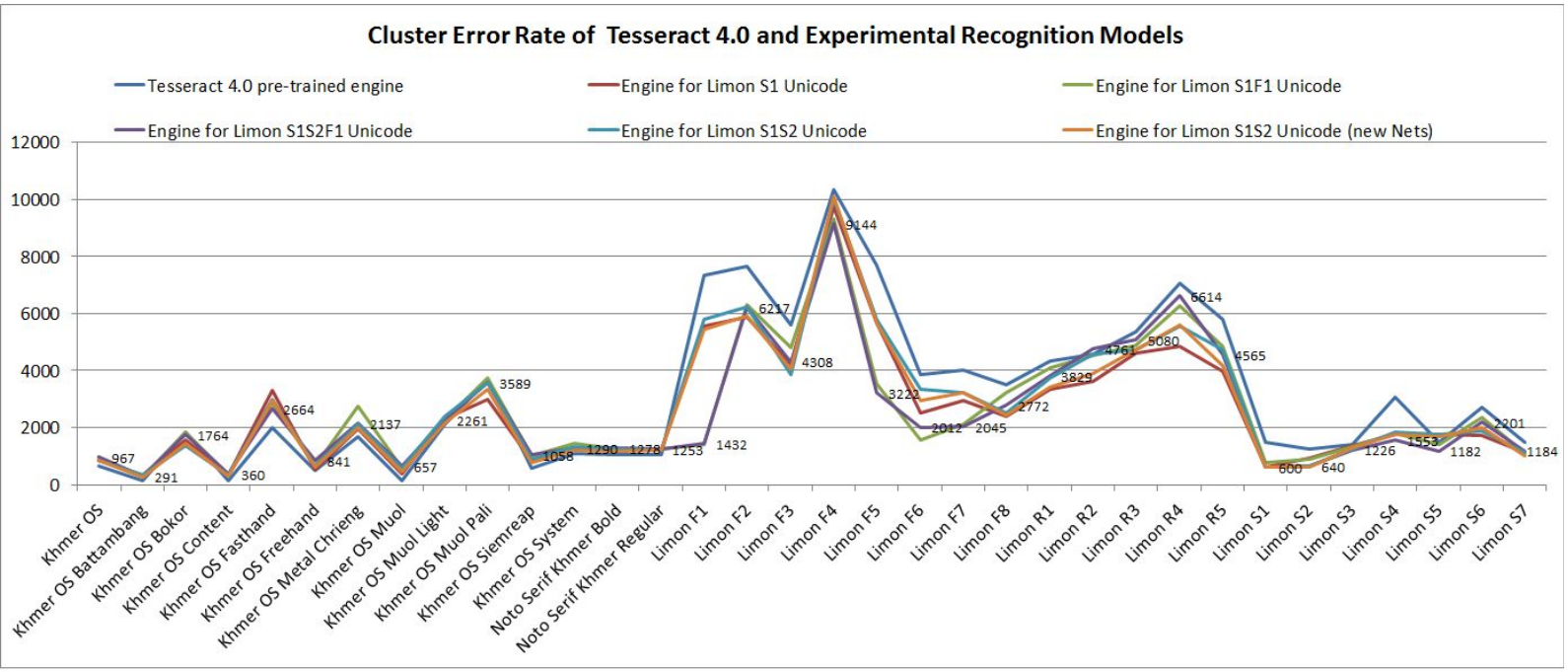

Figure 7: The Error rate of CLL Accuracy of six recognition models for Khmer Unicode and Limon fonts

Engine: Fine Tune Limon S1 & S2 Unicode

(new net_spec value "[1,36,0,1 Ct3,3,16 Mp3,3 Lfys48 Lfx96 Lrx96 Lfx256 O1c111]")

Testing Text: Characters : 19618 Clusters : 11272

Font Name	Size	Character Accuracy		Cluster Accuracy	
		Errors	Accuracy	Errors	Accuracy
Average [Khmer Unicode fonts] :	12	2,663.79	86.42%	1,333.64	88.17%

Font Name	Size	Character Accuracy		Cluster Accuracy	
		Errors	Accuracy	Errors	Accuracy
Average [Limon fonts] :	22	5,901.50	69.92%	3,524.45	68.73%

Engine: Fine Tune Limon S1 & S2 Unicode

(same net_space with Tesseract 4.0 "[1,48,0,1 Ct3,3,16 Mp3,3 Lfys64 Lfx96 Lrx96 Lfx384 O1c1]")

Font Name	Size	Character Accuracy		Cluster Accuracy	
		Errors	Accuracy	Errors	Accuracy
Average [Khmer Unicode fonts] :	12	2,815.07	85.65%	1,418.43	87.42%

Font Name	Size	Character Accuracy		Cluster Accuracy	
		Errors	Accuracy	Errors	Accuracy
Average [Limon fonts] :	22	6,154.30	68.46%	3,656.70	67.56%

Figure 8: The Average Accuracy of Engine Limon S1S2 Unicode and Engine Limon S1S2 Unicode (new nets)

6.2 Images with noise

Input images are the test set image files with an extension as *.tiff which were scanned from National Gazette newspaper publishcation. They contains texts of the font "Limon S1", "Limon S2", "Limon F1", and "Limon R1" with the equivalent size of 12pt. For the experimental purpose, all testing images with noise totally contain 4180 characters which were equivalent to 2205 clusters. One sample input image in figure 9 has a little bit of noise with the resolution of 300 dpi and the size of 563x480px. Practically, the noise, the

resolution and the size of images are the significant factors affecting the recognition result.

Unlike pre-trained Tesseract 4.0 engine, which could recognize the input images only 78.90% in character level (CHL) and 82.78% in cluster level (CLL), the fine tune Khmer OCR fine tune engines produced output texts better, especially for Limon fonts that have been trained.

Table 2 showed an accuracy results of both CHL and CLL levels of all input images with the font "Limon S1", generated by six different Khmer OCR Engines. Among them, the best

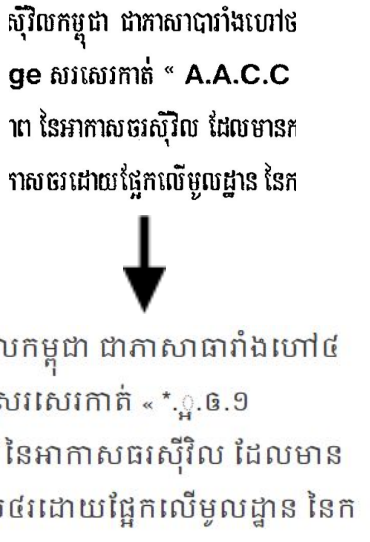

Figure 9: Input image and its corresponding output texts generated by one of Khmer OCR fine tune engines

Table 2: Accuracy Measurement of Input Images generated by 6 different Khmer OCR engines

Testing Text:		Characters: 4180		Cluster: 2205	
No.	Engine	Character		Cluster	
		Error	Accuracy	Error	Accuracy
1	Tesseract 4.0 pre-trained engine	882	78.90%	380	82.78%
2	Engine for LimonS1Unicode	837	79.97%	374	83.03%
3	Engine for LimonS1S2Unicode (new Nets)	825	80.27%	358	83.78%
4	Engine for LimonS1S2Unicode	792	81.05%	332	84.94%
5	Engine for LimonS1F1Unicode	731	82.51%	284	87.13%
6	Engine for LimonS1S2F1Unicode	778	81.40%	270	87.76%

engine was "***Engine for Limon S1 F1 Unicode***" which could recognize the image with font "Limon S1" up to 82.51% (CHL) and 87.13% (CLL). And the engine ***"Engine for Limon S1 S2 F1 Unicode"*** also does likewise, if it is compared in term of CLL accuracy which was 87.76%. The performances of five fine tune Khmer OCR engines compared favorably with ***Tesseract 4.0 pre-trained engine*** as displayed in Figure 10.

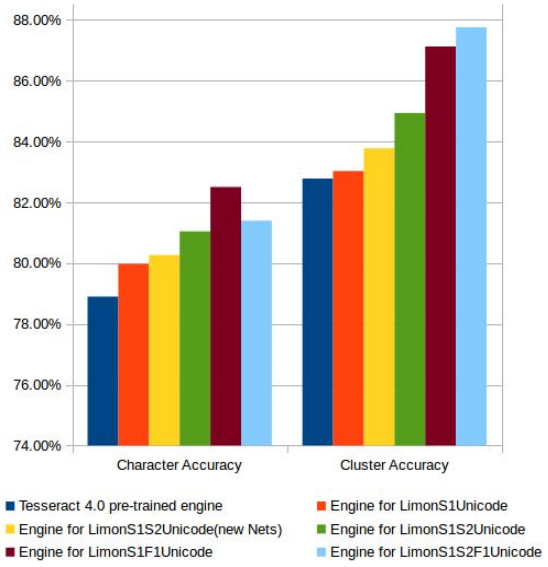

Figure 10: The chart of accuracy measurement comparing in 6 Khmer OCR engines

Both engines "***Engine for Limon S1 F1 Unicode***" and "***Engine for Limon S1 S2 F1 Unicode***", their character and cluster error rates, are decreased by 731 (CHL) and 270 (CLL) respectively (see figure 11).

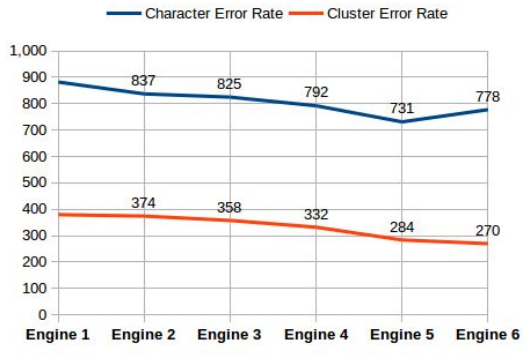

Figure 11: The chart of character and cluster error rate comparing in 6 Khmer OCR engines

VII. Conclusion

The inclusion of deep neural networks in Tesseract 4.0 has dramatically changed open source OCR software for Khmer. From being able to only recognize fonts and sizes that had been specifically trained (in earlier versions), Tesseract 4.0 is not only able to recognize the fonts that it has been trained on, but also other fonts, even if with a lower accuracy. Training on Khmer Unicode fonts has led to levels of recognition of over 90% for the trained fonts, but high levels (even if lower) have also been obtained for other selected legacy fonts. Training of the legacy fonts (by first turning the fonts into Unicode fonts) has led to similar levels of accuracy for the Unicode fonts (over 90%), even if the increase of the sample has reduced slightly the accuracy for the Unicode fonts. It is expected that better results might be obtained by adding new fonts to the model or using different network specifications. Regarding real documents, accuracy is also much higher than models that did not use neural networks, but improvement in pre-processing will also lead to high accuracy. Accuracy over 90% provides sufficiently good output for post-processing [28] using language models, providing hope to reach within the next year accuracy over 95% on fonts that have been trained.

References

[1] Tesseract 4.0 Github Wiki, https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00

[2] Pre-trained OCR engines provided by Tesseract 4.0, https://github.com/tesseract-ocr/tessdata

[3] FontForge Tool https://fontforge. github.io

[4] C. CHEY, P.KUMHOM, and K. CHAMNONGTHAI. *Khmer Printed Characters Recognition Using Lagendre Moment Descriptor, 2003.*

[5] C. CHEY, P.KUMHOM, and K. CHAMNONGTHAI. *Khmer Printed Characters Recognition Using Wavelet Descriptors, 2006.*

[6] I. LENG IENG, K. SOCHENDA and C. SOKHOUR, *Khmer OCR for Limn R1 Size 22 Report,* PAN Localization Cambodia (PLC) of IDRC, 2009.

[7] KRUY VANNA, *A Proposed Multi-Feature Extraction Method for Khmer OCR,* Graduate School of Global Information and Telecommunication Studies, WASEDA University, 2011.

[8] En Sovann, *Khmer Optical Character Recognition.* Engineering Degree Thesis, 2011.

[9] Hao Jeudi, *Additional Research on Khmer Optical Character Recognition.* Research Report, 2011.

[10] Iech SETHA, *Khmer Printed Character Recognition*, RUPP, 2012.

[11] Sok Pongsametrey, *Khmer Printed Character Recognition using Support Vector Machine (SVM) based*. Master thesis at RUPP, 2013, http://www.apsipa.org/proceedings_2014/Data/paper/1407.pdf

[12] Hann Meng et al. 2014, *Khmer Character Recognition using Artificial Neural Network,* Faculty of engineering, Lucian Blaga University of Sibiu, Romania. http://www.apsipa.org/proceedings_2014/Data/paper/1408.pdf

[13] Mr. Vichet Chear. *Optical Character Recognition Engine for Khmer Language.* National Institute of Posts, Telecoms & ICT. http://rnd.niptict.edu.kh/ocr/index.php#

[14] Mr. PHAN Neth, *Long Short-Term Memory based for Khmer Optical Character Recognition*, INSTITUTE DE TECHNOLOGY DU CAMBODIA, 2016.

[15] Tutorial on Deep Learning and Neural Networks, https://deeplearning4j.org/neuralnet-overview

[16] Tutorial on Viable-Graph Size Languages used in Neural Networks Specification, https://github.com/tesseract-ocr/tesseract/wiki/VGSLSpecs

[17] Tesseract Tutorial on How to fine tune for Impact font https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00#fine-tuning-for-impact

[18] Tesseract Tutorial on Overview of training process https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00#overview-of-training-process

[19] Tesseract Tutorial On Creating training data

https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00#creating-training-data

[20] Tesseract Tutorial On Creating starter traineddata https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00#creating-starter-traineddata

[21] Tesseract Tutorial On Training from the scratch https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00#training-from-scratch

[22] Tesseract Tutorial On Combining the output files https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00#combining-the-output-files

[23] Tesseract Tutorial On Understanding the various files used during training https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00#understanding-the-various-files-used-during-training

[24] ISRI Analytic Tools, https://github.com/eddieantonio/isri-ocr-evaluation-tools

[25] ISRI Analytic Tools Userguide https://pdfs.semanticscholar.org/dc3e/f1e05b4b629de5db721efb156d82556ff362.pdf

[26] Tesseract langdata https://github.com/tesseract-ocr/langdata/tree/master/khm

[27] Image pre-processing techniques on input images, https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality#image-processing

[28] OCR post-processing for Khmer language: Error detection using Conditional Random Field http://ona2017.khmernlp.org/wp-content/uploads/paper/papers/paper/ona2017_OCR_post_processing_for_.pdf