International Journal of Research in Engineering, Science and Management
Volume-1, Issue-9, September-2018
www.ijresm.com | ISSN (Online): 2581-5782

Optical Character Recognition (OCR)

Cheripelli Vijay[1], Attarde Gokarna[2], Badade Sagar[3]
[1,2,3]*Student, Department of Computer Engineering, MGMCET, Navi Mumbai, India*

***Abstract*—Aim to understand, utilize and improve the open source Optical Character Recognizer (OCR) software, OCR opus, to better handle some of the more complex recognition issues such as unique language alphabets and special characters such as mathematical symbols. We extended the functionality of OCR opus to work with any language by creating support for UTF-8 character encoding. We also created a character and language model for the Hungarian language. This will allow other users of the software to perform character recognition on Hungarian input without having to train a completely new character model**

***Index Terms*—data mining, neural networks, natural language programming, deep learning**

I. Introduction

We are moving forward to a more digitized world. Computer and PDA screens are replacing the traditional books and newspapers. Also the large amount of paper archives which requires maintenance as paper decays over time lead to the idea of digitizing them instead of simply scanning them. This requires recognition software that is capable in an ideal version of reading as well as humans. Such OCR software is also needed for reading bank checks and postal addresses. Automating these two tasks can save many hours of human work. OCRopus was created by Professor Tom Breuel from the DFKI (German Research Center for Artificial Intelligence at Kaiserslautern, Germany). Google sponsored the project on April 09, 2007 with the goal of providing an open source OCR system capable of performing multiple digitization functions. The application of this software ranged from general desktop use and simple document conversion to historical document analysis and reading aids for visually impaired users

II. History of OCR

The idea of OCR technology has been around for a long time and even predates electronic computers.

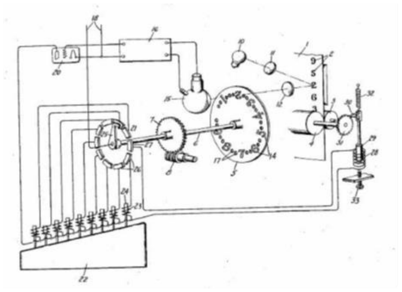


Fig. 1. Statistical machine design by Paul W. Handel

This is an image of the original OCR design proposed by Paul W. Handel in 1931. He applied for a patent for a device "in which successive comparisons are made between a character and a character image." A photo-electric apparatus would be used to respond to a coincidence of a character and an image. This means you would shine a light through a filter and, if the light matches up with the correct character of the filter, enough light will come back through the filter and trigger some acceptance mechanism for the corresponding character. This was the first documented vision of this type of technology. The world has come a long way since this prototype.

III. Techniques

A. Template Matching Method

In 1956, Kelner and Glauberman used magnetic shift registers to project two-dimensional information. The reason for this is to reduce the complexity and make it easier to interpret the information. A printed input character on paper is scanned by a photo detector through a slit. The reflected light on the input paper allows the photo detector to segment the character by calculating the proportion of the black portion within the slit. This proportion value is sent to a register which converts the analog values to digital values. These samples would then be matched to a template by taking the total sum of the differences between each sampled value and the corresponding template value. While this machine was not commercialized, it gives us important insight into the dimensionality of characters. In essence, characters are two-dimensional, and if we want to reduce the dimension to one, we must change the shape of the character for the machine to recognize it.

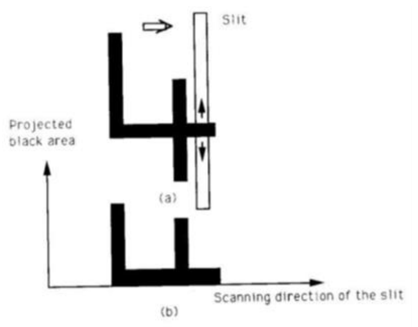


Fig. 2.Illustration of 2-D reduction to 1-D by a slit. (a) An input numeral "4" and a slit scanned from left to right. (b) Black area projected onto x axis, the scanning direction of the slit

B. *Peephole Method*

This is the simplest logical template matching method. Pixels from different zones of the binary character are matched to template characters. An example would be in the letter A, where a pixel would be selected from the white hole in the center, the black section of the stem, and then some others outside of the letter.

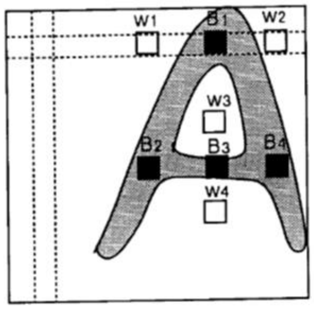


Fig. 3. Illustration of the peephole method

Each template character would have its own mapping of these zones that could be matched with the character that needs to be recognized. The peephole method was first executed with a program called Electronic Reading Automation in 1957.

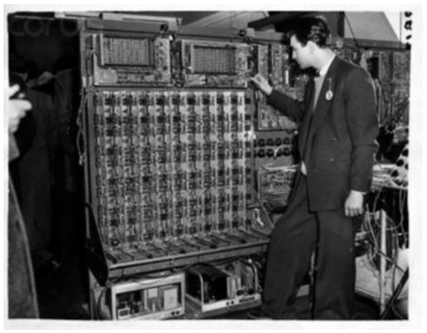


Fig. 4. The Solartron electronic reading automaton

This was produced by Solartron Electronics Groups Ltd. and was used on numbers printed from a cash register. It could read 120 characters per second, which was quite fast for its time, and used 100 peepholes to distinguish characters.

C. *Structured Analysis Method*

It is very difficult to create a template for handwritten characters. The variations would be too large to have an accurate or functional template. This is where the structure analysis method came into play. This method analyzes the character as a structure that can be broken down into parts. The features of these parts and the relationship between them are then observed to determine the correct character. The issue with this method is how to choose these features and relationships to properly identify all of the different possible characters. Peepholes can be viewed on a larger scale. Instead of single pixels, we can now look at a slit or 'stroke' of pixels and determine their relationship with other slits. This technique was first proposed in 1954 with William S. Rohland's "Character Sensing System" patent using a single vertical scan. The features of the slits are the number of black regions present in each slit. This is called the cross counting technique.

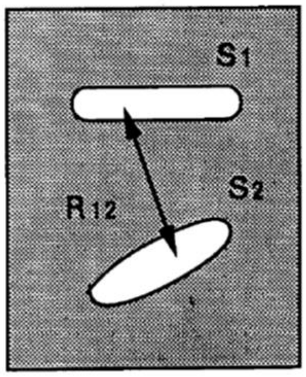


Fig. 5. Extension of the peephole method to structure analysis

IV. OCRopus

The project was initially expected to run for three years as a support for three Ph.D. students but was later released as software under the Apache license. This means that the project is now open source and free to use with the preservation of the copyright notice and disclaimer. The major advance in the program's development was the incorporation of the Tesseract character recognizer along with growing language modeling tools. The last operational version is OCRopus 0.4.4 (alpha).

A. *Language Modeling*

In natural language processing, language models are probabilistic models whose main goal is to assign a probability to a word or a sentence. In speech recognition for example, they are used to predict the next word given the previous ones or revise an entire sentence and compare its likelihood against another similar sounding sentence. For OCR systems, language models are used in a similar fashion to determine the probability of a generated word to occur in a sentence. This can be done in different ways.

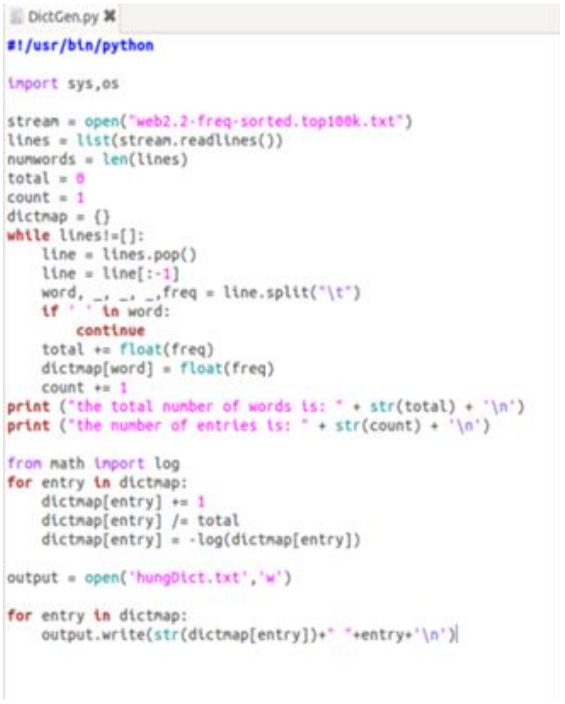
One way is the grammar approach, which defines the possible links and relations between words depending on their position in a sentence. This can help determine whether we are expecting the next word to be a verb, adjective, noun, conjunction, etc. and proceed to reject a word that cannot fit. One obstacle facing this approach is long term dependencies in languages such as English. For example, we can have sentences like "The building that I painted collapsed" and "The building that I painted yesterday". In the first sentence we have two verbs in a row. In the second we have "I painted yesterday" which is a very weak sentence and if we do not consider the entire context and the possibility of having long term dependencies, this can lead to a lot of complexity. It is almost impossible to have an exhaustive model of the entire grammar of a language.

Another language modeling method is to use the word frequencies and this can be done with different levels of complexity. The simplest one known as a unigram model is to use a word corpus which is generated from a set of texts in the desired language and then each word is associated its frequency in those texts. This information can then be used to rank the words by their probability of appearing in that language. The size of the word corpus can improve the accuracy but can also increase the ambiguity when it includes extremely rare words. For this reason there exists custom word corpora depending on the size, the topics and the era of the text supports it was generated from. N-gram models are an extension of unigram models. They simply look at the probability of the occurrence of a word following two or more other words. In practice unigram models proved to be sufficient and there are some trends to combine different language modeling approaches to reach better accuracy.

B. Language Modeling Implementation

OCR opus opted for the use of weighted final transducers to create language models. These FSTs provide a lot of advantages such as the possibility of composing language models or concatenating them.

The OCR opus documentation specifies that language modeling is done within the ocrofst collection that provides different tools and scripts for creating language models in FST form. These scripts are written in the python programming language that has a dedicated library for FST handling called "pyopenfst".



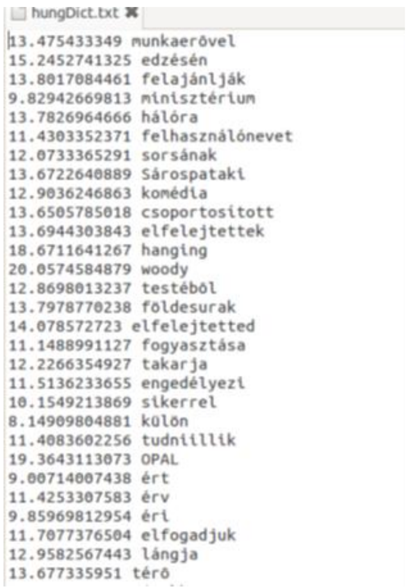
DictGen.py

```
#!/usr/bin/python

import sys,os

stream = open("web2.2-freq-sorted.top100k.txt")
lines = list(stream.readlines())
numwords = len(lines)
total = 0
count = 1
dictmap = {}
while lines!=[]:
    line = lines.pop()
    line = line[:-1]
    word, _, _, _,freq = line.split("\t")
    if ' ' in word:
        continue
    total += float(freq)
    dictmap[word] = float(freq)
    count += 1
print ("the total number of words is: " + str(total) + '\n')
print ("the number of entries is: " + str(count) + '\n')

from math import log
for entry in dictmap:
    dictmap[entry] += 1
    dictmap[entry] /= total
    dictmap[entry] = -log(dictmap[entry])

output = open('hungDict.txt','w')

for entry in dictmap:
    output.write(str(dictmap[entry])+" "+entry+'\n')
```

Fig. 6. Dictionary generation code

We decided to use and modify these scripts to create a language model for the Hungarian Language. The ocropus-lm-dict2linefst.py script is a fst language model generator that takes in a dictionary with word entries and their corresponding inverted natural logarithm probabilities to produce the corresponding FST. The reason for using the ln() operation is to avoid underflow since the probabilities are in the 0 to 1 range. The first step was to generate such dictionary from a word corpus. We chose to use the "Hungarian Webcorpus" that can be found at http://mokk.bme.hu/resources/webcorpus/ as it is available under a permissive Open Content License. For this we wrote a python script that can create the needed dictionary from any word corpus from the "Hungarian Webcorpus"

This script generated the file hungDict.txt that contained the Hungarian words along with their negative log probabilities as shown in the screen shot below.



hungDict.txt

```
13.475433349 munkaerővel
15.2452741325 edzésén
13.8017084461 felajánlják
9.82942669813 minisztérium
13.7826964666 hálóra
11.4303352371 felhasználónevet
12.0733365291 sorsának
13.6722640889 Sárospataki
12.9036246863 komédia
13.6505785018 csoportosított
13.6944303843 elfelejtettek
18.6711641267 hanging
20.0574584879 woody
12.8698013237 testéből
13.7978770238 földesurak
14.078572723 elfelejtetted
11.1488991127 fogyasztása
12.2266354927 takarja
11.5136233655 engedélyezi
10.1549213869 sikerrel
8.14909804881 külön
11.4083602256 tudniillik
19.3643113073 OPAL
9.00714007438 ért
11.4253307583 érv
9.85969812954 éri
11.7077376504 elfogadjuk
12.9582567443 lángja
13.677335951 térő
```

Fig. 7. Cost dictionary for the Hungarian language

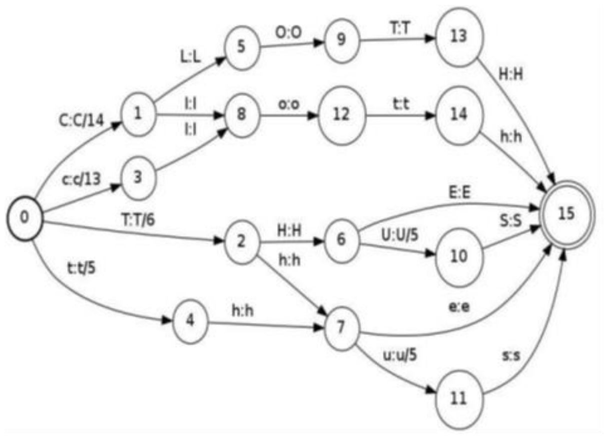


Fig. 8. Sample portion of an English language model FST

The next step was to use the ocropus-lm-dict2linefst script to generate the corresponding FST to this Hungarian dictionary. When trying to run this script we immediately faced the issue of not being able to process all Hungarian characters. The reason for this is that the script was initially written to support the Latin-2 encoding that does not have the capability of representing letters with accents; a necessary feature in the Hungarian Language.

To solve this problem we had to change the script to read the Hungarian dictionary in the UTF-8 encoding and make the appropriate changes in the FST creation pipeline. This lead us to refine the Add String() method in the pyopen fst library which is now able to accept any UF8 encoded character.

Finally after creating our language model FST, we introduced it into the OCR opus pipeline.

This picked out errors and in coherencies of the raw output from the character model recognition step.

The output of our Hungarian language model shows a significant improvement in accuracy. The mistakes that exist seemed to be related to segmentation errors rather than inconsistencies in our character and language model.

V. Using of OCRopus Software

The code base for OCR opus has gone through many revisions in the past few years and lacked sufficient updates to their documentation. This particular revision was meant to consolidate many of the OCR steps into simpler, concise functions. In the process, some less useful functions remained in the code base and the useful ones were not marked as so. This made our initial use of this software challenging. To alleviate some of the challenges with this revision, we created a step-by-step guide starting from the installation of OCR opus and ending with text files, character models, and language models. This also includes some first-hand experience with errors and elapsed time for some of the more CPU intensive commands.

Note: Anything bolded in the following sections is a command that can be run in a Linux terminal.

VI. Result

After successfully creating our Hungarian character and language models, we assessed the accuracy of the OCR opus software. We compared the results of our models versus the default English models on a Hungarian algebra textbook written in 1977 by László Fuchs. We were able to successfully recognize Hungarian accented characters and increase the overall accuracy. We used a character based approach to assess the accuracy and increase the rate of correct recognition by 8%. The original accuracy with the English character model was 86% on a sample of 1700 characters and we increased this to 93.5% with our Hungarian character model. We manually calculated the accuracy because the ground truth data for this text did not exist in digital form. From our tests, we have concluded that our character and language model yield significantly better results than the default English models.

VII. Conclusion

The goal of OCR opus is to provide an accessible, flexible, and simple tool to preform optical character recognition. In its current state, it is not the most user friendly utility and still has many kinks to work out. This is all understandable because it is in an alpha stage of development, and will require some more attention before an official release. OCR opus does an amazing job pre-processing and segmenting images and allows for many fine adjustments to fulfill a variety of user needs. It is now just a matter of reorganizing and optimizing the code to create a user friendly experience. With time, we believe OCR opus will be one of leading names in optical character recognition software.

References

[1] Hyvärinen, Aapo, and Erkki Oja. "Algorithms and Applications." Independent Component Analysis (2000): 1-31. Web. Jan.-Apr. 2012.
[2] Mori, Shunji, Ching Y. Suen, and Kazuhiko Yamamoto. Historical Review of OCR Research and Development. Tech. no. 0018-9219. Vol. 80. IEEE, 1992. Print. Proceedings of the IEEE.
[3] Holley, Rose. "How Good Can It Get? Analysing and Improving OCR Accuracy in Large Scale Historic Newspaper Digitisation Programs." D-Lib Magazine. Web. 28 Mar. 2012. <http://www.dlib.org/dlib/march09/holley/03holley.html>.
[4] Breuel, Thomas M. The OCRopus Open Source OCR System. Tech. DFKI and U. Kaiserslautern, Oct. 2007. Web. 5 Apr. 2012.
[5] Handel, Paul W. Statistical Machine. General Electric Company, assignee. Patent 1915993. 27 June 1933. Print.
[6] Smith, Ray. "Tesseract OCR Engine." Lecture. Google Code. Google Inc, 2007. Web. Mar.-Apr. 2012. <http://tesseract-ocr.googlecode.com/files/TesseractOSCON.pdf>.
[7] Teh, Yee Whye, Simon Osindero, and Geoffrey E. Hinton. "Energy-Based Models for Sparse Overcomplete Representations." Journal of Machine Learning Research 4, 03 Dec. 2003. Web.
[8] Mohri, Mehryar, Fernando Pereira, and Michael Riley. "Weighted Finite-State Transducers in Speech Recognition." Publications of Mehryar Mohri. 2000. Web. 10 Apr. 2012. <http://www.cs.nyu.edu/~mohri/pub/asr2000.ps>.
[9] "Finite State Automata." Strona Główna. Web. 10 Apr. 2012. <http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/thesis/node12.html>. 51
[10] Greenfield, Kara and Sarah Judd. “Open Source Natural Language Processing.” Worcester Polytechnic Institute. Web. 28 Apr. 2010. <http://www.wpi.edu/Pubs/E-project/Available/E-project-042810-055257/>.