MES'S WADIA COLLEGE OF ENGINEERING, PUNE

Honors* in Artificial Intelligence and Machine Learning Fourth year of Engineering

410302: Machine learning Laboratory

NAME OF STUDENT:	**CLASS: BE**
SEMESTER/YEAR: VII	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY: Dr. N. F. Shaikh	**EXPERIMENT NO: 02**

TITLE: OCR using ANN.

PROBLEM STATEMENT: Recognize optical character using ANN

OBJECTIVES:

- To understand the different graphical and Hidden Markov models of learning

OUTCOMES:

- Select the appropriate type of neural network architecture and draw the inferences from the different graphical and hidden Markov models.

PRE-REQUISITES:

1. Knowledge of python programming
2. Basic knowledge of machine learning and neural networks

THEORY:

OCR = Optical Character Recognition. In other words, OCR systems transform a two-dimensional image of text, that could contain machine printed or handwritten text from its image representation into machine-readable text. OCR as a process generally consists of several sub-processes to perform as accurately as possible. The subprocesses are:

Preprocessing of the Image

Text Localization

Character Segmentation

Character Recognition

Post Processing

The sub-processes in the list above of course can differ, but these are roughly steps needed to approach automatic character recognition. In OCR software, it's main aim to identify and capture all the unique words using different languages from written text characters.

For almost two decades, optical character recognition systems have been widely used to provide automated text entry into computerized systems. Yet in all this time, conventional OCR systems (like zonal OCR) have never overcome their inability to read more than a

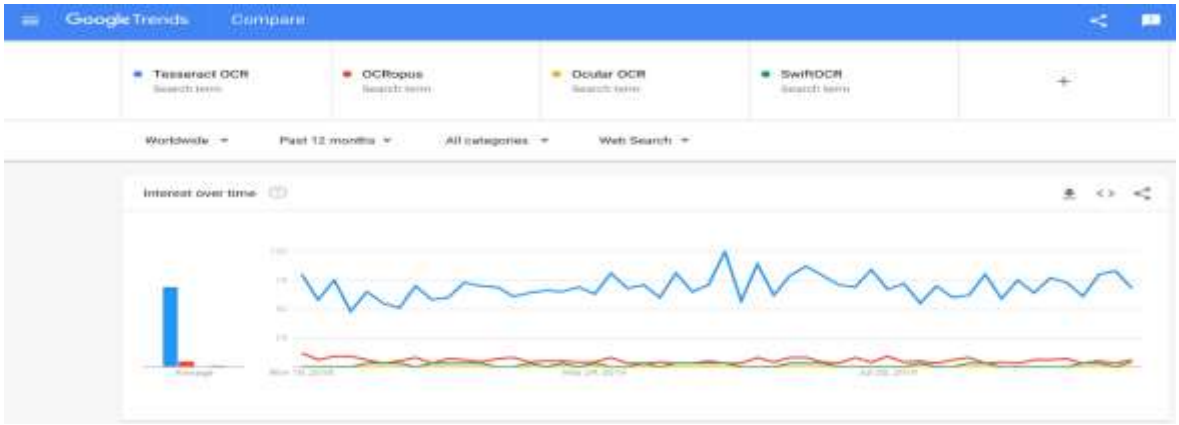
handful of type fonts and page formats. Proportionally spaced type (which includes virtually all typeset copy), laser printer fonts, and even many non-proportional typewriter fonts, have remained beyond the reach of these systems. And as a result, conventional OCR has never achieved more than a marginal impact on the total number of documents needing conversion into digital form.

Optical Character Recognition remains a challenging problem when text occurs in unconstrained environments, like natural scenes, due to geometrical distortions, complex backgrounds, and diverse fonts. The technology still holds an immense potential due to the various use-cases of deep learning based OCR like :

- building license plate readers
- digitizing invoices
- digitizing menus
- digitizing ID cards

Open Source OCR Tools

Tesseract - an open-source OCR engine that has gained popularity among OCR developers. Even though it can be painful to implement and modify sometimes, there weren't too many free and powerful OCR alternatives on the market for the longest time. Tesseract began as a Ph.D. research project in HP Labs, Bristol. It gained popularity and was developed by HP between 1984 and 1994. In 2005 HP released Tesseract as an open-source software. **Since 2006 it is developed by Google.**



google trends comparison for different open source OCR tools

OCRopus - OCRopus is an open-source OCR system allowing easy evaluation and reuse of the OCR components by both researchers and companies. **A collection of document analysis programs, not a turn-key OCR system.** To apply it to your documents, you may need to do some image preprocessing, and possibly also train new models. In addition to the recognition scripts themselves, there are several scripts for ground truth editing and correction, measuring error rates, determining confusion matrices that are easy to use and edit.

Ocular - Ocular works best on documents printed using a hand press, including those written in multiple languages. It operates using the command line. **It is a state-of-the-art historical OCR system. Its primary features are:**

- Unsupervised learning of unknown fonts: requires only document images and a corpus of text.
- Ability to handle noisy documents: inconsistent inking, spacing, vertical alignment
- Support for multilingual documents, including those that have considerable word-level code-switching.
- Unsupervised learning of orthographic variation patterns including archaic spellings and printer shorthand.
- Simultaneous, joint transcription into both diplomatic (literal) and normalized forms.

SwiftOCR - I will also mention the OCR engine written in Swift since there is huge development being made into advancing the use of the Swift as the development programming language used for deep learning. Check out blog to find out more why. SwiftOCR is a fast and simple OCR library that uses neural networks for image recognition. **SwiftOCR claims that their engine outperforms well known Tessaract library.**

Tesseract OCR

Tesseract is an open source text recognition (OCR) Engine, available under the Apache 2.0 license. It can be used directly, or (for programmers) using an API to extract printed text from images. It supports a wide variety of languages. Tesseract doesn't have a built-in GUI, but there are several available from the 3rdParty page. Tesseract is compatible with many programming languages and frameworks through wrappers that can be found here. It can be used with the existing layout analysis to recognize text within a large document, or it can be used in conjunction with an external text detector to recognize text from an image of a single text line.

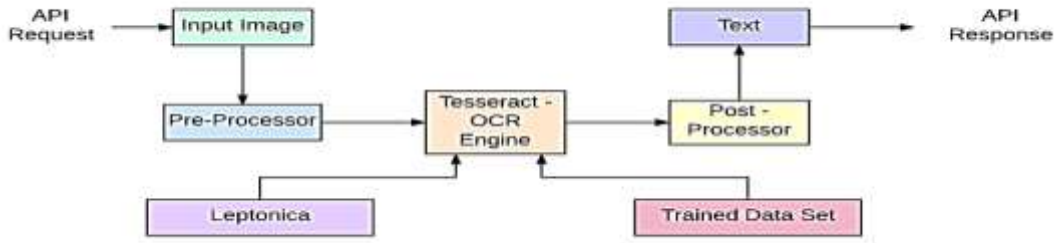


Fig: OCR Process Flow to build API with Tesseract from a blog post

Tesseract 4.00 includes a new neural network subsystem configured as a text line recognizer. It has its origins in OCRopus' Python-based LSTM implementation but has been redesigned for Tesseract in C++. The neural network system in Tesseract pre-dates TensorFlow but is compatible with it, as there is a network description language called Variable Graph Specification Language (VGSL), that is also available for TensorFlow.

To recognize an image containing a single character, we typically use a Convolutional Neural Network (CNN). Text of arbitrary length is a sequence of characters, and such problems are solved using RNNs and LSTM is a popular form of RNN. Read this post to learn more about LSTM.

Technology - How it works

LSTMs are great at learning sequences but slow down a lot when the number of states is too large. There are empirical results that suggest it is better to ask an LSTM to learn a long sequence than a short sequence of many classes. Tesseract developed from OCRopus model in Python which was a fork of a LSMT in C++, called CLSTM. CLSTM is an implementation of the LSTM recurrent neural network model in C++, using the Eigen library for numerical computations.

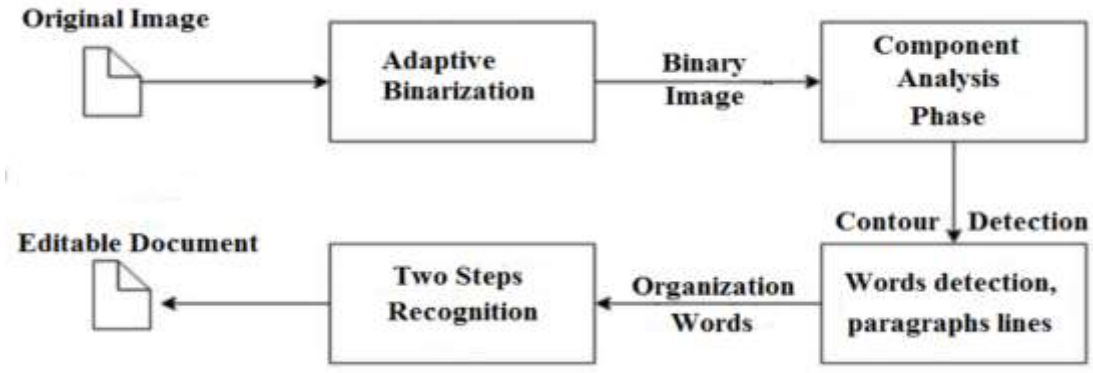


Fig: Tesseract 3 OCR process from paper

Legacy Tesseract 3.x was dependant on the multi-stage process where we can differentiate steps:

- Word finding
- Line finding
- Character classification

Word finding was done by organizing text lines into blobs, and the lines and regions are analyzed for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character spacing. Recognition then proceeds as a two-pass process. In the first pass, an attempt is made to recognize each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text lower down the page. Modernization of the Tesseract tool was an effort on code cleaning and adding a new LSTM model. The input image is processed in boxes (rectangle) line by line feeding into the LSTM model and giving output. In the image below we can visualize how it works.

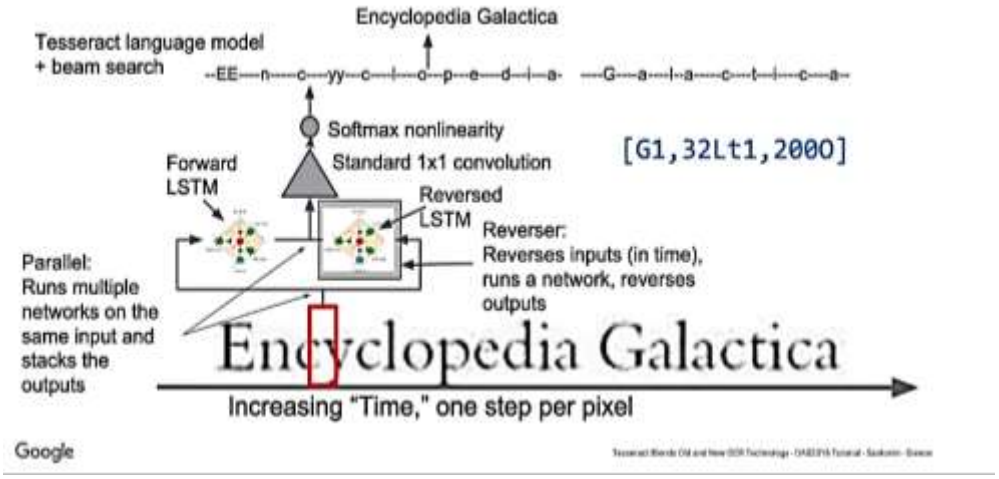


Fig: How Tesseract uses LSTM model presentation

After adding a new training tool and training the model with a lot of data and fonts, Tesseract achieves better performance. Still, not good enough to work on handwritten text and weird fonts. It is possible to fine-tune or retrain top layers for experimentation.

Installing Tesseract

Installing tesseract on Windows is easy with the precompiled binaries found here. Do not forget to edit "path" environment variable and add tesseract path. For Linux or Mac installation it is installed with few commands.

After the installation verify that everything is working by typing command in the terminal or cmd:

```
$ tesseract --version
```

And you will see the output similar to:

tesseract 4.0.0
leptonica-1.76.0
libjpeg 9c : libpng 1.6.34 : libtiff 4.0.9 : zlib 1.2.8
Found AVX2
Found AVX
Found SSE

You can install the python wrapper for tesseract after this using pip.
`$ pip install pytesseract`

Tesseract library is shipped with a handy command-line tool called tesseract. We can use this tool to perform OCR on images and the output is stored in a text file. If we want to integrate Tesseract in our C++ or Python code, we will use Tesseract's API.

Running Tesseract with CLI

Call the Tesseract engine on the image with *image_path* and convert image to text, written line by line in the command prompt by typing the following:

```
$ tesseract image_path stdout
```

To write the output text in a file:

```
$ tesseract image_path text_result.txt
```

To specify the language model name, write language shortcut after *-l* flag, by default it takes English language:

```
$ tesseract image_path text_result.txt -l eng
```

By default, Tesseract expects a page of text when it segments an image. If you're just seeking to OCR a small region, try a different segmentation mode, using the *--psm* argument. There are 14 modes available which can be found here. By default, Tesseract fully automates the page segmentation but does not perform orientation and script detection. To specify the parameter, type the following:

```
$ tesseract image_path text_result.txt -l eng --psm 6
```

**There is also one more important argument, OCR engine mode (oem).

Tesseract 4 has two OCR engines — Legacy Tesseract engine and LSTM engine.

There are four modes of operation chosen using the --oem option.

0 Legacy engine only.
1 Neural nets LSTM engine only.
2 Legacy + LSTM engines.
3 Default, based on what is available.

OCR with Pytesseract and OpenCV

Pytesseract is a wrapper for Tesseract-OCR Engine. It is also useful as a stand-alone invocation script to tesseract, as it can read all image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, and others. More info about Python approach read here. The code for this tutorial can be found in this repository https://nanonets.com/blog/ocr-with-tesseract/#:~:text=OCR%20with%20Pytesseract%20and%20OpenCV%20Pytesseract%20is%20a,others.%20More%20info%20about%20Python%20approach%20read%20here

QUESTIONS:

1. What are popular OCR Applications in the Real World?
2. Summarize Text Recognition with Tesseract OCR in your own words.