**[langname].[fontname].[expN].[file-extension]**

**OCR**

Any Typical machine learning OCR pipeline follows the following steps :

* Image preprocessing
* Text detection
* Text recognition

## Preprocessing

1. Remove the noise from the image
2. Remove the complex background from the image
3. Handle the different lightning condition in the image

01. Invert an Image:

03: Binarization:

04: Noise Reduction:

05: Dilation and Erosion:

06: Rotation and Deskewing:

07: Removing Borders:

08: Missing Borders:

09 : Transparency / Alpha Channel

### CRNN

This neural network architecture integrates feature extraction, sequence modeling, and transcription into a unified framework. This model does not need character segmentation. The convolution neural network extracts features from the input image(text detected region). The deep bidirectional recurrent neural network predicts label sequence with some relation between the characters. The transcription layer converts the per-frame made by RNN into a label sequence.

We can not expect the OCR model to be 100 % accurate. Still, we have achieved good results with the EAST model and Tesseract. Adding more filters for processing the image would help in improving the performance of the model.

### [Tesseract](https://nanonets.com/blog/ocr-with-tesseract/)

The capability of the Tesseract was mostly limited to structured text data. It would perform quite poorly in unstructured text with significant noise. Further development in tesseract has been sponsored by Google since 2006.

The latest stable version 4.1.0 is released on July 7, 2019. This version is significantly more accurate on the unstructured text as well.

Multiple OCRs in the industry perform well, but out of all of them, Tesseract OCR is the most reliable. Tesseract is unique because it comprises various functionalities you can leverage to customize it for multiple tasks.

## **How does Tesseract OCR Python work?**

The tesseract library uses a defined set of techniques for Optical Character Recognition processing. First, the image is converted into binary then a connected component analysis takes place, which stores the outlines of the components. These outlines are then gathered as blobs. Next, blobs are structured into text lines which are further broken down into words based on the spacing of characters.

Once the words are detected, the next phase is about recognition. Recognition is a two-step process. In the first step, each word is tried to be recognized, and satisfactory words are passed to an adaptive classifier. When everything is parsed once, a second pass is taken to recognize text that were not recognized in the first pass.

Ultimately, the algorithm makes a final pass to resolve fuzzy spaces and locate small-cap text

**TRAIN A TESSERACT OCR :**

# Steps involved:

* Gathering and naming image files.
* Generating Box files.
* Annotating Box files.
* Training Tesseract.

**Comparative models OCR**

OCR Prediction is not only dependent on the model and also on a lot of other factors like clarity, greyscale of the image, hyperparameter, weightage given, etc.

Tesseract is performing well for high-resolution images. Certain morphological operations such as dilation, erosion, OTSU binarization can help increase pytesseract performance.

EasyOCR is lightweight model which is giving a good performance for receipt or PDF conversion. It is giving more accurate results with organized texts like pdf files, receipts, bills.

Keras-OCR is image specific OCR tool. If text is inside the image and their fonts and colors are unorganized, Keras-ocr gives good results.

Though there are no hard and fast rules, we can consider the above three points while trying to choose for OCR tool.

FLASK API for OCR

<https://towardsdatascience.com/implementing-optical-character-recognition-ocr-using-pytesseract-5f42cf62ddcc>

# Generating the tuples of filenames

files = os.listdir(srcdir)

jpgs = [x for x in files if x.endswith('.jpg')]

boxes = [x for x in files if x.endswith('.box')]

trainfiles = list(zip(jpgs, boxes))

# generating TR files and unicode charecter extraction

unicharset = f"unicharset\_extractor --output\_unicharset ../../{destdir}/unicharset "

unicharset\_args = f""

errorfiles = []

for image, box in trainfiles:

    unicharset\_args += f"{box} "

    if os.path.isfile(f"{destdir}/{image[:-4]}.tr"):

        continue

    try:

        print(image)

        os.system(f"tesseract {srcdir}/{image} {destdir}/{image[:-4]} nobatch box.train")

    except:

        errorfiles.append((image, box))

os.chdir(srcdir)

subprocess.run(unicharset+unicharset\_args)

os.chdir('../../')

# Creating font properties file

with open(f"{destdir}/font\_properties", 'w') as f:

    f.write("credit\_card 0 0 0 1 0")

# # Getting all .tr files and training

output = '../../tesseract/trainoutput'

trfiles = [f for f in os.listdir(destdir) if f.endswith('.tr')]

os.chdir(destdir)

mftraining = f"mftraining -F font\_properties -U unicharset -O {output}/fra.unicharset -D {output}"

cntraining = f"cntraining -D {output}"

for file in trfiles:

    mftraining += f" {file}"

    cntraining += f" {file}"

subprocess.run(mftraining)

subprocess.run(cntraining)

os.chdir('../../')

# # Renaming training files and merging them

os.chdir(output[6:])

os.rename('inttemp', 'fra.inttemp')

os.rename('normproto', 'fra.normproto')

# os.rename('pffmtable', 'fra.pffmtable')

os.rename('shapetable', 'fra.shapetable')

os.system(f"combine\_tessdata fra.")

# Writing log file

if len(errorfiles) == 0:

    errorfiles.append(('no', 'Error'))

with open('tesseract/scripts/logs.txt', 'w') as f:

    f.write('\n'.join('%s %s' % x for x in errorfiles))