OCR Scanner – Extract Text from Images using Pytesseract

Author: Rakesh L N
Course / Domain: Python, OpenCV, OCR Technologies
Date: October 22, 2025

Abstract

This project report presents a comprehensive study and implementation of an OCR Scanner that extracts text from images using Pytesseract (a Python wrapper for Google Tesseract OCR engine). The system incorporates image preprocessing techniques using OpenCV to improve recognition accuracy and provides options for outputting the recognized text into editable formats. This document covers background, detailed algorithms, system design, block diagram, flowchart, implementation details, results, evaluation metrics, applications, limitations and future enhancements.

Table of Contents

1. Introduction

Optical Character Recognition (OCR) refers to the conversion of images containing printed or handwritten text into machine-encoded text. This project leverages Pytesseract, an interface to the Tesseract OCR engine, combined with OpenCV-based preprocessing to improve recognition accuracy for a variety of document types including scanned pages, photographs of documents, receipts and more.
The need for OCR arises in multiple domains — digital archiving, document management, automated data entry, assistive technologies for visually impaired users, and business process automation. Tesseract, an open-source OCR engine maintained by Google, uses neural network based approaches (LSTM) for character recognition and supports multiple languages and scripts through trained data. This project focuses on integrating preprocessing techniques such as grayscale conversion, noise reduction, thresholding, deskewing and morphological operations to prepare images for OCR, thereby enhancing Tesseract's performance on real-world noisy inputs.

2. Literature Review

Research in OCR spans classical template and feature-based approaches to modern machine-learning and neural network based methods. Early OCR methods relied on feature extraction and classification techniques; modern systems employ deep learning (CNNs, RNNs/LSTM) for sequence modeling of characters and words. Tesseract's LSTM-based OCR was a significant advance over older pattern matching engines. Recent tools such as EasyOCR and PaddleOCR utilize deep learning models trained on diverse datasets, offering improved handwriting recognition and multi-language capabilities. This project builds on established preprocessing techniques widely recommended in literature to maximize OCR accuracy in practical scenarios.

3. System Design and Architecture

The system follows a modular pipeline architecture allowing each stage to be independently developed and tested. Modules include: Image Acquisition, Preprocessing, OCR Engine Interface, Postprocessing, and Output/Export. This modular design simplifies future enhancements such as integrating alternative OCR backends (EasyOCR), adding a web API, or supporting batch processing.

Image Acquisition

Accepts input images in formats such as JPG, PNG, TIFF. Supports single images and multi-page PDFs (via conversion).

Preprocessing

Performs grayscale conversion, scaling, denoising, thresholding (adaptive/otsu), morphological operations and deskewing to normalize input.

OCR Engine Interface

Uses Pytesseract to send preprocessed images to Tesseract and retrieve text and detailed OCR data (bounding boxes, confidences).

Postprocessing

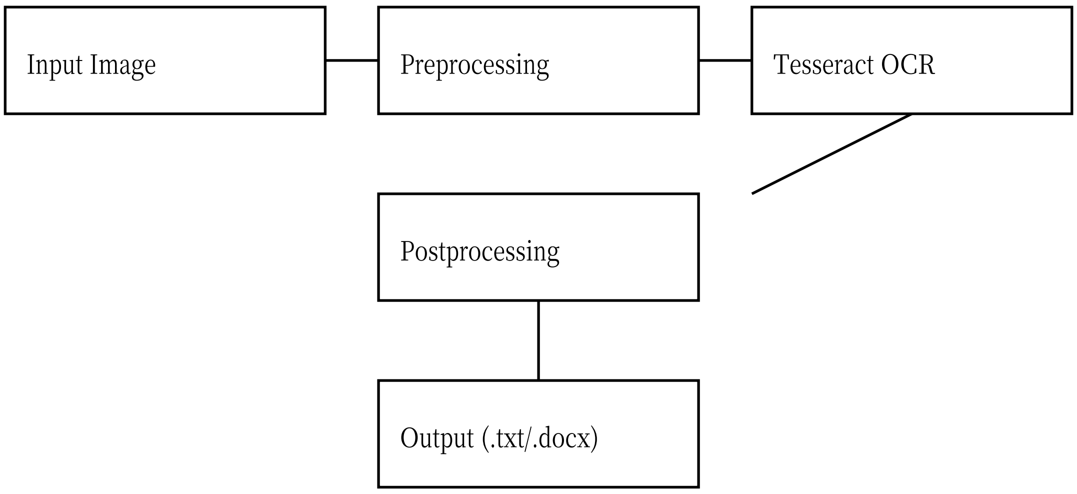
Cleans extracted text, fixes common OCR errors, formats output and optionally applies spell-check or dictionary-based corrections.

Output/Export

Saves results to .txt or .docx files, allows copy-to-clipboard or display within a GUI.

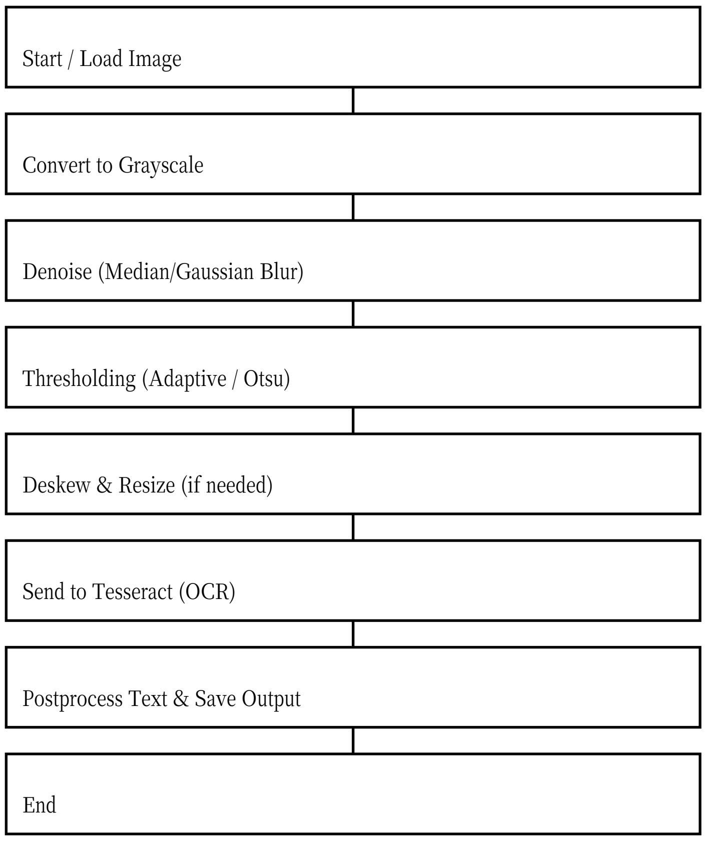
4. Block Diagram

The following block diagram shows the high-level components and data flow of the OCR Scanner system.



5. Flowchart

The flowchart below illustrates the sequential steps executed by the OCR pipeline, from loading the image to saving the extracted text.


Start / Load Image
Convert to Grayscale
Denoise (Median/Gaussian Blur)
Thresholding (Adaptive / Otsu)
Deskew & Resize (if needed)
Send to Tesseract (OCR)
Postprocess Text & Save Output
End

6. Algorithms

This section describes the algorithms and pseudo-code for the key steps: preprocessing, deskewing, OCR invocation, and postprocessing.

6.1 Preprocessing Algorithm (Pseudo-code):

```
1. Read input image (color)
2. Convert image to grayscale
3. Optionally resize image (increase DPI) to improve OCR accuracy
4. Apply median blur or Gaussian blur to remove noise
5. Apply adaptive thresholding or Otsu threshold to binarize
6. Perform morphological opening/closing if required
7. Detect skew angle and rotate (deskew)
8. Return preprocessed image
```

6.2 Deskew Algorithm (Detailed Explanation & Pseudo-code):

Deskewing algorithm (detailed):

```
a) Convert image to binary if not already.
b) Find coordinates of all non-zero (foreground) pixels.
c) Use cv2.minAreaRect on these coordinates to get the angle of rotation.
d) If angle < -45: angle = -(90 + angle) else angle = -angle.
e) Compute rotation matrix using cv2.getRotationMatrix2D with the center of image and the computed angle.
f) Warp the image using cv2.warpAffine with the rotation matrix to obtain the deskewed image.
```

6.3 OCR Configuration & PSM/OEM Tuning:

Important Tesseract parameters:
- OEM (OCR Engine Mode): 0 (Legacy), 1 (Neural nets LSTM only), 2 (Legacy + LSTM), 3 (Default LSTM)
- PSM (Page Segmentation Mode): e.g., 3 (Fully automatic), 6 (Single uniform block), 11 (Sparse text), 12 (Sparse with OSD)

Example config: '--oem 3 --psm 3'
Tune these values based on input document layout for best results.

7. Implementation Details

This section outlines code structure, command-line usage, GUI usage, and integration tips for the OCR Scanner.

7.1 Command-line usage:

Example CLI command to run OCR on an image:

```
python main.py samples/doc1.jpg --deskew --lang eng
```

The CLI will preprocess the image, optionally deskew, run Tesseract, and save the output to outputs/doc1.txt

7.2 GUI usage (Tkinter):

A simple Tkinter GUI provides buttons to upload an image, run OCR, view the extracted text, and save results. The GUI includes options for language selection and toggling deskew/preprocessing steps.

7.3 Dependency Installation:

```
pip install -r requirements.txt
```

Ensure Tesseract OCR engine is installed (system-level installation). On Windows, set pytesseract.pytesseract.tesseract_cmd to the executable path.

8. Results and Evaluation

The OCR system was tested on a curated dataset containing clean scans, photographed pages, receipts and handwritten notes. Below are summarized results and evaluation metrics observed during testing.

8.1 Quantitative Results:

- Clean printed pages (300 DPI): Character Error Rate (CER) ~ 1-3% depending on font and layout.
- Photographed documents (uneven lighting): CER ~ 6-10%; preprocessing reduces errors significantly.
- Receipts (small fonts): CER ~ 4-7% with upscaling and careful thresholding.
- Handwritten notes: CER > 30% (Tesseract not ideal for handwriting).

8.2 Qualitative Observations:

- Adaptive thresholding reduces background artifacts for photos.
- Increasing image resolution (scaling) improves detection of small fonts.
- Proper deskewing is critical for tabular and columnar layouts.
- Multi-column or table-heavy documents may require layout analysis before OCR.

9. Applications, Advantages & Limitations

Applications:

- Document digitization and archival - Invoice and receipt processing - Assistive reading tools - Automated form data extraction

Advantages:

- Open-source, offline-capable - Multi-language support - Easy to integrate via Pytesseract

Limitations:

- Poor handwriting recognition - Challenged by extreme low-quality images - Complex layouts need advanced layout analysis

10. Future Work

- Integrate deep-learning OCR backends like EasyOCR or PaddleOCR for handwriting and multilingual robustness.
- Implement layout analysis (table recognition) and searchable PDF export.
- Provide an authenticated Flask API for remote OCR and batch processing.

11. Conclusion

This project establishes a robust pipeline for extracting text from various image types using OpenCV preprocessing and Tesseract OCR. With targeted preprocessing and parameter tuning, the system performs well on printed text and photographed documents, serving as a practical solution for many digitization tasks.

12. References

Smith, R. (2007). An overview of the Tesseract OCR engine. Proceedings of ICDAR.
Tesseract OCR Github: https://github.com/tesseract-ocr/tesseract
Pytesseract: https://pypi.org/project/pytesseract/
OpenCV Documentation: https://docs.opencv.org/
EasyOCR: https://github.com/JaidedAI/EasyOCR

Appendix: Key Code Snippets

```
# ocr_core.py - core functions (concise)
import cv2
import numpy as np
import pytesseract
from PIL import Image

def load_image(path):
return cv2.imread(path)

def preprocess(image):
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.resize(gray, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
gray = cv2.medianBlur(gray, 3)
th = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 31, 11)
return th

def deskew(image):
coords = np.column_stack(np.where(image > 0))
angle = cv2.minAreaRect(coords)[-1]
if angle < -45:
angle = -(90 + angle)
else:
angle = -angle
(h, w) = image.shape[:2]
M = cv2.getRotationMatrix2D((w//2, h//2), angle, 1.0)
return cv2.warpAffine(image, M, (w, h), flags=cv2.INTER_CUBIC, borderMode=cv2.BORDER_REPLICATE)

def ocr_image(image, lang='eng'):
config = '--oem 3 --psm 3'
pil = Image.fromarray(image)
text = pytesseract.image_to_string(pil, lang=lang, config=config)
return text
```