**Python Optical Character Recognition Review Notes**

#### Introduction

#### What is Optical Character Recognition?

#### Uses of OCR

#### What we’ll use

#### Setup

#### Implementation

##### Phase 1: OCR Script  
##### Phase 2: Flask Web Interface

#### What we haven’t done

#### Conclusion

->

### Introduction

### What is Optical Character Recognition?

### Uses of OCR

### What We’ll Use?

### Setup

### Implementation

#### OCR Script

#### Flask Web Interface

### What We Haven’t Done

### Conclusion

**Comment: All top level headings should use the ###(<h3>) syntax, and all subsequent headings should use ####(<h4>), ##### (<h5>) and ######(<h6>) respectively.**

**The #(<h1>) and ##(<h2>) headings are reserved on our website, where the # heading is the title of the article.**

**This is something I probably wasn’t clear about when I first laid out our norms so I went ahead and changed the headings to different sizes as seen above.**

**Also, we usually try to optimize the heading titles to be concise (ex: We don’t need “Phase 1” and “Phase 2”).**

**Also, in “What We’ll Use?” and “What We Haven’t Done” the titles weren’t capitalized according to the conventions. If you’re unsure which words to capitalize, visit capitalizemytitle.com, it’s a great service that does the job for you!**

### Introduction

In this post, we will delve into the depth of Optical Character recognition and its application areas. We will also build a simple script that will help us detect characters from images and expose this through a Flask application for a more convenient interaction medium.

->

### Introduction

Humans can understand the contents of an image simply by looking. We perceive the text on the image as text and can read it.

Computers don’t work the same way. They need something more concrete, organized in a way they can understand.

This is where \*Optical Character Recognition\* (OCR) kicks in. Whether it’s recognition of car plates from a camera, or hand-written documents that should be converted into a digital copy, this technique is very useful. While it’s not always perfect, it’s very convenient and makes it a lot easier and faster for some people to do their jobs.

In this article, we will delve into the depth of Optical Character Recognition and its application areas. We will also build a simple script in Python that will help us detect characters from images and expose this through a Flask application for a more convenient interaction medium.

**Comment: When writing an introduction, I avoid simply stating “In this article, we’ll do x and y”. This is mostly due to the fact that the introduction section is the section that’ll pop up as a preview on google search results to engage a potential reader.**

**This is the perfect place to include a few keywords like “Optical Character Recognition”, “OCR”, “Python”, “Flask” etc.**

**These are all possible search terms someone might combine to search for this article so this is also a good SEO move.**

I went for it because it is completely open-source and being developed and maintained by the giant that is Google.

->

This library is completely open-source and is being developed and maintained by the giant that is Google.

**Comment: When possible, avoid using “me” and “you” language. While it’s good to keep article conversational, sometimes it can look unprofessional.**

**Prefer the “we” language or even passive voice.**

In case you’ll not be using Pipenv, you can always use the [Pip](https://pip.pypa.io/en/stable/installing/) and [Virtual Environment](https://docs.python.org/3/tutorial/venv.html) approach. I have attached useful resources to help you get started with Pip and Virtual environments.

**Comment: I’m assuming that you’re referring to the two links you’ve added beforehand. This sentence sounds a bit misleading as we haven’t really attached anything and some readers might get confused by the sentence.**

**Instead, we could use something along the lines of:**In case you’ll not be using Pipenv, you can always use the Pip and Virtual Environment approach. Follow the official documentation to help you get started with [Pip](https://pip.pypa.io/en/stable/installing/) and [Virtual Environment](https://docs.python.org/3/tutorial/venv.html):

**Simply by changing the position of the links in the text, we can ommit the confusing sentence where people might expect an attachement.**

With the setup complete, we can now create a simple function that takes an image and returns the text detected in the image - This will be the core of our project.

Create a file called ocr\_core.py and add the following code. I’ll comment and explain the code.

->

With the setup complete, we can now create a simple function that takes an image and returns the text detected in the image - This will be the core of our project:

**Comment: The second paragraph is redundant. Instead of stating the name of the file in the sentence, we can simply include the part of the `.py` file that shows the name. Also, try to avoid preambles like “as follows”, “the following code”, “like so” etc. These can usually be omitted for the sake of brevity and clarity.**

**In this case, we can omit the whole sentence and simply replace it with a “:”.**

**Also, when including filenames such as this one, format it as a snippet. (ocr\_core.py -> `ocr\_core.py`)**

```

…some code

```

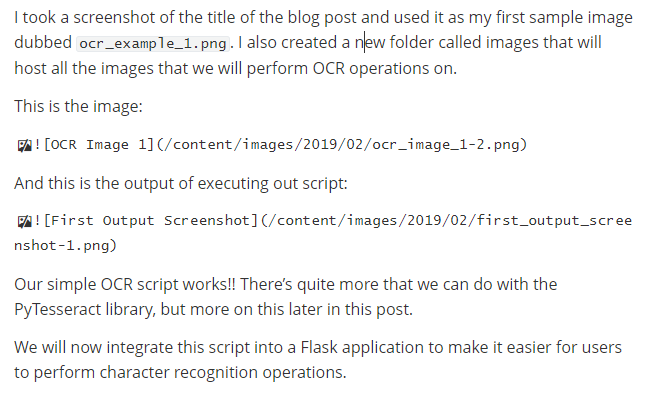
->

```python

…some code

```

**Comment: Markdown covers language-specific formatting. By including the name of the language of the code, the language keywords get highlighted.**



**Comment: Again, try to avoid direct “I” language.**

**Instead, something along the lines of:**

“Let’s see how the script fares with a simple image containing some text:

![]()

And upon running the piece of code, we’re greeted with this:

![]()

Our simple OCR script works! Obviously, this was somewhat easy since this is digital text, perfect and precise, unlike handwriting. There’s a lot more that we can do with the PyTesseract library, but more on this later in the post.

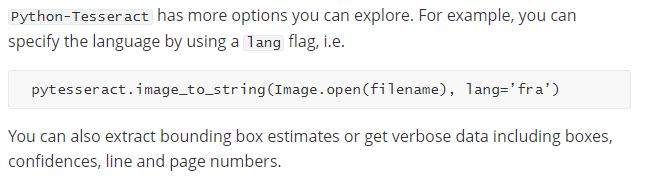
Let’s integrate this script into a Flask application first, to make it easier to upload images and perform character recognition operations.”

Create an `app.py` file with the following contents:

->

Now, let’s define a basic route:

**Comment: Again, the same as before.**



**Comment: I’m interested to read more about PyTesseract, especially in this blog post. Could we expand this section a bit and show some simple examples?**