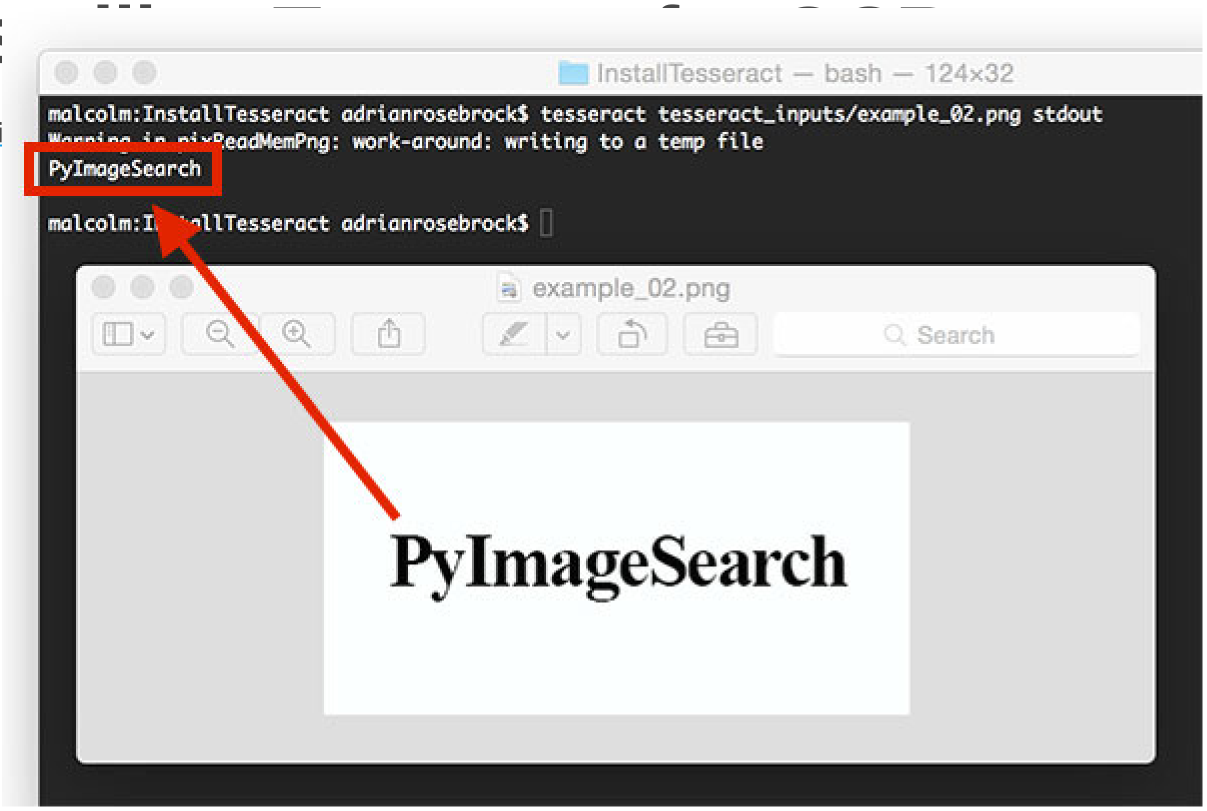
Ins

by **Adri**



```
malcolm:InstallTesseract adrianrosebrock$ tesseract tesseract_inputs/example_02.png stdout
Warning in pixReadMemPng: work-around: writing to a temp file
PyImageSearch

malcolm:InstallTesseract adrianrosebrock$
```

Today's blog post is part one in a two part series on installing and using the **Tesseract library** for Optical Character Recognition (OCR).

OCR is the automatic process of converting typed, handwritten, or printed text to machine-encoded text that we can access and manipulate via a string variable.

Part one of this series will focus on installing and configuring Tesseract on your machine, followed by utilizing the

`tesseract` command to apply OCR to input images.

In next week's blog post we'll discover how to use the Python "bindings" to the Tesseract library to call Tesseract *directly* from your Python script.

To learn more about Tesseract and how it can be used for OCR, *just keep reading*.

Looking for the source code to this post?

JUMP RIGHT TO THE DOWNLOADS SECTION →

Installing Tesseract for OCR

Tesseract, originally developed by Hewlett Packard in the 1980s, was open-sourced in 2005. Later, in 2006, Google adopted the project and has been a sponsor ever since.

The Tesseract software works with many natural languages from English (initially) to Punjabi to Yiddish. Since the updates in 2015, it now supports over *100 written languages* and has code in place so that it can easily be trained on other languages as well.

Originally a C program, it was ported to C++ in 1998. The software is headless and can be executed via the command line. It does not come with a GUI but there are several other software packages that wrap around Tesseract to provide a GUI interface.

To read more about Tesseract visit the **project page** and read the **Wikipedia article**.

In this blog post we will:

- Install Tesseract on our systems.
- Validate that the Tesseract install is working correctly.
- Try Tesseract OCR on some sample input images.

After going through this tutorial you will have the knowledge to run Tesseract on your own images.

Step #1: Install Tesseract

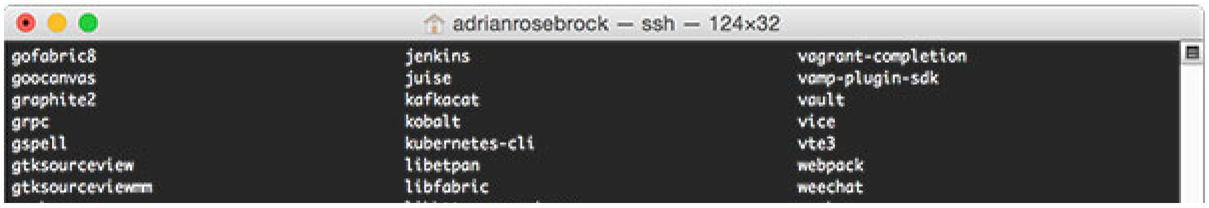
In order to use the Tesseract library, we first need to install it on our system.

For ***macOS users***, we'll be using **Homebrew** to install Tesseract:

→ **Click here to download the code**

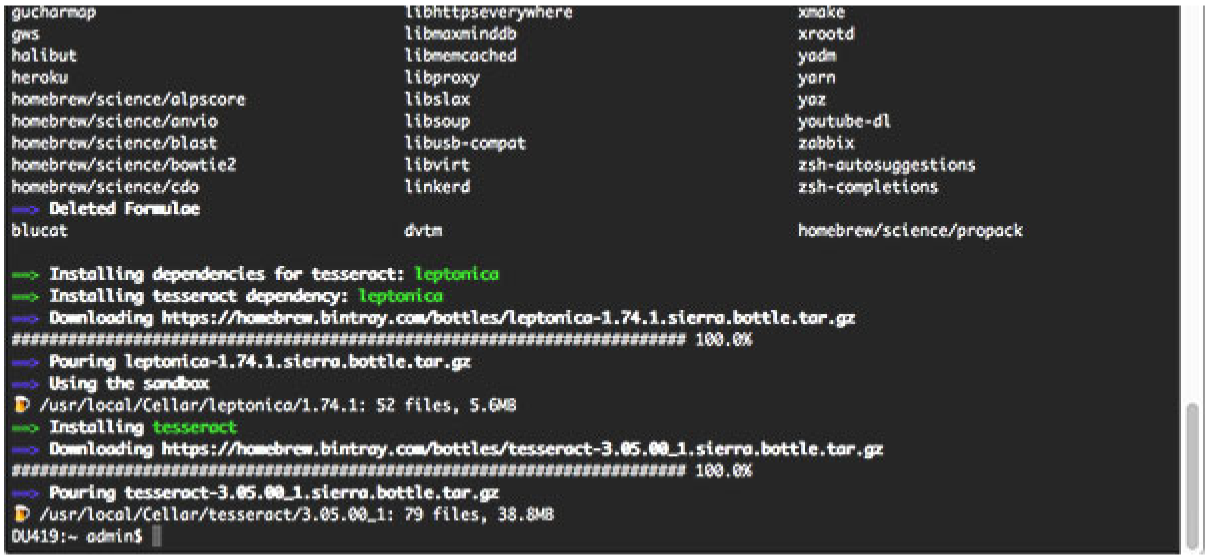
Installing Tesseract for OCR

```
$ brew install tesseract
```



adrianrosebrock — ssh — 124×32

```
gofabric8                      jenkins                        vagrant-completion
goocanvas                      juise                          vamp-plugin-sdk
graphite2                      kafkacat                       vault
grpc                           kobalt                         vice
gspell                         kubernetes-cli                 vte3
gtksourceview                  libetpan                       webpack
gtksourceviewmm                libfabric                      weechat
```



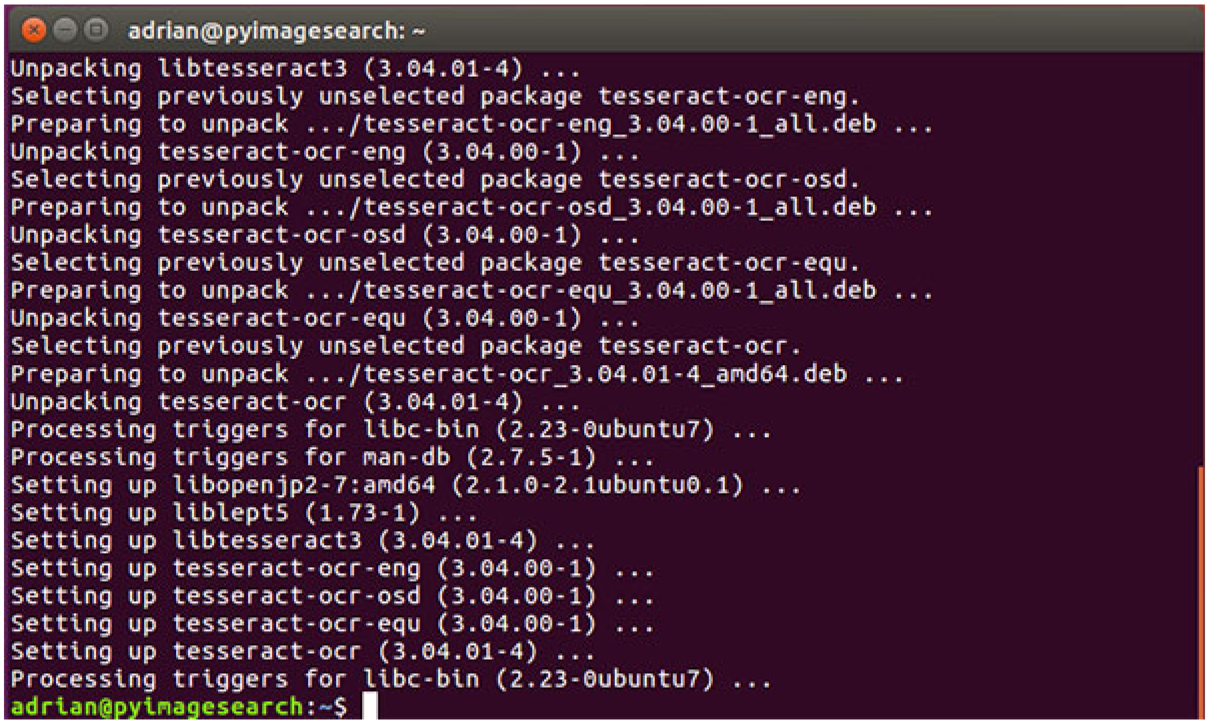
```
gucharmap                                libhttpseverywhere                      xmake
gws                                      libmaxminddb                            xrootd
halibut                                  libmemcached                            yadm
heroku                                   libproxy                                yarn
homebrew/science/alpscore                libslax                                 yaz
homebrew/science/anvio                   libsoup                                 youtube-dl
homebrew/science/blast                   libusb-compat                           zabbix
homebrew/science/bowtie2                 libvirt                                 zsh-autosuggestions
homebrew/science/cdo                     linkerd                                 zsh-completions
==> Deleted Formulae
blucat                                   dvtm                                    homebrew/science/propack

==> Installing dependencies for tesseract: leptonica
==> Installing tesseract dependency: leptonica
==> Downloading https://homebrew.bintray.com/bottles/leptonica-1.74.1.sierra.bottle.tar.gz
######################################################################## 100.0%
==> Pouring leptonica-1.74.1.sierra.bottle.tar.gz
==> Using the sandbox
🍺 /usr/local/Cellar/leptonica/1.74.1: 52 files, 5.6MB
==> Installing tesseract
==> Downloading https://homebrew.bintray.com/bottles/tesseract-3.05.00_1.sierra.bottle.tar.gz
######################################################################## 100.0%
==> Pouring tesseract-3.05.00_1.sierra.bottle.tar.gz
🍺 /usr/local/Cellar/tesseract/3.05.00_1: 79 files, 38.8MB
DU419:~ admin$
```

Figure 1: Installing Tesseract OCR on macOS.

If you're using the ***Ubuntu operating system***, simply use `apt-get` to install Tesseract OCR:

→ Click here to download the code

```
Installing Tesseract for OCR
$ sudo apt-get install tesseract-ocr
```



adrian@pyimagesearch: ~

```
Unpacking libtesseract3 (3.04.01-4) ...
Selecting previously unselected package tesseract-ocr-eng.
Preparing to unpack .../tesseract-ocr-eng_3.04.00-1_all.deb ...
Unpacking tesseract-ocr-eng (3.04.00-1) ...
Selecting previously unselected package tesseract-ocr-osd.
Preparing to unpack .../tesseract-ocr-osd_3.04.00-1_all.deb ...
Unpacking tesseract-ocr-osd (3.04.00-1) ...
Selecting previously unselected package tesseract-ocr-equ.
Preparing to unpack .../tesseract-ocr-equ_3.04.00-1_all.deb ...
Unpacking tesseract-ocr-equ (3.04.00-1) ...
Selecting previously unselected package tesseract-ocr.
Preparing to unpack .../tesseract-ocr_3.04.01-4_amd64.deb ...
Unpacking tesseract-ocr (3.04.01-4) ...
Processing triggers for libc-bin (2.23-0ubuntu7) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up libopenjp2-7:amd64 (2.1.0-2.1ubuntu0.1) ...
Setting up liblept5 (1.73-1) ...
Setting up libtesseract3 (3.04.01-4) ...
Setting up tesseract-ocr-eng (3.04.00-1) ...
Setting up tesseract-ocr-osd (3.04.00-1) ...
Setting up tesseract-ocr-equ (3.04.00-1) ...
Setting up tesseract-ocr (3.04.01-4) ...
Processing triggers for libc-bin (2.23-0ubuntu7) ...
adrian@pyimagesearch:~$
```

Figure 2: Installing Tesseract OCR on Ubuntu.

For ***Windows***, please consult **Tesseract documentation** as PyImageSearch does not support or recommend Windows for computer vision development.

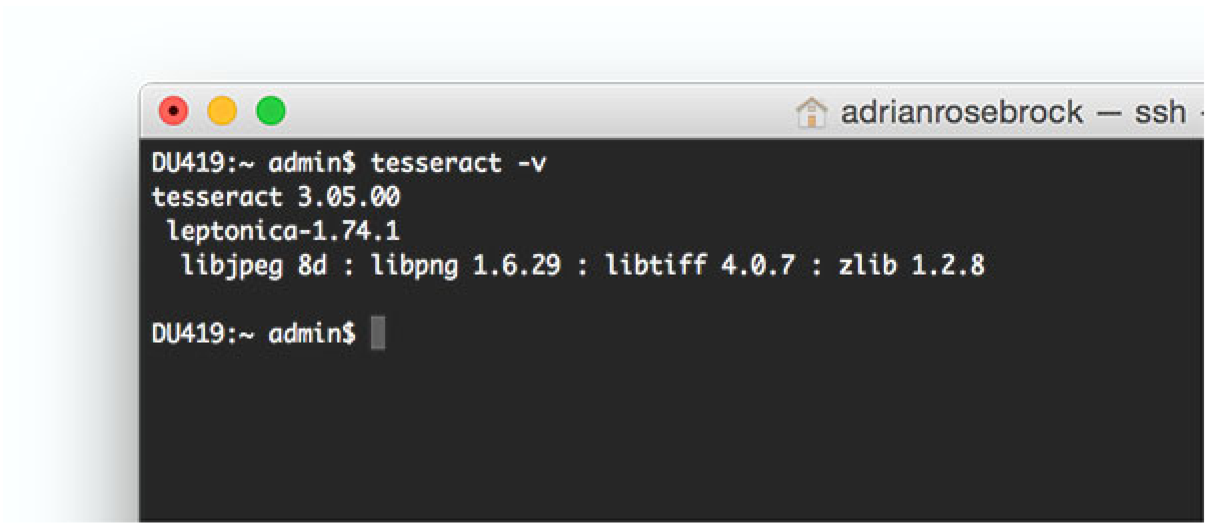
Step #2: Validate that Tesseract has been installed

To validate that Tesseract has been successfully installed on your machine, execute the following command:

→ Click here to download the code

Installing Tesseract for OCR

```
$ tesseract -v
tesseract 3.05.00
 leptonica-1.74.1
  libjpeg 8d : libpng 1.6.29 : libtiff 4.0.7 : zlib 1.2.8
```



```
DU419:~ admin$ tesseract -v
tesseract 3.05.00
 leptonica-1.74.1
  libjpeg 8d : libpng 1.6.29 : libtiff 4.0.7 : zlib 1.2.8

DU419:~ admin$
```

Figure 3: Validating that Tesseract has been successfully installed on my machine.

You should see the Tesseract version printed to your screen, along with a list of image file format libraries Tesseract is compatible with.

If you instead get the error:

→ Click here to download the code

Installing Tesseract for OCR

```
-bash: tesseract: command not found
```

Then Tesseract was not properly installed on your system. Go back to ***Step #1*** and check for errors. Additionally, you may need to update your

`PATH` variable (for advanced users only).

Step #3: Test out Tesseract OCR

For Tesseract OCR to obtain reasonable results, you'll want to supply images that are ***cleanly pre-processed***.

When utilizing Tesseract, I recommend:

- Using as an input image with as high resolution and DPI as possible.
- Applying thresholding to segment the text from the background.
- Ensuring the foreground is as *clearly* segmented from the background as possible (i.e., no pixelations or character deformations).
- Applying **text skew correction** to the input image to ensure the text is properly aligned.

Deviations from these recommendations can lead to incorrect OCR results as we'll find out later in this tutorial.

Now, let's apply OCR to the following image:

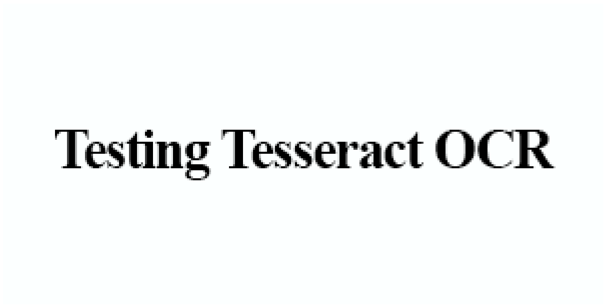

Testing Tesseract OCR

Figure 4: An example image we are going to apply OCR to using Tesseract.

Simply enter the following command in your terminal:

→ **Click here to download the code**

Installing Tesseract for OCR

```
$ tesseract tesseract_inputs/example_01.png stdout
Warning in pixReadMemPng: work-around: writing to a temp file
Testing Tesseract OCR
```

Correct! Tesseract correctly identified, *"Testing Tesseract OCR"*, and printed it in the terminal.

Next, let's try this image:

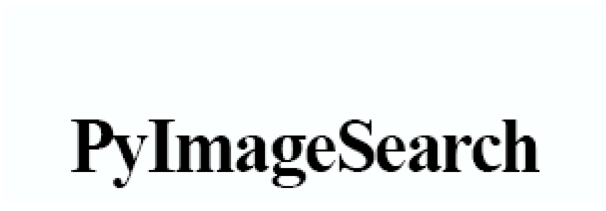
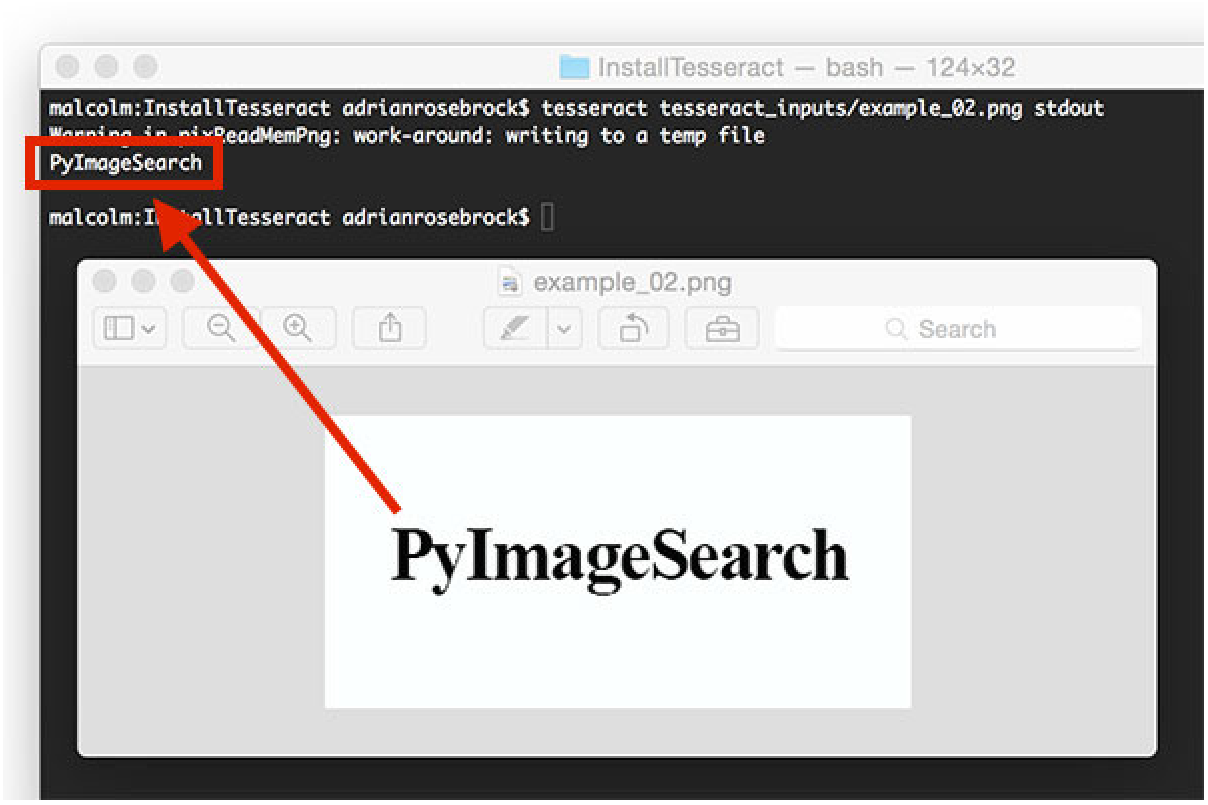

PyImageSearch

Figure 5: A second example image to apply Optical Character Recognition to using Tesseract.

Enter the following in your terminal, noting the changed input filename:

→ Click here to download the code

```
Installing Tesseract for OCR
$ tesseract tesseract_inputs/example_02.png stdout
Warning in pixReadMemPng: work-around: writing to a temp file
PyImageSearch
```

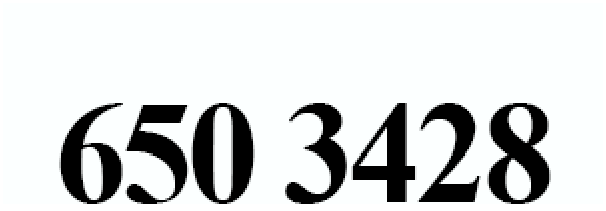


```
malcolm:InstallTesseract adrianrosebrock$ tesseract tesseract_inputs/example_02.png stdout
Warning in pixReadMemPng: work-around: writing to a temp file
PyImageSearch

malcolm:InstallTesseract adrianrosebrock$
```

Figure 6: Tesseract is able to correctly OCR our image.

Success! Tesseract correctly identified the text, *"PyImageSearch"*, in the image.

Now, let's try OCR'ing *digits* as opposed to *alphabetic characters*:



650 3428

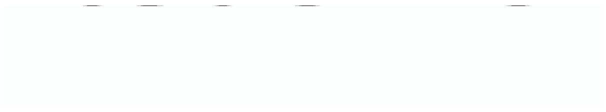


Figure 7: Using Tesseract to OCR digits in images.

This example uses the command line `digits` switch to *only* report digits:

→ **Click here to download the code**

Installing Tesseract for OCR

```
$ tesseract tesseract_inputs/example_03.png stdout digits
Warning in pixReadMemPng: work-around: writing to a temp file
650 3428
```

Once again, Tesseract correctly identified our string of characters (in this case digits only).

In each of these three situations Tesseract was able to correctly OCR all of our images — and you may even be thinking that Tesseract is the right tool for *all* OCR uses cases.

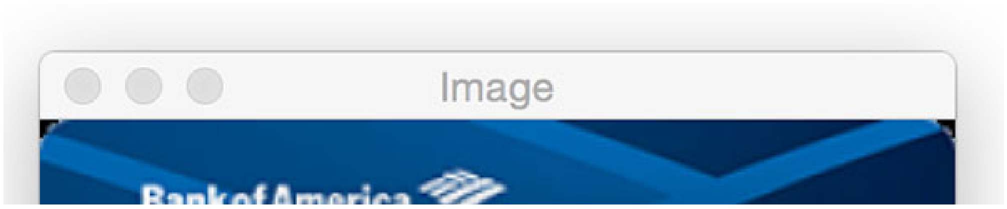
However, as we’ll find out in the next section, Tesseract has a number of limitations.

Limitations of Tesseract for OCR

A few weeks ago I was working on a project to recognize the 16-digit numbers on credit cards.

I was easily able to write Python code to localize each of the four groups of 4-digits.

Here is an example 4-digit region of interest:



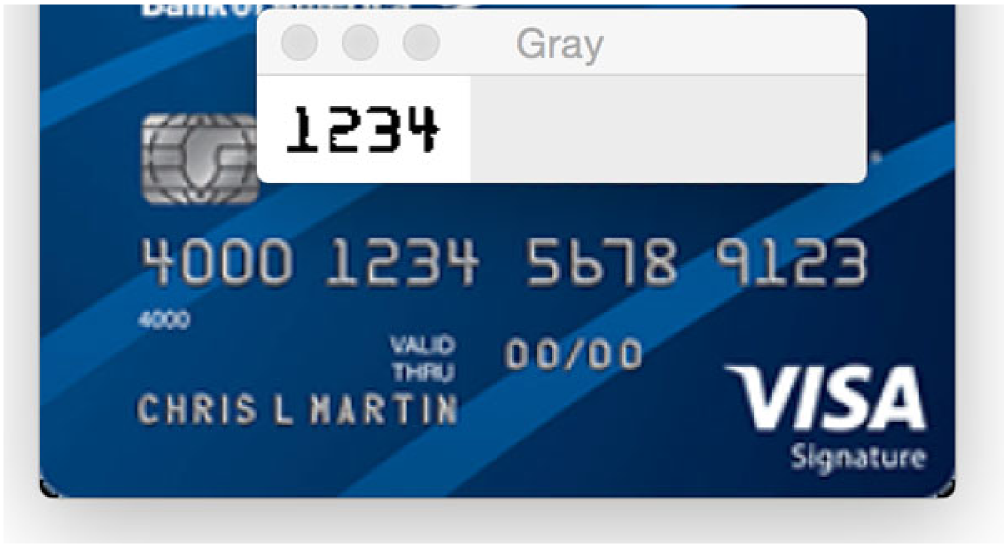


Figure 8: Localizing a 4-digit grouping of characters on a credit card.

However, when I tried to apply Tesseract to the following image, the results were dissatisfying:

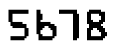


Figure 9: Trying to apply Tesseract to "noisy" images.

→ **Click here to download the code**

Installing Tesseract for OCR

```
$ tesseract tesseract_inputs/example_04.png stdout digits
Warning in pixReadMemPng: work-around: writing to a temp file
5513
```

Notice how Tesseract reported

`5513` , but the image clearly shows `5678` .

Unfortunately, this is a great example of a limitation of Tesseract. While we have segmented the foreground text from background, the pixelated nature of the text "confuses" Tesseract. It's also likely that Tesseract was not trained on a credit card-like font.

Tesseract is best suited when building document processing pipelines where images are scanned in, pre-processed, and then Optical Character Recognition needs to be applied.

We should note that Tesseract is *not* an off-the-shelf solution to OCR that will work in all (or even most) image processing and computer vision applications.

In order to accomplish that, you'll need to apply feature extraction techniques, machine learning, and deep learning.

A great example of applying feature extraction and machine learning to build a handwriting recognition system can be found inside my book, ***Practical Python and OpenCV***.

Summary

Today we learned how to install and configure Tesseract on our machines, the first part in a two part series on using Tesseract for OCR. We then used the

`tesseract` binary to apply OCR to input images.

However, we found out that unless our images are cleanly segmented Tesseract will give poor results. In the case of "noisy" input images, we'll likely obtain better accuracy by training a custom machine learning model to recognize characters in our *specific* use case.

Tesseract is *best suited* for situations with high resolution inputs where the foreground text is ***cleanly segmented*** from the background.

Next week we'll learn how to access Tesseract via Python code, so stay tuned.