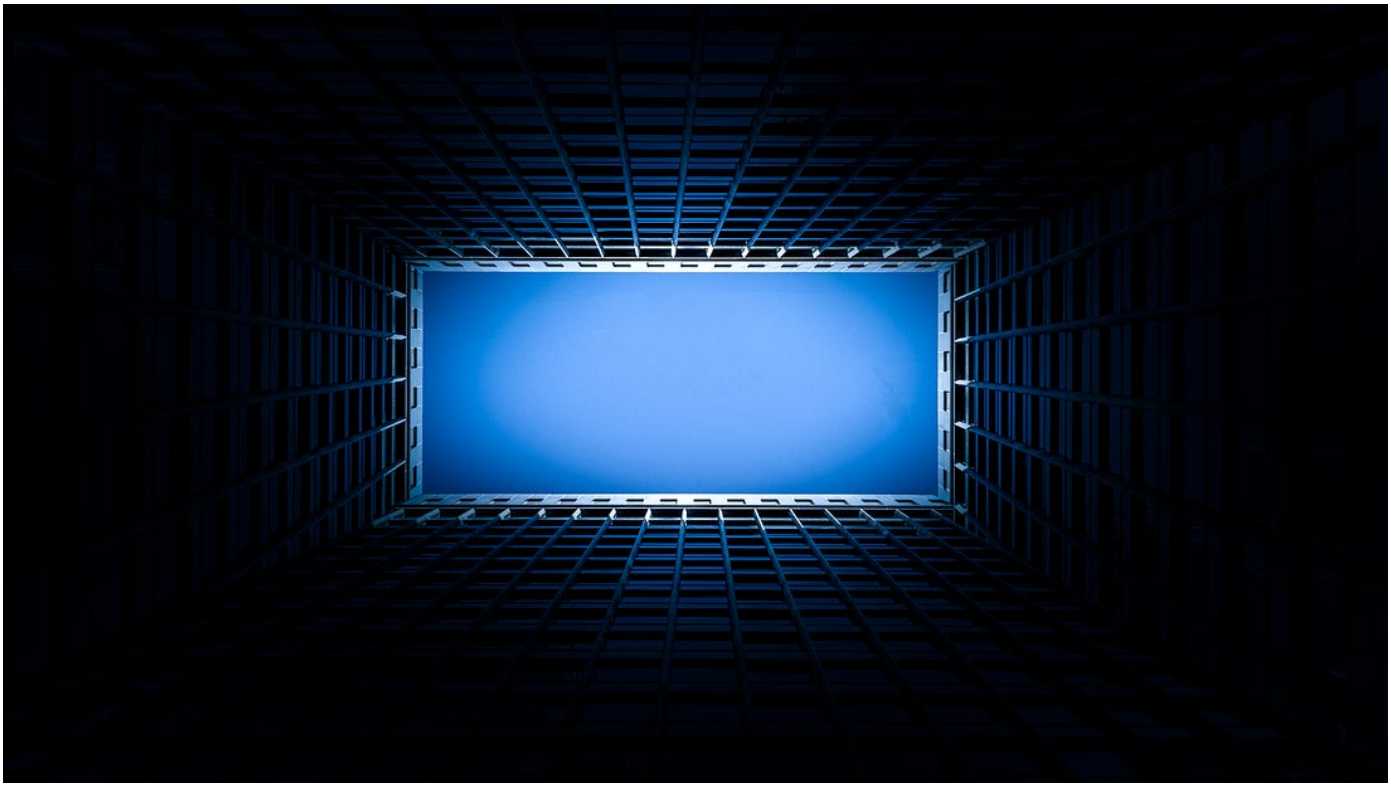


Photo by Lysander Yuen on Unsplash

✨ Member-only story

How to Perform Image Segmentation with Thresholding Using OpenCV

Simple, Otsu, and Adaptive Thresholding Implementation with Examples

Rashida Nasrin Sucky · Follow
Published in Towards Data Science
6 min read · Jan 12

Listen Share More

One of the most commonly used segmentation techniques is thresholding. It is widely used in computer vision and is very useful to separate the object that we are interested

in from the background. This article will focus on the thresholding techniques in OpenCV.

There are three different types of thresholding. We will know how they work at a high level and basically focus on how to use them in OpenCV with examples.

What is Thresholding

Thresholding is the process of binarization of an image. If you get confused between grayscale images and binary images as I used to in my beginning, in grayscale images, there might be an array of shades. But in the binary images, the pixel values are either 0 or 255.

Simple Thresholding

This is the most basic type of thresholding and is harder to use in a lot of practical applications. But still, we need to start with learning this one to understand how thresholding actually works. We will see how simple thresholding can work well.

Simple thresholding can work well in very controlled lighting conditions where there is high contrast between the foreground (the object we are interested in, in the image) and background. The hardest part about simple thresholding is, we need to manually supply the threshold values which may take a lot of trials. In that way, it may become very time-consuming. Let's work on an example.

We will use the following image for this tutorial. Please feel free to save this image and follow along.


60%
60% hi
39% low
INDOOR HUMIDITY
73°F
73°F hi
68°F low
TEMPERATURE

Photo by Lucas Hoang on Unsplash

First, we need to import the necessary libraries.

```
import cv2
import matplotlib.pyplot as plt
```

Reading the image using OpenCV as an array:

```
image = cv2.imread("meter.jpg")
```

As thresholding is binarization, we start with a grayscale image. Also, using a Gaussian blur on the grayscale image will help get rid of some high-frequency edges and noises.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (7, 7), 0)
```

Here is the simple thresholding:

```
(T, thresh) = cv2.threshold(blurred, 45, 255, cv2.THRESH_BINARY)
```

The first parameter is the blurred image,

45 is the threshold value that says if the pixel value is greater than 45 it will become 255, otherwise 0. So, the third parameter is 255. If you choose a value different than 255, such as 200, the pixels greater than 45 will become 200.

Please feel free to try with a different value and see what happens.

> I decided to use a threshold value of 45 after a lot of trials.

So, it took some time. Here is what ‘thresh’ looks like:

```
cv2.imwrite('a.png', thresh)
```

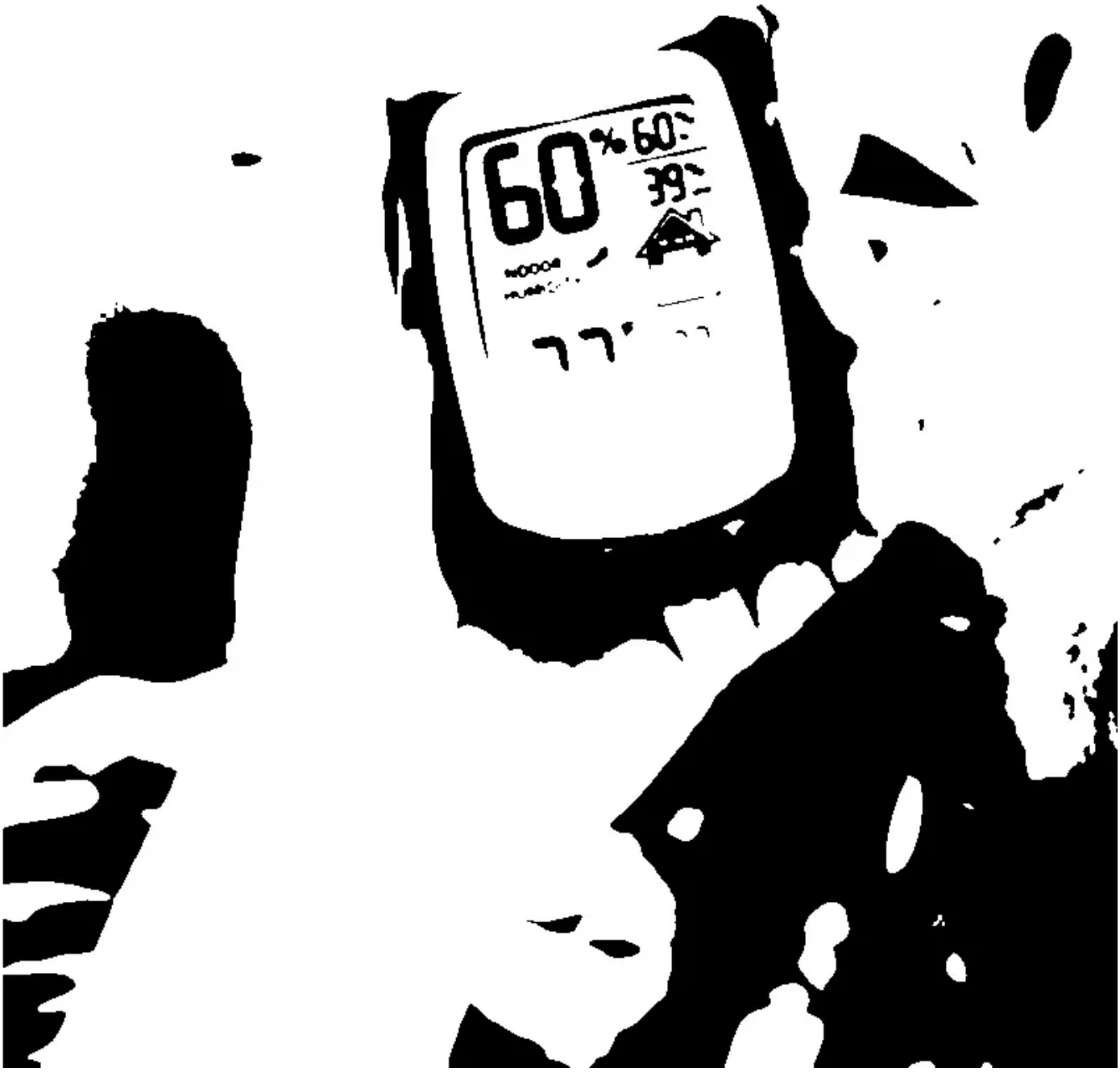


Image by Author

We can use cv2.THRESH_BINARY_INV as well which will do the otherwise. Let's see how it looks if the same threshold value 45 is used with cv2.THRESH_BINARY_INV:

```
(T, threshInv) = cv2.threshold(blurred, 45, 255, cv2.THRESH_BINARY_INV)
```

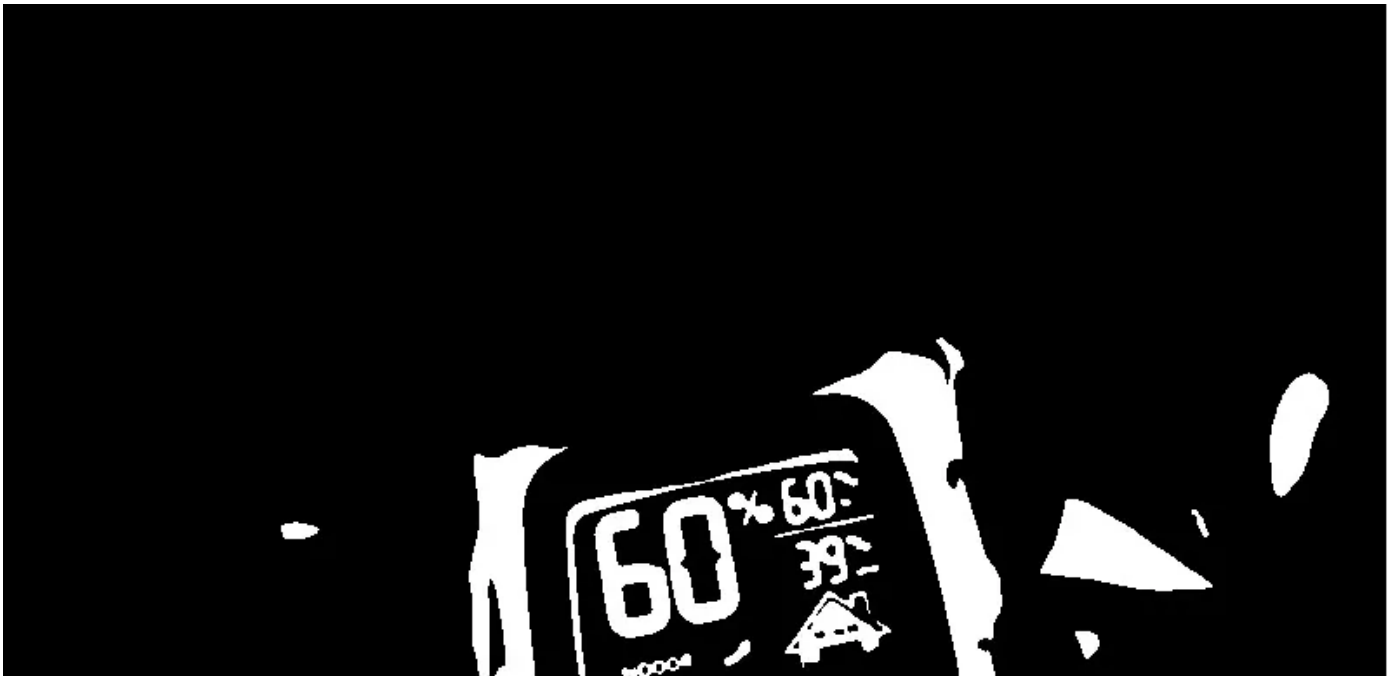


Image by Author

In this case, when the pixel is greater than the threshold value of 45, it becomes 0 and 255 otherwise.

As I mentioned before I had to go through quite a few trials to find that threshold value of 45.

> But if the lighting conditions is different for the same picture this threshold value wouldn't work.

That's why in a real-world application where we may not have prior knowledge about the lighting condition, it is hard to work this way. **We need to find an optimum threshold value that can scale.**

Otsu Thresholding

In Otsu's thresholding, OpenCV finds an optimum threshold value for you. It takes the grayscale image and computes the optimum threshold value, T.

Here is a high-level overview of how Otsu's method chooses the threshold value. If a distribution is made with all the pixel values of a grayscale image, Otsu's method finds a threshold that optimally separates the peaks of the distribution.

```
(T, threshOtsu) = cv2.threshold(blurred, 0, 255,
                cv2.THRESH_BINARY | cv2.THRESH_OTSU)
```

Look, here we used 0 as the threshold value. Simply because Ostu's will find the optimal threshold value for us.

Let's see what is the optimal threshold value:

```
T
```

Output:

```
112.0
```

Here is the image of the ‘threshOtsu’:

```
cv2.imwrite("au.jpg", threshOtsu)
```

Image by Author

We wanted to focus on the meter reading part, right? So, how can we do that from this image? Just masking the original image using bitwise_and operation.

```
masked1 = cv2.bitwise_and(image, image, mask=threshInvOtsu)
cv2.imwrite('amasked.jpg', masked1)
```

Here is the image:

Image by Author

Lots of noises in the background are smoothed out.

The problem with both simple and Otsu's thresholding methods is they use a single thresholding value all over the image which may not be a good idea. The thresholding value that works well for one area of the image may not be good for the other areas of the image.

Adaptive Thresholding

The adaptive thresholding method considers a small set of pixels and computes a threshold for each small region of the image. We will explain some more after this example:

```
thresh = cv2.adaptiveThreshold(blurred, 255,
                               cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                              cv2.THRESH_BINARY, 17, 3)
cv2.imwrite('a1.jpg', thresh)
```

Here the second parameter 255 means, when the pixel is greater than the local threshold it becomes 255. The fifth parameter 17 says it will be a 17x17 region of the image to compute a local threshold. Finally, 3 is a constant. This value is subtracted from the computed threshold value. Based on your project you need to choose a region of the image to compute the local threshold and the constant.

Here is what the image looks like:

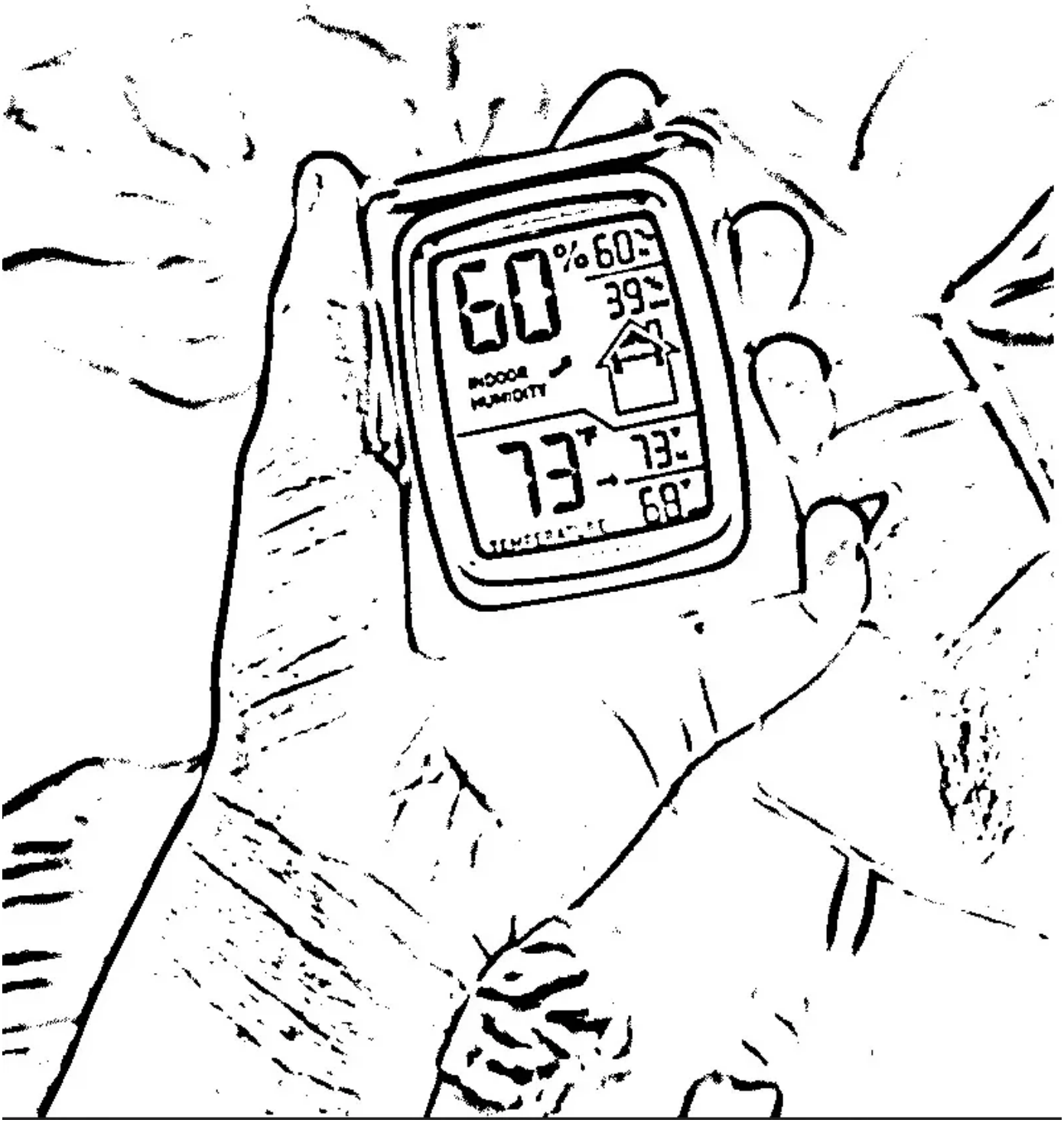


Image by Author

As per the discussion above, adaptive thresholding also needs some trials to find the last two parameters. But still, the method itself does a lot for us.

Conclusion

All three thresholding methods are important. Each of them may come in handy based on your situation or your project. Please feel free to try different images.

More Reading

Morphological Operations for Image Preprocessing in OpenCV, in Detail

Erosion, dilation, opening, closing, morphological gradient, tophat / whitehat, and blackhat explained with examples

towardsdatascience.com

Some Basic Image Preprocessing Operations for Beginners in Python

OpenCV for beginners: move or translate, resize, and cropping

towardsdatascience.com

A Step-by-Step Tutorial to Develop a Multi-Output Model in TensorFlow

With complete code

towardsdatascience.com

Some Simple But Advanced Styling in Python's Matplotlib Visualization

Adding Some Extra Flavor to Your Python Plots

pub.towardsai.net

Precision, Recall, and F1 Score of Multiclass Classification — Learn in Depth

Manual Calculation From a Confusion Matrix and the Syntax of sklearn Library

towardsdatascience.com

How to Make Animated and Racing Bar Plots in Python

Complete Working Code

towardsdatascience.com

Data Science | Machine Learning | Artificial Intelligence | Programming | Computer Vision

Follow

Written by Rashida Nasrin Sucky

5.9K Followers · Writer for Towards Data Science

MS in Applied Data Analytics from Boston University. Read my blog: https://regenerativetoday.com/

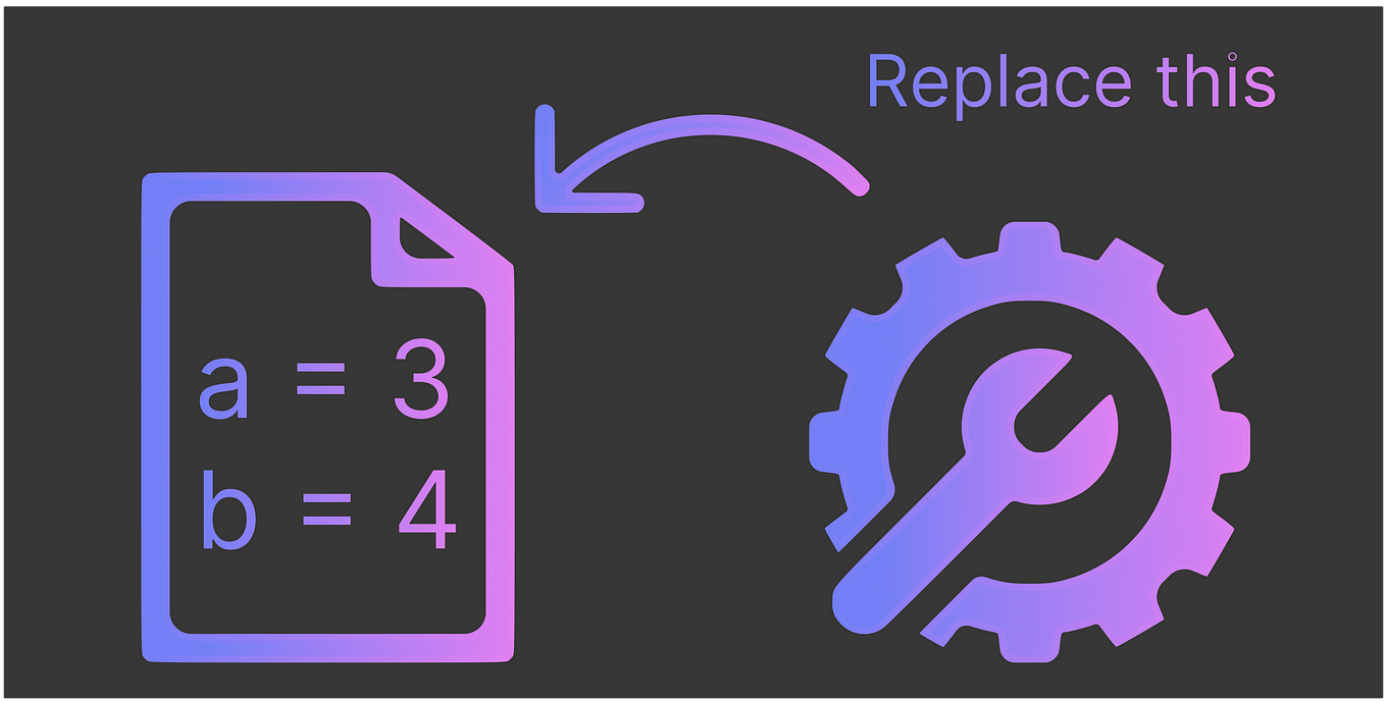
More from Rashida Nasrin Sucky and Towards Data Science

Rashida Nasrin Sucky in Towards Data Science

Pivot and Unpivot Functions in BigQuery For Better Data Manipulation

A detailed tutorial

· 5 min read · Jul 22, 2022



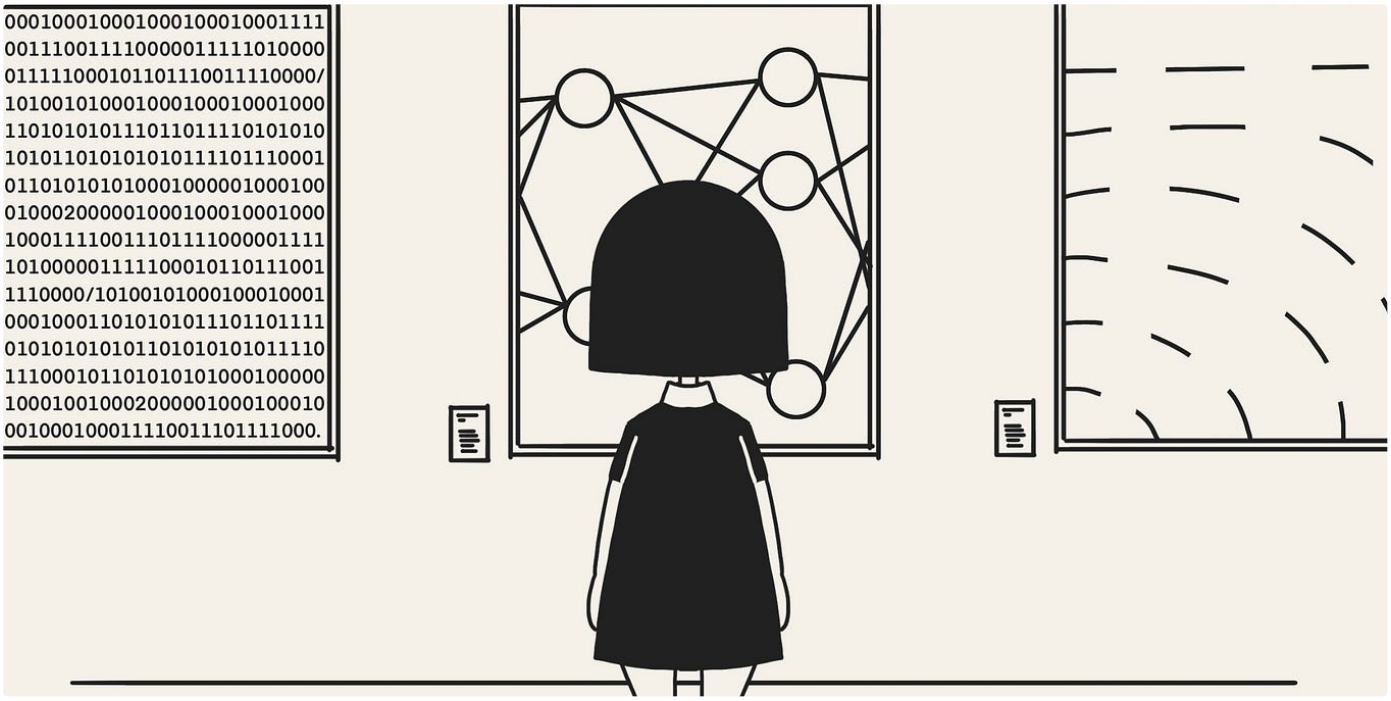
Khuyen Tran in Towards Data Science

Stop Hard Coding in a Data Science Project—Use Config Files Instead

And How to Efficiently Interact with Config Files in Python

✨ · 6 min read · May 26

1.7K 20



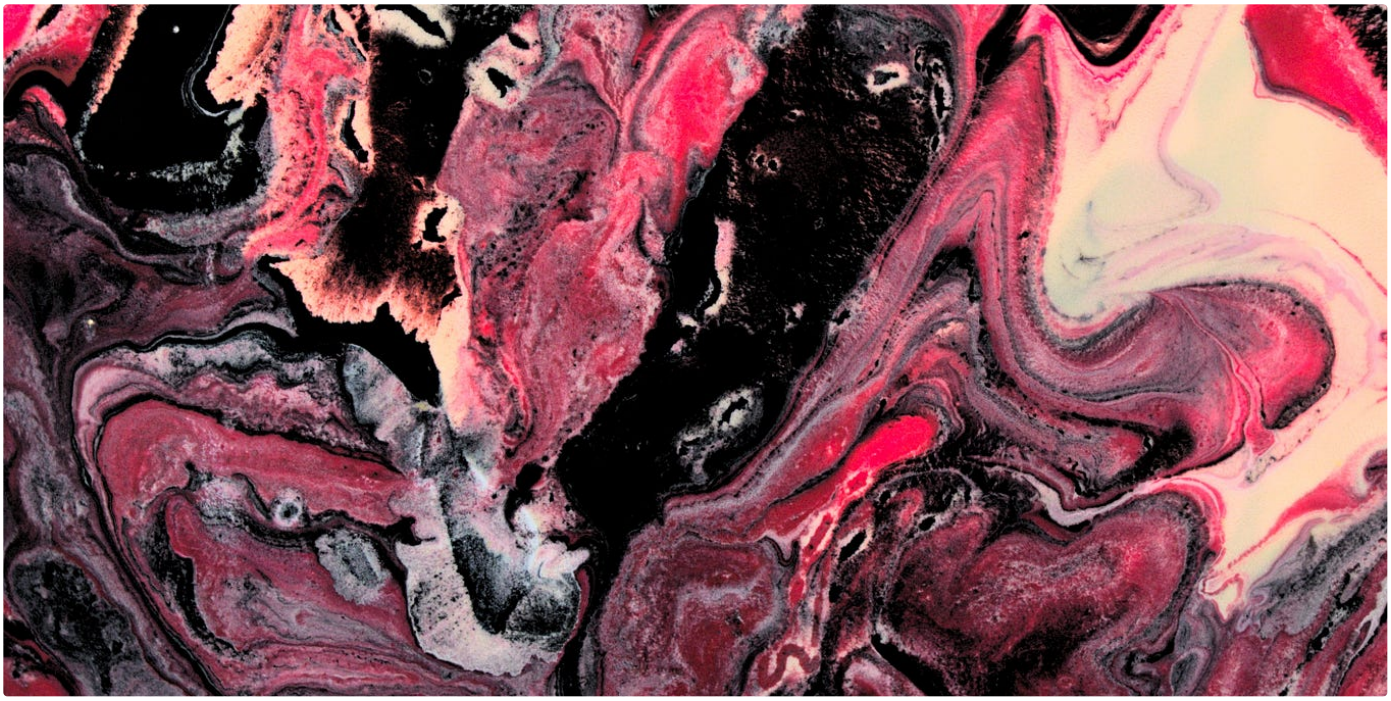
Leonie Monigatti in Towards Data Science

10 Exciting Project Ideas Using Large Language Models (LLMs) for Your Portfolio

Learn how to build apps and showcase your skills with large language models (LLMs). Get started today!

✨ · 11 min read · May 15

1.5K 9



Rashida Nasrin Sucky in Towards Data Science

Multivariate Linear Regression in Python Step by Step

Learn to develop a multivariate linear regression for any number of variables in Python from scratch.

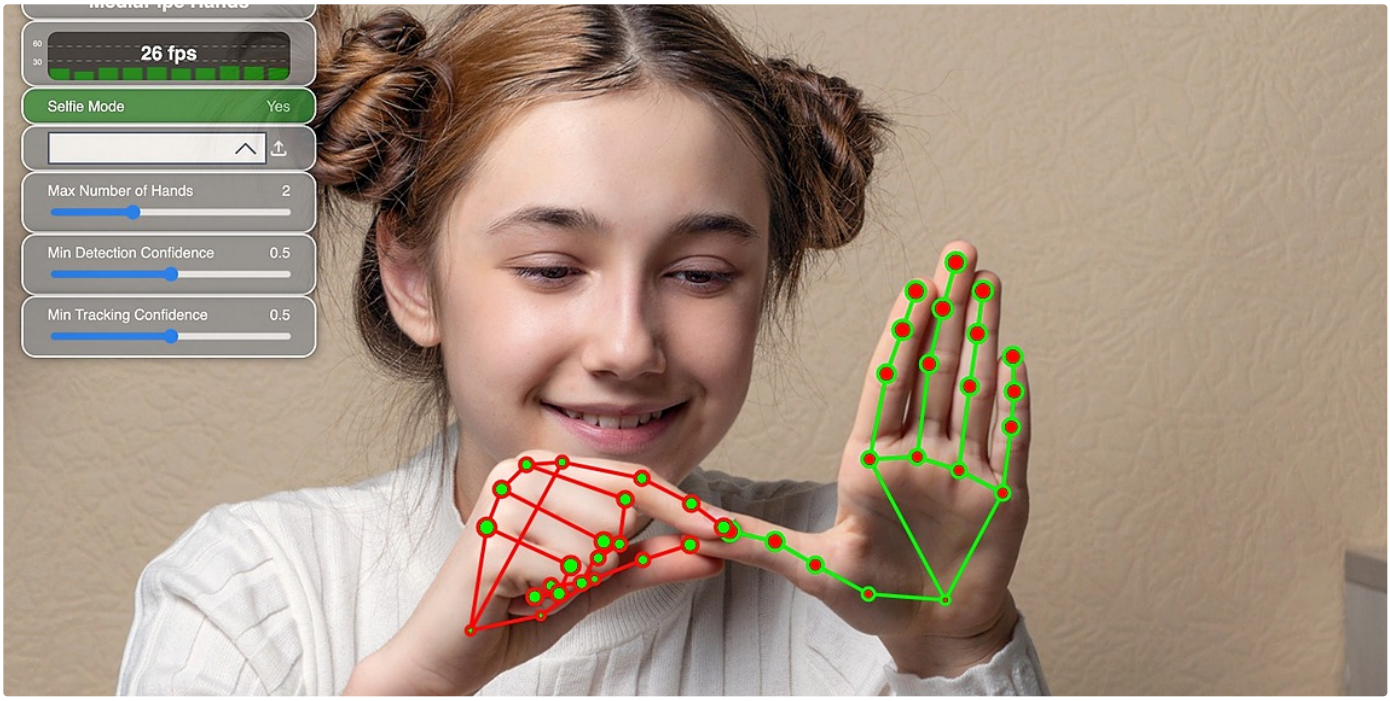
· 6 min read · Jun 25, 2020

Herscovici Robert

How to speed up Image Registration with OpenCV by 100x

While there are many great articles explaining Image Registration, I couldn't find anything that shows how to make this procedure run in a...

5 min read · Jan 5



Keshanth Jude in Dev Genius

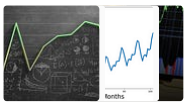
Object Detection with MediaPipe

After the massive release of ChatGPT, the world has grown curious (and scared) of Artificial Intelligence and Machine Learning. This blog...

4 min read · Jan 3

168

Lists



Predictive Modeling w/ Python

18 stories · 29 saves

Practical Guides to Machine Learning

10 stories · 46 saves

Natural Language Processing

356 stories · 15 saves

ChatGPT prompts

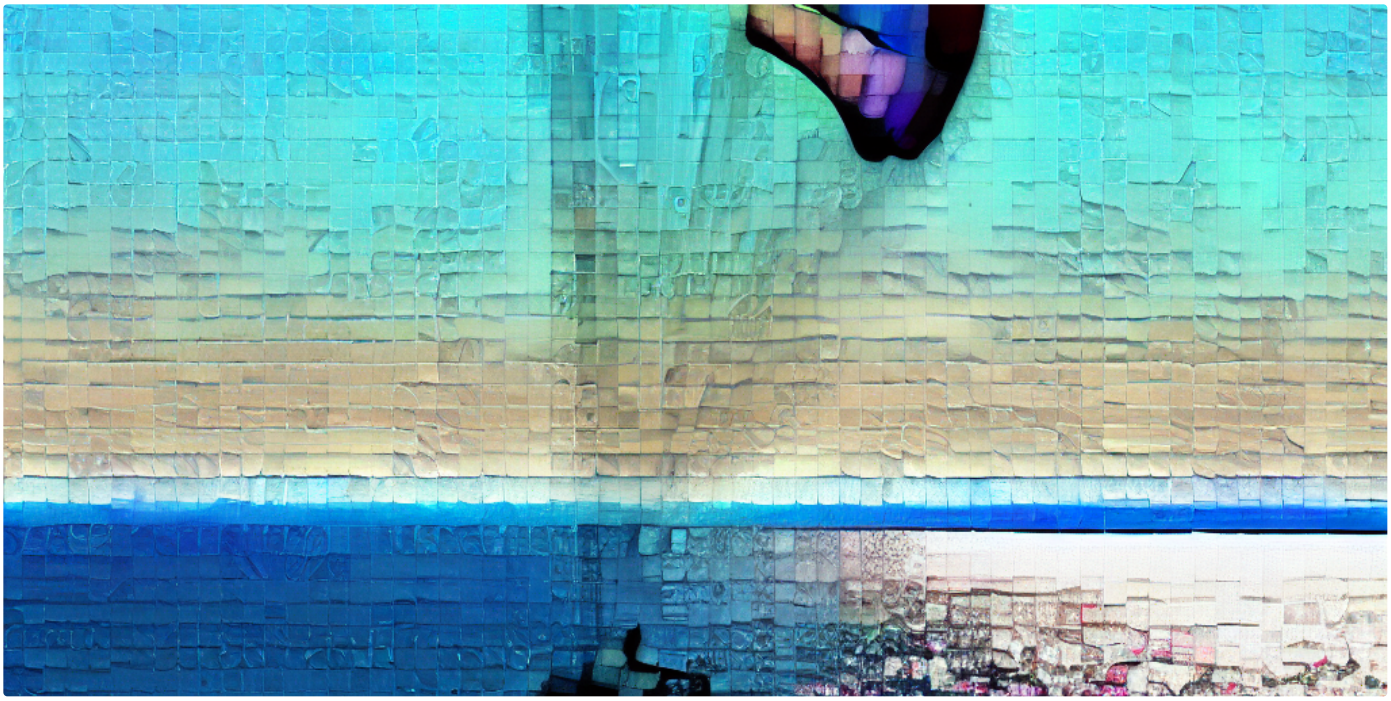
17 stories · 19 saves

Shahar Gino

Sport-Analytics with YOLO et al.

YOLO is truly amazing and allows a working out-of-the-box solution for various real-time detection problems, even with minimal programming...

4 min read · Feb 12

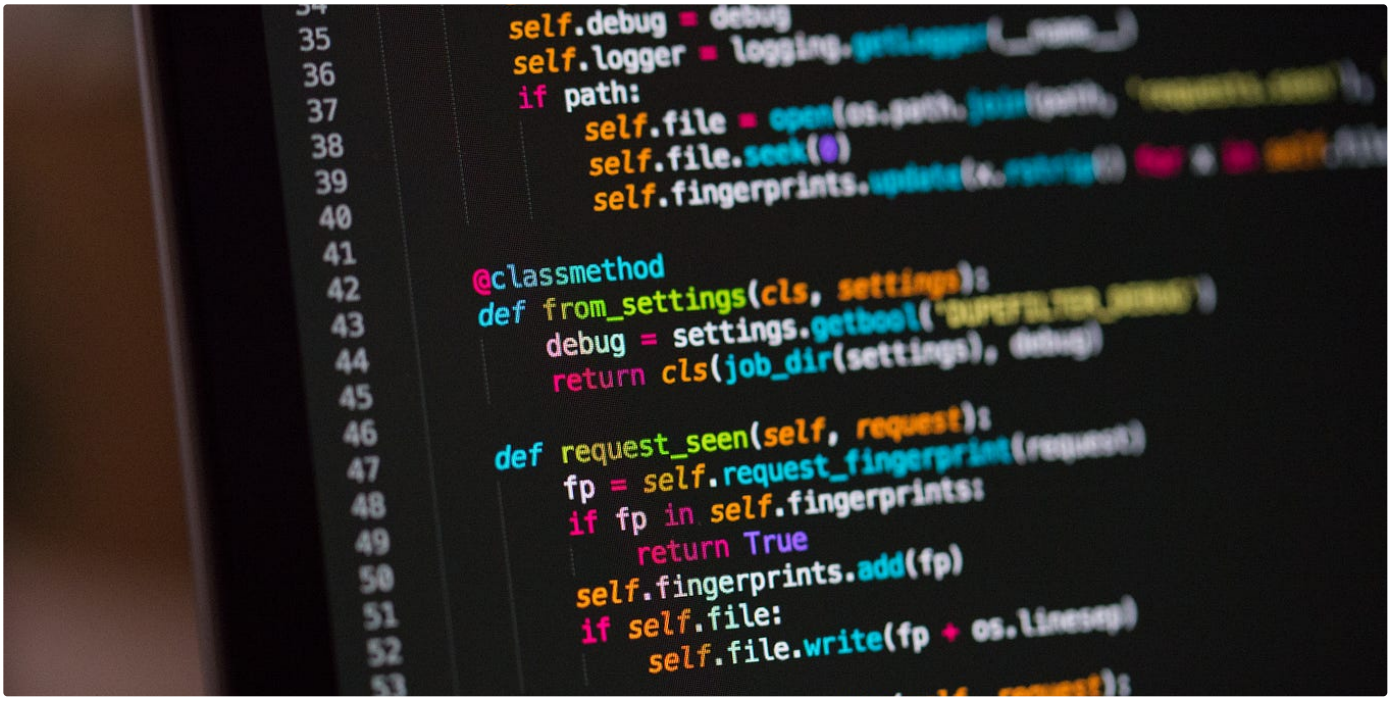


Francis Camarao

Image Processing using Python—Image Segmentation

In this article, we embark on an exciting journey through the realm of image segmentation as we delve into the implementation of this...

5 min read · Jun 17



Maximilian Strauss in Better Programming

A Practical Guide To Extract Text From Images (OCR) in Python

How to use optical character recognition with three libraries

· 7 min read · Jan 10

Aarafat Islam in TheLatestAI

Transforming Reality: Turn Your Photos into Cartoons with OpenCV

A Step-by-Step Guide to Converting Your Photos into Cartoon-style Images Using OpenCV's Image Processing Techniques.

8 min read · Apr 8