Digi-Lighter: Digitize your Highlights

Detecting (physically) highlighted text using OCR and colour-specific contouring

Shaham · Follow

8 min read · Apr 16, 2018

> Web-App hosted @ digilight.shaham.me (Sorry, Link is not currently live!)

Motivation

Some of us are data-hoarders. Specifically for myself — whether it's the engineering part of me or just a semi-neurotic personality trait inherited from my ancestors — it agitates me to know that some set of information that I've collected will eventually become lost/corrupted/inaccessible to me in the coming future. Akin to the pre-Information Age personality of the person-who-keeps-all-his-boxes, we have issues with letting data 'go' (missing). Couple that with an equally peculiar desire to work with our hands and

interact with offline materials like wood, paper, etc., we gracelessly enter ourselves into a dilemma.

This project began with a specific version of the dilemma above: **I highlight what I'd like to revisit in books, papers, etc. but if something is not at the tips of my fingertips (a.k.a 'on my phone'), the chances of it being revisited are next to NULL.** (Also, externalizing quotations, passages, etc. from your brain frees up its Random Access Memory for analysis). So, I start thinking of alternatives: I try Kindle, PDFs and highlight within them — I try taking pictures of pages and tagging them — I try apps like EverNote and try to make it a habit to put my discovered-offline quotations on there ***— I try keeping a journal where I periodically write important things. I try all this with no avail — Kindle is not a book, it does not smell like a book, so it is not a book —Pictures get lost, I can't search the text in pictures and let's face it, no one has time to tag pictures — I am way, way too lazy to redundantly record information and actually type out what I read, and rightfully so, we have technology, we should not have to. I start exploring options, I come across some fancy-gimmicky highlighter sold online, made in China, that performs real-time OCR letter by letter as you slide it across the page. It's still not good enough, because a) they aren't highlighters, they don't smell and feel like highlighters, so they are not highlighters b) who wants to carry around a heavy duty $99 digital-highlighter and then 'charge' it, my God and c) even if the information is stored online (possibly), it is lost offline. And then, suddenly, the great wisdom of Shia Labeouf hits me: "Just Do It! (yourself)".

Overview of Application

But how? The Samurai-Buddhist Miyamoto Musashi once wrote that for getting things done and doing them right, "Perceive that which cannot be seen with the eye". But for the computer, the saying here should be flipped,

to do things ‘right’ and in the way we want them done, ‘it must perceive what is perceived with the eye’.

So we begin the task of trying to get the computer to **Extract Highlights from pictures, Digitize the Content, and then Store the Information.** The user (You and I) should be able to take pictures of their highlights and this application would make this information digitally-accessible, providing all the text-related features one would get from a digital copy. The user could then store this information automatically (integrate with user’s EverNote account).

> ***Task:*** *Extract only the user-highlighted portion from pictures to digitize and store.*
>
> ***Goals:*** *Search-ability of text, accessibility of notes, ease-of-use, ability to digitally select/copy highlighted text*

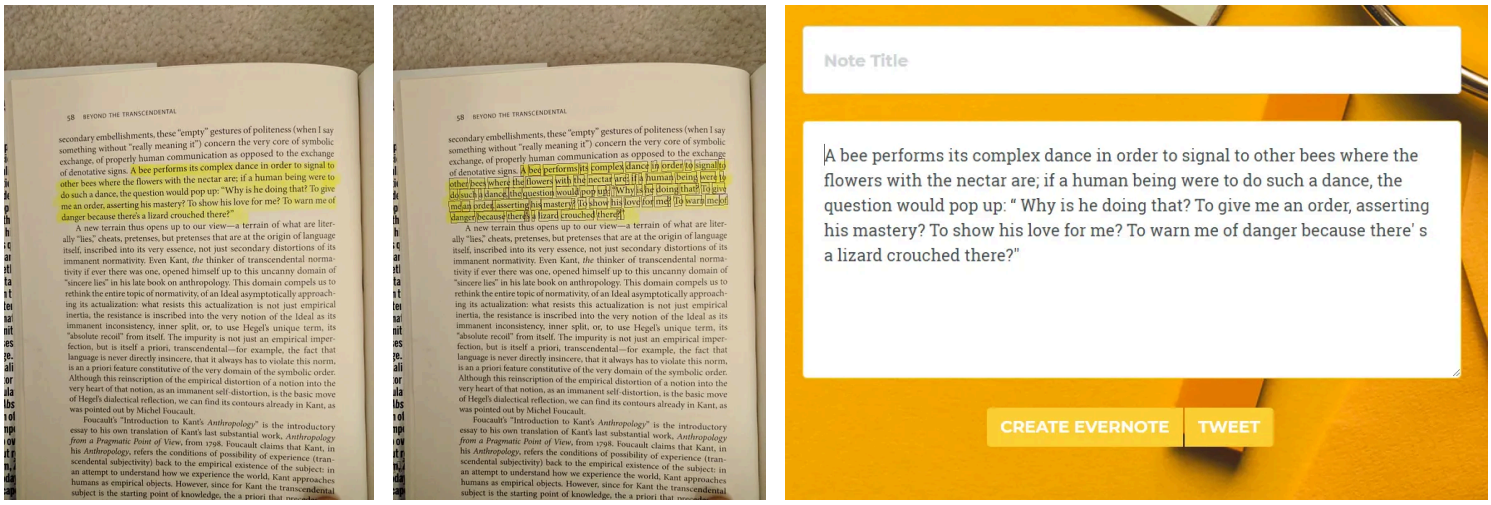


Figure 1: Sample of Digi-Lighter Application

The Tech

The project involves 2+1 fundamental modules: Image Processing (contours, colour detection, etc.) and OCR (Optical Character Recognition) — the extra module is the Web side to make this user-friendly and store the information.

OCR — Optical Character Recognition

Since digitizing characters is an old problem and one that's been widely attacked by the tech world, there is no need to re-invent or attempt a custom version. I simply used Google's Vision API.

Image Processing

I used OpenCV's Python library. This involved OpenCV's built-in contouring methods (which as we'll see, didn't quite do the trick even with some custom tweaks), and overall handling of images.

Web

Since I was using Python anyways, the easiest option was to use Flask. I also used EverNote's API to automatically create notes for users and save highlight extractions in their notebooks.

For the rest of this write-up, I quickly want to share an attempt at contouring user-highlights from pictures and the final algorithm that worked (kinda).

Initial Contouring Attempts

OpenCV Colour-based Threshold and Contouring

The most obvious place to start is with the built-in tools. The trick here has to do with colour: before contouring, I had to implement a colour threshold (manually set to detect yellow highlight colours). Below is some code showing this process.

```
#highlight colour range
hsv_lower=[22, 30, 30]
hsv_upper=[45, 255, 255]

image = cv2.imread(img_path)

# rgb to HSV color spave conversion
hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

HSV_lower = np.array(hsv_lower, np.uint8)  # Lower HSV value
HSV_upper = np.array(hsv_upper, np.uint8)  # Upper HSV value

#Threshold
frame_threshed = cv2.inRange(hsv_img, HSV_lower, HSV_upper)

# find connected components
_, contours, hierarchy, = cv2.findContours(frame_threshed,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
```

After contours are extracted, we notice a lot of tiny contours all over the image due to colour shades that our threshold method will have highlighted. To eliminate this noise and filter down to large contours (the assumption here is that the larger contours would be the highlighted portions), we can perform some statistical analysis on the area of the contours to keep only those outside a specific number of standard deviations from the mean(i.e. all contours 4 standard deviations away from the mean).

At this point, there's some real-world problems we encounter: users don't really highlight in a way that is ideal for computers to contour purely off colour. Furthermore, sometimes, the contour is too close to words and cuts off some letters, making OCR difficult. Therefore, I wrote some quick logic to expand the contour out by a percentage across the x and y axis and then perform some smoothing/interpolation. After this process, here's some examples of what we get:

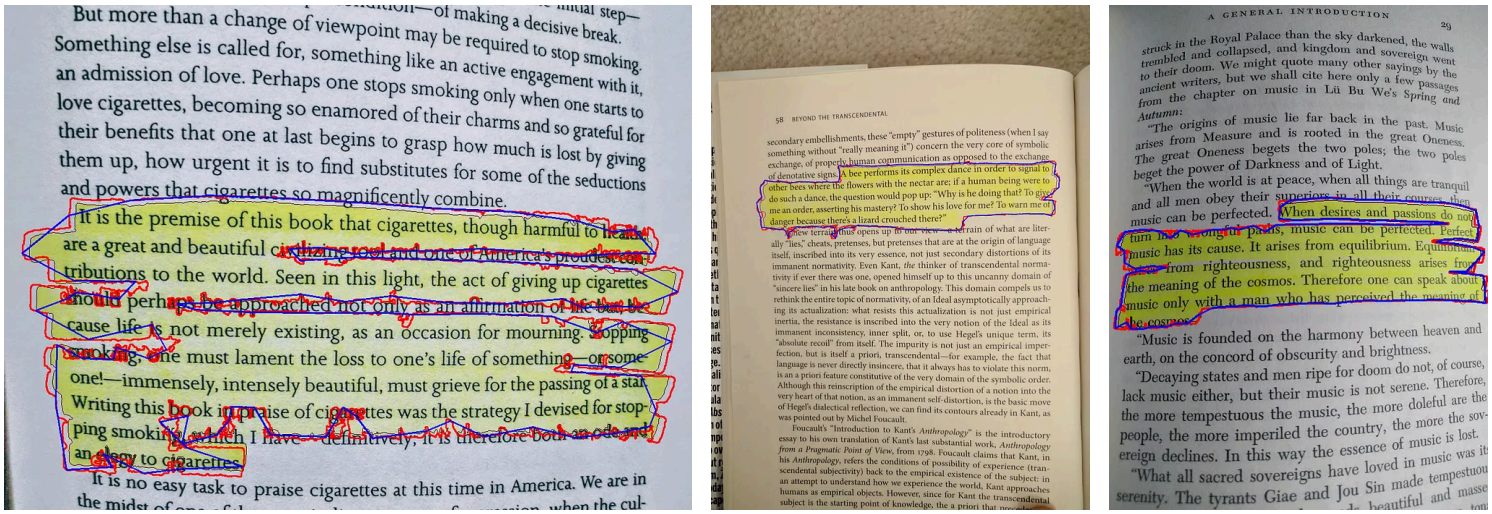


Figure 2a, 2b, 2c: Examples of Highlight Contours (red=contour on thresholding, blue=expanded and smoothened curve)

With this, we can get something manageable — although you clearly see issues (words cut off, mid-paragraph areas not captured, etc.) that will make OCR difficult. However, considering most people highlight rather casually, to ask of clean highlighting from users to make a computer's life is easy is still too demanding. Therefore, we move to another logic of "contouring" that's not really "contouring" at all...

Post OCR "contouring"

Contouring, essentially, is boundary detection (edge detection but finding closed curves) so, yes, this is cheating. Through the method in this section, instead of performing an explicit boundary-detection step, we find the highlighted boundary implicitly — it seems to be a better technique to deal with our problem which is really the initial task. Rather than deciding what is highlighted right off the image like a human eye would, the computer has another advantage: computation power. Therefore, we can approach this in a less intuitive manner: **OCR first, and then determine, word-by-word, if it is 'highlighted' or not.**

We can do this by using our colour-specific thresholding once again. Firstly, using Google's API, alongside the text, we can also extract a boundary-box around each word OCR-ed.

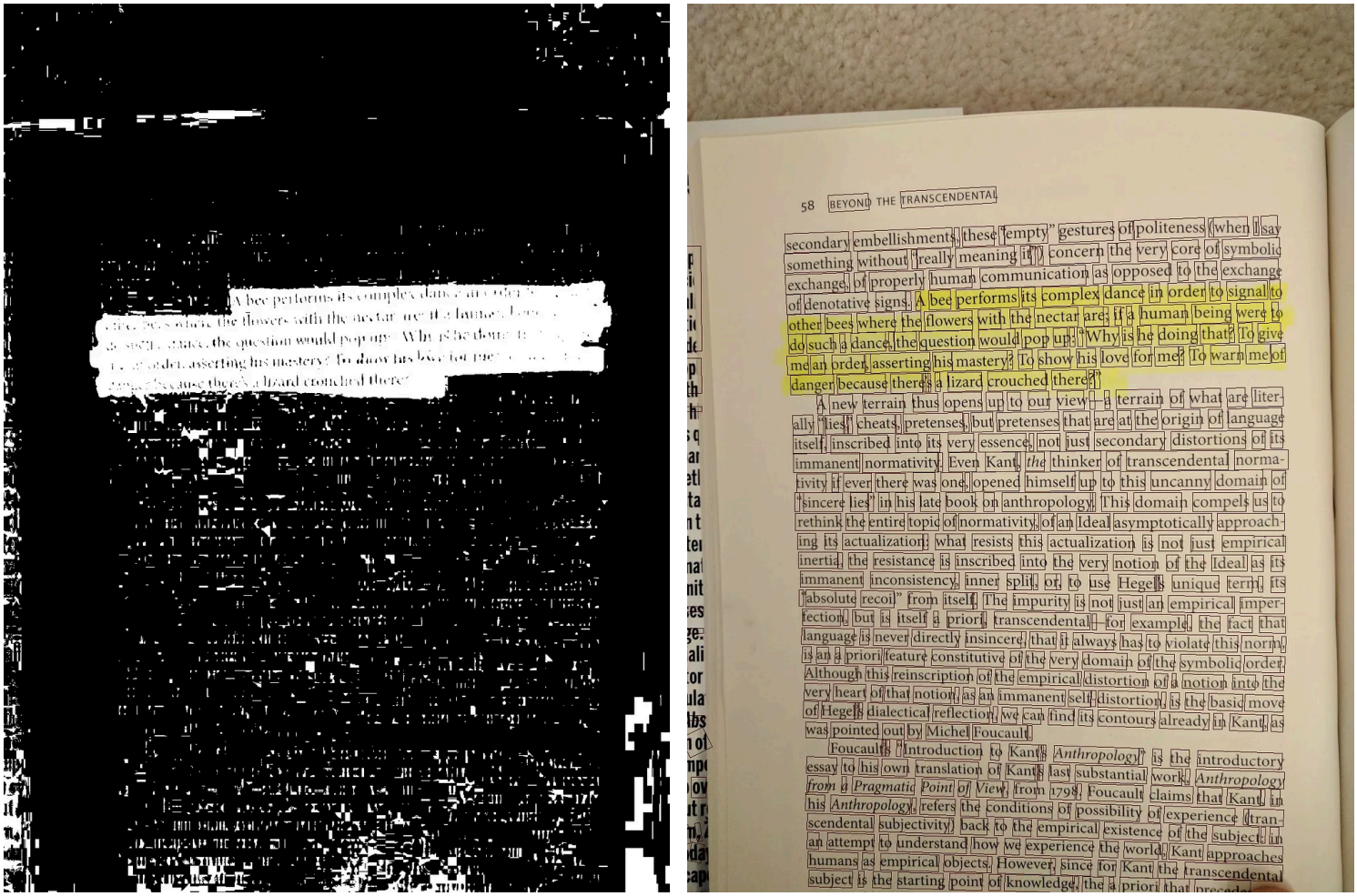


Figure 3: Thresholding and Boundary-Boxes of Words

From here, we iterate word-by-word and perform a simple test to determine if the word is 'highlighted' or not:

```
Threshold_Ratio = <percentage of area that must be 'highlighted'>
Above_Threshold_Area = sum(pixels not above threshold)

if Above_Threshold_Area>Total_Word_Area*Threshold_Ratio:
     Highlighted = True
```

With this logic, we can account for weakly-highlighted areas and solve a bunch of issues arising from messy and noisy highlights. Couple this with some more logic for edge cases and common scenarios — (ex. if a word is sandwiched between highlighted words, then consider it highlighted) — we achieve a more accurate output than our previous attempt:

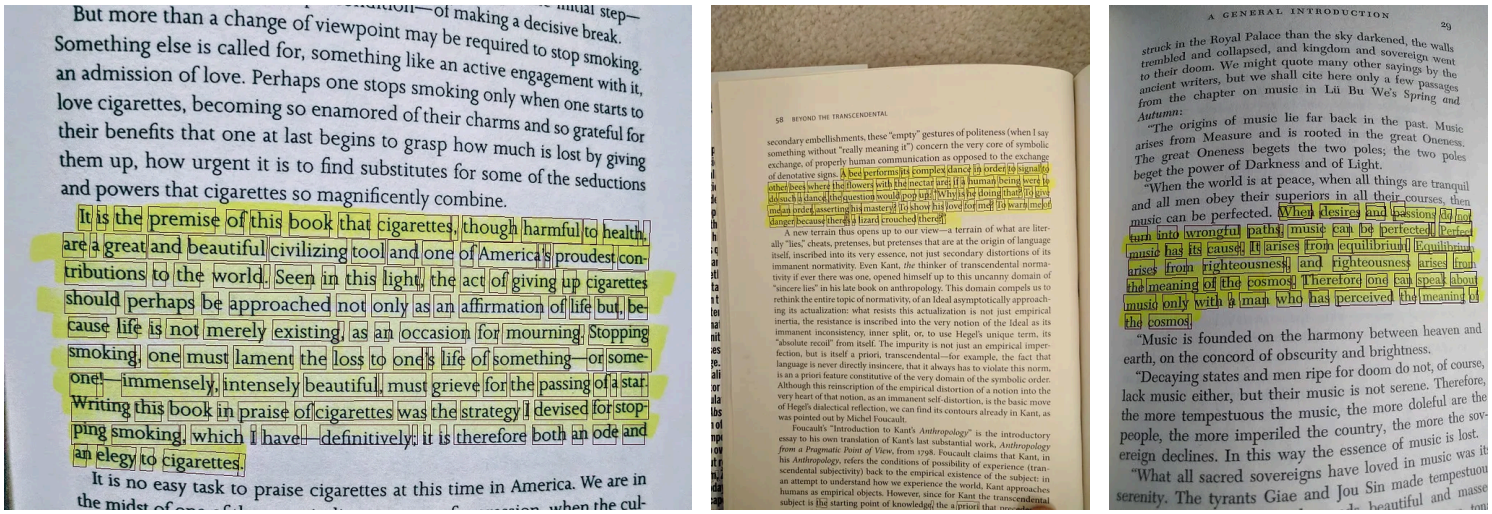


Figure 4a, 4b, 4c: Post-OCR Contoured Examples

Conclusion

That's about the crux of this experimental side project. Personally, this was an introduction to some basic image processing methods for me and so, if anyone has any suggestions/ideas/critiques, feel free to share. A reminder that I have hosted this web-application at the link below — try it out, see if it works for you, be advised it might be a bit slow, I haven't bothered to optimize the code yet.

Web-App hosted @ digilight.shaham.me

***EverNote: A Reasonable Alternative

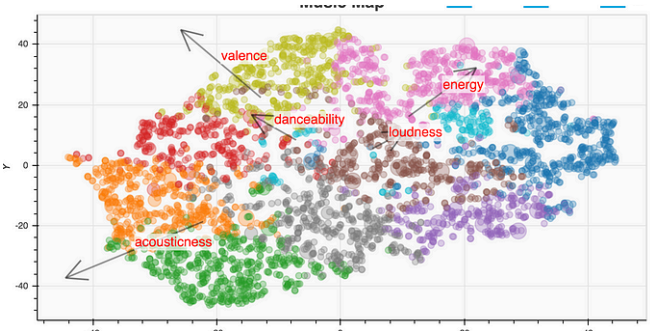
In the process of developing this (a day before I finished…ugh), EverNote released a new integration feature that connects directly to your Google Photo's gallery and allows you to easily create a note of an image containing text. Once a note is created, EverNote's ability to recognize text in images kicks in. Though there is no specific 'highlighting' concept, the EverNote application does actually OCR text from images that you attach — this text is then searchable through the EverNote application. At first, attaching images to EverNote manually was painful so this was not a viable alternative in my case, though now with this recent integration I might give it another change. There are still factors that may *not* make this the best fit for you:

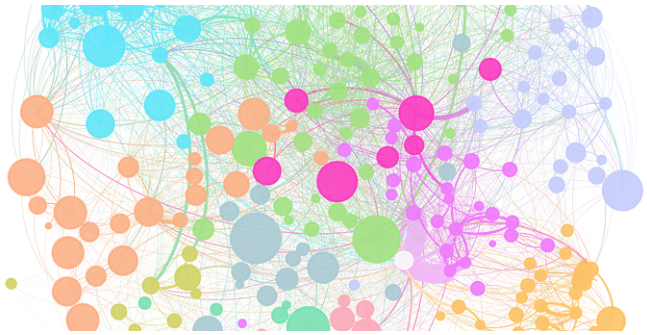
- It does not allow you to 'select/copy' text from the image
- I have found that its OCR is not that great(ironically, specifically highlighted words seems to not be recognized at times)
- Each image captured from your gallery saves to one note, making it harder to create a collection of multiple images in a single note

Image Processing | Ocr | Notes | Information | Opencv

Written by Shaham

More from Shaham





Shaham in Towards Data Science

The Art of Creating a Mixtape—A Data Science Approach

Using Data Science techniques to map out a Music Library in 2D to create a spatial...

9 min read · Mar 6, 2019

64 1

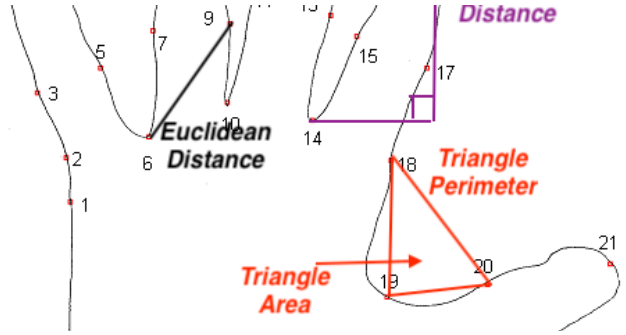
Shaham in Towards Data Science

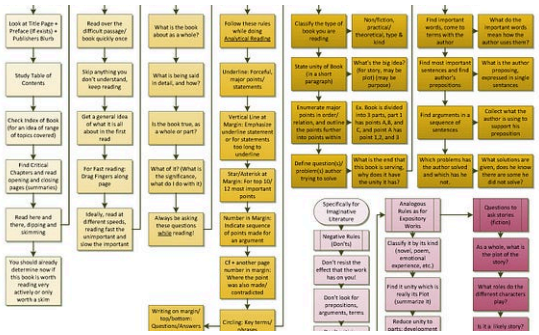
Generating Twitter Ego-Networks & Detecting Ego-Communities

A Graph-based approach to community detection in Twitter Networks

21 min read · Dec 12, 2018

427 28





Shaham in Towards Data Science

Sample Hand-Geometry Biometric Identification System

A simple example of the Process of Modelling for Biometrics

8 min read · Feb 21, 2019

10

Shaham

Adler's "How to Read a Book" Flow Chart

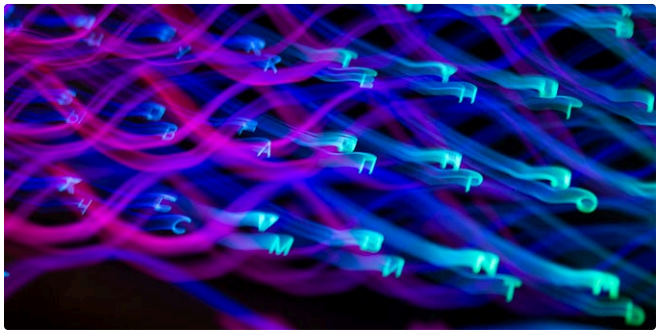
Summary Visualization of the Classic Guide to Intelligent Reading

1 min read · Jan 22, 2019

35

See all from Shaham

Recommended from Medium



Ahmed Amine Hamrouni

Scanning Documents with Perspective Transformation usin...

5 min read · Nov 19, 2023

25

Sandaruwan He... in Data Science and Machine Le...

Image Enhancement with Python

Image enhancement is a fundamental process in the field of image processing,...

18 min read · Feb 24, 2024

206

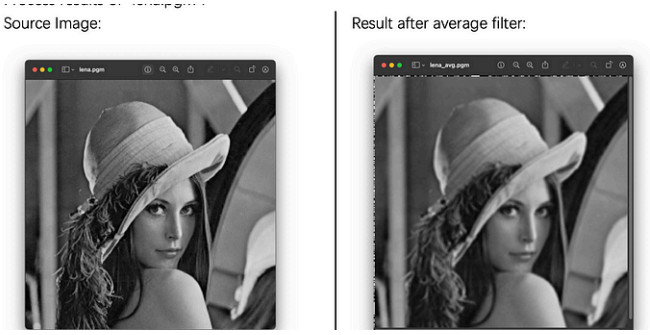
Lists

Productivity

240 stories · 415 saves

Natural Language Processing

1411 stories · 909 saves



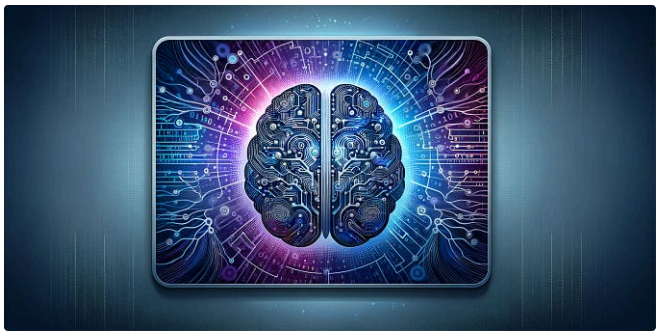
Zhe LIN

Digital Image Processing in C (Chapter 1): Mean and Median Filter

Mean Filter and Median Filter with Complete Code in C

4 min read · Nov 15, 2023

1 1



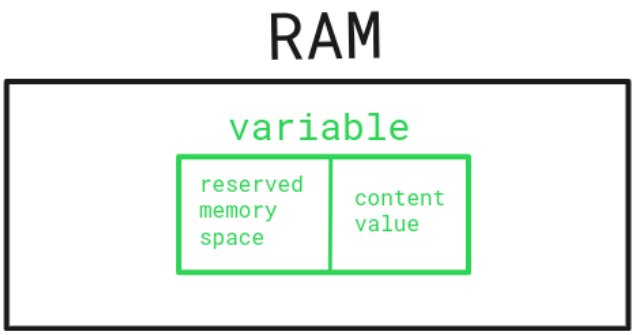
Cristian Leo in Towards Data Science

The Math Behind Neural Networks

Dive into Neural Networks, the backbone of modern AI, understand its mathematics,...

· 28 min read · Mar 28, 2024

3.1K 20



Jake Page

The guide to Git I never had.

Doctors have stethoscopes.

13 min read · Apr 11, 2024

3.6K 33

João Loss

C pointers: what you must know

I'm 100% sure that at the of this read your C codes will get to the next level, just trust me...

13 min read · Jan 23, 2024

6 2

See more recommendations