

Matrix operations on License Plate Detector and Recognizer (LPDR)

Wiktor Jakub Maj
Bachelor degree student at PJAiT

07.09.2020

Contents

1	Abstract	2
2	Introduction	2
3	System architecture	3
3.1	Gathering optical signal	4
3.2	Image processing	5
3.3	License Plate Detection (LPD)	6
3.4	Affine transformation	7
3.5	Normalization	7
3.6	Non-max suppression	7
3.7	Sizing	8
3.8	T-matrix	8
3.9	Warping perspective	10
3.10	Rescaling	10
3.11	Denoising	10
3.12	Eroding	10
3.13	License Plate Recognition (LPR)	11
3.14	Adaptive thresholding	11
3.15	Connected component analysis	11
3.16	Word recognition	11
3.17	Classification	11
4	Metrics	12
5	Acknowledgements	12

1 Abstract

Many of today's computer vision systems offer ready to use, high-end solutions (especially in cloud platforms) that can be used in production environments. The workload of researchers/engineers and open source communities raises service quality and enables users to use all kinds of systems without getting involved in operations behind. A good computer vision system should work in many unconstrained scenarios, but in case of failure, the degree of complication usually makes users unable to do anything. Rather than dive into direct operations, it is usually easier to leave or switch the system into a different one if it does not interrupt organization structure. The main goal of this paper is to magnify the operations behind a fully-customized LPDR system and to stimulate human curiosity as well as mine.

2 Introduction

License Plate Detector and Recognizer (LPDR)¹ is a low-code, scalable, and easy to modify object detection framework with optical character recognition. It was designed to be a simplified wrapper of state of the art systems without getting a significant deterioration in performance (see [Metrics](#)). LPDR is a product of an open-source initiative² and is publicly available for personal or commercial use. This article describes operations on matrices as well as an overview of underlying processes behind the user interface. The following paper bases 0.1 version of LPDR and does not cover further releases. If you use lpdr in your work please cite my paper!

¹ <https://github.com/szachovy/lpdr>

² <https://opensource.org/licenses/MIT>

3 System architecture

The initial way of system execution steps are processed in the pipeline shown as below:

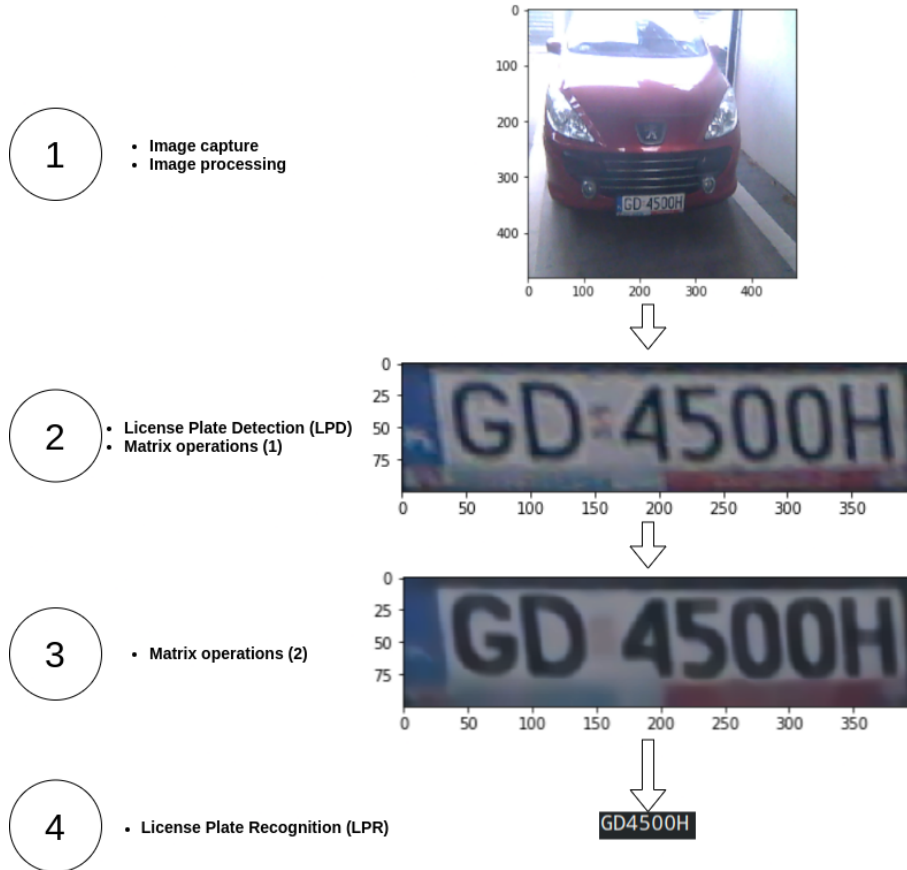


Figure 1: Sequence of computing processes.

One of the main advantages of LPDR is that every step can be replaced or omitted if the desired output from previous steps matches the constraints for the next steps, which are also modifiable. Depending on the individual needs, LPDR ought to be used in the following ways:

	<u>LPDR as a microservice</u>	<u>LPDR as a Python package</u>
Execution:	system environment	python object
Input:	graphical file or optical source	graphical file or optical source
Output:	stdout/stderr	python string

3.1 Gathering optical signal

LPDR uses a set of classes, functions, and utilities from The OpenCV³ Video I/O module for signal propagation from camera devices.

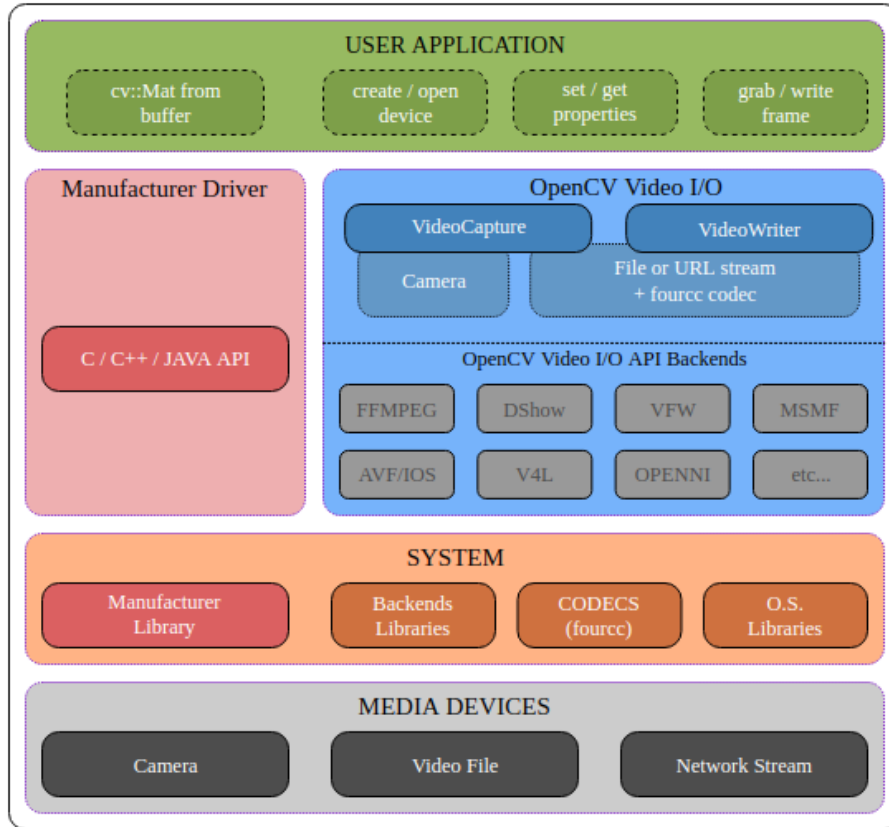


Figure 2: Video I/O⁴ with OpenCV

With the capture flag set to True, it writes the last frame with the specified image width and height (in pixels) to the file. The image format is chosen based on the filename, only 8-bit (or 16-bit unsigned (CV-16U) in case of PNG, JPEG 2000, and TIFF) single-channel or 3-channel (with ‘BGR’ channel order) images are saved⁵.

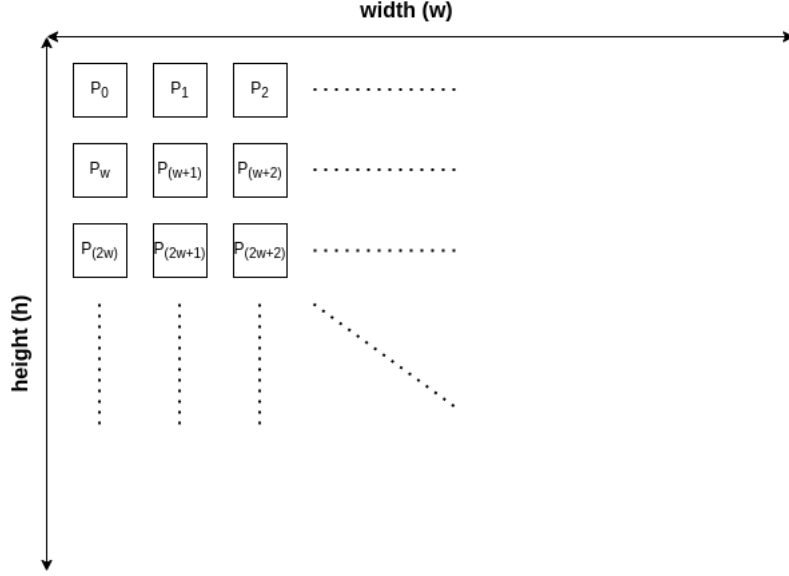
³ <https://opencv.org/>

⁴ https://docs.opencv.org/3.4/d0/da7/videoio_overview.html

⁵ https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html

3.2 Image processing

We can describe our image as a matrix, where each point responds to one BGR pixel represented as 3×1 vector.



$$\begin{aligned} P_0 &= [b_0, g_0, r_0] \\ P_1 &= [b_1, g_1, r_1] \\ P_{(w+1)} &= [b_{(w+1)}, g_{(w+1)}, r_{(w+1)}] \dots \end{aligned}$$

For each n point:

$$P_n := (J_3 \cdot P_n) / 255$$

where J_3 is anti-diagonal identity matrix, and $n \in \langle 0, (w * h) - 1 \rangle, P_n \in \langle 0, 1 \rangle$

To increase speed LPDR uses downsampling with area interpolation. Let represent area interpolation as rectangular pooling object with a size of:

$$a_h = \frac{h}{H}, a_w = \frac{w}{W} \text{ where: } H \leq h, W \leq w, H = W, (H, W) \in \mathbb{N}$$

are scaled width and height. Assuming that $p \in \langle 0, \max(original_h - a_h, original_w - a_w) \rangle$ is the place in rectangle at the given grid of pixels area, and a_{P_n} indicates the area of given P_n , that have not been covered by previous interpolations.

$$P_n \mapsto \frac{\sum_{i=p}^{j=p+a_w} \int a_{P_{[i]}} * P_{[i]}}{a_w}$$

for $p \in \langle 0, (original_w - a_w) \rangle$ when moving horizontally first. Then:

$$P_n \mapsto \frac{\sum_{i=p}^{j=p+a_h} \int a_{P_{[i]}} * P_{[i]}}{a_h}$$

for vertical move, $p \in \langle 0, (original_h - a_h) \rangle$ and $h := H, w := W$.

3.3 License Plate Detection (LPD)

Given output matrix, where the last channel represents the color channels. LPDR uses pretrained Warped Planar Object Detection Network (WPOD-NET)[9] which bases on the YOLOv2[6] network.

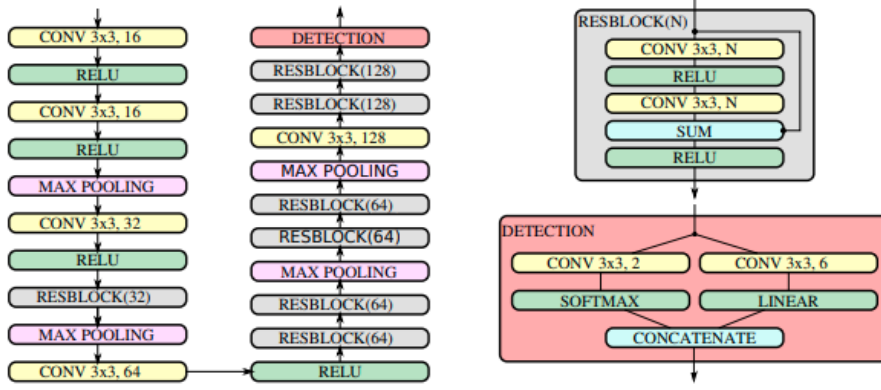


Figure 3: Detailed WPOD-NET architecture [9].

For each P_n :

$$\begin{bmatrix} r_n \\ g_n \\ b_n \end{bmatrix} \xrightarrow{\text{prediction}} \begin{bmatrix} p_{n_o} \\ p_{n_n} \\ a_{n_1} \\ a_{n_2} \\ a_{n_3} \\ a_{n_4} \\ a_{n_5} \\ a_{n_6} \end{bmatrix}$$

- p_{n_o} and p_{n_n} are object / non-object probabilities activated by a softmax function [9].
- $a_{n_{1..6}}$ are affine parameters emerged from identity $F(x) = x$ linear layer as the activation function [9].

Let $Q \ni P_{0..m}$, where $m = w * h - 1$ after downsampling:

$$S_n = \{Q_n \forall P_n : p_{n_o} > \theta\}$$

,where θ is a threshold for license plate probability of detection, and $S \subseteq Q$ is a filtered set of points.

3.4 Affine transformation

In each $S_n \in S$ point, $a_{n1..6}$ are used to build local affine transformation⁶ matrix [9]:

$$S_n := \begin{bmatrix} \max(a_{n1}, 0) & a_{n2} \\ a_{n4} & \max(a_{n5}, 0) \end{bmatrix} \cdot l + \begin{bmatrix} a_{n3} \\ a_{n6} \end{bmatrix}$$

,where l is the loss function $\forall n$ given by:

$$l = \begin{bmatrix} -0.5 & 0.5 & 0.5 & -0.5 \\ -0.5 & -0.5 & 0.5 & 0.5 \end{bmatrix}$$

3.5 Normalization

To match the output resolution, each point needs to be rescaled and recentered [9].

$$S_n := \frac{(S_n * \alpha + \begin{bmatrix} y_n + 0.5 \\ x_n + 0.5 \end{bmatrix})}{\begin{bmatrix} h/s \\ 1/s \end{bmatrix}}$$

,where:

α is a constant normalization parameter.

(x_n, y_n) are S_n coordinates in the grid of Q .

s is a network stride multiplication of four max pooling layers.

3.6 Non-max suppression

Let assign each point of $S_n \in S$ matrix as following: $S_n = \begin{bmatrix} v_{n11}, v_{n12}, v_{n13}, v_{n14} \\ v_{n21}, v_{n22}, v_{n23}, v_{n24} \end{bmatrix}$

The top left and bottom right corner of S_n rectangle coordinates are:

$$\begin{aligned} tl_n &= [\min(v_{n11}, v_{n12}, v_{n13}, v_{n14}), \min(v_{n21}, v_{n22}, v_{n23}, v_{n24})] \\ br_n &= [\max(v_{n11}, v_{n12}, v_{n13}, v_{n14}), \max(v_{n21}, v_{n22}, v_{n23}, v_{n24})] \end{aligned}$$

To get rid of overlapping boxes. The next filter called non-maximum suppression [8] (NMS), which uses "Intersection over Union" [7] (IoU) function, is superimposed on S boxes.

Let take $S_n \in S$ and $S_m \in S$, then:

$$\begin{aligned} I_{nm} &= S_n \cap S_m = \\ &(\min(br_{n1}, br_{m1}) - \max(tl_{n1}, tl_{m1})) * (\min(br_{n2}, br_{m2}) - \max(tl_{n2}, tl_{m2})) \end{aligned}$$

$$U_{nm} = S_n \cup S_m = ((br_{n2} - tl_{n2}) * (br_{n1} - tl_{n1})) + ((br_{m2} - tl_{m2}) * (br_{m1} - tl_{m1})) - I_{nm}$$

$$\varphi = \frac{I_{nm}}{U_{nm}}$$

⁶ https://en.wikipedia.org/wiki/Affine_transformation

The steps of Non-max suppression implementation can be described as:

1. Let define the domain of some function to be $D = \|S\|$ and $D \preceq D_k$
2. Let $f(n) = p_{n_o}$ for $n \in D$, where each p_{n_o} point directly corresponds to $S_n \in S$ point.
3. Select $k_1 = \operatorname{argmax}_n f(n)$ for $n \in D$ and put S_{k_1} box as a main box, update $D_k := D - \{k_1\}$
4. Select $k_2 = \operatorname{argmax}_n f(n)$ for $n \in D_k$ and put S_{k_2} box as an overlapping box, update $D_k := D_k - \{k_2\}$
5. Compute its overlap with other boxes from step 4, and remove boxes when nearest box overlaps more than φ .
6. If not returned, go back to step 3 with $D := D - \{k_1\}$ and repeat the steps until $D = \emptyset$

This will remove all boxes that have a large overlap with the highest probability boxes.

3.7 Sizing

Let's call the final box: $B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \end{bmatrix} \in S$

with corresponding $\exists tl_n = tl_B$ and $\exists br_n = br_B$ to the B box, the output matrix *height/width* ratio is:

$$\tau = \frac{tl_{B_1} - br_{B_1}}{tl_{B_2} - br_{B_2}}$$

$$\frac{h_o}{w_o} = \begin{cases} 1, & \text{if } \tau \geq \phi \\ v \in (0, 1) : \mathbb{R}, & \text{otherwise} \end{cases}$$

ϕ is a constant threshold for license plate size determination.

3.8 T-matrix

The T^* vector is the last column of real unitary matrix of singular value decomposition (SVD) [3].

$$A = U \Sigma V^*$$

Figure 4: Representation of T^* vector extraction.

where U is an $w_o \times h_o - 1$ real unitary matrix,
 Σ is an $w_o \times h_o$ rectangular diagonal matrix with non-negative real numbers on the diagonal,
and A is created from final box B and output box size dimensions as follows:

0	0	0	-b ₁₁	-b ₂₁	-1	0	0	0
b ₁₁	b ₂₁	1	0	0	0	0	0	0
0	0	0	-b ₁₂	-b ₂₂	-1	0	0	0
b ₁₂	b ₂₂	1	0	0	0	-w ₀ * b ₁₂	-w ₀ * b ₂₂	-w ₀
0	0	0	-b ₁₃	-b ₂₃	-1	h ₀ * b ₁₃	h ₀ * b ₂₃	h ₀
b ₁₃	b ₂₃	1	0	0	0	-w ₀ * b ₁₃	-w ₀ * b ₂₃	-w ₀
0	0	0	-b ₁₄	-b ₂₄	-1	h ₀ * b ₁₄	h ₀ * b ₂₄	h ₀
b ₁₄	b ₂₄	1	0	0	0	0	0	0

Figure 5: 8×9 A matrix

T matrix (coefficients transformation) is a vectorization factor of T^* matrix:

$$vec_{3,3}^{-1} \circ vec : \mathbb{R}^{9 \times 1} \rightarrow \mathbb{R}^{3,3},$$

$$T = vec_{3,3}^{-1}(vec(T^*)).$$

3.9 Warping perspective

Performs perspective warping of the source image using the given transform coefficients⁷. Let denote each point in T matrix as:

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}$$

Transform the source image to the output image using the pixel coordinates (x, y) bordered by a plate size⁷:

$$x := \frac{(t_{11} * x + t_{12} * y + t_{13})}{(t_{31} * x + t_{32} * y + t_{33})}, \quad y := \frac{(t_{21} * x + t_{22} * y + t_{23})}{(t_{31} * x + t_{32} * y + t_{33})}$$

3.10 Rescaling

Rescaling and converting output matrix for future operations.

$$Y = \lfloor Y * 255 \rfloor$$

3.11 Denoising

The principle of the first denoising methods was quite simple: replacing the color of a pixel with an average of the colors of nearby pixels [1]. Denoising of the Y matrix happens with high filter strength regulation parameters for luminance components.

$$Y_n := \frac{1}{C_n} \sum_{q \in B(n,r)} Y_{n^*} * w(n, n^*)$$

, where:

$$C_n = \sum_{q \in B(n,r)} w(n, n^*),$$

$B(n, r)$ indicates a neighborhood centered at n and with size $(2r + 1) \times (2r + 1)$ pixels and the weight $w(n, n^*)$ depends on the squared Euclidean distance $d^2 = d^2(B(n, f), B(n^*, f))$ of the $(2f + 1) \times (2f + 1)$ color patches centered respectively at n and n^* [1].

3.12 Eroding

To highlight the intensity of individual characters. The minimal pixel value overlapped by a 5×5 kernel is replaced to all image pixels under the anchor point with that minimal value⁸. For $n \in \langle 0, (h_o - 5) * (w_o - 5) - 1 \rangle$:

$$Y_{2*w_o+(n+2)} := \min(\bigcup_{j=\lfloor n/w_o \rfloor}^{\lfloor n/w_o \rfloor + 4} \bigcup_{i=n}^{n+4} Y_{i+(j*w_o)})$$

⁷ https://scc.ustc.edu.cn/zlsc/sugon/intel/ipp/ipp_manual/IPPI/ippi_ch12/funcn_WarpPerspective.htm

⁸ https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

3.13 License Plate Recognition (LPR)

Tesseract is an OCR engine that was developed by HP. In late 2005, HP released Tesseract as open source software and now is maintained by Google.

Processing follows a traditional step-by-step pipeline:

Y \longrightarrow **Adaptive thresholding** \longrightarrow **Connected component analytics** \longrightarrow **Word Recognition** \longrightarrow **Classification**

3.14 Adaptive thresholding

Several methods can be used to compute the adaptive threshold value of the Y matrix. Tesseract uses *Otsu* method [2].

3.15 Connected component analysis

Connected components labeling is used to assign each region a unique label, enabling the individual objects to be distinguished, and extracts a set of features of the object represented by the region to classify each region into one of two or more classes [4].

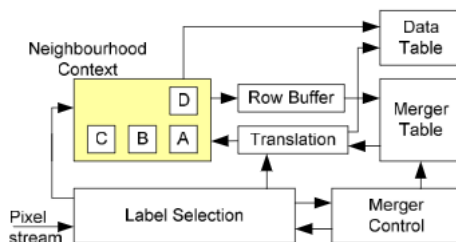


Figure 6: Connected components analysis architecture [4].

3.16 Word recognition

Then, the outlines are converted into Blobs. Blobs are organized into text lines, and the lines and regions are analyzed for some fixed area or equivalent text size. Text is divided into words using definite spaces and fuzzy spaces [5].

3.17 Classification

Classification proceeds as a two-step process.

1. **Class pruner** - creates a shortlist of character classes that the unknown might match. Each feature fetches, from a coarsely quantized 3-dimensional lookup table, a bit-vector of classes that it might match, and the bit-vectors are summed over all the features. The classes with the highest counts (after correcting for expected number of features) become the short-list for the next step [10].

2. **Configuration** - the distance calculation process keeps a record of the total similarity evidence of each feature in each configuration, as well as of each prototype. The best combined distance, which is calculated from the summed feature and prototype evidences, is the best overall the stored configurations of the class [10].

4 Metrics

No. input images	tests passed	average time
100	82	3.42687 sec

- No. input images - Total number of random car/motorbike images with visible car plate which include:
 - 80 White Polish car plates
 - 5 Yellow Polish car plates
 - 10 White Polish motorbike plates
 - 5 Unconstrained Polish car plates
- tests passed - Number of input images that the program execution output matches the license plate characters defined by a human (me).
- average time - the average time of batch execution 100 images on Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz

5 Acknowledgements

Special thanks to S.M. Silva and C.R. Jung for their efforts in making ALPR in Unconstrained Scenarios⁹.

OpenCV³, Numpy¹⁰, Pytesseract¹¹, and Python¹² contributors for overall open-source activity. HP¹³ for Tesseract¹⁴ creation, and Google¹⁵ for maintaining it.

⁹ <https://github.com/sergiomsilva/alpr-unconstrained>

¹⁰ <https://numpy.org/>

¹¹ <https://github.com/madmaze/pytesseract>

¹² <https://github.com/python>

¹³ <https://www.labs.hpe.com/>

¹⁴ <https://github.com/tesseract-ocr/tesseract>

¹⁵ <https://about.google/intl/en-GB/>

References

- [1] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. “Non-Local Means Denoising”. In: *Image Processing On Line* 1 (2011), pp. 208–212. DOI: [10.5201/ipol.2011.bcm_nlm](https://doi.org/10.5201/ipol.2011.bcm_nlm).
- [2] Nilanjan Dey et al. “Adaptive thresholding: A comparative study”. In: July 2014. DOI: [10.1109/ICCICCT.2014.6993140](https://doi.org/10.1109/ICCICCT.2014.6993140).
- [3] Dan Kalman. “A singularly valuable decomposition: The SVD of a matrix”. In: *College Math Journal* 27 (1996), pp. 2–23.
- [4] Ni Ma, Donald Bailey, and Christopher Johnston. “Optimised single pass connected components analysis”. In: Jan. 2009, pp. 185–192. DOI: [10.1109/FPT.2008.4762382](https://doi.org/10.1109/FPT.2008.4762382).
- [5] Chirag Patel, Atul Patel, and Dharmendra Patel. “Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study”. In: *International Journal of Computer Applications* 55 (Oct. 2012), pp. 50–56. DOI: [10.5120/8794-2784](https://doi.org/10.5120/8794-2784).
- [6] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *arXiv preprint arXiv:1612.08242* (2016).
- [7] Seyed Hamid Rezatofighi et al. “Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression”. In: *CoRR* abs/1902.09630 (2019). arXiv: [1902.09630](https://arxiv.org/abs/1902.09630). URL: <http://arxiv.org/abs/1902.09630>.
- [8] Rasmus Rothe, Matthieu Guillaumin, and Luc Van Gool. “Non-Maximum Suppression for Object Detection by Passing Messages between Windows”. In: vol. 9003. Apr. 2015. DOI: [10.1007/978-3-319-16865-4_19](https://doi.org/10.1007/978-3-319-16865-4_19).
- [9] S. M. Silva and C. R. Jung. “License Plate Detection and Recognition in Unconstrained Scenarios”. In: *2018 European Conference on Computer Vision (ECCV)*. Sept. 2018, pp. 580–596. DOI: [10.1007/978-3-030-01258-8_36](https://doi.org/10.1007/978-3-030-01258-8_36).
- [10] R. Smith. “An Overview of the Tesseract OCR Engine”. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. Sept. 2007, pp. 629–633. DOI: [10.1109/ICDAR.2007.4376991](https://doi.org/10.1109/ICDAR.2007.4376991).