Computational Techniques of Detection, Estimation and Identification 2021-2022

Research project

José Pedro Araujo Azevedo
Pedro Francisco Fermeiro Gonçalves

1. Introduction

This report describes the procedures, analyzes and conclusions obtained in carrying out the project of Computational Techniques of Detection, Estimation and Identification, which consists of the recognition of QR codes.

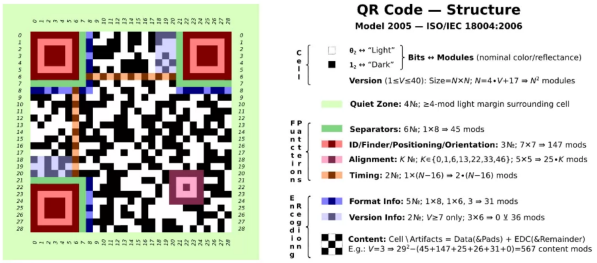

Figure 1: ”Anatomy” of the QR code.

The three large squares in the corners are for the sensor to position itself. The rest of the cells contain information, alongside error correction coding.

The more error correction a QR code has, the less data it can store and incidentally, the more data that is stored within a QR code, the more squares it will have. More squares are also required as the level of error correction is increased and artistic QR codes can forego data capacity for aesthetics by creating an image with some of the blocks[1].

In this project it was decided to approach different methods for its realization.
Histograms of Oriented Gradients, matlab pre-made algorithms, was created by Pedro Gonçalves.
Opencv aproaches and neural networks, was carried out by José Azevedo.

2. Matlab using ”readBarcode” function

2.1. Procedure

In a first phase, we analyzed the entire dataset using the Matlab *readBarcode* function, which is a QR code detector included in Matlab libraries, which analyzes whether the QR codes we had in the dataset could be read. After doing the *verification_readBarcode* code, it analyzed the entire dataset and made the results available in a table (which can be found in the annex), with the characteristics of each image (Type, Name, Condition, and Quality) and if a reading was successfully performed , or not, taking the value of 1 in the positive case and 0 in the negative case.

2.2. Results obtained

	1 Tipo	2 Condição	3 Qualidade	4 Nome	5 Leu o QR
1	'Digital'	'-'	'-'	'S-fundo-br...	1
2	'Digital'	'-'	'-'	'S-fundo-p...	1
3	'Digital'	'-'	'-'	'S-fundo-p...	1
4	'Digital'	'-'	'-'	'S-fundo-p...	0
5	'Digital'	'-'	'-'	'S-fundo-p...	1

Figure 2: Final product, using ”readBarcode” function.

After analyzing the table in question, we can see that the results were not very positive because in 84 images available, the code was only able to recognize the QR code in 4 of them, all of which belong to the digital format. In all other cases, whether on plain paper, on photographic glossy paper, on Rough Paper, in a dark light condition, low light condition, or in normal light conditions, with good or poor printing quality, the algorithm was not able to recognize any QR code presented in the remaining images.

3. Histogram of Oriented Gradients

3.1. Brief Introduction

Histogram of Oriented Gradients is a feature descriptor that is often used to extract features from dataset containing images. This descriptor focuses on the structure or the shape of an object, and is able to provide the edge direction. This is done by extracting the gradient and orientation (commonly known as magnitude and direction) of the edges. This is done in localized portions, this means that the image is broken down into smaller regions and for each region, the gradients and orientation is calculated. Finally, the HOG generates a Histogram for each of these regions separately.

3.2. Calculating Gradients

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Figure 3: Example of pixel values in an image.

The gradients are the small change in the x and y directions, and this is done by taking pixel values, and to determine the gradient in x-direction we need to subtract the value on the left from the pixel value of the right, and similarly, to calculate the gradient in the y-direction, we subtract the pixel value below from the pixel value above the selected pixel.

Then we end with two matrices, one for the gradients in x-direction and one for the gradients in y-direction, where the magnitude would be higher when there is a sharp change in intensity, such as around the edges. This process is repeated for all the pixels in a image.

3.3. Calculate the Magnitude and Orientation

Once we have the gradients, we can determine the magnitude and direction for each pixel value. Magnitude is calculated by

$$\sqrt{(G_x)^2 + (G_y)^2} \tag{1}$$

and the orientation is calculated by

$$\tan\phi = \frac{G_y}{G_x} \tag{2}$$

where ϕ is the angle corresponding to the orientation. After doing this we now have, for every pixel value the total gradient and the orientation, needed to generate the histogram.

3.4. Expected Results

Initially, it was thought to use the extractHOGFeatures() function to extract the features of the QR code. After having these, the idea would be to go through the rest of the images within a cycle.

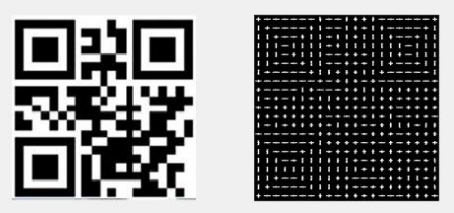

Figure 4: Hog features.

Each image would be divided into blocks, and in each of these blocks the HOG features would be extracted. Then we would compare these features with those of the QR code. If these were equivalent according to a certain metric and a threshold, the existence of the QR code was confirmed.

In addition to this method, two more methods were found in Matlab libraries (HOG , hog_feature_vector).

4. Pre-made algorithms in OpenCV

4.1. Using QRCodeDetector() function

This function detects and decodes QR code.
Unfortunately it was only possible to detect in the digital images(4 images).

Figure 5: Final product, image detected: Selos-Descaracterizados-Digital.

4.2. Using pyzbar module

In this section, we went through all the images and applied the detection algorithm in 4.2.1.
In 4.2.2. the webcam was pointed at the screen and the images were scrolled.

4.2.1 Importing images

The results were identical with the addition that it was possible to detect some images in normal room light and the digital ones, a total of 7 images.

Figure 6: Final product, image detected: Photographic-Glossy-Paper-NRL.

4.2.2 Using Webcam

It was decided to test with the webcam as well and the results were better since it was possible to detect most of the images, except for those that had little brightness.

Figure 7: Final product, image detected: Paper-Normal-Not-so-LL-HQ.

4.3. Conclusion

Obviously these algorithms work ”well” on digital images and are not designed for images under low light effects, blur, etc.

In this section, we also tried to manipulate the dataset images using the ImageEnhance PIL. Adjusting brightness, contrast and sharpness however the results were identical to those reported.

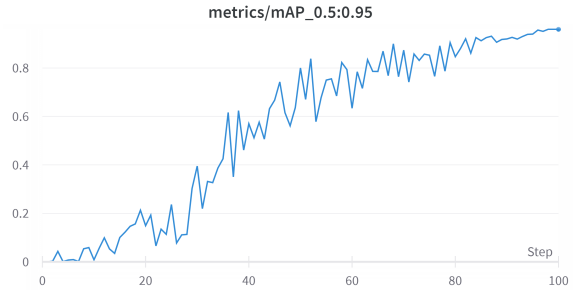

Figure 10: mAP graph(mean average precision).

5. Neural Networks: YOLOv5

5.1. Yolo overview and pre-train notes

YOLO, meaning ”You only look once”, is an object detection algorithm that divides images into a grid system. Each cell in the grid is responsible for detecting objects within itself. Yolov5 has pretrained checkpoints for diferent architectures, we used ”yolov5l”.

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Figure 8: Diferent pretrained checkpoints avaliable.

To perform a custom training, it is necessary to have the annotations of the entire dataset. We made use of a tool named ”roboflow” in which we annotated the 84 images from the dataset provided. One of the many advantages of yolov5 is that many transformations are applied to the dataset, such as: rotations, scale, blur. This enriches the dataset and brings better results to the model, avoiding overfitting.

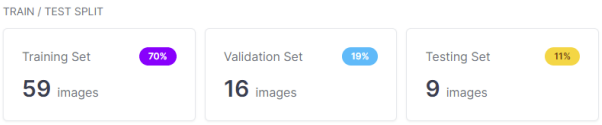

Figure 9: Dataset division.

5.2. Results obtained

The ”wandb” tool was used to analyze the performance of the model.

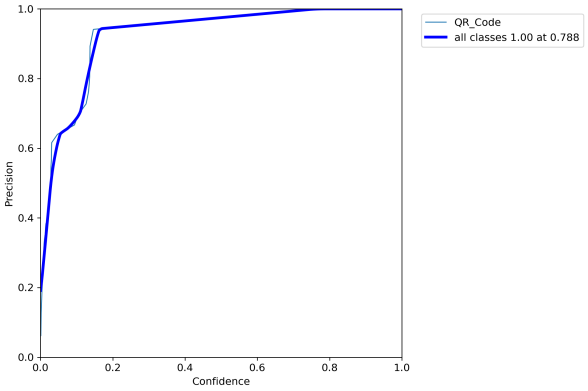

Figure 11: Precision curve.

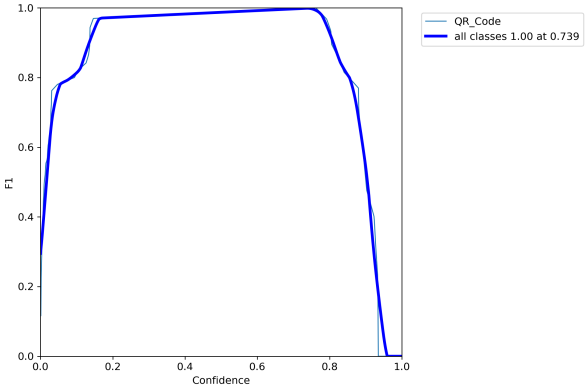

Figure 12: F1 curve.

From the F1 curve, the confidence value that optimizes the precision and recall is 0.739. Starting at about 0.8, the recall value begins to suffer, and the precision value is still roughly at the maximum value.

These results are considered quite good, since the classifier was able to identify the Qr code in all images.

Figure 13: results predicted from the validation batch.

5.3. Conclusions

The results obtained in neural networks were quite good, and the yolov5 network does data augmentation, it is possible to identify the QR code in all images, whatever the condition.

Note: Results in more detail attached.

6. Overall conclusions

Detecting and reading QR codes in images taken under adverse conditions (plain paper, photographic glossy paper, rough paper, etc.) is a complicated task, especially in different lighting conditions.

Functions already created (Matlab) cannot detect QR codes under these conditions, with the exception of digital images. However, in openCV it was possible to detect and read the QR code in digital images and some in normal light conditions.

With HOG, it was intended to improve the detection of the Qr code in the dataset, as explained in section 3.4. However, this could not be verified.

Using Neural Networks(yolov5l), performing a training of 100 epochs, it was possible to detect the QR code in all adverse conditions.

References

https://www.techspot.com/guides/1676-qr-code-explained/

Matlab readBarcode
https://www.mathworks.com/help/vision/ref/readbarcode.html

HOG FEATURES

https://www.mathworks.com/matlabcentral/fileexchange/63609-hog-features?s_tid=srchtitle_hog_2

https://www.mathworks.com/matlabcentral/fileexchange/28689-hog-descriptor-for-matlab?s_tid=srchtitle_hog_5

https://www.mathworks.com/matlabcentral/fileexchange/46408-histogram-of-oriented-gradients-hog-code-using-matlab

https://www.mathworks.com/help/vision/ref/extracthogfeatures.html

OpenCV

https://opencv.org

https://docs.opencv.org/4.x/de/dc3/classcv_1_1QRCodeDetector.html

https://pypi.org/project/pyzbar/

https://www.geeksforgeeks.org/image-enhancement-in-pil/

Yolov5

https://github.com/ultralytics/yolov5