Final project: QR Image Classification and Localization

Cristian David González Carrillo
National University of Colombia
Bogotá, Colombia
crdgonzalezca@unal.edu.co

Nicolás Restrepo Torres
National University of Colombia
Bogotá, Colombia
nrestrepot@unal.edu.co

Click here to watch Video

***Index Terms*—detection, model, qr, computer vision**

I. Introduction

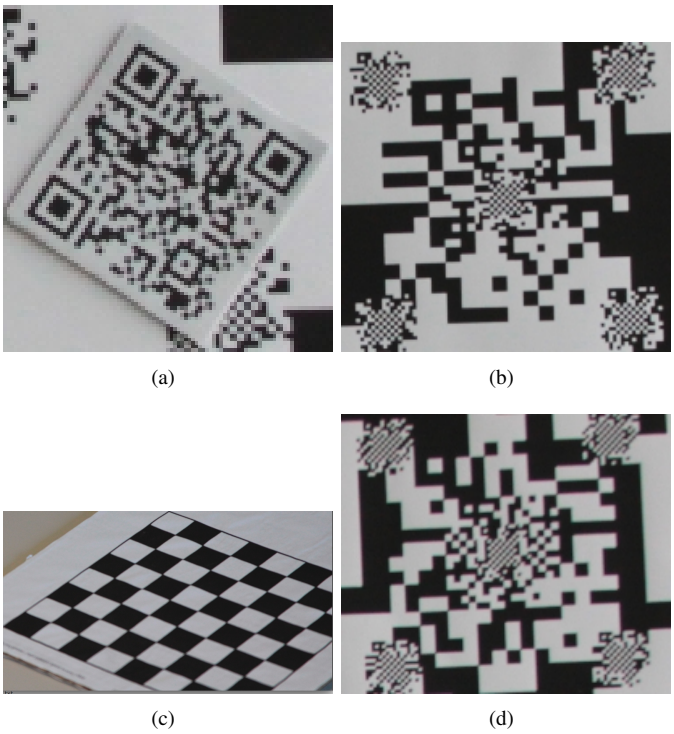
The current project aims to solve a particular problem within the field of Computer Vision (CV) and image processing known as a QR detector. The Quick Response (QR) code was first developed in 1994 by Denso Wave Incorporated, Japan. From that point on, it came into general use as an identification mark for all kinds of commercial products, advertisements, and other public announcements. In scholarly journals, the QR code is used to provide immediate direction to the journal homepage or specific content such as figures or videos. [1]. Nowadays is currently relevant even for medical applications regarding the clinical staff, that could use smartphones with QR codes for contemporaneous access to relevant information to support the Just in Time Learning (JIT-L) paradigm [2]

II. Exploratory analysis

II-1 *Qualitative analysis*

The QR codes contained into the images are in a variety of different settings of illumination, orientation, size, position and quantity. We have checked the dataset while labeling and we can make some assumptions: the dataset contains a group of images we consider will be difficult to detect correctly, given the likeness to a QR code. Thus, the probability of wrong predictions increases. These images are composed by non-QR codes but with similar combinations in color, contrast and frequency to the conditions in which we would find a QR-target image as shown in figure 1.

Now regarding another mixed set of qualitative measures we can see in the figure 2 a sample of the variety of these measures. The illumination goes from bright to dark, it can get mixed with two contrasting level of illumination in the same image. The majority of them are tilted with different angles of inclination in the space that results in images that seems to be in perfect 2D space as well as another that are very close to be just a line. The QR codes can be superimposed in textures that do not correspond to QR-code as well as textures that are QR-codes and finally, there are multiple QR-codes per image.



(a) (b) (c) (d)

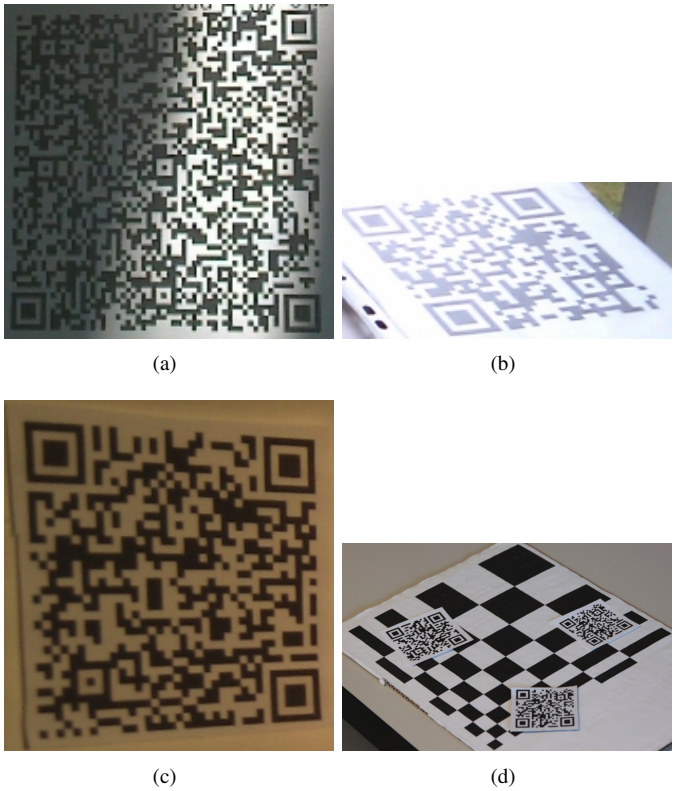
Fig. 1: (a) Mixed brightness (b) Tilted QR (c) Dark illumination (d) QR overlapped with a non QR-like texture

II-2 *Quantitative analysis*

The dataset is composed by a total 96 images with resolutions varying from 768x1024 pixels, being the lowest, and 4752x3168 pixels being the highest. The average size of the images is 3671x2430 pixels and it is around 2350x3750 pixels as we can see in the boxplot of the widths and heights of the image in the figure 3.

The figure 4 shows the distribution $width$ and $height$ of the QR boxes accordingly to the YOLO format. This data is very scattered in all the possible positions and this is beneficial to the model for not over-fitting into the position of the bound box, but we strongly believe that this will not affect the chosen model.

In Figure 5 we can see that the annotations are not as sparse as the width and height of the images ranging from a median



(a) (b)

(c) (d)

Fig. 2: (a) QR-Code (b) (c) (d) Non QR-Codes

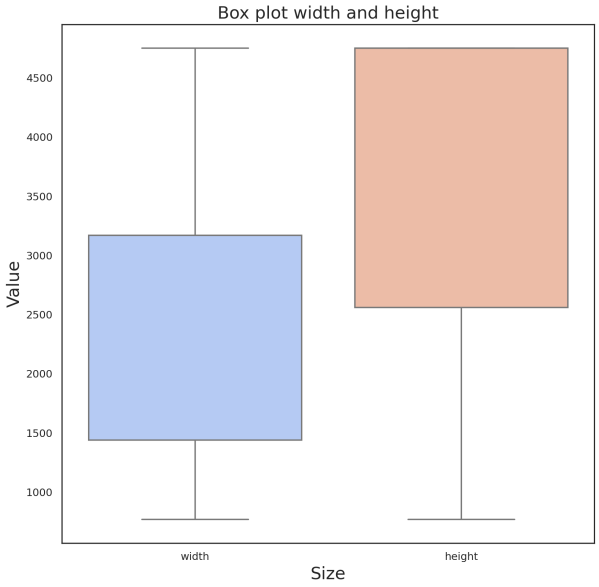


Fig. 3: Box plot width and height of dataset

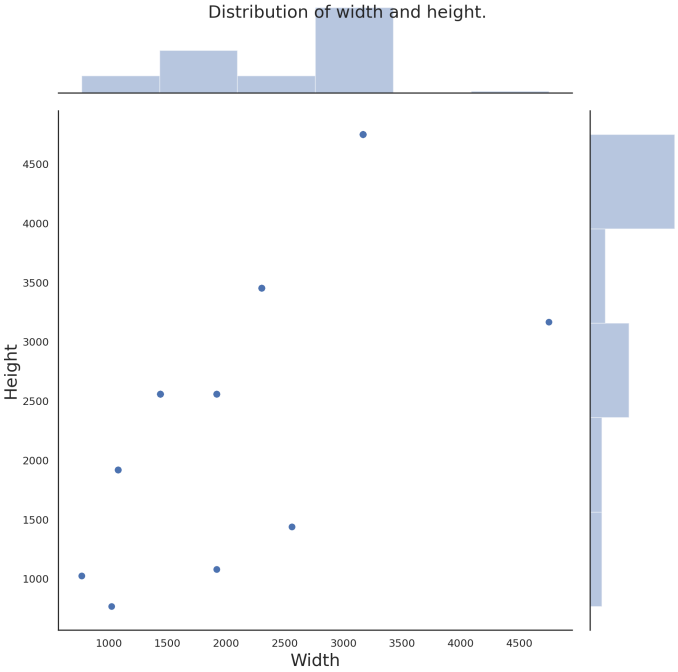


Fig. 4: Distribution of images

from 300x500 pixels to 700x1350 pixels being the biggest. This is a double edged sword to the model, since the more the QR are similar in size, the easier is to find patterns when the dataset is not as big, but this can be a overfit to a more diverse of dataset with different size of the boxes that we can expect to predict.

Additionally, we can see the sizes of the bounding boxes measured in width in height in the figure 6. This is very important in advance to decide the aspect ratios width-height since that will be used in principle as part of the anchor boxes to detect multiple QR in a image given the former data by applying a simple lineal regression.

But taking a closer look to the scatter we see that the best approach for getting these anchors is by applying a *k-means* algorithm, where the number of anchors matches the number of centers and the width and the height gives the algorithm initial parameters, the result can be seen in the figure 7

III. Project plan

For starters, it is important to take into account that some assumptions that worked for other problems not necessarily hold for the current project or the current problem within the field image classification, so we will check thoroughly about common approaches prior to the project, learned in the course. These set of ideas must be followed to address common problems that could arise in developing this detector. For this we follow the suggested pipeline of work, that is, starting with a basic sketching idea about the architecture of the model to use, implementing it to check the results and finally gain experience and clean up the details of the idea. Also we want to focus in the following points:

A. *Exploration of predefined architectures*

Since the objection detection is a matter of great importance in the computer vision field as mentioned before, we plan to

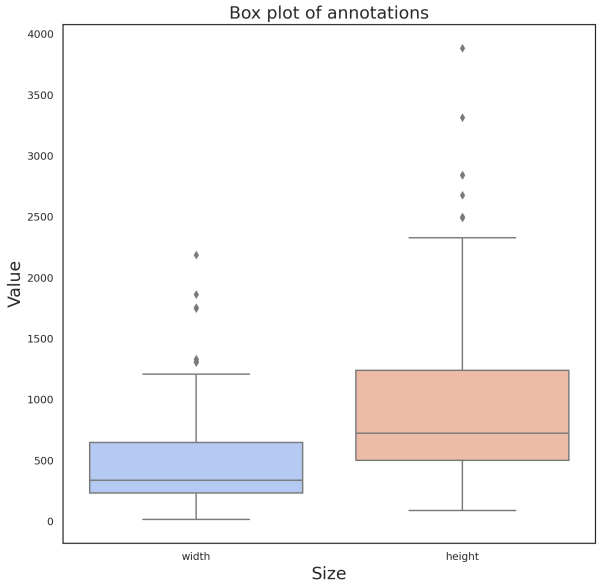


Fig. 5: Box plot of generated boxes

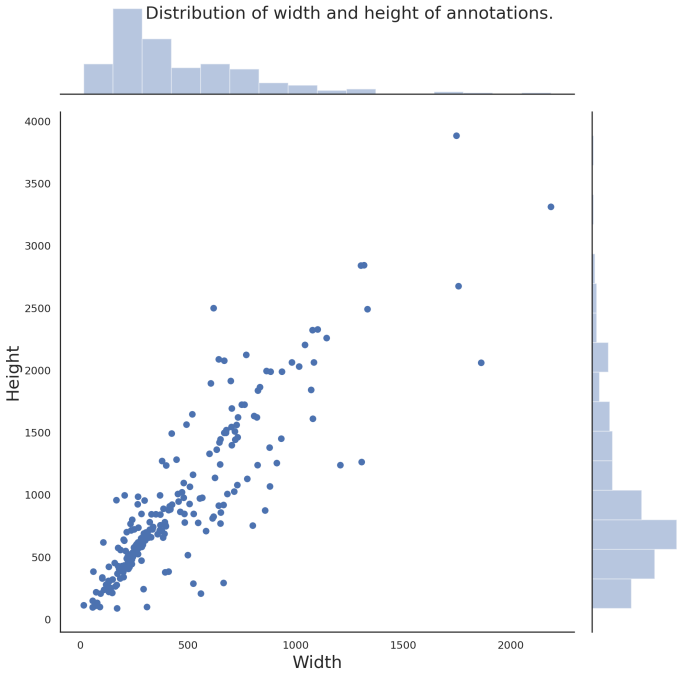


Fig. 6: Width and height distribution of generated boxes

iterate over the already existent weights and architectures that have led to results, we think that this is the expected approach for the project since we need to get results in the most fast and reliable way

III-A1 *Exploration of hyperparameters*

The first of the assumptions that must be checked as well, refers mainly to the exploration of hyperparameters, this is,

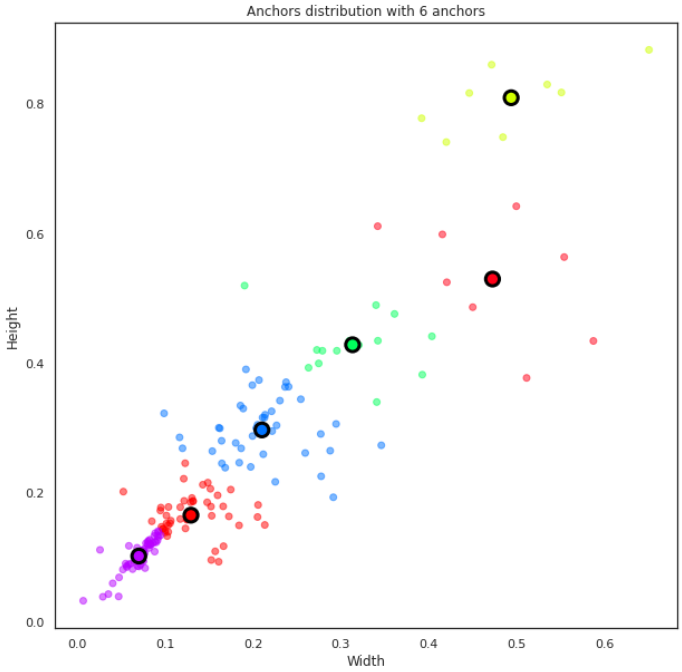


Fig. 7: K-means applied to get anchors

getting the right constraints for the model and architecture chosen in order to maximize the single metric of evaluation (mentioned in detail in other section) in which the model is performing currently. We start with the basic set of hyperparameters provided by the model that we are basing on (given the starting architecture that will be discussed later on). These first set of hyperparameters described will be generally the first model to test in a final table of comparison (table I) and we plan on iterating on the rest of the tuneable parameters with an approach of orthogonalization, controlling the changes to see what happens in the clearer way possible in the model.

III-A2 *Data splitting*

Generally in Machine Learning the data is divided into the following subsets:

- Training set
- Hold-out cross validation set: development set
- Test set

We know that the way of handling data when is not as numerous, is by using the rule of thumb of splitting the sets like this correspondingly to the list above: 60% - 20% - 20%. But since we do not have prior access to the training data we have to decide just upon the development or validation set. We just need to decide how to split the 100% of the currently acquired data and we consider the following as a great start: 70% - 30%. Later on, when we get the test dataset the splitting will encompass all the datasets.

III-A3 *Checking data distribution*

The images that will be part of the training and development set comes from the same distribution in principle, since apparently the photos has been taken with the same camera and mostly of them has uniform resolutions (see exploration

analysis). We can conclude from this that everything in the train and validation sets comes from the same distribution even if we apply a technique of data augmentation to get the most of the images in diverse transformations. The ideal case in which the models performs is by assuming that the test data comes also from the same distribution, but since this is unknown, we add images from different distribution as negatives and positives to reinforce the characteristics of the images that are not being recognized well when we will carry out error analysis. Even more important, if we find that the validation dataset that will be published later comes from a totally different distribution, we are ready to shuffle the images accordingly so that the model can train itself in the distribution of the given validation dataset.

III-A4 *Addressing mismatched data*

This is a common problem with bigger datasets that must be thought in order to decide the better trade-off solution in terms of the context of the dataset regarding some mismatch data. This is done generally through an error analysis, but since our dataset is limited to less than hundred images, this can be done manually and prior to the any processing of the images. The trade-off here is to check carefully into mislabeled data and correct it. And this is accomplished by something particular about this project in which we play the role as "labellers" as well, so this trade-off is managed entirely for our team. The problem with mismatched data that concern us is all the images that could harm the model (since it is possible that is enormously sensitive to small changes in the chosen bounding boxes), by not having exactly bounding boxed or that are extremely tilted to one side, see figure 8 (a). Not only the QR-codes happen to be small but crossed by a handrail.

Other problem to address here is to set a threshold in which we decide if a QR code is a valid one when part of it is cropped as we can see in the figure 9, due to the fact that a half of a QR possibly can not carry out all the information that is needed to decode its information and because the other exactly part of the unseen QR can be a totally different thing. We decide initially to set a threshold of the half, if more than the half of the QR is a valid one.

III-A5 *Model decision*

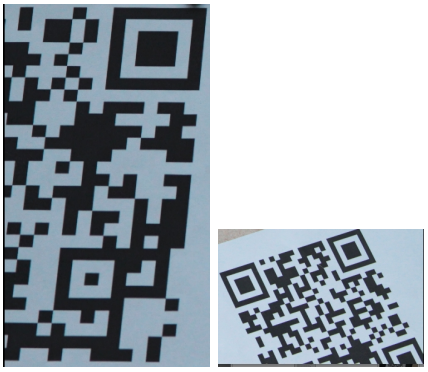
We know that the most common architectures within the image processing field in the current state of art includes basically these models:

- Different versions of YOLO: You only look once, algorithm
- Sliding windows algorithm with and without a convolution implementation
- RCNN: Region Convolutional Neural Network
- Faster variations on RCNN given more effective ways to segmentate the image

We plan to iterate first on YOLO and its variations, the different versions of it. The reason for this apriori is the theory researched prior to this project, the sliding windows algorithm is a implementation based on a conventional grid that will be the most brute force approach just with an additional optimization that is a convolutional implementation based on

(a) (b)

Fig. 8: (a) Grouped QRs (b) Tilted QRs



(a) (b)

Fig. 9: Accordingly to the 50% threshold: (a) Not valid QR (b) Valid QR

what is called in the state of art as 1-convolution filters. RCNN and its faster variations are based on segmentation approaches that make use of all the computational power for performing this task, this segmentation tries to delimit any possible objects in a image and can be convenient when we are trying to detect multiple classes, but this is not the case.

III-A6 *Improving architecture*

The problem to address has certain components that must be looked into, one of them is the anchor boxes. The anchor boxes that helps us to determine more than two objects in the same sub-division of the grid in the specific implementation of YOLO algorithm comes with a predefined boxes that are

too small for the problem, but we know already from the exploration analysis this can be done with a k-means algorithm. A stricter *non-max suppression* criteria in the training phase could also help the model to ignore the most difficult images. The regularization L2 must be applied somehow to try to avoid high variance of the model and other problems such as exploding gradients, vanishing gradients we assume to be addressed by the selected model. The inputs will be normalized.

III-A7 *Transfer learning*

This probably the most direct approach after realizing that the dataset is very small, given the desired distribution. We don't know for sure which specific architecture of YOLO to use or which pre-trained weights could be suitable for the needs of this problem for example, but we know that we will need some kind of transfer learning to get the most out of the dataset. This could be via a fine-tuning in which just the last layers of a big network are trained or a specific set of layers or by training every layer based on the specific pre-set of weights. This will be described in detail in the section *model details*

III-A8 *Evaluation metric*

It is important to set a metric from the beginning of the project to assure consistent results along all the experiments proposed. The metric associated with this kind of detection is and the one that we chose as single evaluation metric is the **Average Precision and Mean Average Precision (mAP)**, because combines both precision and recall together. mAP is the mean of the AP calculated for all the classes and will be the **optimizing** metric. This is not to be confused with the standardized set of metrics that we will be using to present the result that are graphs of precision versus recall metrics. The precision and the recall will be the **satisfying** metric with a minimum threshold of 80%.

III-A9 *Human-level performance*

The analysis of this error, called base or Bayesian error gives us a notion about how much avoidable bias we have into consideration of the model and given the initial labeling of the dataset and comparing between results in our same group we can conclude that is very low, we are capable to distinguish QR codes normally from other type of QR Codes such as: DataMatrices, Aztec codes, Trllcodes, Shotcodes, mCodes, Beetagg codes and others.

III-A10 *End-to-end approach*

The hand-made engineering approach in this case would be to have a recognizer of squares and combining it with the specific specification of encodings within a QR code, but since we already discussed how we are able to use the best approach that is known to this moment, that is YOLO, this will imply or rebuilding the model from scratch and try to build different detectors for it. For this we decide to rather perform an end-to-end approach that will work like a black box recognizing directly QR codes and putting the correct bounding boxes.

III-A11 *Error analysis*

After the labeling there is some doubts about which other QR should be ignored, which threshold to set of the minimum appearance of the QR code in the screen or what to do with the images that were shown in the data exploration analysis. Since the dataset is small and the images are crucial to the final predictions, we plan to make thoroughly error analysis and re-labeling the images accordingly. This can make no harm to the model and set the efficiency at the highest possible.

IV. Model details

A. Chosen version

We have decided to use YOLOv3 for this project. This is because is a very popular network with a lot of resources to use and still simple for beginners in this kind of models like this. [3].

The first versions of YOLO struggled with small objects. However, with YOLOv3 there is a better performance for small objects, and that because of using short cut connections that are common in residual networks [4]. However comparing to the previous versions, YOLOv3 has worse performance on medium and larger size which will be important to consider in the error analysis.

B. How YOLO works

Since the classification and localization network can detect only one object that means any grid cell can detect only one object. Because of this grid idea, there are some problems that are addressed with non-max suppression (a technique that is not described in detailed here) such as a maximum number of predictions per grid, multiple objects in the same grid and a object being of enough size to be contained in more than one squared of the grid. Non-max suppression and YOLO generally describes the false positives, true positives and false negatives with a metric called Intersection of Union. There is no true negatives because that would mean that the model is able to detect where there is not an object and this does not make sense. Since the ground truth boxes are selected by us we are 100% sure that there is and object inside the ground truth box, accordingly any box with high IOU with the truth box will also surround the same object, then the higher the IOU, the higher the possibility that an object occurs inside the predicted box.

C. Architecture description

This network is inspired by the GoogleNet model for image classification, but instead of the inception modules used by GoogleNet, accordingly to a file referenced by the creator of the darknet fork that will be explained later [5], the real YOLO version 3 model has 106 layers in which we will be training 24 layers in the tiny version and 75 layers in the full version, each of the distribution from filters, channels, type of layers among others can be consulted in the references as was cited here.

IV-C1 *YOLO Loss function:*

This model uses sum-squared error **SSE** for the loss function because it claims that is easier to optimize, it is described in multiple parts

- Localization loss:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \tag{1}$$

- Confidence loss

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \tag{2}$$

- Classification loss

$$\sum_{i=0}^{S^2} 1_i^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \tag{3}$$

So finally the loss function is just the sum of each of the temps. To describe briefly what is is loss function calculating it is important to mention that:

- The loss function penalizes classification error only if there is an object in that grid cell.
- Since we have B=2 bounding boxes for each cell we need to choose one of them for the loss and this will be the box that has the highest IOU with the ground truth box so the loss will penalizes localization loss if that box is responsible for the ground truth box.
- SSE weights localization error equally with classification error which may not be ideal.

V. Experimental setup

A. Repository

We store all the important inputs to the model in a repository hosted in GitHub to allow centralized access to it from the notebook, this includes:

- **assets:** Contains all the images and other assets used for this report and as output of the trained models.
- **darknet_conf:** This contains the mandatory files to start and run a experiment in our current settings such as: 1) paths to the list of paths that contain datasets of training, validation and testing; 2) parameters of configuration for the pretrained network in its tiny version and its full version; 3) a classes file 4) A configuration file necessary for darknet network, among others files and configurations.
- **dataset:** contains the original images provided for the project plus an additional set of images marked with the name "own_x" where x is a number. This also contains the respective annotations for them
- **model_data:** contains all the paths to the train and data set that was mentioned in the *darknet_conf* section.
- **negative_dataset:** contains images with empty annotations mentioned previously to help the model to discard false positives.
- **augmentation_dataset:** contains images with augmented images in a series of transformations mentioned previously to help the model to cope with bigger images better.
- **utils:** script that has all the utilities used throughout the model such as: resizing of annotations, loading of annotations, splitting of dataset, scaling of anchors, among others.

B. Annotations on images

We used *labelImg* (see experimental setup) to label images accordingly to YOLO format provided by the software into text files in the same folder as images with same name. A file named "classes.txt" is saved with the single class "QR". YOLO uses relative values rather than raw pixel values. In other words, the format desired is the following:

```
<class_id> <center_x> <center_y> <width> <height>
```

Where each center refers to the percentage of the width. In other words, for the center-x if the image is 800px wide, and the center-x is at 400px, the center-x would be written as 0.5. Also, in YOLO we do actually need them all to be separate .txt files, rather than in one big long file, so this processing is made in the utils script that load the annotations provided by default by *labelImg*

C. Data splitting

We split the 96 images into three subsets: Training, Validation and Testing in a basis of 70% for training, 20% validation and 10% for training. We have chosen this as as stated before, is a quite used method in Machine Learning due to the small size of dataset.

D. Darknet

Darknet is one of the better forks of the implementation of YOLO repository [3] and a tutorial notebook [6] provided by various authors and hosted by AlexeyAB as implementation of the version of YOLO in its third version. We decided to only use the tiny version using already trained weights with the COCO dataset. To perform all the trainings it was necessary to use a GPU-capable environment, for that we used *Google Colaboratory* as this provides $GPUTeslaP100 - PCIE - 16GB$ for free.

E. Google Drive

To backup all the results in real time, we created a shared unity in Google Drive and created a symbolic link to access it from any of our accounts. As per the process of developing the model, we train several model (experiments) we generate some logs and backups and this get saved into folders that are marked with the number of the experiment.

F. Fine-tuning training

For performing the training we use the file *yolov3-tiny.cfg* and we change the crucial hyperparameters from here (see table I to alter the performance of the model. There is a *freeze* parameter in that allows us to control how many layers in the model are trainable and will be change given two options:

freeze whole darknet53 body (that is trained on other dataset) or freeze all but 3 output layers.

G. Anchors generation

Using the algorithm of k-means learned in class we get the anchors for a given input size of the images that is another hyperparameter. A graphical result of this is in the Figure 7.

H. Results validation

For this, we needed to test the images against each dataset and for this we set the mini-batch size to 1 and the subdivisions to 1. Also it is possible to set a different set of hyperparameters, specifically the image size, which is changed to 832x832 image as this allows the model to predict smaller images in the validation performing. This is done and will be considered in the results. After this, we use a repository which will help us to calculate automatically the different metrics as Precision vs. Recall curve [7]. In order to generate the data for precision and recall, we generate two sets of files required for the author of the repository: ground truth files, where we indicate the real boxes that should have been detected in a slightly different format:

```
<class> <left> <top> <width> <height>
```

We use a function in our utils converting the annotations to this format. Later we need the detection files, that we parse from the JSON file provided for the darknet prediction as part of the results, from this file we make the same conversion to the same format. After everything is correct in here we get a *results.txt* that contains two lists with the precision vs recall measurements. We perform this for each experiment in which we are interested in (the best ones) and we get the graphs that are in section of results.

I. Summarize of used software and libraries

- LabelImg: To generate a corresponding .txt with the coordinates of the center of the QR and the proportion
- Open CV: library to process images
- Darknet
- Github
- Google Collaboratory

J. Reproducibility

The notebook can be executed to see all the experimental setup running.

VI. Results

We performed some experiments iterating in certain parameters we consider to be the most important ones, these results of all the experiments can be found in Table I. From the table the reader can assume all the parameters from previous models hold for the next models except of course, for the parameters that make the experiment different from the other. The first experiment is not considered as we could not take logs. All the experiments are performed until 2000 epochs, this number comes from the experimental setup where it tends to stagnate.

#Exp	Description of experiment
2	IOU threshold of 0.6
3	IOU threshold of 0.8
4	Image size of 416x416 (Initial dataset)
5	Image size of 608x608 (Initial dataset)
6	Image size of 960x960 (Initial dataset)
7	Image size of 608x608 with anchors updated (Initial dataset)
8	Image size of 832x832 with anchors updated (Initial dataset)
9	Image size of 960x960 + Negative images
10	Image size of 608x608 + Corrected annotations
11	Image size of 608x608 + Negative images
12	Image size of 608x608 + Additional + Augmentation on some difficult images
13	Image size of 608x608 + Negative images + Additional + Augmentation on some difficult images

TABLE I: Performed experiments

We graph the AP obtained as discussed before that is the optimizing metric and the result is in the figure 10. From this we can observe that the changes in other hyperparameters different to the input size of the image are not making big changes in the best model that results from the number of selected epochs. If the IOU does not affect so much the model, that means that the confidence of the boxes obtained generally are ranging in similar values and therefore the non-max suppression is removing the same boxes for different levels of IOU. Another thing to consider is that as the AP curves are often zigzag curves going up and down as we can see in our graph and comparing different curves in the same plot usually is not an easy task. This is due to the curves that tend to cross each other much frequently. That's why Average Precision (AP), a numerical metric, helps to compare our different experiment but another graphs such as precision vs recall are needed.

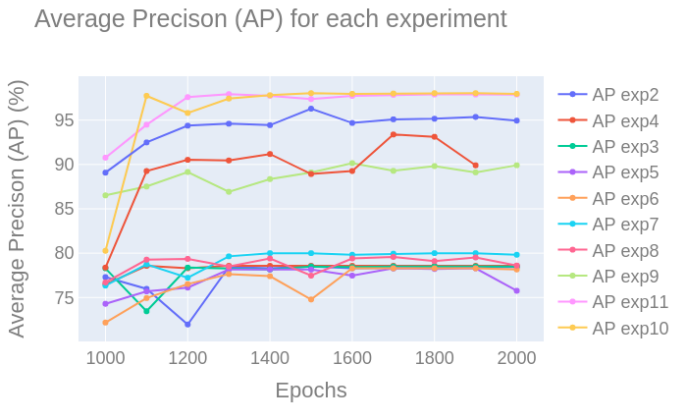


Fig. 10: Average Precision (AP) for each experiment

We graph the losses obtained as discussed before in the loss function that is applied and we can see in the figure 11 that the loss tends to be pretty similar after around 300 epochs, for this we reduce this to lesser epochs and we can see the next graph in the figure 12. We can take from this that the models are able to learn fast, and thus confirming that the loss function described above is suitable for the object detection problem and for this problem in particular.

Now, with the repository for the metrics provided we can see the results of precision against recall for the best three selected experiments: for training in the figure 13, for validation in the figure 14 and for testing in the figure 15.

From these plots we can determine if the model is a good QR detector in particular if in our single class its precision stays high as recall increases, which means that we vary the confidence threshold, the precision and recall will still be high. That's why the precision versus recall curve usually starts with high precision values, decreasing as recall increases.

A. *Best model*

Even though the models of experiments 10 and 11 look quite similar in their results, the 11 looks to be overfitting on

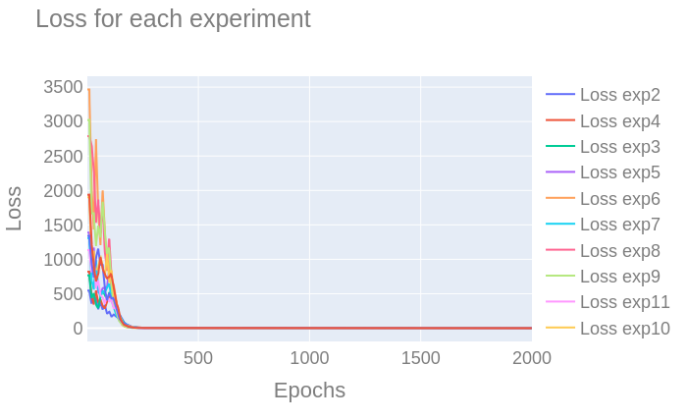


Fig. 11: Losses for each experiment - 2000 epochs

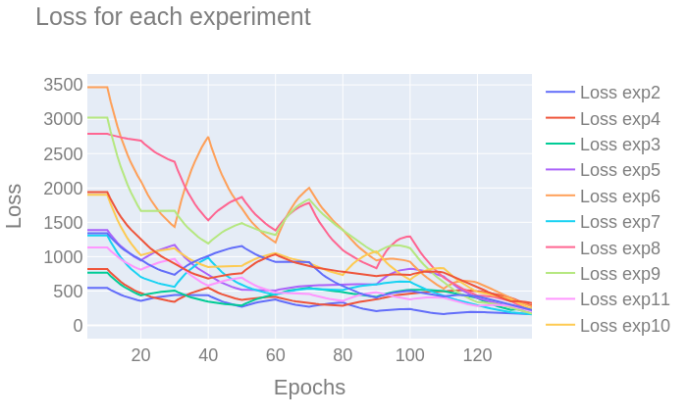


Fig. 12: Losses for each experiment - 150 epochs

training set, that way it is not able to generalize and have as good results for images that it has not seen before. We have the model 10 at the table I. We have chosen this model for the combined results in the three datasets, because following the Occam's razor principle explains the datasets at the best with the simplest explanation since it does not have negatives from other distributions that could add noise in the model and make it more complex. This model performs well in the three scenarios and we can see an example in the figure 16.

VII. Discussion

The problem of object detection is widely known and we consider that the results outperforms the expectations given the fraction of the dataset to the which we had access. If we were to have the dataset contained for a set of smartphones as an example, or just as a brand we will have millions of images with all kinds of different settings, this of course would other different discussion on the scope of privacy but in the field of applying machine learning to this task, there could be seriously big. The reason for the model to performs in the way it did it is because of all the ground research with YOLO, so it is important to acknowledge its weaknesses [8]:

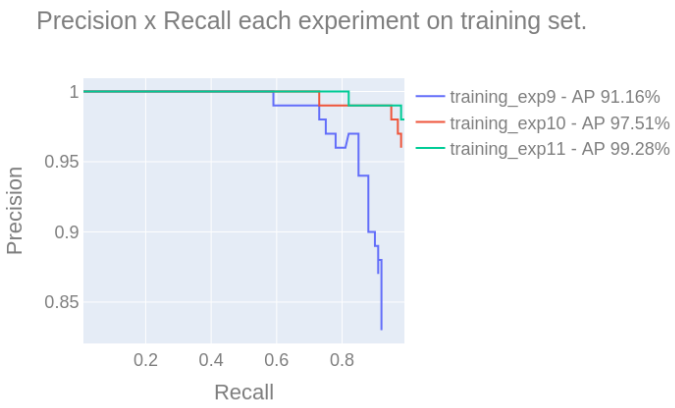


Fig. 13: Precision vs. recall on training dataset

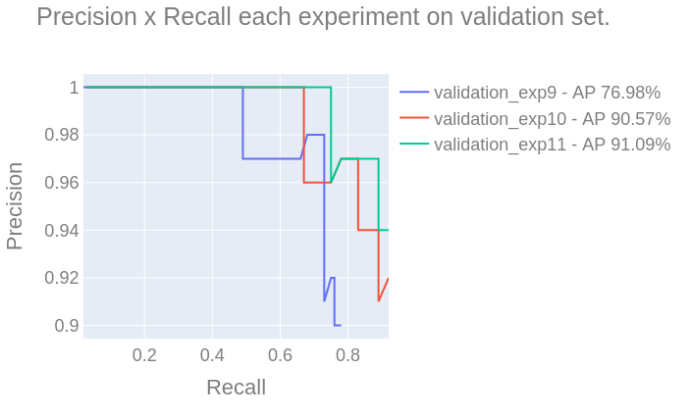


Fig. 14: Precision vs. recall on validation dataset

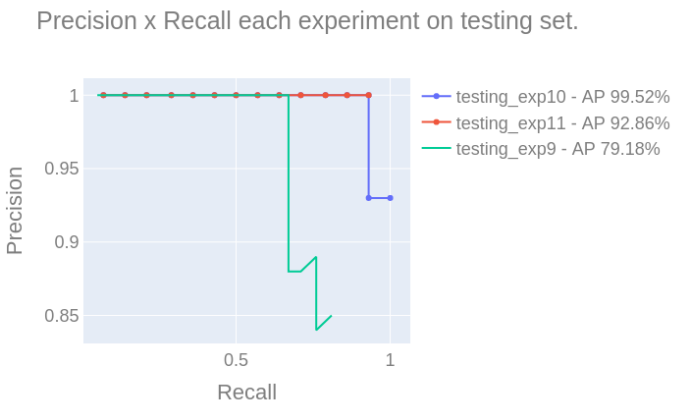


Fig. 15: Precision vs. recall on testing dataset

Fig. 16: Final predictions of example with chosen model

- if the model was trained on large images of an object, it will not be able to detect small images of the same object
- It struggles with small objects that appear in groups, such as flocks of birds.
- It struggles to generalize to objects in new or unusual aspect ratios or configurations.
- The architecture in the current paper is not able to achieve state of the art accuracy.

Also it is to consider that in terms of the final recall and precision results there is another way to identify the performance of the current model and that is by identifying only relevant objects and focusing in the false positives, that would mean a high precision, and finding all ground truth objects that means a high recall. A poor object detector needs to increase the number of detected objects, i.e the false positives in order to retrieve all ground truth objects with a high recall. So we can deliver a different model as a result if we are prioritizing different things such as the precision of the box, or the quantity of correct boxes, in this case we looking for a high recall since it more relevant to find a QR that to find it with a high precision. That is part of the model that has been considered as the best. A future work in improving precision would be mostly done through more augmentation, bigger networks, other YOLO versions, bigger datasets with the same distribution, among others.

VIII. Conclusions

- It is important to decide in the planning phase at best, what will count as the class that will be classified and what is the proportion of the full image that will count within the class.
- The future work of the model would include a bigger dataset from the same distribution taking into account what will be the context in which will be used, bigger computational power with a multi-GPU environment ideally and a bigger network to reduce overfitting or a different version of YOLO.
- Since the distribution of the dataset in the testing dataset is similar to the testing one, the model that does not get to learn other images that are negatives and it is important to make experiment with this, even if the distribution do not match.
- The trade-off between the time of training a lot of models that do not specialize and one model that is comprised

from a bigger network was decided in a lot of smaller models, the risk would be to high for this particular project in which we are asked to deliver results in a determined window frame.

- Since the testing distribution is the one that will be for use in the real context case, we say that the model performs its task and it is successful.
- The weakness on the model are the following: big QR code contained in the images (for the YOLO version used), extremely tiny QR codes and certain textures in certain angles such as the labels of a board box.

REFERENCES

[1] J. H. Chang, "An introduction to using QR codes in scholarly journals," *Science Editing*, vol. 1, no. 2, pp. 113–117, 8 2014.

[2] J. T. Jamu, H. Lowi-Jones, and C. Mitchell, "Clinical education Just in time? Using QR codes for multi-professional learning in clinical practice," 2016. [Online]. Available: http://dx.doi.org/10.1016/j.nepr.2016.03.007

[3] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," Tech. Rep. [Online]. Available: https://pjreddie.com/yolo/.

[4] "YOLO, YOLOv2 and YOLOv3: All You want to know — by Amro Kamal — Medium." [Online]. Available: shorturl.at/sxHUW

[5] "YOLO v3 Layers." [Online]. Available: shorturl.at/pqAP1

[6] "YOLOv3_Tutorial.ipynb - Colaboratory." [Online]. Available: shorturl.at/gxL79

[7] R. Padilla, S. Lima Netto, and E. A. B. da Silva, "Survey on Performance Metrics for Object-Detection Algorithms," 2020. [Online]. Available: shorturl.at/nHKQU

[8] "An Introduction to YOLO (You Only Look Once) – Vipul Vaibhaw." [Online]. Available: shorturl.at/yMNRY