Staff detection with marker from a top-view camera

Iezzi Christian, Incicco Emanuele

***Abstract*—In the world of deep learning, the theme of detection refers to the process of object detection and extracting features within an image. We are going to present a project where the ArUco markers are identified within the images, trying to better use or evolve the already existing deep learning techniques that allow the recognition of the markers only when they are oriented parallel to the camera. Indeed in this project several transformations have been applied to the ArUco in order to create a complete dataset and with this, the YOLOv3 and the YOLOv5 have been trained.**

I. Introduction

In this paper a detection problem is discussed and illustrated. In particular the aim of this article is to examine the task we have dealt with, the techniques used to solve it and the results obtained. The objective was to be able to carry out a staff detection inside a store. The detection had to be carried out through the use of markers and for this purpose ArUco markers were used. In addition, it was supposed to use a camera with a top-view perspective. Various transformations have been made to the ArUco, in order to have a dataset as vast and complete as possible to be able to simulate the positions that the markers could have had in reality, also considering occlusions and variations in brightness. Then the dataset was built using a COCO dataset and then applying the markers. Finally this was provided to neural networks to train them in the detection task. The networks used were YOLOv3 and YOLOv5.

II. Related Works

A. Neural network for object detection

Convolutional neural networks are the standard tool for carrying out object detection tasks, among these the most used are R-CNN, fast R-CNN, SSD and YOLO in its various versions. With the addition of the latest versions, however, YOLO has proven to have superior performance in terms of detection speed, as it can predict objects in real time, and in terms of results that are accurate and with minimal background errors. Furthermore, in [1] these different networks have been used and compared precisely for the detection of ArUco markers and it has been shown that the YOLO, in particular the YOLOv3, performs better over long distances and also achieves excellent results before the others. Finally, in presence of occlusions it has been shown that the YOLO remain robust with respect to all occlusions that do not exceed a 30% of the ArUco, while the other networks report acceptable results only if the coverage of the marker does not exceed 20%.

Dipartimento di Ingegneria dell'Informazione (DII), Università Politecnica delle Marche, Via Brecce Bianche 12, 60131, Ancona (Italy).

B. ArUco markers

For the choice of the marker to use, several articles were analyzed to understand which was the best to use, so that there weren't excessive difficulties in capturing and identifying the marker in the various transformations. In [2] the QR codes and ArUco markers were explored, to assist people with visual impairments in indoor navigation and there were difficulties in recognizing QR codes in variations in brightness and in adverse conditions, as in the case of occlusion. Instead there was a greater precision in the identification of the ArUco in presence of ambient noise and position variations. Furthermore, in [1] it was shown that several neural networks (such as YOLO, SSD, R-CNN) were able to identify the ArUco even in presence of occlusions. Finally, in [2] after a comparison between QR codes and ArUco markers it was shown that ArUco can be detected from distances up to 4 meters while QR codes were limited to only 2 meters. For these reasons, the ArUco were chosen to carry out our task.

III. Materials and Methods

A. Dataset

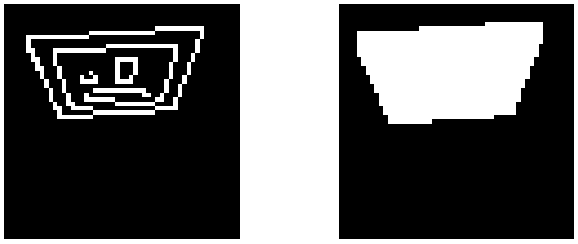
The dataset used to train the networks was custom-built. Initially the ArUco were generated (tests were carried out using 15 ArUco) and then a series of transformations were applied.

The first type of transformation applied was the perspective transformation of the ArUco. In fact, since the camera is arranged in top-view, it is necessary to simulate what the camera sees in this setting, therefore a perspective transformation of the generated ArUco was necessary.

Once these perspective transformations had been carried out for each ArUco, rotation transformations and flips of the ArUco were carried out randomly (the only flip used was the horizontal one and, at the same time, the vertical one, so as to effectively simulate the cases that will occur in the real context). Finally, occlusion transformations of the ArUco were performed, so as to simulate possible occlusions that may occur in the real case.

After having applied these transformations to the ArUco, the COCO dataset has been downloaded and each transformation of the ArUco has been overlayed in a COCO image randomly chosen, so as to obtain random images that contain the ArUco with the top-view disposition simulated by the perspective transformation.

In order to overlay the images, only the cutout of the transformed ArUco was superimposed on the COCO images; this cutout was obtained by exploiting the Canny Edge Detection, an edge detection algorithm made available by the OpenCV library[3]. Once the edges were obtained using Canny Edge Detection, the contours of the ArUco were found



(a) Canny Edge Detection (b) ArUco mask

Fig. 1: An example of ArUco pre-processing

(a) Canny Edge Detection (b) ArUco mask

Fig. 2: An example of Occluded ArUco pre-processing

and everything within these contours was kept, i.e. the ArUco. After having cut out the ArUco, it was possible to proceed with the overlay.

The labels were also created automatically. In order to do that the image of the cut out ArUco was taken (before overlaying) and from it the maximum and minimum X and Y of the white pixels were extracted and then this information was used to write the label file. In fact, each ArUco has a white border, and by calculating the maximum and minimum X and Y of the white pixels it was possible to calculate the minimum rectangle containing the entire ArUco (transformed or not).

B. Detection Network and Features Extraction

The neural network that was used is YOLO [4] (You Only Look Once), in particular were used YOLOv3 [5] and YOLOv5 [6]. The YOLO applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. Then with a post-processing, in output are obtained only the final bounding

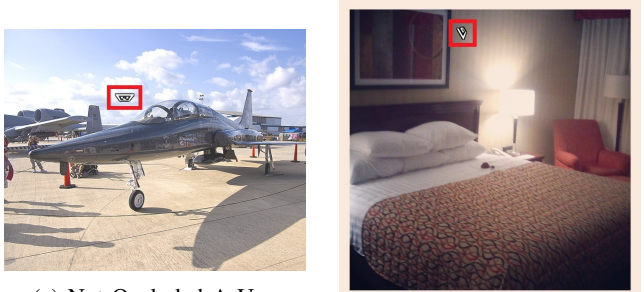


Fig. 4: YOLOv3 Architecture

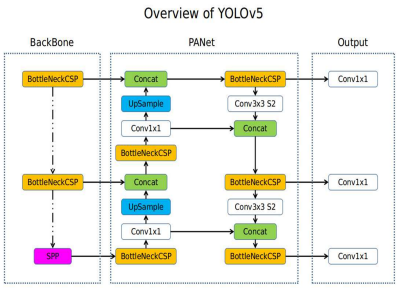


Fig. 5: YOLOv5 Architecture

boxes. In YOLO v3 (image 4), given an input image, we get a multiscale forecast, so we have grids of different sizes where we see that each cell of the grid as we go along with the scale becomes smaller and therefore we can take care of predicting an object smaller in size within the image. The YOLO v5 (image 5) has three important parts: Backbone, Neck and Head. Backbone is mainly used to extract important features from the given input image. In YOLO v5 the CSP (Cross Stage Partial) are used as a backbone to extract rich in informative features from an input image. Neck is mainly used to generate feature pyramids, those help models to generalized well on object scaling. It helps to identify the same object with different sizes and scales. In YOLO v5 PANet is used for as neck to get feature pyramids. The model Head is mainly used to perform the final detection part. It applied anchor boxes on features and generates final output vectors with class probabilities, objectness scores, and bounding boxes.

(a) Not Occluded ArUco (b) Occluded ArUco

Fig. 3: An example of test set

C. Training Settings

The dataset obtained, as described in the previous section, contains 6,000 images, to carry out the training of the networks the dataset was divided into training set, validation set and test set. For the division the percentages, respectively, of 70%, 20% and 10% referred to each class of ArUco were considered, in order to obtain a balanced dataset for the considered classes.

The training of the network YOLOv3 and of the YOLOv5 was carried out starting from the weights available in the ultralytics repositories[7][6] and obtained by training these networks on the COCO dataset, therefore pre-trained networks on the COCO dataset have been used.

For the training of the YOLO networks we used the Weights & Biases[8] tool, in order to easily visualize and have a history of the training trends.

D. Performance Metrics

The performance metrics used were Precision, Recall and two COCO mAP metrics, namely mAP@.5 and mAP@.5:.95. In this section we will briefly give the definitions of these metrics.

Precision is defined as the ratio of TP to the sum of TP and FP. This metric expresses the proportion of positive identifications that are actually correct. Recall is defined as the ratio of TP to the sum of TP and FN. This metric expresses measures the proportions of actually positives that have been found.

The other two metrics mAP@.5 and mAP@.5:.95 are two COCO metrics. Before introducing them the definitions of IoU, AP and mAP are needed. The Intersection over Union (IoU) is defined as the ratio between the area of intersection between the prediction and the ground-truth and the area of union between the prediction and the ground-truth. The Average Precision (AP) metric is defined from the precision-recall curve, which is the precision averaged over each unique recall level. Since the AP metric is defined for only one class (K=1) when there are K>1 classes, the Mean Average Precision (mAP) metric is defined as the average AP over all K classes.

COCO mAP metrics, unlike the VOC mAP which is unique, set different IoU thresholds. The mAP@.5 metric is equivalent to the Pascal VOC mAP metric, both of which are calculated by setting the IoU threshold at 0.5. The metric mAP@.5:.95 is the mAP metric averaged over ten IoU thresholds (0.50 to 0.95 in steps of 0.05).

IV. Results and Discussion

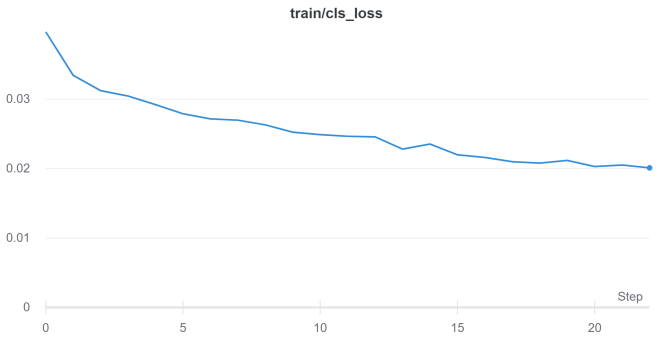
The metrics for the YOLOv3 and YOLOv5 networks described in the previous section were calculated considering a batch size of 32 and 640x640 images.

As can be seen in images I and II, better results were obtained with the YOLOv3 network. While the difference is slight for precision it is more pronounced for recall, mAP@.5 and mAP@.5:.95.

Images 6 and 7 show the training and validation classification loss trends for YOLOv3 and YOLOv5. The classification loss measures the correctness of the classification of each bounding box predicted for its class. In addition to the classification loss from the training of the YOLO networks, the box loss was also obtained. The latter measures how tight the bounding box predicted by the model is around the object, i.e. it would be the loss referred to the offset of the bounding boxes. We also have the objectness loss which measures whether a predicted bounding box actually contains an object.

V. Conclusion

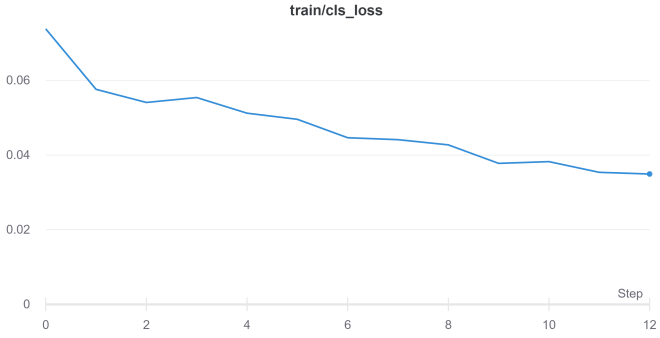
Future developments that can be made from this work can be implementing additional transformations for the ArUco, so



(a) Training Classification Loss

(b) Validation Classification Loss

Fig. 6: Classification Loss YOLOv3



(a) Training Classification Loss

(b) Validation Classification Loss

Fig. 7: Classification Loss YOLOv5

TABLE I: Test Results YOLOv3

Class	Images	Labels	Precision	Recall	mAP@.5	mAP@.5:.95
all	600	600	0.975	0.969	0.989	0.87
aruco0	600	40	0.975	0.96	0.991	0.876
aruco1	600	40	0.902	0.95	0.981	0.864
aruco2	600	40	0.949	0.923	0.968	0.86
aruco3	600	40	1	1	0.995	0.868
aruco4	600	40	0.928	0.967	0.979	0.863
aruco5	600	40	1	0.961	0.993	0.89
aruco6	600	40	1	0.995	0.995	0.868
aruco7	600	40	0.942	1	0.99	0.866
aruco8	600	40	0.998	0.975	0.995	0.868
aruco9	600	40	1	0.998	0.995	0.869
aruco10	600	40	0.977	0.975	0.995	0.908
aruco11	600	40	1	0.983	0.995	0.886
aruco12	600	40	0.974	0.925	0.986	0.845
aruco13	600	40	1	0.977	0.995	0.872
aruco14	600	40	0.974	0.943	0.981	0.845

TABLE II: Test Results YOLOv5

Class	Images	Labels	Precision	Recall	mAP@.5	mAP@.5:.95
all	600	600	0.784	0.724	0.803	0.654
aruco0	600	40	0.604	0.75	0.611	0.501
aruco1	600	40	0.614	0.75	0.761	0.62
aruco2	600	40	1	0.437	0.74	0.609
aruco3	600	40	0.806	0.825	0.897	0.689
aruco4	600	40	0.542	0.725	0.651	0.54
aruco5	600	40	0.916	0.813	0.913	0.763
aruco6	600	40	0.879	0.729	0.878	0.66
aruco7	600	40	0.779	0.375	0.637	0.534
aruco8	600	40	0.926	0.941	0.978	0.822
aruco9	600	40	0.604	0.775	0.703	0.591
aruco10	600	40	0.891	0.975	0.97	0.806
aruco11	600	40	0.833	0.775	0.908	0.744
aruco12	600	40	0.826	0.948	0.95	0.749
aruco13	600	40	0.869	0.499	0.79	0.657
aruco14	600	40	0.668	0.55	0.661	0.526

as to introduce additional noise into the dataset and strengthen the networks that would be trained on this dataset. These new implementations can be transformations that simulate new occlusions. For now only occlusions of the edges of the ArUco have been implemented. However, also diagonal occlusions of the corners of the ArUco or occlusions of the centre of the ArUco can be implemented.

Another possible development could be to collect a real dataset of the staff, wearing the ArUco markers, with a camera placed in top-view and evaluate the networks on this dataset coming from a real situation and, eventually, use it also for one or more additional training sessions.

Another possible future development could be to use other state-of-the-art networks, for example the SSD, in order to compare the results obtained with the YOLO networks to those obtained with another network.

REFERENCES

[1] B. Li, J. Wu, X. Tan, and B. Wang, "Aruco marker detection under occlusion using convolutional neural network," in *2020 5th International Conference on Automation, Control and Robotics Engineering (CACRE)*, pp. 706–711, 2020.

[2] M. Elgendy, T. Guzsvinecz, and C. Sik-Lanyi, "Identification of markers in challenging conditions for people with visual impairment using convolutional neural network," *Applied Sciences*, vol. 9, no. 23, 2019.

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[5] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[6] "https://github.com/ultralytics/yolov5."

[7] "https://github.com/ultralytics/yolov3."

[8] L. Biewald, "Experiment tracking with weights and biases," 2020. Software available from wandb.com.