Performance of Recent tiny/small YOLO Versions in the Context of Top-view Fisheye Images

Benoit Faure, Nathan Odic, Olfa Haggui, Baptiste Magnier

Performance of Recent tiny/small YOLO Versions in the Context of Top-view Fisheye Images

Benoît Faure[1], Nathan Odic[1], Olfa Haggui[2], and Baptiste Magnier[1]

[1] Euromov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France
{benoit.faure, nathan.odic}@mines-ales.org, baptiste.magnier@mines-ales.fr
[2] LIRMM, Univ. Montpellier, CNRS, Montpellier, France, olfa.haggui@lirmm.fr

Abstract. With the spreading of the computer vision field, human detection and tracking are problems more relevant than ever. However, due to the complexity of fisheye images, current lightweight detection models show difficulty when processing them. The aim of this article is to compare the performance on fisheye images of current real time detection solutions, specifically YOLOv3-tiny, YOLOv4-tiny and YOLOv5-small. Experiments carried out using a top-view fisheye camera, show faster performance but the very poor detection quality from YOLOv4-tiny. YOLOv5-small, while being slightly slower, gives a far better detection than both other solutions. The database created for this paper is available online. In conclusion, the current review shows YOLOv5-small is the best solution out of the 3 reviewed in a fast, real time, fisheye application.

Keywords: Human detection · YOLOv3, v4, v5· Fisheye camera.

1 Introduction and Motivations

Human detection is a challenging task owing to their variable appearance and wide range of poses [4], especially with fisheye camera. Indeed, a fisheye lens enables images to be acquired with a broad field of view [16,8,12]. However, pedestrians appear in different shapes, sizes and at various orientations. Unfortunately, most of the existing people-detection algorithms are designed for standard/perspective camera images where people appear upright. On the one hand, people-detection algorithms using classical feature extraction have been adapted to fisheye images [2,18,3]. On the other hand, the detection performances are improved with the great success of deep learning. For instance, the YOLO (You Only Look Once [14]) is a reliable detector based on Deep Convolutional Neural Network and remains commonly used for real-time object detection. Recently, different algorithms based on YOLO provide much faster and more accurate results than previous algorithms aimed at detecting people in fisheye images without any pre-processing [5,7]. Meanwhile, different YOLO versions exist, the last are: YOLOv3 [15], YOLOv4 [17], and YOLOv5 [10]. The small/tiny implementations of these versions are optimum regarding limited hardware resources. The aim of this article is to compare the person's detection performance of these versions on fisheye images. After discussing the differences between each algorithm, each YOLO version is trained on fisheye images, using fisheye datasets.

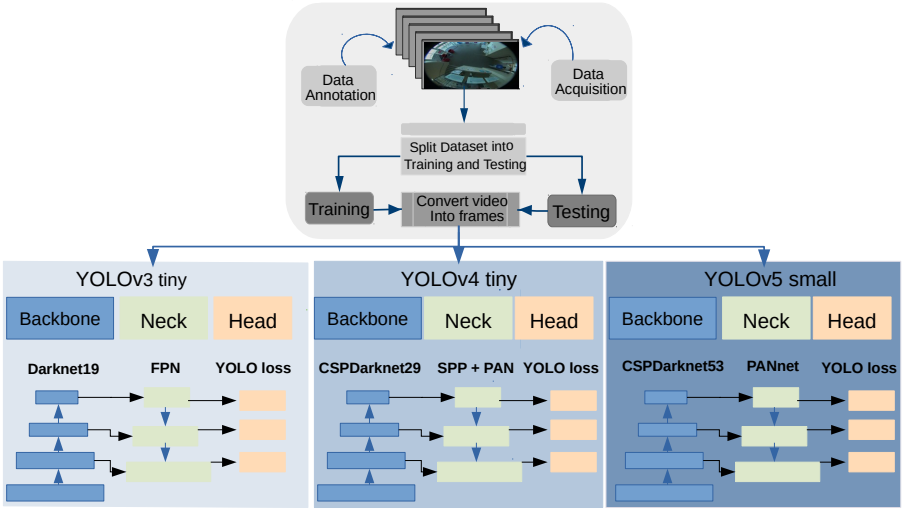


Fig. 1. Flow diagram of different YOLO architectures studied in this communication.

2 Detection Method

In this communication, human detection methods in top down fisheye cameras are evaluated. The detection model used in this paper is the state of the art method YOLO, the different versions of it we compared are shown in Fig.1.

2.1 YOLO: an Architecture for human detection

This section details the developments of the YOLO architecture based on Deep Convolutional Neural Network detection model and its extensions: YOLOv3-tiny, YOLOv4-tiny and YOLOv5-small architectures. Originally, YOLO is a state of art Object Detector which can perform object detection in real-time with a good accuracy [14]. It treats the detection task as a regression downside and has been wide utilized in image process fields. There are various deep learning algorithms, but they are unable detecting an object in a single run. Meanwhile, YOLO makes the detection in a single forward propagation through a neural network, which makes it suitable for real-time applications. This property has made YOLO very popular amongst the other deep learning algorithms. Recently, several versions, e.g., YOLOv3 [15], also YOLOv4 [17], and lastly YOLOv5 [10], have additionally been developed to improve the classification performance on complicated datasets, and, to increase the quantity of data within the feature map. Usually, in the architecture of the YOLO algorithm, the head and neural network type are the same for all of the algorithms, whereas backbone, neck, and loss function are different, as detailed below.

YOLOv3-tiny: In order to benefit computers having limited hardware resources, YOLOv3-tiny algorithm (denoted YOLOv3-t) is the preferable version of YOLOv3. Indeed, it is easy for YOLOv3-t [9][21] network to satisfy real-time requirements on a standard computer with a limited Graphics Processing Unit (GPU). In fact, YOLOv3-t is the simplified and light version of the original YOLOv3 [15]. It operates with the same operating principle as original model, but with a varied number of parameters in which the depth of convolutional layer is reduced. Originally, YOLOv3 [15] utilizes the architecture of darknet53, and then uses many 1×1 and 3×3 convolution kernels to extract features. The Darknet19 structure of the YOLOv3-t network, within structure contains only seven convolutional layers and small number of 1×1 and 3×3 convolutional layers is used as feature extractor to achieve the desired effect in miniaturized devices. Eventually, to reduce the dimensionality size through the network: pooling layer is applied. However, its convolutional layer structure still uses the same structure of 2D-Convolution, Batch Normalization and an activation function LeakyRelu (Leaky Rectified Linear Unit) as the YOLOv3 algorithm. This network simplification requires this model to occupy less amount of memory and, consequently, will improve the speed of the detection process.

YOLOv4-tiny: As a modified version of YOLOv3, YOLOv4 [17] is used in this work with a derived version. Thus, YOLOv4-tiny (denoted YOLOv4-t) uses Cross Stage Partial Network (CSPNet) in Darknet, creating a new feature extractor backbone called CSPDarknet53. However, to help it achieve these fast speeds, YOLOv4-t [11] utilizes a couple of different changes from the original YOLOv4 network. Foremost, the number of convolutional layers in the CSP backbone are compressed with a total of 29 pre-trained convolutional layers. Additionally, the number of YOLO layers has been reduced to two instead of three and there are fewer anchor boxes for prediction. We can use YOLOv4-t for a faster training and a faster detection. Therefore, the neck is composed of a Spatial Pyramid Pooling (SPP) layer and PANet path aggregation. They are used for feature aggregation to improve the receptive field and short out important features from the backbone. In addition, the head is composed of YOLO layer. Fundamentally, the image is fed to CSPDarknet for feature extraction and then to path aggregation network PANet for fusion. At last, YOLOv4-t is the better option when compared with YOLOv4 as faster inference time is more important when working with a real-time object detection environment.

YOLOv5-small: In this context, YOLOv5 is different from the previous releases. The v5 model has shown a substantial performance increase from its predecessors. In addition, YOLOv5 [10] comes with its various versions: YOLOv5-s, the small version, YOLOv5-m, the medium version, YOLOv5-l, the large version and YOLOv5-x, the extra-large version, each having its own unique characteristic. Since this study focuses on real-time detection, the speed is a factor of the utmost importance, hence the smallest version has been chosen as the representative of the YOLOv5 family for its performance analysis. The backbone

is CSPDarknet53 and solves the repetitive gradient information in large backbones and integrates gradient change into feature map; that reduces the inference speed, increases accuracy, and reduces the model size by decreasing the parameters. Furthermore, it uses a path aggregation network (PANet) as neck to boost the information flow. PANet adopts a new feature pyramid network (FPN) that includes several bottom ups and top down layers. Consequently, this improves the propagation of low-level features in the model. The PANet improves the localization in lower layers, which enhances the localization accuracy of the object. In addition, the head in YOLOv5 is the same as YOLOv4 and YOLOv3, generating three different outputs of feature maps to achieve multi-scale prediction. Finally, it helps to enhance the prediction of small to large objects efficiently.

2.2 Dataset

Images in these datasets were resized to 640×640 pixels because YOLO requires square images whose height and width are multiples of 32. The image size of 640×640 was chosen based on preliminary tests with YOLOv5-s.

Public Datasets: Primarily, as a base for the algorithm, weights trained on the COCO dataset [13] were used. This dataset provides numerous images containing human examples. Even though the images presented are not fisheye and not necessarily top view, they gave a big data baseline for our model. The objective being, to start fisheye training with weights already able to recognize the general features of a human figure. A few benchmarks and datasets have been created in order to train and evaluate people detection algorithms for fisheye images. Such datasets are used to specialize our model on fisheye cameras. Fisheye images pose challenges not encountered in images taken with a standard lens. These include illumination variations across the image and big variations in the angle at which a person appears. Additionally, fisheye images distort figures in it. The distortion is much more prominent on the edge of the images, with big objects being affected the most. The distortion causes shrinking of the objects, this makes it harder for the YOLO algorithm to discern it.

The chosen dataset for specialization is the MIRROR Worlds dataset[3]. However, preliminary tests on these datasets gave poor detection results. That is because annotations of this dataset are imprecise, as shown in Fig. 2. These

[3] http://www2.icat.vt.edu/mirrorworlds/

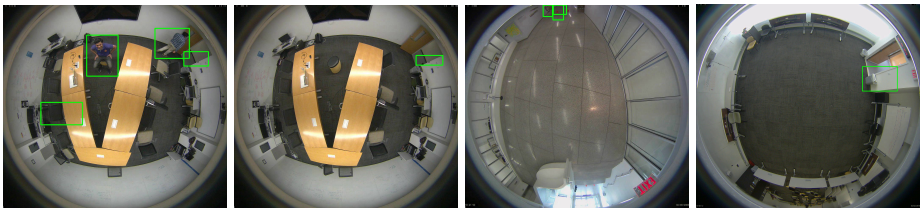


Fig. 2. Dataset MIRROR, originally not perfectly annotated.

Table 1. Experimental characteristics of the used datasets.

Database	MIRROR (new annotations)	Our dataset
Number of videos	19	31
Number of images for training	821	1492
Number of images for test	204	377 + 406
Moving camera	No	Yes

annotations were at times not bounding anything, or bounding completely different objects, objects that ended up being detected as human by the detection algorithm. We thus decided to relabel the MIRROR dataset using *roboflow*[4]. Subsequently, 1025 annotated images were obtained; the split for this dataset is 80% training and 20% test. Additionally, we decided to create a new dataset to further increase the variations in environments and situations the algorithm could find itself in.

Our Dataset: The images for the new dataset were collected in the CERIS laboratory of the IMT mines Alès. Videos were acquired with a fisheye camera (Basler ace acA2040-120uc) mounted on a pole and placed around 3m in height. Initially, images were pulled from videos from both inside and outside the laboratory's hangar. During each capture, the camera was static, 2-3 people moved around the camera taking different positions. Examples include sitting, lying down and crouching. Furthermore, the brightness level of the images is very low inside the hangar, outside, however the images are very clear. The split for this dataset is 80% train and 20% test; it is available online with tied annotations[5].

A second part of the dataset was made, the objective of which being to test the effectiveness of each algorithm (see Sec. 2.1) on images set in environments they were not familiar with. For this part of the dataset, images were pulled from videos taken outdoors and on the street outside. Consequently, these images are different from the one detailed previously. So far, 406 images were annotated, and between 0 and 3 discernible people appear in each image. Finally, the Tab. 1 represents the characteristics of the datasets we used with 406 annotated images of the different scene than others utilized only for the tests.

Data Annotation: Data labelling is an essential step in a supervised machine learning task requiring significant manual work. To annotate our new dataset, boxes containing people are manually annotated. In order to do so, the *Roboflow* Tool is used. From a technical aspect, the pixel representing the center of the BBox, as well as its height and width in pixels are defined. Consequently, each BBox is represented by five parameters:

- (x, y): pixel coordinates of the BBox center,
- w and h: width and height of the BBox respectively,
- class: ID of the object category.

The parameters x, y, w and h are essential for the evaluation presented in the next section, whereas the ID concerns mainly tracking processes.

[4] https://roboflow.com/annotate
[5] https://github.com/BenoitFaureIMT/CERIS_FishEye

3 Experiments and Evaluations

3.1 Evaluation Metrics

To analyze the progress of the network, we utilize the five following metrics: mean average precision 0.5 and 0.5:0.95, precision, recall and CIoU.

Mean Average Precision After running the network on a certain number of images, of which the real BBox sizes and positions are known (i.e., called ground truth), the IoU (Intersection over Union) is computed between each detected BBox and all real BBoxes, the corresponding box with the highest IoU can be considered as the detected BBox: $IoU = \frac{Intersection\ area\ of\ both\ boxes}{Union\ area\ of\ both\ boxes}$. A threshold α is then set to compute the mean average precision mAP_α:

$$mAP_\alpha = \frac{Number\ of\ detections\ where\ the\ coressponding\ IoU \geq \alpha}{Number\ of\ detections\ from\ the\ neural\ network} \tag{1}$$

Moreover, it is possible to take the average of different values of α, to show the progress of the network over a range of metrics simultaneously. With this idea we declare $mAP_{0.5:0.95}$:

$$mAP_{0.5:0.95} = \frac{1}{10}\sum_{k=0}^{9}(mAP\ (0.5 + 0.05 * k) \tag{2}$$

Precision: When a detected BBox obtains an IoU above 0, it is counted as a positive. However, the box of the detected object could be very inaccurate, for example by bounding only the top half, or by actually covering twice or thrice the area of the real box. To counter this problem, we set a threshold at 0.5, above which a detection is counted as a true positive, and below which, it is counted as a false positive: $precision = \frac{total\ amount\ of\ true\ positives}{number\ of\ true\ positives\ +\ number\ of\ false\ positives}$.

Recall: It represents the percentage of detections, which were not missed, a real BBox which was not detected represents a false negative and define the recall: $recall = \frac{total\ amount\ of\ true\ positives}{total\ amount\ of\ (true\ positives\ +\ false\ negatives)}$.

CIoU: Complete IoU bounding boss regression [22] allows for fast convergence towards the ground truth BBox. It is used in the training of YOLOv3-t, YOLOv4-t and YOLOv5-s. The metric combines the overlap between predicted and ground truth, as well as the center of the BBoxes and their aspect ratio.

3.2 Training of the models

To train the different YOLO networks, we used the respective *PyTorch* implementations of each architecture. The definition for the architecture of YOLOv4-t was obtained from a github repository in [26], which based it on Ultralytics' yolov5 implementation. Finally, the definition for YOLOv3-t [24], YOLOv5-s [23]

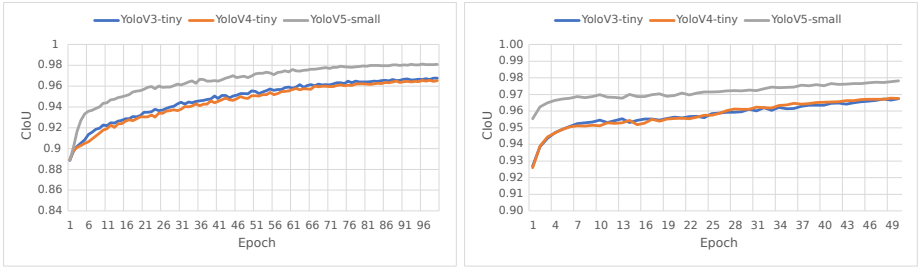
Table 2. Training parameters for YOLOv3-t, YOLOv4-t and YOLOv5-s

Version	Training on MIRROR dataset			Training on our dataset			Hyper parameters
	Epochs	Batch size	Image size	Epochs	Batch size	Image size	
YOLOv3-t	100	64	640	50	64	640	Default
YOLOv4-t	100	64	640	50	64	640	Default
YOLOv5-s	100	64	640	50	64	640	Default

and the methods for training [25] the networks were obtained from the Ultralytics github repository, the company that developed the YOLOv5 architecture.

As mentioned in the Sec. 2.2, the training started using weights pre-trained on the COCO dataset [13]. Each YOLO version is then specialized on fisheye cameras, using first the MIRROR dataset (own annotated) before moving training onto our own dataset. We chose to train on the datasets in this order, as the context in which the network will be applied is expected to be closer to the environment we obtained images from. The settings for the training of each YOLO version are detailed in the Tab. 2.

The batch size was maximized while considering the GPU's memory limitations could hold, this was done as to follow indications from Ultralytics on training the model. The image size was set to correspond to the size of COCO images and hyper-parameters were set to their default values. To quantify how close we are to overfitting the model, we observed the rate at which the CIoU increased. In an attempt to limit the risk of overfitting, epochs for both datasets were chosen as to stop training before this rate became negligible. The Fig. 3 represents the evolution of the CIoU during training on the MIRROR dataset (a) and our dataset (b). The graphs start very close to each other in (a) as they were all trained on the COCO dataset, however YOLOv5-s shows better performance in learning especially at the start of training where it begins converging much faster. In Fig. 3(b), the graphs start separate, this is due to the better score YOLOv5-s had at the end of training on the MIRROR dataset. The combination of these two graphs demonstrates a better learning performance of the YOLOv5-s model.



(a) MIRROR Dataset (new annotations) (b) Proposed Dataset

Fig. 3. Evolution of CIoU during training of each YOLO model

Table 3. Performance of YOLOv3-t, YOLOv4-t and YOLOv5-s on the MIRROR dataset after being trained on it (computer equipped with GPU Tesla K80).

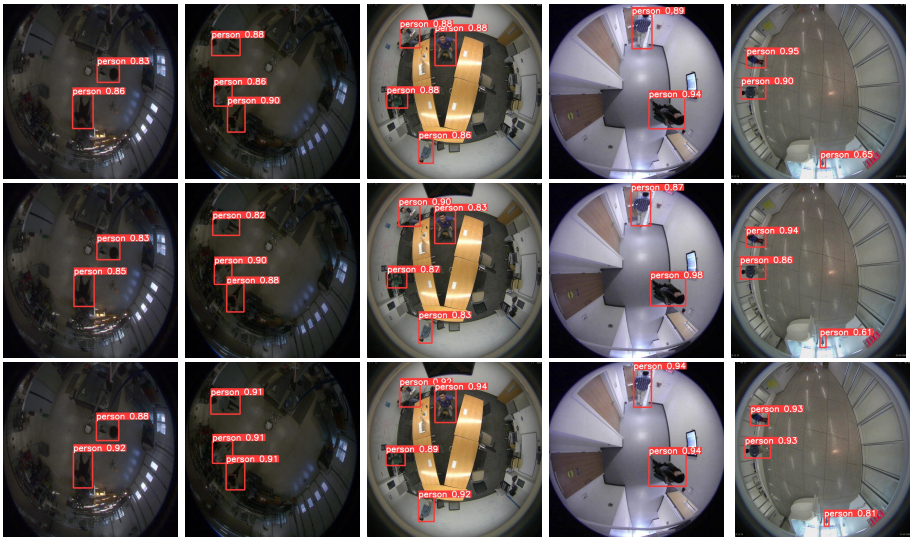
Version	Precision	Recall	$mAP_{0.5}$	$mAP_{0.5:0.95}$	Detection (ms)
YOLOv3-t	0.964	0.922	0.976	0.617	19.8
YOLOv4-t	0.947	0.952	0.975	0.587	14.7
YOLOv5-s	0.953	0.974	0.989	0.736	25.3

Table 4. Performance of YOLOv3-t, YOLOv4-t and YOLOv5-s on our dataset after being trained on both MIRROR and our new dataset (computer with GPU Tesla K80).

Version	Precision	Recall	$mAP_{0.5}$	$mAP_{0.5:0.95}$	Detection (ms)
YOLOv3-t	0.988	0.961	0.984	0.644	20.1
YOLOv4-t	0.976	0.936	0.976	0.626	15.2
YOLOv5-s	0.987	0.991	0.993	0.724	25.8

3.3 Performance of the models on images in a familiar context

The Tab. 3 gives the final metrics of the training and testing on the MIRROR dataset, while the Tab. 4 gives the final metrics of the testing on our dataset of models trained on both our dataset and the MIRROR dataset. The results are similar across all 3 versions for Precision, Recall and $mAP_{0.5}$, however YOLOv5-s has better results for $mAP_{0.5:0.95}$. In terms of the detection quality, YOLOv5-s performed best, with YOLOv3-t and YOLOv4-t as second and third place respectively. However, in detection speed the rankings are reversed, the increase in speed from YOLOv4-t comes from the small size of the neural network (weight file is 6.3Mo) relative to YOLOv3-t (17.4Mo) and YOLOv5-s (14.5Mo).



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

Fig. 4. People detection on top-view fisheye images. The used algorithms for the detection are in the first row: YOLOv3-t, second row: YOLOv4-t and third row: YOLOv5-s. Images in (a)-(b) correspond to our new image dataset whereas in (c)-(e), they are tied to MIRROR dataset with new annotations.

Table 5. YOLOv3-t, YOLOv4-t and YOLOv5-s performances in an unfamiliar context.

Version	Precision	Recall	$mAP_{0.5}$	$mAP_{0.5:0.95}$	Detection (ms)
YOLOv3-t	0.374	0.221	0.207	0.054	19.8
YOLOv4-t	0.191	0.109	0.089	0.021	14.4
YOLOv5-s	0.820	0.656	0.739	0.389	24.2
GPU	Tesla K80				

To finish testing, all 3 models were used to detect people in the same dataset they were trained on. A sample of the data obtained is shown in Fig. 4. All 3 algorithms perform well in both a dark (a)-(b) and light (c)-(e) contexts. The algorithms were also able to detect and bound small entities as shown in column (e) of Fig. 4.

3.4 Performance of the models on images in an unfamiliar context

Results shown previously (Sec. 3.3) give a very optimistic view of the algorithms performance. However, the high similarity between the BBoxes provided by each model, as well as the very high performance in low light contexts possibly indicates a partial memorization of the dataset. To investigate this, all 3 trained YOLO versions were tested on the second part of our dataset. As the images and contexts differ greatly in comparison to images our models were trained on, it would give a better indication of the models performance.

The Tab. 5 summarizes the results obtained in the test, the rankings are consistent with the results obtained previously; however the distance between each model is much bigger. On all precision metrics, YOLOv5-s performs much better than the other two metrics. The $mAP_{0.5}$ of the model is at 0.739, meaning that the algorithm is capable of bounding it's detection relatively well. Furthermore, the model displays a high precision (0.82), indicating the low number of false positives in an unfamiliar context. Finally, the recall is at 0.656, it indicates that the model missed around 35% of detection; this figure is high, but in the context of a tracking algorithm could be negligible if compensated properly.

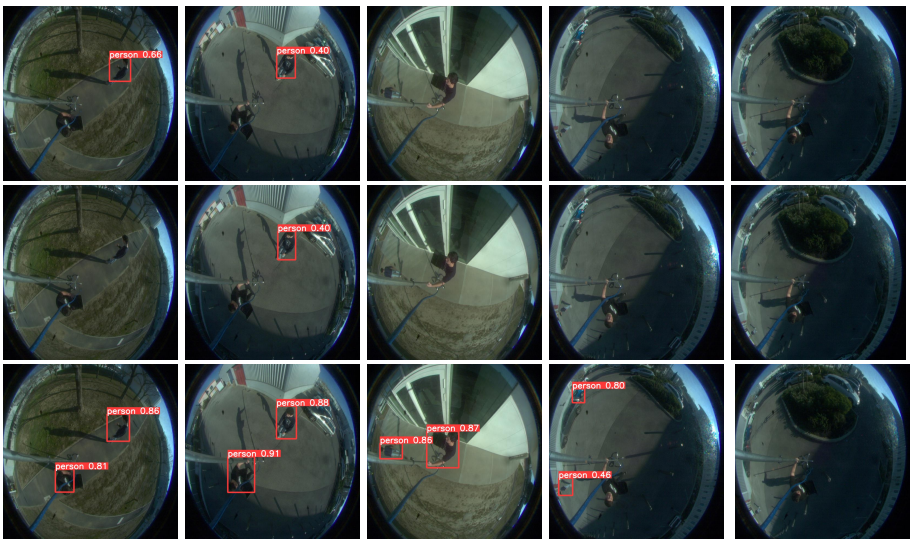
The Fig. 5 shows a sample of the images obtained from the tests. While these images have been selected, they represent quite well the obtained images. In that respect, YOLOv5-s outperforms the other networks, but still has its limitations, as shown by the metrics discussed previously. Column (e) shows two false negatives, one (center) is due to the black clothes on a dark background, and another (bottom left) appears when the person walks into the very edge of the image. This same person was detected a few frames before in the column (d). This column also shows a false negative (center) and a false positive (top left) also appearing at the edge of the frame where cars can be seen distorted. Nevertheless, looking at columns (a), (b) and (c) where the images are very clear, YOLOv5-s was able to properly detect all people present in the picture.

While it is true that YOLOv3-t and YOLOv4-t both have faster detection speeds than YOLOv5, the difference between each algorithm is about 5ms. The detection speeds of YOLOv5-small are fast enough to be used in real time cases (i.e., $>$10 fps), making detection speed a secondary factor in the comparison of

each network. The difference is further dwarfed by the very high performance of YOLOv5-s on detection metrics in comparison to YOLOv3-t and YOLOv4-t. We expect YOLOv4-t's very poor performance to come from it's small size, because of it, it memorized the dataset rather than learn important features.

4 Conclusion

This paper presents a comparison in the performance of YOLOv3-tiny, YOLOv4-tiny and YOLOv5-small when applied to a problem involving fisheye images. During training, even though YOLOv5-s shows slightly better convergence, all 3 networks give similar detection scores. However, the same cannot be said about the networks' performance on datasets pulled from contexts they are unfamiliar with. In these cases, YOLOv5-s surpasses both YOLOv3-t and YOLOv4-t by a large margin, while giving itself acceptable detection performances. Moreover, while YOLOv5-s displays the slowest detection time, it is a small disadvantage as the detection speeds are still fast enough (25ms per frame) for real time applications. To further the argument in favor of YOLOv5-s, YOLOv4-t and YOLOv3-t have detection times of the same order of magnitude as YOLOv5-s, making the differences negligible in comparison to the differences in detection quality. Subsequently, YOLOv5-s is the best solution out of the 3 proposed for a real time, fisheye detection application. This solution was retained to act as the detection portion of a real time tracking algorithm [19,6,20], applied to videos acquired from a top-view fisheye camera, and mounted on an aerial drone or for security cameras. As a reminder, the dataset is available online (see Sec. 2.2).



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

Fig. 5. People detection on top-view fisheye images on a new dataset without training on the same data. The used algorithms for the detection are in the first row: YOLOv3-t, second row: YOLOv4-t and third row: YOLOv5-s.

References

1. Ahmed, I., Ahmad, M., Ahmad, A. and Jeon, G. (2020). Top view multiple people tracking by detection using deep SORT and YOLOv3 with transfer learning: within 5G infrastructure. International Journal of Machine Learning and Cybernetics. 12(11), 3053-3067.
2. Chiang, A. T., and Wang, Y. (2014). Human detection in fish-eye images using HOG-based detectors over rotated windows. IEEE ICMEW, 1-6.
3. Demirkus, M., Wang, L., Eschey, M., Kaestle, H., and Galasso, F. (2017). People Detection in Fish-eye Top-views. In VISIGRAPP (5: VISAPP), 141-148.
4. Dollar, P., Wojek, C., Schiele, B., and Perona, P. (2011). Pedestrian detection: An evaluation of the state of the art. IEEE TPAMI, 34(4), 743-761.
5. Duan, Z., Tezcan, O., Nakamura, H., Ishwar, P., and Konrad, J. (2020). RAPiD: rotation-aware people detection in overhead fisheye images. IEEE/CVF CVPR Workshops, 636-637.
6. Haggui, O., Tchalim, M.A. and Magnier, B. (2021). A Comparison of OpenCV Algorithms for Human Tracking with a Moving Perspective Camera. IEEE EUVIP.
7. Haggui, O., Bayd. H,. A.,Magnier, B., and Aberkane, A. (2021) Human Detection in Moving Fisheye Camera using an Improved YOLOv3 Framework. IEEE MMSP.
8. Hansen, P., Corke, P., and Boles, W. (2010). Wide-angle visual feature matching for outdoor localization. The International J. of Robotics Research, 29(2-3), 267-297.
9. He, W., Huang, Z., Wei, Z., Li, C., Guo, B. (2019) TF-YOLO: An Improved Incremental Network for Real-Time Object Detection. Appl. Sci. , 9, 3225.
10. Iyer, R., Shashikant Ringe, P., Varadharajan Iyer, R. and Prabhulal Bhensdadiya, K. (2021). Comparison of YOLOv3, YOLOv5s and MobileNet-SSD V2 for Real-Time Mask Detection. International Research J. of Engineering and Technology, 8(7), 2395–0056.
11. Jiang, Z., Zhao, L., Li, S., and Jia, Y. (2020). Real-time object detection method based on improved YOLOv4-tiny. arXiv preprint arXiv:2011.04244.
12. Kumler, J. J., and Bauer, M. L. (2000). Fish-eye lens designs and their relative performance. In Current developments in lens design and optical systems engineering. International Society for Optics and Photonics. 4093, 360-369.
13. Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., and Zitnick, C. L. (2014). Microsoft COCO: Common Objects in COntext. In ECCV, 740-755.
14. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. IEEE CVPR, 779-788.
15. Redmon, J., and Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.
16. Scaramuzza, D., and Ikeuchi, K. (2014). Omnidirectional camera. Springer.
17. Shaniya, P., Jati, G., Alhamidi, M.R., Caesarendra, W. and Jatmiko, W. (2021). YOLOv4 RGBT Human Detection on Unmanned Aerial Vehicle Perspective. IEEE IWBIS, 41-46.
18. Wang, T., Chang, C. W., and Wu, Y. S. (2017). Template-based people detection using a single downward-viewing fisheye camera. IEEE ISPACS, 719-723.
19. Wojke, N., Bewley, A. and Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. IEEE ICIP, 3645-3649.
20. Talaoubrid, H., Vert, M., Hayat, K., and Magnier, B. (2022). Human Tracking in Top-View Fisheye Images: Analysis of Familiar Similarity Measures via HOG and against Various Color Spaces. Journal of Imaging, 8(4), 115.

21. Xiao, D., Shan, F., Li, Z., Le, B.T., Liu, X., and Li, X. (2019). A Target Detection Model Based on Improved Tiny-Yolov3 Under the Environment of Mining Truck. IEEE Access, 7, 123757-123764.
22. Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., and Ren, D. (2020). Distance-IoU Loss: Faster and better learning for bounding box regression. In AAAI.
23. Ultralytics (2021) yolov5s [source code]. https://github.com/ultralytics/yolov5/blob/master/models/yolov5s.yaml
24. Ultralytics (2021) yolov5-tiny [source code]. https://github.com/ultralytics/yolov5/blob/master/models/hub/yolov3-tiny.yaml
25. Ultralytics (2021) train [source code]. https://github.com/ultralytics/yolov5/blob/master/train.py
26. Yolov4-tiny [source code]. https://github.com/WongKinYiu/PyTorch_YOLOv4/blob/u5_preview/models/yolov4-tiny.yaml