Evaluation of Computer Vision-Based Person Detection on Low-Cost Embedded Systems

Francesco Pasti[1] and Nicola Bellotto[1,2]
[1]*Dept. of Information Engineering, University of Padova, Italy*
[2]*School of Computer Science, University of Lincoln, U.K.*

Keywords: Embedded Systems, Person Detection, Computer Vision, Edge Computing

Abstract: Person detection applications based on computer vision techniques often rely on complex Convolutional Neural Networks that require powerful hardware in order achieve good runtime performance. The work of this paper has been developed with the aim of implementing a safety system, based on computer vision algorithms, able to detect people in working environments using an embedded device. Possible applications for such safety systems include remote site monitoring and autonomous mobile robots in warehouses and industrial premises. Similar studies already exist in the literature, but they mostly rely on systems like NVidia Jetson that, with a Cuda enabled GPU, are able to provide satisfactory results. This, however, comes with a significant downside as such devices are usually expensive and require significant power consumption. The current paper instead is going to consider various implementations of computer vision-based person detection on two power-efficient and inexpensive devices, namely Raspberry Pi 3 and 4. In order to do so, some solutions based on off-the-shelf algorithms are first explored by reporting experimental results based on relevant performance metrics. Then, the paper presents a newly-created custom architecture, called eYOLO, that tries to solve some limitations of the previous systems. The experimental evaluation demonstrates the good performance of the proposed approach and suggests ways for further improvement.

1 INTRODUCTION

Computer vision-based person detection is a key feature of many real-world applications, including distributed monitoring systems and autonomous mobile robots in warehouses, factories, and other industrial scenarios. Often there also strong limitations in terms of budget and power consumption, so a trade-off needs to be found between the performance of the person detection algorithm and the hardware requirements to achieve satisfactory runtime performance.

Most state-of-art solutions rely on rather complex Convolutional Neural Networks (CNNs) or similar variants, which typically require sufficiently powerful hardware to detect people from live video streams. To this end, the application motivating the current research is the implementation of an embedded vision system for human safety, which is able to detect people in different working environments (e.g. see Figure 1) and, in case, trigger an opportune response by a local mobile robot or a remote monitoring station.

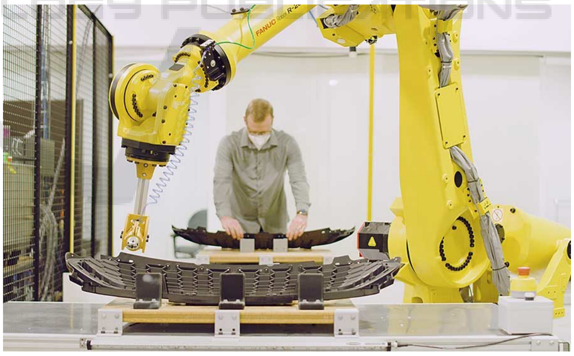


Figure 1: Worker in a potentially unsafe situation.

Although similar studies exist in the literature, they typically rely on high-performing embedded systems like the NVidia Jetson TX2 and Nano, since these integrate Cuda-enabled GPU that provide enough computing power to run sufficiently fast CNN-based algorithms. Such a choice however comes with a significant downside, as these devices are relatively expensive and require too much energy to run on small battery-powered devices. In this paper, instead, two more energy-saving and less expensive devices are being considered, i.e. a Raspberry Pi 3 and a Raspberry Pi 4. The main challenge is that these two platforms are not equipped with a powerful

Pasti, F. and Bellotto, N.
Evaluation of Computer Vision-Based Person Detection on Low-Cost Embedded Systems.
DOI: 10.5220/0011797400003417
In *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - Volume 5: VISAPP*, pages 282-293
ISBN: 978-989-758-634-7; ISSN: 2184-4321

GPU and thus are not able to perform the same huge amount of calculations required by modern CNNs.

This paper therefore is going to analyse some relevant work on vision-based object detection, with a particular focus on solutions for embedded devices. Recent studies mostly focused on CNNs that, due to the parallelism structure, are solved efficiently by GPU-equipped devices. A different approach is taken by this paper, by first considering some old classic methods (i.e. not CNN-based) for person detection, which were developed at a time where computers much less powerful and that could be suitable for modern-day embedded systems. These algorithms will be compared to YOLO v3, a representative solution of CNN-based architectures.

After the initial study of the above methods, the paper is going to present the implementation of a new customized CNN, inspired by the original version of YOLO, where some original techniques and ideas are adopted to reduce as much as possible the time required for detecting people on live-stream video frames with Raspberry Pi 3 and 4, while maintaining an acceptable level of accuracy for some safety applications. These performances will be compared to the previous off-the-shelf methods.

The remainder of the paper is conceptually divided in two parts. The first one includes the related work in Section 2 and the evaluation of old and newer off-the-shelf algorithms for person detection on embedded systems, presented in Section 3, which forms a baseline for further improvements. The second part starts with Section 4, proposing a new version of YOLO that represents a trade-off between computational and detection performance on embedded systems, and concludes in Section 5 with a discussion of the results and suggestions for future work.

2 RELATED WORK

The task of vision-based object detection is a challenging problem. There can be a huge number of pixels to be processed and the object could be in any position at multiple scales. The well-known approach of Viola & Jones (Viola and Jones, 2001) extracts Haar features, effectively reducing the number of parameters necessary to represent the pixel intensities of the image. Based on these features, a cascade of classifiers is used to detect objects through a fast computation process. The structure of the cascade resembles a decision tree, where a positive result of the first classifier triggers an evaluation of the second, and so on, while a negative results causes the rejection of the sub-window currently analyzed. In practice, each stage of the cascade acts as a filter, where the first one discarding a sample prevents further processing by all the subsequent stages, drastically reducing the computational effort. Viola & Jones' approach is therefore very efficient and potentially useful for low-cost embedded systems since it does not rely on complex CNN computations.

Another classic method, which is not based on CNNs or other recent neural network architectures, is the Histogram of Oriented Gradients (HOG) approach introduced by Dalal & Triggs (Dalal and Triggs, 2005). The HOG features are combined with a linear Support Vector Machine (SVM). Their peculiarity is that they are able to provide, from an input image, the number of gradient orientations in localized portions of such image, and contain more information with respect to the simpler Haar features. For this reason they also provide better results for person detection, while maintaining a relatively low computational cost.

Although the above and other feature-based methods are still popular for practical applications of embedded computer vision (Farouk Khalifa et al., 2019), more recent solutions for object detection and image classification are typically implemented with CNNs and other (deep) neural network architectures. The main difference from previous previous approaches is that CNN do not use hand-crafted features, as Haar or HOG. Instead, they perform an end-to-end process to learn the best feature representation that fits the problem.

The work by Girshick et al. (Girshick et al., 2014), for example, is one of the early approaches showing that a CNN-based solution leads to far better performances with respect to the previous feature extraction-based ones. Previous CNN object detection systems were based on sliding windows or regression. The Region-based CNN (R-CNN) of Girshick et al. solves the object localization problem by using a ”recognition using regions” paradigm. Each image generates 2000 independent region proposals, from which fixed-length features are then extracted using a CNN and later classified with a linear SVM. This approach gives excellent detection results, but its main disadvantage is a selective-search algorithm that increases significantly the inference time. Furthermore, each part of the algorithm needs separate training processes, making the method difficult to implement and modify.

An improved version, called Fast R-CNN (Girshick, 2015), addressed the training process and the inference time problems. The new network takes as input the entire image and a set of Region of Interest (RoI) proposals to achieve better performances.

In (Ren et al., 2015), the authors introduced Faster R-CNN, another method that incorporates a new Region Proposal Network (RPN) and shares the computation burden with Fast R-CNN, allowing the whole process to run even faster.

Another interesting algorithm for object detection is the Single Shot MultiBox Detector (SSD) proposed by Liu et al. (Liu et al., 2016). This is a single network that, diffently from the previous ones, does not re-sample feature maps of pixels on RoI hypothesis, which makes it extremely fast while keeping the same detection performance. It is a feed-forward network that produces a fixed size of bounding boxes and scores for a specific class of objects. Multiple convolutional feature layers are added at the end of the base network in order to allow different size predictions. The output of each feature layer works like a grid: each grid cell produces a prediction with a score and an offset relative to the default bounding box. Using an end-to-end architecture and treating the detection as a regression problem are the tricks allowing the network to be really fast while still producing good results.

Perhaps the most popular CNN for object detection is YOLO by Redmond et al. (Redmon et al., 2016). It considers object detection as a single regression problem with the image as input to the CNN and output a series of vectors containing the objects location with corresponding class probabilities. In particular, the image is divided by the system into an SxS grid, and the cell containing the object's center is the one considered for its detection. During training, a custom loss function is optimized to avoid overconfident selection of cells that contain predictions against the ones that do not, which could lead to instability and model divergence.

A series of incremental improvements has been made to the original YOLO network. In particular, YOLO v2 by Redmond et al. (Redmon and Farhadi, 2017) uses higher resolution for training, exploits batch normalization to improve the performances, and uses anchor boxes. The latter are provided as a list of predefined boxes that best match the desired objects, and the predicted box is then scaled with respect to the anchors. YOLO v3, instead, also by Redmond et al. (Redmon and Farhadi, 2018), adds an objectiveness score to bounding box predictions, using three separate levels of granularity to improve the detection of smaller objects. More recent versions of YOLO, by Sruthi et al. (Sruthi et al., 2021) and Li et al. (Li et al., 2022), have then been proposed by different authors, making use of new techniques such as loss functions based on Intersection over Union (IoU) and adopting state-of-the-art CNNs as a classification backbone.

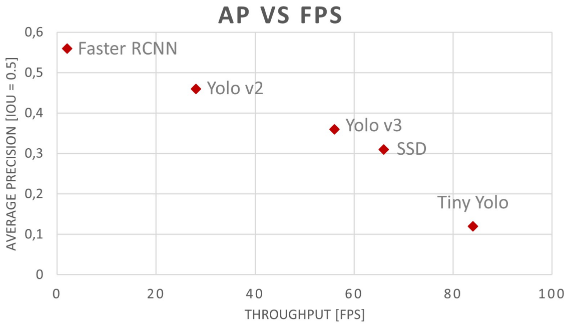


Figure 2: Comparison of the Average Precision versus the throughput FPS performed by some popular CNN-based architectures on a Jetson TX2 (Kim et al., 2019).

Continuous advancements in embedded hardware and deep learning architectures enabled research on vision-based object and person detection with resource-constrained devices. Recent work by Kim et al. (Kim et al., 2019), for example, provides a good overview of person detection on embedded devices, exploring some of the most popular CNNs such as YOLO, SSD, and R-CNN. The authors also evaluated these algorithms on an in-house proprietary dataset, reporting in particular the runtime performance on an NVidia Jetson TX2. All the models were adapted for the task of person detection, trained and tested on their dataset about costumers in stores. As highlighted in the paper, finding a good CNN for an embedded device deployed in the real world is not only a matter of how well the person detection performs, but also how fast it accomplishes the task. In their case, the most computationally efficient and faster versions of CNN are the Tiny YOLO ones, as illustrated in Figure 2. However, YOLO v3-416 and SSD (VGG-500) provide the best trade-off between average precision and throughput.

As CNNs are getting more efficient and embedded devices more powerful, applications that combine both are also becoming more popular. Some lightweight YOLO-based architecture have been developed for non-GPU computers (Huang et al., 2018), although the availability of high-end GPUs enable faster and more accurate deep learning solutions. However, this comes with high expenses and power consumption. The work by A. Suezen et al. (Süzen et al., 2020) discusses the implementation of a CNN for 13 object categories classification evaluated on three different single-board computers: NVidia Jetson Nano, NVidia Jetson TX2, and Raspberry Pi 4. Although all these are embedded systems with limited resources, the key difference between the two NVidia boards and the Raspberry Pi is that the former have a CUDA-capable GPU, which is optimized to perform

Table 1: Benchmarking of the NVidia Jetson Nano, TX2, and the Raspberry Pi 4 (Süzen et al., 2020).

Dataset	Accuracy [%]			Time [sec]			Total Power [W]		
	TX2	**Nano**	**PI 4**	**TX2**	**Nano**	**PI 4**	**TX2**	**Nano**	**PI 4**
Idle	-	-	-	-	-	-	3.275	1.23	0.30
5K	87.6	67.5	87.2	23	32	173	7.50	3.73	3.4
10K	93.8	93.9	91.6	32	58	372	8.10	5.57	3.6

the big parallel computation required by CNNs. Table 1 reports the results based on their classification network trained and tested on 5 different datasets, including the time refers needed to classify all the testing images.

As further shown in more recent work (Wu et al., 2021), it is clear that GPU-enabled devices have a huge advantage in terms of speed. For each dataset in Table 1 indeed, the Pi 4 is outperformed by one order of magnitude. The results, however, show that this comes with significant extra cost, both in terms of power consumption (Pi 4 uses up to 35% and 55% less power in total compared to Nano and TX2, respectively) and price (Pi 4 is typically 3 and 10 times cheaper than Nano and TX2, respectively).

3 CLASSIC VS CNN-BASED PERSON DETECTION

The literature review highlighted a knowledge gap in vision-based person detection on non-GPU embedded systems. Although (Kim et al., 2019) consideredd various person detection algorithms for embedded devices, there are a couple of key differences between their work and the following evaluation. First of all, the authors tested all their CNNs on an in-house proprietary dataset suitable for in-store applications. This is significantly different from the working environment (indoor and outdoor) considered in this paper, where people classification is more likely to be affected by occlusions, variability of the surroundings, and weather conditions. The second but perhaps most important difference is that the devices evaluated in this paper are cheap Raspberry Pi 3 and 4 without GPU, instead of the more expensive and power consuming NVidia Jetson TX2 used by previous authors. This section therefore will evaluate three well known approaches for person detection on Raspberry Pi. The results will provide a baseline for the improved architecture proposed in Section 4.

3.1 Evaluation Methodology

Three person detection solutions described in the related work are going to be implemented and evaluated on Raspberry Pi (see Figure 3). These are Viola & Jones (Viola and Jones, 2001), HOG detector (Dalal and Triggs, 2005) and YOLO v3 (Redmon and Farhadi, 2018). The first two are interesting because developed when standard computers had similar performances to modern Raspberry Pi 3 and 4. The more recent YOLO v3, instead, well represents the CNN-based object detectors available nowadays. In particular, for the Viola & Jones two implementations will be considered: one only for full-body detection, and another one integrating also also upper- and lower-body detections to deal with partial human occlusions. The evaluation discussed in this section refers to the off-the-shelf version of the detectors, i.e. the original algorithms proposed by their authors, without any significant variations or code adaptations other than those necessary to compile and run the software on the target system. Their performances are going to be evaluated with a particular focus on the quality of the detection and the processing time on Raspberry Pi.

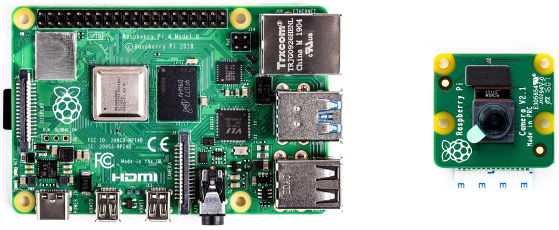


Figure 3: Raspberry Pi 4 (left) and Pi Camera v2.1 (right).

The experiments have been performed on images and ground truth bounding boxes from the Microsoft COCO 2017 dataset (Lin et al., 2014), which is a large-scale dataset addressing three core research problems in scene understanding: detecting non-iconic views of objects, contextual reasoning, and precise 2D localization. In particular, for the scope of this paper, the availability of non-iconic views has been considered a main factor in choosing COCO rather than other datasets (see Figure 4). Indeed, since the target application is person detection for safety in working environments, it is important to consider cases where people are occluded or assume non-canonical poses, differently from other datasets and studies.

In particular, two testing datasets have been downloaded from COCO. One contains 578 variable-sized images of people, as well as some images without them, and will be used throughout the paper for comparing all the algorithms. The second dataset contains 300 images similar to the previous one, but their size is fixed at 640 × 480 (i.e. VGA resolution). It will be used only in Section 3.4 to evaluate the detection performance depending on the size of the input images.

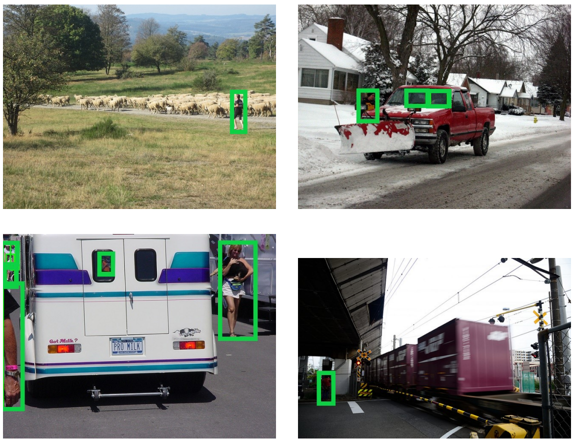


Figure 4: Some challenging images from the COCO dataset for the ”Person” class, highlighted by green rectangles.

A standard Intersection over Union (IoU) metric will be used to discern the bounding boxes of true positives, false positives, and false negatives, and from them assess the quality of the algorithms by computing Precision, Recall, and F1 score. The runtime perfomance of the detectors on the Raspberry Pi 3 and 4 will be then evaluated varying the scale of the input images. Finally, the quality of detection will be re-assessed to see how well the algorithms perform under the same time constraint.

3.2 Quality of the Detection

As expected, Figure 5 shows that the precision of the off-the-shelf YOLO is significantly better than the other algorithms, especially for larger IoU values. Viola and HOG detectors are relatively similar. The superiority of YOLO is even more evident in Figure 6 for the recall metric. In particular, when the IoU threshold is not too high, YOLO always find more than 50% of the people, while the other two methods perform drastically worse. The same differences are also captured by the F1 scores in Figure 7.

A consideration worth doing is that the YOLO v3 implementation used in this evaluation was originally trained to detect 80 classes of objects and not just people. This means the network’s discrimination power is superior to the other methods, resulting in less false positives. It should also be mentioned that HoG features are more suitable for human detection than Viola’s Haar features, even when three classifiers are combined to detect different body parts. This is because Haar features are most effective for face detection and less for other body parts.

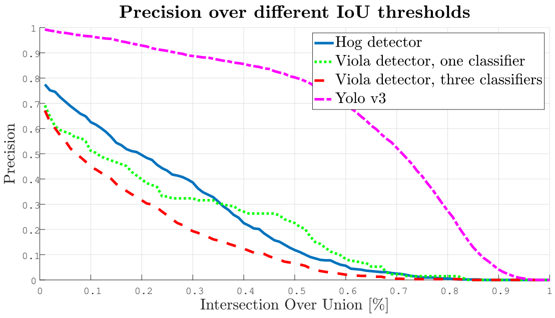


Figure 5: Precision vs IoU for Viola & Jones, HOG, and YOLO detectors.

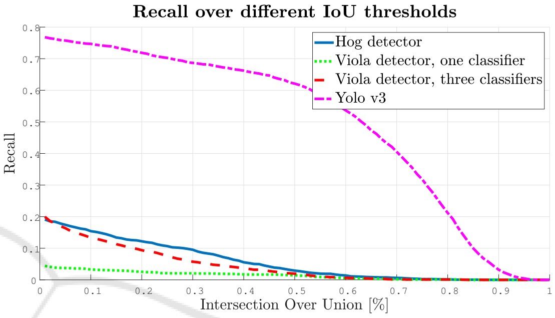


Figure 6: Recall vs IoU for Viola & Jones, HOG, and YOLO detectors.

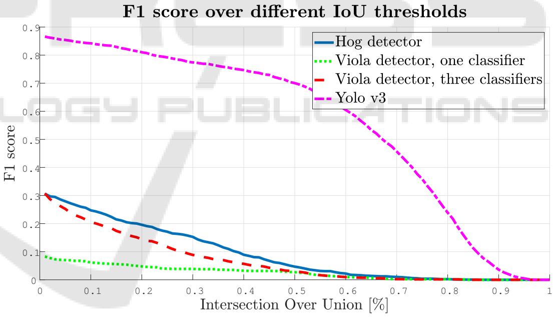


Figure 7: F1 score vs IoU for Viola & Jones, HOG, and YOLO detectors.

It is clear that the detection performances of HOG and Viola are less suitable for any safety application, where it is obviously important to avoid any misdetection. However, it is also worth remembering that the COCO dataset includes people in non-canonical poses, at multiple scales, often in the image’s background and occluded by other objects. This makes it particularly challenging and harder to achieve high scores compared to simpler datasets.

3.3 Runtime at Different Image Scales

The runtime performance of the three detectors were measured by iteratively decreasing the size of the input image. The experiments were performed with an OpenCV C++ implementation of the algorithms repeatedly feeding the classifiers with the same input

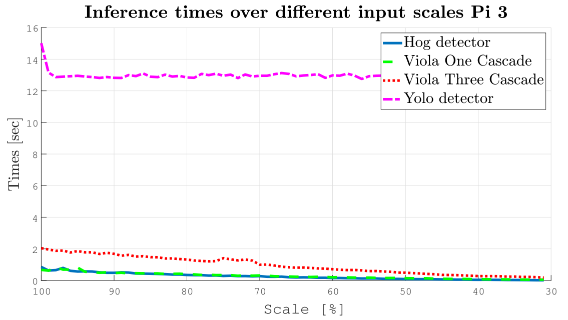


Figure 8: Detection time vs image scale on Raspberry Pi 3.

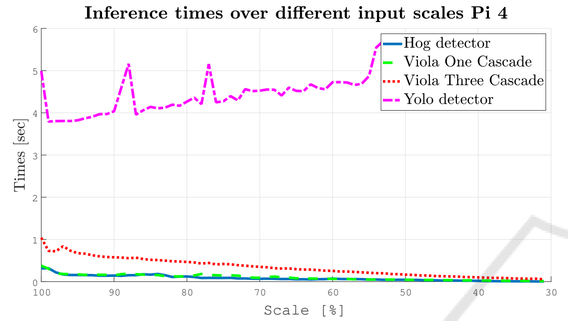


Figure 9: Detection time vs image scale on Raspberry Pi 4.

image after applying a linear scaling factor of 0.1 (i.e. *new size* = *old size* - 10%). The original input image was the standard VGA 640 × 480, as this is often used in industrial applications. These experiments were carried out on both Raspberry Pi 3 and 4.

The results in Figure 8 shows that the runtime of YOLO is considerably bigger than the other two algorithms, but also that it does not change by varying the image scale. This is because the input size YOLO v3's network is fixed to 416 × 416, so images of different sizes are always re-scaled automatically to that dimension. Furthermore, the 52 convolutional layers and the 40,549,216 parameters of its network require a huge number of operations to be performed by the CPU, making the detection extremely slow on a Raspberry Pi. This is also due to the large number of object classes handled by the original YOLO, although only the "Person" category was of interest for the current application.

The other classic detectors instead perform significantly better, and for them the detection time decreases linearly with the image scale. In particular, it can be noticed in the figure that the Viola detector based on three classifiers takes about three times longer to run compared to the single classifier version. A final consideration can be made about the significant improvement of the Rasberry Pi 4 in Figure 9, showing indeed how better hardware can provide faster computation. In this case, the Raspberry Pi 4's detection runs approximately three times faster than the older Pi 3, which is consistent with the manufacturer's information (Upton, 2019).

3.4 Quality at Different Image Scales

Previous results shows how decreasing the input size of the image decreases the runtime of all the person detectors but YOLO. It is then interesting to evaluate the quality of such detection according to the metrics used in Section 3.2 for varying image scales. For this comparison, the 300 images of size 640 × 480 of the second test dataset, mentioned in Section 3.1, will be used to calculate precision, recall, and F1 score at different image scales with a fixed IoU threshold. The literature usually sets the latter to $IoU = 0.5$ (Padilla et al., 2020). However, looking at Figure 6, it can be noticed that too many people would be missed using this value. Since in most safety applications it is more important to detect whether there is a person rather than missing any, the threshold in this case has been set to $IoU = 0.1$.

To perform this evaluation, the ground truth bounding boxes had to be scaled down together with the images. The lower the resolution though, the higher is the chance that a predicted detection intersects the ground truth. Moreover, the scaling process introduces a sort of image filtering that removes many details potentially confused as humans, decreasing the number of false positives. For this reason, the precision increases for all the detectors when the image scale decreases. Figure 10 illustrates these results, together with a confirmation in Figures 11-12 that the YOLO detector maintains its advantage compared to the other feature-based methods, similarly to what was already observed in Figures 5-7.

3.5 Online Application Results

Following the results in previous sections, an online application of video-stream person detection can be evaluated on the Raspberry Pi 3 and 4. As previously explained, YOLO's runtime is too high on these embedded devices and it is therefore considered in the following tables just for reference (as shown in Figures 8 and 9, YOLO would require 4 seconds to process one single frame on the Pi 4, and 13 seconds on the Pi 3). The IoU threshold has been set again to 0.1. A target runtime performance of 2 FPS was chosen, since this is sufficiently good for many CCTV applications (Cohen et al., 2009), although higher framerates might be desirable.

Given the target time $t = 0.5$ sec (i.e. 2 FPS), the necessary scale factors can be retrieved from Fig-

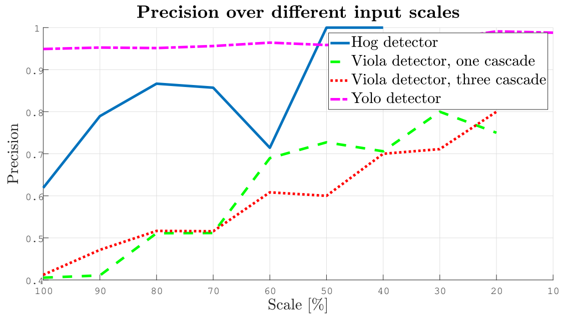


Figure 10: Precision vs image scale.

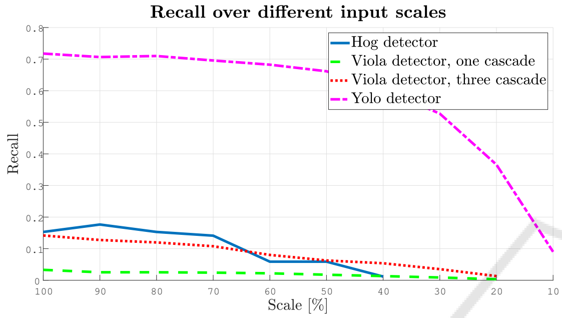


Figure 11: Recall vs image scale.

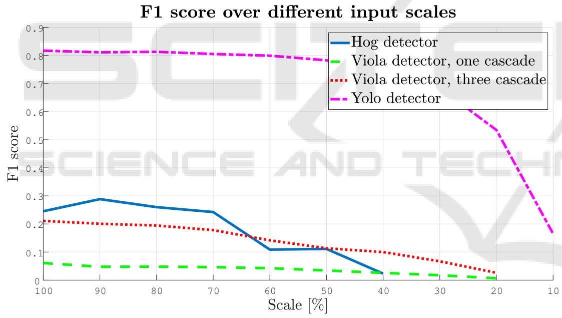


Figure 12: F1 score vs image scale.

ures 8 and 9 for each detector and Raspberry Pi version. From the scale factors, precision, recall, and F1 score can then be obtained from the detection performance in Figures 10-12. The results are reported in Tables 2 and 3, respectively.

As it can be noticed from the tables, the best solution among the classic and faster detectors is HOG, although its results are clearly not astonishing. The performances in term of recall and F1 score are indeed very poor, although, as previously stated, the COCO dataset sets a very challenging test benchmark. It is likely the results would be generally more satisfactory in many real-world applications.

Despite the above considerations, the detection results could and should be potentially be much better with a faster YOLO implementation, as hinted also by the same tables. In the remaining part of this paper, the focus will be therefore on a newly customised version of the latter, developing a light-weight version of the neural network for single class “Person” detection, suitable for online applications on Raspberry Pi.

Table 2: Comparison of different feature-based detectors for a target runtime of 2 FPS on Raspberry Pi 3.

	FPS	Scale	Precision	Recall	F1 score
HOG detector	2	**80%**	**87%**	**16%**	**26%**
Viola one casc.	2	80%	51%	2%	6%
Viola three casc.	2	50%	60%	6%	11%
YOLO v3	0.07 (max)	-	95%	71%	81%

Table 3: Comparison of different feature-based detectors for a target runtime of 2 FPS on Raspberry Pi 4.

	FPS	Scale	Precision	Recall	F1 score
HOG detector	2	**100%**	**62%**	**18%**	**26%**
Viola one casc.	2	100%	40%	4%	8%
Viola three casc.	2	70%	51%	11%	19%
YOLO v3	0.2 (max)	-	95%	71%	81%

4 CUSTOMISED YOLO

The previous section evaluated some off-the-shelf solutions for vision-based person detection on the Raspberry Pi 3 and 4. It has been shown that classic feature-based solutions are fast but perform poorly, in terms of precision and recall, compared to more recent CNN-based detectors like YOLO. The latter though is too computationally demanding and slow for online applications with such low-cost embedded systems. In this section, therefore, a YOLO-based network is proposed to reach a better compromise between detection’s quality and runtime performance. From now on this new architecture for embedded systems will be simply referred to as **e**YOLO.

4.1 Architecture Design

YOLO was initially designed to detect multiple classes of objects. Given that only people must detected in the current application, some simplifications can be made to reduce the number of parameters and speed up the computation, making eYolo suitable for low-cost and non-GPU embedded devices.

The main changes here considered are reducing the size of the network’s output and removing some of its layers. The original YOLO takes in input an 448×448 RGB image, while the output is encoded as a $7 \times 7 \times 30$ tensor. This means that the image is divided into a 7×7 grid, and each cell of such grid outputs a vector of length 30 composed by the following quantities:

- 2 bounding boxes in YOLO's format

  $$\vec{B} = \begin{bmatrix} P_C & B_X & B_Y & B_W & B_H \end{bmatrix}$$

  where P_C is a confidence value and the remaining ones are, in order, X-position, Y-position, width, and height of the bounding box;
- 20 entries, one for each of the 20 classes.

Since only one class needs to be detected, the output can be encoded by a simpler $7 \times 7 \times 5$ tensor, where each cell has an associated vector of length 5, equal to a single $\vec{B}$. This means the network would predict a single bounding box, which can only belong to the "Person" class.

As reported in Table 4, eYOLO is a sequential network. Its convolutional layers use a Leaky ReLU activation function and are all followed by batch normalisation in order to avoid overfitting. The input image has a resolution of $448 \times 448 \times 3$. MaxPooling layers and some convolutional ones reduce the feature map size to produce a $10 \times 10 \times 256$ layer. Finally, a combination of fully connected layers produces the 245 predictions that are re-shaped to produce the needed $7 \times 7 \times 5$ output.

During the training process, an optimised version of the original YOLO's loss function $\mathcal{L}$ is used:

$$\begin{aligned}\mathcal{L} \;=\; & \lambda_{coord} \sum_{i=1}^{S^2} \mathbb{1}_i^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ & + \lambda_{coord} \sum_{i=1}^{S^2} \mathbb{1}_i^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\ & + \lambda_{noobj} \sum_{i=1}^{S^2} \mathbb{1}_i^{noobj} [(p_i(C) - \hat{p}_i(C))^2] \\ & + \sum_{i=1}^{S^2} \mathbb{1}_{i,j}^{obj} [(p_i(C) - \hat{p}_i(C))^2]. \qquad (1)\end{aligned}$$

In the equation, x, y, w, and h refer to the top-left corner's coordinates, the width, and the height of the predicted bounding boxes, respectively. These parameters, as well as the boxes probabilities $p(C)$, are compared with the respective ground truths, identified by the ˆ (hat) symbol, for each cell of the $S \times S$ grid, where $S = 8$. The operator $\mathbb{1}_i^{obj}$ denotes if an object appears in cell i. The multiplier $\lambda_{coord} = 5$ increases the loss for cells containing objects, while $\lambda_{noobj} = 0.5$ decreases the loss for cells not containing objects.

Table 4: Sequence of layers in eYOLO.

Layer type	Filters/Filter size	Output shape
Conv2D	16/(7x7)	448x448x16
MaxPooling2D	(2x2)s2	224x224x16
Conv2D	48/(3x3)	224x224x48
MaxPooling2D	(2x2)s2	112x112x48
Conv2D	32/(1x1)	112x112x32
Conv2D	64/(3x3)	112x112x64
Conv2D	32/(1x1)	112x112x32
Conv2D	64/(3x3)	112x112x64
MaxPooling2D	(2x2)s2	56x56x64
Conv2D	64/(1x1)	56x56x64
Conv2D	128/(3x3)	56x56x128
Conv2D	64/(1x1)	56x56x64
Conv2D	128/(3x3)	56x56x128
MaxPooling2D	(2x2)s2	28x28x128
Conv2D	128/(1x1)	28x28x128
Conv2D	256/(3x3)	28x28x256
Conv2D	128/(1x1)	28x28x128
Conv2D	256/(3x3)	28x28x256
Conv2D	256/(3x3)	28x28x256
Conv2D	256/(3x3)s2	14x14x256
Conv2D	256/(3x3)	12x12x256
Conv2D	256/(3x3)	10x10x256
Flatten	-	25600
Dense	-	128
Dense	-	256
Dense	-	245
Reshape	-	7x7x5

Equation (1) differs from the original one in several ways. Indeed there is only one box for each cell, so there is no need to consider the inner sum for each $\vec{B}$. Furthermore, $\mathbb{1}_i^{obj}$ was originally computed for all the cells containing a ground truth, considering only the bounding box prediction with the biggest IoU. In (1) instead, $\mathbb{1}_i^{obj}$ includes only those cells with a ground truth different from zero, and since there is just one prediction per cell, it directly uses its bounding box to compute the loss. Finally, there is no need to compute the loss of multiple classes, since here there is only one, and the probability of the latter is equal to the probability of the bounding box itself.

4.2 Experimental Results

The network was trained on a dataset of 13,000 images from COCO. The dataset was split in 70% for training and 30% for validation. The test dataset, also from COCO, was the same with 578 images mentioned in Section 3.1 and later used for assessing the quality of the detection.

As shown in Figures 13-15, eYOLO outperforms both HOG and Viola detectors. As expected, its results are far from those of YOLO v3, which is significantly bigger and therefore better in the detection task, although the latter comes at the expenses of time. eYOLO instead is able to reach 1 FPS on the Raspberry Pi 3 and ∼3 FPS on the Pi 4. It is therefore much faster than YOLO v3 and slower than Viola and HOG, although better than the latter two in terms of detection quality, as evidenced by the same figures. Fig-

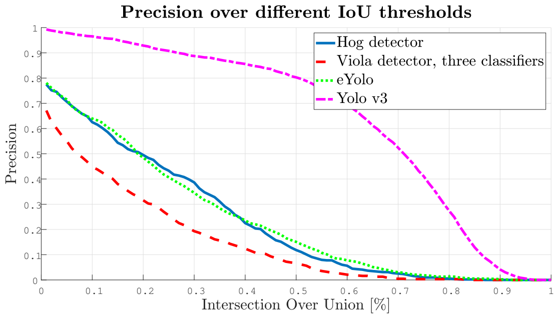


Figure 13: Precision vs IoU for eYOLO detector.

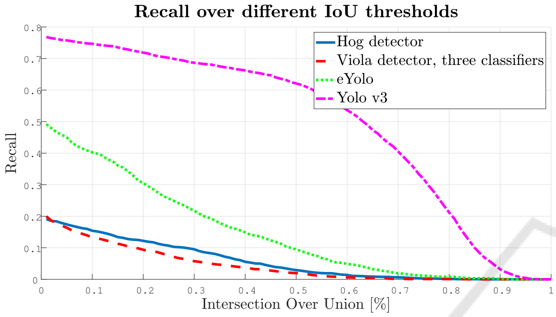


Figure 14: Recall vs IoU for eYOLO detector.

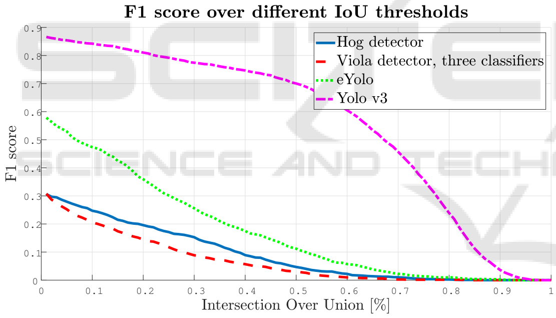


Figure 15: F1 score vs IoU for eYOLO detector.

ure 16 shows also the precision-recall (PR) curves for three different values of IoU, namely 0.1, 0.25, and 0.5. The average precision (AP) for the three cases are 29.47%, 9.34%, and 3.11%, respectively.

Finally, some qualitative results are illustrated in Figure 17 to compare the new eYOLO detector against the original YOLO v3 and HOG in typical working scenarios. From these examples it is evident that, despite the simplifications introduced to speed up the detection process, eYOLO performs significantly better than HOG and reasonably well compared to YOLO v3, although its reduced accuracy can affect the detection of distant people (e.g. false negatives on the rightmost image).

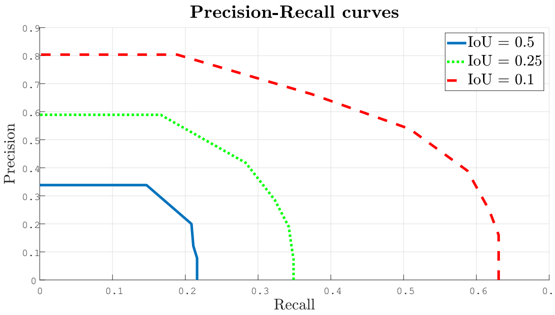


Figure 16: Precision-Recall curve for eYOLO detector.

4.3 Discussion

The results here presented are a compromise between speed and detection performances. Clearly the latter are not as good as those reported for other non-GPU solutions, such as Tiny-YOLO and YOLO-LITE (Huang et al., 2018), although the results for the single class detection in this work are not directly comparable to the multi-class problems addressed by other authors. It is important to stress also that the ARM-based architecture of the Raspberry Pi boards is significantly less powerful than the Intel CPU used in other studies, independently from the adoption of a GPU or not. The main difference of course is that ARM processors are designed for power-savvy applications on embedded systems.

A larger and more accurate network would produce better detections but would also decrease the runtime performance, while a smaller and faster network would loose significantly in terms of precision and recall. This is mainly noticeable in the eYOLO's loss of precision. Indeed, this detector often predicts bounding boxes that are too big, too small, or not-centered with respect to the ground truth, although generally placed nearby people. This means that, when the minimum IoU threshold increases, most of these quasi-correct bounding boxes will be treated as false positives, considerably decreasing the precision. When the IoU threshold is low instead, performances are much better. In Figure 18, some examples of such bounding boxes are shown together with their IoU score. The problem is mainly imputable to the network and its combination of hyperparameters, as previously explained. However, another possible cause is the training dataset that, besides individuals, contains also body parts and group of people, which likely confuses the network's learning. For specific applications, it might be worth to evaluate the network's performance under simpler scenarios where human appearance is more homogeneous.

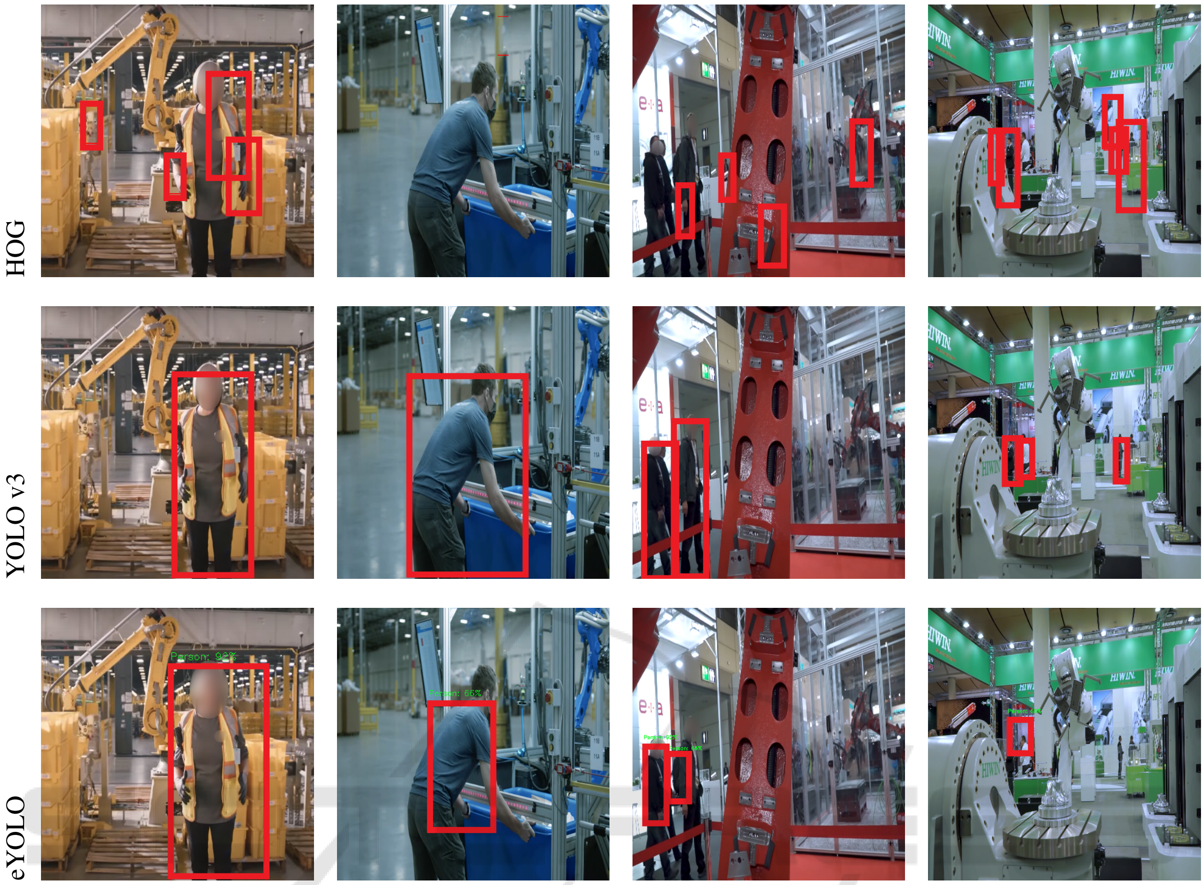


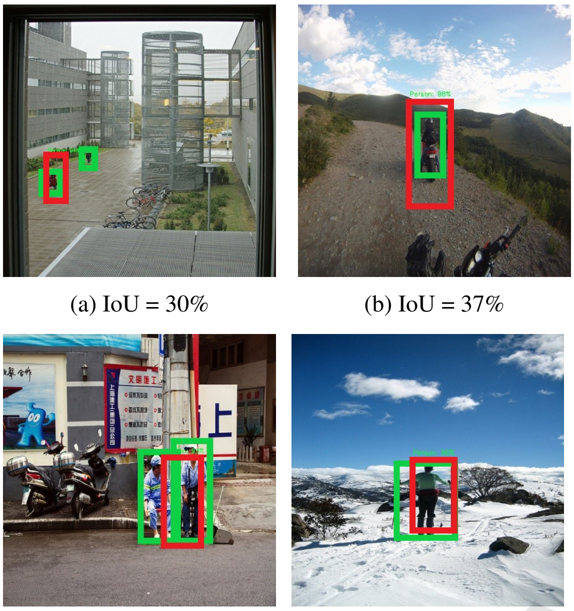
Figure 17: Person detection in typical working scenarios with HOG (top row), YOLO v3 (middle), and eYOLO (bottom).

5 CONCLUSIONS

This paper evaluated some classic computer vision-based algorithms for person detection on low-cost embedded system, with a particular focus on safety applications for work environments. It proposed also a new YOLO-based detector suitable for processing live videos with a Raspberry Pi. The main challenge with respect to previous works has been developing an embedded solution that does not rely on GPU- or powerful CPU-based systems, since these would inevitably affect its cost and energy consumption.

Three off-the shelf solutions have been first implemented and analysed to create a baseline for further improvements. Traditional machine learning approaches such as Viola & Jones and HOG proved to be good in terms of runtime performance but unsuitable for the actual detection task. A YOLO deep learning architecture, instead, showed to be very accurate in detecting people, but it required too much time to process a single frame, making this algorithm not suitable for embedded applications.

The comparison served as a baseline and inspiration to develop a new YOLO-based person detector, with several optimisations to reduce the original network's size and achieve a satisfactory runtime performance. The adopted strategy to re-design the network from the bottom up included a customised loss function and the batch generator for the training data. Developing a custom solution from scratch, however, is not straightforward. Indeed, seeking the right combination of hyperparameters and convolutional layers without increasing the inference time is a tedious process. Nevertheless, the final eYOLO architecture is a good compromise between hardware and software performance, and can serves as a complementary aid for many safety monitoring applications. On the one end, it is clear that the results obtained for large IoU are not always satisfactory, since the proposed bounding boxes are often too big or too small with respect to the ground truth. On the other hand, this is not always a problem, as the simple detection of any person in a frame, independently from the exact location, might be sufficient to trigger an opportune response by the



(a) IoU = 30%

(b) IoU = 37%

(c) IoU = 47%

(d) IoU = 60%

Figure 18: Example of person detections (ground-truth in green; eYOLO's output in red) and actual IoU percentages.

safety system.

There are several ways in which the work of this paper can be further improved to achieve better detection and runtime performances on low-cost embedded devices. In particular, it would be worth exploring a detection solution based on centroids rather than bounding boxes. As already reported in the paper, decreasing the number of parameters is very useful to increase the runtime performance of the network. A CNN detecting only the center of a person could achieve that, and at the same time provide enough information for safety purposes. Future work should also include a comparison to other methods (e.g. SSD, Tiny-YOLO) and adapt the most recent networks, such as YOLO v7, to embedded systems, possibly pushing their applications even further on low-cost/low-energy microcontroller-based devices.

ACKNOWLEDGEMENTS

This work has received funding from the EU H2020 research and innovation programme under grant agreement No. 101017274 (DARKO).

REFERENCES

Cohen, N., Gattuso, J., and MacLennan-Brown, K. (2009). *CCTV Operational Requirements Manual*. UK Home Office, Scientific Development Branch.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893.

Farouk Khalifa, A., Badr, E., and Elmahdy, H. N. (2019). A survey on human detection surveillance systems for raspberry pi. *Image and Vision Computing*, 85:1–13.

Girshick, R. (2015). Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587.

Huang, R., Pedoeem, J., and Chen, C. (2018). Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. In *IEEE Int. Conf. on Big Data (Big Data)*, pages 2503–2510.

Kim, C. E., Oghaz, M. M. D., Fajtl, J., Argyriou, V., and Remagnino, P. (2019). A comparison of embedded deep learning methods for person detection. In *Proc. of the 14th Int. Conf. on Computer Vision Theory and Applications (VISAPP)*.

Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., Li, Y., Zhang, B., Liang, Y., Zhou, L., Xu, X., Chu, X., Wei, X., and Wei, X. (2022). Yolov6: A single-stage object detection framework for industrial applications.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 740–755. Springer International Publishing.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing.

Padilla, R., Netto, S. L., and da Silva, E. A. B. (2020). A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Redmon, J. and Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. In *CoRR*, volume abs/1804.02767.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Sruthi, M., Poovathingal, M. J., Nandana, V., Lakshmi, S., Samshad, M., and Sudeesh, V. (2021). Yolov5 based open-source uav for human detection during search and rescue (sar). In *2021 International Conference on Advances in Computing and Communications (ICACC)*. IEEE.

Süzen, A. A., Duman, B., and Sen, B. (2020). Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *Int. Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*.

Upton, E. (2019). Raspberry pi 4 on sale now from $35. https://www.raspberrypi.com/news/raspberry-pi-4-on-sale-now-from-35/.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*.

Wu, W., Chen, C., and Lee, J. (2021). Embedded yolo: Faster and lighter object detection. In *Proc. of the Int. Conf. on Multimedia Retrieval (ICMR)*, pages 560–565.