DETECTING SOCCER BALLS WITH REDUCED NEURAL NETWORKS: A COMPARISON OF MULTIPLE ARCHITECTURES UNDER CONSTRAINED HARDWARE SCENARIOS

Douglas De Rizzo Meneghetti
Department of Electrical Engineering
FEI University Center
São Bernardo do Campo, SP 09850-901
douglasrizzo@fei.edu.br

Thiago Pedro Donadon Homem
Federal Institute of Education, Science and Technology of São Paulo
São Paulo, SP 05110-000
thiagohomem@ifsp.edu.br

Jonas Henrique Renolfi de Oliveira
Department of Electrical Engineering
FEI University Center
São Bernardo do Campo, SP 09850-901
jonashro@gmail.com

Isaac Jesus da Silva
Department of Electrical Engineering
FEI University Center
São Bernardo do Campo, SP 09850-901
isaacjesus@fei.edu.br

Danilo Hernani Perico
Department of Electrical Engineering
FEI University Center
São Bernardo do Campo, SP 09850-901
dperico@fei.edu.br

Reinaldo Augusto da Costa Bianchi
Department of Electrical Engineering
FEI University Center
São Bernardo do Campo, SP 09850-901
rbianchi@fei.edu.br

Jan. 4, 2021

ABSTRACT

Object detection techniques that achieve state-of-the-art detection accuracy employ convolutional neural networks, implemented to have lower latency in graphics processing units. Some hardware systems, such as mobile robots, operate under constrained hardware situations, but still benefit from object detection capabilities. Multiple network models have been proposed, achieving comparable accuracy with reduced architectures and leaner operations. Motivated by the need to create a near real-time object detection system for a soccer team of mobile robots operating with x86 CPU-only embedded computers, this work analyses the average precision and inference time of multiple object detection systems in a constrained hardware setting. We train open implementations of MobileNetV2 and MobileNetV3 models with different underlying architectures, achieved by changing their input and width multipliers, as well as YOLOv3, TinyYOLOv3, YOLOv4 and TinyYOLOv4 in an annotated image dataset captured using a mobile robot. We emphasize the speed/accuracy trade-off in the models by reporting their average precision on a test data set and their inference time in videos at different resolutions, under constrained and unconstrained hardware configurations. Results show that MobileNetV3 models have a good trade-off between average precision and inference time in constrained scenarios only, while MobileNetV2 with high width multipliers are appropriate for server-side inference. YOLO models in their official implementations are not suitable for inference in CPUs.

Keywords Object detection · Convolutional neural networks · Mobile robots · Constrained hardware

1 Introduction

The recent successes in the field of object detection are mostly due to the use of deep neural networks, more specifically convolutional neural networks (CNN). The underlying operations that compose CNNs are highly optimized for fast execution in graphics processing units (GPU). However, in some domains, GPUs may be unavailable and these processes must be executed in CPUs. One such domain is mobile robotics, comprised of systems such as autonomous terrestrial, aerial or underwater vehicles, as well as specialized units, such as humanoid robots. These systems may be subject to limitations regarding space, weight and energy consumption, constraining the robot's hardware to contain only CPUs, thus hindering the performance of systems based on deep learning techniques.

With these limitations in mind, this work presents an empirical and comparative analysis of the performance of recent CNN architectures proposed for constrained hardware settings and applied to object detection tasks, with the goal of emphasizing the speed/accuracy trade-off present in current models. Motivated by the creation of a near real-time soccer ball detection system to be executed in mobile robots equipped with embedded computers, that contain only CPUs under an x86 architecture and no graphics processing units, we train multiple models in the same annotated image dataset and compare their average precision (AP) in a test data set, as well as their inference times in both constrained and unconstrained hardware settings.

In preceding work [1], we presented a similar analysis of MobileNetV1, a CNN whose architecture was optimized for low latency inference in mobile phones, adapted for the task of object detection in an Intel NUC mini-PC equipped with a Core-i7 CPU. Here, we expand our work by analyzing MobileNetV2 [2] and MobileNetV3 [3], models that incorporate more recent advances in deep learning research to their architectures in order to achieve even lower latency in CPU-only mobile systems. We focus on benchmarking the performance of both architectures under different combinations of their width and input multipliers, two hyperparameters that control the underlying architecture of the MobileNets. In this work, we also include the YOLOv3 [4] and YOLOv4 [5] object detection models and their "tiny" counterparts, due to also being one-stage CNN-based object detectors.

With this study, we aim to fill an information gap related to the performance of the selected CNN models and object detection techniques, which are aimed at fast and accurate inference in mobile systems, when executed in a hardware platform typically used in mobile robots. In the case of the MobileNets, we investigate how a change in their width and input multipliers, two hyperparameters that control the overall topology of the model, affect their final accuracy and latency in the proposed task and hardware. For YOLO, we showcase their performance in the same task, as well as corroborate their high latency when executed in CPUs, especially less powerful ones.

We also expect, given our use case, to provide useful information to teams who participate in humanoid soccer competitions, regarding the capabilities of the selected models to perform near real-time inference with acceptable precision in a dataset with relevant applications to the competition. More broadly, this study is also aimed towards guiding the choice of a low latency, high accuracy object detection model to be executed in embedded computers with an x86 architecture, with a focus on recent MobileNet models that perform object detection using the Single Shot MultiBox Detector (SSD) [6] method, and multiple YOLO and TinyYOLO implementations.

This work also investigates the latency of the selected implementations of the models by gathering their inference times when processing videos in multiple input resolutions. These measurements are done in both constrained and unconstrained environments, providing readers with more relevant results to make decisions regarding the choice of neural network model to use in a single-object detection system under local, mobile, constrained hardware scenarios, but also accounting for situations in which remote and/or unconstrained hardware configurations may be available.

The text is organized as follows: section 2 lists the recent advances in the state-of-the-art in object detection, as well as techniques to create smaller network topologies while still maintaining high detection accuracy. We also describe the network architectures utilized in this work and their detection mechanisms. Section 3 depicts related works. The experimental methodology is presented in section 4. Results are presented and discussed in section 5. Lastly, section 6 provides the conclusions and future work.

2 Research Background

In recent years, object detection techniques have advanced at great pace due to the equally fast advances of deep learning and convolutional neural networks applied to computer vision [7]. Two-stage detectors such as Faster R-CNN [8] first generate region proposals and then detect objects only in the selected regions, while one-stage detectors such as YOLO [9] and SSD [10] generate bounding box coordinates and class predictions at the same time, with YOLO using only convolutional layers for this task.

More recently, effort has centered around building strategies for efficiently scaling network models, reaching a trade-off between FLOPS, number of trainable parameters and accuracy. The MobileNetV3 architecture [3] has been partially achieved via hardware-aware neural architecture search techniques [11, 12], while AmoebaNet's architecture [13], which achieved state-of-the-art classification accuracy on ImageNet [14], was evolved using evolutionary algorithms.

Other techniques attempt to shrink or expand the dimensions of convolutional layers using hyperparameters. MobileNetV1 [10] introduced the width multiplier and input resolution parameters, discussed later in the text, while EfficientNet [15] and EfficientDet [16] use a compound coefficient to scale all three dimensions of convolutional layers in order to maximize the network's accuracy. This, allied with neural architecture search, introduced the current state-of-the-art in image classification and object detection using CNNs.

2.1 MobileNets

MobileNets [10] are convolutional neural network architectures whose number of trainable parameters can be controlled by two hyperparameters. The first is the *width multiplier* $\alpha \in (0, 1]$, which controls the number of channels in each layer of the network. Smaller values of α reduce the number of parameters in each layer of the network uniformly, also reducing computational cost. The second parameter is the *resolution multiplier* $\rho \in (0, 1]$, which is used to reduce the resolution of the input images and, consequently, the number of operations throughout all layers of the network.

Additional features introduced in MobileNetV1 are batch normalization [17] for learning stabilization, as well as depthwise-separable convolutions [18], a convolution operation that uses fewer parameters to achieve comparable results to regular convolutions.

MobileNetV2 [2] advanced the state-of-the-art by introducing linear bottleneck layers in the network, reducing the size of the inputs in subsequent layers while preventing information from being lost by non-linear activation functions. The ReLU6 non-linearity [19] was chosen instead of regular ReLU to prevent loss of information when calculations with low-precision data types are performed.

Finally, the MobileNetV3 [3] builds upon MobileNetV2 and MnasNet [12] by using Mobile Neural Architecture Search (MNAS) [12], an algorithm that manipulates blocks of layers in the network, with the goal of maximizing accuracy while ensuring inference is conducted under a latency budget. The architecture is further simplified by the use of the NetAdapt algorithm [11], which reduces the number of filters in each layer of each block created by MNAS, further reducing network latency while trying to minimally affect network accuracy. For each iteration that either NAS algorithm is applied to the current network, its inference latency is measured in a mobile phone to ensure that the resulting model architecture is optimized for that type of hardware [3].

Both MobileNetV2 as well as the gradual enhancements performed in MobileNetV3 have their latency measured in the ARM-based processor of the Google Pixel 1 [2, 3]. In this work, we expect the performance gain achieved by these enhancements to also be visible in our target hardware, which differs from the original hardware the networks were tested on.

2.2 Single-Shot MultiBox Detector

The Single-Shot MultiBox Detector (SSD) [6] is a technique that utilizes a convolutional neural network, called the base network, combined with multiple subsequent convolutional filters of different sizes, to perform detection under different scales and aspect ratios in multiple regions of an input image. The feature maps of the base network may be pretrained in a classification or detection class. When training the network for a detection task, SSD employs techniques such as data augmentation and hard negative mining for faster training, as well as a loss function that is a weighted sum of both localization and classification losses.

2.3 You Only Look Once

You Only Look Once (YOLO) [9] simplified the object detection problem, which was then composed of a region proposal step followed by an image classification step, to a single regression step composed of bounding box coordinates and class probabilities. YOLOv2 [20] introduced the use of anchor boxes to the algorithm, a technique that allowed the detection of multiple objects with different aspect ratios in the same quadrant of an input image, using only convolutional layers.

YOLOv3 [4] introduces the prediction of bounding box coordinates across multiple scales and the use of residual layers [21] to speed up training, while YOLOv4 [5] adapts multiple data augmentation and feature extraction techniques to allow efficient training and inference of a model on a single GPU with 8 to 16 GB of VRAM.

3 Related Work

This work lies at the intersection of two problems, which may be considered complementary to each other. The first is the execution of computer vision tasks in embedded computers, which introduces complications regarding energy efficiency, storage and memory capacity and processing power. The second relates to compressing, reducing or simplifying a CNN model to achieve lower latency while sacrificing as little model accuracy as possible.

3.1 Neural network reduction strategies

One way to make neural network inference feasible in embedded computers is to use techniques that simplify existing network architectures before deployment in the constrained hardware. The goal of this approach is to either reduce the amount of memory necessary to store the model; the number of multiply–accumulate operations (MAC) in a forward pass of the network, or other more direct metrics, such as the wall-clock inference time. Given the possible loss in representation power of the model resulting from its reduction, a loss of accuracy in the target task is usually expected as trade-off.

One model reduction technique is network pruning [22], which consists in the detection and removal of low-valued weights from a network model, followed by the retraining of the network, promoting the re-purposing of the remaining weights. There have been works that have applied pruning in order to make network models more fitting to be executed on embedded systems [23, 24].

Another approach to reduce the memory necessary to store a neural network is the quantization of the network's parameters [25], a technique which consists in representing numbers from a range of values by a less precise subset of values, consuming less memory and possibly making operations simpler. A more extreme approach to quantization are the binary neural networks [26, 27], whose weights are represented by sets of binary or ternary values, enabling straightforward implementation of the models direct in the hardware level, at the cost of unstable training and reduced precision. Other works have already covered the applications of quantization [28, 29, 30] and binarization [29] of networks for use in embedded systems.

Model compression [31, 32], also known as knowledge distillation [33] is a process in which one shallow network (called the student) learns to imitate the output of one or more networks with more complex architectures (the teachers), achieving better results than if trained directly in the source dataset. This is usually done by using the predictions of the teacher models to classify or generate synthetic dataset, which is then used to train the student model. Thus, the student learns to approximate the function modeled by teacher ensemble instead of the function that represents the original data, which may be more accurate.

There have also been works that focus on representing neural network weights in alternative ways, *e.g.* using low-rank approximations of the weight matrices [34]. However, these techniques may require domain knowledge to be implemented without side-effects. Another option is to rethink the operations that compose the neural networks. One example of relevance to this work are depthwise separable convolutions [18, 35], which decompose the standard convolution operation applied to multiple channels into multiple single-channel convolutions, followed by a single pointwise convolution that joins the results from multiple channels. Depthwise separable convolutions are used in MobileNetV2 and V3, which are analyzed in this work.

Neural architecture search is yet another family of techniques aimed towards adapting a network's architecture according to an optimization objective and algorithm. A relevant example to this work is MNAS [12], a reinforcement learning-based technique which aims to maximize model accuracy and minimize wall-clock latency in mobile devices.

For individual works that build upon the aforementioned network reduction techniques, the reader is pointed to surveys on the topic [36, 34, 37].

3.2 CNNs in embedded systems

Due to the relative novelty of the use of deep convolutional neural networks in computer vision tasks, their transposition to more specialized and energy-efficient hardware is an ongoing topic of research. One category of such specialized hardware are the portable NVIDIA Jetson development boards, equipped with an ARM CPU and CUDA-capable GPU. There have been MobileNetV1 [38] and V2 [39] implementations aimed towards object detection tasks in Jetson boards, as well as Fast R-CNN [40] and Faster R-CNN [41].

Previous work [42] has analyzed the inference time of multiple CNNs in image classification tasks under the Jetson TX1. A linear upper bound was observed when comparing model accuracy with inference time, i.e. models that took longer to perform a forward pass on an image were more accurate and the relation between both quantities was linear.

In order to minimize memory usage and accelerate convolutions in the Jetson TX1, another work [23] explores the representation of CNN weights as sparse matrices and the use of different sparse matrix multiplication algorithms as alternatives to the convolution operation.

There have also been efforts to implement CNNs in field-programmable gate arrays (FPGA), such as the proposal of toolsets and methods to convert, accelerate and analyze the latency and precision of CNN operations in the new hardware [43, 44]. A common trend in these works is the observation of a negligible loss of precision due to the quantization of the parameters of the network [29, 44], which is greatly offset by a considerable gain in terms of energy efficiency and latency. Some works report a speed up of 10 to 16 times in inference when compared to executing the same architectures in the aforementioned Jetson boards [45, 44].

Another work [46] similar to this one has analyzed YOLOv3, YOLOv4, TinyYOLOv3, TinyYOLOv4 and a custom network both in a server equipped with an Intel Xeon E5-2678 v3 and an NVIDIA GeForce GTX 1080Ti, as well as a Raspberry Pi 3B. Like this work, they have reported prohibitively high inference times in their constrained hardware, ranging from 3.225 seconds to 5.555 seconds when doing a forward pass in a single image. Unlike them, we provide an alternative with faster inference times for our constrained hardware.

3.3 Discussion

This section has presented related works in two main areas of which this work is an intersection of: neural network reduction techniques and the execution of neural networks in embedded devices.

This work differs from its related works by focusing on the analysis of convolutional network architectures applied to object detection tasks in mobile computers with an x86 architecture processor. More specifically, we analyse the MobileNetV2 and V3 neural networks in a different target hardware than their original one and how their precision and inference times differ when a set of hyperparameters that dictate their architectures (input and width multipliers) change. We also provide the same analysis for the YOLOv3, YOLOv4, TinyYOLOv3 and TinyYOLOv4 networks, due to their prominence in object detection tasks with high frame rate needs.

4 Experimental Methodology

This study is motivated by the development of a computer vision system for an autonomous humanoid robot, which provides the robot with the capabilities of detecting soccer balls in a time that is compatible with the dynamics of the game. The target hardware of the vision system is CPU-only embedded computer of the x86 architecture, such as a mini-PC. We identify two properties that are desirable in an object detection system, given the aforementioned requirements. The first is high detection accuracy, a desirable feature in all object detection systems. The second is low latency, providing the robot with updated object locations under actionable intervals of time.

These two properties can be considered opposite, since higher accuracy usually indicates a more complex model, with more layers and operations. Also, when working with mobile robots and embedded computers, there is a tendency towards employing simpler machine learning models, in order to achieve acceptable operating thresholds with regards to time and power consumption, while sacrificing as little precision as possible.

To compose a detection system with low latency and high accuracy, we have selected multiple recent neural network architectures developed for computer vision tasks which attempt to reach a balance between both properties. Models were selected according to their performance in object detection tasks, their availability for the x86 processor architecture and homogeneity in pretraining in the MSCOCO [47] dataset.

The remainder of this section describes the selected neural network models; the image dataset used to train the models and analyze their performance; the humanoid robot our experiments are geared towards (with no loss of generality over other x86 embedded systems); and the hardware used for training, as well as the different hardware used for testing the models under constrained and unconstrained scenarios, providing even more broad and informative results.

4.1 Network architectures and training

Twenty MobileNetV2 configurations were tested by modifying the values for the width and resolution multipliers. For the width multiplier, the values 1, 0.75, 0.5 and 0.35 were used, resulting in networks with 3.47, 2.61, 1.95 and 1.66 million trainable parameters, respectively. The values used for the resolution multiplier were chosen so that the input resolution of the network is equal to 224, 192, 160, 128 and 96. The combined values of both hyperparameters resulted in a total of twenty models that were trained using the soccer ball dataset, described on section 4.2.

MobileNetV3 [3] models are composed of the "Large" and "Small" variants, both with width multipliers of 1 and 0.75, possessing 5.4 (Large, $\alpha = 1$), 4 (Large, $\alpha = 0.75$), 2.9 (Small, $\alpha = 1$) and 2.4 million (Small, $\alpha = 0.75$) trainable parameters, as well as minimalistic versions of both variants with $\alpha = 1$, possessing 3.9 and 2 million parameters. Minimalistic models do not contain the more advanced squeeze-and-excite units, hard-swish, and 5×5 convolutions operations from the non-minimalistic counterparts. YOLO models are composed of the v3 [4] and v4 [5] versions of the neural networks, as well as their "tiny" counterparts.

The MobileNet implementations selected for this work are provided in the TensorFlow Object Detection API [48], while YOLO is provided in the original paper implementation [5]. All models used pretrained weights learned in the COCO dataset [47].[1]

4.2 Image dataset

The dataset used in this work [49][2] consists of 4364 images in 1920×1080 resolution, collected from the point-of-view of the humanoid robot described in section 4.4. A fish-eye lens is used to maximize the field of view of the robot and so all images in the dataset also inherit this feature.

Of the 4364 images, 4014 compose the annotated training set and 250, the annotated test set. In these sets, the soccer balls visualized by the robot have been marked with bounding boxes. The training and test sets were collected from different sets of videos, with the purpose of minimizing data correlation.

Each image contains a single soccer ball, captured under multiples lighting conditions, as well as at different angles and distances from the camera. There are pictures of both stationary and moving soccer balls. Figure 1b presents examples of the dataset.

4.3 Training procedure

The models were trained in a server with Intel Xeon Gold 5118@2.3 GHz processors totaling 48 CPUs, 192 GB of RAM and an NVIDIA Tesla V100-PCIE with 16 GB of memory, running CentOS 7.6.1810. The MobileNet models were trained for 50000 training steps. The YOLO and TinyYOLO models were trained for a total of 6000 training steps, following recommendations from the original developers of the model, given the number of classes to be detected.

All MobileNetV2 models were trained using batches of 32 images and the RMSProp optimizer with initial learning rate of $4 \cdot 10^{-3}$, an exponential decay schedule with a decay factor of 0.95 and a momentum coefficient of 0.9. All MobileNetV3 models were trained using batches of 32 images, stochastic gradient descent with initial learning rate of 0.4, a cosine decay schedule and a momentum coefficient of 0.9.

YOLO and TinyYOLO models were trained using batches of 64 images, stochastic gradient descent with a momentum coefficient of 0.9. YOLOv3 and TinyYOLOv3 models used a learning rate of 10^{-3}, while YOLOv4 and TinyYOLOv4 used a learning rate of $2 \cdot 10^{-3}$. Two step decays at 80% and 90% of the training were applied to these learning rates.

4.4 Humanoid Robot

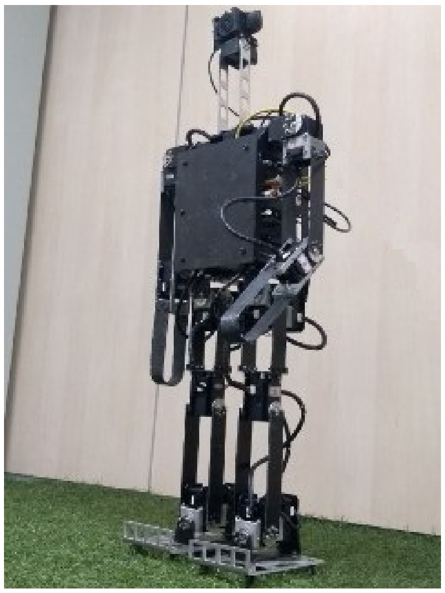
Our use case for this work is a mobile humanoid robot built for the task of playing soccer. One of its capabilities is the detection of soccer balls through visual inputs. The robot weighs about 5.9 kg and measures 81 cm in height. It is composed of 19 Dynamixel servomotors (a combination of MX-64, MX-106 and XM430 models), totaling 19 degrees of freedom. The humanoid robot uses a Genius WideCam F100 (Full HD) camera for image capture and a CH Robotics UM7 orientation sensor. The center of mass has a height of 36.1 cm and the robot has a foot area of 174 cm^2. Other measurements include 39.5 cm of shoulder length, 38.5 cm of leg height, 18.8 cm of neck height and 38.5 cm of arm length. The robot is equipped with an Intel NUC Core i7 mini-PC. A picture of the robot is presented in Figure 1a.

4.5 Performance metrics

To evaluate the performance of the selected models in both time and correctness, we gathered their average precision and inference time during the experiments. The meaning of these two measures is explained in this section.

[1]In order to facilitate the replication of these results and encourage the development of similar approaches by other researchers, the software used in this paper is available for download at: `https://github.com/douglasrizzo/JINT2020-ball-detection`.

[2]The dataset was also made available at `http://ieee-dataport.org/open-access/open-soccer-ball-dataset`


(a) One of the teen-sized robots of the RoboFEI team

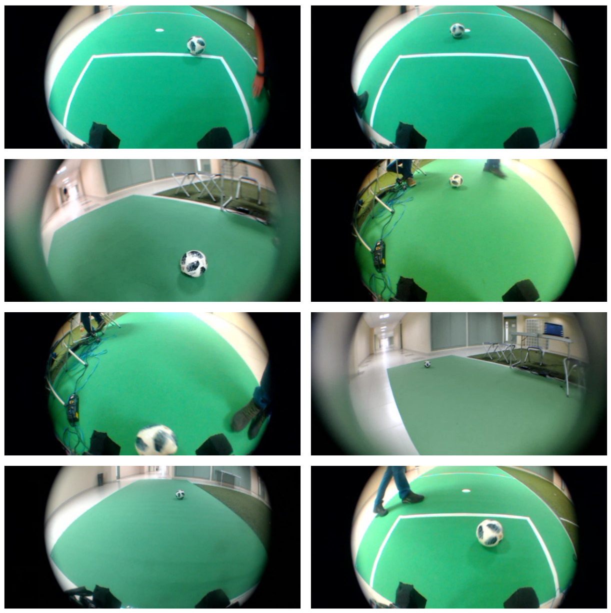

(b) Examples of images from the soccer ball dataset

Figure 1: Image dataset and humanoid robot used to collect it.

4.5.1 Average precision

In this work, both ground-truth annotations and predictions are represented as rectangular bounding boxes. Given a predicted rectangle A and a ground truth rectangle B, it is possible to calculate a similarity measure between both rectangles, called *intersection over union* (IoU),

$$IoU = \frac{(A \cap B)}{(A \cup B)}$$

When the IoU among both rectangles surpasses a certain threshold t, the detection represented by A is considered a true positive (TP). Otherwise, if no other predicted rectangle has an acceptable $IoU \geq t$ wrt. B, the failed detection is considered a false negative (FN). Lastly, all predicted boxes which have $IoU < t$ for all ground truth boxes in an image are considered false positives (FP). These quantities allow us to calculate a detector's precision and recall,

$$Precision = \frac{TP}{TP + FN}, \quad Recall = \frac{TP}{TP + FP}.$$

By ordering all predictions in order of decreasing confidence and calculating precision and recall at each new prediction, we arrive at a monotonically decreasing precision/recall (PR) curve. Average precision (AP) is the area under the PR curve. It is a single-valued metric which summarizes model performance when retrieving objects of a single class.

4.5.2 Inference time

In this work, inference time is defined as the time it takes a detection model to generate predictions for a given input image, in milliseconds. To get a more accurate measure, all models operated over a 30-second video taken from the robot's point-of-view and the inference time of the model is taken as the average time over all frames of the video. Experiments were conducted in the same video at the native resolution of 1920×1080 pixels, as well as other versions scaled down to 1280×720, 640×480 and 480×360 pixels. All networks then processed these four versions of the same video and the mean inference time over all frames of each video for each network was recorded.

Table 1: Average precision (higher is better) and inference time in milliseconds (lower is better) of the trained models. The five best in each category are marked in bold.

Network	Width mult.	Input res.	AP	Inference time (ms) Core i5	V100	Xeon
MobileNetV2	0.35	96	0.4065	99.964	65.37	64.959
		128	0.7095	101.166	51.062	49.642
		160	0.6304	88.651	52.642	50.97
		192	0.6756	87.006	53.613	52.232
		224	0.4984	**78.553**	42.852	**41.703**
	0.5	96	0.4065	91.403	58.919	57.626
		128	0.6986	81.08	44.529	43.388
		160	0.3361	91.775	58.288	58.616
		192	0.0944	116.076	65.629	64.828
		224	0.3253	**78.624**	42.759	43.18
	0.75	96	0.7284	86.569	51.065	51.528
		128	0.6954	84.159	**42.097**	**41.866**
		160	0.6679	81.351	**41.883**	**42.309**
		192	0.6952	**78.699**	**42.347**	**41.776**
		224	0.7874	85.186	48.343	47.854
	1	96	**0.8133**	122.853	56.992	57.83
		128	0.7672	82.277	46.921	47.799
		160	**0.8597**	88.886	52.278	52.569
		192	0.3632	110.75	61.263	60.052
		224	**0.8177**	79.547	42.438	**42.183**
MobileNetV3 (large min.)	1	224	0.6007	85.808	58.706	59.581
MobileNetV3 (large)	0.75	224	**0.8847**	89.362	63.515	63.703
MobileNetV3 (large)	1	224	0.6875	120.045	88.017	91.369
MobileNetV3 (small min.)	1	224	0.6024	**79.142**	48.68	49.236
MobileNetV3 (small)	0.75	224	0.7067	**60.654**	49.328	47.741
MobileNetV3 (small)	1	224	**0.8651**	96.975	70.689	70.017
TinyYOLOv3			0.3381	588.235	**33.557**	85.47
TinyYOLOv4			0.3504	714.286	**29.851**	119.048
YOLOv3			0.1355	5000	44.248	588.235
YOLOv4			0.1419	5000	50	833.333

4.6 Constrained hardware for inference

The constrained hardware configuration in which the inference time of the models was captured is equipped with an i5-4210U CPU @ 1.70GHz and 8 GB of RAM and no GPU, in line with the hardware typically used by an autonomous mobile robot. For comparison purposes, the same experiments were performed in the training computer, under GPU and CPU-only settings. In all cases, when CPU-only experiments were executed, all CPU cores were allowed to be utilized.

5 Results

Table 1 displays the AP of all trained models in the test set, as well as the inference time in milliseconds for different hardware configurations. In this table, the inference time was calculated with the videos in their native 1920×1080 resolution. These results allow us to conclude that the canonical YOLOv3 and YOLOv4 implementations, as well as their tiny counterparts, are not optimized for inference on CPUs, achieving the highest inference times of all models in the Intel Core i5-4210U. In fact, it is stated by the author of the model [5] that their implementations are optimized for inference in single GPUs. This can be seen by the comparatively low times achieved in the Tesla V100 GPU, especially by the TinyYOLO models, which achieved the lowest inference times of all models.

As for the MobileNet models, we can see that MobileNetV3 and MobileNetV2 with $\alpha = 1$ achieved the highest AP in the test data set. However, with the high variation in inference times between all combinations of hyperparameters, the results from Table 1 alone do not provide enough information to compare model performances. To remedy that, we

Table 2: Normalized scores of the trained models under different hardware. Higher is better. A normalized score of 1 indicates that the model performed the best under that hardware, compared with the others.

Network	Width mult.	Input res.	Normalized score Core i5	V100	Xeon
MobileNetV2	0.35	96	0.349	0.323	0.323
		128	0.602	0.721	0.737
		160	0.61	0.622	0.638
		192	0.666	0.654	0.667
		224	0.545	0.604	0.617
	0.5	96	0.382	0.358	0.364
		128	0.74	0.814	0.831
		160	0.314	0.299	0.296
		192	0.07	0.075	0.075
		224	0.355	0.395	0.389
	0.75	96	0.722	0.74	0.729
		128	0.709	**0.857**	**0.857**
		160	0.705	0.828	0.814
		192	0.758	**0.852**	**0.858**
		224	0.793	0.845	0.849
	1	96	0.568	0.741	0.726
		128	**0.8**	**0.849**	**0.828**
		160	**0.83**	**0.853**	**0.844**
		192	0.281	0.308	0.312
		224	**0.882**	**1**	**1**
MobileNetV3 (large min.)	1	224	0.601	0.531	0.52
MobileNetV3 (large)	0.75	224	**0.85**	0.723	0.716
MobileNetV3 (large)	1	224	0.492	0.405	0.388
MobileNetV3 (small min.)	1	224	0.653	0.642	0.631
MobileNetV3 (small)	0.75	224	**1**	0.744	0.764
MobileNetV3 (small)	1	224	0.766	0.635	0.637
TinyYOLOv3			0.049	0.523	0.204
TinyYOLOv4			0.042	0.609	0.152
YOLOv3			0.002	0.159	0.012
YOLOv4			0.002	0.147	0.009

calculate a performance score for each neural network in each hardware setting $p_{m,h} = \frac{AP_m}{t_{m,h}}$, where AP_m represents the AP of network m (a value that is hardware-independent) and h, the hardware setting the inference time t of model m was gathered from. Then, the performance scores of all models in the same hardware are normalized by the highest performance score in that hardware, leading to the normalized score $s_{m,h} = \frac{p_{m,h}}{\max_\eta p_{\eta,h}}$. Achieving a normalized score $s_{m,h} = 1$ means that model m had the highest AP/inference time ratio off all models in hardware h.[3]

Table 2 presents the normalized scores of all models. Overall, MobileNetV2 models with width multipliers $\alpha \in \{0.75, 1\}$ had the best scores of all MobileNetV2 models in all hardware settings. We can also see that the five best models in unconstrained hardware settings are the same for both CPU and GPU. However, when operating under the Intel Core i5 4210U processor, both MobileNetV3 models (large and small) with $\alpha = 0.75$ achieved the highest scores, making MobileNetV3 models a viable option for an object detection system that operates under constrained hardware settings, whereas they did not exhibit the same performance in GPUs.

[3]This score may easily break or be less informative if one neural network in the sample has disproportionately low inference time or high AP. However, given the well-behaved values presented in Table 1, we consider the use of the proposed score appropriate for the purposes of our analysis.

Table 3: Inference time of YOLO and TinyYOLO models when processing videos of multiple input resolutions.

Network	Hardware	Inference time mean	Inference time std. dev.
TinyYOLOv3	Tesla V100	41.957	10.022
	Xeon Gold 5118	88.983	3.369
	i5-4210U	588.235	0.000
TinyYOLOv4	Tesla V100	38.808	9.136
	Xeon Gold 5118	114.529	6.094
	i5-4210U	714.286	0.000
YOLOv3	Tesla V100	46.726	4.081
	Xeon Gold 5118	588.235	0.000
	i5-4210U	5000.000	0.000
YOLOv4	Tesla V100	50.385	0.676
	Xeon Gold 5118	817.308	32.051
	i5-4210U	5000.000	0.000

5.1 Performance with different input video resolutions

This section presents the inference times in milliseconds of all model implementations when processing input videos of several resolutions (1920×1080, 1280×720, 640×480, 480×360). Table 3 presents the mean and standard deviation of the inference time of all YOLO models for the tested hardware configurations. Overall, all YOLO and TinyYOLO models achieved low standard deviation in this test, implying that their implementation [5] is indifferent to the input resolution of images.

As for the MobileNet results, we first discuss the distributions of results in the three hardware settings. In total, 104 values were collected in each setting (twenty V2 and six V3 models applied to videos in four resolutions), with the following means and standard deviations: $\mu_{\text{i5}} = 68.292$, $\sigma_{\text{i5}} = 17.374$, $\mu_{\text{V100}} = 47.24$, $\sigma_{\text{V100}} = 8.756$, $\mu_{\text{Xeon}} = 47.009$ and $\sigma_{\text{Xeon}} = 8.981$.

Due to the similarity in the inference times collected from the NVIDIA Tesla V100 GPU and the Intel Xeon Gold 5118, we executed a two-sample Kolmogorov-Smirnov test between the two samples, with a result of $p = 0.97371$, indicating that the measurements collected in the 48 quad-core CPUs and the single GPU are similar with high statistical relevance. Because of that, in this section we only report results for the MobileNets in the NVIDIA Tesla V100 GPU and the Intel i5-4210U processor.

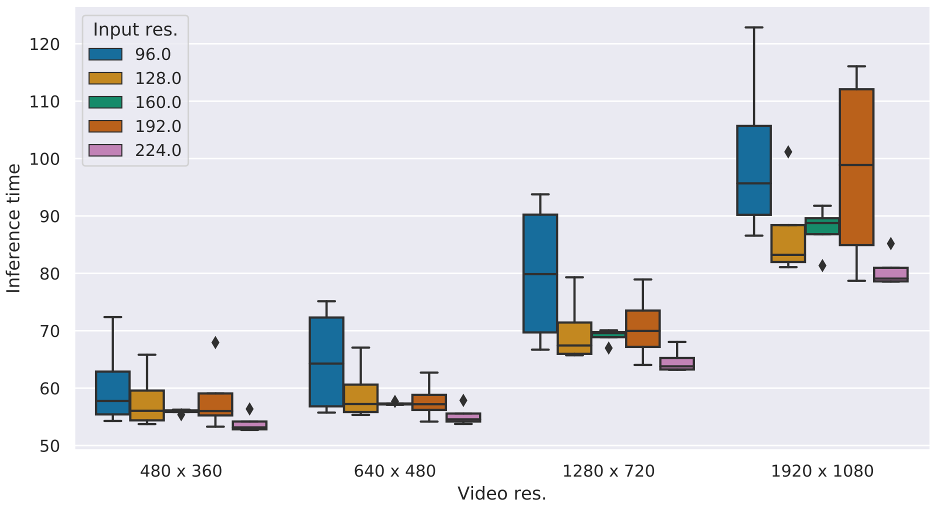
The distance between μ_{i5} and μ_{V100} is indicative of the performance lost by executing deep learning models in constrained CPUs, while a larger standard deviation on the CPU (σ_{i5}) indicate that there is a larger variation in network performance, given the resolution of the input video.

Figures 2 and 3 present a series of boxplots containing the inference time by frame in milliseconds for the MobileNets when processing the same video under multiple resolutions in the Intel i5-4210U CPU and the NVIDIA Tesla V100 GPU, respectively. The central line in each box represents the median value in the group; boxes represent the first and third quartiles of the data; and whiskers represent the minimum and maximum acceptable range, whereas dots outside the whiskers are considered outliers.

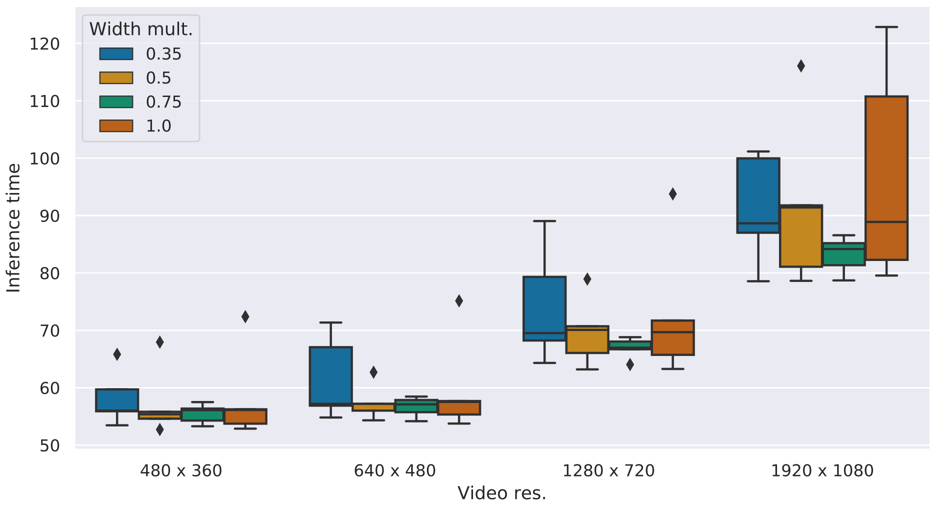
In all of the figures, it is possible to observe a gradual decrease in inference time as the resolution of the input videos decreases. This information may be relevant, as the largest input resolution used by a MobileNet model is 224×224. Furthermore, the implementations used in this work [48] already operate in downscaled images, with a resolution of 300×300 pixels. Both of these observations indicate that using a low-resolution input feed for object detection is a valid strategy to achieve lower inference times. This speedup is visualized in both constrained CPU and GPU settings.

To discover whether the distributions presented in Figures 2 and 3 can be considered different with statistical significance, a series of one-way analysis of variance (ANOVA) tests were performed, whose results can be viewed in Table 4. We can see that MobileNetV2 is sensitive to the input resolution hyperparameter only when inference is done on a GPU and not the constrained CPU.

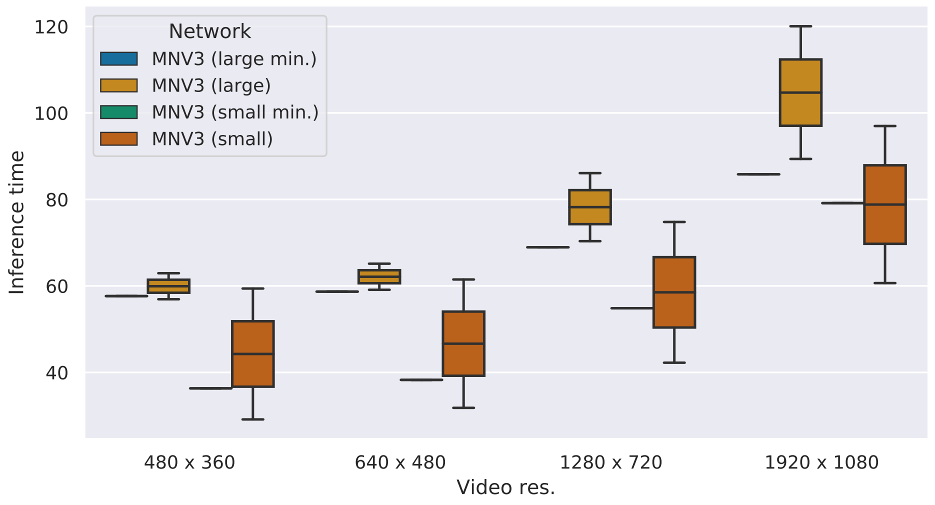
The other results in Table 4 also confirm that there is a difference in feeding video frames in different resolutions to the TensorFlow implementations of the MobileNets. MobileNetV2 demonstrated statistically significant differences both on the constrained CPU and the GPU, while MobileNetV3 demonstrated statistically significant differences only in the



(a) MobileNetV2 by input resolution on i5-4210U

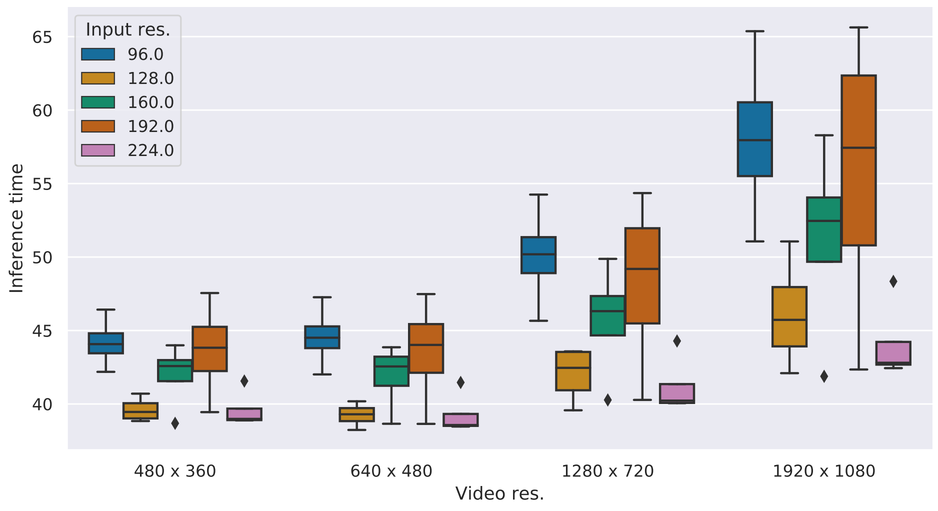


(b) MobileNetV2 by width multiplier on i5-4210U

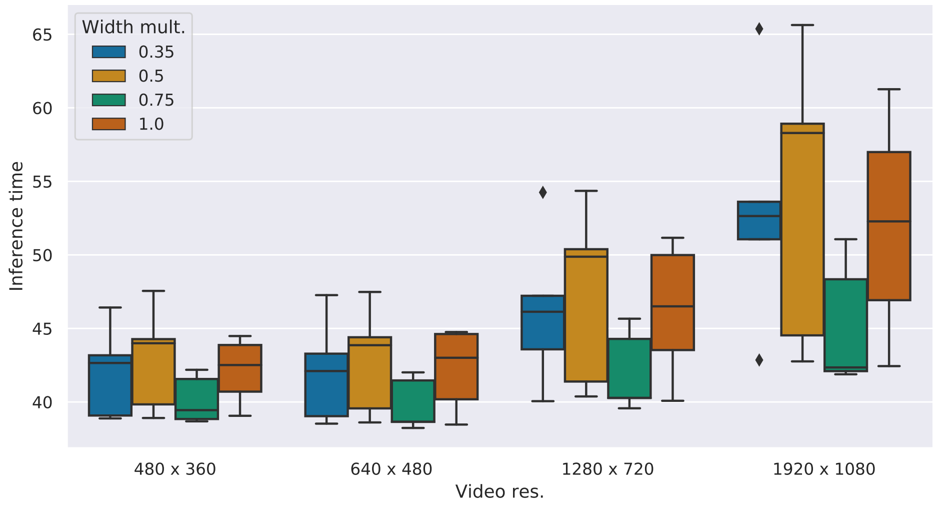


(c) MobileNetV3 on i5-4210U

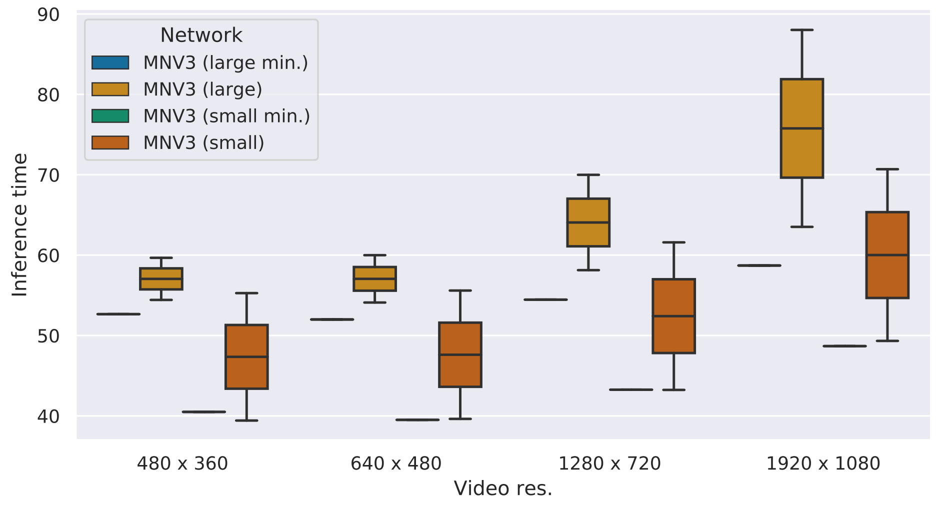
Figure 2: Inference time of MobileNetV2 and V3 models in the Intel i5-4210U.



(a) MobileNetV2 by input resolution on V100



(b) MobileNetV2 by width multiplier on V100



(c) MobileNetV3 on V100

Figure 3: Inference time of MobileNetV2 and V3 models in the NVIDIA Tesla V100.

Table 4: One-way analysis of variance of the inference times achieved by the MobileNets. Values in bold indicate results that were different with $p \leq 5\%$.

Network	**Parameter**	**Hardware**	F	p
MobileNetV2	Input res.	Core i5	1.476	0.2176
		V100	6.773	**0.0001**
	Width mult.	Core i5	0.639	0.5921
		V100	2.606	0.0577
	Video res.	Core i5	63.744	$\mathbf{1.0595 \cdot 10^{-20}}$
		V100	14.929	$\mathbf{9.8719 \cdot 10^{-8}}$
MobileNetV3	Width mult.	Core i5	2.471	0.1302
		V100	1.873	0.1848
	Video res.	Core i5	7.350	**0.0016**
		V100	1.885	0.1646

constrained CPU. Lastly, in all cases, a change in the width multiplier of the networks did not result in a significant change in their inference times.

To discover which combinations of the aforementioned parameters actually resulted in lower inference times for each neural network, the ANOVA tests were followed by a series of Tukey's Honestly Significant Difference (HSD) tests, whose results are displayed in Figures 4 and 5. Whenever there is no vertical overlap between the intervals covered by two pairs of lines, their distributions are considered different with a statistical significance greater than 95%. It can be seen on Figure 4a that MobileNets with input resolutions of 128 and 224 pixels achieve lower inference time than the ones with input resolutions of 96 and 192 pixels.

The other subfigures confirm that feeding video frames in a natively low resolution to the MobileNets result in lower inference times. In all cases, the optimal video resolutions are 640×480 and 480×360, in contrast with 1920×1080. In the case of MobileNetV2 on CPU, 1280×720 achieves a statistically significant compromise among the two aforementioned groups.

5.2 Discussion

The results presented in Tables 1 and 2 show that MobileNetV2 models with $\alpha \leq 0.5$ do not achieve the top results with regards to AP. While some of the V2 models do achieve low inference times (in bold in Table 1), the ANOVA results from Table 4 indicate that the differences in latency resulting from a change in the α hyperparameter are not statistically significant, which leads us to the conclusion that, under our experimental settings, MobileNetV2 models with $\alpha > 0.5$ should be prioritized.

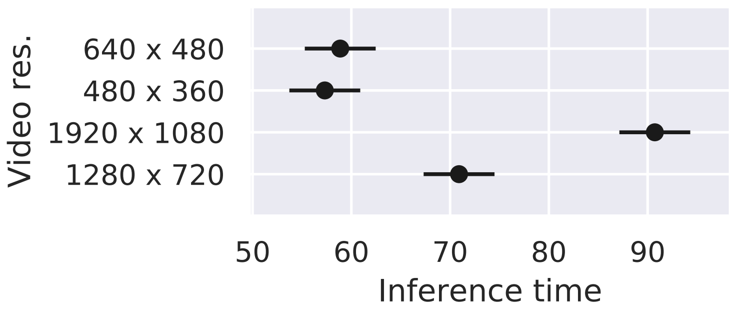
Another interesting observation is that MobileNetV3 is the only model whose latency lowered (in comparison with the other models) in the less powerful CPU, in comparison with the server CPU and GPU. This is a strong indication that the optimizations performed in MobileNetV3 [48] on top of MobileNetV2 and MnasNet, which initially targeted a performance gain in ARM-based mobile phones, are also visible in Intel-based x86 processors.

Lastly, since the MobileNets studied in this work receive images with a maximum input size of 224×224 pixels, the observation that using video feeds with smaller resolutions boost the inference time of all MobileNets is a clear indicator that low latency object detection applications should always prioritize using the lowest video resolution possible to avoid any overhead related to processing or resizing high definition images before each forward pass.

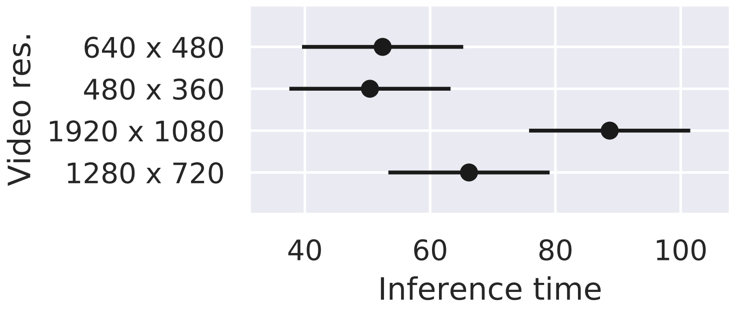
As for YOLO, its high latency in CPUs can be attributed to the optimization of its canonical implementation towards single GPU systems [5]. More specifically, the particularly high latency in low power CPUs achieved in this work for all YOLO models is in line with related work that also performed inference time measurements of the same models, albeit in different hardware [46].

6 Conclusions

This work presented a comparative analysis of the precision and inference time of reduced CNN architectures and single-stage object detectors in the task of single-class object detection under constrained hardware situations. Multiple MobileNetV2 and V3 models, as well as YOLOv3, YOLOv4 and their tiny counterparts were trained to detect soccer



(a) Inference time by input video resolution, MobileNetV2



(b) Inference time by input video resolution, i5

Figure 4: Tukey's HSD tests applied to the MobileNet groups who had $p \leq 5\%$ in the ANOVA tests on the Intel Core i5-4210U.

balls and tested in a computer with an Intel Core i5-4210U CPU and no graphics capabilities, following the use case of a humanoid robot operating with an embedded computer.

We used open implementations of all networks, provided either from the TensorFlow Object Detection API [48] or from the official YOLO repositories [5] and compared their average precision in a test data set, as well as their inference times in the constrained CPU setting, an unconstrained CPU setting and a server-class GPU, when operating on videos under multiple resolutions.

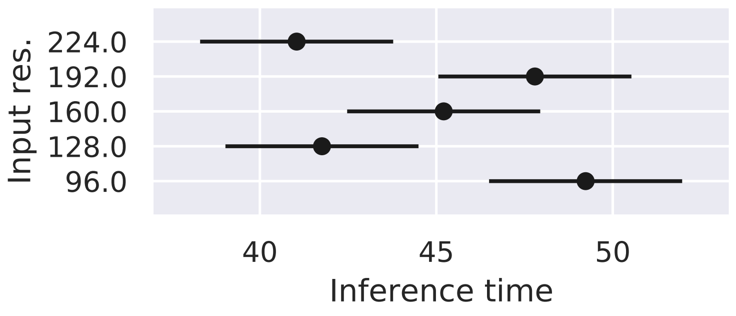
Results have shown that MobileNetV2 models with high width multipliers have the best trade-off between average precision and inference time in unconstrained hardware settings, being suitable when executing inference in remote servers is an option. However, while MobileNetV3 models did not show remarkable performance when operating in the unconstrained hardware settings, a performance boost was observed when inference was executed under a local, constrained, CPU-only scenario. Lastly, the official implementations of YOLO and TinyYOLO, being optimized for inference in GPUs, displayed poor results in our low-end Intel Core i5-4210U processor.

References

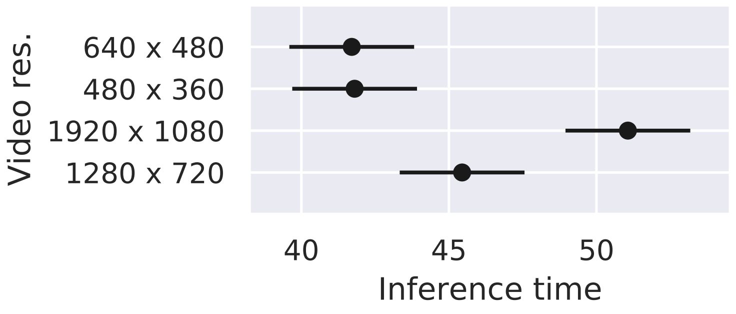
[1] Jonas Henrique Renolfi de Oliveira, Isaac Jesus da Silva, Thiago Pedro Donadon Homem, Douglas De Rizzo Meneghetti, Danilo Hernani Perico, and Reinaldo Augusto da Costa Bianchi. Object detection under constrained hardware scenarios: A comparative study of reduced convolutional network architectures. In *2019 XVI Latin American Robotics Symposium and VII Brazilian Robotics Symposium (LARS/SBR)*. IEEE, October 2019.

[2] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, June 2018.

[3] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun



(a) Inference time by width multiplier, MobileNetV2



(b) Inference time by input video resolution, MobileNetV2

Figure 5: Tukey's HSD tests applied to the MobileNet groups who had $p \leq 5\%$ in the ANOVA tests on The NVIDIA.

Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[4] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv:1804.02767 [cs]*, April 2018.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs, eess]*, April 2020.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In *Computer Vision – ECCV 2016*, volume 9905, pages 21–37. Springer International Publishing, 2016.

[7] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2019.

[8] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, abs/1506.01497, 2015.

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015.

[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861, 2017.

[11] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *European Conference on Computer Vision (ECCV)*, September 2018.

[12] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *arXiv:1807.11626 [cs]*, May 2019.

[13] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. In *33st AAAI Conference on Artificial Intelligence, AAAI 2019*, February 2019.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[15] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [cs, stat]*, September 2020.

[16] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. *arXiv:1911.09070 [cs, eess]*, July 2020.

[17] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456, June 2015.

[18] Laurent Sifre. *Rigid-Motion Scattering for Image Classification*. Ph. D. Thesis, Ecole Polytechnique, CMAP, Palaiseau , France, 2014.

[19] A. Krizhevsky. Convolutional deep belief networks on CIFAR-10. Technical report, 2010.

[20] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *arXiv:1612.08242 [cs]*, December 2016.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[22] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28:1135–1143, 2015.

[23] Q. Li, Q. Xiao, and Y. Liang. Enabling high performance deep learning networks on embedded systems. In *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 8405–8410, October 2017.

[24] C. Alippi, S. Disabato, and M. Roveri. Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 212–223, April 2018.

[25] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *arXiv:1609.07061 [cs]*, September 2016.

[26] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830 [cs]*, March 2016.

[27] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary Neural Networks: A Survey. *Pattern Recognition*, 105:107281, September 2020.

[28] B. Jian, C. Yu, and Y. Jinshou. Neural Networks with Limited Precision Weights and Its Application in Embedded Systems. In *2010 Second International Workshop on Education Technology and Computer Science*, volume 1, pages 86–91, March 2010.

[29] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang. Accelerating low bit-width convolutional neural networks with embedded FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, September 2017.

[30] S. Tripathi, G. Dane, B. Kang, V. Bhaskaran, and T. Nguyen. LCDet: Low-Complexity Fully-Convolutional Neural Networks for Object Detection in Embedded Systems. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 411–420, July 2017.

[31] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '06*, page 535, Philadelphia, PA, USA, 2006. ACM Press.

[32] Lei Jimmy Ba and Rich Caruana. Do Deep Nets Really Need to be Deep? *arXiv:1312.6184 [cs]*, October 2014.

[33] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [cs, stat]*, March 2015.

[34] Wolfgang Roth, Günther Schindler, Matthias Zöhrer, Lukas Pfeifenberger, Robert Peharz, Sebastian Tschiatschek, Holger Fröning, Franz Pernkopf, and Zoubin Ghahramani. Resource-Efficient Neural Networks for Embedded Systems. *arXiv:2001.03048 [cs, stat]*, January 2020.

[35] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, July 2017.

[36] V. Sze, Y. Chen, T. Yang, and J. S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*, 105(12):2295–2329, December 2017.

[37] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A Survey of Model Compression and Acceleration for Deep Neural Networks. *IEEE Signal Processing Magazine*, 35(1):126–136, June 2020.

[38] M. Niazi-Razavi, A. Savadi, and H. Noori. Toward real-time object detection on heterogeneous embedded systems. In *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 450–454, October 2019.

[39] Zuopeng Zhao, Zhongxin Zhang, Xinzheng Xu, Yi Xu, Hualin Yan, and Lan Zhang. A Lightweight Object Detection Network for Real-Time Detection of Driver Handheld Call on Embedded Devices. https://www.hindawi.com/journals/cin/2020/6616584/, December 2020.

[40] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang. Towards Real-Time Object Detection on Embedded Systems. *IEEE Transactions on Emerging Topics in Computing*, 6(3):417–431, July 2018.

[41] Uziel Jaramillo-Avila and Sean R. Anderson. Foveated Image Processing for Faster Object Detection and Recognition in Embedded Systems Using Deep Convolutional Neural Networks. In Uriel Martinez-Hernandez, Vasiliki Vouloutsi, Anna Mura, Michael Mangan, Minoru Asada, Tony J. Prescott, and Paul F.M.J. Verschure, editors, *Biomimetic and Biohybrid Systems*, Lecture Notes in Computer Science, pages 193–204, Cham, 2019. Springer International Publishing.

[42] A. Canziani, E. Culurciello, and A. Paszke. Evaluation of neural network architectures for embedded systems. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017.

[43] Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. Deploying Deep Neural Networks in the Embedded Space. *arXiv:1806.08616 [cs]*, June 2018.

[44] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang. Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):35–47, January 2018.

[45] M. Bettoni, G. Urgese, Y. Kobayashi, E. Macii, and A. Acquaviva. A Convolutional Neural Network Fully Implemented on FPGA for Embedded Platforms. In *2017 New Generation of CAS (NGCAS)*, pages 49–52, September 2017.

[46] Zicong Jiang, Liquan Zhao, Shuaiyang Li, and Yanfei Jia. Real-time object detection method based on improved YOLOv4-tiny. *arXiv:2011.04244 [cs]*, December 2020.

[47] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 740–755, Cham, 2014. Springer International Publishing.

[48] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017.

[49] Reinaldo Augusto da Costa Bianchi, Danilo Hernani Perico, Thiago Pedro Donadon Homem, Isaac Jesus da Silva, and Douglas De Rizzo Meneghetti. Open soccer ball dataset. IEEE Dataport, September 2020.

Acknowledgements

The authors acknowledge the São Paulo Research Foundation (FAPESP Grant 2019/07665-4) for supporting this project. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. This is a post-peer-review, pre-copyedit version of an article published in the Journal of Intelligent & Robotic Systems. The final authenticated version is available online at: https://doi.org/10.1007/s10846-021-01336-y.