Smartphone-based real-time object recognition architecture for portable and constrained systems

Martinez-Alpiste, Ignacio; Golcarenarenji, Gelayol; Wang, Qi; Alcaraz-Calero, Jose Maria

Published in:
Journal of Real-Time Image Processing

DOI:
10.1007/s11554-021-01164-1

E-pub ahead of print: 01/09/2021

Document Version
Publisher's PDF, also known as Version of record

Journal of Real-Time Image Processing
https://doi.org/10.1007/s11554-021-01164-1

ORIGINAL RESEARCH PAPER

Smartphone-based real-time object recognition architecture for portable and constrained systems

Ignacio Martinez-Alpiste[1] · Gelayol Golcarenarenji[1] · Qi Wang[1] · Jose Maria Alcaraz-Calero[1]

Received: 2 January 2021 / Accepted: 16 August 2021

Abstract
Machine learning algorithms based on convolutional neural networks (CNNs) have recently been explored in a myriad of object detection applications. Nonetheless, many devices with limited computation resources and strict power consumption constraints are not suitable to run such algorithms designed for high-performance computers. Hence, a novel smartphone-based architecture intended for portable and constrained systems is designed and implemented to run CNN-based object recognition in real time and with high efficiency. The system is designed and optimised by leveraging the integration of the best of its kind from the state-of-the-art machine learning platforms including OpenCV, TensorFlow Lite, and Qualcomm Snapdragon informed by empirical testing and evaluation of each candidate framework in a comparable scenario with a high demanding neural network. The final system has been prototyped combining the strengths from these frameworks and led to a new machine learning-based object recognition execution environment embedded in a smartphone with advantageous performance compared with the previous frameworks.

Keywords Machine learning · Object recognition · Deep Learning platforms · CNN · YOLOv3 · Embedded systems

1 Introduction

Artificial Intelligence (AI) has gained momentum in recent years in light of the huge potential in a wide range of applications, and there is an emerging trend to run machine learning in lightweight, embedded systems, such as smartphones for high mobility, low cost, rapid deployment and other benefits. Smartphone-based convolutional neural network (CNN) capabilities are an enabling technology for a variety of machine learning empowered novel use cases, where object recognition is required in outdoor areas, where there are other limiting factors such as the need of freedom of movement for the user, or the limited infrastructure and coverage available in the geographical area. The computational power of smartphones has drastically increased in the past few years and they are now comparable with desktop computers available some years ago. Nevertheless, running such CNN models on mobile devices is still challenging owing to the limited computing power and energy available [12]. Traditionally, CNN models run on high performance computing servers due to hardware requirements and are not available to operate on smartphones. To overcome these issues, there is a vital need for a machine learning framework suitable for smartphones to perform computing-intensive computer vision tasks, such as object recognition.

Due to an emerging interest in the Android operating system, some popular deep learning frameworks were ported to this operating system including TensorFlow Mobile (TFM), TensorFlow lite (TFL), OpenCV and Qualcomm Snapdragon. These platforms are destined to run the inference task on mobile phones, suitable for scenarios, where there is poor or no connectivity [34]. Each of these popular frameworks has its own strengths and weaknesses, which should be taken into account in selecting the best architecture for object recognition. However, this is an open question yet to be resolved in practical terms. Given the highly complicated and heterogeneous models in these frameworks, an empirical testing and evaluation based methodology would be the most practical approach. An object recognition system is generally composed of three steps, namely, image preprocessing, image recognition and image tracking. For each step, the selection of the best mobile platform based on evaluation is required to obtain the best system performance. A platform

✉ Ignacio Martinez-Alpiste
ignacio.alpiste@uws.ac.uk

[1] University of the West of Scotland, Paisley, Scotland, UK

Published online: 01 September 2021

is typically evaluated based on criteria including accuracy, speed, RAM usage and model size [21]. Image recognition is a computational expensive process able to mark with a bounding box the objective in an image; however, although object tracking is much less computational expensive, it needs the initial coordinates of the object to follow its movements in the next frames. Thus, the combination of image recognition and image tracking may create competitive system by reducing computation.

CNN-based object detectors are divided into two main categories: one-stage and two-stage detectors. Two-stage detectors are computationally expensive as first the regions of interest (ROI) are extracted and then the classification will take place. This method is not suitable for real-time object recognition on devices with power limitations. Meanwhile, one-stage detectors such as You Only Look Once (YOLO) [26] and Single Shot Detector (SSD) [17] achieve real-time object recognition by concurrently selecting and classifying the ROIs. These methods are faster at the cost of lower accuracy. YOLOv3 is faster and more accurate than other YOLO-based detectors and thus is explored in this study to, first, analyse the popular machine learning platforms and second to comprehensively test the proposed architecture. Hence, the main contributions of this paper are summarised as follows:

1. Practical deployment and testing of popular machine learning frameworks, comprehensive evaluation and analysis of popular mobile platforms based on criteria affecting the system performance, and contributing to an empirical methodology in selecting the most suitable technologies in machine learning based object recognition tasks.
2. Design and implementation of a novel object recognition architecture informed by the empirical analysis of the popular machine learning frameworks under study in a smartphone-based operational environment, leading to a highly optimised system of practical use.
3. New machine learning benchmark for object recognition systems in a smartphone platform, providing new insights into the performance of such systems.
4. Discussions regarding the limitations of the proposed architecture on running object recognition systems on portable devices.

The rest of the paper is structured as follows: Sect. 2 reviews popular object recognition algorithms and related work, and Sect. 3 further provides a detailed analysis and assessment of each of these individual machine learning frameworks. Section 4 experimentally tests and evaluates the previous analysis realised for each deployed platform. Furthermore, it presents the design and implementation of the proposed architecture. empirical results of the proposed architecture. Section 5 tests and compares the system. Finally, Sect. 6 concludes the paper.

2 Related work

2.1 Object recognition

As mentioned, object recognition algorithms based on CNN, can be classified into two main categories: two-stage and one-stage detectors. In two-stage detectors, such as Fast R-CNN [9], Faster R-CNN [28] , and R-FCN [8], the region proposal is generated in the first stage. In the second stage, the object classification and the regression of bounding-box (bbox) will be conducted. These methods are high in accuracy but slow in recognition. In one-stage detectors, such as SSD and YOLO, the object classification and bbox regression are performed simultaneously without a region proposal stage. These methods are fast in recognition but have low accuracy. YOLOv3 has a fast speed in multiple object recognition in a single inference. Furthermore, the low accuracy of YOLO and YOLOv2 [25, 26] is solved using a multi-stage recognition method. YOLOv3 is used when the system has enough computational resources. Tiny-YOLOv3 is for constrained environments and less accurate than YOLOv3 [2, 27].

2.2 Analysis of previous smartphone object recognition work

This subsection provides an overview of various use cases in the literature related to object recognition on Android mobile devices. There is a collection of existing work regarding popular machine learning (ML) platforms and machine-learning object recognition on mobile devices. Table 1 presents and compares the published results of using different CNNs on smartphones in relation to the study in this paper. The values represented in Table 1 are composed by accuracy, speed and model size. If a studied publication provides multiple results, these will be summarised between brackets. In [12], Ignatov et al. presented a review of the current state of deep learning and described the popular AI frameworks and the limitations of running AI including object recognition on smartphones that were considered for the implementation of our novel architecture. In [4], an architecture was presented for Ultra-low Power Binary-Weight CNN Acceleration. Although the work stated that by using binary-weights during training, a comparable classification could be achieved with non-binary weights, this can still reduce the accuracy. Moreover, an Android benchmark set was provided in [22] on OpenGL platform for low-power mobile GPUs. As stated by the authors, this android benchmark, however, is not ideal for GPU benchmarking and does

Table 1 Comparison of object recognition use cases on smartphones in literature

	Aspects						
Ref	Objective	Exec Environment	Algorithm	Platform	Accuracy	Speed (inference time)	Model size
[20]	Common Objects	Android	YOLOV3/ Tiny-YOLOv3	DJI+OpenCV	55.3/ 33.1%	[0.08/ 1.37] fps	248/ 35.4 MB
[21]	Common Objects	Android	SSD/ YOLOv3/ Tiny-YOLOv3	DJI + OpenCV/ DJI+Tensorflow	41.2/ 55.3/ 33.1%	[0.06, 1] fps	[23, 237] MB
[30]	Face	Android	Customed CNN	Google API	"small error"	*NG*	*NG*
[29]	Face	Smartphone	Alexnet& SVM	RenderScript	[88%, 96%]	[0.03, 0.17] fps	*NG*
[35]	Human	Smartphone	Googlenet/ Mobilenet + SSD	Keras/ Caffe	97.32%	*NG*	*NG*
[33]	Art Sculpture	Ipad	Alexnet Googlenet	Keras/ Caffe	[57.4, 59.3]%	[1.01, 3.4] fps	*NG*
[18]	Road damage	Smartphone	Inception V2/ Mobilenet + SSD	*NG*	Above 75%	0.66 fps	*NG*
[7]	Pedestrian	Tablet/Smartphone	AdaBoost	Qualcomm	*NG*	[8, 20] fps	*NG*
[13]	Food	Android	HOG&SVM	OpenCV/VLFEAT	79.2%	*NG*	*NG*
[14]	3D asset	Smartphone	SSD	Tensorflow lite	75%	*NG*	22 MB
[15]	Vehicle	Android	Squeezenet	Tensorflow	76.7%	*NG*	8 MB
[11]	Gesture	Android	3L MLP	Matlab	95%	*NG*	*NG*
[23]	Marker	Smartphone	LightDenseYOLO	Snapdragon OpenCV/	96%	20 fps	*NG*
TP	Common Objects	Android	Tiny-YOLOv3	Tensorflow Lite/ Snapdragon	33.1% COCO DS	17.7 fps	33.8 MB

NG not given, *TP* this paper, *italic* lack of information, *bold* our approach

not measure power efficiency. In addition, Alpiste et al. [20] presented benchmarking results regarding the performance of OpenCV as a popular computer vision and machine learning platform in two different studies using YOLOv3 and Tiny-YOLOv3 algorithms embedded on a smartphone. In these studies, however, the authors did not use other popular machine learning platforms including TFM, TFL, and Snapdragon to achieve close to real-time speed for aforementioned algorithms. As shown in Table 1, the studies in [7, 14, 18, 29, 30, 33, 35], and [15] used ML platforms, such as TFL, Google API, Snapdragon and Caffe and computer vision platforms such as OpenCV. Others deploy OpenCV, which combines machine learning with computer vision. These studies, however, did not provide all the necessary metrics including model size to make the comparisons [21]. In [23], a marker tracker was designed using a light-weight YOLO-based algorithm. Although the work achieved real-time speed (20 fps) using this algorithm, the images under study (markers) are easily detectable. Hence, the obtained accuracy (96%) is expected.

In summary, none of the above studies has covered a thorough analysis of the popular machine learning frameworks to provide a novel architecture for object recognition systems in constrained environments. For the first time, this paper provides a novel architecture for object recognition on constrained environments considering all the metrics that affect the system performance.

3 Analysis of deep learning mobile platforms

This section analyses the interesting different popular machine learning frameworks deployed on portable platforms with a focus on smartphones. The investigation includes four open-source machine learning libraries including OpenCV, TFM, TensorFlow Lite and Qualcomm Neural Processing Software Development Kit (SDK). Other platforms such as Keras were not considered, because it creates an abstraction layer over TensorFlow increasing the overhead and losing the control over complex architectures. For instance, Google API was not considered, because it is prepared with a set of ML models already included in TensorFlow, making Google API easy-to-use for non experts developers. Due to the focus of the study being on the object recognition process and this process being the most resource-intensive task, state-of-the-art mature and consolidated ML platforms were selected.

Figure 1 illustrates the comprehensive architecture of the machine learning libraries analysed and compared in this study. To test the proposed architecture shown in Fig. 1, we employ a state-of-the-art CNN YOLOv3. The standard version of YOLOv3 destined for non-constrained environments is utilised. Based on the architecture, the YOLOv3 model is in a format of a configuration file and a weight file [2]. These files are then integrated and converted to a format appropriate for the destined framework. To realise the conversion between different platform formats, Keras, which is a machine learning platform, is utilised. YOLOv3 is considered a high-demanding CNN in constrained environments. The architecture of YOLOv3 comprises 106 fully convolutional layers. This demands very high computational power for hardware. The main intention of testing this highly demanding neural network (YOLOv3) in our portable environment is to stress the system and evaluate the performance of the frameworks in high-demanding situations. If we are able to obtain acceptable results with the standard version, we will achieve greater results with lighter versions of CNN models in terms of inference time.

In the following subsections, the deployment of different platforms is thoroughly explained according to Fig. 1.

3.1 OpenCV

Open Source Computer Vision Library (OpenCV) is an open-source framework for image and video analysis [6]. Although this framework has been deployed to focus on image processing, it also provides functions for real-time machine learning. OpenCV has the ability to read the neural network configuration and the weight file directly without a need for a converter to load the CNN model into memory in an appropriate format. Meanwhile, OpenCV is limited in terms of compatibility with the processing units of smartphones. The platform is just able to run the neural network models using the Central Processing Unit (CPU), missing the opportunity to take advantage of high processing power of the Graphic Processing Unit (GPU) and the Digital Signal Processor (DSP).

As mentioned, due to being very mature and powerful, OpenCV is well implemented and adopted all around the world by developers. Its main focus is on real-time image processing, which makes it effective and fast for image preprocessing of the recognition process.

3.2 TensorFlow mobile

TensorFlow Mobile is a machine learning platform developed by Google. TFM aims to support different modern computing devices from smartphones [1] to embedded devices, such as Raspberry Pi and mobile applications [3]. TFM was derived from TensorFlow. Almost all the operations in the standard TensorFlow are supported by the TFM library. On account of its general implementation to be applicable in a wide variety of architectures, TFM has a drawback in performance when it comes to constrained environments. This limitation is not only related to computation power

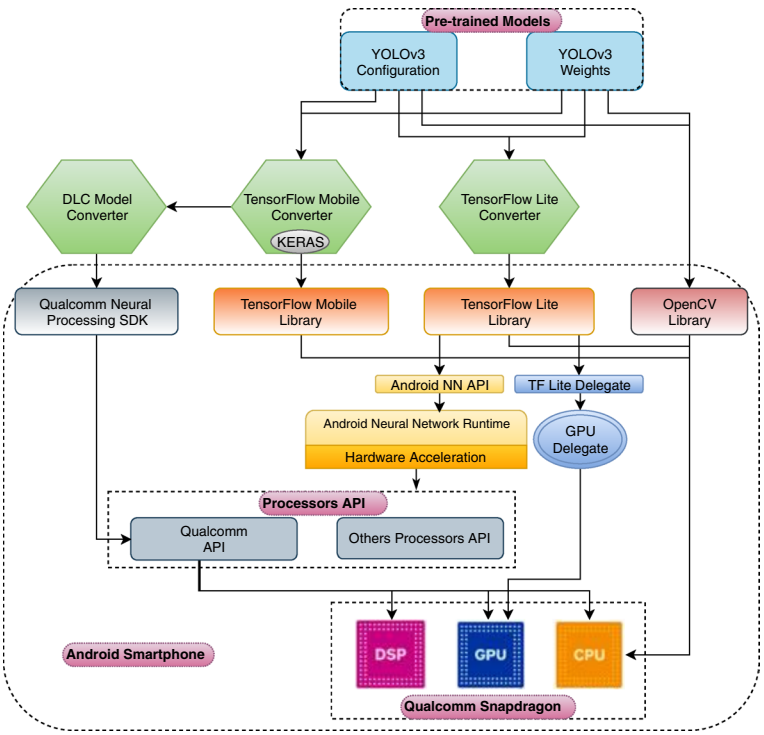

Fig. 1 State of the art technologies for object recognition and its deployment in smartphones

and memory but also in power consumption. This platform is now deprecated and substituted by TFL which mainly focuses on systems with low computational resources. Nevertheless, TFM is still in use in applications that have not been updated.

Unlike OpenCV, TFM needs a converter to transform the neural network configuration file and its weights into a protobuf file (.pb), which is fully compatible with this library. Using the protobuf file, TensorFlow can apply different optimisation techniques such as quantisation to further improve the speed of execution. Although this type of model optimisation may be beneficial for limited environments, it may compromise the accuracy. Similar to OpenCV, TFM merely runs the trained model on a CPU. This increases the portability between processors. This, however, causes the platform not to be easily deployable on constrained environments.

3.3 TensorFlow lite

TensorFlow Lite is the light version of TFM for constrained environments reducing latency and increasing efficiency. In addition, released by Google, TFL, is a machine learning framework for running TensorFlow models on smartphones, embedded systems and Internet of Things (IoT) devices with low latency and small battery size [32]. Comparable to TFM, TFL needs a converter to alter the neural network configuration file and its weights into a flatbuffer model (.tflite).

Meanwhile, TFL is more versatile than TFM, being able to execute machine learning algorithms on all on-board processors. Identical to TFM and OpenCV, TensorFlow Lite can execute the neural network on a CPU in a straightforward manner, although TFL gives faster inference time.

In addition, TFL has the ability to delegate part or all of the execution of an algorithm to mobile GPU. This delegation is through two key methods: GPU delegation [31] and android neural network API (NNAPI) [5]. In GPU delegation technique, TFL defines an abstraction layer for the programmer to communicate with the GPU. This delegation assists to obtain a better performance in terms of inference time and resource consumption. Although, being compatible with IOS smartphones and multitude of Android devices, several models and operations are not supported. This limits the developers' ability to execute different types of neural networks on this platform. For instance, YOLOv3 operations are not supported on this platform. However, operations for Inception and Mobilenet models are supported and used in TFL tutorials.

As already stated, another method of executing the CNN on GPU is through NNAPI. An interface has been implemented for TFL to communicate with the Android Neural Network API. NNAPI intends to use hardware accelerators for machine learning frameworks. To realise this, vendors must provide drivers for their own processors to utilise the API and make it compatible with on-device processors including CPU, GPU and DSP. NNAPI is then able to apply the neural network operations from TFL to mobile device hardware. Nonetheless, similar to the GPU Delegation technique, there are limitations when running YOLOv3 operations.

3.4 Qualcomm neural network processing SDK

The Snapdragon Neural Processing Engine SDK (SNPE) is a Qualcomm Snapdragon software accelerated at runtime to run deep neural networks [24]. Our system has integrated a Qualcomm Snapdragon processor to be compatible with SNPE. As Fig. 1 displays, SNPE needs a model formatted into a Deep Learning Container (DLC) file; therefore, a converter is employed. This converter supplied by Qualcomm demands a protobuf model as an input; thus, the system also needs the same converter as TFM.

This platform is fully compatible with its own drivers providing this ability to directly manipulate onboard processors (CPU, GPU and DSP). This will result in improved efficiency. There are two main drawbacks of deploying this SDK. First, unlike previous platforms, compatibility is decreased due to not all smartphones are equipped with Qualcomm Snapdragon. Second, the deployment of SNPE is more complicated because of low-level features the developers should deal with, forcing them to have a deeper understanding of the hardware.

4 The proposed system

This section describes the proposed system deployed on a smartphone for object recognition. First, a comprehensive evaluation was performed to obtain the best frameworks for our proposed object recognition system. Second, the system is established based on the results from Sect. 4.1, and a concurrent object recognition pipeline is defined.

4.1 Empirical platform evaluation criteria for system design

In this subsection, a complete evaluation is done to select the best frameworks for our system design. Three steps are involved in an object recognition system including image preprocessing, image recognition and image tracking. To achieve the best performance of the object recognition system, the best platform should be selected for each aforementioned step based on the metrics affecting the system performance. To this end, this section provides a step-by-step approach to choose the best platform for each step and obtain the most efficient and fast machine learning recognition

architecture after deploying and implementing each platform in the same environment.

4.2 Preprocessing

Preprocessing is a fundamental step when the recognition pipeline is running. It transforms raw frames (YUV) into a proper format for the input of the neural network. This step includes image scaling, image normalisation and colour space transformation. Image scaling is necessary to reduce the input size from 1280×720 to 416×416 to maintain a trade-off between speed and accuracy and achieve real-time detection for critical applications. Similar to the YOLO's authors, in this manuscript we have selected an input size of 416×416 to provide easier result comparison with the platforms of other researchers. Image normalisation is another stage to scale pixel values from 0 to 255 to a range of 0–1. Finally, a colour space transformation is needed in some cases. OpenCV loads the frame in BGR. Hence, a colour space transformation to RGB is needed. TFL and SNPE do not provide this step, thereby relying on Android libraries and lasting 56.3 ms. OpenCV, however, produces great results because of being powerful and mature in image processing with 9.3 ms [6].

4.3 Model size

The size of the models results in memory usage. Snapdragon and TFL approaches use serialisation to convert the configuration file and a weight file of the CNN into a readable format. This leads to a better result in terms of model size. In our study, the format of the flatbuffer model is suitable for TensorFlow Lite. Meanwhile, the double conversion from Protobuf to DLC in SNPE decreases the model size. Table 2 shows the size of the models for YOLOv3 for different formats related to each machine learning platform.

Table 2 Empirical factor comparison for each Machine Learning Platform

Factor	OpenCV	TF Lite	SNPE
Preprocessing	9.3 ms	56.3 ms	56.3 ms
Model size	237.08 MB	236.66 MB	212.41 MB
Loading time	456 ms	271 ms	2618 ms
Accuracy	Algorithm dependent (not platform)		
Speed (Inf. Time)	5203ms	4379 ms	595 ms
Average RAM	633 MB	263 MB	707 MB
Maximum RAM	1.5 GB	1 GB	1.1 GB
Battery	3280 mAh	3280 mAh	3280 mAh
Temperature	46 °C	46 °C	46 °C
Tracker	Sequential	Concurrent	Not Provided

4.4 Loading time

Smaller models do not necessarily mean the model will be loaded faster it into memory. The lowest model to be loaded is the DLC from Qualcomm Snapdragon. The next comes the OpenCV, which loads two files separately in 456 ms which is an acceptable result. Finally, the interpreter of the flatbuffer format is the fastest among all. Table 2 shows the average loading time of each model based on platform and format.

Nevertheless, the average loading time is not a determinant key factor. This is due to the fact that the models are just loaded once into memory at the beginning of the process.

4.5 Accuracy

Accuracy does not depend on the machine learning platform, where the neural network is executed. It depends intrinsically on the algorithm itself and the received training. Hence, the three platforms perform equally in terms of accuracy. The only way to improve the accuracy and modify the speed and the model size is through the model optimisation.

4.6 Speed

In constrained environments, speed is the key metric to be influenced. Inference time is the time taken from when a frame starts being processed until it obtains the results in terms of object recognition. Table 2 shows the inference time of recognising objects for each frame based on given platforms. OpenCV is the slowest one. Similarly, TFL obtained the same results. Snapdragon Library had the best result with almost two frames per second. In addition, Fig. 2 shows the cumulative average over 500 frames for each platform. As TFL and OpenCV are executed in CPU, the first iterations of the processes require more time due to the access to memory. Over time, these values achieve a controlled and stable execution. As observed, the inference time taken for TFL and OpenCV increases gradually. This means that

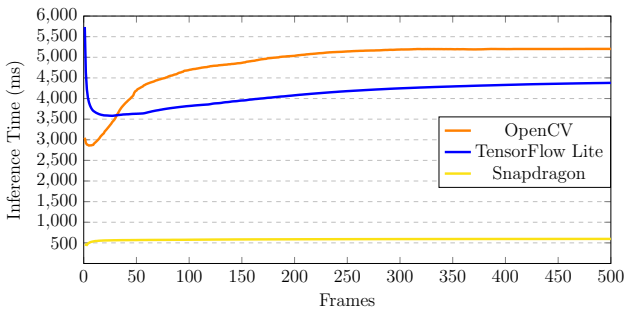

Fig. 2 Accumulative average of the inference time from the execution of YOLOv3 for each Machine Learning Platform

both platforms suffer a saturation when the frames are being processed by the neural network. In contrast, Snapdragon remains stable for each frame.

4.7 Average RAM usage

Highly compute-intensive tasks such as object recognition and image processing may lead to higher average RAM usage. This subsection evaluates the utilised RAM over an hour during the object recognition process. Table 2 shows that TFL, Snapdragon and OpenCV have the RAM usage of 236 MB, 707 MB and 633 MB, respectively. Thus, TFL performs the best.

4.8 Maximum RAM usage

At some points, RAM usage may hit a peak over the recognition process causing anomalies in the system. Table 2 also shows that TFL and Snapdragon perform almost equally and OpenCV uses 0.5 GB more memory when reaches maximum. These average and maximum RAM usage are not significant in a smartphone system when the smartphone has more than 4GB of memory.

4.9 Battery

In constrained environments, the battery may drain rapidly when executing compute-intensive algorithms. In our system, the battery consumption was very similar for all the concerned platforms due to the most resource-intensive task being the screen brightness while playing the videos, which is a compulsory process for all. In our system, 3280 mAh battery was drained for all the platforms after an hour of CNN execution.

4.10 Temperature

A compute-intensive process affects the temperature stabilisation, especially in constrained environments. The high temperature in a device can decrease the performance to a large extent in machine learning processes. In our use case, temperature increased until reached a plateau of 46–47°C for all the platforms.

4.11 Tracker

As defined in the speed subsection, real-time speed may not be achievable by deep CNN models such as YOLOv3 on constrained systems; therefore, a tracker as an external aid is needed to create a visual perception of real time for the final user. Among the machine learning platforms studied in this paper, OpenCV and TFL are the only platforms that provide trackers. As a result, Snapdragon was not evaluated in this section. The authors of [21] provided two reasons in favour of TFL when it comes to tracking. First of all, TensorFlow is faster than OpenCV when tracking an object. Second, TensorFlow has the ability to simultaneously execute the tracker to follow multiple recognised objects in parallel using different cores. These reasons make TFL the best choice when tracking multiple objects on the screen. For detection, as we implement the algorithm in Snapdragon, the decision of using more than one core relies on the API.

4.12 Design of the system

In this subsection, we propose a new system based on integration of the best of its kind component frameworks that have been empirically justified. The frameworks chosen to be integrated in our proposed system correspond to the results presented in Sect. 4.1. Our proposed system follows the three standard processes: image preprocessing, object recognition and object tracking. For the preprocessing step, OpenCV was selected as the most appropriate platform due to its strength in image processing. Regarding the recognition process, the Snapdragon framework was chosen due to its efficient support of GPU acceleration. Finally, the tracking process was carried out using the TFL platform for its speed and capability in tracking multiple objects simultaneously.

These three processes are executed in two concurrent threads, a recognition thread which performs the preprocesing and the image recognition process, and a second thread which leads the tracking process. Figure 3 depicts the workflow, since the video is decoded until the frames are shown on the screen with its correspondent recognition. The aforementioned workflow are explained in the following steps.

1. The proposed approach loads the object recognition model on the correspondent processor. As the chosen framework is SNPE, the model is loaded on the GPU to reduce the inference time.

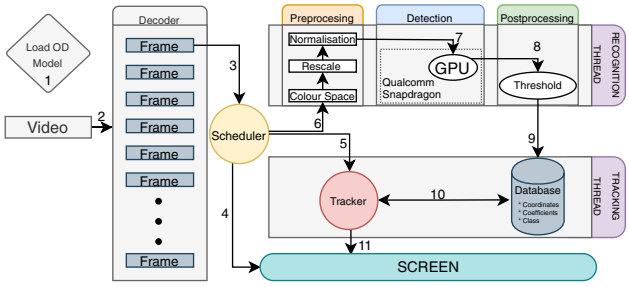

Fig. 3 Design of the proposed system. Recognition Thread includes preprocessing (OpenCV) and recognition (Snapdragon). Tracking Thread is led by TensorFlow platform

2. The video is decoded and each frame is stored in a first in first out (FIFO) queue for further processing.
3. The Scheduler is a process that manages the frames by assigning tasks to them. In this work, this process is mandatory to achieve real-time detection from the video feed. The frames extracted from the FIFO queue and delivered to to the recognition or tracking Threads for processing when any of them are idle. However, if both threads are currently occupied, the frame is marked as non-processed. This process will release every frame as soon as possible to achieve real-time processing. The Scheduler allows the neural network designer to abstract the design of the CNN from the implementation in the system by delivering the latest frame captured by the camera. If the CNN is computationally expensive, the Scheduler will adapt to a lower speed by providing the latest frame, and thus, the results are reliable in time. In contrast, if the CNN execution is faster than real time, the Scheduler will provide all available frames without discarding any.
4. The Scheduler displays all the frames that are sent on the screen to the final user.
5. The Tracker thread receives the frames from the Scheduler when idle.
6. The Recognition thread receives the frames from the Scheduler when idle. First, OpenCV preprocesses the frames. This action is divided into three main operations, namely, colour space transformation, compression and re-escalation of the frame, and image normalisation. At this point, the frame is ready for the recognition process.
7. SNPE executes the CNN and passes forward the preprocessed frame. This is the most time-consuming task of the object recognition pipeline.
8. The result is obtained by the execution of the CNN. The coordinates of the detected object, the class of each object and the confidence score are extracted from the results.
9. The previous results are stored in the Tracker database along with the frame.
10. Each frame received by the Tracker will consult the database to obtain the values for tracking. The tracker compares the current frame with the frame stored in the database, and the new location of the objects is updated in the database. If the Tracker loses an object, the entry of the database referring to that particular object will be deleted.
11. The Tracker updates the position of the recognised object on the screen.

This system workflow is implemented and tested in Sect. 5 for each ML platform and the proposed system.

5 Experimental results

5.1 Testbed

The deployed testbed was executed on a "Xiaomi Blackshark" smartphone with Qualcomm "Snapdragon 845" processor with 8 cores at 2.8 GHz and 8 GB of RAM memory. It has an integrated multilayer liquid-cooling to reduce the processor temperature. As "Snapdragon 845" is a popular processor choice, it was selected to study the proposed architecture. Table 3 summarises the specs of the smartphone and the version of the machine learning platform deployed. The machine learning algorithm deployed for testing is YOLOv3. Moreover, the results were obtained from the thread related to object recognition process which is specialised for recognition of small objects. This layer is more computationally expensive than other layers that detect bigger objects. YOLOv3 was trained with COCO data set with an obtained accuracy of 55.3 mAP. The COCO data set [16] is a very wide and complex data set for comparison. It comprises 80 object categories with more than 1.5 million object instances. It also has over 300k images in different environments and scenarios. In this section, we deploy standard YOLOv3 due to its computational expensive resources needed to be executed. At these tests, we want to stress the systems and demand the maximum performance of each approach to be easily comparable.

Table 3 Smartphone specs and machine learning platforms version

Smartphone specs	
Processor	Snapdragon 845
Instruction	ARMv8-A
CPU	4× 2.8 GHz Kryo 385, 4 × 1.8 GHz Kryo
CPU Frec.	2800 MHz
GPU	Qualcomm Adreno 630, 710 MHz
GPU Frec.	710 MHz
L1 Cache	32 kB + 32 kB
L2 Cache	1536 kB
L3 Cache	2048 kB
RAM	8 GB, 1866 MHz
Storage	256 GB
Android	Version 9
Machine learning platforms version	
OpenCV	4.0.0
TensorFlow	2.0.0
SNPE	1.26.0

5.2 Approach evaluation

To evaluate the performance of the proposed system, we carried out extensive experiments. The proposed system was compared with three other alternative approaches: Snapdragon, TFL and OpenCV. These three alternative approaches adopted their own built-in processes for preprocessing, recognition and tracking. The only exception process is tracking in Snapdragon. As Snapdragon has not provided any tracking algorithms, the TensorFlow tracker was adopted in the Snapdragon pipeline to allow the comparison across the pipelines under study.

In the experiments, videos were recorded at 24 fps, and then fed into the proposed system. The videos were recorded at 24 fps, since it is a standard in every smartphone camera and it allows a good balance for viewing experience and real-time processing in our platform. An experiment was conducted to count how many frames were processed by the recognition process and the tracking process. It is noted that the preprocessing process is included as a part of the recognition process due to the fact that preprocessing will always be executed just before the recognition process. The frames that are not handled are called "dropped" frames. These discarded frames correspond to the frames that are not tracked or recognised. In addition, the recognised frames were also processed by the Tracker, and thus the sum of tracked frames and dropped frames must be 24 in each second of time.

Figures 4, 5 and 6 show the number of frames managed for 30 s. In total, 720 frames have been considered. Figure 4 exhibits the number of frames recognised per second. As it can be observed, TFL and OpenCV are just able to detect one frame every 3–4 s. In contrast, Snapdragon and our approach achieve between 1 and 2 frames per second. Nevertheless, our approach detects more frames per seconds, because it has a lower preprocessing rate.

Figure 5 illustrates the experimental results regarding the tracked frames. As explained in Sect. 3, the best tracking process was realised by TensorFlow, and due to all the approaches with the exception of OpenCV executes this Tracker, they obtain good results. Nonetheless, there is an overload when executing TFL and OpenCV systems leading to a decrease in the performance in both platforms. The same trend to decrease in the performance is depicted in Fig. 6, which exposes the dropped frames. Over each second, TFL and OpenCV increase the number of frames dropped over the time spent.

Although the previous figures show the number of frames handled for 30 s, Fig. 7 illustrates the final number of frames tested by the different systems for recognition, tracking and being dropped. Whereas the recognition and tracking processes should take as many frames as possible, the dropped frames should be kept as minimal as possible. In conclusion, the worst-performing object recognition system was OpenCV, which discarded more frames than tracked in contrast to our approach, which was the best-performing object recognition system in this group.

Figure 8 depicts a stacked bar graph of the four different object recognition pipelines (preprocessing, recognition and tracking). Each bar in the chart represents one object recognition pipeline.

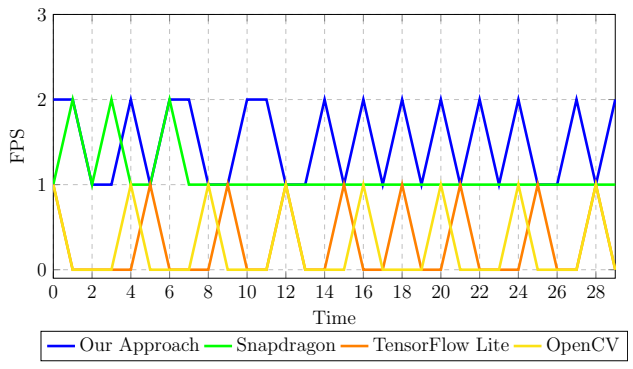

Fig. 4 Comparative of frames recognised per second in the different Machine Learning Platforms

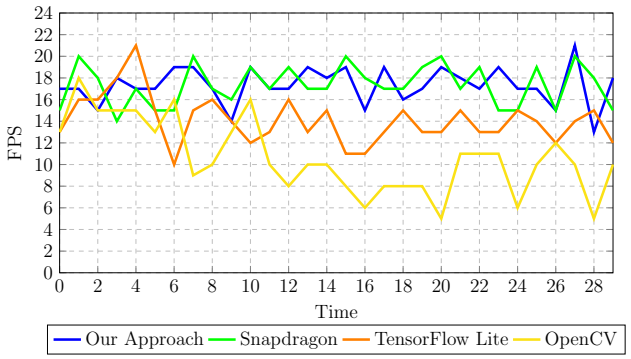

Fig. 5 Comparative of frames tracked per second in the different Machine Learning Platforms

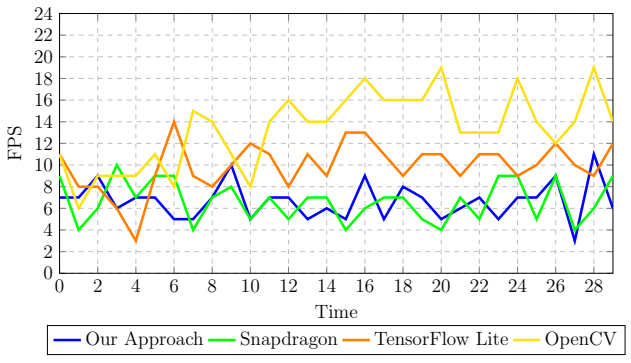

Fig. 6 Comparative of frames dropped per second in the different Machine Learning Platforms

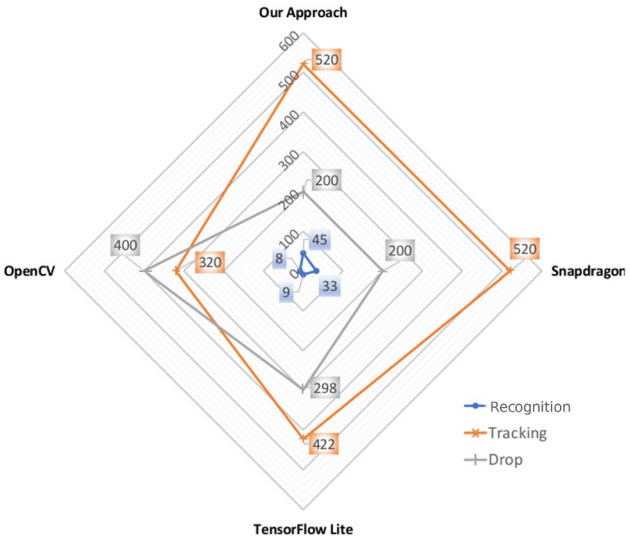

Fig. 7 Representation of the frames recognised, tracked and dropped for each machine learning platforms for 30 s

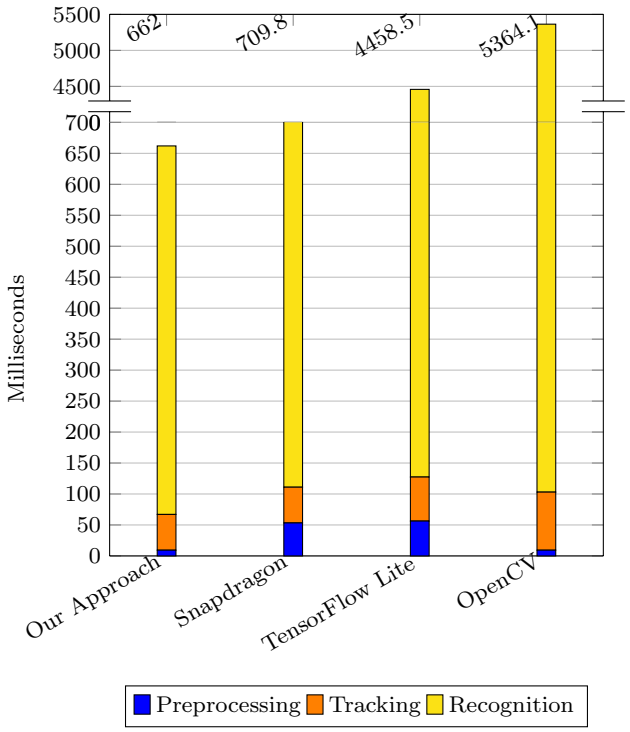

Fig. 8 Comparison of each system with the different three processes stacked

The results obtained by OpenCV and TensorFlow pipelines demonstrated that there is still much room for enhancement to achieve smoother recognition even on a latest-generation smartphone. Nevertheless, the limitations of these frameworks are exposed when working on Snapdragon GPUs with a process time of lower than one frame per second. Our integrated approach takes the best of each state of the art platform, thereby being 88% faster than Open- CV, 86% faster than TFL and 7% faster than Snapdragon.

Although the accuracy does not depend on the ML platform, we present a screenshot (Fig. 9) taken from the proposed system. This image was taken at the University of the West of Scotland premises and after preprocessing in the system, our system was able to recognise the objects including most of the cars parked and one person walking at high accuracy.

In terms of parameters that affect the performance of constrained environments, the battery consumption is very similar to the state-of-the-art ML platforms. While our approach has a battery drainage of 680 mAh, the others have 720 mAh. In terms of temperature, our approach achieves a plateau of 43 degrees, three degrees less than standard ML platforms.

5.3 Evaluation of the approach in a real scenario

As stated previously, we have deployed and tested standard YOLOv3 in the scenario to stress the system. In this subsection, we will focus on executing our approach with an algorithm designed for constrained environments, such as Tiny-YOLOv3. This model was trained with the popular COCO data set. While computationally expensive algorithms (Faster R-CNN, YOLOv3, SSD) achieve more than 45% of accuracy, Tiny-YOLOv3 obtained an accuracy of 33.1 mAP with a model size of only 33.8 MB which is more suitable for constrained environments. Although the accuracy is lower in comparison to computationally expensive algorithms, this model is able to detect big/medium size objects with good accuracy and can still be used for use cases in this regard in constrained environments which is not possible when using computationally expensive algorithms; It, however, loses accuracy for small size objects.

Fig. 9 Screenshot from the smartphone while executing the proposed approach. It recognised the cars parked and one person walking

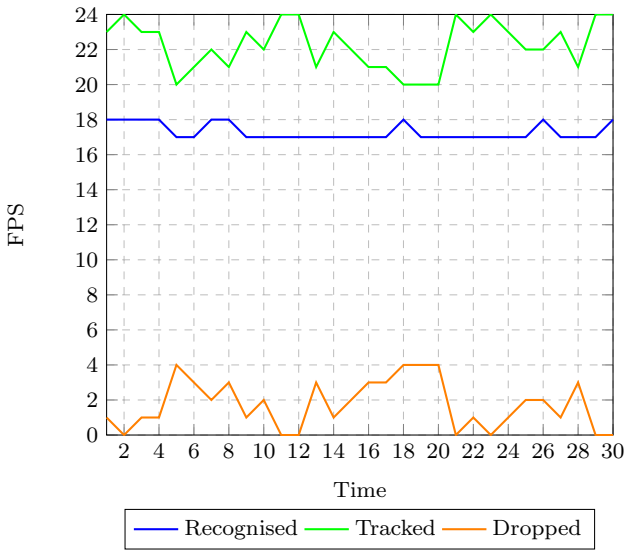

Fig. 10 Frames processed by Tiny-YOLOv3 over 30 s at 24 fps

Figure 10 shows the number of frames per second (out of 24) treated by each process. As apparent in the figure, the Tracker performs at an average of 22.2 fps, leading to an average of 1.8 dropped fps. These results provide a perception of real time for the final user. Regarding the recognition, Tiny-YOLOv3 achieves 17.3 fps. The recognition time starts when the frame is read from the queue in the recognition thread and stops when the results are shown on the screen including preprocessing, CNN execution and postprocessing stages. Nevertheless, if we measure the inference time of the CNN model running on the system, we obtain an average of 47.5 ms per frame (i.e., 21 fps). Our approach with Tiny-YOLO just achieves a battery drainage of 520 mAh and reaches a plateau of 38 degrees, which is 8 degrees lower than standard ML platform approaches. In the following subsection (5.4), different bottlenecks are discussed to further improve the system.

5.4 Discussion

This subsection further analyses the time breakdown of different stages and the bottleneck in the system focusing on our approach. As shown in Fig. 8, each stage (preprocessing, recognition and tracking) can be evaluated separately in the whole object recognition pipeline. For a real scenario as described in Sect. 5.3, the most time-consuming stage is the recognition process with 47.5ms per frame, representing the 83.92% of the whole object recognition pipeline. For preprocesing, OpenCV took 9.1 ms (15.26% of the whole system) being the fastest in the whole object recognition system. Finally, the least time-consuming stage is the tracking. The tracking process merely took an average time of 21ms per frame, although we did not discuss this process as the implementation of the Tracker did not cause any delay in the proposed solution due to thread parallelism. Any overhead related to internal processes of the application or the operating system was not considered to calculate the duration of each stage. Since the videos were recorded at 24 fps, when it comes to an ideal system, it took 41.6ms for each frame to be processed before being discarded by the new frame. Although our approach is able to preprocess in real time without losing any frames, the bottleneck lies in the recognition process. The recognition pipeline lasted 56.6ms; hence, our approach was able to process 17.7 frames out of 24 frames.

Figure 11 depicts a comparison between our approach and an ideal framework that is able to achieve real-time recognition for each frame and thus does not require any tracking stage. The ideal framework presented in Fig. 11 was obtained by extrapolating the same percentages of our system executing Tiny-YOLOv3 on the ideal system. To achieve this ideal system, it is expected that the time for the recognition process to be reduced in future GPU hardware developments. Nevertheless, 9.1 ms spent by preprocessing just leaves 32.5 ms for the CNN to recognise objects. This means that not only the time for recognition process should be reduced, but also the preprocessing should be decreased to allow the system to have more time for object recognition. Based on this manuscript, our approach is able to reduce the inference time by 99.1% in comparison to the basic implementation. However, there is still an extra 26% to reduce to achieve 24 fps in a constrained environment. Several approaches could be taken to further reduce the inference time [19], such as quantisation techniques or faster feature

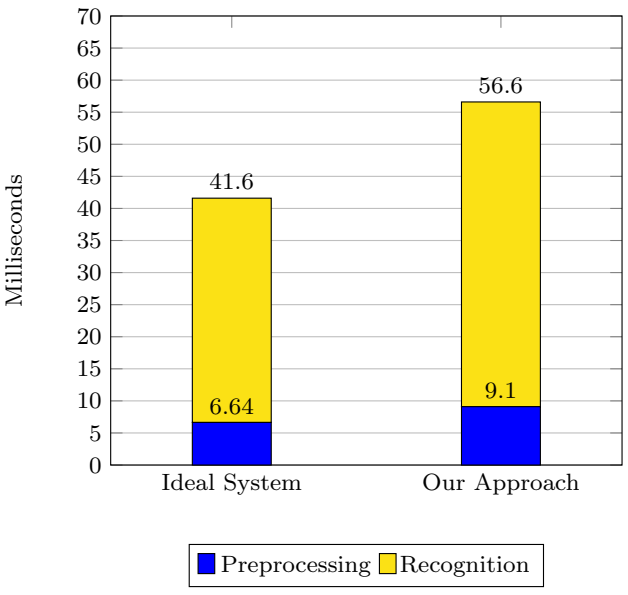

Fig. 11 Comparison of our approach with a ideal model that achieves real-time recognition per frame

extraction methods [10]. Nevertheless, they may reduce the accuracy. Other techniques such as management of multiple GPU cores may lead to a reduction in the inference time and it may not affect the accuracy. However, this implementation is time-consuming as it is not currently allowed by the machine learning platforms.

6 Concluding remarks

A novel architecture has been proposed for machine learning empowered object recognition in smartphone platforms, which are typically resource constrained. The proposed system takes advantage of the best of its kind in the three building blocks from a group of popular machine learning frameworks, based on an empirical evaluation. Consequently, the system outperformed the alternative approaches significantly. The methodology of object recognition system was deployed with high accuracy and efficiency. The experiment results have showed close-to-real-time performance of 17.7 fps speed at 33.1 mAP of accuracy and with a 33.8 MB model size. The present study revealed that the proposed system has the potential to be further improved towards per-frame real-time object recognition on mobile devices. Such a capability is expected to contribute to the wide use of real-time object recognition applications that require a highly mobile platform.

Acknowledgements This work was in part funded by the EU Horizon 2020 5G-PPP 5G-INDUCE project ("Open cooperative 5G experimentation platforms for the industrial sector NetApps") with Grant number H2020-ICT-2020-2/101016941. The authors would like to thank all the partners in this project for their support.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467 (2016)
2. Alexey, A.B.: Darknet. https://github.com/AlexeyAB. Accessed 18 Feb 2019
3. Alsing, O.: Mobile object detection using TensorFlow Lite and transfer learning. Degree Project Comput. Sci. Eng. (2018)
4. Andri, R., Cavigelli, L., Rossi, D., Benini, L., Yoda, N.N.: An architecture for ultralow power binary-weight CNN acceleration. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(1), 48–60 (2018). https://doi.org/10.1109/TCAD.2017.2682138
5. Android: the android neural networks API: https://developer.android.com/ndk/guides/neuralnetworks. Accessed 20 June 2020
6. Bradski, G.: The OpenCV library. Dr. Dobb's Journal of Software Tools (2000)
7. Costea, A.D., Vesa, A.V., Nedevschi, S.: Fast pedestrian detection for mobile devices. In: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, pp. 2364–2369 (2015). https://doi.org/10.1109/ITSC.2015.382
8. Dai, J., Li, Y., He, K., Sun, J.: R-fcn: object detection via region-based fully convolutional networks. In: Advances in neural information processing systems, pp. 379–387 (2016)
9. Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision 2015 International Conference on Computer Vision, ICCV 2015, pp. 1440–1448 (2015). https://doi.org/10.1109/ICCV.2015.169
10. Golcarenarenji, G., Martinez-Alpiste, I., Wang, Q., Alcaraz-Calero, J.M.: Efficient real-time human detection using unmanned aerial vehicles optical imagery. Int. J. Remote Sens. **42**(7), 2440–2462 (2021). https://doi.org/10.1080/01431161.2020.1862435
11. Idris, M.I., Zabidi, A., Yassin, I.M., Ali, M.S.A.M.: Human posture recognition using android smartphone and artificial neural network. In: 2015 IEEE 6th Control and System Graduate Research Colloquium (ICSGRC), pp. 120–124 (2015). https://doi.org/10.1109/ICSGRC.2015.7412477
12. Ignatov, A., Timofte, R., Chou, W., Wang, K., Wu, M., Hartley, T., Van Gool, L.: AI benchmark: running deep neural networks on android smartphones. In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11133 LNCS, pp. 288–314. Springer (2019). https://doi.org/10.1007/978-3-030-11021-5_19
13. Kawano, Y., Yanai, K.: FoodCam: a real-time food recognition system on a smartphone. Multim. Tools Appl. **74**(14), 5263–5287 (2015). https://doi.org/10.1007/s11042-014-2000-8
14. Kostoeva, R., Upadhyay, R., Sapar, Y., Zakhor, A.: Indoor 3D interactive asset detection using a smartphone. Remote Sens. Spat. Inf. Sci. ISPRS Arch. Photogramm. **42**(2/W13), 811–817 (2019). https://doi.org/10.5194/isprs-archives-XLII-2-W13-811-2019
15. Li, Z., Rao, Z.: Object detection and its implementation on android devices, pp. 1–8 (2014)
16. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: common objects in context. In: European Conference on Computer Vision, pp. 740–755. Springer (2014)
17. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9905 LNCS, pp. 21–37 (2016). https://doi.org/10.1007/978-3-319-46448-0_2
18. Maeda, H., Sekimoto, Y., Seto, T., Kashiyama, T., Omata, H.: Road Damage Detection Using Deep Neural Networks with Images Captured Through a Smartphone, pp. 4–6 (2018). arXiv:1801.09454. http://arxiv.org/abs/1801.09454
19. Martinez-Alpiste, I., Golcarenarenji, G., Wang, Q., et al.: A dynamic discarding technique to increase speed and preserve

accuracy for YOLOv3. Neural Comput. Appl. **33**, 9961–9973 (2021). https://doi.org/10.1007/s00521-021-05764-7
20. Martinez-Alpiste, I., Casaseca-de-la Higuera, P., Alcaraz-Calero, J., Grecos, C., Wang, Q.: Benchmarking machine-learning-based object detection on a uav and mobile platform. In: 2019 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, pp. 1–6 (2019)
21. Martinez-Alpiste, I., Casaseca-de-la Higuera, P., Alcaraz-Calero, J.M., Grecos, C., Wang, Q.: Smartphone-based object recognition with embedded machine learning intelligence for unmanned aerial vehicles. J. Field Robot. **37**(3), 404–420 (2020)
22. Nah, J.H., Suh, Y., Lim, Y.: L-Bench: an android benchmark set for low-power mobile GPUs. Comput. Graphics **61**, 40–49 (2016). https://doi.org/10.1016/j.cag.2016.09.002
23. Nguyen, P.H., Arsalan, M., Koo, J.H., Naqvi, R.A., Truong, N.Q., Park, K.R.: LightdenseYOLO: a fast and accurate marker tracker for autonomous UAV landing by visible light camera sensor on drone. Sensors (Switzerland) **18**(6), 1–30 (2018). https://doi.org/10.3390/s18061703
24. Qualcomm: Snapdragon neural processing engine SDK (2019). URL https://developer.qualcomm.com/docs/snpe/overview.html. Accessed 20 June 2020
25. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2016-Dec, pp. 779–788 (2016). https://doi.org/10.1109/CVPR.2016.91
26. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. In: Proceedings—30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 2017-Jan, pp. 6517–6525 (2017). https://doi.org/10.1109/CVPR.2017.690
27. Redmon, J., Farhadi, A.: YOLOv3: An Incremental Improvement (2018). http://arxiv.org/abs/1804.02767
28. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1137–1149 (2017). https://doi.org/10.1109/TPAMI.2016.2577031
29. Sarkar, S., Patel, V.M., Chellappa, R.: Deep feature-based face detection on mobile devices. In: ISBA 2016—IEEE International Conference on Identity, Security and Behavior Analysis (2016). https://doi.org/10.1109/ISBA.2016.7477230
30. Stoimenov, S., Tsenov, G.T., Mladenov, V.M.: Face recognition system in android using neural networks. In: 2016 13th Symposium on Neural Networks and Applications, NEUREL 2016, pp. 1–4 (2016). https://doi.org/10.1109/NEUREL.2016.7800138
31. TensorFlow: GPU delegation. https://www.tensorflow.org/lite/performance/gpu. Accessed 1 Feb 2020
32. TensorFlow: Tensorflow lite (2019). https://www.tensorflow.org/lite/guide. Accessed 1 Feb 2020
33. Tobias, L., Ducournau, A., Rousseau, F., Mercier, G., Fablet, R.: Convolutional neural networks for object recognition on mobile devices: a case study. In: Proceedings—International Conference on Pattern Recognition, pp. 3530–3535 (2017). https://doi.org/10.1109/ICPR.2016.7900181
34. Xu, M., Liu, J., Liu, Y., Lin, F.X., Liu, Y., Liu, X.: A first look at deep learning apps on smartphones. In: The World Wide Web Conference on—WWW '19 (May), 2125–2136 (2019). https://doi.org/10.1145/3308558.3313591. http://dl.acm.org/citation.cfm?doid=3308558.3313591
35. Yong, S.P., Yeong, Y.C.: Human object detection in forest with deep learning based on Drone's vision. In: 2018 4th International Conference on Computer and Information Sciences: Revolutionising Digital Landscape for Sustainable Smart Society, ICCOINS 2018—Proceedings, pp. 1–5 (2018). https://doi.org/10.1109/ICCOINS.2018.8510564

Ignacio Martinez-Alpiste is a Postdoctoral Researcher in Artificial Intelligence at the University of the West of Scotland, UK. He is a winner of the UK Times Higher Education (THE) Awards 2020 – Knowledge Exchange and the 2020 Scotland CeeD Industry Awards – Innovation Award. He is Co-Investigator for the CENSIS S-RPAS, SmartCranes, NG-RPAS and for EU Horizon 2020 funded projects (SELFNET, 5G-INDUCE and ARCADIAN-IoT). His main research interests include AI with applications in Unmanned Aerial Systems, 5G networks and video networking, among others. He is also a reviewer of international scientific journals and conferences. He received his PhD at the UWS.

Gelayol Golcarenarenji is a Postdoctoral Researcher in artificial intelligence at the University of the West of Scotland (UWS), UK, where she is working on the 5G-INDUCE European project as well as the NG-RPAS project. Her current research interests include artificial intelligence, machine learning, deep learning methods and applications. Previously, she was involved in "Smart crane Real-Time Object Detection" and "Smart Remotely Piloted Aircraft Systems (S-RPAS) for Real-Time detection" of missing people and she and her team won the CEED-SCOTLAND innovation award, and UK Times Higher Education (THE) Awards 2021 knowledge exchange award for the project. She received her PhD degree from Deakin University, Australia, where she was involved in the automation of carbon fiber production line using image processing and industrial machine learning techniques. Her research to date has resulted in more than 20 peer-reviewed journal papers, conference papers, and book chapters.

Qi Wang is a Professor in Next-Generation Smart Networks and Services at UWS. He has served as a Board Member of the EU 5G-PPP Technology Board, and Member of several 5G-PPP Working Groups, Scotland's Developing AI and AI-Enabled Products and Services Working Group, and ITU-T Focus Group on Autonomous Networks. He was the Co-Technical Manager for EU Horizon 2020 5G projects SELFNET and SliceNet, and Principal Investigator (PI) or Co-PI for projects funded by EU Horizon 2020 (5G INDUCE, 6G BRAINS, ARCADIAN-IoT), UK EPSRC, UK CENSIS and so on. He has published over 170 papers and is a winner of several Best Paper Awards. He is a winner of the UK Times Higher Education (THE) Awards 2020 - Knowledge Exchange or Transfer Initiative of the Year Award, and the 2020 Scotland CeeD Industry Awards - Innovation Award, among others.

Jose Maria Alcaraz-Calero is a Professor in Networks at the School of Engineering and Computing at the UWS. He is an IEEE Senior Member with experience in 5G networks, network slicing, monitoring, automation and management. In the academic side, he has more than 150 publications SCI-indexed international journals, conferences and books. He has been involved in 20 editorial activities in the most prestigious journal in the field and served as chair in 20 international flagship conferences and contributes as a Technical Programme Committee member in more than 100 international conferences. In the industrial side, he has more than 50 patents and intellectual property rights. From the leadership perspective, Jose M. has significant experience as Principal Investigator or as Co-Investigator in more than 20 research projects at local, national and especially at European and international level. He is the Co-Technical Manager for EU Horizon 2020 projects SELFNET and SliceNet.