2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications (DS-RT) | 978-1-6654-3326-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/DS-RT52167.2021.9576139

A study on real-time image processing applications with edge computing support for mobile devices

Gabriele Proietti Mattia, Roberto Beraldi
Department of Computer, Control and Management Engineering "Antonio Ruberti",
Sapienza University of Rome,
Email: proiettimattia@diag.uniroma1.it, beraldi@diag.uniroma1.it

***Abstract*—Different libraries allow performing computer vision tasks, e.g., object recognition, in almost every mobile device that has a computing capability. In modern smartphones, such tasks are compute-intensive, energy hungry computation running on the GPU or the particular Machine Learning (ML) processor embedded in the device. Task offloading is a strategy adopted to move compute-intensive tasks and hence their energy consumption to external computers, in the edge network or in the cloud. In this paper, we report an experimental study that measure under different mobile computer vision set-ups the energy reduction when the inference of an image processing is moved to an edge node, and the capability to still meet real-time requirements.**

In particular, our experiments show that offloading the task – in our case real-time object recognition – to a possible next-to-the-user node allows saving about the 70% of battery consumption while maintaining the same frame rate (fps) that local processing can achieve.

I. Introduction

Image-processing based applications for smartphones and mobile devices are growing at an extraordinary rate due to the level of maturity achieved by most support technologies (e.g. computer vision, hardware, AI). Application domains range from immersive multi-gaming, online shopping, virtual browsing, remote assistance, etc. Experts predict that the Augmented/Virtual Reality (AR/VR) industry will reach over $25 billion by 2025 and that growth will continue steadily.

Cloud VR/AR services are currently offered by the main cloud providers, but they will hardly meet both the above conditions. Edge/Fog computation capability will likely complement and improve these services, however, running these tasks directly in mobile devices is progressively possible with inference latencies, for example in the case of Convolutional Neural Networks (CNN), that are comparable to the ones that before was only expected on high-end GPUs [1] – obviously with less but still acceptable accuracy. These results can be achieved, not only by using the CPU and GPU resources of the device but also by directly implementing particular chips, also called coprocessors [2], that are designed for executing the most recurring machine learning operations. For example, Google, starting with the Pixel 4 smartphone, implemented the Pixel Neural Core, a dedicated core for efficiently performing image-related machine learning (ML) tasks. Indeed, the most common applications for having Deep Neural Networks (DNN) directly running in the device regard [3]: photo improvement [4], activity recognition and tracking [5], [6], image classification [7], face recognition [8], real-time object recognition especially for supporting AR (Augmented Reality) applications [9] (like for example the landmark recognition). All of these applications can now easily run on smartphones, even with very low processing latency but this strategy has a clear downside. Indeed, running a DNN requires a non-negligible number of operations (that are often referred as FLOPs – floating-point operations), as a consequence, we expect a significant impact on the power consumption, that is a critical aspect for mobile applications. The approach for reducing this problem is to compress the DNN model [10], [11] by pruning the network, decomposing layers or use quantized weights for the model. Obviously, by losing information and model complexity, we lose something in the accuracy of the neural network, but in some cases, this loss is so minimal that the performances of the network are still acceptable.

The second approach to solve the problem is the offloading [12], namely making the device to execute the inference on an external server and then locally parsing the results. Task offloading consists of delegating part of the image processing to a remote server. The convenience of this technique presupposes three conditions: (1) the energy cost of offloading operations at the device is less than the energy cost required if the delegated calculation is performed on-board; (2) the response time and more generally the processing latency should guarantee at least the frame rate measured when processing is all local, and does not produce a lag in the rendering of the video; (3) the image processing precision metrics, e.g., accuracy, does not deteriorate.

For example, if we use a 'classic' cloud provider we can obtain very low inference latencies, but the network latency can reach values of 100ms or more, and therefore this cannot be suitable for performing VR applications. In addition, some AR applications require shared virtual objects (also called "cloud anchors") that must be updated in real-time by multiple users, which make the adoption of a low latency and a high throughput cloud service mandatory. In these application scenarios, edge/fog computing can provide a solution[1].

In this work we report the results of the measurements of

978-1-6654-3326-6/21/$31.00 ©2021 IEEE

[1]Google is experimenting/suggesting solutions in this sense for its AR core SDK

some experiments conducted on real mobile devices and using state-of-the-art and open-source video and image processing libraries, with the aim of concretely verifying what are the energy consumption and processing capabilities measured in frames per second. The useful use of this study lies in acquiring concrete data that can be used as criteria of choice in the design of offloading algorithms.

In this work, we investigated the energy compromise that a deep learning task for object recognition imposes to a mobile device and how much we can gain in terms of power consumption when the task is completely offloaded to a backend server equipped with a good GPU as well as the achievable frame rate.

For doing this we set up an environment by using the most common libraries that allow doing inference with CNNs, like TensorFlow[2], OpenCV[3] and Darknet[4], and a set of pre-trained networks. From this, by using a sample video, we run the object recognition frame by frame, testing different set-ups, libraries and neural networks by using a custom Android application (Figure 1) and a Python Flask backend. We have conducted several benchmarks for deeply understanding the energy impact of a neural network deployed in a mobile system but also for evaluating how the offloading can have a beneficial effect on power consumption though preserving inference latencies. As the main source for power consumption data, we used the values offered by the Android OS that have been cross-checked by using a USB power meter.

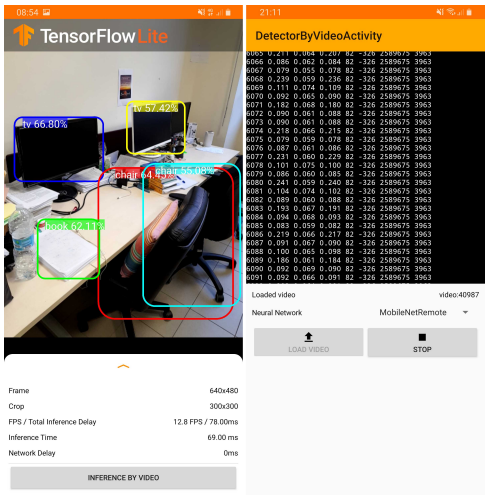

Fig. 1: TensorFlow Lite Application for mobile object recognition.

The rest of this paper is organized as follows. In Section II, we give some background related to deep learning and object recognition. In Section III, different experimental set-ups are introduced with all the details about libraries used and the hardware. Section IV presents the results of the experiments and Section V shows related work to field mobile deep learning, its energy efficiency and frameworks for running mobile neural networks in mobile devices. Conclusions with future work are proposed in Section VI.

[2]https://www.tensorflow.org
[3]https://opencv.org
[4]https://pjreddie.com/darknet/

II. Background

The main purpose of this work is to find out when and how offloading a deep neural network inference task is convenient, especially under a power consumption point of view. In particular, we focus on the object recognition, namely the task of recognizing the highest number of objects present in a photo and classify them by also giving the coordinates of where they can be found. The most used pattern for classifying objects is the one of setting up a Convolutional Neural Network (CNN), that is a neural network which takes grid-like data as input, and instead of performing matrix multiplications for describing the interaction between neurons' input and output (like in the classic Artificial Neural Network model), they use the convolution operation [13]. The convolution has particular properties that make CNNs very successful with images.

When a CNN is used for classification, the final neurons layer that is used is the *fully connected layer*, which has a number of neurons equal to the number of categories that we are classifying. Instead, performing object detection requires that the final layer is generally replaced with a *detection network*, namely another set of *hidden layers* which are able to localize and classify objects inside the photography given as input the features extracted by the CNN. The two major examples of detection networks are Fast R-CNN [14] and SSD [15]. This is one approach to the object detection, but it is not the only one, indeed CNNs like YOLO [16] do not use a final detection network, but they try to do all at once.

The performance of a neural network of the kind mentioned above, is described by:

- the mean average precision (mAP), that is the mean average precision across all the categories of objects that the network can recognize;
- the FLOPS, as already mentioned, the number of operations that the network requires for generating the output.

III. Experimental Setup

Our purpose is, first of all, to set up a complete environment which allows recognizing objects that are framed by the mobile camera. Once the environment is ready, as a first case we consider that the mobile acts autonomously, then the device will be "assisted" by a backend which is characterized by a medium-level GPU and offers a very low latency communication with the device. This is the common environment that is provided, for example, in a fog/edge infrastructure [17].

In our setup, we consider that we are offering a real-time application, and therefore the maximum inference latency per-frame that we can expect is $\approx$ 50ms. If we consider that the device camera is able to provide 30fps, then we are allowing that the recognition does not fall under the 20fps limit. This is a relaxed limit since here we did not test the most advanced hardware, but it is still acceptable for a subset of AR applications.

In the tests that we conducted, we used two different libraries for setting up the task environment. The two environments make use of the following libraries for running a CNN: Tensorflow, OpenCV and Darknet.

A. Mobile

The first step for running a mobile object recognition task is to set up an application which is able to capture the camera frames and process them one by one. Once the frame has been captured, this must be passed to a neural network that tries to find the objects and then returns their class and their position within a box. According to the library that is used, there are different strategies that we can follow.

OpenCV The OpenCV library is an open-source library which implements a very high number of features that are specifically related to the computer vision: image/video processing and machine learning tasks. The other core feature of the library is that it is available for almost any computing platform, and in particular the Android version comes with some pre-written solutions for capturing the camera frames and running a callback function for each of them. In particular, all the frames that pass when we are processing one frame are lost, since a new frame is processed only when the callback function returns.

The first version of our mobile application for running the experiments has been built completely with OpenCV 4.2.0 (Figure 2). The DNN subpackage of the library allows loading the main model formats for neural networks[5] like TensorFlow, Caffe, Torch, ONNX and Darknet. In our experiments, as shown in Figure 2, we run the *TinyYoloV3* [16] CNN a reduced version of the YOLO network. This first kind of experiment showed a big drawback of the OpenCV library: the lack of GPU support. The obtained values are in line with the ones obtained in other works [18].

The complete inference process that we built is the following:

1) when a frame is captured by the camera the callback function `OnCameraFrame` is called by passing as a parameter the frame in the OpenCV `Mat` format;
2) the frame is converted to RGB (from RGBA if we use the OpenCV camera object or YUV if we use the native camera object), scaled to fit the neural network input size (for example 416x416 for TinyYOLOV3, or 300x300 for MobileNet) and set as input to the neural network;
3) the neural network is run with the `forward` command;
4) the output matrix is parsed and non-maximum suppression [19] is applied to the output boxes;
5) boxes are drawn in the input frame;

TensorFlow Lite TensorFlow Lite is a completely different library with respect to TensorFlow, it supports a reduced set of its features, but it is optimized for running with low-power devices like smartphones or Raspberry Pi[6]. These optimizations require that the TF models must be converted in a format that is readable by the library, the ".tflite" format. Differently from the OpenCV library, TensorFlow Lite natively allows using the device's GPU, thus allowing decreasing the inference time drastically.

Fig. 2: OpenCV object detection with YOLO

The TensorFlow team provides many examples of how using their libraries, in particular for TensorFlow Lite there is an example application which implements the real-time object recognition by using MobileNet neural network [7] and the mobile camera. The main difference with the OpenCV solution is that the application offers two image layers, in the lower one the camera frames are directly displayed, even if we are doing inference on them, in the upper one only the object boxes are displayed; this strategy allows making the camera stream fluid during the inference process.

The second version of our experimental application has been built upon the example mentioned above, some core parts have been rewritten to implement a benchmark process that takes as input a video file and allows analyzing it frame by frame. Figure 1 shows two screenshots of the application. On the left, there is the real-time object recognition with the essential parameters displayed on the screen, and on the right, there is the activity which we implemented for loading videos. As we will see in Section IV, for every frame we log the frame number, the inference and network latency, the battery percentage, the instantaneous current (mA), residual battery capacity (mAh) and the battery voltage (mV). The inference process is precisely equal to the OpenCV version with the only difference that the inference result is not directly drawn to the frame but is passed to a layer that draws boxes on top of the camera frame. Within the application, the neural network that is implemented is MobileNet [20], which is a class of CNN specifically designed for mobile and embedded devices deployment.

B. Edge

We implemented the object recognition service by firstly using TensorFlow and then Darknet by wrapping them with the

[5]https://docs.opencv.org/master/d6/d0f/group__dnn.html
[6]https://www.raspberrypi.org/

[7]https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android

Python Flask[8] library. Finally, we attached the mobile device to a Wi-Fi hotspot created from the PC, thus simulating an edge offloading infrastructure. The working paradigm is the following:

1) the device captures the frame and scales it to match the neural network input size – in this phase, the mobile also choose the compression level of the image, that is a critical parameter since it determines the network latency;
2) the edge device that already loaded the neural network performs the inference and returns the result as a response;
3) the mobile device visualizes the results on the screen;

To natively exploit the GPU, before installing any neural network library, we need to obtain the CUDA toolkit and the cuDNN SDK manually. In our experiments, with the latest nVidia drivers 440.33, we used the CUDA toolkit 10.1 and the cuDNN 7.6.5 (compatible with TensorFlow 2.1[9]).

TensorFlow TensorFlow is an open-source library for performing machine learning tasks. As in the mobile case, we used neural pre-trained networks model available at the TensorFlow website[10].

Darknet We used Darknet [21] to run the YOLO neural network. Darknet is a neural network runtime environment written in C, and it is from the same authors of YOLO. The library must be compiled and then imported it in the Python web server script.

C. Equipment

In our experiments, we used a Samsung Galaxy Note8, a smartphone equipped with Exynos Octa 8895 @ 2.31Ghz processor, 6GB of RAM and a Mali G71 MP20 GPU with a computing capability of 374GFlops and 29.80GB/s of memory bandwidth. The OS of the device is Android 9.0 Pie.

As edge device which allows performing object detection, we used a PC with 16GB RAM, AMD FX-8350 processor and an nVidia GTX 1070 GPU with a computing capability of 5.73TFlops and a memory bandwidth of 256.3GB/s. We installed all the neural network frameworks on Ubuntu 18.04 LTS.

IV. Measurements and Results

The experiments have not been conducted by using the device camera but by analyzing a video. The video that we used is a view of the Warsaw city from a car[11]. The original video has been converted from the resolution of 3840x2160 to 720x576, cut to 5 minutes (or 9000 frames) and then analyzed in the device by using the JavaCV library[12], that binds together OpenCV and other media tools, like FFmpeg, the most common library for processing videos. The main reason for this conversion is due to the memory and the time required to load the video. During the video processing, as introduced in Section III, we logged for every frame not only the timings but also the battery data that comes from the *BatteryManager* service of the Android OS. These values have been then cross-validated with a USB power meter, as shown in Figure 6. As far as regards the OS battery values, from the experiments emerged that they are not returned by every device, but every vendor decides whether log or not to log them. Specifically, the Samsung Galaxy Note8 device that we used for tests, the values do not change every time they are requested, but they are updated within a specific interval of time, in particular, the residual battery capacity is updated in multiple of 3.139mAh.

Results of the experiments are summarized in Table I, which describes the environment used along with the timings and battery data. The neural networks that we tested are:

- *FakeNet*, that is a placeholder of processing frame by frame by doing nothing, this kind of test is done to understand which is the baseline consumption of the device;
- *MobileNet*, in the specific test we used a quantized MobileNet v1.0 with SSD for the TF Lite framework[13], and for the TensorFlow framework we MobileNet v2.0 with SSD[14];
- *YOLOTinyV3*, that is a smaller version of the YOLO neural network [16][15].

All the neural networks that we used are pre-trained on the COCO dataset[16], a very large dataset of segmented images with 80 objects categories.

What emerges from the experiments, as we can see in Figure 3 and Figure 4, is that running the object detection remotely allows to save about the 70% of the battery energy, in particular, about $45J$ are saved when the object recognition is offloaded to the edge. Moreover, across the entire test, the recognition task has a constant energy consumption, this is justified by the fact that the number of objects recognized has not a great impact on the number of operation executed by the neural network, we assume that the oscillations are due to the underlying operating system of the device. As far as regards the inference latencies, in our tests, the remotely deployed TinyYOLOV3 allowed to reach about 20FPS, that is slightly less than the locally deployed MobileNet. These values are justified by two factor, first of all, the computational power of the nVidia GPU and, secondly, by the network latency that is in the order of 27ms – a value that depends on the specific Wi-Fi protocol used by the network adapter. In Figure 5 the behaviour of the inference latency is shown, and as we can observe the offloading scheme of the recognition framework allows for a more stable inference time, but this is essentially

[8]https://www.fullstackpython.com/flask.html

[9]https://www.tensorflow.org/install/source#tested_build_configurations

[10]https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

[11]https://archive.org/details/0002201705192

[12]https://github.com/bytedeco/javacv

[13]The codename of the network, available at the TF repository, is `coco_ssd_mobilenet_v1_1.0_quant_2018_06_29`

[14]The codename of the network, available at the TF repository, is `ssd_mobilenet_v1_coco_2018_01_28`

[15]https://pjreddie.com/darknet/yolo/

[16]https://cocodataset.org

justified by Android operating system and all the background services that have been unpredictably activated during the test. What emerges is also that YOLOTinyV3 in remote reaches the same performances of the local MobileNet but with no impact on energy, this is a clear example of the beneficial effect of the offloading mechanism.

Summarizing, offloading the object recognition has almost a zero-impact on the energy consumption, and in real fog/edge deployment with latest "Wi-Fi 6" [22] technology and more powerful GPUs could also allow reaching real-time inference latencies, i.e. 30FPS.

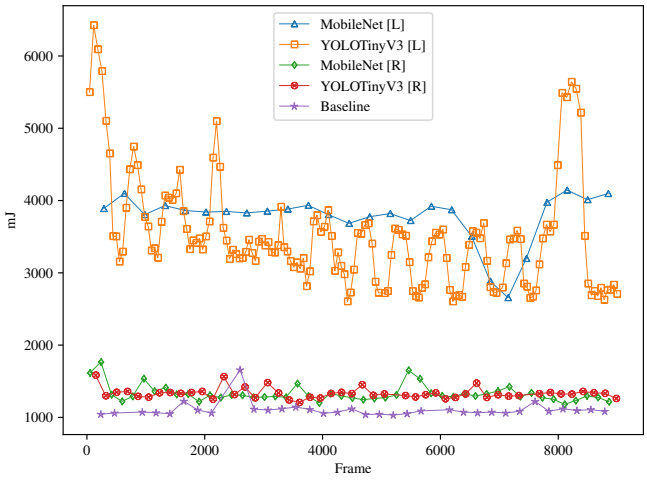

Fig. 3: Power consumption per-frame.

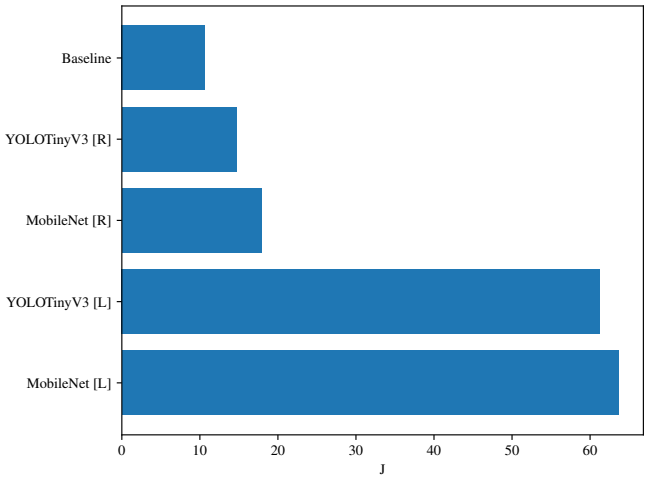

Fig. 4: Total power consumption comparison.

V. Related Work

Different works investigated the energy efficiency of neural networks or tried to provide solutions for assisting mobile devices in computer vision tasks.

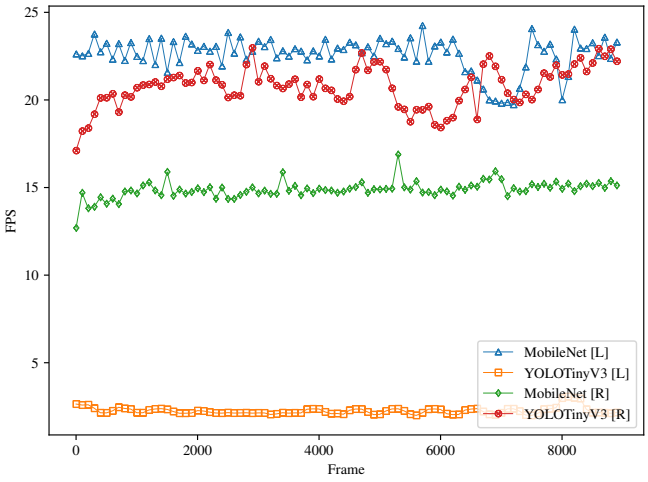

Fig. 5: Inference latency behaviour in FPS.

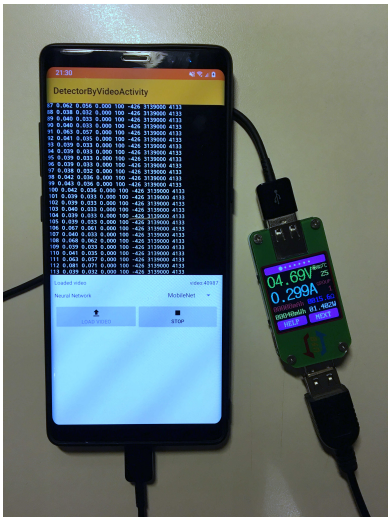

Fig. 6: The USB power meter and the mobile device used for experiments

A. Mobile neural networks

Neural networks represent the core of deep learning [13], they allow performing various machine learning tasks, starting from the simple classification to the generation of new data [23] by using any kind of input data, like numbers, images and audio. The major challenge, that mobile devices open, is the one of making the network most efficient as possible in order to drastically reduce the power consumption but without losing accuracy. This can be done in different ways [24] as pruning some parts of the network, decomposing tensors and quantizing the weights. Every optimization is done to reduce the number of operations that needs to be performed for running the network.

In [24] a method for compressing a CNN is proposed. The method is composed of 3 steps and it is a one-shot process that can be easily implemented by using publicly available tools.

In [20] are presented a series of lightweight neural networks,

		DNN Framework		*Average Time (ms)*			**FPS**	*Energy Consumption*	
	Neural Network	Mobile	Edge	Inference	Network	Total	Average	Instant (mA)	Cumulative (J)
Local	FakeNet	-	-	-	-	-	-	264.31	10.67
	MobileNet	TF Lite	-	46.1	-	46.1	21.70	918.32	63.65
	YOLOTinyV3	OpenCV	-	423.9	-	423.9	2.36	950.97	61.28
Remote	MobileNet	-	TensorFlow	42.1	25.6	67.7	14.80	330.45	17.91
	YOLOTinyV3	-	Darknet	21.9	27.0	48.9	20.44	360.25	14.70

TABLE I: Summary of the main results of the experiments.

called *MobileNets*, that are optimized for running in mobile devices, in particular, they use the "depthwise separable convolution", a form of factorized convolution which is able to drastically reduce the computation load and the size of the model.

In [1] an example of efficient neural network for mobile application is designed and evaluated. The network uses "pointwise group convolution" as for MobileNets, and "channel shuffling", a technique which allows exploiting the convolution in a more computation-wise manner.

B. Frameworks for mobile neural networks

Another set of works is not just focused on the designing the super light neural network but they are aimed to provide a complete solution for allowing deep learning tasks on mobile, both on their own and with the assistance of a backend server. Frameworks like the ones presented in [25]–[27] support computer vision tasks on mobiles by using the GPU and for supporting real-time applications, in particular, they implement an engine that is able to load and run CNN models in a fast and energy-efficient manner. Otherwise, many solutions allow offloading the deep learning tasks to the edge, for example in [12] shows how is possible to have a synergy between the mobile and the edge taking into account parameters like video quality, battery consumption, accuracy and latency in the case of an AR application. A similar proposal is presented in [28] but not designed for real-time applications.

In all of these works what is missing is a clear depiction of the actual energy consumption gain of the offloading task, mainly when we want to use open-source frameworks that are freely available on the web.

VI. Conclusions and Future Work

In this work, we set up a complete environment for allowing a real-time object recognition task, firstly running it locally in the device and then offloading it on the edge. We used free and open-source frameworks, we assessed their maturity and their ease of use. The experiments that we conducted shows that, despite the fact that we can now rely on a big set of neural networks deeply optimized for mobile devices, it is far more convenient to offload a deep learning task to the fog/edge network. This strategy is even more corroborated by the fact that the edge environments are able to offer very low latencies.

However, this is only a first attempt to test the ground on this field. Indeed, we envision as future work to conduct more experiments by exploiting a wider range of neural networks and frameworks for assessing their convenience even with other types of computer vision tasks, also by exploiting the WiFi6 technology that allows to drastically reduce network latencies. Moreover, we also envision to use in edge/fog layer less powerful computing units, like for example Raspberry Pis equipped with specialized ML processing chips (e.g. Coral USB Accelerator[17]).

References

[1] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[2] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 682–687.

[3] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017.

[4] A. Ignatov, N. Kobyshev, R. Timofte, K. Vanhoey, and L. Van Gool, "Dslr-quality photos on mobile devices with deep convolutional networks," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[5] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019.

[6] V. Radu, N. D. Lane, S. Bhattacharya, C. Mascolo, M. K. Marina, and F. Kawsar, "Towards multimodal deep learning for activity recognition on mobile devices," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*. ACM, 2016, pp. 185–188.

[7] Y. Rivenson, H. Ceylan Koydemir, H. Wang, Z. Wei, Z. Ren, H. Günaydın, Y. Zhang, Z. Gorocs, K. Liang, D. Tseng *et al.*, "Deep learning enhanced mobile-phone microscopy," *Acs Photonics*, vol. 5, no. 6, pp. 2354–2364, 2018.

[8] B. Amos, B. Ludwiczuk, M. Satyanarayanan *et al.*, "Openface: A general-purpose face recognition library with mobile applications," *CMU School of Computer Science*, vol. 6, 2016.

[9] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering deep learning to mobile devices via offloading," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 2017, pp. 42–47.

[10] R. Xie, X. Jia, L. Wang, and K. Wu, "Energy efficiency enhancement for cnn-based deep mobile sensing," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 161–167, June 2019.

[11] N. D. Lane and P. Georgiev, "Can deep learning revolutionize mobile sensing?" in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM, 2015, pp. 117–122.

[12] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[17] https://www.coral.ai/products/accelerator

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[14] R. Girshick, "Fast r-cnn," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.
[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.
[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
[17] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*. ACM, 2015, pp. 37–42.
[18] J. Pedoeem and R. Huang, "Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers," *arXiv preprint arXiv:1811.05588*, 2018.
[19] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.
[21] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.
[22] W.-F. Alliance, "Wi-fi 6: High performance, next generation wi-fi," 2018, Tech. Rep., 2018. [Online]. Available: https://www.wi-fi.org/download.php?file=/sites/default/files/private/Wi-Fi_6_White_Paper_20181003.pdf
[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
[24] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.06530
[25] L. N. Huynh, R. K. Balan, and Y. Lee, "Deepsense: A gpu-based deep convolutional neural network framework on commodity mobile devices," in *Proceedings of the 2016 Workshop on Wearable Systems and Applications*. ACM, 2016, pp. 25–30.
[26] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 82–95.
[27] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 2016, p. 23.
[28] S. Tuli, N. Basumatary, and R. Buyya, "Edgelens: Deep learning based object detection in integrated iot, fog and cloud computing environments," *arXiv preprint arXiv:1906.11056*, 2019.