How is Energy Consumed in Smartphone Deep Learning Apps?
Executing Locally vs. Remotely

Haoxin Wang*, BaekGyu Kim†, Jiang Xie*, and Zhu Han‡
*University of North Carolina at Charlotte, Charlotte, NC 28223, U.S.A.
†Toyota Motor North America (TMNA) R&D InfoTech Labs, U.S.A.
‡University of Houston, Houston, TX 77004, U.S.A.

***Abstract*—Applying deep learning to object detection provides the capability to accurately detect and classify complex objects in the real world. However, currently, few mobile applications use deep learning because such technology is computation- and energy-intensive. This paper, to the best of our knowledge, presents the first detailed experimental study of the smartphone's energy consumption and the detection latency of executing deep Convolutional Neural Networks (CNN) optimized object detection, either locally on the smartphone or remotely on an edge server. We experiment with a variety of smartphones, obtaining different levels of computation capacities, in order to ensure that we are not profiling a specific device. Our detailed measurements refine the energy analysis of smartphones and reveal some interesting perspectives regarding the energy consumption of executing the deep CNN optimized object detection. We believe that these findings will guide the design of energy efficient processing pipeline of the CNN optimized object detection.**

I. Introduction

With the advancement in *Deep Learning* in the past few years, we are able to create complex machine learning models for detecting objects in real-time video frames. This advancement has the potential to make Augmented Reality (AR) devices highly intelligent and enable industries to favor machine learning models with superior performance. For example, AR automotive applications (e.g., deep learning-based AR head-up-displays (HUDs)) are promised to help increase road safety, bring intuitive activities to driving, and enhance driving experience in the future. Meanwhile, as people nowadays are using their smartphones to a larger extent and also expect increasingly advanced performance from their mobile applications, the industry needs to adopt more advanced technologies to meet such expectations. One such adoption could be the use of deep learning-based AR applications.

However, few mobile AR applications use deep learning today because of inadequate infrastructure support (e.g., limited computation capacity and battery resource of smartphones). Deep learning algorithms are computation-intensive, and executed locally in ill-equipped smartphones may not provide acceptable latency for end users. For instance, in Deepmon [1], it takes approximately 600 ms for small and medium *convolutional neural network* (CNN) models and almost 3 seconds for large CNN models to process one frame, which is obviously not acceptable for real-time processing.

Two research directions have emerged to address this challenge. The first direction is to tailor the computation-intensive deep learning algorithms to be executed on smartphones. For instance, Tiny-YOLO [2] that has only 9 convolutional layers (24 convolutional layers in a full YOLO network) is developed and optimised for the use on embedded and mobile devices. Tensorflow Lite [3] is TensorFlow's lightweight solution for embedded and mobile devices. It enables low-latency inference of on-device machine learning models with a small binary size and fast performance supporting hardware acceleration. However, the reduction of the inference latency is at the cost of the precision degradation of the detection. The other research direction that is widely used in running deep learning in smartphones is to transfer all the computation data to more powerful infrastructures (e.g., the remote cloud and edge servers) and execute deep learning algorithms there [4]–[6]. Such offloading-based solutions can reduce the inference latency and extend smartphones' battery life only when the network access is reliable and sufficiently fast.

Our motivation. Although the complexity and capabilities of smartphones are growing at an amazing pace, smartphones are expected to continually become lighter and slimmer. When combined with energy-hungry deep learning-based applications, the limited battery capacity allowed by these expectations now motivates significant investment into smartphone power management research. In order to better investigate and understand the relationship between the energy consumption and the performance of deep learning-based applications, we propose the following questions:

1) *How is energy consumed when a deep learning-based application is executed locally?*
2) *Does smartphone's computation capacity impact the energy consumption when a deep learning-based application is executed locally?* It is intuitive that the smartphone with higher computation capacity can achieve a lower inference latency. However, the energy consumption is more complicated to analyze. This is because, for example, the smartphone with more powerful processors may drain its battery faster.
3) *Does transferring all the computation data to a powerful infrastructure significantly decrease the energy consumption and latency?* When a deep learning-based application is executed remotely, communication latency is non-negligible and unstable, especially in wireless networks. Previous work [7] shows that smartphone's radio interfaces account for up to 50% of the total power budget. In addition, improved communication speeds generally

This work was supported in part by the US National Science Foundation (NSF) under Grant No. 1718666, 1731675, 1910667, and 1910891 and funds from Toyota Motor North America.

978-1-7281-0962-6/19/$31.00 ©2019 IEEE
Authorized licensed use limited to: University of North Carolina at Charlotte. Downloaded on October 28,2020 at 07:55:11 UTC from IEEE Xplore. Restrictions apply.

TABLE I
SMARTPHONES AND THE EDGE SERVER USED IN OUR STUDY.

Manufacturer	Samsung	Google	Asus	Nvidia
Model	Galaxy S5	Nexus 6	ZenFone AR	Jetson AGX Xavier
OS	Android 6.0.1	Android 5.1.1	Android 7.0	Ubuntu 18.04 LTS aarch64
SoC	Snapdragon 801 (28 nm)	Snapdragon 805 (28 nm)	Snapdragon 821 (14 nm)	Xavier
CPU	32-bit 4-core 2.5GHz Krait 400	32-bit 4-core 2.7GHz Krait 450	64-bit 4-core 2.4GHz Kryo	64-bit 8-core 2.26GHz Carmel
GPU	578MHz Adreno 330	600MHz Adreno 420	653MHz Adreno 530	512-core 1377MHz Volta with 64-TensorCores
RAM	2GB	3GB	6GB	16GB
WiFi	802.11n/ac, MIMO 2×2	802.11n/ac, MIMO 2×2	802.11n/ac/ad, MIMO 2×2	—
Release date	April 2014	November 2014	July 2017	September 2018

come at the cost of higher power consumption [8].

4) *Besides the network condition, what impact the energy consumption and latency when executed remotely, and how?*

Our contributions. In this paper, we conduct the first detailed experimental study of the energy consumption and the performance of a deep CNN optimized object detection application on smartphones. We experiment with a variety of smartphones, obtaining different levels of computation capacities, in order to ensure that we are not profiling a specific device. *Our goal is to identify common trends across different devices that can potentially guide the design of energy optimizations for future CNN optimized object detection applications.* We make the following contributions:

1) Developing two Android applications that perform real-time object detection: one is running a small CNN locally on the smartphone and the other is running a large CNN remotely on an edge server.
2) Evaluating the energy consumption and the latency of each phase in the implemented end-to-end deep CNN optimized object detection processing pipeline. We perform these measurements on multiple smartphones with different levels of computation capacities.
3) Evaluating the impact of smartphone image post processing on smartphone's power consumption and object detection performance and providing insights on future energy-efficient design.

II. RELATED WORK

Deep learning. In recent years, applying CNNs to object detection has been proven to achieve excellent performance [1], [2], [9]. In [10], the speed and accuracy trade-offs of various modern CNN models are compared. However, none of these works considered the performance of running CNNs on smartphones.

Experimental study on CNNs. Although existing papers have extensively investigated how to run CNN models on mobile devices, including model compression of CNNs [11], GPU acceleration [1], and only processing important frames [12], none of these works considered the energy consumption of executing CNNs on smartphones. In [13], a small number of measurements on the battery drain of running a CNN on a powerful smartphone are conducted. However, firstly, its battery drain results are reported by the Android OS that can only provide coarse-grained results. For example, it only shows the total battery usage of running a CNN on a smartphone for 30 minutes. Secondly, it only studies running CNNs on smartphones with high computation capabilities and the experimental results are not comparable to smartphones with poor computation capabilities.

III. EXPERIMENTAL METHODOLOGY

A. Hardware setup

Our study was performed using three different smartphones. We summarize their characteristics in Table I. We classify them into two classes, *low-end* and *high-end* smartphones, according to their general hardware performance tested by using an Antutu benchmark [14]. The testing results are shown in Table II. In addition, we emulate an edge server with an Nvidia Jetson AGX Xavier, which connects to a WiFi access point (AP) through a 1Gbps Ethernet cable. Details of our equipped edge server are shown in Table I.

TABLE II
CLASSIFICATIONS OF THE TESTED SMARTPHONES.

Smartphone	S5	Nexus 6	ZenFone AR
CPU score	36871	37521	58531
GPU score	6678	18063	67286
Image processing score	3103	6862	11321
Total score	66414	80047	173472
Class	Low-end	Low-end	High-end

B. Software implementation

Edge server side. The edge server is developed to process the video frames and send the detection results back to smartphones. We implement two major modules on the edge server. The first one is the communication service handler module which performs authentication and establishes a socket connection with smartphones. This module is also responsible for dispatching the detection results to corresponding smartphones. The second one is the object detection module which is designed based on a custom framework called Darknet [15] with GPU acceleration and runs YOLOv3 [2], a large neural network model with 24 convolutional layers. The YOLOv3 model used in our experiments is trained on COCO dataset [16] and can detect 80 classes.

Smartphone side. We implement two scenarios for our experimental study. The first one is executing deep learning on smartphones, defined as *local execution*. In this scenario,

Authorized licensed use limited to: University of North Carolina at Charlotte. Downloaded on October 28,2020 at 07:55:11 UTC from IEEE Xplore. Restrictions apply.

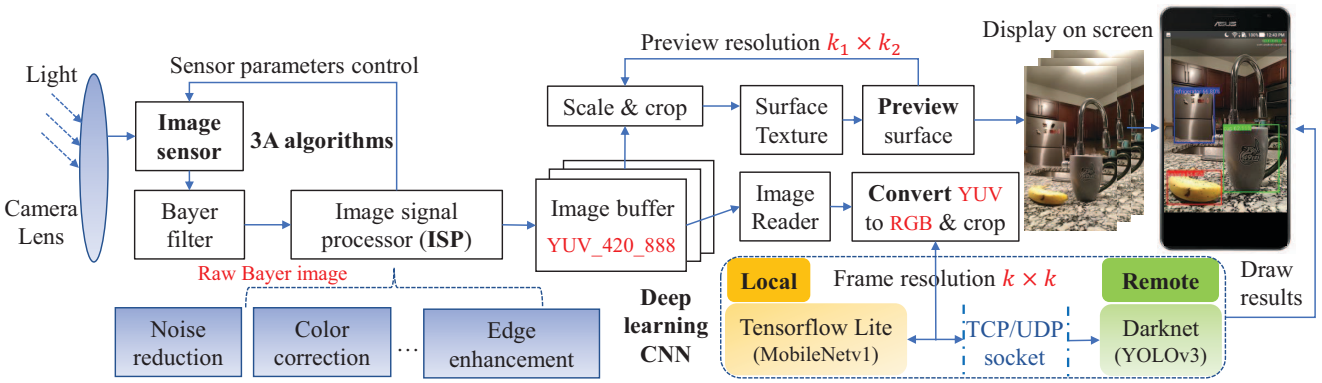


Fig. 1. Processing pipeline of the deep CNN optimized object detection application implemented in this paper.

the Android implementation is based on a light framework called Tensorflow Lite [3] which is TensorFlow's lightweight solution for embedded and mobile devices. It runs a small neural network model, called MobileNetv1 [11]. In order to run MobileNetv1 with different frame resolutions in Tensorflow Lite on smartphones, we convert a pre-trained MobileNetv1 SSD model to the FlatBuffers format. The second one is executing deep learning on our equipped edge server, defined as *remote execution*. In this scenario, a smartphone transfers the converted RGB frames to the edge server through a socket connection in real time. To avoid having the server process stale frames, the smartphone sends the latest captured frame to the server and waits to receive the detection result before sending the next frame for processing. The detailed processing pipeline is shown in Fig. 1.

C. Power measurement setup

To measure the power consumption, we use an external power monitor, a Monsoon Power Monitor, to provide power supply for the smartphone. Different from old smartphone models, modern smartphones like Nexus 6 have very tiny battery connectors, making it very challenging to connect the power monitor to them. To solve this problem, we modify the battery connection of Nexus 6 by designing a customized circuit and soldering it to the smartphone's power input interface. In addition, the power measurements are taken with the screen on, with the Bluetooth/LTE radios disabled, and with minimal background application activity, ensuring that the smartphone's *base power* is low and does not vary unpredictably over time. For the measurements of the power consumption in local execution, base power is defined as the power consumed by the smartphone when its WiFi interface is turned off. For the measurements of the power consumption in remote execution, base power is defined as the power consumed when the smartphone is connected to the AP without any data transmission activity [7], [17].

IV. Experimental Results

In this section, we describe our efforts towards measuring and understanding the energy consumption and the performance of running deep CNNs on both high-end and low-end smartphones.

A. Key metrics

Currently, object detection applications focus on the following two critical metrics:

1) Latency/frames per second (FPS): Latency is the total time needed to obtain the detection results on one video frame (i.e., usually shown as one or multiple bounding boxes that identify the location and classification of the objects in a frame). In this paper, it is defined as the time period from the moment the *Image Reader* acquiring one camera captured image frame to the moment the bounding boxes are drawn on the smartphone's screen, as depicted in Fig. 1. In local execution, the per frame total latency includes the time used for converting the YUV frame to the RGB frame, cropping the frame to the fitted resolution $k \times k$, and executing deep learning, defined as *inference latency*, on the smartphone. In remote execution, the per frame total latency includes, besides the convert and crop latencies that are both executed locally on the smartphone, the communication latency (i.e., transmitting the frame and receiving the results) and the inference latency on the edge server.

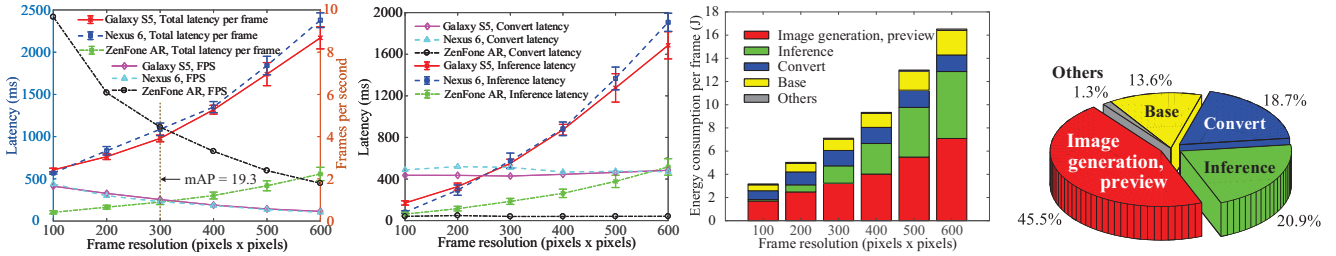
2) Accuracy: The mean average precision (mAP) is a commonly used performance metric in object detection. Better performance is indicated by a higher mAP value. Specifically, the average precision [18] is computed as the area under the precision/recall curve through numerical integration. The mAP is the mean of the average precision across all classes.

B. Local execution vs. remote execution

We first evaluate the object detection performance of both local execution and remote execution in terms of latency, FPS, accuracy, and energy consumption, as shown in Fig. 2 and 3. The preview resolution is set to $k_1 \times k_2 = 640 \times 480$ pixels. In remote execution, we use a WiFi 5 GHz channel and TCP socket connection to transfer data between smartphones and the edge server.

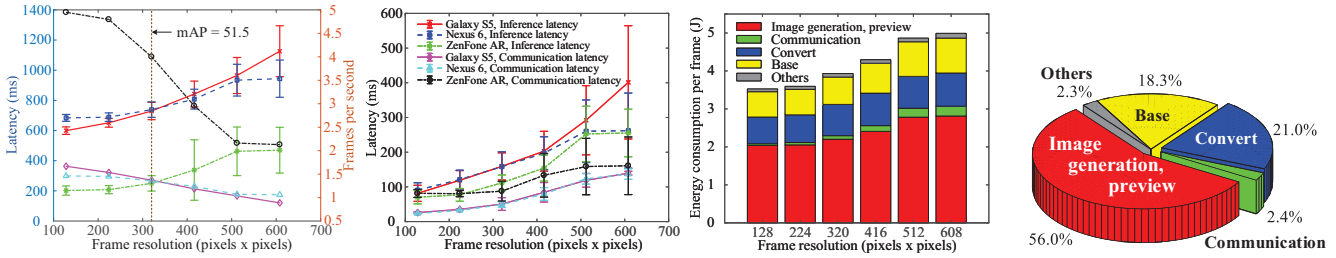
Local execution. First, we examine the total latency of executing object detection with different frame resolutions, from 100×100 to 600×600 pixels, in our three smartphones. The experimental results are shown in Fig. 2(a). We find that (1) *a higher frame resolution always results in a higher per frame total latency.* For example, for Nexus 6, the per frame total latency surges from 569.8 ms to 2378.7 ms when the

Authorized licensed use limited to: University of North Carolina at Charlotte. Downloaded on October 28,2020 at 07:55:11 UTC from IEEE Xplore. Restrictions apply.



(a) Total latency per frame and FPS. (b) Convert and inference latency per frame. (c) Average energy consumption per frame breakdown (Nexus 6). (d) Average percentage breakdown of energy consumed in executing 300 × 300 MobileNetv1 SSD model (Nexus 6).

Fig. 2. Local execution results.



(a) Total latency per frame and FPS. (b) Inference and communication latency per frame. (c) Average energy consumption per frame breakdown (Nexus 6). (d) Average percentage breakdown of energy consumed in executing 320 × 320 YOLOv3 model (Nexus 6).

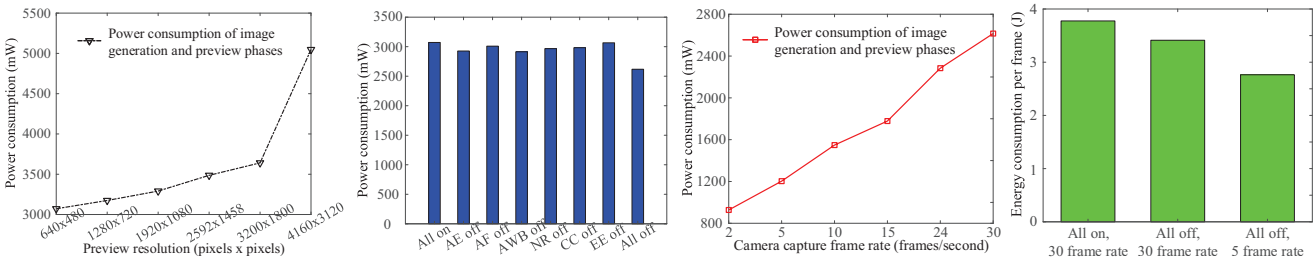
Fig. 3. Remote execution results.

frame resolution increases from 100×100 to 600×600 pixels. (2) *The high-end smartphone achieves a significantly lower per frame total latency compared to the low-end smartphones.* For example, when the frame resolution is 300×300 pixels, the per frame total latency of ZenFone AR is only 20.7% and 22.9% of that of Nexus 6 and Galaxy S5, respectively.

Second, we measure the latency of each phase in the processing pipeline. We show the latency of the two highest time-consuming phases, convert and inference latency, in Fig. 2(b), which comes up to 95% of the per frame total latency. We find that (1) *for both high-end and low-end smartphones, the convert latency does not vary much when the frame resolution increases.* This is because no matter what the frame resolution $k \times k$ is configured, every YUV frame is converted to an RGB frame with the preview resolution $k_1 \times k_2$ first. After the convert is completed, the RGB frame will be resized to $k \times k$ pixels. (2) *For low-end smartphones, the largest time-consuming phase is converting a YUV frame to an RGB frame when the frame resolution is smaller than* 300×300 *pixels. In contrast, when the frame resolution is larger than* 300×300 *pixels, inference becomes the largest latency source.* For example, for Nexus 6, the convert latency is 85.6% of the per frame total latency when the frame resolution is 100×100 pixels; whereas the inference latency is 80.2% of the per frame total latency when the frame resolution is 600×600 pixels. This is rather significant because most of the previous work that evaluates the per frame latency of object detection executed in smartphones only consider the inference latency. However, our experimental results indicate that the convert latency is non-negligible and sometimes larger than the inference latency. (3) Interestingly, *when the frame resolution is small, the inference latency of the high-end and low-end smartphones are comparable. However, the convert latency of the high-end smartphone is always considerably smaller than that of the low-end smartphones.* For example, when the frame resolution is 100×100 pixels, the inference latencies of ZenFone AR and Nexus 6 are 62.5 ms and 81.3 ms, respectively, whereas the convert latency of ZenFone AR is 41.2 ms which is only 8.4% of the convert latency of Nexus 6. This result indicates that when the frame resolution is low, converting a YUV frame to an RGB frame is more computation-intensive than running a small CNN in smartphones and low-end smartphones are unable to generate the same convert performance as high-end smartphones in terms of the convert latency.

Third, to dissect the energy drain through different processing pipeline phases, we first break down the per frame total energy consumption as follows: image generation, preview, inference, convert, base, and others. We make the following observations from Fig. 2(c) and 2(d). (1) *The image generation and the preview always contribute the highest energy consumption in the smartphone and grows significantly as the frame resolution increases.* Specifically, in Nexus 6, when the frame resolution is 300×300 pixels, on average a whopping 45.5% of the per frame total energy consumption is from the image generation and preview. The reason why the image generation process consumes considerably high energy is executing the 3A (i.e., auto-focus (AF), auto-exposure (AE), and auto-white-balance (AWB)) and multiple fine-grained image post processing algorithms (e.g., noise reduction (NR), color correction (CC), and edge enhancement (EE)) on image signal processor (ISP). These sophisticated algorithms are designed to make an image that is captured by the smartphone camera look perfect. *However, is it always necessary for the camera captured frame to be processed by all of those energy-hungry image processing algorithms in order to achieve a successful*

Authorized licensed use limited to: University of North Carolina at Charlotte. Downloaded on October 28,2020 at 07:55:11 UTC from IEEE Xplore. Restrictions apply.



(a) Preview resolution vs. power consumption.

(b) 3A and image post processing algorithms vs. power consumption.

(c) Camera capture frame rate vs. power consumption.

(d) Comparison of the energy consumption per frame (remote execution).

Fig. 4. Power consumption analyses of the image generation and preview phases.

object detection result? In addition, the number of frames captured by the camera per second is a fixed value (e.g., 24 or 30 frames/second) or in a range (e.g., $[7, 30]$ frames/second), which is controlled by the AE algorithm. However, for low-end smartphones, even if the frame resolution is small, the detection FPS is still less than 2, as shown in Fig. 2(a), which is far slower than the camera capture frame rate. Furthermore, the CNN always extracts the latest captured frame, which indicates that, *from the perspective of the energy efficiency of the object detection pipeline, capturing frames with a fast rate is unnecessary and energy-inefficient.* (2) *The inference energy consumption grows dramatically as the frame resolution increases.* For example, it accounts for 4.1% and 33.9% of the per frame total energy consumption when the frame resolution is 100×100 and 600×600 pixels, respectively.

Remote execution. We next compare against the remote execution scenario where the CNN is run on the implemented edge server with a 5GHz WiFi link to the smartphone. Note that we conduct our measurements in different network conditions (e.g., the Received Signal Strength Indicator (RSSI) at the tested smartphones or the network bandwidth gradually drops down). However, due to the page limitation, we only present our experimental results obtained in an excellent network condition (i.e., the RSSI is in the range of -15 and -20 dBm). First, we compare the latency and FPS, as shown in Fig. 3(a) and 3(b), and make the following observations. (1) *For the high-end smartphone, the per frame total latency (FPS) is larger (lower) than that of the local execution scenario when the frame resolution is smaller than* 512×512 *pixels* (note that this observation may differ depending on how powerful the server's GPU is). This observation supports the fact that lots of recently released smartphones with high computation power possess the capability to run a small CNN model. However, the mAP of the large CNN model on the server is better than that of the small CNN model on the smartphone (e.g., $mAP = 51.5$ on the server and $mAP = 19.3$ on the smartphone when the frame resolution is around 300×300 pixels). Generally, different implementation cases have variant latency/accuracy requirements. For example, the AR cognitive assistance case where a high-end wearable device helps visually impaired people to navigate on a street may need a low latency but can tolerate a relatively high number of false positives (i.e., false alarms are fine but missing any potential threats on the street is costly) [13]. In contrast, an AR used for recommending products in shopping malls or supermarkets may tolerate a long latency but require high detection accuracy. Therefore, choosing the appropriate execution approach (i.e., local or remote) in different implementation cases is critical.

Furthermore, (2) *for the low-end smartphones, the per frame total latency (FPS) is slightly lower (higher) than that of the local execution scenario.* The reason why the latency does not decrease significantly is the high convert latency which is executed by the smartphone in both local and remote execution cases. This observation is rather significant for deciding what computation tasks should be transferred to the server. Most of the existing works simply consider transferring the converted RGB frames to the server. *However, for low-end smartphones, only executing the CNN in the server is inadequate to achieve an acceptable FPS. Converting YUV to RGB frames remotely is also desirable.* (3) Interestingly, as shown in Fig. 3(b), *the frames transmitted by ZenFone AR obtain less inference latency than the frames transmitted by Galaxy S5 under the same conditions (i.e., the same frame resolution and camera view).* We repeated these measurements several times and got the same results although, until now, we do not have a definite explanation for this result.

In addition, Fig. 3(c) and 3(d) analyze the energy drain of the smartphone by different processing pipeline phases in the remote execution case, including image generation, preview, communication, convert, base, and others. Compared to the local execution scenario, we have the following observations. (1) *Similar to the local execution, image generation and preview are the biggest energy consuming phases.* For example, it comes up to 56.0% of the per frame total energy consumption when the frame resolution is 320×320 pixels. (2) *Transmitting one frame and receiving the result consume less energy than our expectation, when the wireless network condition is excellent.* For example, on average it only accounts for 2.4% of the per frame total energy consumption when the frame resolution is 320×320 pixels. (3) *The remote execution saves approximately* 53% *energy per frame on average when the frame resolution is larger than* 128×128 *pixels.* However, it consumes 12.9% more energy per frame than the local execution when the frame resolution is 128×128 pixels. This observation is rather significant, which demonstrates that *running deep learning remotely does not always consume less energy than the local execution, even when the network quality*

Authorized licensed use limited to: University of North Carolina at Charlotte. Downloaded on October 28,2020 at 07:55:11 UTC from IEEE Xplore. Restrictions apply.

(a) All enabled. (b) All disabled.

Fig. 5. Comparison of the object detection results (remote execution).

is excellent.

C. Power consumption of the image generation and preview

As we observed in Section IV-B, the image generation and the preview are the most energy-consuming phases in both local and remote execution cases. Thus, to reduce the energy consumption of the object detection processing pipeline, we must improve the energy efficiency of these two phases. We seek to understand the interactions between the power consumption and various factors (e.g., the preview resolution, 3A, and several image post processing algorithms) as follows.

Preview resolution vs. power consumption. We first examine how the preview resolution influences the power consumption of the image generation and preview phases, as shown in Fig. 4(a). We find that *as the preview resolution grows, the power consumption increases dramatically.* Therefore, a preview with a higher frame resolution on the smartphone provides a better quality preview for users, but at the expense of battery drain, which is applicable for both local and remote execution cases.

Image post processing and 3A algorithms vs. power consumption. We next examine the effect of multiple image post processing and 3A algorithms on the power consumption of the image generation and preview phases, as shown in Figs. 4(b) and 4(c). Note that when the AE is disabled, we manually set the camera ISO and exposure time to 400 and 20 ms, respectively. We observe that (1) *disabling the 3A, NR, CC, and EE algorithms decreases the power consumption by* 14.8%. We conduct another experiment to understand if disabling these algorithms would impact the object detection performance. As shown in Fig. 5, the detection performance does not degrade. (2) *Accelerating the camera capture frame rate significantly increases the power consumption.* As we discussed in Section IV-B, the maximum detection FPS that the low-end smartphones can obtain is around 2; a capture rate larger than 2 frames/second is unnecessary and energy-inefficient from the perspective of energy efficiency. Furthermore, we compare the per frame energy consumption among three cases, as depicted in Fig. 4(d): all enabled with camera capture frame rate 30, all disabled with camera capture frame rate 30, and all disabled with camera capture frame rate 5. We find that (3) *the per frame energy consumption of the second and the third cases decreases by approximately* 10% *and* 27%*, respectively, compared to the first case.*

V. Conclusion

In this paper, we presented the first detailed experimental study of the energy consumption and the performance of a deep CNN optimized object detection application on a variety of smartphones. We examined both local and remote execution cases. We found that the performance of the object detection is heavily affected by different generations of smartphones. Although executing deep learning on remote edge servers is one of the most commonly used approaches to assist low-end smartphones in improving their energy efficiency and performance, contrary to our expectation, remote execution does not always consume less energy and obtain lower latency, as compared to local execution, even when the network quality is excellent. Overall, we believe that our findings give great insights and guidelines to the future design of energy-efficient processing pipeline of CNN optimized object detection.

References

[1] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile GPU-based deep learning framework for continuous vision applications," in *Proc. ACM Mobisys*, 2017, pp. 82–95.
[2] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.
[3] "Tensorflow lite," https://www.tensorflow.org/lite/.
[4] H. Wang, J. Xie, and T. Han, "A smart service rebuilding scheme across cloudlets via mobile AR frame feature mapping," in *Proc. IEEE ICC*, 2018, pp. 1–6.
[5] P. Jain, J. Manweiler, and R. R. Choudhury, "Low bandwidth offload for mobile AR," in *Proc. ACM CoNEXT*, 2016, pp. 237–251.
[6] H. Wang, B. Kim, J. Xie, and Z. Han, "E-auto: A communication scheme for connected vehicles with edge-assisted autonomous driving," in *Proc. IEEE ICC*, 2019, pp. 1–6.
[7] H. Wang, J. Xie, and X. Liu, "Rethinking mobile devices' energy efficiency in WLAN management services," in *Proc. IEEE SECON*, 2018, pp. 1–9.
[8] S. K. Saha, P. Deshpande, P. P. Inamdar, R. K. Sheshadri, and D. Koutsonikolas, "Power-throughput tradeoffs of 802.11 n/ac in smartphones," in *Proc. IEEE INFOCOM*, 2015, pp. 100–108.
[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proc. Advances in Neural Information Processing Systems*, 2015, pp. 91–99.
[10] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proc. IEEE CVPR*, 2017.
[11] 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," *arXiv preprint arXiv:1704.04861*, 2017.
[12] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. ACM Sensys*, 2015, pp. 155–168.
[13] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering deep learning to mobile devices via offloading," in *Proc. ACM Workshop on Virtual Reality and Augmented Reality Network*, 2017, pp. 42–47.
[14] "Antutu benchmark," https://www.antutu.com/en/.
[15] J. Redmon, "Darknet: Open source neural networks in C," http://pjreddie.com/darknet/, 2013–2016.
[16] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proc. European Conference on Computer Vision*, 2014.
[17] A. M. Srivatsa and J. Xie, "A performance study of mobile handoff delay in IEEE 802.11-based wireless mesh networks," in *Proc. IEEE ICC*, 2008, pp. 2485–2489.
[18] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

Authorized licensed use limited to: University of North Carolina at Charlotte. Downloaded on October 28,2020 at 07:55:11 UTC from IEEE Xplore. Restrictions apply.