Real-Time, CNN-Based Assistive Device for Visually Impaired People

Khaled Jouini
Mohamed Hédi Maaloul
and Ouajdi Korbaa

MARS Research Lab LR17ES05
ISITCom
University of Sousse, Tunisia.

***Abstract*—Visual impairment limits people's ability to move about unaided and interact with the surrounding world. This paper aims to leverage recent advances in deep learning to assist visually impaired people in their daily challenges. The high accuracy of deep learning comes at the expense of high computational requirements for both the training and the inference phases. To meet the computational requirements of deep learning, a common approach is to move data from the assistive device to distant servers ($i.e.$ cloud-based inference). Such data movement requires a fast and active network connection and raises latency, cost, and privacy issues. In contrast with most of exiting assistive devices, in our work we move the computation to where data resides and opt for an approach where inference is performed directly "on" device ($i.e.$ on-device-based inference). Running state-of-the-art deep learning models for a real-time inference on devices with limited resources is a challenging problem that cannot be solved without trading accuracy for speed (no free lunch). In this paper we conduct an extensive experimental study of 12 state-of-the-art object detectors, to strike the best trade-off between speed and accuracy. Our experimental study shows that by choosing the right models, frameworks, and compression techniques, we can achieve decent inference speed with very low accuracy drop.**

I. Introduction

Visual impairment limits people's ability to interact with the surrounding world and can negatively affect their socio-economic integration. The World Health Organization estimates that there are approximately 285 million visually impaired and blind worldwide [1]. The prevalence of vision impairment in low- and middle-income regions is estimated to be four times higher than in high-income regions [1]. A common goal in computer vision research is to mimic human visual system and automate tasks that it can perform ($e.g.$, scene recognition, object detection, etc.). In this work, we are interested in leveraging computer vision and deep learning methods to implement an inexpensive yet effective device to assist visually impaired people in their daily challenges.

Object recognition is the cornerstone of devices intended to aid visually impaired people. It consists in drawing a bounding box around each object in an image and associating with each box the class of the surrounded object and a confidence score. Modern object detectors, such as SSD ($i.e.$ Single Shot multi-box Detector) models [2] and YOLO ($i.e.$ You Only Look Once) models [3], [4], [5], [6] are typically built on deep CNNs (Convolutional Neural Networks). In contrast with traditional computer vision methods, modern detectors perform the tasks of feature extraction, object localization, and object classification in one go ($i.e.$, End-To-End).

Due to recent advances that partially solved the vanishing gradient problem ($e.g.$, residual blocs [7]), current CNNs are becoming deeper and more complex (up to tens of layers). Such CNNs can match or even exceed human vision in accuracy and have, by far, outperformed traditional computer vision methods in a variety of fields and in various challenges [8]. However, the high accuracy of deep CNNs comes at the expense of high computational and memory requirements for both the training and the inference phases. Training is computationally expensive due to millions of parameters that need to be iteratively refined [9]. Inference is expensive due to billions of multiply-accumulate (MACs) operations that need to be performed on input data [9]. Due to their high computational requirements, it is difficult to execute deep CNNs on embedded and mobile devices with limited resources [9], [10].

There exist several CNN-based systems designed to support visually impaired people [11], [12], [13], [14]. Unfortunately, most of them inherit the drawbacks of deep CNNs and fail to provide a full on-device, real-time object recognition.

In this work, we intend to provide visually impaired people with an actual real-time on-device inference. In particular, this paper focuses on reviewing and studying recent lightweight CNN-based object detectors. Lightweight CNN-based object detectors aim to be faster and less demanding on resources than conventional CNNs, at the expense of a "small" accuracy loss. Used in conjunction with compression techniques, lightweight architectures make feasible the deployment of deep CNNs on devices with constrained resources. The purpose of our study is to strike the best trade-off between speed and accuracy. To the best of our knowledge, there exists no previous work on assistive devices that compares or uses lightweight CNN architectures. Our work's main contributions are therefore: (i) the implementation of an inexpensive device, specifically tailored for visually impaired people and allowing an effective real-time inference; and (ii) a study on the running of lightweight object detectors on end devices.

The remainder of this paper is organized as follows. Section II briefly reviews previous work on CNN-based object detectors and CNN-based assistive devices. Section III details the implementation steps of our system and presents an extensive experimental study of state-of-the-art lightweight detectors. Finally, section IV concludes the paper.

II. Background

A. One-stage object detectors

Early CNN-based object detectors like R-CNN [15], perform the task of object detection in two stages. They first generate region proposals ($i.e.$, areas that potentially contain an object) and then run a classifier on the proposed regions [15]. While the accuracy of two-stages detectors is often good, they are relatively slow since they require running the model's detection and classification portions several times. Recent one-stage detectors, in contrast, perform only a single pass through the CNN and simultaneously predict multiple bounding boxes and class probabilities in one go. One-stage detectors are in general, faster than two-stage detectors while achieving comparable accuracy.

One-stage detectors are based on two ideas. The first idea consists in framing detection as a regression problem. The second is to use a fixed-size grid of detectors. Two of the most popular one-stage detectors are YOLO [3] and SSD [2].

1) YOLO (You only Look Once): The YOLO method, introduced by Redmon et al. [3] in 2016, was the first proposed one-stage object detector. In its original implementation, YOLO divides spatially each input image into a fixed grid of $S \times S$ cells. Each grid cell predicts k bounding boxes and confidence scores for those boxes. YOLO therefore always predicts the same number of bounding boxes ($S \times S \times k$). As post-processing, boxes having confidence scores below a predefined threshold are dropped out. The remaining boxes undergo a non-maximum suppression to eliminate eventual duplicate detections.

Redmon et al. also proposed YOLOv2 [4] and YOLOv3 [5], respectively in 2017 and 2018. YOLOv2 and YOLOv3 implement several techniques to improve speed and/or accuracy ($e.g.$, batch normalization, k-means, etc.). In particular, YOLOv2 predefines k different box shapes, called anchor boxes, that correspond to the k most common object shapes in the dataset. YOLOv2 chooses anchors by running k-means clustering on all the bounding boxes from all the training images. Each object in a training image is then assigned to the grid cell that contains the object's midpoint and to the anchor box with the highest overlap divided by non-overlap (called Intersect-over-Union, or IoU). While the fixed grid forces the model to learn specialized object detectors in specific locations, anchors force the detectors inside the cells to each specialize in a particular object shape. A salient feature of YOLOv3, the last YOLO version proposed by Redmon et al., is that it uses three different scale grids ($i.e.$ three different scale feature maps) to better handle objects of various scales.

Most of existing CNN-based object detectors are composed of at least two parts, a backbone for feature extraction and a head for object classification and bounding box regression. Some of the most recent object detectors insert in addition some layers, called neck, between the backbone and the head. The neck part is used to collect feature maps from different stages [6]. YOLOv2 uses a custom CNN named Darknet-19 (having 19 layers) as backbone. YOLOv3 is based on a deeper version of Darknet named Darknet-53 (having 53 layers). In addition, YOLOv3 uses FPN (Feature Pyramid Network) as neck to extract features of different scales from the backbone.

TABLE I
Number of layers in regular and lightweight YOLO versions.

Model	# of Layers	Pretrained Convolutional Weights
YOLOv2	32	199 MB
YOLOv2-Tiny	16	43 MB
YOLOv3	106	236 MB
YOLOv3-Tiny	24	34 MB
YOLOv4	162	245 MB
YOLOv4-Tiny	38	23 MB

Bochkovskiy et al. proposed YOLOv4 [6] in 2020. YOLO v4 is an evolution of YOLOv3 that achieves state-of-the-art results by combining two categories of methods: Bag of Freebies (BoF) and Bag of Specials (BoS). BoF refers to methods, such as Mosaic and CutMix data augmentation, DropBlock regularization, and CIoU loss, that improve accuracy without increasing the inference time. BoS refers to methods, such as Weighted-Residual-Connections (WRC), Cross-Stage-Partial-connections (CSP), and Mish-activation, that slightly increase the inference time but significantly improve accuracy. YOLOv4 uses CSPDarknet53 [16] as backbone, Spatial pyramid pooling (SPP), and Path Aggregation Network (PAN) as neck, and the same head as YOLOv3. YOLOv4 also selects optimal hyper-parameters by applying genetic algorithms.

Along with regular versions, lightweight versions of YOLO (with the suffix "-tiny") were also proposed. Lightweight YOLO versions use less convolutional and pooling layers than regular architectures (Table I), which results in a higher processing speed, but also in a reduced accuracy. YOLOv3-tiny and YOLOv4-tiny, in addition, use two feature maps instead of the three used in the regular corresponding models.

2) SSD (Single Shot MultiBox Detector): Conceptually, YOLO, and SSD [2] follow the same principles and differ only in details. To detect objects of various scales, the input image in SSD is passed through a series of convolutional layers, generating several sets of feature maps at different scales. SSD also uses different default bounding boxes (anchors) with different scales and shapes (aspect ratios) and then adjusts the bounding boxes as part of the prediction. In contrast with YOLO, SSD's default bounding boxes ($i.e.$ anchors) are independent of the dataset. In different feature maps, the scale of default anchors is computed with regularly space between the highest layer and the lowest layer.

Like YOLO, SSD always produces the same number of bounding boxes and performs a non-maximum suppression to yield the final predictions. In contrast with YOLO, SSD is designed to be independent of the underlying backbone. In the original SSD implementation, the VGG-16 [2] model was

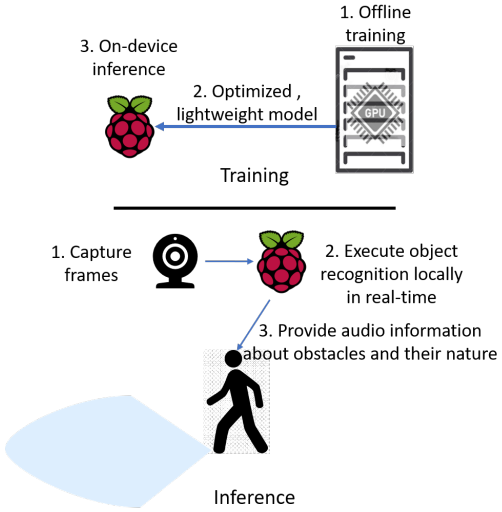

Fig. 1. Inference and training are decoupled.

Fig. 2. The proposed system takes the form of a camera harness incorporating a nanocomputer.

used as backbone. In this research, we use MobileNet [2], instead. MobileNet uses depth-wise separable convolutions to build lightweight networks, suitable for mobile and embedded systems. The association of MoblileNet and SSD for (a lightweight) object detection was suggested in [2] and called SSD Lite. Its main strength is that it uses depth wise separable convolutions for the SSD layers as well to speed up the head part of the network.

B. Evaluation of the accuracy of object detectors

Evaluating the accuracy of object detectors is non trivial because there are two distinct tasks to measure: (i) a classification task ($i.e.$ determining the nature of an object); and (ii) a regression task ($i.e.$ determining the bounding box of an object). The regression task is typically evaluated by the IoU, which measures the overlap between a predicted bounding box and the corresponding ground truth bounding box. A detection is considered to be a true positive only if the predicted class matches the actual class, and the predicted bounding box has an IoU greater than a predefined threshold with the ground truth bounding box. If one of the above two conditions is not satisfied, the detection is considered to be a false positive.

The most commonly used accuracy evaluation metric in object detection is the "Average Precision (AP)" [1], whose values range from 0 to 100 (higher values are better). For the COCO challenge [17], the AP score is averaged over 80 classes and 10 IoU thresholds, ranging from 0.5 to 0.95 with a step size of 0.05. The COCO AP is written as AP@[.50:.05:.95] or simply as AP, while the AP at fixed IoU such as IoU=0.5, is written as AP@0.5. Averaging over 10 IoU thresholds instead of considering a single IoU threshold of .50 (as in PASCAL VOC challenge) tends to reward models that are better at precise localization.

C. CNN-Based assistive devices for visually impaired people

Existing CNN-based assistive systems providing on-device inference, such as [11], often have very limited capabilities ($e.g.$, recognition limited to faces, stairs, etc.). On the other hand, existing systems providing full object recognition capabilities either use GPU-based computers, unsuitable for convenient mobility [12], [13], or use a mobile/embedded device, but rely on remote servers (cloud-based inference). Moving data for inference from the source to the cloud requires an active and fast internet connection, and raises latency, cost, privacy, and scalability issues [18]. As an example, real experiments showed that offloading a camera frame to a cloud server and executing an image classification task, takes from 350 ms to more than 600 ms end-to-end ($\approx$ 1.67 to $\approx$ 2.86 Frame Per Second) [18].

The most recent work on CNN-based assistive devices [14], opts for an on-device inference and uses YOLOv2-tiny for object recognition and MTCNN [19] for facial identification. The paper, however, does not investigate Speed/Accuracy trade-offs to identify the best-suited lightweight object detector. As shown in section III, YOLOv2-tiny is the least accurate tested model (figure 3), while not being the fastest one.

III. System Implementation and Performance Benchmarking

A. System overview

The architecture of our solution is depicted in 1. We implemented our system on a Raspberry Pi 4B having a 1.5 GHz 64-bit SoC quad core ARM Cortex-A72 (ARM v8) processor, 8 GB LPDDR4-3200 SDRAM and a 8 MP Raspberry Pi V2 camera. The system portability is achieved using a rechargeable battery. We found that this $\approx$ 80$ nanocomputer offers a good trade-off between cost, practicality, and computation power.

We implemented a Text-to-Speech (TTS) engine and a CNN-based lightweight object detector on our nanocomputer. As illustrated in figure 1, the camera continually captures images (frames) of the environment in front of the user. The captured frames are transmitted to the object detector module, which processes them and infers the nature and the localization of the detected objects. The text-to-speech module is then triggered to provide audio information about obstacles and their nature.

[1] In some challenges such as COCO there is no distinction between "Average Precision (AP)" and "mean Average Precision (mAP)"

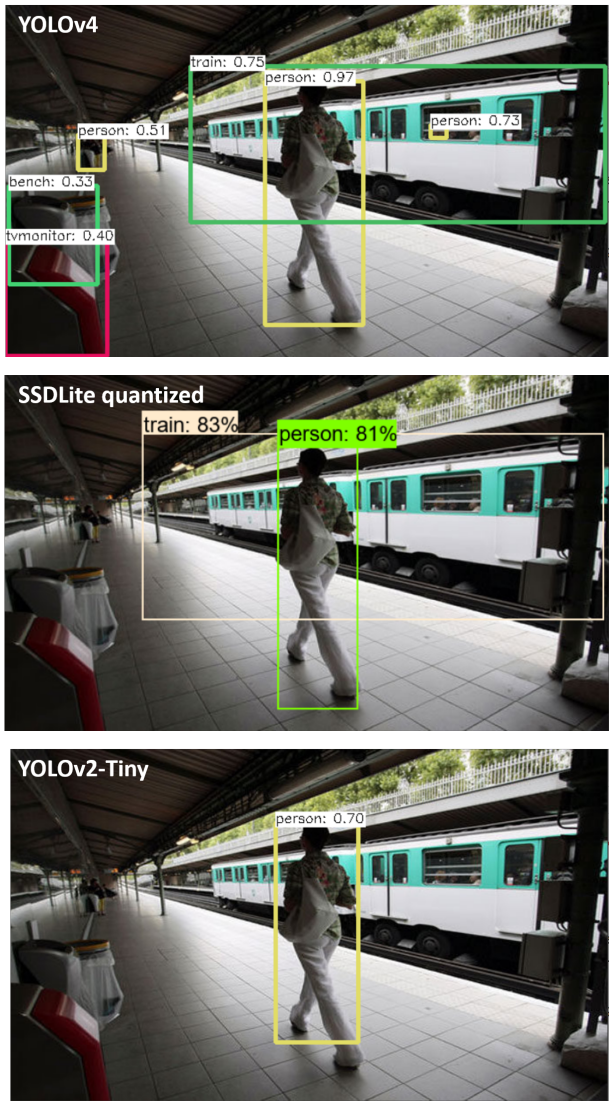

Fig. 3. Differences in detection between the most accurate tested model (YOLOv4), the least accurate tested model (YOLOv2-tiny) and one of the models that offers a good speed-accuracy trade-off (SSDLite MobileNetv2 quantized).

As mentioned before, in most existing assistive systems with full recognition capabilities, both the learning AND the inference processes are run by GPU-based computers/ servers. For each image captured by the device, we need to stream it to the server and back. Such an approach cannot satisfy strict end-to-end, low-latency requirements needed for real-time and interactive applications. In our work, the learning and the inference processes are decoupled. As the computational requirements of the learning process exceed by far the capacity of an embedded system, training is kept off-line. However, in contrast with most of existing systems, we transform the obtained off-line model into an optimized lightweight detector. The lightweight detector is then deployed on the device for a real-time on-device inference ($i.e.$, data locality). The flowchart of inference and training processes are shown in 1. As illustrated in figure 2, our system takes the form of a camera harness embedding a nanocomputer.

As lightweight detectors are inherently less accurate than regular ones, we implemented and conducted an extensive study on state-of-the-art fast detectors to strike the best trade-off between accuracy and inference speed.

B. System implementation

Our system was developed using python and OpenCV 4.5.1. The `cv2.VideoCapture()` module was used to capture the video stream from the Raspberry camera. The text-to-speech module was developed using pyttsx3 [20] which, in contrast with alternative libraries, works entirely offline.

We selected 6 YOLO models: YOLOv2, YOLOv3, YOLOv4 and their corresponding tiny variants. We also selected 3 SSD-MobileNet models and then generated the corresponding quantized variants. The AP@0.5:0.95 accuracy of the selected models ranges from 5.9% to 44.4% and the FPS (Frame per Second) from $\approx$ 0.17 to $\approx$ 11.39 (figure 5). Figure 3 illustrates the differences in detection between the most and the least accurate tested model and one of the tested models that offers a good speed-accuracy trade-off.

1) SSD Models: The evaluation of SSD models was done using both TensorFlow and TensorFlow Lite frameworks without any accelerator hardware, CPU overclocking or special tuning. Inferencing was carried out with SSD MobileNetv1, SSD MobileNetv2, and SSDLite MobileNetv2 models, all trained on the Common Objects in Context (COCO) dataset. Pretrained SSD models were downloaded from the TensorFlow detection model zoo [21]. Tensorflow Lite models were generated using the TFLiteConverter API, which allows to generate frozen graphs (tflite_graph.pb) and to convert frozen graphs to the Tensorflow Lite flatbuffer format (detect.tflite).

Model optimization techniques (called also neural network compression) aim to build models with reduced memory footprint, latency, size, and energy consumption, at the cost of the least possible accuracy degradation. Although optional for models running on GPU-based servers, model optimization techniques are mandatory for on-device models.

At the current stage of our work, we experimented the "post-training 8-bit quantization", which is an efficient compression technique that requires neither specialized hardware nor retraining. Quantization compresses a model by lowering the number of bits required to represent its parameters ($e.g.$ weights and activation outputs). In our quantized models (generated using TensorFlow Lite post-training quantization tools), 32-bit floating-point numbers are converted to the nearest 8 bit integers. As shown in table II, the obtained models are on average ≈ 4 times smaller than the corresponding baseline models.

2) YOLO Models: Choosing the best-suited inference framework is crucial for real-time object detection applications running on devices with limited resources. Native Darknet was designed to run on GPUs and is relatively slow on CPUs [22]. In our work, we used the OpenCV Deep Neural Network inference engine (OpenCV DNN) instead. In addition

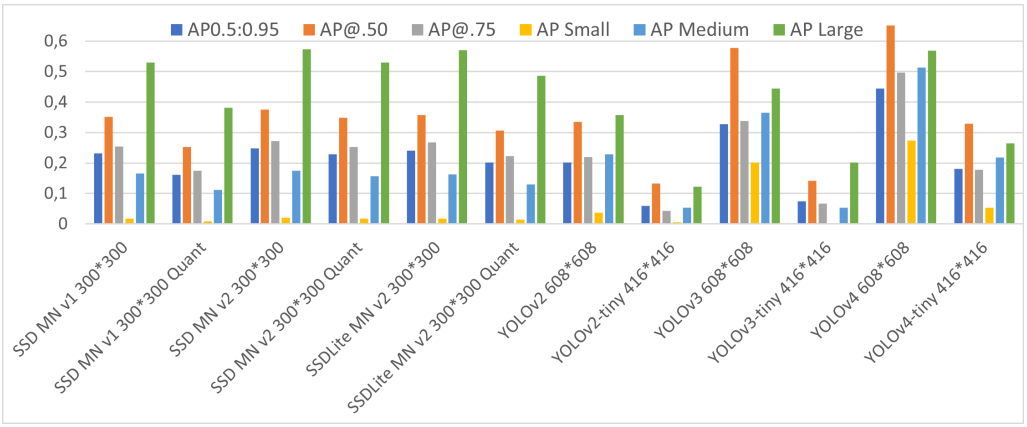

Fig. 4. Comparison of the accuracy of the studied object detectors.

TABLE II
SSD MODELS SIZE COMPARISON.

Model	TF model (frozen inference graph)	Generated TF Lite quantized model (detect.tflite)
SSD MobileNetv1	27.7 MB	6.59 MB
SSD MobileNetv2	66.4 MB	16.1 MB
SSDLite MobileNetv2	18.9 MB	4.43 MB

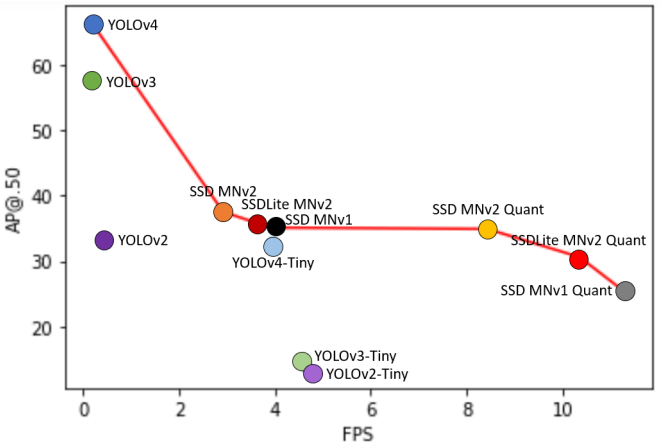

Fig. 5. Speed/accuracy tradeoffs (the Pareto frontier is plotted in red).

to its efficient C/C++ implementation, OpenCV DNN uses several optimizations to speed up inference with little or no loss in accuracy (*e.g.* layer fusion, pruning, etc.). YOLOv4 authors stated that OpenCV DNN is "the fastest inference implementation of YOLO on CPU devices" [23].

YOLO models, pretrained on MS COCO (graph and weights), were loaded in OpenCV using the `cv2.dnn.readNetFromDarknet()` function. The `cv2.dnn.NMSBoxes()` function was used to remove redundant overlapping bounding boxes. We set the DNN backend to OpenCV (`cv2.dnn.DNN_BACKEND_OPENCV`) and the target to CPU (`cv2.dnn.DNN_TARGET_CPU`). The `getUnconnectedOutLayers()` function was used to identify the output layer(s) of the network, which is essential to run the inference function `forward()`. We should notice that OpenCV's DNN module doesn't support quantized models at the time of writing this paper.

C. Experimental study

Investigating Speed/Accuracy trade-offs is of great interest for environments with limited resources [24]. In our work, we conducted two sets of experiments to evaluate the impact of model optimization on accuracy and on inference time.

YOLO and Tensorflow researchers released several easy-to-run pre-trained models and provided their respective AP and FPS [21], [25]. Despite these efforts, there are several reasons that make it difficult for practitioners to decide which model is best suited to their applications [24]. First, the claimed accuracy of YOLO and SSD models are obtained on different datasets (*e.g.* COCO 2014 minival set for SSD models and COCO test-dev set for YOLO models). Second, some important metrics are often omitted (*e.g.* the claimed AP of YOLOv3-tiny is an AP@.5, while the claimed AP of SSD models are AP@[.50:.05:.95]). Finally, except for some SSD models, all reported FPS are obtained on GPUs or using neural sticks, and no indication is given on inference time on CPUs.

To provide a fair comparison between regular and optimized models, we evaluate their accuracy over the COCO val2017 dataset (5k images) using the pycocotools API [17]. For inference speed benchmarking, to get close to the actual use conditions of our system, we considered 10 videos of 2 minutes each, recorded with our camera and involving different indoor and outdoor environments in normal stability/walking conditions (constant and natural walking speed). Reported FPS are average of three consecutive runs and are only based on model run time (*i.e.* `forward()`, `tf.session.run()` and `tf.lite.Interpreter.invoke()`). Preprocessing of an image is not taken into account, nor text-to-speech conversion. Figure 4 depicts the overall AP, the AP for large, medium and small objects on the COCO 2017 validation dataset. Figure 5 presents an exploration of the space of speed/accuracy trade-offs.

As shown in figure 5, the lightweight variants of YOLO are much faster than regular ones: in our experiments, YOLOv2-tiny, YOLOv3-tiny and YOLOv4-Tiny are respectively 10.46,

26.61 and 21.13 times faster than YOLOv2, YOLOv3 and YOLOv4. As shown in figure 4, this inference speed-up is accompanied by a significant drop in accuracy: YOLOv2-tiny, YOLO-V3-tiny and YOLOv4-tiny are respectively 3.42, 4.43 and 2.47 times less accurate than the corresponding regular models. Among the aforementioned tiny models, only YOLOv4-tiny manages to keep a decent overall AP.

In contrast with regular YOLO models, baseline SSD-mobilenet models are, in general, fast enough to run on a low-end device such as the Raspberry Pi 4 (figure5). As shown in figure 5, using Tensorflow Lite in conjunction with post-training quantization allows a substantial inferencing speed-up with a relatively small loss of accuracy. To give an order of magnitude, in our experiments Tensorflow Lite used in conjunction with post-training integer quantization, decreases inferencing time by a factor of 2.86 with an accuracy drop ranging from 3.9% to 7.1%. According to our experiments, SSD MobileNetv1 quantized and SSDLite MobileNetv2 quantized offer both good speed-accuracy trade-offs. The latter was retained as the object detector of our assistive device.

IV. Conclusion

This paper presents a CNN-based assistive device, intended for visually impaired people. Although our system uses off-the-shelf, low-cost materials, it achieves a real-time and accurate object recognition.

In contrast with most of existing CNN-based assistive devices, our system adopts an on-device inference approach. On-device inference offers a variety of benefits: lower inference time (by removing the need to stream data to the cloud and back); increased security and privacy; increased reliability (by removing the need for a fast and active internet connection); and reduced costs.

Running state-of-the-art object detectors on devices with constrained resources is challenging and often prohibitive due to their high computational requirements. The approach adopted in our work is to consider lightweight variants of existing models and apply compression techniques to reduce model size and speed up inferencing time. Besides, we performed an extensive experimental comparison of state-of-the-art lightweight object detectors to identify models and frameworks that achieve the best speed/accuracy trade-offs. We hope this will help choosing the appropriate model when deploying object detection on devices with limited resources.

As a part of our future work, we intend to exploit temporal information from videos to predict movements and avoid eventual collisions.

References

[1] World Health Organization, "Blindness and vision impairment," https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment, 2019, [Accessed: 2020-06-01].

[2] 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," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[3] J. Redmond, S. Divvala, R. Girshick, and A. Farhadi, "Unified, real-time object detection," *CoRR*, vol. abs/1505.06798, 2017.

[4] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525.

[5] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[9] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[10] S. Ravi, "Projectionnet: Learning efficient on-device deep networks using neural projections," *arxiv*, 2017. [Online]. Available: https://arxiv.org/pdf/1708.00630.pdf

[11] B. Lin, C. Lee, and P. Chiang, "Simple smartphone-based guiding system for visually impaired people," *Sensors*, vol. 17, no. 6, p. 1371, 2017. [Online]. Available: https://doi.org/10.3390/s17061371

[12] B. Mocanu, R. Tapu, and T. Zaharia, "Seeing without sight : An automatic cognition system dedicated to blind and visually impaired people," in *2017 IEEE International Conference on Computer Vision Workshops, ICCV Workshops 2017*, 2017, pp. 1452–1459.

[13] ——, "Design of a CNN face recognition system dedicated to blinds," in *IEEE International Conference on Consumer Electronics, ICCE 2019, Las Vegas, NV, USA, January 11-13, 2019*. IEEE, 2019, pp. 1–2. [Online]. Available: https://doi.org/10.1109/ICCE.2019.8661933

[14] F. Rahman, I. J. Ritun, N. Farhin, and J. Uddin, "An assistive model for visually impaired people using yolo and mtcnn," in *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy*, ser. ICCSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 225–230.

[15] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[16] C. Wang, H. M. Liao, Y. Wu, P. Chen, J. Hsieh, and I. Yeh, "Cspnet: A new backbone that can enhance learning capability of CNN," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. IEEE, 2020, pp. 1571–1580.

[17] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[18] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 35–48.

[19] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[20] pyttsx3, https://pypi.org/project/pyttsx3/, [Accessed: 2020-06-01].

[21] TensorFlow 1 Detection Model Zoo, https://github.com/tensorflow/models/, [Accessed: 2020-06-01].

[22] Y. Koo, S. Kim, and Y.-g. Ha, "Opencl-darknet: implementation and optimization of opencl-based deep learning object detection framework," *World Wide Web*, pp. 1–21, 02 2020.

[23] Alexey Bochkovskiy, https://github.com/AlexeyAB/darknet, [Accessed: 2021-01-01].

[24] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," *CoRR*, vol. abs/1505.06798, 2017.

[25] YOLO: Real-Time Object Detection, https://pjreddie.com/darknet/yolo/, [Accessed: 2020-06-01].