Can AI replace conventional markerless tracking? A comparative performance study for mobile augmented reality based on artificial intelligence.

Roberto Pierdicca[1], Flavio Tonetto[2] Marco Mameli[3], Riccardo Rosati[3], and Primo Zingaretti[3]

[1] Dipartimento di Ingegneria Civile Edile e dell'Architettura (DICEA), Università Politecnica delle Marche, Via Brecce Bianche, 60131, Ancona, Italy
{r.pierdicca}@staff.univpm.it,
[2] Dipartimento di Ingegneria dell'Informazione (DII), Università Politecnica delle Marche, Via Brecce Bianche, 60131, Ancona, Italy
{p.zingaretti}@staff.univpm.it {m.mameli, r.rosati}@pm.univpm.it
[3] Sinergia Consulenze Srl, Viale Goffredo Mameli 44, 61121, Pesaro, Italy
ftonetto@sinergia.it

Abstract. AR is struggling to achieve its maturity for the mass market. Indeed, there are still many challenging issues that are waiting to be discovered and improved in AR related fields. Artificial Intelligence seems the more promising solution to overcome these limitations; indeed, they can be combined to obtain unique and immersive experiences. Thus, in this work, we focus on integrating DL models into the pipeline of AR development. This paper describes an experiment performed as comparative study, to evaluate if classification and/or object detection can be used an alternative way to track objects in AR. In other words, we implemented a mobile application that is capable of exploiting AI based model for classification and object detection and, at the same time, project the results in AR environment. Several off-the-shelf devices have been used, in order to make the comparison consistent, and to provide the community with useful insights over the opportunity to integrate AI models in AR environment and to what extent this can be convenient or not. Performance tests have been made in terms of both memory consumption and processing time, as well as for Android and iOS based applications.

Keywords: Augmented Reality · Object Detection · Deep Learning · Tracking · AI4XR

1 Introduction

The increase of eXtended Reality (XR) applications is due to their use in almost every domain of day life [1]. Thanks to the technological advances, companies and research centres are investing and acting to spread XR applications, whilst

the end users are growing their interest given the irrefutable usefulness to facilitate daily tasks [37]. XR proved its benefit in education [13, 28], industry [27], cultural heritage [4, 26, 10], medicine [40], assistive technology in elderly care [25], prototyping [19] and many more.

Among them, AR is struggling to achieve its maturity for the mass market. Indeed, there are still many challenging issues that are waiting to be discovered and improved in AR related fields. One of the major difficulties is that there are several AR markers on the market, each with its own unique encoded information algorithm [23]. They usually require the users to modify their original material contents in some way, either partially or completely. Another problem is the marker identification process, which utilises the standard computer vision-based feature extraction approaches, such as scale-invariant feature transformations or histograms of oriented gradients [17], for classification tasks. These mathematical methods are vulnerable to unanticipated real-world lighting, marker orientation and unexpected noises [7]. Deep learning (DL) approaches that use Convolutional Neural Networks (CNNs) models are a useful tool to overcome the standard computer vision difficulties in the AR marker identification process as in [29, 11, 24]. The review presented in [12] highlights that the functionality of AR increases using DL based approaches. To facilitate the development of AR experiences by overcoming the above-mentioned limitations, we started from the following assumption: AR and AI are disruptive technologies that, albeit being two distinct technologies, can be combined to obtain unique and immersive experiences. Thus, in this work, we focus on integrating DL models into the pipeline of AR development.

Considering this due premise, to facilitate the reader understanding the main technologies used in our experiments, we performed a comparative study to evaluate if classification and/or object detection can be used an alternative way to track objects in AR. In other words, we implemented a mobile application that is capable of exploiting AI based model for classification and object detection and, at the same time, project the results in AR environment. Several off-the-shelf devices have been used, in order to make the comparison consistent, and to provide the community with useful insights over the opportunity to integrate AI models in AR environment and to what extent this can be convenient or not. Performance tests have been made in terms of both memory consumption and processing time, as well as for Android and iOS based applications.

The remainder of the paper is organised as follows. In section 2, the most recent paper that use deep learning approaches for mobile phone application. Section 3 describes the specific objectives, the problems encountered and the techniques adopted. The results obtained are described in section 4. Finally, section 5 describes the degree of achievement of the objectives and the ideas for future developments.

2 Related works

Several approaches have been proposed in literature that exploit DL to produce different application for mobile devices. In this section we present most recent application for smartphones based on deep learning for classifciation and object detection tasks in different sector. For example, in agricultural sector there are several applications for mobile devices able to classify and detect objects acquired by the camera and using deep learning method. The work of [36] proposes a mobile phone application based on Inceptionv3 [39] network that aims to support smallholder farmers in Tanzania to early detect banana diseases. Once the image of the leaf is acquired, the mobile application was able to detect in real time and with good performance two possible diseases. This study is important since allows to prevent serious damage to crops through a common and inexpensive tool, by avoiding major damage in poor regions. In the same context the work of [6] proposes a smartphone application based on a deep learning approach able to identify disease on tomatoes leaves. The authors implement an application based on MobilNet [9] model that is able to identify 10 common diseases of tomatoes. The choice of MobilNet depends on the fact that this model works on mobile phone with acceptable speed and perform high classification accuracy. The work of [22] aims to increase performances in the identification of diseases in the tomatoes leaves. They implemented a novel mobile phone application based on Convolutional Neural Network by performing automatic background subtraction for leaf, since this task enhances disease recognition accuracy. Once the user captures the image of the leaf, the segmentation app removes the background features so that only the target leaf remains. Also in the recent work of [3] an application for smartphone is presented, with the aim to prevent damage to crops. The authors do not only identify the pest but unlike all other works are able to also recognise the scale insects. A complex architecture consisted of Faster region-based convolutional networks (Faster R-CNNs) [32], single-shot multibox detectors (SSDs) [16], and You Only Look Once v4 (YOLO v4) [30], is implemented to identify and locate scale pests in the image acquired by the camera of the smartphone.

In the ambit of healthcare, the work that deals with the support tool for impaired and blind persons is presented by [21]. The authors proposes an application based on YOLO that detects and recognises the objects in the image acquired by the camera. After recognising the objects in the scene, a browser based voice library narrates to the person what's in front of them. Similar is the mobile application proposed by [34], that describes an android app that in real time helps visually impaired people in understanding their surrounding environment. As in the previous work, the camera of the smartphone captures the images, but a Tensorflow's object detection API detects the objects. Then the objects are converted into an audio output through android's text-to-speech library.

Following we present some application for smartphone that also implement an augmented reality method to add information to the reality. In food and beverages sector there are several applications for mobile devices that combine object

detection tasks and augmented reality contents. Interesting is the "HealthCAM" application proposed by [2] that combines augmented reality and machine learning techniques, in particular Mobilenet SSD model. From an image acquired by the phone's camera the app identifies a snack and in real time presents a visual warning on the camera that indicates the degree of sugar, sodium and fat of the snack. Similar is the work of [20] that proposes a mobile application that recognises fruits and vegetable and gives information, searched in the web, about nutritional property of the detected object. Unlike the previous approach, this application uses the object recognition model in Tensorflow API to be embedded in the devices. Another interesting paper is presented in [38], where AI is used to perform the the object detection task. The authors develop an iOS application, which combines AR with the object detection and recognition tasks. The purpose is to verify if the combination of these fields is possible with the modern tools available. The application must be an alternative option to traditional furniture assembly manuals.

The literature already briefly reviewed demonstrates the interest in blending AI models into AR applications; none of the existing papers, however, perform a comparative analysis over the real performances of such integration. Our work cover this gap.

3 Materials and methods

3.1 Classification task

The architecture used for the classification task is the MobileNetV2 [35]. It is a family of lightweight CNN architectures suitable for image classification within embedded device. Despite its relatively small architecture, what makes this model widely used is the much lower computing power required to perform real-time inferences. The number of parameters and weight storage consumption of the MobilenetV2 model compared to other state-of-the-art classification methods are reported in Table 1. Thereby, this aspect allowed this network to be established as the reference model for mobile devices, embedded systems, and more. MobileNet uses depthwise separable convolutions, significantly reducing the number of parameters compared to CNNs with regular convolutions with the same depth. A depthwise separable convolution is a factorised convolution constituted by two operations:

- depthwise convolution: this convolution is originated from the idea that a filter's depth and spatial dimension can be separated;
- pointwise convolution: convolution with a kernel size of $1x1$ that simply combines the features created by the depthwise convolution.

Essentially, depthwise separable convolutions splits kernel into two separate kernels, one for filtering (the depthwise one) and the other for combining (the pointwise one). Moreover, MobileNetV2 presents the addition of another two basic structures, which are the linear bottle-neck layer and the reverse residual

structure: these features contribute to speed up model convergence and prevent from vanishing gradient. The overall architecture is describe in [35]. In our case, we use MobileNetV2 trained on the well-known ImageNet dataset [5], that is an image database in which each node of the hierarchy is depicted by hundreds and thousands of images. The network takes as input a tensor with shape $[224, 224, 3]$ (corresponding to height, width and number of channel of the image respectively) and return as output a tensor with shape $[1, 1, 1000]$ with the probability that the image represents a specific class among the 1000 classes of the entire dataset.

Classification models	Parameter (Million)	Weight storage (MB)
DenseNet	8.1	33
VGG-16	138.3	528
AlexNet	60	220
InceptionV3	23.8	92
ResNet-101	44	171
MobileNetV2	3.5	14

Table 1: Number of parameters and weight storage of state-of-the-art classification models.

3.2 Object Detection task

The state-of-the-art object detection approaches include objectness detection (OD), salient object detection (SOD) and category-specific object detection (COD) [8]. The task the plugin aims to perform is the COD one, where the goal is to detect multiple predefined object categories from each given image, identifying at the same time the image regions that may contain the objects of interest but also the specific object category of each region. In particular, according to the literature ([41, 15]), there are generally two main categories of COD methods: region proposal-based or two-stage approaches (e.g.,Faster R-CNN [33]) and regression/classification-based or one-stage approaches (e.g. YOLO [30] and SSD [16]). The former first generate a set of proposal bounding-boxes by using region proposal methods and then pass the detected object proposals to the CNN classifiers; the latter work dividing the region proposals into categories at the moment of generation. One-stage detectors are preferred for embedded real-time application since they are highly efficient: the entire detection task runs in real-time with acceptable memory and storage demands, which is a key aspect for mobile/wearable devices that have limited computational capabilities and storage space. Table 2 shows speed (measured as frames per second [FPS]) and storage consumption comparison on COCO [14] test set. The reported computational cost comparison suggests that regression-based COD methods are much faster than region proposal-based methods, showing that one-stage approaches are the most promising COD direction for real-time application.

For this reason, the architecture we used for object detection task is the Tiny YOLOv2 [31], which belongs to the family of regression-based one stage detectors. The network is constituted by convolutional layers with a 3x3 kernel and max-pooling layers with a 2x2 kernel. The number of layers is reduced compared to the traditional YOLOv2 architecture: it presents only 9 convolutional layers and 6 pooling layers. We use Tiny YOLOv2 trained on UEC-Food100 benchmark dataset [18], which contains 100 categories of dishes. For this network, the input layer expects a tensor with shape $[3, 416, 416]$, while the output layer generates a $[525, 13, 13]$ tensor. The output divides the image into a 13x13 grid, where each cell in the grid consisting of 525 values. Each grid cell contains 5 potential object bounding boxes, and each bounding box is represented by the following 105 elements:

- x: the x position of the bounding box center relative to the grid cell it's associated with;
- y: the y position of the bounding box center relative to the grid cell it's associated with;
- w: the width of the bounding box;
- h: the height of the bounding box;
- o: the confidence value that an object exists within the bounding box, also known as "objectness score";
- $p1$-$p100$: class probabilities for each of the 100 classes predicted by the model for the considered dataset.

Object detection models	FPS	Weight storage (MB)
Faster R-CNN	4	168
Retinanet-101	11	228
SSD500	19	77
YOLOv1	45	190
Tiny YOLOv1	155	58
YOLOv2	40	202
Tiny YOLOv2	244	43
YOLOv3	35	237
Tiny YOLOv3	220	34

Table 2: Frame per second (FPS) performed on a Pascal Titan X and weight storage of principal state-of-the-art object detection models.

4 Experimental Processing

In this section, performance analysis of the devices listed in tables 3 and 4 are reported. This analysis focuses on the model execution performance of the two DL models Mobilenet v2 and TinyYolo v2. In this section we will analyse the

motivation for this analysis, the configurations and the motivations that led to the choice of the employed devices and finally an examples of the results of the execution of the models are showed in Fig. 1.

Model(Model Number)	Processor	GPU Execution Units
iPad Air 64GB (iPad13,1)	A14 Bionic	4 EU
iPhone 11 128GB (iPhone12,1)	A13 Bionic	4 EU
iPhone SE 2 256GB (iPhone12,8)	A13 Bionic	4 EU

Table 3: List of Apple devices used for the test execution. The GPU for the AX SOC series the GPU has the same name of the chip.

Model (Model Number)	Processor	GPU Model and Execution Unit (EU)
xiaomi Mi 10T (M2007J3SY)	Snapdragon 888	Adreno 660 - 2 EU
xiaomi RedMi Note 9 Pro	Snapdragon 750G	Adreno 619 - 2 EU
xiaomi RedMi Note 9T (M2007J22G)	Mediatek Dimensity 800U	Mali G57 - 3 EU
Samung Galaxy Note 20 (SM-N981B)	Exinos 990	Mali G77 - 11 EU
Motorola One Fusion+	Snapdragon 730	Adreno 618 - 2 EU

Table 4: List of Android devices from different manufactures, with different SOC configuration.

The used indicator for the tests are:

- **Model Processing Time** : the time needed by the physical devices to obtain the inference result from model;
- **System Memory Size** : the total amount of RAM of the devices;
- **Managed Total Memory** : The total amount of RAM used by the application during the execution;

4.1 Motivation for the choice of devices

For the Apple models the selected devices are representative of a range of devices currently available on the market, in fact at the time of writing the article it is possible to buy the iPhone SE 2, iPhone 11, iPhone 12 and iPhone 13 models, for the choice of the iPad model we based on a choice of popularity as the iPad Air 4th generation was among the best selling models. For this reason these devices have been selected in order to approximate the tests carried out on devices that are on the market and that are owned by a wide audience of people and possible users of final products made with the workflow presented here.

For the Android models, since there is a wide choice of possible configurations, the choice was made by considering, at present, the terminals of the best-selling manufacturers and interweaving this information with the 3 possible

SOC configurations that can be found on the market. In this case, find configurations based on Mediatek SOC in Italy is very complex, so we used only one device based on this SOC, for the Snapdragon configurations the choice fell on the top model at the (time of testing), ie the SOC Snapdragon 888, while the terminals with SOC Snapdragon 750G and 730 are terminals with performance and features comparable with the chip Mediatek Dimensity 800U and therefore have been included in the list of devices. In addition, a Samsung terminal from the Note 20 series based on the Exynos 990 chip has also been included, thus enabling a comparison with a different SOC, which from a conceptual point of view is similar to the Snapdragon 888 chips, differing in two aspects such as the manufacturing technology (7nm vs 5nm) and the GPU model. Tablets were not selected for the Android segment as they use identical SOCs to smartphones but have less RAM.

4.2 Test configuration and description

The aim of the tests carried out on these devices was to provide information on the inference times of the two models and the amount of memory used during execution. For this reason, two applications were created, using the framework presented, in order to simulate the real use of the framework. In particular, these applications, once started, loaded the model in memory and loaded ready-to-use images directly from the application assets. For each model, images were selected from the test dataset belonging to the dataset used for the training of the model itself. From each test dataset 4000 images were selected and loaded into the application, which simulated their acquisition by the device's camera. In this way, the tests were conducted in parallel on all the devices, connected to a power source to keep the battery charged throughout the duration of the test. Moreover, thanks to this configuration, the tests carried out used images of the same resolution for all the devices, which on the one hand undermines the specialisation of the test on the resolutions of the cameras inside the smartphones, but also allows us to understand how different hardware levels can affect the same amount of data to be analysed, providing more useful data for the purpose of the test, namely to understand the behaviour of the devices currently on the market for the use of neural network models within augmented reality. Precisely from this point of view, as previously mentioned, the studies carried out are based on the inference times of the two models and on the amount of memory used to obtain the results. In order to obtain the graphs that will be analysed in Section 4.4, a system of data collection from the terminals based on logging was used; in fact, at the end of the model inference, the collected data are sent to a collection server. The data collected for each inference are: inference time, memory occupied, smartphone model, SOC model, operating system version.

4.3 Test Application execution

In addition to the model performance tests, the execution test was carried out on mobile devices, the results of which are shown in Fig. 1 . In particular,

in Fig. 1a there is the execution of the YOLO network trained on the food dataset, the result displays in the top left corner the class of the recognised object and the bounding box around the object projected in reality through the use of Unity's ARFoundation. Fig. 1b shows the result of the execution of the MobileNet network trained on ImageNet, the model correctly recognizes the visualized object, a mouse, and projects in reality the name of the same for the visualization in the centre of the object.

(a) (b)

Fig. 1: Example of classification running.

Following, the performance graphs are presented and analyzed based on the inference times and the memory used by the app during the processing phase.

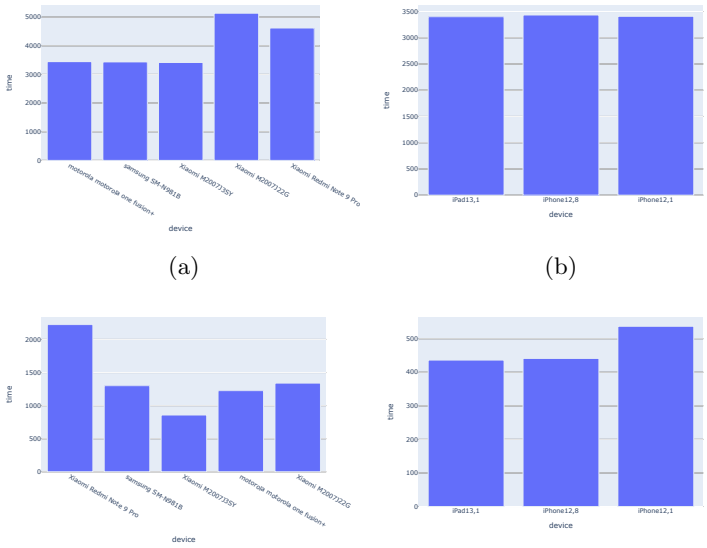
4.4 Graph analysis

The behaviour of the various terminals with the various models are analysed. The analysis is be performed separately based on the software platform.

The inference time is a parameter of fundamental importance for a neural network model and it becomes even more important considering the purpose of using these models in the context of AR applications. In fact, when using a smartphone for augmented reality, we expect to obtain a visual response in a few tenths of a second for the application to be engaging and usable by the user and the user does not decide to abandon its use. Considering that using neural networks within an app, not only do we have to wait for the calculation times necessary for the network to return a result, but there are also times necessary for the pre-processing of the data and for the post-processing of the results for the visualisation. Assuming constant pre-processing and post-processing times, a hypothesis that is possible and plausible thanks to software optimisation and today's SOCs, which for these tasks have performances that differ by hundreds of thousandths of a second between the various categories under examination, it can be stated that the latency time that has the greatest impact on usability is precisely the network inference time.

Fig. 2: Examples of detection and labelling running.

On the devices available for testing, considering the average latency time over the whole test and excluding possible outliers outside the 99% quartile, we



(a) (b)

(c) (d)

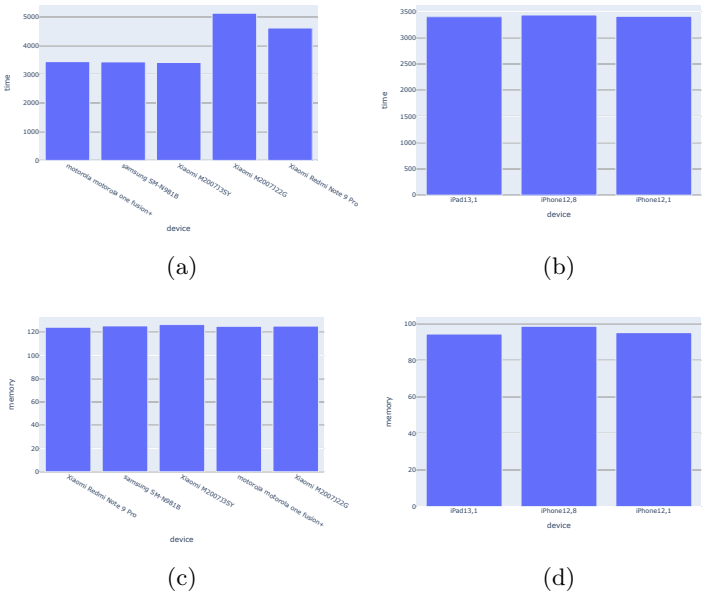
Fig. 3: Graphs of the average inference time of the models under analysis. Fig. 3a and Fig. 3b show the average execution time of MobileNet for Android and iOS respectively. Fig. 3c and Fig. 3d show the average execution time of Tiny-YOLO for Android and iOS respectively. For the name of the smartphone models see Tab.4 and Tab.3

obtained the average times in Fig. 3 for both software platforms most used in mobile today.

Observing Fig. 3a and Fig. 3c inherent to the Android system, it can be seen that MobileNet requires on average more time for inference than Tiny-YOLO.

Observing the network architectures, it can be seen that MobileNet has skip-connections which require more computational time than a model with only consecutive connections and no skip-connections such as YOLO, and it is for this reason that MobileNet, despite being a model with fewer parameters (as demonstrated also by the amount of memory occupied Fig.4) requires more computation time than YOLO.

Continuing on the graphs of Android devices we can see how the model of the SOC can influence, in fact considering the smartphones Xiaomi Mi 10T and Samsung Galaxy Note 20 that contain the top of the range SOCs in the execution of the MobileNet model present the best time. The Xiaomi Mi 10T, and consequently the Snapdragon 888 SOC, are also confirmed to be the best for running the YOLO model. It is precisely the execution of the YOLO model that allows us to understand how the configuration of the SOC also influences this, as the two top-of-the-range models (Xiaomi and Samsung) use SOCs that



(a) (b)

(c) (d)

Fig. 4: In Fig. 4a and Fig. 4b there is the average memory usage during the execution of MobileNet for Android and iOS respectively. Fig. 4c and Fig. 4d show the average memory usage while running Tiny-YOLO for Android and iOS respectively. For the name of the smartphone models see Tab.4 and Tab.3

have similar architectures but differ in terms of resource management and GPU, and this with models that are more robust in terms of the number of parameters (such as the tiny-YOLO architecture) leads to a difference in inference times. Furthermore, focusing on mid-range smartphones shows that it is no longer just the SOC that makes a difference but also the software customisations. These can affect runtimes so much so that the Motorola One Fusion+ with pure Android and no customisations, which has a lower capacity SOC, runs faster than the Xiaomi Redmi Note 9T, a difference that is even more noticeable in the graph in Fig. 3c compared to the MobileNet graph in Fig. 3a.

Considering now iOS terminals, with iPhone and iPad, and analysing the MobileNet graph in Fig. 3b the execution does not show substantial differences in execution time. While the execution of the tiny-YOLO model of iPhone 11 requires a longer time to perform the inference, this difference with respect to iPhone SE 2 is due to the need of iPhone 11 to manage a terminal with a more advanced screen and sensors (which, even if the smartphone is not in use, generate data and require processing) and this can cause delays in the inference of the model, as visible in Fig.3d.

In the case of the memory graphs, Fig 4, it can be seen that for both software platforms there is a constant use of the amount of memory, although in different

amounts. As a matter of fact, in the case of MobileNet Fig 4a and Fig 4b we can see that the amount of memory is less than the amount required by the tiny-YOLO model Fig 4c and Fig 4d . In addition to this by comparing the amount required per platform it can be seen that iOS based devices require less memory to load the model. This less exorbitant request is due to the software optimization in the compilation phase realized through the request and the obligation to use X-Code to obtain the apps for iOS, which applies software optimizations on the computational side that allow to obtain both lower inference times and lower memory usage and make the application realized using these models allow a realtime use unlike android devices that, despite being realtime, have higher latency times that in some cases could result in a frustration of use by the end user.

This analysis is also confirmed by the graphs in Fig. 5 and Fig. 6. In fact, from them we can see that on iOS, despite the presence of three peaks for the amount of memory used, its variation is narrow in terms of absolute values for the MobileNet (in Fig. 5a and Fig. 5d we considered iPhone 12.8). While in the case of Android terminals we find a high variation of memory usage for both terminals. Continuing on the memory analysis for the YOLO architecture, Fig. 6a Fig.6b Fig. 6c, we see that the memory occupied is always lower than the Android terminals but this time, although the amounts used are lower on iOS, the range of variation is similar for all three terminals under analysis. Analysing again the execution times, the previous hypotheses are also confirmed by the way the execution time peaks are presented: on iOS, even though YOLO values are outside the central peak, they define very narrow ranges, confirming that the optimisation applied by X-Code affects the execution of the models. While on Android terminals it is immediately noticeable that the SOC 888 presents narrow peaks even if the high complexity of the MobileNet (the skip connections) cause the presence of a wider range than on iOS. While the SOC 750G in the case of MobileNet presents a bell with a wide variance that makes us immediately understand how its performance is strongly influenced by the presence of a more complex architecture in terms of connections. While in the case of YOLO, a linear model in terms of connections, it also presents a narrow peak on higher execution values, although there are some values that, even if they are out of scale, suggest that this SOC has difficulties to manage neural models in conjunction with the use of augmented reality.

5 Conclusions and future works

In this study, a comparative analysis for the integration of AI-models into AR environments was performed. Two test applications have been developed and the execution time and memory usage data have been analysed, demonstrating the feasibility and possibility of integrating and using neural network models in mobile applications for augmented reality. The applications have been developed for both classification and object detection tasks with state of the art neural networks, specifically integrated to be run on mobile devices. Given the statistics

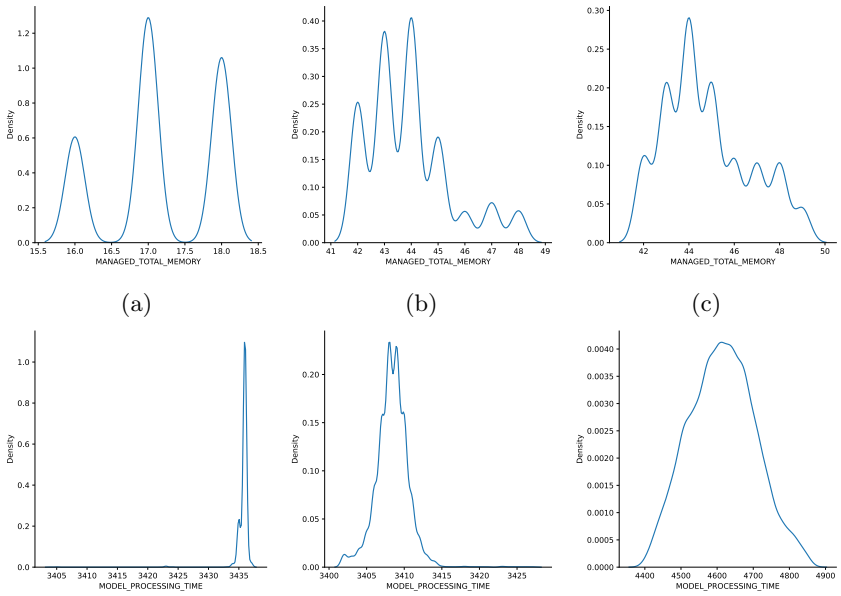


Fig. 5: The distribution plot of three terminal, two from the Tab 4 and one from the Tab. 3 for the MobileNet architecture. In Fig 5a and Fig. 5d was depicted the distribution, respectively, of the memory usage and processing time for the execution of the MobileNet on the iPhone 12,8. In Fig.5b and Fig. 5e the memory and the processing time for the Xiaomi Mi 10T. In Fig. 5c and Fig. 5f the memory and processing time distribution for the Xiaomi RedMi Note 9 Pro

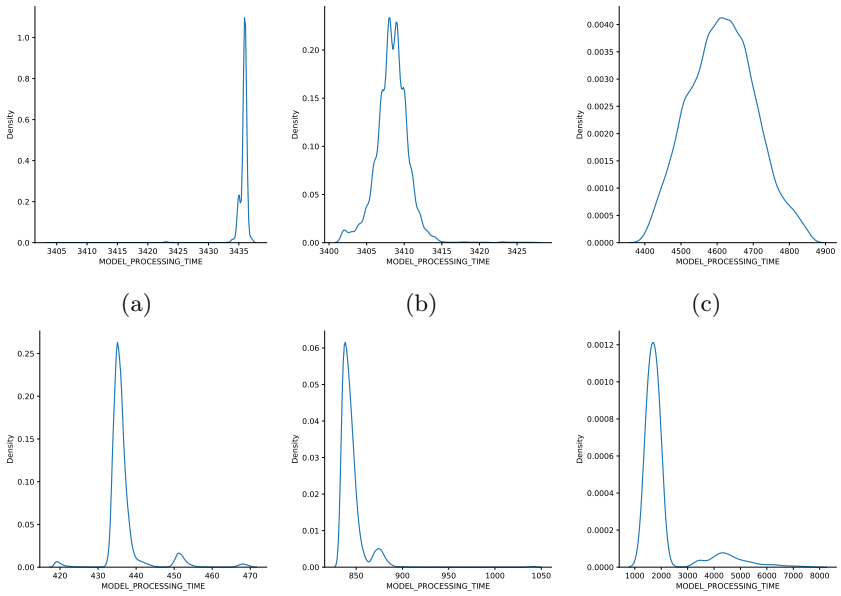


Fig. 6: The distribution plot of three terminal, two from the Tab 4 and one from the Tab. 3 for the YOLO architecture. In Fig 5a and Fig. 5d was depicted the distribution, respectively, of the memory usage and processing time for the execution of the YOLO on the iPhone 12,8. In Fig.5b and Fig. 5e the memory and the processing time for the Xiaomi Mi 10T. In Fig. 5c and Fig. 5f the memory and processing time distribution for the Xiaomi RedMi Note 9 Pro

here presented, we can state that the models run in real time even in medium-low level devices, demonstrating that the integration is in its maturity. Several limitations still exixts, and our future work will push on the following directions. First of all, performances are based on the automatic management by the SoCs, which manage the memory occupation between GPU and CPU. By developing a native solution, it will be possible to integrate AI models that better fits with the devices' architectures.

More in general, our group is working on the release of a dedicate framework, integrated in Unity, that will enable the user to integrate AI models directly with Barracuda, overcoming all the limitations described in this work. The corpus of the main idea, the source code end demos can be found at (https://github.com/SinergiaGit/DeepReality) and (https://assetstore.unity.com/packages/tools/integration/deepreality-201076).

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme through the XR4ALL project with grant agreement No 825545.

References

1. Bekele, M.K., Pierdicca, R., Frontoni, E., Malinverni, E.S., Gain, J.: A survey of augmented, virtual, and mixed reality for cultural heritage. Journal on Computing and Cultural Heritage (JOCCH) **11**(2), 1–36 (2018)
2. Cantillo, D., Cervantes, B., Cardona, J.: Healthcam: Machine learning models on mobile devices for unhealthy packaged food detection and classification. In: 2020 IEEE International Conference on E-health Networking, Application & Services (HEALTHCOM). pp. 1–6. IEEE (2021)
3. Chen, J.W., Lin, W.J., Cheng, H.J., Hung, C.L., Lin, C.Y., Chen, S.P.: A smartphone-based application for scale pest detection using multiple-object detection methods. Electronics **10**(4), 372 (2021)
4. Clini, P., Frontoni, E., Quattrini, R., Pierdicca, R.: Augmented reality experience: From high-resolution acquisition to real time augmented contents. Advances in Multimedia **2014** (2014)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
6. Elhassouny, A., Smarandache, F.: Smart mobile application to recognize tomato leaf diseases using convolutional neural networks. In: 2019 International Conference of Computer Science and Renewable Energies (ICCSRE). pp. 1–4. IEEE (2019)
7. Gammeter, S., Gassmann, A., Bossard, L., Quack, T., Van Gool, L.: Server-side object recognition and client-side object tracking for mobile augmented reality. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops. pp. 1–8. IEEE (2010)
8. Han, J., Zhang, D., Cheng, G., Liu, N., Xu, D.: Advanced deep-learning techniques for salient and category-specific object detection: a survey. IEEE Signal Processing Magazine **35**(1), 84–100 (2018)

9. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
10. Khan, M.A., Israr, S., Almogren, A.S., Din, I.U., Almogren, A., Rodrigues, J.J.: Using augmented reality and deep learning to enhance taxila museum experience. Journal of Real-Time Image Processing **18**(2), 321–332 (2021)
11. Lalonde, J.F.: Deep learning for augmented reality. In: 2018 17th Workshop on Information Optics (WIO). pp. 1–3. IEEE (2018)
12. Lampropoulos, G., Keramopoulos, E., Diamantaras, K.: Enhancing the functionality of augmented reality using deep learning, semantic web and knowledge graphs: A review. Visual Informatics **4**(1), 32–42 (2020)
13. Lin, P.H., Chen, S.Y.: Design and evaluation of a deep learning recommendation based augmented reality system for teaching programming and computational thinking. IEEE Access **8**, 45689–45699 (2020)
14. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
15. Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., Pietikäinen, M.: Deep learning for generic object detection: A survey. International journal of computer vision **128**(2), 261–318 (2020)
16. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision. pp. 21–37. Springer (2016)
17. Lowe, D.G.: Object recognition from local scale-invariant features. In: Proceedings of the seventh IEEE international conference on computer vision. vol. 2, pp. 1150–1157. Ieee (1999)
18. Matsuda, Y., Hoashi, H., Yanai, K.: Recognition of multiple-food images by detecting candidate regions. In: 2012 IEEE International Conference on Multimedia and Expo. pp. 25–30. IEEE (2012)
19. Monteiro, P., Gonçalves, G., Coelho, H., Melo, M., Bessa, M.: Hands-free interaction in immersive virtual reality: A systematic review. IEEE Trans. Vis. Comput. Graph. **27**(5), 2702–2713 (2021)
20. Muñoz Bocanegra, R., et al.: Aprendizaje profundo en dispositivo portable para el reconocimiento de frutas y verduras (2019)
21. Nasreen, J., Arif, W., Shaikh, A.A., Muhammad, Y., Abdullah, M.: Object detection and narrator for visually impaired people. In: 2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS). pp. 1–4. IEEE (2019)
22. Ngugi, L.C., Abdelwahab, M., Abo-Zahhad, M.: Tomato leaf segmentation algorithms for mobile phone applications using deep learning. Computers and Electronics in Agriculture **178**, 105788 (2020)
23. Nguyen, M., Tran, H., Le, H., Yan, W.Q.: A tile based colour picture with hidden qr code for augmented reality and beyond. In: Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology. pp. 1–4 (2017)
24. Park, K.B., Kim, M., Choi, S.H., Lee, J.Y.: Deep learning-based smart task assistance in wearable augmented reality. Robotics and Computer-Integrated Manufacturing **63**, 101887 (2020)
25. Park, Y.J., Ro, H., Lee, N.K., Han, T.D.: Deep-care: Projection-based home care augmented reality system with deep learning for elderly. Applied Sciences **9**(18), 3897 (2019)

26. Pescarin, S.: Digital heritage into practice. SCIRES-IT-SCIentific RESearch and Information Technology **6**(1), 1–4 (2016)
27. Pierdicca, R., Frontoni, E., Pollini, R., Trani, M., Verdini, L.: The use of augmented reality glasses for the application in industry 4.0. In: International Conference on Augmented Reality, Virtual Reality and Computer Graphics. pp. 389–401. Springer (2017)
28. Puggioni, M., Frontoni, E., Paolanti, M., Pierdicca, R.: Scoolar: An educational platform to improve students' learning through virtual reality. IEEE Access **9**, 21059–21070 (2021)
29. Rao, J., Qiao, Y., Ren, F., Wang, J., Du, Q.: A mobile outdoor augmented reality method combining deep learning object detection and spatial relationships for geovisualization. Sensors **17**(9), 1951 (2017)
30. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
31. Redmon, J., Farhadi, A.: Yolo9000: better, faster, stronger. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7263–7271 (2017)
32. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems **28** (2015)
33. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: towards real-time object detection with region proposal networks. IEEE transactions on pattern analysis and machine intelligence **39**(6), 1137–1149 (2016)
34. Salunkhe, A., Raut, M., Santra, S., Bhagwat, S.: Android-based object recognition application for visually impaired. In: ITM Web of Conferences. vol. 40, p. 03001. EDP Sciences (2021)
35. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018)
36. Sanga, S., Mero, V., Machuve, D., Mwanganda, D.: Mobile-based deep learning models for banana diseases detection. arXiv preprint arXiv:2004.03718 (2020)
37. Sereno, M., Wang, X., Besançon, L., McGuffin, M.J., Isenberg, T.: Collaborative work in augmented reality: A survey. IEEE Transactions on Visualization and Computer Graphics (2020)
38. Svensson, J., Atles, J.: Object detection in augmented reality. Master's Theses in Mathematical Sciences (2018)
39. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
40. Tanzi, L., Piazzolla, P., Porpiglia, F., Vezzetti, E.: Real-time deep learning semantic segmentation during intra-operative surgery for 3d augmented reality assistance. International Journal of Computer Assisted Radiology and Surgery **16**(9), 1435–1445 (2021)
41. Zhao, Z.Q., Zheng, P., Xu, S.t., Wu, X.: Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems **30**(11), 3212–3232 (2019)