Automatic content generation based on Deep Learning within Unity plugin

Cappanera Simone, Pasqualini Michele, Mameli Marco, Rosati Riccardo, Frontoni Emanuele

***Abstract*—Thanks to the computational improvements of mobile devices, the quality of the augmented reality applications has also improved, above all because each software has integrated libraries for the creation of AR (augmented reality) applications. Starting from these libraries, Unity has created a middleware that allows developers to create applications for the two main mobile platforms to which they can add the library that allows them to use deep learning models to obtain more attractive applications.**

The integration of deep learning into augmented reality can indeed provide a lot of real-time, interactive, user-centered and user-friendly intelligent applications which focus on enhancing the quality of experience and quality of services for end-users. Deep learning technology aims at instilling intelligence into systems with a view to enhancing and improving their efficiency and effectiveness. Object detection, image processing and computer vision are three of the main fields in which it is used.

The XR4ALL project intervenes in this context, trying to merge both approaches to combine the world of augmented reality with the world of deep learning. In particular, in this project, two central deep learning tasks were completed within an augmented reality application, built in Unity and based on Barracuda. The first task covered is object detection, completed through the use of the neural networks Tiny YOLO v2 and Tiny YOLO v3, while the second task is classification, achieved thanks to the MobileNetV1 and MobileNetV2 networks.

I. Introduction

The use of augmented reality in the industrial context still faces some challenges for intuitive, ergonomic and precise experiences. Enriching the real user experience via AR, can be done at different levels: head up display, face enhancement systems such as Snapchat filters, identification and analysis of the precise area in the image with immediate feedback and spatialized embedding of virtual objects in the real world. This research uses AR system based on Convolutional Neural Networks for scene recognition.

Indeed, Convolutional Neural Networks are in the state of the art for the solution of problems of classification, segmentation and object detection within images or videos. Object detection is the task of detecting instances of objects of a certain class within an image. The state-of-the-art methods can be categorized into two main types: one-stage methods and two stage-methods. One-stage methods prioritize inference speed, and example models include YOLO, SSD and RetinaNet. Two-stage methods prioritize detection accuracy, and example models include Faster R-CNN, Mask R-CNN and Cascade R-CNN. In this case, the Tiny YOLOv2 and Tiny YOLOv3 have been tested in smartphones to detect object in images. The YOLO networks are based on the idea of segmenting an image into smaller images. The image is split into a square grid of dimensions SxS. For each cells, YOLO calculates bounding boxes.

Dipartimento di Ingegneria dell'Informazione (DII), Università Politecnica delle Marche, Via Brecce Bianche 12, 60131, Ancona (Italy), m.mameli@univpm.it, r.rosati@univpm.it, e.frontoni@univpm.it, s1102490@studenti.univpm.it, s1101226@studenti.univpm.it

Fig. 1: An example of augmented reality

The goal of image classification is to classify the image by assigning it to a specific label. Typically, image classification refers to images in which only one object appears and is analyzed. In this classification task has been used MobileNetV1 and MobileNetV2 neural networks.

These tasks have been implemented in the Unity framework thanks to the Barracuda package. Unity is a cross-platform game engine developed by Unity Technologies, that has been gradually extended to support a variety of desktop, mobile, console and virtual reality platforms, while the Barracuda package is a lightweight cross-platform neural network inference library for Unity, that can run neural networks on both the GPU and CPU. The novelty of this work consists precisely in integrating a deep learning model in Unity plugin in order to have an AR framework for smartphones that is highly and easily customizable.

This article is structured as follows: some related works are presented in section II. Section III discusses the methods used to complete the different tasks and the related network architectures. Section IV presents all the results obtained and the related discussions. The conclusions are presented in Section V.

II. Related Works

After the short introduction, we present and analyze related applications and systems which combine AR with deep learn-

ing.

Akgul et al. [1] carried out a study in order to tackle some AR detection problems by modifying current deep learning architectures. They reviewed similar state-of-the-art detection methods for AR tracking. Furthermore, they introduced and presented their deep CNN detector called DeepAR which was based on the well-known CNN architecture AlexNet [2] and followed feature-based detection approaches.

Limmer et al. [3] proposed and presented a deep multi-scale CNN based approach capable of predicting and localizing the road course at night using camera sensors leveraging deep learning techniques. For their dataset, they used 7095 full-scene-labeled near infrared (NIR) images showing road scenes in various weather conditions, seasons and landscapes during night. This approach showed somewhat worse performance in precise conditions compared to other algorithms.

Połap et al. [4] focused on interpreting a reality-based real-time environment evaluation in order to help users be alert of and detect impending obstacles. Their approach put emphasis on analyzing the surrounding environment through diverse sensors to prevent collisions with oncoming objects. Their hybrid architecture uses AR techniques, deep learning algorithms and a dedicated method of data extraction from the samples obtained in real time to detect and estimate as much of the incoming information as possible.

Lin et al. [5] conducted a study in which they proposed and showcased a campus tour application utilizing advanced AR technologies, such as computer vision and object recognition. Furthermore, they used deep learning to improve the object recognition ability and positioning techniques so as to reduce search delays. They quoted that their application achieved attraction recognition accuracy rate of 90% and as such, the attractions can be easily identified by their appearance and characteristics. However, they also stated that the use of three-dimensional (3D) models with CNN was significantly delayed when running on mobile devices.

This project overcomes these problems by using less expensive architectures from the computational point of view that make it highly suitable for mobile devices.

III. Architectures and Methods

The section analyzes the architectures and methodologies used for the study of the problem. Section A presents the architecture used for the object detection, while section B describes the architecture used for image classification. Finally, Section C describes the method used to integrate a deep learning model into Unity.

A. Object Detection

The architectures used for object detection are Tiny YOLOv2 and Tiny YOLOv3. We used this type of architectures because we have to implement a Unity application which runs on smartphones, therefore with a reduced hardware capacity.

In this project these types of architectures have been used because the complexity of the architecture is reduced. They have a reduced number of convolutional layers compared to the classic YOLOv2 and YOLOv3 architectures. Also, the Barracuda package does not support all types of layers, only some operators and parameters are allowed.

Fig. 2: AR object detection with Tiny YOLOv3

Tiny YOLOv2 is trained on FOOD100 dataset. Ideally, it can detect 100 categories of dishes. Instead, Tiny YOLOv3 is trained on COCO dataset that is a large-scale object detection, segmentation, and captioning dataset. The data structures used to describe the inputs and outputs of the model are known as tensors. Tensors can be thought of as containers that store data in N-dimensions. In the case of Tiny YOLOv2, the input layer expects a tensor of dimensions 3 x 416 x 416, while the output layer generates an output tensor of dimensions 525 x 13 x 13. So, the model takes an image 3(RGB) x 416px x 416px as input and passes it through the different layers to produce an output. The output divides the input image into a 13 x 13 grid, with each cell in the grid consisting of 525 values. Each grid cell contains 5 potential object bounding boxes. A bounding box has 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.

The output generated by the pre-trained model is a float array of length 88725, representing the elements of a tensor with dimensions 525 x 13 x 13. The same thing happens in Tiny YOLOv3, but unlike Tiny YOLOv2, in this case the output layers generate two output tensors.

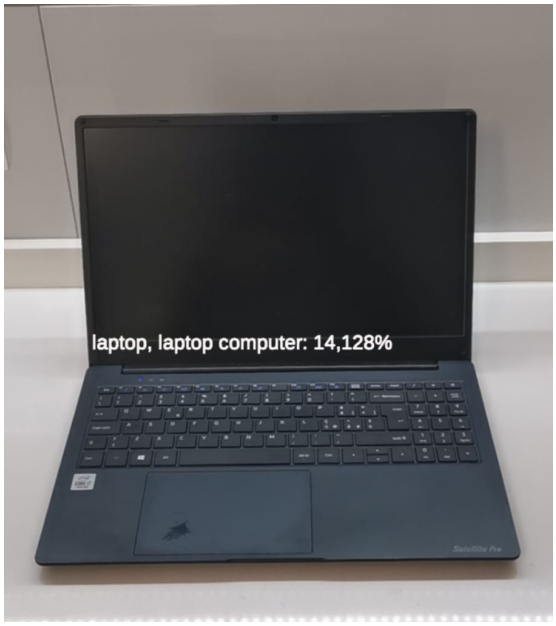


Fig. 3: AR image classification with MobileNetV1

B. Image Classification

The architectures used for image classification are MobileNetV1 and MobileNetV2. MobileNet is a CNN architecture model for image classification and mobile vision. There are other models as well, but what makes MobileNet special is the much lower computing power to run or apply transfer learning. This makes it perfect for mobile devices, embedded systems, and more. MobileNet uses depthwise separable convolutions. It significantly reduces the number of parameters compared to the network with regular convolutions with the same depth in the nets. This results in lightweight deep neural networks. A depthwise separable convolution is made from two operations:

- Depthwise separable 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.

Depthwise separable convolution is a factorized convolution that factorizes standard convolution into a depthwise convolution and a 1*1 convolution called pointwise convolution. Depthwise separable convolutions splits kernel into two separate kernels for filtering and combining. Depthwise convolution is used for filterning whereas pointwise convolution is used for combining [6]. MobileNetV1 and MobileNetV2 are trained on ImageNet dataset that is an image database in which each node of the hierarchy is depicted by hundreds and thousands of images. These networks take a tensor of dimensions 224 x 224 x 3 as input and return a tensor 1 x 1 x 1001 as output representing one of the 1000 classes of the dataset with the probability that the image represents that class.

Network	Task	Dataset	Classes
Tiny YOLOv2	Object detection	FOOD100	100
Tiny YOLOv3	Object detection	COCO	80
MobileNetV1	Classification	ImageNet	1000
MobiletNetV2	Classification	ImageNet	1000

TABLE I: Table of architectures

C. How to integrate a deep learning model into Unity

There are several steps to integrate a deep learning model into Unity. First, the architecture of the neural network and the dataset on which the model will be trained are chosen. Afterwards, the model is trained in PyTorch or TensorFlow. At this point, the model is converted to ONNX format.The Open Neural Network Exchange (ONNX) is an open source format for AI (Artificial Intelligence) models. ONNX supports interoperability between frameworks. This means you can train a model in one of the many popular machine learning frameworks like PyTorch, convert it into ONNX format and consume the ONNX model in a different framework like ML.NET. Image 4 shows the workflow to export a deep learning model trained in TensorFlow in Onnx format. The last step is to insert the model in ONNX format in the Unity environment, making sure it is compatible with the latest version of the Barracuda package installed.

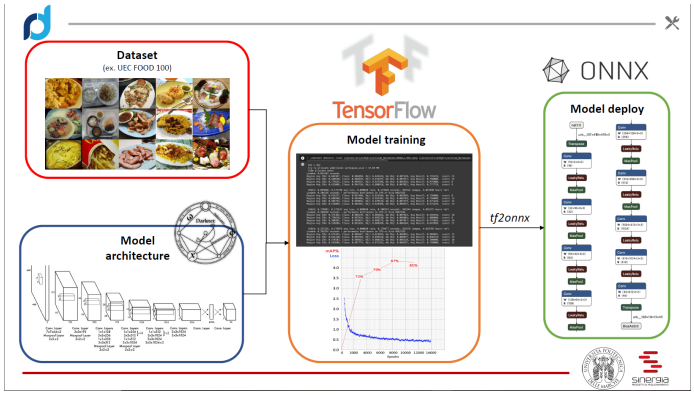


Fig. 4: Workflow image

IV. Results and Discussion

For the object detection task, the Tiny YOLOv2 architecture detects objects with high accuracy. It detects different types of dishes and foods. Tiny YOLO works on the same principles as YOLO but with a reduced number of parameters. It only has 9 convolutional layers, compared to YOLO's 24. This allow YOLO to be used inside mobile devices for augmented reality applications. The problem of this architecture lies in the fact that on low-performance mobile devices the application is not very fluid and there are often lag issues. It was thought that Tiny YOLOv3 could overcome this fps problem thanks to its much lower number of network parameters than tiny YOLOv2, but it presents the same lag issues on low-performance mobile devices.

Instead, on the other hand, in the image classification, we have taken the ONNX model of a MobileNetV1 already

trained on Imagenet. But, with this architecture, the application runs smoothly and without lag issues even on low-performance mobile devices. The same considerations also apply to MobileNetV2. Some details are shown in the table II:

Network	Input size	Parameters	Volume
Tiny YOLOv2	416 x 416 x 3	17.1 M	43 MB
Tiny YOLOv3	416 x 416 x 3	8.7 M	33.8 MB
MobileNetV1	224 x 224 x 3	4.2 M	16.1 MB
MobiletNetV2	224 x 224 x 3	3.4 M	13.3 MB

TABLE II: Table of parameters

The hardware used during the development included computer for running software, and smartphones. Moreover, with the aim of testing usability and performance of execution a sample of mobile devices has been used (a device for each range of price/processor). The tables III show the statistics relating to the various smartphones used during the testing of the application.

Model	Xiaomi Mi 10T
Processor	Snapdragon 888-series
Performance	High
GPU	Adreno (TM) 650
Graphics Memory Size	2048 Byte

Model	Xiaomi Redmi Note 9 Pro
Processor	Snapdragon 7xx series
Performance	Mid
GPU	Adreno (TM) 618
Graphics Memory Size	2048 Byte

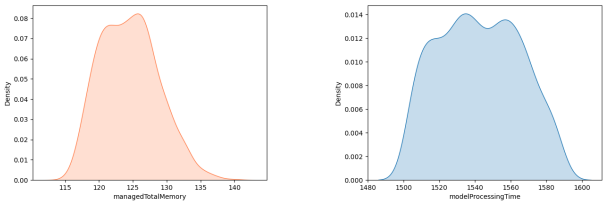
Model	Xiaomi Redmi Note 9T
Processor	Mediatek 800 series
Performance	Mid
GPU	Mali-G57
Graphics Memory Size	1024 Byte

Model	Samsung Galaxy Note 20
Processor	Exynos 990 series
Performance	Mid to High
GPU	Mali-G77
Graphics Memory Size	2048 Byte

Model	Motorola One Fusion+
Processor	Snapdragon 730
Performance	Mid
GPU	Adreno (TM) 618
Graphics Memory Size	2048 Byte

TABLE III: Tables of statistics on mobile devices

Below are the graphs related to the Adreno 618 GPU that indicate the total amount of RAM used by the application during the execution (Managed Total Memory) and the time needed by the physical devices to obtain the inference result from the model (Model Processing Time).



(a) Managed Total Memory (b) Model Processing time

Fig. 5: Images of results on Adreno 618 GPU

V. Conclusion

The approach offered in this paper has proved that it's possible to merge the world of augmented reality with that of deep learning, bringing classic deep learning tasks to mobile devices in real time. Despite this, there are still some issues to improve, especially in low-performance mobile devices.

Future works could be based on the improvement of the aforementioned problems and on the implementation in an AR environment of new classic deep learning tasks, such as segmentation. Another future development may be to work on network pruning in order to find the right compromise between performance and computation. Pruning is the compression of the model which aims to reduce the size of the models by minimizing the loss of precision or performance.

References

[1] Akgul O., Penekli H., Genc Y., "Applying deep learning in augmented reality tracking", 2016, 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), IEEE (2016), pp. 47-54.

[2] Krizhevsky A., Sutskever I., Hinton G.E., "Imagenet classification with deep convolutional neural networks", 2012, Advances in Neural Information Processing Systems (2012), pp. 1097-1105.

[3] Limmer M., Forster J., Baudach D., Schüle F., Schweiger R., Lensch H.P., "Robust deep-learning-based road-prediction for augmented reality navigation systems at night", 2016, IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), IEEE (2016), pp. 1888-1895.

[4] Połap D., Kesik K., Książek K., Woźniak M., "Obstacle detection as a safety alert in augmented reality models by the use of deep learning techniques", 2017, Sensors, 17 (12) (2017), pp. 1-16.

[5] Lin C.-H., Chung Y., Chou B.-Y., Chen H.-Y., Tsai C.-Y., "A novel campus navigation app with augmented reality and deep learning", 2018, IEEE International Conference on Applied System Invention (ICASI), IEEE (2018), pp. 1075-1077.

[6] Nepal Prabin, "MobileNet Architecture Explained," 2020.

[7] Rachel Huang, Jonathan Pedoeem, Cuixian Chen, "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers", 2019.

[8] Joseph Redmon, Ali Farhadi, "YOLOv3: An Incremental Improvement", 2018.

[9] Mohammad Javad Shafiee, Brendan Chywl, Francis Li, Alexander Wong, "Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video", 2019.