Real-time Object Recognition on the Microsoft Hololens

Adam Ibrahim
ai@cs.ucsb.edu

John B. Lanier
johnblanier@gmail.com

Brandon Huynh
bhuynh@cs.ucsb.edu

John O'Donovan
jod@cs.ucsb.edu

Tobias Höllerer
holl@cs.ucsb.edu

***Abstract*—We investigate the possibility of running object recognition tasks on the Microsoft HoloLens Augmented Reality (AR) headset in real time. Two different approaches are implemented: one using the Microsoft Project Oxford API, the other using a pre-trained neural network (YOLO) running on a local server. We find that while there are workarounds for the latency, the main obstacle to real-time performance is the HoloLens itself.**

***Index Terms*—Object Recognition, Convolutional Neural Networks, Microsoft HoloLens, Augmented Reality.**

I. Introduction

Since the revival of interest in neural networks in 2012[1], Convolutional Neural Networks (CNN) have become the state of the art in object recognition tasks, even beating humans on some datasets[2]. Their high accuracy has led them to be used in a wide range of applications, such as self-driving cars [3][4][5], healthcare[6], and e-commerce[7]. In parallel, Augmented and Virtual reality (AR and VR) are undergoing a fast expansion due to advances in technology. Microsoft's HoloLens, for example, is a fully portable, stand-alone AR headset capable of tracking the environment in real-time with centimeter accuracy. This opens up a myriad of possibilities, such as the ongoing language learning project, which leverages the HoloLens's sensors to virtually annotate physical objects in order to teach the user new vocabulary words. Machine learning could play a key role by handling the automatic identification and localisation of objects seen by the HoloLens's cameras, allowing users to use such a system anywhere without having to manually assign the labels beforehand. Furthermore, the use of machine learning could enable new interaction techniques in AR[8].

In order to test whether the use of such algorithms with the HoloLens is viable, we try two different approaches: using an API, and using algorithms running on our own servers. We limit ourselves to object recognition as a first step. Note that running such algorithms directly on the HoloLens was also attempted, but quickly discarded due to insufficient processing power leading to abysmal framerate. For the API, we use Microsoft's Project Oxford, which comes with free credits. On the servers, we use YOLO[9][10], due to the model's very competitive inference speed and portability. The questions we aim to answer are:

1) Can object recognition as a service be used in real time with the HoloLens ?
2) Can a custom neural network running on consumer computers be used in real time with the HoloLens ?

II. Pipeline

A. HoloLens client application

As of the end of 2016, the Microsoft HoloLens does not expose an API to send a video feed at low latency. Therefore, we instead use an asynchronous coroutine to take pictures at regular intervals or as fast as possible, which allows us to take up to 6 to 7 pictures per second. The pictures are then turned into a byte array which is sent to the Microsoft Project Oxford API using a JSON request, or the YOLO server. The client is configured to wait for a reply from the server, and display the information obtained from the server in the app for the user to see. If bounding boxes and locations are provided with the labels by the server, they are rendered at the relevant location on the client application. One the application gets a response for the server, it can take a new picture for the server to process.

B. API

The Microsoft Project Oxford API consists in a computer vision service, capable of performing different classification tasks such as object recognition, face recognition or emotion recognition. We solely focus on the object recognition service in this project. The API accepts an image, and returns labels for a few objects recognised in the scene. The API recognises up to 2000 classes as of 2017. Note that the API does not provide location or boundary information, only labels.

C. YOLO server

We set up YOLO on a computer connected to the same network as the HoloLens to receive the images sent by the HoloLens app. We are free to use one of the existing YOLO weights pre-trained on different datasets and classes[9][10], or train our own weights. YOLO then reports bounding boxes and labels for detected objects in the input image, which are sent back to the HoloLens client to display.

D. Addressing latency

Taking a picture and sending it through the network takes up to several dozen milliseconds on the HoloLens. The processing time for the local YOLO server and the APIs is comparable, ranging from 100-300 ms when YOLO's weights are trained on the VOC or COCO datasets. As the HoloLens may have

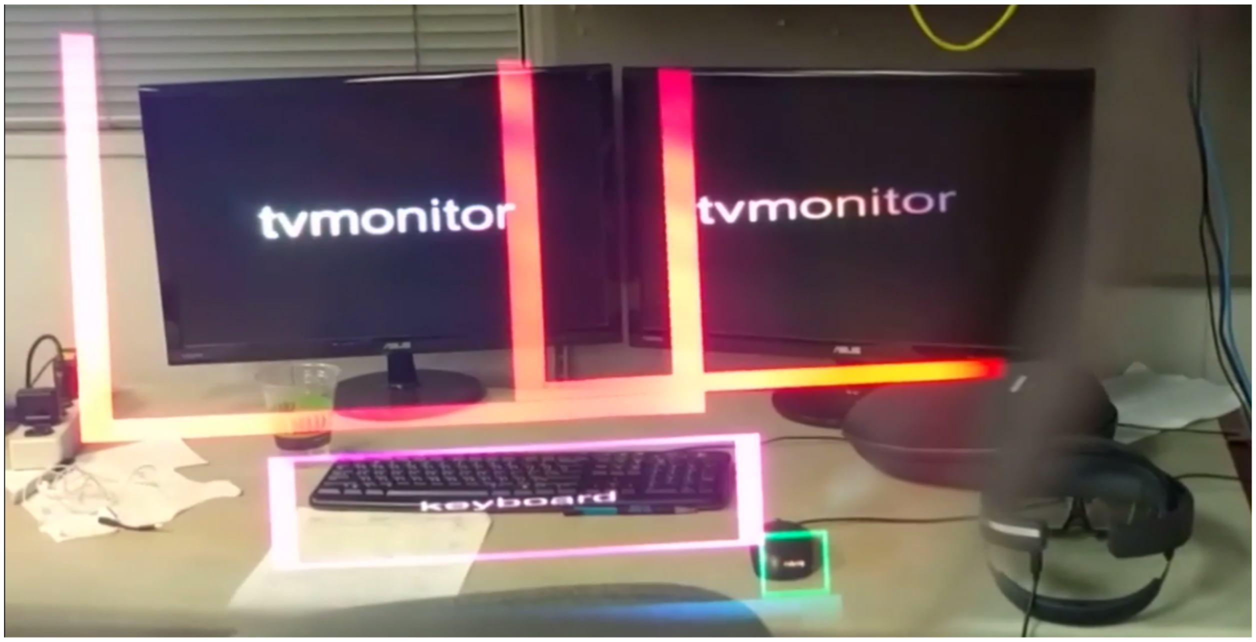

Fig. 1: YOLO interfaced with the HoloLens, as seen from the latter.

moved in the meanwhile, and inference being run on a 2D image, we must be careful to account for the latency for the labels and bounding boxes to be accurately placed. More specifically, labels and bounding boxes coordinates are given in a camera's 2D image space. As can be seen on Fig.2, if the camera moves between the moment the picture is taken and the moment when the server's answer is received, rendering the bounding boxes and labels at the position predicted from the input image in the camera's image plane will lead to erroneously placements. In order to solve that problem, we store the camera-to-world's matrix in memory at the moment a picture is taken. This allows us to have a copy of the main camera's (MC) configuration at that time, which we denote on Fig.2 by TC (temporary camera). The query is sent to the remote server, which processes it, and returns labels and a bounding box if applicable. Those are then rendered in the temporary camera's image plane instead of the main camera's, allowing us to place the labels and bounding boxes where the objects are in 3D space, even if they are out of the main camera's field of view, as long as only the HoloLens has moved with respect to the mapped room in the meanwhile. Note that if the objects move too with respect to the mapped room, this fix will not work.

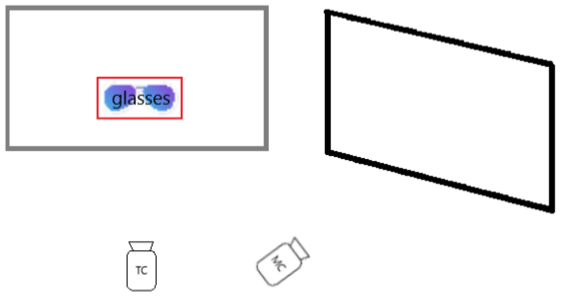

Fig. 2: Illustration of a solution for the misplacement of labels and bounding boxes due to latency: the camera is in configuration TC when taking a picture, and configuration MC when receiving the server's answer. The labels and boxes are placed in the coordinate system defined by camera TC instead of the current camera's.

III. Limitations and Future Work

A. HoloLens

While trying to perform processing directly on the HoloLens using OpenCV, we quickly ran into an issue with the available resources on the HoloLens: the frame rate would drop dramatically, affecting negatively the whole performance of the app, including tracking. Moreover, the use of the camera in an app prevents us from using the mixed reality capture functionality to see what the user sees when using the app. These issues might not be present with other AR devices or future versions of the HoloLens.

B. Server

The server was able to handle extremely well the VOC and COCO weights, but struggled with the YOLO9000 weights. As inference using the YOLO9000 weights (and 9000 classes) takes 2-3 seconds on an NVIDIA GTX 980 Ti, we had to limit ourselves to the weights trained on the VOC and COCO datasets, which respectively have 20 and 80 classes. This limits our ability to compare the API and YOLO as the API handles significantly more classes than VOC and COCO. Note that this could be addressed by using more powerful GPUs in parallel with the YOLO9000 weights. Another core problem with the server is that latency affected the system's performance in ways that could not be fixed by solutions as simple as the temporary camera trick. For example, the processing time prevents the labels and bounding boxes from moving smoothly with the labelled objects in the field of view when the user is walking or looking around. Moreover, labels sometimes change (mislabelling) due to changes in illumination and pose as the user moves. A workaround could be to keep track of assigned labels in the room's 3D mesh instead of simply rendering them in the 2D image plane and discarding them at the next query. This may work as long as there are not too many classes, as otherwise some labels will stack if the system recognises objects at different resolutions in the same scene. Segmentation could help getting 3D boundaries of objects from the initial 2D prediction, with predicted classes for a given object being tracked in time to stabilise the classification by displaying the label that has the most votes.

IV. Conclusion

The success of the YOLO server in providing a framework that could be used in real-time to perform machine learning in conjunction with the HoloLens opens the way for more customised processing. Both the API and YOLO, even when the in-house server was not accessed locally, were found to be viable for real-time performance. However, the API's inability to localise the objects and provide bounding boxes limits its potential for AR. The successful interfacing of the HoloLens and a server running machine learning algorithms allows us to focus our efforts on building our own models and algorithms, or attempt to use available models on our servers. This could allow us to address some of the problems encountered while using YOLO and enable new applications such as the ongoing language learning and hybrid server modelling projects.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[3] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[5] J. Li, X. Liang, S. Shen, T. Xu, J. Feng, and S. Yan, "Scale-aware fast r-cnn for pedestrian detection," *IEEE Transactions on Multimedia*, 2017.

[6] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in Bioinformatics*, p. bbx044, 2017.

[7] D. Etherington, "Amazon puts image recognition into its main ios app prepare to be even more showroomed, retailers," *Techcrunch*, 2014.

[8] N. Xu, B. Price, S. Cohen, J. Yang, and T. S. Huang, "Deep interactive object selection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 373–381, 2016.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.

[10] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.