Accelerating Handheld Object Pose Estimation on Smartphones in Unity with TensorFlow Lite

Valentin Miu
vmiu@bournemouth.ac.uk
Centre for Digital Entertainment, Bournemouth University
PlayFusion Ltd., Cambridge

Oleg Fryazinov
ofryazinov@bournemouth.ac.uk
The National Centre for Computer Animation,
Bournemouth University

1 Introduction

Recent developments in computer vision methods and techniques were boosted by the application of machine learning methods to increase precision. In particular, CV techniques found a good number of applications with relation to augmented and mixed reality (AR/MR). However, implementation depends on the frameworks which are often aimed at desktop machines, not mobile devices.

In this work, we discuss the 6DoF tracking of handheld objects, which is of particular interest to augmented reality applications. This field requires precise pose estimation obtained in real-time to ensure user-friendly interaction with mixed reality scenes. Also, we focus on smartphones as our target devices, which requires our approach to be low-resource and computationally efficient. In particular, we consider the tracking of a handheld hair curler, to train inexperienced users in its proper operation.

We use the C API and JNI of TensorFlow Lite and leverage its Android and iOS GPU support, as well as Unity3D's support for complex AR scenes. Rapid pose inference is implemented on both smartphone GPUs and MacOS/Windows CPUs through a native C++ Unity3D plugin. Additionally, several custom or off-the-shelf were run through the plugin, with minimal code changes to the Unity scripts.

2 Method and Results

The pose estimation approach is based on [3]. This uses a YOLOv2 network with a darknet-19 backbone [2], with the last convolution layer modified to output the 2D projection positions of the object centroid and the 3D bounding box corners, as well as the object class (here open/closed), and detection and class confidences. The 6DoF pose is then retrieved using a perspective-and-point algorithm. To optimize this model for mobile devices, a smaller existing feature extractor (tinyYOLOv2) is modified and trained. Given the perceived lack of TensorFlow Lite support for the LeakyReLU activations of the backbone, all activations were changed to ReLU (this was later shown to be unnecessary, given the equivalence of the Lite-supported PReLU operation at inference time). Although the LeakyReLU-activated ImageNet-pretrained tinyYOLOv2 weights were used as a starting point, this did not significantly affect convergence.

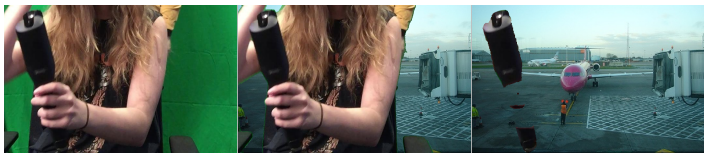

Figure 1: An cropped unkeyed source frame (a), the keyed frame with a randomized background (b), and the segmented curler frame with a randomized background (c). Only the latter two were used for training.

The training image dataset were frames from videos, most with green-screened backgrounds, taken with a variety of cameras. The pose of the curler and the opening angle were annotated manually, using a custom Unity3D tool to align a CAD model of the curler to a dataset video frame. Using an optical flow algorithm, frames near manually annotated "keyframes" were able to be annotated automatically, boosting the size of the dataset from 2400 to 16700 images. The green-screen backgrounds were keyed out and replaced with random images from the Pascal VOC 2012 dataset [1]. Given the similarity between frames and the small number of different subjects, the segmented curlers alone were overlaid onto random backgrounds (see Figure 1) and added to the greenscreen-keyed images, to allow for generalization during training. This brought the effective dataset size to 33400 images. Since some images were portrait and some were landscape, they were padded to a square input size during training.

Training was done on a Titan Xp and a subset of a 8-GPU NVIDIA RTX 2080 Ti cluster. Conversion was done manually by re-creating the graph in TensorFlow and copying the weights, and then converting to Lite.

The tinyYOLOv2 model achieved real-time performance on both iOS and Android devices for an input size of 224x224, and fast performance for a size of 512x512 (Table 1). Although reasonably successful in the approximation of yaw and pitch, the method has as of yet not proven successful in estimating roll angle ground truth, in part due to lack of visual curler feature variation with roll angle variation, especially when tested on unseen data in in-app testing.

	GPU fwd. pass (prep.)	Per-point RMS error for 2D projection
iPhone XS (224)	35 ms (12 ms)	10.93 px ($48.7 \cdot 10^{-3} \cdot 224$)
OnePlus 6 (224)	70 ms (10 ms)	
iPhone XS (512)	90 ms (65 ms)	4.32 px ($8.44 \cdot 10^{-3} \cdot 512$)
OnePlus 6 (512)	125 ms (30 ms)	

Table 1: Forward pass and preprocessing step latency. Since the latter is done asynchronously on the CPU, the forward pass represents the time between subsequent annotation outputs, while the sum of the two is the time between frame capture and annotation output.

Given that the bounding box corners were often far from the curler, causing an offset from the relevant features, inference of 12 manually selected points on the curler, as well as the object centroid, was attempted. This did not lead to a noticeable improvement, and caused the detection confidence value to be too low to reliably threshold from misdetections.

Given its suitability for smartphone inference due to its low operation count, as well as its precision-increasing skip connections, a version of the pose network using a MobileNetV2 feature extractor was constructed. While this decreased the forward pass times by about 35%, ensuring proper misdetection thresholding in unseen data with this model is an ongoing issue.

Our implementation allows to go beyond the original task of the tracking of handheld hair curler. In addition to the pose estimation network, several off-the-shelf or custom models were tested with this plugin, including a MobileNetV2-SSDLite bounding box detector, a PoseNet human pose estimator, and hair and face segmentation networks. The results show that our implementation allows efficiency on mobile devices with a potential to improve the results even further.

Acknowledgments

We would like to thank PlayFusion Limited for allowing use their codebase and expertise to create an implementation of our method. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2): 303–338, jun 2010.

[2] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 6517–6525, dec 2017.

[3] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-Time Seamless Single Shot 6D Object Pose Prediction. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 292–301. IEEE, dec 2018.