Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

OpenPose-Plus: A Flexible and High-performance Human Pose Estimation Framework

Marina Meilă MMP@STAT.WASHINGTON.EDU
Department of Statistics
University of Washington
Seattle, WA 98195-4322, USA

Michael I. Jordan JORDAN@CS.BERKELEY.EDU
Division of Computer Science and Department of Statistics
University of California
Berkeley, CA 94720-1776, USA

Editor: Kevin Murphy and Bernhard Schölkopf

Abstract

Keywords: Human Pose Estimation, Flexible High-level APIs, High-performance System

1. Introduction

Human pose estimation is a core task in computer vision. Its goal is to localise human anatomical key-points (e.g., elbow, wrist, etc.) and use the detected key-points to infer human pose. Estimating human pose has many important applications, such as interactive gaming (**?**), automated supermarket (**?**), surveillance camera (**?**) and self-driving cars (**?**). This has motivated practitioners and scientists to invent many useful pose estimation algorithms (**?????**) recently.

It is important to design, test and deploy the pose estimation algorithms in real-world environments. Developers find this difficult for two major challenges: (1) They often need to *flexibly* customise the algorithms based on pose estimation applications. These applications contain many dependent components used in the training and inference phases. During training, different applications need different training datasets, data augmentation, model architectures and logic for interpreting detected key-points. During inference, these applications also need to achieve different ways of handling real-time data inputs and embedded platforms (e.g., TensorRT or CPU); (2) Pose estimation applications produces challenging data input rates (i.e., processing tens of high-resolution images per second on an embedded platform (**?**)). Developers must ensure no bottleneck can occur in all the components in a pipeline , and thus can achieve the preferred real-time processing.

Existing pose estimation systems are unable to fully address these challenges. On one end, there are domain-specific systems such as OpenPose (**?**) which achieves high-performance using a monolithic low-level implementation. There are no APIs for configuring the systems, and as a result, the developers have to largely modify or rewrite the the systems based on the specific requirements from applications. On the other end, there are general-purpose high-level CPU/GPU engines such as TensorFlow and PyTorch that

©2000 Marina Meilă and Michael I. Jordan.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v1/meila00a.html.

can be used for pose estimation These engines allow the users to easily customise different components; but with major performance overheads.

To close this gap, we design a human pose estimation framework OpenPose-Plus that aims to provide both flexibility and high-performance. OpenPose-Plus has two novel designs: (i) high-level Python APIs which allows users to customise abstracted components that are pervasively used in training and deploying pose estimation algorithms; (ii) a high-performance runtime that can automatically run these components on embedded hardware: TensorRT, minimise data movement to achieve low-latency, and collectively schedule the component to ensure all input can be processed in real-time.

We demonstrate the generality of our design by using OpenPose-Plus to implement various state-of-the-art pose estimation applications (**???**). Evaluation result shows that OpenPose-Plus is able to process 30 [resolution?] images per second produced by a high-resolution camera on a commodity 1070 GPU, which goes beyond the 24 images/s real-time processing requirement. OpenPose-Plus was open-sourced in October 2018 with a Apache 2 licence. Since after, it has attracted numerous industry and academic users, and created a large use base on Github (700 stars by the date of submission).

2. Architecture and Runtime Design

3. High-level APIs and Applications

4. Conclusion

References