Real-time 2D Multi-Person Pose Estimation on CPU: Lightweight OpenPose

Daniil Osokin
Intel
daniil.osokin@intel.com

Abstract

In this work we adapt multi-person pose estimation architecture to use it on edge devices. We follow the bottom-up approach from OpenPose [3], the winner of COCO 2016 Keypoints Challenge, because of its decent quality and robustness to number of people inside the frame. With proposed network design and optimized post-processing code the full solution runs at 28 frames per second (fps) on Intel® NUC 6i7KYB mini PC and 26 fps on Core i7-6850K CPU. The network model has 4.1M parameters and 9 billions floating-point operations (GFLOPs) complexity, which is just ∼15% of the baseline 2-stage OpenPose with almost the same quality. The code and model are available as a part of Intel® OpenVINO™ Toolkit.

1 Introduction

Multi-person pose estimation is an important task and may be used in different domains, such as action recognition, motion capture, sports, etc. The task is to predict a pose skeleton for every person in an image. The skeleton consists of keypoints (or joints): ankles, knees, hips, elbows, etc.

Human pose estimation accuracy was greatly improved with the help of convolutional neural networks (CNNs) [6], [4], [16]. However, there is a little research on compact, yet efficient pose estimation methods. In [9] authors show a simplified Mask R-CNN keypoint detector demo on a mobile phone, running at 10 fps, however neither implementation details nor accuracy characteristics were provided. We have also found the open-source repository [10] with human pose estimation network. Author reported inference speed of 4.2 fps on 2.8GHz Quad-core CPU and 10 fps on Jetson TX2 board.

In our work we optimize the popular method OpenPose and show how modern design techniques of CNNs can be used for pose estimation task. As a result, our solution runs at:

- 28 fps on mini PC Intel® NUC, which consumes little power and has 45 watt CPU TDP.
- 26 fps on a usual CPU without the need of a graphic card.

The accuracy of the optimized version nearly matches the baseline: Average Precision (AP) drop is less than 1%.

2 Related Work

Multi-person pose estimation problem can usually be approached in two ways. The first one, called *top-down*, applies a person detector and then runs a pose estimation algorithm per every detected person. So pose estimation problem is decoupled into two subproblems, and the state-of-the-art achievements from both areas can be utilized. The inference speed of this approach strongly depends on number of detected people inside the image.

The second one, called *bottom-up*, more robust to the number of people. At first all keypoints are detected in a given image, then they are grouped by human instances. Such approach usually faster than the previous, since it finds keypoints once and does not rerun pose estimation for each person.

In [11] authors proposed the fastest method to date with state-of-the-art quality among bottom-up methods, which runs 23 fps on a single GTX 1080 Ti graphic card for an image with 3 persons. They note, that performance will degrade to 15 fps for image with 20 persons. We based our work on the popular bottom-up method OpenPose, it has almost invariant to number of people inference time.

3 Analysis of the Original OpenPose

3.1 Inference Pipeline

Similar to all bottom-up methods, OpenPose pipeline consist of two parts:

- Inference of Neural Network to provide two tensors: keypoint heatmaps and their pairwise relations (part affinity fields, pafs). This output is downsampled 8 times.
- Grouping keypoints by person instances. It includes upsampling tensors to original image size, keypoints extraction at the heatmaps peaks and their grouping by instances.

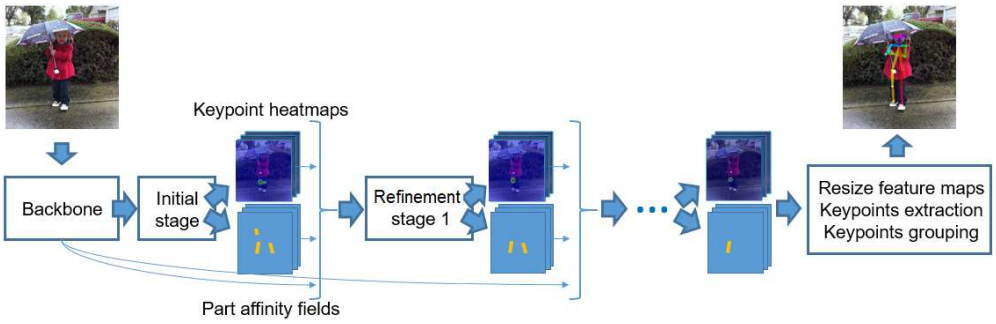

Figure 1: OpenPose pipeline.

The network first extracts features, then performs initial estimation of heatmaps and pafs, after that 5 refinement stages are performed. It is able to find 18 types of keypoints. Then grouping procedure searches the best pair (by affinity) for each keypoint, from the predefined list of keypoint pairs, e.g. *left elbow* and *left wrist*, *right hip* and *right knee*, *left eye* and *left ear*, and so on, 19 pairs overall. The pipeline is illustrated in Fig. 1. During inference, input image is resized to match network input size by height, the width is scaled to preserve image aspect ratio, then padded to the multiple of 8.

3.2 Complexity Analysis

The original implementation uses VGG-19 backbone [14] cut to *conv4_2* layer as a features extractor. Then two extra convolutional layers *conv4_3* and *conv4_4* are added. After that initial and 5 refinement stages are made.

Each stage consists of two parallel branches: one for heatmaps estimation and one for pafs. The two branches have the same design, shown in Table 1. We set network input resolution to 368x368 in our comparison and use the same COCO validation subset as in original paper, single scale testing is performed. The test CPU is Intel® Core™ i7-6850K, 3.6GHz. Table 2 shows the trade-off between accuracy and number of refinement stages.

It can be seen, that the latter stages give less improvement per GFLOPs, so for the optimized version we will keep only the first two stages: the initial stage and a single refinement stage.

The profile for the post-processing part is summarized in the Table 3. It was obtained by running the code, which was written in C++ with OpenCV [2]. Despite the grouping itself is lightweight, other parts are subject to optimization.

Table 1: OpenPose stages design. Each stage has 2 parallel branches (single is shown).

Initial	Refinement
conv 3x3/128	conv 7x7/128
conv 3x3/128	conv 7x7/128
conv 3x3/128	conv 7x7/128
conv 1x1/512	conv 7x7/128
	conv 7x7/128
	conv 1x1/128

Table 2: Accuracy vs. Complexity of OpenPose on COCO validation set.

	AP, %	GFLOPs	GFLOPs total
Backbone	n/a	37.8	37.8
conv4_3	n/a	2.5	40.3
conv4_4	n/a	0.6	40.9
Initial stage	35.5	2.2	43.1
Refinement stage 1	43.4	18.6	61.7
Refinement stage 2	46.2	18.6	80.3
Refinement stage 3	47.4	18.6	98.9
Refinement stage 4	48.1	18.6	117.5
Refinement stage 5	48.6	18.6	136.1

Table 3: Initial performance of post-processing and grouping.

	Resize feature maps	Extract keypoints	Group keypoints	Total
Fps	10.5	1.81	454	**1.54**

4 Optimization

4.1 Network Design

All experiments were performed with the default training parameters form the original paper, and we used the COCO dataset [12] to train on. As pointed above, we keep only initial and first refinement stage. However, the rest stages can provide regularizing effect, so the final network was retrained with additional stages, but the first two are used. Such procedure gives $\sim$1% AP improvement.

4.1.1 Lightweight Backbone

Since time when VGG nets were proposed, few lightweight network topologies with similar or even better classification accuracy were designed [7], [8], [13]. We evaluated networks from MobileNet family to replace the VGG feature extractor and started from MobileNet v1.

In a naive way, if we keep all layers till deepest, which matched output tensor resolution, it leads to significant accuracy drop. This might be due to shallowness and weak feature representation. To save spatial resolution and reuse backbone weights we use dilated convolution [17]. Stride of *conv4_2/dw* layer was removed and dilation parameter value was set to 2 for succeeding *conv5_1/dw* layer to preserve receptive field. So we use all layers till *conv5_5* block. Addition of *conv5_6* block improves the accuracy, but at cost of performance. We also tried more lightweight backbone MobileNet v2, however it did not show good result, see Table 4.

4.1.2 Lightweight Refinement Stage

To produce new estimation of keypoint heatmaps and pafs the refinement stage takes features from backbone, concatenated with previous estimation of keypoint heatmaps and pafs. Motivated by this fact we decided to share the most of computations between heatmaps and pafs and use *single prediction branch* in initial and refinement stage. We share all layers except the two last, which directly produce keypoint heatmaps and pafs, see Fig. 2.

Then each convolution with 7x7 kernel size was replaced by a convolutional block with the same receptive field, to capture long-range spatial dependencies [15]. We conducted series of experiments with this block design and observed that it's enough to have three consecutive convolutions with 1x1, 3x3, and 3x3 kernel size, the latter with dilation parameter equals to 2, to preserve initial receptive field. Because the network became deeper, we added residual connection [5] for each such block.

Table 4: Lightweight backbone selection study (the initial and refinement stages have original OpenPose design).

	AP, %	GFLOPs
MobileNet v1 *(cut to conv4_1)*	37.9	23.3
Dilated MobileNet v1 *(cut to conv5_5)*	42.8	27.7
Dilated MobileNet v1 *(cut to conv5_6)*	43.2	31.3
Dilated MobileNet v2 *(cut to conv6_3)*	39.6	27.2

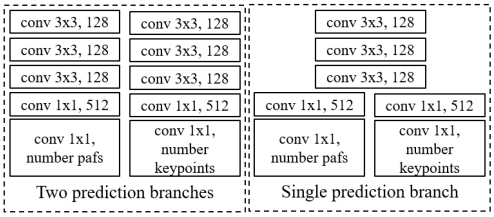

Figure 2: Original two prediction branches and proposed single prediction branch for the initial stage. We also apply this scheme for the refinement stage.

The final design visualized in Fig. 3, it has ∼2.5 times less complexity than convolution with 7x7 kernel. We also replaced conv4_3 with 3 depthwise separable convolutions, channels number was reduced from 256 to 128. The complexity and accuracy of the proposed network design are shown in the Table 5.

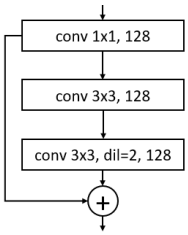

Figure 3: Design of convolutional block for replacement convolutions with 7x7 kernel size in refinement stage.

Table 5: Accuracy versus Complexity of proposed network on COCO validation set.

	AP, %	GFLOPs	GFLOPs total
Dilated MobileNet v1	n/a	3.7	3.7
conv4_3	n/a	0.3	4
conv4_4	n/a	0.3	4.3
Initial stage	35	1.3	5.6
Refinement stage 1	41.4	3.4	9
2-stage network, retrained with all refinement stages	**42.8**	n/a	**9**

4.2 Fast Post-processing

We profiled the code and removed extra memory allocations, parallelized keypoints extraction with OpenCV's routine. This made code significantly faster, and the last bottleneck was the resize feature maps to the input image size.

We decided to skip the resize step and performed grouping directly on network output, but accuracy dropped significantly. Thus step with upsampling feature maps cannot be avoided, but it is not necessary to do it to input image size. Our experiments shown, that with upsample factor 8 the accuracy is the same, as if resize to input image size. We used upsample factor 4 for the demo purposes.

4.3 Inference

For the network inference we use the Intel® OpenVINO™ Toolkit R4 [1], which provides optimized inference across different hardware, such as CPU, GPU, FPGA, etc. Final performance numbers are shown in the Table 6, they were measured for a challenging video with more than 20 estimated poses.

We used two devices: Intel NUC6i7KYB, which performed inference on the integrated GPU Iris Pro Graphics P580 in half-precision floating-point format (FP16), and 6-core Core i7-6850K CPU, which performed inference in single-precision floating-point format (FP32). Network input size was set to 456x256, which is similar to 368x368, but with 16:9 aspect ratio, suitable for processing video streams.

Table 6: Final inference fps for a video with more than 20 estimated poses. Numbers in braces are network inference and post-processing fps.

	NUC	CPU
Baseline	1.17 (3.92/1.66)	0.95 (2.47/1.54)
Proposed	**28** (33/160)	**26** (33/125)

5 Conclusion

In this work, we approached the problem of human pose estimation network, suitable for real-time performance on edge devices. We proposed the solution, based on OpenPose method, with heavily optimized network design and post-processing code. The accuracy versus network complexity ratio was increased in more than 6.5 times due to the use of dilated MobileNet v1 feature extractor with depthwise separable convolutions and design of lightweight refinement stage with residual connections. The network can be downloaded as a part of the OpenVINO Toolkit under the name *human-pose-estimation-0001*. The network description is available in the Open Model Zoo repository.

The full solution runs in real time on a usual CPU, as well as on NUC mini PC and closely matches accuracy of the baseline 2-stage network. Some techniques may further improve performance and accuracy, such as quantization, pruning, knowledge distillation. We left them for the future research.

References

[1] OpenVINO Toolkit. `https://software.intel.com/en-us/openvino-toolkit`.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Z. Cao, T. Simon, S. Wei, and Y. Sheikh. Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. In *CVPR*, 2017.

[4] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu. RMPE: Regional Multi-person Pose Estimation. In *ICCV*, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *ICCV*, 2017.

[7] S. Hong, B. Roh, K.-H. Kim, Y. Cheon, and M. Park. PVANet: Lightweight Deep Neural Networks for Real-time Object Detection. In *arXiv preprint arXiv:1611.08588*, 2016.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *arXiv preprint arXiv:1704.04861*, 2017.

[9] A. Jindal and et al. Enabling full body ar with mask r-cnn2go. In *https://research.fb.com/enabling-full-body-ar-with-mask-r-cnn2go/*, 2018.

[10] I. Kim. tf-pose-estimation. `https://github.com/ildoonet/tf-pose-estimation`.

[11] M. Kocabas, S. Karagoz, and E. Akbas. MultiPoseNet: Fast multi-person pose estimation using pose residual network. In *ECCV*, 2018.

[12] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *ECCV*, 2014.

[13] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[15] S. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *CVPR*, 2016.

[16] B. Xiao, H. Wu, and Y. Wei. Simple Baselines for Human Pose Estimation and Tracking. In *ECCV*, 2018.

[17] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *CVPR*, 2017.