XiNet-pose: Extremely lightweight pose detection for microcontrollers

Alberto Ancilotto, Francesco Paissan, Elisabetta Farella
aancilotto@fbk.eu, fpaissan@fbk.eu, efarella@fbk.eu
E3DA Unit, Digital Society center, *Fondazione Bruno Kessler (FBK)*, Trento, Italy

***Abstract*—Accurate human body keypoint detection is crucial in fields like medicine, entertainment, and VR. However, it often demands complex neural networks best suited for high-compute environments. This work instead presents a keypoint detection approach targeting embedded devices with very low computational resources, such as microcontrollers. The proposed end-to-end solution is based on the development and optimization of each component of a neural network specifically designed for highly constrained devices. Our methodology works top-down, from object to keypoint detection, unlike alternative bottom-up approaches relying instead on complex decoding algorithms or additional processing steps. The proposed network is optimized to ensure maximum compatibility with different embedded runtimes by making use of commonly used operators. We demonstrate the viability of our approach using an STM32H7 microcontroller with 2MB of Flash and 1MB of RAM. We achieve a maximum mAP of 57.9 without relying on external RAM, and good detection performance at latencies down to 133ms per frame.**

***Index Terms*—human pose estimation, pose tracking, realtime, embedded devices, microcontrollers**

Fig. 1. Examples of pose detections with the proposed approach.

I. Introduction

Human pose estimation refers to estimating the position of various keypoints of the human body (such as shoulders, hands, and feet) from an image or video input stream. Accurate pose detection of the human body from images has important uses in many fields, such as medicine, entertainment, human-computer interaction, virtual reality, and countless others [1], [2], [4], [7], [8]. Though the problem of human body keypoint detection is simple to understand, effective and accurate solutions usually rely on complex neural networks and, therefore, are generally suited to environments with vast computational capabilities, such as workstations. Bringing these approaches to smaller, low-power devices can enable analysis in devices such as smartphones, action cameras, and other consumer devices. Further reducing the complexity of these approaches may also allow these solutions to be embedded directly within video sensors, removing the need for an extra processor for video analysis and leading to even more significant reductions in system complexity and power demands. In this work, we build on recent developments in scalable backbones. The MobileNet family [9], [10], [20], for example, focuses on using depthwise convolutions and inverted residual blocks to significantly reduce the number of parameters; ShuffleNet [23] and ShuffleNetV2 [17] rely on pointwise convolutions and channel shuffle to reduce computation cost while maintaining accuracy; EfficientNet [21] and EfficientNetV2 [22] use instead an advanced network scaling principle coupled with a novel architecture search algorithm. PhiNets [18] and its improved version XiNets [3] push the computational load reduction even further by adopting a hardware-aware scaling approach. Starting from scalable solutions tailored explicitly for very tiny embedded devices such as microcontrollers [3], [15], [18], we develop our keypoint detector with an approach similar to that used by YoloV8-pose [12]. Unlike competing methods, this solves the pose detection problem as an extension of the object detection task, well addressed in computer vision.

Our solution is novel in its end-to-end nature, requiring no supplementary processing after the neural network execution, while being applicable to small embedded devices and microcontrollers. In the following, we will first review related works, particularly focusing on keypoint detection and literature approaches to reduce complexity. Afterwards, we introduce the methodology adopted, demonstrating the advantage of a bottom-up approach combined with a scalable backbone. We then analyze the different components that make up a pose detection network and optimize them one by one for maximum efficiency on MCU. Finally, we evaluate our network against other equivalent solutions and verify the actual on-device performance.

We acknowledge the support of the PNRR project FAIR (PE00000013)

979-8-3503-4859-0/DATE24/© 2024 EDAA

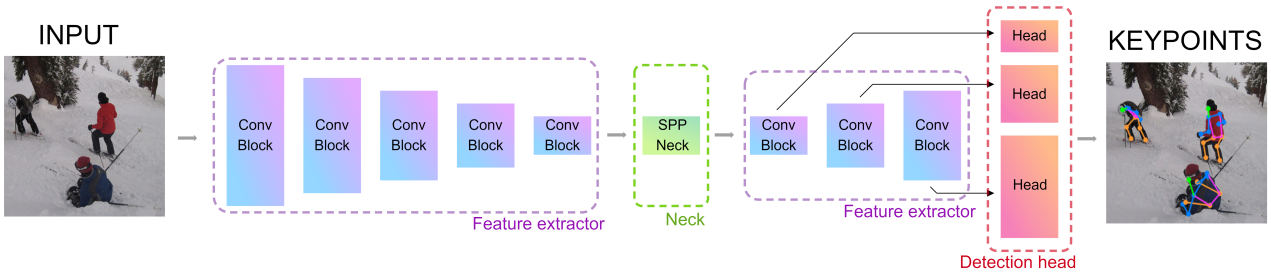

Fig. 2. Structure of the reference network in [12].

II. Related works

The pose detection task has recently been addressed through solutions based on deep convolutional neural networks [4], [7], [13]. However, these approaches often make use of highly complex networks to achieve satisfactory performance.

Lately, some studies have focused on reducing the complexity of these methods, [8], [24] with the goal of lowering latency due to processing. Nevertheless, these approaches still require far more operations than what is possible in real-time with a microcontroller. Moreover, they often rely on depthwise convolutional blocks, which are inefficient when executed on embedded devices [3].

Two different approaches for solving the pose detection task exist in the literature:

- Bottom-up, where various target keypoints are identified (usually by heatmaps) and combined into a number of target subjects. The advantage is the use of simpler backbones, but the complexity augments in reconstructing targets from the set of detected keypoints.
- Top-down, where the keypoint detection problem is solved as an extension of the object detection task. Given an input frame, several objects within it are detected, and a sequence of keypoints is associated with each. The consequence is that more complex networks are required. Still, subsequent processing is avoided, as the set of output keypoints is already associated with a target.

A further limitation of many low-complexity convolutional methods is their reliance on bottom-up strategies and complex decoding mechanisms. While this allows simpler networks for keypoint localization, the performance greatly depends on the following steps for decoding the heatmap outputs.

A notable example of these latter is PifPaf [14]. PifPaf is based on lightweight architectures such as ShuffleNetV2 [17] and MobileNetV3 [9], and it is one of the lightest and most efficient pose detectors in the literature. However, the output of the network, consisting of a collection of heatmaps and composite fields for keypoint association, requires a decoding algorithm that may come to demand more operations than the detection network itself. In addition, the performance of the approach depends heavily on both the network and the decoding algorithm, and reducing either of the two parts causes large performance drops.

Another work demonstrating a significant reduction in parameters and operations for the pose detection task is presented in [24]. This work follows the bottom-up logic. Although lightweight, the network proposed presents some critical issues for execution on microcontrollers, such as the use of inefficient operations (depthwise separable convolutions) or computation steps with massive usage of RAM (multi-scale coordinate attention, group deconvolution). RAM can be a limited resource in many embedded devices.

III. Methodology

A. Yolo for pose detection

Yolo-pose [12] is a deep learning model that can estimate the pose of people in images or videos. It is a modified version of the Yolo object detection model, specifically adapted for pose estimation.

The operation of the network is similar to that of Yolo, which is used for object detection. However, while the object detector outputs a pair of points $[(x_1, y_1), (x_2, y_2)]$ to identify a bounding box around the detected object, Yolo-pose identifies 17 different keypoints. Figure 2 shows the basic network structure.

As for other detectors in the Yolo family, Yolo-pose relies on a bottom-up approach. Each image is run only once through the network, and in a single inference, multiple objects can be detected, each already associated with its set of keypoints. No subsequent processing is required, unlike approaches relying on heatmaps or fields.

Because of its simplicity and low computational complexity, we used this as a baseline to develop our pose detector.

B. Xinet scalable backbone

We based our feature extractor backbone on a scalable architecture of the XiNet family [3]. This network has demonstrated exemplary performance when coupled with the Yolo detection head and is easily adaptable to different hardware architectures. It consists of convolutional blocks designed to optimize efficiency in on-device execution while ensuring maximum compatibility with other runtimes. These blocks use significantly fewer operations than a classical 2D convolution. Furthermore, they allow for easy tuning of the number of parameters, operations, and RAM usage by adjusting the α, β, and γ hyperparameters. This ensures that the network can be tailored to the convolutional capabilities of the target hardware platform. The optimal utilization of platform resources is guaranteed by the scaling algorithm presented in [3], [18], called Hardware Aware Scaling (HAS), which allows one-shot

tuning of the three hyperparameters α, β, and γ by knowing the amounts of FLASH memory, RAM, and speed of the target microcontroller. Consequently, the network makes the best use of all the computational capabilities of the platform, maximizing performance.

C. Dataset and evaluation metric

We use the COCO dataset [16] to evaluate the performance of the proposed approach. The COCO subset for pose detection consists of more than 200K images containing more than 250K labeled instances, highlighting the positions of 17 keypoints along the body. We use the OKS-adjusted mAP metric to evaluate the performance of the different detectors.

D. Target device and toolchain

We designed our network based on the computational capabilities of an STM32H743 microcontroller. This platform is equipped with 2MB of Flash memory; it is, therefore, possible to use networks composed of up to $2M$ parameters when quantized in `uint8` (one byte per weight). In addition, the MCU has 1MB of RAM and supports clock speeds up to 480MHz. The network used was quantized to `uint8` using TFLite. To run CNN on the device, we used the proprietary tool STM32Cube.Ai to convert the network.

IV. Network design

To create a more efficient network for our target device, we focused on optimizing three key aspects of the pose detector

- The convolutional blocks used throughout the network
- The detection head architecture
- The network neck architecture
- The pruning strategy applied to pose detection

A. Convolutional blocks

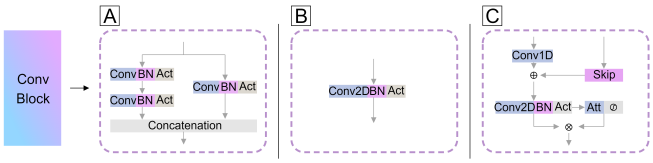

Fig. 3. The different convolutional blocks tested. A - C2f block from [12]; B - Simple Conv2D block, C - XiNet convolutional block, detailed in [3].

The base YoloV8 architecture [12] relies on a sequence of `C2f` blocks, a cross-stage partial bottleneck block [11] containing a pair of convolution operations. This block demonstrated excellent performance in object detectors in an extensive range of computational complexities (from $5B$ to $200B$ Multiply and Accumulate operations, MAC). However, the number of operations this block requires and the large amount of memory needed may not make it the best choice for very resource-constrained devices. To verify this, we trained different networks using different convolutional blocks as backbone:

- A sequence `C2f` blocks, as in the default implementation [12]
- A sequence of 2D convolutions, simplifying the backbone feature extractor
- A sequence of *Xinet* convolutional blocks [3], purposefully designed for low-resource embedded devices

In all tests, the feature extractor's baseline architecture remained the same. It consisted of 5 convolutional blocks that downscaled the input feature map by a factor of $32\times$. The network head comprised a single scale pose detection head to highlight the effects of the various backbones. The number of channels used for each convolutional block started from 32α and doubled every time the feature map was downsampled. We adjusted the scaling factor α to reach networks that had parameters as close as possible to $2M$, which was the chosen platform's limit. Due to the high values of α necessary with the smaller XiNet blocks, we tested an additional configuration that doubled the network's depth using XiNet convolutions and several channels closer to the other two backbones.

TABLE I
Performance achieved using different convolutional structures.

Convolutional Block	Parameters	MAC	mAP
C2F	1.72 M	3.2 G	26.5
2D Convolution	1.93 M	1.8 G	58.5
XiNet	1.81 M	1.4 G	58.7
XiNet (2* depth)	2.04 M	1.7 G	**60.2**

Table I shows how, while C2F blocks may be the optimal choice for larger networks, their performance drops off quickly when scaling the network down. Moreover, architectures based on these blocks require more operations for the same number of parameters and show higher RAM usage during inference. In contrast, a much simpler Conv2D implementation allows for better-performing networks at lower FLOP counts. More satisfactory results can be achieved by employing purposefully designed convolutional blocks such as XiNet. This has the advantage of limiting the number of operations needed in the block, requiring a smaller amount of RAM and allowing advanced scaling techniques such as Hardware Aware Scaling (HAS) [18].

B. Detection head design

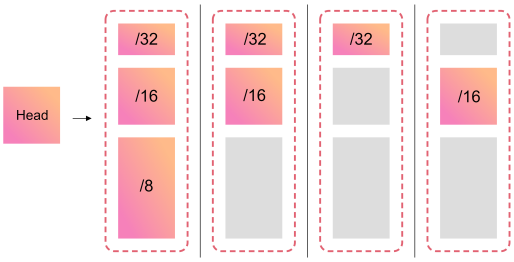

Fig. 4. Different structures for the detection head tested. Other combinations did not provide meaningful performance. Size is proportional to MAC count.

The official implementation of YoloV8 relies on a 3-scale detection head, with downsampling factors of $8\times$, $16\times$ and $32\times$. While this was proven to perform well for the target network complexities of the original work, this is not necessarily true for smaller networks, and indeed, we demonstrated how a

simpler detection head can bring better results. Following the same testing procedure as in the previous section, we tested different configurations of the detection head to identify which offered the best performance in the range of computational complexity considered. Again, we scaled the various networks by varying α to obtain networks with around 2M parameters. The best-performing network from the previous step was used as a feature extractor.

TABLE II
EFFECTS OF USING DIFFERENT DETECTION HEAD SCALES AND NUMBER ON PERFORMANCE

Head downsampling	Parameters	MAC	mAP
32×, 16×, 8×	2.05 M	4.8 G	46.4
32×, 16×	2.10 M	3.1 G	55.8
32×	1.97 M	0.7 G	51.6
16×	2.04 M	1.6 G	**60.2**

It is noticeable from Table II how simpler detection head structures result in better network performance. This occurs due to the greater complexity of the detection head when more scales are used - this, in turn, requires scaling down the feature extractor to meet the required parameter count, favouring simpler structures. Detection heads with fewer scales show the additional advantage of limiting the number of MAC operations required during execution, leading to faster networks and lower latency.

C. SPPF neck

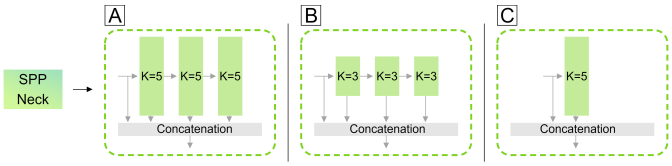

Fig. 5. Different structures for the network neck tested. A is used in the reference implementation. B and C attempt to lower the computational cost of the block by reducing the filter size or number of layers.

The latest generations of Yolo family detectors use a Spatial Pyramid Pooling (SPP) [19] neck between feature extractor and detection head. This is effective when clustering features at different scales, improving performance over earlier architectures. However, when working with devices with reduced computational capacity, this approach has two significant drawbacks:

- It relies on a sequence of MaxPooling operations at different scales, which may not be supported by different runtimes used on microcontrollers
- Although it requires few operations, MaxPooling is typically performed on CPUs, even in embedded devices with convolutional accelerators, significantly increasing latency.

We tested the performance of our approach with different configurations of Spatial Pyramid Pooling:

- The default implementation used in YoloV8 [12], based on 3 levels of pooling with kernel k=5 (A in Figure 5)
- An implementation with an equal number of levels, but using k=3 to decrease the number of operations (B in Figure 5)
- An implementation with the same kernel k=5 but a single pyramid level (C in Figure 5)

As in the previous experiments, the α parameter was adjusted for each configuration to obtain networks with approximately the same number of parameters. The results obtained are shown in Table III.

TABLE III
COMPARISON BETWEEN DIFFERENT NETWORK NECK ARCHITECTURES

SPP configuration	Parameters	mAP
Not present	1.97 M	**60.2**
3 levels, K=5	2.06 M	59.0
3 levels, K=3	2.12 M	59.5
1 level, K= 5	2.10 M	59.3

Again, networks with such low computational complexity do not benefit significantly from using complex structures such as spatial pyramid pooling, which instead considerably degrades performance. For this reason, the proposed architecture does not use this block.

D. Network pruning

We investigated the effects of pruning on the proposed architecture. Specifically, in developing the architecture, we suggest using pruning to adapt the network to the proposed requirements instead of making smaller networks. This stems from the observation that, as demonstrated in sections IV-B and IV-C, network performance seems to be determined by the size of the feature extractor instead of the complexity of the detection head used. In fact, both experiments demonstrated better performance by using a simpler detection head with a more complex detector (higher α) instead of vice versa.

It was verified that, by slightly relaxing the computational requirements (i.e., allowing for more than $2M$ parameters), significant performance gains can be achieved, as the networks are generally underfitting due to the problem's complexity and the target device's reduced computational capacity.

Because of this, allocating even a small amount of extra resources (parameters) allows significant performance gains. In this section, we want to analyze whether starting from one of these slightly larger networks may be convenient and adapt it to the device using pruning instead of directly using a smaller architecture that fits the computational limitations.

We trained 4 different networks with number of parameters $2M$ $(+0\%)$, $2.2M$ $(+10\%)$, $2.4M$ $(+20\%)$, $2.8M$ $(+40\%)$, then pruned them all down to $2M$ parameters and measured the performance achieved.

We relied on filter pruning for this operation. This technique implements a structured pruning approach that relies on the complete removal of groups of filters, estimated by minimizing an importance metric for each filter, evaluated on a testing dataset. Filter pruning allows entire filters to be removed from the network, enabling efficient execution with runtimes perhaps not optimized for other types of pruning (structured/unstructured), thus ensuring maximum compatibility and minimal overhead. Filter pruning has been implemented as presented in [5], [6]. Results obtained are shown in table IV

TABLE IV
COMPARISON BETWEEN DIFFERENT PRUNING AMOUNTS. NETWORKS WERE TRAINED WITH 10% TO 40% MORE PARAMETERS THAN ALLOWED, AND THE SAME AMOUNT WAS REMOVED USING FILTER PRUNING [5].

Pruning amount	Parameters (after pruning)	mAP
0%	1.97 M	60.2
10%	2.01 M	**61.7**
20%	1.91 M	60.9
40%	2.10 M	60.3

It can be seen how using a 10% to 20% larger network and pruning it, produces slightly higher performance than directly training a smaller network. Therefore, for our approach, we propose to build an architecture with 10% of additional parameters and use pruning to bring it back to the correct computational complexity.

E. Input resolution and network complexity

The number of operations of a neural network depends on the square of the input resolution. Therefore, decreasing the input resolution is an effective way to reduce the number of MACs without re-training. By default, Yolo uses a resolution of 640×640, which optimizes performance in the object detection task. However, pose detection often deals with larger objects than simple object detection tasks (e.g., the object is much closer to the camera, as in uses of motion tracking/ VR/ medical images, etc.). We therefore analyzed the performance of our pose detector in two cases:

- Using all the annotations in the COCO dataset, regardless of their size
- Filtering out the smallest objects and keeping only those with size $> 96 \times 96$ pixels, i.e. $> 2.25\%$ of the frame area. This corresponds to the default `Large` metric used for COCO evaluation

In fact, many practical applications do not require processing very tiny objects. Figure 6 shows results for different input resolutions and object sizes.

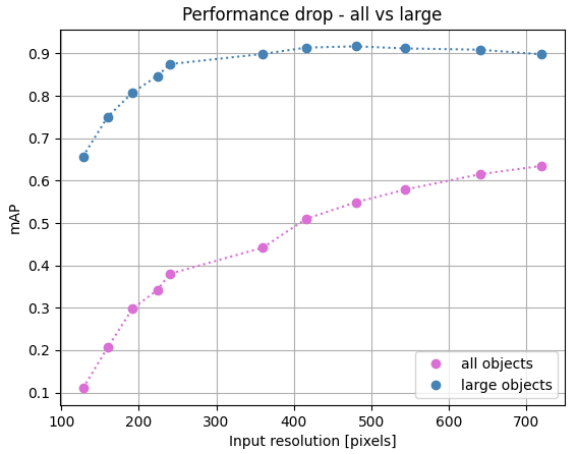

Fig. 6. Comparison of performance drop considering the whole dataset (purple) and removing tiny objects (blue). In real-world applications where the detection of "microscopic" objects is not required, resolutions down to 240×240 can be used without significant performance drops. If detection of tiny targets is required, the suggested tradeoff is 416×416.

Removing small objects allows for significantly lower resolutions (thus lower computational complexity) without significant performance drops. Performance decreases slightly at higher resolutions due to the smaller receptive fields of the final network layers. In particular, an input resolution of 240×240 causes only a drop of $\Delta mAP = 3$ compared to using the full resolution input, while allowing an 86% decrease in operations ($162M$ MAC, Table V). If, on the other hand, detection of tiny objects is needed, a good tradeoff can still be achieved with an input 416×416, which guarantees a mAP greater than 0.5 with less than $500M$ MAC (Table V).

V. RESULTS

Not many approaches targeting pose detection on MCUs address computational requirements similar to the proposed solution. However, some lightweight, easily scalable techniques allow detectors with similar requirements to the proposed solution. Among them, PifPaf [14] is the one that achieves the best performance with very low complexity, using backbones such as MobilenetV3 and ShuffleNetV2. This allows it to scale to the computational requirements of our target device easily.

The following results directly compare the performance of the proposed approach, PifPaf, and the YoloV8 detector on which it is based. It should be considered, however, that while our approach and YoloV8 are end-to-end strategies, PifPaf requires a subsequent heatmap decoding step, the computational cost of which is not negligible.

For comparison with other methods, the Yolo- and PifPaf-based networks were scaled to fall within the $2M$ parameters allowed by the target device. In both cases, all the convolutional filters of the various layers were reduced by the same α factor so that they would fall within the allowed values. Both networks were re-trained with the original training strategy proposed by the respective works [12], [14]. We present the results of our approach using four different input resolutions, optimal for devices with different computational capabilities. As for the other SOTA approaches, we evaluate the input resolution presented in the respective works.

TABLE V
COMPARISON BETWEEN OUR PROPOSED APPROACH AND COMPARABLE STATE-OF-THE-ART DETECTORS. (*) DENOTES APPROACHES THAT REQUIRE ADDITIONAL DECODING OPERATIONS THAT ARE NOT ACCOUNTED FOR IN THE NUMBER OF MACS.

Detector	Parameters	MAC	mAP
Yolov8-pose	2.05 M	3.49 G	27.1
PifPaf (ShuffleNetV2)*	2.04 M	19.5 G	50.0
PifPaf (MobileNetV3)*	1.45 M	22.0 G	47.1
XiNet-pose @720	1.97 M	1.45 G	**63.5**
XiNet-pose @640	1.97 M	1.15 G	61.7
XiNet-pose @416	1.97 M	0.49 G	51.0
XiNet-pose @240	1.97 M	**0.16 G**	38.0

VI. ON-DEVICE TESTING

We evaluated our approach on the target device (`STM32H743`) to verify latency and RAM usage in a real-world application. We used the STM32Cube.Ai tool from STM to bring the network to the device. Table VI details the results of the networks proposed in Section V.

TABLE VI
RESULTS FROM ON-DEVICE TESTING OF THE PROPOSED NETWORKS. (*) MARKS NETWORKS THAT REQUIRED OPTIMIZATIONS FOR RAM USAGE THAT INCREASED LATENCY. (**) MARKS NETWORKS THAT REQUIRED ADDITIONAL EXTERNAL DRAM TO EXECUTE.

Input	**mAP**$_{ALL}$	**mAP**$_{L}$	**Latency**	**RAM**	**Energy**
640×640	61.7	91.4	-	1637 KB**	-
544×544	57.9	91.4	2140 ms*	1050 KB*	274 mJ
416×416	51.0	91.3	779 ms	692 KB	98 mJ
240×240	38.0	87.5	133 ms	232 KB	17 mJ

We evaluated only networks with input resolutions up to 640×640, as this value proved to be the limit of the internal RAM capacity of the chosen MCU (higher resolutions could still be run relying on external DRAM chips, sacrificing latency). For lower resolutions, the approach demonstrated latencies below $800ms$, reaching $133ms$ for the smallest network. In a real-world application, assuming there is no need to locate very tiny targets, this configuration allows for multi-target pose detection at 7.5 fps with minimal performance drop compared to largely more complex solutions. By sacrificing latency for performance, using the `'optimize for RAM usage'` option of STM32Cube.Ai, it was possible to run networks with resolutions up to 544×544, achieving a maximum on-device mAP of 57.9 considering even very small objects, without relying on external DRAM chips.

VII. CONCLUSIONS

This work introduces XiNet-pose, a pose detection algorithm based on neural networks targeting low-resource embedded devices such as microcontrollers. Our primary objective was to ensure seamless scalability and compatibility across different devices. We verified the strategy on a microcontroller `STM32H743`, achieving a maximum mAP of 57.9 on the COCO-keypoint test set, without the need for any external DRAM chip. In real-world scenarios, where detection of extremely small targets ($< 2\%$ of the image area) is not needed, the Xinet-pose method demonstrated excellent trade-offs between performance and complexity at resolutions of 240×240, allowing networks with latencies of $133ms$ and mAPs of 38.7 (whole dataset) / 87.5 (removing tiny objects). These networks enable reasonable performance in real-world pose detection tasks, reducing the complexity by two orders of magnitude compared to the smallest Yolo-pose network.

REFERENCES

[1] Gianluca Amprimo, Giulia Masi, Lorenzo Priano, Corrado Azzaro, Federica Galli, Giuseppe Pettiti, Alessandro Mauro, and Claudia Ferraris. Assessment tasks and virtual exergames for remote monitoring of parkinson's disease: An integrated approach based on azure kinect. *Sensors*, 22(21), 2022.

[2] Gianluca Amprimo, Irene Rechichi, Claudia Ferraris, and Gabriella Olmo. Objective assessment of the finger tapping task in parkinson's disease and control subjects using azure kinect and machine learning. In *2023 IEEE 36th International Symposium on Computer-Based Medical Systems (CBMS)*, pages 640–645, 2023.

[3] Alberto Ancilotto, Francesco Paissan, and Elisabetta Farella. Xinet: Efficient neural networks for tinyml. *2023 IEEE International Conference on Computer Vision (ICCV)*, 2023.

[4] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(1):172–186, jan 2021.

[5] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16091–16101, June 2023.

[6] Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. *ArXiv*, abs/2305.10924, 2023.

[7] Hao-Shu Fang, Jiefeng Li, Hongyang Tang, Chao Xu, Haoyi Zhu, Yuliang Xiu, Yong-Lu Li, and Cewu Lu. Alphapose: Whole-body regional multi-person pose estimation and tracking in real-time. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(6):7157–7173, nov 2022.

[8] Daniel Groos, Heri Ramampiaro, and Espen AF Ihlen. Efficient-pose: Scalable single-person pose estimation. *Applied Intelligence*, 51(4):2518–2533, apr 2021.

[9] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[11] Glenn Jocher. YOLOv5 by Ultralytics, May 2020.

[12] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics, Jan. 2023.

[13] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *ICCV*, pages 2938–2946. IEEE Computer Society, 2015.

[14] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. Pifpaf: Composite fields for human pose estimation. pages 11969–11978, 06 2019.

[15] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. 10 2021.

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*. Springer International Publishing, 2014.

[17] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part XIV*, page 122–138, Berlin, Heidelberg, 2018. Springer-Verlag.

[18] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. Phinets: A scalable backbone for low-power ai at the edge. *ACM Transactions on Embedded Computing Systems*, 21:1 – 18, 2021.

[19] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *ArXiv*, abs/1804.02767, 2018.

[20] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[21] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 05 2019.

[22] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 10096–10106. PMLR, 2021.

[23] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.

[24] Zhe Zhang, Jie Tang, and Gangshan Wu. Simple and lightweight human pose estimation. *CoRR*, abs/1911.10346, 2019.