Low-latency Human Portrait Segmentation Network Optimized for CPU Inference

Dzianis Pirshtuk
Faculty of Applied Mathematics
and Computer Science
Belarusian State University
Minsk, Belarus
ORCID: 0009-0009-7831-1953

***Abstract*—In this paper, we discuss a design of fast and lightweight neural networks for working in real-time under very strict resource constraints and describe a human portrait segmentation method with temporal consistency based on an encoder-decoder architecture with a state-of-the-art CPU optimized PP-LCNet backbone and a custom decoder. Proposed neural network can process about 150-500 frames per second using only a single CPU thread with high accuracy and can be used for virtual background replacements in video conferencing and other augmented reality cases.**

***Index Terms*—Portrait segmentation, image segmentation, neural networks, computer vision, augmented reality.**

I. Introduction

In recent years, significant progress has been made in scaling up GPU capacity for training and inference of neural networks. But many people do not update their devices as quickly, so quite often neural networks must also be used on outdated consumer devices. Even more difficult is the case when it need to process video streams at a speed of 30 frames per second on such devices. Moreover, unlike desktop computers, the energy-saving processors of many laptops and smartphones cannot operate at high efficiency for a long time. Either the processor overheats and throttling, or the battery drains quickly. Also note that moving calculations to the cloud is not a solution due for real-time video processing tasks to costs, high delays in data transmission over the Internet and data privacy concerns. So for tasks such as replacing the background during a video call, person segmentation should only be performed on the user's device. Therefore, the task of developing an architecture for human segmentation in video with minimal usage of computing resources is truly relevant.

In this article we review how to design neural networks for inference under very strict resource constraints and propose a novel neural network architecture that achieves state-of-the-art accuracy on two different datasets while running in 150-500 frames per second (FPS) using only a single CPU thread. In other words, our method utilizes about $6\% - 20\%$ of only one CPU core.

The discussed methods can be partially re-used in other real-time recognition tasks.

II. Neural Network Architecture

We propose to use an asymmetrical encoder-decoder with a neural architecture search optimized efficient encoder, a lightweight decoder, only partial usage of skip-connections and customized feature fusion module unlike a popular U-Net [1] architecture for image segmentation.

The general architecture of our segmentation neural network is illustrated in Fig 1.

For reducing segmentation mask jittering and achieving temporal consistency between video frames using only fully-convolutional neural network without computationally expensive for real-time applications recurrent neural network architectures we propose using a neural network with 2 inputs:

1) RGB image $224 \times 224 \times 3$,
2) Prior mask: segmentation mask $224 \times 224 \times 1$ computed for the previous frame.

Here we follow the original approach proposed [2] with a difference that Google engineers use 4-channel single RGB-prior input. We propose to use 2 separate inputs. This is actually an important difference. Single 4-channel input will be better for GPU inference. On GPU, tensor data is sliced into 4-channels [3]. So using 4-channel input is as fast as using 3-channel input on GPU, and using the second 1-channel prior input will be more expensive. But these is no such restriction for CPU inference, so using 2 inputs has some advantages. Directly mixing the prior data into the decoder leaves the encoder input as a regular image, unlike using the single 4-channel RGB-prior as in articles [2], [4]. Therefore, the encoder weights can be initialized not with random numbers, but with weights obtained during training for another task, such as, for example, classification of the dataset ImageNet. As [5] shows, fune-tuning of pre-trained encoder substantially reduces training time and also helps to prevent over-fitting on small datasets.

We use a part of PP-LCNet-0.5x network [6] as an image encoder. PP-LCNet belongs to MobileNetV3 family of neural network architectures [7]. This family of networks is intended for use on consumer devices and environments with limited computing resources such as personal computers, smartphones, embedding systems and web browsers. Let's review several general principles of optimization neural network architectures for inference with such restrictions which we followed too:

1) Try to never use kernels like 7×7 and larger. Avoid even using of kernels 5×5 [8]. A cascade of several

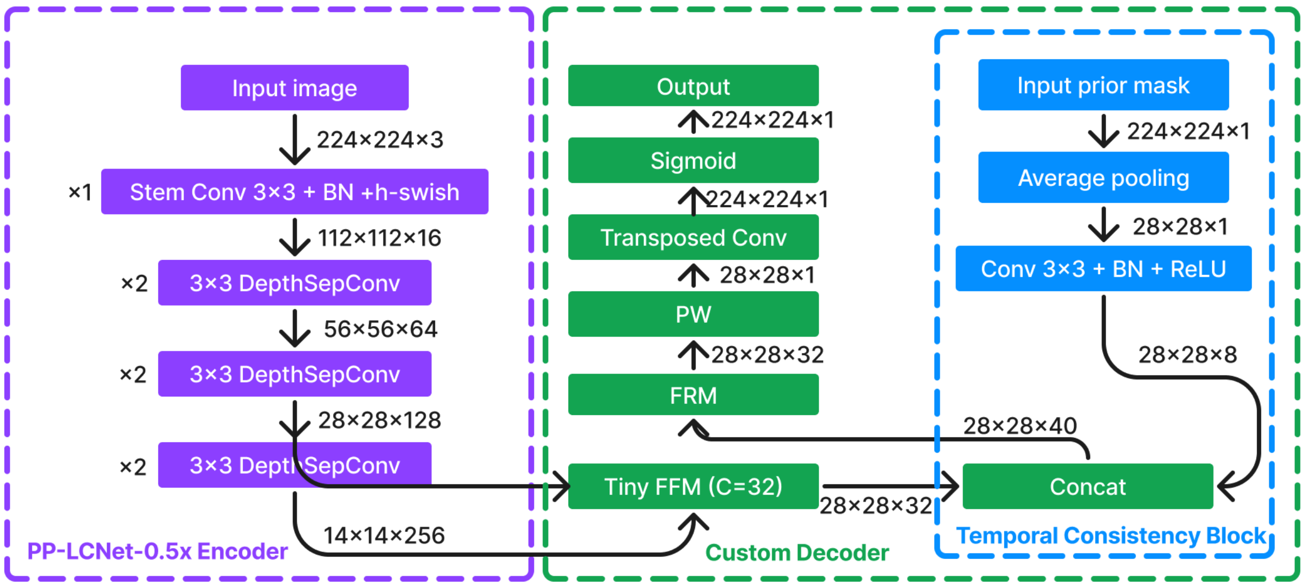

Fig. 1. Portrait Segmentation Architecture. The stem part uses a standart convolution 3×3 with stride 2 for fast downsampling. Following to [6] DepthSepConv means depth-wise convolution 3x3 + batch-normalization + h-swish activation + (point-wise) convolution 1x1 + batch-normalization + h-swish activation. Tiny FFM and FRM mean our Tiny Feature Fusion Module and Feature Refinement Module respectively, detailed below. BN means batch-normalization. Here PW means 1×1 (point-wise) convolution without batch-normalization and with linear ReLU activation.

convolutions 3×3 with non-linear activations can have a same receptive field as a large convolution with more number of non-linearity.

2) Using separable convolutions instead of standard convolutions to reduce number of multiply-add operations and layer parameters (weights) [8].
To apply a standart convolution with kernel size $h \times w$ and n filters with bias parameters to an tensor $H\times W\times C$ we need about $HWnwhC + HWn$ multiply-add operations. But a pair of a depth-wise convolution with kernel-size $h \times w$ and a point-wise convolution 1×1 with n filters with bias parameters requires only about $HWCwh + HWC + HWCn + HWn$ multiply-add operations. So we can expect that separable convolutions will be in $\frac{nwhC+HWn}{Cwh+C+Cn+n}$ times faster. For the most popular case $h = w = 3$ we have theoretical acceleration in $\frac{9nC}{Cn+10C+n} \to 9$ times for large n or C.
And a standart convolution with bias have $nhwC + n$ weights. A pair of separable depth-wise and point-wise operations has $hw + C + nC + n$ weights that is asymptotically better. For popular case $h = w = 3$ we have $9nC + n$ versus $nC + C + n + 9$, or approximately in 9 times less parameters for a separable convolution and large enough n and C.
Note these architectures use an additional not-linear activation between depth-wise and point-wise convolutions following the greedy strategy of non-linearity number maximization. And also note that a sequence of operations ”point-wise convolution + activation + depth-wise convolution + activation + point-wise convolution + activation” can be considered as a kernel-trick.

3) Using residual connections [9]. Sure, a separable convolution is only a rough approximation of a standart convolution, but it helps to train computationally cheap deeper neural networks with residual connections with larger number of non-linear activation functions too. Here residual connections help avoid gradient decay, separable convolutions reduce using of computational resources and memory usage.

4) Using squeeze-and-excitation (SE) block (Fig. 2) as a channel-wise attention [7], [10], [11] to get better hidden feature representations. It helps to evaluate what filters are useful now and turn-off unuseful ones as potentially noisy in order to get cleaner signal.

5) Activation function hard-swish [7] $\text{h-swish}(x) = \frac{x\,\text{ReLU6}(x+3)}{6} = \frac{x\min(\max(x+3,0),6)}{6}$. Hard-swish is a type of activation function based on activation function swish $= x \cdot \text{sigmoid}(x)$ that helps to avoid a common problem known as “dying ReLU,” [12] but replaces the computationally expensive sigmoid with a piece-wise linear analogue (hard-sigmoid). “Dying ReLU” makes a lot of computing inside neural network useless, so it’s very important to avoid this if we are looking the best architecture for fast and accurate inference. So we can see that h-swish is a trade-off between simple and dangerous $\text{ReLU}(x) = \max(x, 0)$ and too complex swish.

6) Cost of computing and memory operations depends on device architecture and other restrictions (as availability of Single Instruction/Multiple Data (SIMD) commands

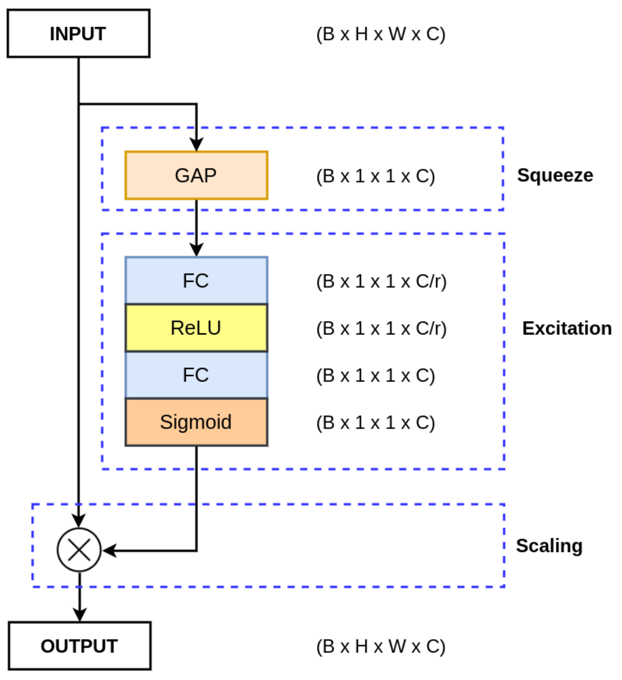

Fig. 2. A detailed diagram of the Squeeze and Excitation Network with proper dimensions and the different operations [11].

in WebAssembly virtual machine for inference in web browser). h-swish, SE-block works well on CPU due to CPU memory cache architecture, but they are not so good on GPU due to GPU huge latency of memory operations [7].

7) All batch-normalization must be fused in inference mode. We use batch-normalization layers after convolutions before non-linear activations. So all our batch normalization operations can be folded in convolution layers [13]. Tensorflow framework fuses such operations during export model to Tensorflow Lite format automatically.

According to benchmarks [6], PP-LCNet currently provides one of the best accuracy-latency trade-off for CPU inference on ImageNet classification. Configuration parameters of PP-LCNET were found by Baidu research engineers using automatic neural architecture search (NAS) [6] and outperform original MobileNetV3, proposed in [7].

For decoder part we propose to use Tiny Feature Fusion Module (Tiny FFM) (Fig. 3). PP-LCNet is NAS-optimized low-latency architecture that evaluates effeciently a lot of encoding features. But reverse upsampling of large number feature maps will be expersive. Using a lot of channels in fully-connected part FFM/SE-block is expensive too. To adjust PP-LCNet to lightweight decoding we add to Feature Fusion Module (FFM) block [14] at the beginning 2 point-wise convolution layers for compression of feature space up to 32 feature maps and call this modification as Tiny Feature Fusion Module (Tiny FFM).

We merge Tiny FFM image features with features extracted from the prior mask using simple tensor concatenation. To improve representation of these information we use Feature Refinement Module, described on the figure 4.

The last 2 layers are a pair of

1) 1-filter point-wise convolution that gives a "dirty segmentation mask" $28\times28\times1$ without batch-normalization and with linear activation,
2) 1-filter transposed convolution with kernel 16×16, a stride equal to 8 and sigmoid output activation for prediction of segmentation mask probabilities.

Note that we use transposed convolutions with a kernel size that is divided by a stride to avoid checkerboard artifacts due to overlapping issues. As noted in [15], such operation is equivalent to "sub-pixel convolution," a technique which has recently had success in image super-resolution. That is, we predict the segmentation mask at low resolution 28×28 and then upscale it by 8x with sub-pixel precision to 224×224.

We think such extremely light decoder provides good results for following reasons:

1) Avoiding of overfitting. Ground truth segmentation masks are not pixel-perfect annotated. As noted in [16] Google R&D engineers asked 7 annotators to re-annotate selfie segmentation masks from their validation dataset and got inter-annotator 98.74% intersection-over-union agreement. It's very difficult to draw accurate curved border between classes, especially in the area of hair, beards, etc. So we have about 1.25% of noise on borders of segmentation masks, and using symmetric U-Net-like encoder-decoder [1], [14] we risk of tuning neural network on the noise and as the result running into the problem of border jitter between video frames.
2) Regularization. We can assume that a portrait of a person is a simply connected region, if we neglect the small details of the hair, the edges of wide frames of glasses, etc. So using aggressive 8 times upsampling we prevent the prediction of false-positive small regions.
3) Re-calculation of class boundaries using the prior input. The impact of additional feature maps extracted from the previous frame mask more explicit for 8 times upsampling in comparing with deeper decoders. The task of refining the boundary relative to the previous frame becomes simpler, and the inter-frame jitter of the predicted masks is reduced.
4) Efficient feature extraction by the state-of-the-art encoder architecture. PP-LCNet avoid dying ReLU problem using h-swich activation and use a lot of filters effectively. So we need only correctly decode a lot of encoder features from the bootle-neck. And proposed Tiny FFM block helps us.
5) Our encoder can initially extracts good features about human details and background items because it was pre-trained on ImageNet.

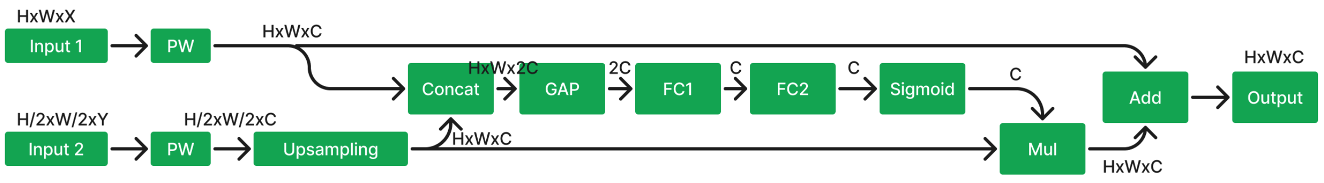

Fig. 3. A detailed view of Tiny Feature Fusion Module. GAP means Global Average Pooling. FC1 and FC2 mean fully-connected layers. FC1 layer has ReLU activation. Concat, Mul and Add means channel concatenation, multiply and adding of 2 tensors respectively. Note that we used bilinear interpolation for tensor upsampling in our experiments. Here PW means 1×1 (point-wise) convolution with batch-normalization and ReLU activation.

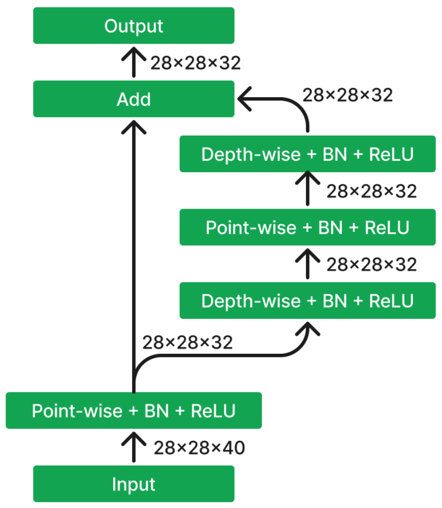

Fig. 4. Feature Refinement Module.

Note that we was also inspired by a neural network architecture proposed in [17]. In that article there was used a neural network with custom encoder and single-layer decoder for human portrait segmentation. But our results significantly exceed the accuracy, number of floating-point operations, and number of parameters in the network. The performance of a lightweight decoder depends significantly on the encoder's ability to compress spatial information into deep feature knowledge representations. Therefore, selection of the right encoder architecture is also an important part of the solution design.

III. Data and training

A. Datasets

Almost all articles about portrait segmentation [17]–[20] use EG1800 dataset [21]. This dataset containts 1800 URLs to Flickr images and corresponding segmentation masks. Since several links to images are already invalid, we use only 1309 images as a training set and 270 images for testing purposes.

Note that EG1800 is a very small dataset and results on this dataset may not generalize well to real-world applications. For comparing with previous articles we train our models on this dataset too, but to show impact of our propositions on segmentation metrics we also use in our additional experiments larger EasyPortrait dataset [22]. This dataset includes 20,000 photos of ordinary people predominantly indoor with fine-grained segmentation masks. All images are collected and labeled using crowdsoursing platforms. All annotated images were devided into training, validation and testing sets, 14,000, 2,000 and 4000 samples, respectively. It's very important that subjects from all 3 sets are not intersecting to avoid data leaks.

B. Data augmentation

In our experiments we use the following basic data augmentation tricks:

- Random horizontal flip.
- Random rotate, scale, padding and crop.
- Brightness and contrast jitter, random gamma correction.
- Approximation of motion blur implemented in the Python library Albumentations [23].

Additionally to train neural networks with the prior mask as the 2nd input we randomly select one of following variants:

1) Empty previous mask (40% of frames). It simulate trains the network to work correctly for the first frame and new objects in scene, as well as boost importance of encoder features.
2) Small affine transformations or/and small elastic transformations (`ElasticTransform` in [23]) of the ground truth mask to train the network to propagate and adjust to the previous frame mask and emulate fast camera movements and rotations (35% frames).
3) Major affine transformations of the ground truth mask to train the network to recognize inadequate masks and discard them (5% frames).
4) Self-prediction of grid distorted image (`GridDistortion` in [23]) to train the network propagate and adjust to the previous frame its own prediction and solve a problem variable shift of prior mask input. We found useful using this variant for 30% samples. These case have a principal difference between our training procedure with prior masks and training procedure, proposed in [2] that use similar way with previous 3 variants only. Note that without this important variant mask propagation doesn't train correctly at all in our experiments.

As we show below, this technique improves image segmentation metrics insignificantly, however, perceptual quality of video segmentation improves significantly.

C. Loss Function and Metrics

To evaluate the models we will use mean intersection-over-union (IoU) metric between ground truth and predicted masks $\frac{TP}{TP+FP+FN}$, where TP, FP and FN are number of true positive, false positive and false negative pixels respectively.

Let y_t be a ground truth mask and y_p be a predicted mask. We use a sum of 3 following loss functions as a target loss function for back-propagation optimization.

1) Binary cross-entropy of per-pixel classification.
2) Differentiable jaccard (IoU) loss as an approximation of target IoU metric (1):

$$\text{jaccard\_loss} = 1 - \frac{\sum y_t * y_p}{\sum y_t + \sum y_p - \sum y_t * y_p + 1}. \tag{1}$$

3) Differentiable boundary jaccard loss [4], [18] to increase weight of hard pixels for classification in training procedure.

D. Training Method

We perform all experiments on NVIDIA RTX 2080 Ti GPU. All models were trained with Tensorflow/Keras framework in mixed precision mode for 30 epochs using a batch size 32 and Adam optimizer. The initial learning rate is set as 0.001 and multiplied by 0.9^{epoch} for all experiments.

IV. Evaluation Results

We measure FPS on 3.60GHz Intel Core i7-7700 with Linux and on Macbook Pro 16 (2019) with 2.6 Ghz 6-core Intel Core i7 in Google Chrome 116. For measure frames per second (FPS) on native x64 platform with TFLite XNNPACK Delegate [24] we use official TFLite performance benchmark [25] for Linux. We use only one thread for all measurements because we're not allowed to use all CPU resources for neural network inference in real life user applications. In the Table I you can see FPS of steady inference state for 1000 runs on random inputs. For measure frames per second (FPS) in web browser with TFLite XNNPACK Delegate with enabled SIMD instructions we use self-written benchmark based on TF.JS TFLite API [26] with averaging of run durations too.

Results of all computational experiments are given in the Table I.

As easy to see, starting learning from pre-trained encoder instead of random initialization of weights improves IoU by 0.7% for EG1800 and by 0.86% on EasyPortrait.

PP-LCNet as a backbone outperforms MobileNetV3 and MobileNetV2, especially with native platform computing. EfficientNet provides better segmentation quality but is too slow.

Our algorithm is as accurate as RCRNet [20], but contains 2 times less floating-point operations and 4 times less parameters. SINet [19] is too slow and less accurate. NAS optimized backbones with h-swish activations and some SE-block like MobileNetV3 or PP-LCNet perform better that custom architecture with Spatial Squeeze Module proposed in [19].

Examples of segmentation images from EasyPortrait test dataset are demonstated on the Figure 5.

V. Conclusion

This paper presents an algorithm of video portrait segmentation in real-time without GPU computing. The experiments performed on EG1800 and EasyPortrait datasets demonstrate the effictiveness of using pre-trained modern NAS-optimized encoders for image segmentation.

The proposed neural network architecture runs much faster than minimal requirements for processing 30 fps video stream even in web browsers via WebAssembly virtual machine. This allows the proposed algorithm to be used on a wide variety of consumer devices, including some older ones. The absence of the need to use a large amount of computing resources for neural network calculations makes it possible to process the video stream, including in parallel with other resource-intensive calculations, which can be useful in real applications. An example of such a case would be listening to video streaming with the replacement of the virtual background during an online computer game. In addition, a small utilization of processor resources allows you to reduce the rate of discharge of the battery of a laptop or mobile phone, which is important for long video conferences.

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-NET: Convolutional Networks for Biomedical Image Segmentation," in Lecture Notes in Computer Science, 2015, pp. 234–241. doi: 10.1007/978-3-319-24574-4_28.

[2] A. Tkachenka et al., "Real-time hair segmentation and recoloring on mobile GPUs.," arXiv (Cornell University), Jul. 2019, [Online]. Available: https://arxiv.org/pdf/1907.06740.pdf

[3] "GPU delegates for TensorFlow Lite," TensorFlow, [Online]. Available: https://www.tensorflow.org/lite/performance/gpu

[4] "Creating better virtual backdrops for video calling, remote presence, and AR," ai.meta.com. https://ai.meta.com/blog/creating-better-virtual-backdrops-for-video-calling-remote-presence-and-ar/ (accessed Sep. 07, 2023).

[5] V. Iglovikov and A. A. Shvets, "TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation," arXiv (Cornell University), Jan. 2018, [Online]. Available: http://export.arxiv.org/pdf/1801.05746

[6] C. Cui et al., "PP-LCNET: a lightweight CPU convolutional neural network," arXiv (Cornell University), Sep. 2021, [Online]. Available: https://arxiv.org/pdf/2109.15099.pdf

[7] A. Howard et al., "Searching for MobileNetV3," 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), 2019, pp. 1314-1324, doi: 10.1109/ICCV.2019.00140.

[8] A. Howard et al., "MobileNets: efficient convolutional neural networks for mobile vision applications," arXiv (Cornell University), Apr. 2017, [Online]. Available: http://export.arxiv.org/pdf/1704.04861

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018. doi:10.1109/cvpr.2018.00474

[10] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-Excitation networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 8, pp. 2011–2023, Aug. 2020, doi: 10.1109/tpami.2019.2913372.

[11] N. Tomar, "Squeeze and excitation networks," Idiot Developer, Oct. 2022, [Online]. Available: https://idiotdeveloper.com/squeeze-and-excitation-networks/

[12] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," arXiv (Cornell University), Oct. 2017, [Online]. Available: https://arxiv.org/pdf/1710.05941.pdf

[13] "GitHub - ZFTurbo/Keras-inference-time-optimizer: Optimize layers structure of Keras model to reduce computation time," GitHub. https://github.com/ZFTurbo/Keras-inference-time-optimizer#formulas

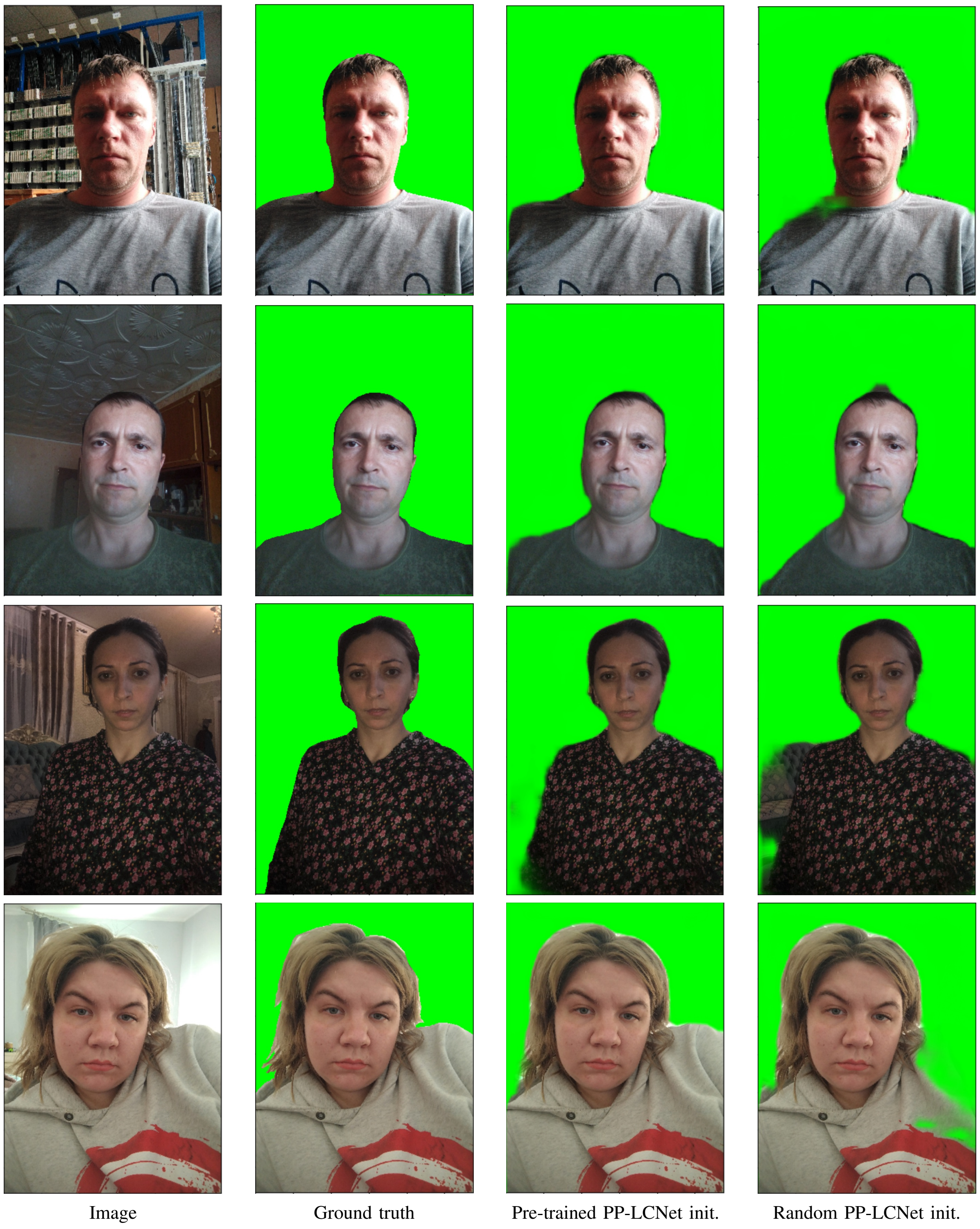

Fig. 5. Examples of segmentation images from EasyPortrait test dataset.

TABLE I
EVALUATION RESULTS

Method	**Params (M)**	**Flops (M)**	**IoU (%)**		**FPS**	
			EG1800	***EasyPortrait***	***Native***	***Web***
PortraitNet [18]	2.080	325	95.99	–	–	–
HLB [17]	0.6	700	94.6	–	–	–
SINet [19]	0.087	64	94.81	–	175	68
RCRNet [20]	0.610	190	96.31	–	–	–
Our proposed solution described above	0.138	91	96.31	96.88	475	167
Without the 2nd input with mask prior	0.138	91	96.15	96.87	475	167
ImageNet pre-trained MobileNetV3-1x backbone instead PP-LCNet-0.5x	0.205	81	96.10	96.52	341	157
ImageNet pre-trained MobileNetV2-0.35x backbone instead PP-LCNet-0.5x	0.102	93	96.03	96.64	330	152
Randomly initialized backbone instead ImageNet pre-trained	0.138	91	95.61	96.02	??	167
ImageNet pre-trained EfficientNetB0 [27] backbone instead PP-LCNet-0.5x	0.102	334	96.47	97.56	111	35

[14] C. Yu, J. Wang, C. Peng, C. Gao, and G. Yu, "BISENET: Bilateral Segmentation Network for Real-Time Semantic Segmentation," in Lecture Notes in Computer Science, 2018, pp. 334–349. doi: 10.1007/978-3-030-01261-8_20.

[15] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and Checkerboard artifacts," Distill, vol. 1, no. 10, Oct. 2016, doi: 10.23915/distill.00003.

[16] S. Pisarchyk, T. Hou, and K. Raveendran, "MLKit Seflie Segmentaton Model Card," 2021. Accessed: Sep. 14, 2023. [Online]. Available: https://developers.google.com/static/ml-kit/images/vision/selfie-segmentation/selfie-model-card.pdf

[17] Y. Li, A. Luo and S. Lyu, "Fast Portrait Segmentation With Highly Light-Weight Network," 2020 IEEE International Conference on Image Processing (ICIP), Abu Dhabi, United Arab Emirates, 2020, pp. 1511–1515, doi: 10.1109/ICIP40778.2020.9190790.

[18] S.-H. Zhang, X. Dong, H. Li, R. Li, and Y.-L. Yang, "PortraitNet: Real-time portrait segmentation network for mobile device," Computers & Graphics, vol. 80, pp. 104–113, 2019. doi:10.1016/j.cag.2019.03.007

[19] H. Park, L. L. Sjösund, Y. Yoo, N. Monet, J. Bang and N. Kwak, "SINet: Extreme Lightweight Portrait Segmentation Networks with Spatial Squeeze Modules and Information Blocking Decoder," 2020 IEEE Winter Conference on Applications of Computer Vision (WACV), Snowmass, CO, USA, 2020, pp. 2055–2063, doi: 10.1109/WACV45572.2020.9093588.

[20] R. Yuan, Y. Cheng, Y. Yan and H. Liu, "Real-time Segmenting Human Portrait at Anywhere," 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Vancouver, BC, Canada, 2023, pp. 2197–2203, doi: 10.1109/CVPRW59228.2023.00213.

[21] X. Shen et al., "Automatic portrait segmentation for image stylization," Computer Graphics Forum, vol. 35, no. 2, pp. 93–102, May 2016, doi: 10.1111/cgf.12814.

[22] A. Kapitanov, K. Kvanchiani, and S. Kirillova, "EasyPortrait – Face Parsing and Portrait Segmentation Dataset," arXiv (Cornell University), Apr. 2023, doi: 10.48550/arxiv.2304.13509.

[23] A. V. Buslaev, V. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: fast and flexible image augmentations," Information, vol. 11, no. 2, p. 125, Feb. 2020, doi: 10.3390/info11020125.

[24] "Accelerating TensorFlow Lite with XNNPACK Integration." https://blog.tensorflow.org/2020/07/accelerating-tensorflow-lite-xnnpack-integration.html

[25] "Performance measurement," TensorFlow, [Online]. Available: https://www.tensorflow.org/lite/performance/measurement

[26] "TensorFlow.js TFLite API." https://js.tensorflow.org/api_tflite/0.0.1-alpha.9/

[27] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," International Conference on Machine Learning, pp. 6105–6114, May 2019, [Online]. Available: http://proceedings.mlr.press/v97/tan19a/tan19a.pdf