BlitzMask: Real-Time Instance Segmentation Approach for Mobile Devices

Vitalii Bulygin
Samsung R&D Institute Ukraine

Dmytro Mykheievskyi
Samsung R&D Institute Ukraine

Kyrylo Kuchynskyi
Samsung R&D Institute Ukraine

Abstract

We propose a fast and low complexity anchor-free instance segmentation approach BlitzMask. For the first time, the approach achieves competitive results for real-time inference on mobile devices. The model architecture modifies CenterNet by adding a new lite head to the CenterNet architecture. The model contains only layers optimized for inference on mobile devices, e.g. batch normalization, standard convolution, depthwise convolution, and can be easily embedded into a mobile device. The instance segmentation task requires finding an arbitrary (not a priori fixed) number of instance masks. The proposed method predicts the number of instance masks separately for each image using a predicted heatmap. Then, it decomposes each instance mask over a predicted spanning set, which is an output of the lite head. The approach uses training from scratch with a new optimization process and a new loss function. A model with EfficientNet-Lite B4 backbone and 320×320 input resolution achieves 28.9 mask AP at 29.2 fps on Samsung S21 GPU and 28.0 mask AP at 39.4 fps on Samsung S21 DSP. This sets a new speed benchmark for inference for instance segmentation on mobile devices.

1 INTRODUCTION

Several modern Computer Vision tasks have been ported to mobile devices, such as Semantic Segmentation (Türkmen and Heikkilä, 2019), (A. Howard et al., 2019), (Orsic et al., 2019), Object Detection (Cai et al., 2020), (Vasu et al., 2022), (A. Howard et al., 2019), and Human Pose Estimation (Bazarevsky et al., 2020), by inventing new computationally light methods for each task. To the best of our knowledge, Instance Segmentation tasks still remain a challenge, and current methods do not achieve sufficient accuracy on these tasks for real-time mobile applications (Minaee et al., 2021), (H. Liu et al., 2020), (Zeng and Sabah, 2020). In the paper, we are addressing this gap with an approach that operates on low-resolution input images and uses layers well supported on mobile devices, e.g. no Deformable Convolution (Dai et al., 2017), Dynamic Convolution (X. Wang, Zhang, et al., 2020), etc. In addition, as it is hard to implement custom operations on mobile devices, we use simple post-processing and only TFLite supported and optimized layers.

Adapting existing instance segmentation approaches to mobile We were unable to find fully comparable benchmarks with performance tested on regular mobile devices for instance segmentation. Why is porting an instance segmentation approach to mobile such a problem?

A naive solution is to simply use an existing instance segmentation approach with light backbones, such as MobileNet, as well as low-resolution input to achieve real-time speed on mobile. Suppose we are adapting well-known SOLO (X. Wang, Kong, et al., 2020), or CenterMask (Y. Wang et al., 2020), or a similar approach to mobile. A single output from SOLO or CenterMask has size $S \cdot S \cdot W \cdot H$, where $S = 32$, $W = H = 128$. The corresponding backbones have 256 output channels, so that a final 3×3 convolution requires $N = 3 \cdot 3 \cdot 256 \cdot W \cdot H \cdot S \cdot S = 38.7$ billion multi-adds (MAdds) - this is only for a single layer. The total number of operations for SOLO or CenterMask will also include calculations by the backbone and remaining head layers. Meanwhile, real-time computation for mobile object detection MobileNetV2-SSDLite requires less than 1 billion MAdds for 300×300 input size (Cai et al., 2020), (Sandler et al., 2018a). This makes a single layer in the SOLO or CenterMask head nearly 40 times heavier than the entire MobileNetv2-SSDLite, and therefore it is impractical to use such approaches on mobile. The authors of SOLOv2 (X. Wang, Zhang, et al., 2020) greatly reduced the computational complexity of the model head using dynamic convolution. However, this is a non-standard layer and its adaption is currently not available for mobile; it is a complex task to produce such an adaptation.

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

YOLACT (Bolya et al., 2019a), YOLACT++ (Bolya et al., 2019b) and similar anchor-based instance segmentation approaches have two main problems: 1) Non-maximum Suppression (NMS), and 2) resolution-specific anchor parameters at each pyramid level. The former has been addressed by GPU-optimized fast-NMS (Bolya et al., 2019a), but it is not available for mobile. The latter brings up a non-trivial task to change input resolution in YOLACT, which can dramatically decrease accuracy. For example, the authors Bolya et al. (2019a) report $AP^{mask} = 29.8$ for 550×550, and $AP^{mask} = 24.9$ for 400×400 input, both with ResNet-101 backbone; our own YOLACT training for 320×320 input size gives only $AP^{mask} = 20.1$ with ResNet-101 backbone. The lightweight YOLACT modification named mobileYOLACT (J. Lee, S. Lee, and Ko, 2021) has low input resolution 320×320 and is designed for mobile environments. However, the best result of the mobileYOLACT (J. Lee, S. Lee, and Ko, 2021) is $AP^{mask}_{50} = 23.0$, which is too low, because AP^{mask} is usually near twice smaller than AP^{mask}_{50} and as we show below our approach is more than twice more accurate than mobileYOLACT and has even faster speed on the mobile device considered in the paper (J. Lee, S. Lee, and Ko, 2021). All these facts prove that using a lightweight backbone and low input resolution for YOLACT approach leads to too low accuracy. Furthermore, YolactEdge (H. Liu et al., 2020) uses TensorRT optimization for 550×550 input with MobileNetV2 backbone to achieve real-time calculation on Jetson AGX Xavier with accuracy $AP^{mask} = 20.8$. However, this model is still too heavy for mobile devices, and we show better accuracy for real-time applications on mobile using the method described below.

In contrast to semantic segmentation, the instance segmentation task has a variable number of instance masks that have to be presented by an output tensor with a fixed size. To resolve this problem we express instance masks as linear combinations of fixed size spanning set with instance-specific coefficients. Our proposed method extends CenterNet by adding instance masks head in a similar way as YOLACT (H. Liu et al., 2020) extends YOLOv3 (Redmon and Farhadi, 2018), so that its numerical complexity is comparable with CenterNet (Zhou, D. Wang, and Krähenbühl, 2019). In contrast to the YOLACT (H. Liu et al., 2020), our output head is less computationally expensive and anchor-free, and our post-processing does not require Non-Maximum Suppression (NMS).

The main contributions of our work are as follows:

- We propose a novel single-stage anchor-free instance segmentation method for real-time execution on mobile devices
- In our approach, CenterNet (Zhou, D. Wang, and Krähenbühl, 2019) object detector is just complemented by a lightweight instance mask head, which requires significantly fewer resources than the entire model.
- We implement a new loss function to train anchor-free single-stage model for instance segmentation
- We propose a simple architecture for porting to mobile devices. It achieves satisfactory accuracy for real-time applications on mobile devices.
- We develop a new training process with learning rate and stochastic gradient momentum schedules that are dependent on current loss function values.
- Our method even with input resolution 320×320 achieves better accuracy and speed then Yolact-Edge (H. Liu et al., 2020) with the same backbone and input resolution 550×550.

The paper is organized as follows. Section 2 reviews related work. In Section 3 describes the BlitzMask approach. Section 4 describes our new model training process and demonstrates results from our modeling approach.

2 RELATED WORK

Recently, real-time instance segmentation approaches have achieved great speed and accuracy on a desktop GPU such as GTX 2080 TI, Tesla V100, or Titan XP (Bolya et al., 2019a), (Y. Wang et al., 2020), (X. Wang, Kong, et al., 2020), (H. Chen et al., 2020), (Xie et al., 2020), (X. Wang, Zhang, et al., 2020), (Bolya et al., 2019b), but its accurate and fast implementation still remains a challenge for mobile devices. An anchor-based single-stage approach YOLACT (Bolya et al., 2019a) is one of the first real-time instance segmentation methods. It extends the object detector YOLOv3 (Redmon and Farhadi, 2018) by adding two heads for prototype masks and mask coefficients, so that instance masks are computed as linear combinations of the prototype masks with the normalized coefficients for each anchor box. The method achieves 29.8 mAP on MS COCO (T.-Y. Lin, Maire, et al., 2014) at 33.5 fps on NVIDIA's Titan Xp graphics card.

A follow-up approach YOLACT++ (Bolya et al., 2019b) uses deformable convolutions (Dai et al., 2017), optimized prediction heads and a fast mask re-scoring head. These optimizations improve mask performance to 34.4 mAP on MS COCO, but reduce speed to 27.3 fps on Titan Xp.

In contrast to the YOLACT approaches, SOLO (X. Wang, Kong, et al., 2020) implements instance segmentation directly, without any dependence on box detection. It divides a picture into an $S \times S$ grid and decouples the original mask prediction into semantic category prediction and instance mask prediction. The former indicates semantic class probabilities for each cell and has $S \times S \times C$ values, where C is

the number of object classes. While the latter has $H \times W$ values for each cell and $S \times S \times H \times W$ values in total, where H is the output height, W is the output width. Each category prediction cell $(i, j), i, j = 0, 1, \ldots, S-1$ is associated with the k^{th} channel of the instance mask prediction, where $k = i \cdot S + j$. This structure allows SOLO to predict a variable number of instances using a fixed number of channels. However, convolution layers with $S^2 \times H \times W$ output make SOLO head computationally expensive. This slows down processing to 22.5 fps on NVIDIA's Tesla V100 with 34.2 mAP on MS COCO.

A greatly accelerated SOLOv2 (in comparison to SOLO) (X. Wang, Zhang, et al., 2020) splits the mask prediction into mask kernel and mask feature map prediction, so that the output size decreases from $S^2 \times H \times W$ to $2S \times H \times W$. Each mask kernel is convolved with a feature map to get an instance mask. The performance speed improves to 46.5 fps on Tesla V100 with 34.0 mAP on MS COCO.

Another approach CenterMask (Y. Wang et al., 2020) is an anchor-free method that adds two heads to CenterNet (Zhou, D. Wang, and Krähenbühl, 2019): 1) a head for local instance information with output size $S^2 \times H \times W$, and 2) a head for semantic segmentation on the whole image with output size $H \times W$. The two outputs are assembled together to produce final instance masks. Similarly to SOLO approach, the first head is computationally expensive, resulting in only 25.2 fps with 32.5 mAP (the CenterMask authors do not report computational machine specifications in their paper).

PolarMask approach (Xie et al., 2020) approximates instance mask contours using 36 points derived intersecting 36 uniformly emitted rays from the object center with the instance mask contour. This method is a generalization of the FCOS object detection (Tian et al., 2019), where bounding boxes are presented as simplest masks with only 4 directions. The contour approximation in PolarMask can be rough, and increasing the count of emitted rays does not always converge to the exact instance mask contour. While PolarMask is a simple and flexible framework, it only achieves 22.9 mAP on MS COCO at 26.3 fps on Tesla V100, which is worse than SOLO (X. Wang, Kong, et al., 2020), YOLACT (Bolya et al., 2019a), CenterMask (Y. Wang et al., 2020) and other modern one-stage instance segmentation approaches.

There are a few studies focusing on real-time instance segmentation methods for mobile devices. For example, YolactEdge (H. Liu et al., 2020) improves YOLACT (Bolya et al., 2019a) by applying TensorRT optimization and using MobilenetV2 (Sandler et al., 2018b) architecture. It achieves 20.8 mAP on MS COCO at 35.7 fps on Jetson AGX Xavier. However, Jetson AGX Xavier device is much faster than regular mobile GPUs, e.g. it is about several times faster than Samsung Adreno 660 GPU S21 (see (Ignatov et al., n.d.), NVIDIA specification website (*Jetson benchmarks* 2022), (*Mobile Ranking* 2022)).

The instance segmentation approach mobileYOLACT (J. Lee, S. Lee, and Ko, 2021) is designed for mobile environments. It modifies YOLACT (Bolya et al., 2019a) by using a lightweight backbone, depthwise separable convolution, a simplified prototype mask generation branch and UINT8 quantization. The mobileYOLACT (J. Lee, S. Lee, and Ko, 2021) has a speed of 21 FPS on Samsung Galaxy S20 with $AP^{mask}_{50} = 23.0$ (authors do not present AP^{mask} values in (J. Lee, S. Lee, and Ko, 2021)).

The lightweight approach EOLO (Zeng and Sabah, 2020) is an anchor-free instance segmentation that works at 16 fps on Raspberry Pi4 with Google Coral USB Accelerator with 11.7 mAP accuracy on MS COCO.

Lastly, we wanted to mention a real-time portrait instance segmentation approach for mobiles (L. Zhu et al., 2019) that generates three outputs: person semantic segmentation, person bounding boxes and superpixels. When there is no overlap among the bounding boxes, the first and second outputs are sufficient for instance segmentation. Otherwise, the method utilizes the third output to resolve occlusions or overlapping regions. It is a challenging task to extend this approach to 80 MS COCO classes or similar datasets with a large number of classes.

3 BLITZMASK

3.1 Model architecture

We propose a new single-stage anchor-free instance segmentation method named BlitzMask. Its architecture is based on CenterNet (Zhou, D. Wang, and Krähenbühl, 2019) and FPN (T.-Y. Lin, Dollár, et al., 2017) (Fig. 1). When implemented for real-time inference on mobile devices, the architecture uses separable convolution (A. G. Howard et al., 2017) with bilinear upsampling instead of a combination of deformable (Dai et al., 2017) and transpose convolutions as described by Zhou, D. Wang, and Krähenbühl, 2019. The reason is that, deformable convolution is not supported, and transpose convolution has high latency on mobile devices (Chiang et al., 2020).

Note that we use only well-known layers optimized for mobile devices on TFlite such as Batch Normalization (Ioffe and Szegedy, 2015), depthwise convolution (A. G. Howard et al., 2017) and convolutional layer (LeCun et al., 1998) . Model output consists of Spanning Set B, Coefficients Λ, Heatmap Y and Size D. The output spatial dimensions are four times less than the input spatial dimensions, channel number $K = 32$ (Fig. 1); we found that this selection provides the best trade off between speed and accuracy in a series of experiments on MS COCO dataset. Furthermore,

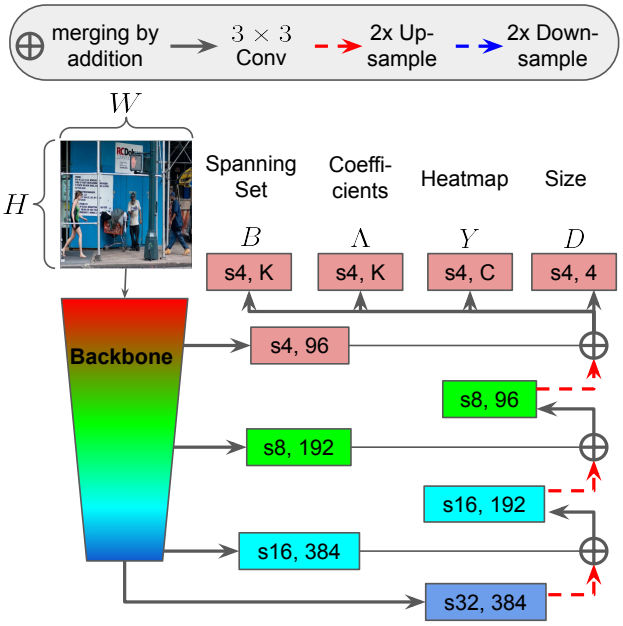

Figure 1: BlitzMask architecture. W and H are input width and height, respectively, C is a number of classes, K is number of channels for both Spanning Set and Coefficients. The solid lines denote 3×3 depthwise separable convolution (A. G. Howard et al., 2017). $\bigoplus$ symbol stands for element-wise summation. The dashed line is bi-linear upsampling by a factor of 2. The labels inside rectangles denote the corresponding stride and number of channels.

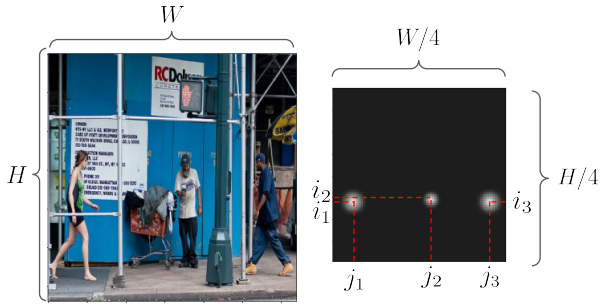

Figure 2: Left is the original image resized to the model input size W, H, right is the single channel of the ground truth for the output Heatmap Y of size $W/4$, $H/4$ (shown not to scale). The Heatmap contains scaled Gaussian distributions centered at the object bounding box center locations (i_1, j_1), (i_2, j_2), (i_3, j_3)

.

we do not use any activation functions for the outputs, except for the sigmoid function for Heatmap Y.

Each channel c of the ground truth for Heatmap Y contains probability density-like function for bounding box center locations for objects of class c (see Fig. 2). So that if a ground truth bounding box center for an object of class c is located at i_0, j_0, then the c^{th} channel of Heatmap ground truth contains scaled Gaussian distribution with maximum value 1 at its center i_0, j_0 with object size-adaptive standard deviation σ defined as described by Law and Deng (2018), Zhou, D. Wang, and Krähenbühl (2019).

Ground truth for Size D contains distances from the object bounding box centers to its four sides $[w_l, h_t, w_r, h_b]$ similarly to Liu et al. (Z. Liu et al., 2020). Then, given a Gaussian center in row i, column j for channel c from the Heatmap Y, a bounding box for an object of class c can be computed as follows:

$$[x, y, w, h] = [4 \cdot j - w_l, 4 \cdot i - h_t, w_l + w_r, h_t + h_b], \quad (1)$$

where (x, y) is the box left top point, (w, h) are the box width, height, $[w_l, h_t, w_r, h_b] = D_{i,j}$. As Heatmap Y has four times lower resolution than model input size, we scale its Gaussian peak locations by a factor of 4 above. Furthermore, due to the lower resolution of the Heatmap, bounding boxes require specifying four sides, instead of two (i.e. width and height), to account for possible offsets. This representation allows placing Gaussian center location everywhere inside bounding box (not only in the box center) and helps in our experiments with modification of Gaussian center location in Section 4.

We express an instance mask as a linear combination of Spanning Set B with corresponding Coefficients Λ (more details are provided in the following sections). We only define ground truth for instance mask, and do not require ground truths for its components used in the derivation, i.e. Spanning Set and Coefficients.

3.2 Loss function

3.2.1 Object detection loss

The Heatmap Y and Size D outputs (Fig. 1) are compared to the corresponding ground truths using object detection loss function L_{box} that consists of focal loss L_{foc} for Heatmap and L_{size} for Size. The focal loss L_{foc} is a penalty-reduced pixelwise logistic regression loss (T.-Y. Lin, Goyal, et al., 2017), (Y. Wang et al., 2020) with perturbation of the first polynomial coefficient of Taylor expansion similar to (Leng et al., 2022) defined below as follows:

$$
\begin{aligned}
L_{foc} &= \frac{-1}{N} \cdot \sum_{i=1}^{H/4} \sum_{j=1}^{W/4} \sum_{c=1}^{C} \begin{cases} A_{i,j}^c & if \;\; \breve{Y}_{i,j}^c = 1 \\ B_{i,j}^c & otherwise \end{cases}, \\
A_{i,j}^c &= \left(1 - Y_{i,j}^c\right)^2 \cdot \left(log\left(Y_{i,j}^c\right) + \epsilon_1 \cdot \left(1 - Y_{i,j}^c\right)\right) \\
B_{i,j}^c &= \left(1 - \breve{Y}_{i,j}^c\right)^4 \left(Y_{i,j}^c\right)^2 \left(log\left(1 - Y_{i,j}^c\right) + \epsilon_2 Y_{i,j}^c\right)
\end{aligned} \tag{2}
$$

where $Y = \left\{Y_{i,j}^c\right\}_{i=1,j=1,c=1}^{H/4,W/4,C}$ is the predicted Heatmap and $\breve{Y}_{i,j}^c$ is the ground truth Heatmap values, N is the number of objects in the input image, ϵ_1 is perturbation coefficient for the first polynomial coefficient of Taylor expansion of the case $\breve{Y}_{i,j}^c = 1$, ϵ_2 is perturbation coefficient for the first coefficient for $\breve{Y}_{i,j}^c \neq 1$. If $\epsilon_1 = \epsilon_2 = 0$ then (2) is penalty-reduced pixelwise logistic regression loss (T.-Y. Lin, Goyal, et al., 2017), however optimal choice of these coefficients improves accuracy as it will be shown below. For each n^{th} ground truth box center for some class c^n at row i^n, column j^n, $n = 1, \ldots, N$ we take prediction values from the Size output $[w_l^n, h_t^n, w_r^n, h_b^n]$ = $D_{i^n,j^n}^{c^n}$ and compute bounding box $b^n = [x^n, y^n, w^n, h^n]$ using (1). We define the following combination of l_1-type loss and UnitBox loss (Yu et al., 2016) penalizes bounding box predictions:

$$
\begin{aligned}
L_{size} &= \frac{1}{N} \cdot \sum_{n=1}^{N} \alpha_1 \cdot \left|b^n - \breve{b}^n\right| - \\
&\alpha_2 \cdot log\left(\frac{b^n \cap \breve{b}^n}{b^n \cup \breve{b}^n}\right) = \alpha_1 \cdot L_{l_1} + \alpha_2 \cdot L_{UnitBox},
\end{aligned} \tag{3}
$$

where the ground truth bounding box $\breve{b}^n = [\breve{x}^n, \breve{y}^n, \breve{w}^n, \breve{h}^n]$, intersection $b^n \cap \breve{b}^n$ and union $b^n \cup \breve{b}^n$ as described by Yu et al. (2016)

Then, the object detection loss can be expressed as a linear combination of (2) and (3) averaged over a batch of images:

$$
L_{box} = \frac{1}{b} \sum_{i=1}^{b} (L_{foc}^i + \alpha_1 \cdot L_{l_1}^i + \alpha_2 \cdot L_{UnitBox}^i), \tag{4}
$$

where L_{foc}^i and $L_{size}^i = \alpha_1 \cdot L_{l_1}^i + \alpha_2 \cdot L_{UnitBox}^i$ are loss functions (2) and (3), respectively, for i^{th} image in a batch. The authors Zhou, D. Wang, and Krähenbühl (2019) use the case $\epsilon_1 = \epsilon_2 = \alpha_2 = 0$, $\alpha_1 = 0.1$ which we call below as 'default OD loss'

3.2.2 Mask loss

We estimate instance masks using Coefficients Λ and Spanning Set B as described below. For each ground truth box center i_n, j_n, $n = 1, \ldots, N$, we fix $\lambda_{n,k} = \Lambda_{i_n,j_n}^k$, $k = 1, \ldots, K$, where N is the number of instances and K is the number of channels in both Coefficients and Heatmap outputs. Then, an instance mask is predicted as a linear combination of the Spanning Set channels B^k, $k = 1, \ldots, K$ with coefficients $\lambda_{n,k}$ (also see Fig. 3):

$$
M^n = \sum_{k=1}^{K} \lambda_{n,k} \cdot B^k \tag{5}
$$

We use bi-linear upsampling of M by a factor of 3 and denote the result as $\hat{M} \in \mathbb{R}^{H/2,W/2}$. While the bi-linear resizing is not required, it notably improved accuracy in our experiments. Then, to define loss function for instance masks as combination of Cross-Entropy loss (Jadon, 2020) and Dice loss (Milletari, Navab, and Ahmadi, 2016). Firstly, let define Cross-Entropy loss matrix $L^n = \left\{L_{i,j}^n\right\}_{i=1,j=1}^{H/2,W/2} \in \mathbb{R}^{H/2,W/2}$:

$$
\begin{aligned}
L^n = &-\breve{M}^n \cdot log\left(\sigma\left(\hat{M}^n\right)\right) - \\
&\left(1 - \breve{M}^n\right) log\left(1 - \sigma\left(\hat{M}^n\right)\right),
\end{aligned} \tag{6}
$$

where σ is a sigmoid function; $\breve{M}^n$, $n = 1, \ldots, N$ are ground truth instance masks; N is a number of instances in the ground truth. The ground truth mask equals 0 everywhere, except for instance location, where it equals 1.

We use values of L^n only inside the n^{th} ground truth bounding box $[i_1^n, i_2^n] \times [j_1^n, j_2^n]$ to define loss function for a single mask:

$$
L_{mask_{CE}}^n = \frac{1}{(i_2^n - i_1^n) \cdot (j_2^n - j_1^n)} \cdot \sum_{i=i_1^n}^{i_2^n} \sum_{j=j_1^n}^{j_2^n} L_{i,j}^n \tag{7}
$$

The normalization coefficient $\frac{1}{(i_2^n - i_1^n) \cdot (j_2^n - j_1^n)}$ allows small and large objects contributing equally to the loss function. The second step is defining the Dice loss (Milletari, Navab, and Ahmadi, 2016)

$$
\begin{aligned}
&L_{mask_{Dice}}^n = \\
&1 - \frac{2 \cdot \sum \breve{M}_{i,j}^n \cdot \sigma\left(\hat{M}_{i,j}^n\right) + 1}{\sum \left(\breve{M}_{i,j}^n\right)^2 + \sum \left(\sigma\left(\hat{M}_{i,j}^n\right)\right)^2 + 1},
\end{aligned} \tag{8}
$$

where summarizing indices are as in (7) $i = i_1^n \cdots i_2^n$, $j = j_1^n \cdots j_2^n$ Then, the loss function for a batch with b images, where i^{th} image has N_i ground truth instance masks, is defined as follows:

Table 1: Brute-forcing loss function parameters using backbone ResNet18**, width $\alpha = 0.5$

Table A: Accuracy of object detector for different values of parameters $\epsilon_2, \alpha_1, \alpha_2$

ϵ_2	0.0	0.0	0.0	0.0	0.0	0.0	-0.05	-0.05	-0.1	0.05	0.1	-0.05	0.05	-0.05
α_1	0.1	0.0	0.1	0.2	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.1	0.1	0.1
α_2	0.0	1.0	1.0	1.0	1.0	2.0	1.0	1.5	1.0	1.0	1.0	0.0	0.0	1.5
AP^{box}	26.1	26.3	26.3	26.2	26.4	26.3	26.6	**26.7**	26.5	26.2	26.0	26.2	25.8	26.6

Table B: Accuracy of BlitzMask for different values of parameters β_1, β_2

β_1	1.0	0.0	2.0	0.0	3.0	0.0	4.0	0.0	1.0	2.0	0.5	3.0
β_2	0.0	1.0	0.0	2.0	0.0	3.0	0.0	4.0	2.0	1.0	2.5	1.0
AP^{mask}	21.8	22.2	21.9	22.3	21.9	22.4	21.7	22.4	22.5	22.3	**22.6**	22.3

$$L_{mask} = \frac{1}{b}\sum_{i=1}^{b}\frac{1}{N_i}\cdot\sum_{n_i=1}^{N_i}\left(\beta_1\cdot L^{i,n_i}_{mask_{CE}} + \beta_2\cdot L^{i,n_i}_{mask_{Dice}}\right) \quad (9)$$

Let the case of $\beta_1 = 1$, $\beta_2 = 0$ is called as 'default mask loss'. Difference between 'default mask loss' and segmentation part of YOLACT loss (Bolya et al., 2019a) is in using the normalization coefficient in (7) and bi-linear upsampling of predicted and ground truth masks inside loss function that improve accuracy essentially.

3.2.3 Overall loss

Finally, an overall loss function for a batch of images is defined as a linear combination of box (4) and mask losses (9):

$$L_{all} = L_{box} + L_{mask} = L_{foc} + \alpha_1\cdot L_{l_1} + \alpha_2\cdot L_{UnitBox} + \beta_1\cdot L_{mask_{CE}} + \beta_2\cdot L_{mask_{Dice}} \quad (10)$$

Taking into account (2) we have 6 unknown parameters $\epsilon_1, \epsilon_2, \alpha_1, \alpha_2, \beta_1, \beta_2$. The 'default loss' is the case of $\epsilon_1 = \epsilon_2 = \alpha_2 = 0$, $\alpha_1 = 0.1$, $\beta_1 = 1$, $\beta_2 = 0$. It contains loss described by Zhou, D. Wang, and Krähenbühl (2019) and our modification of the loss described by Bolya et al. (2019a). Below, we obtain optimal parameters using brute-force.

3.3 Post-processing

The instance segmentation masks are expressed using model outputs - Spanning Set B, Coefficients Λ, Heatmap Y and Size D (see Fig.1 and Fig. 3 for post-processing a single class "person"). First, we compute object centers i_n, j_n, $n = 1, 2, \ldots, N$ by using 3×3 maximum pooling on the heatmap Y as described by Law and Deng (2018), Zhou, D. Wang, and Krähenbühl (2019). We only retain N object centers with scores Y_{i_n, j_n} greater than a

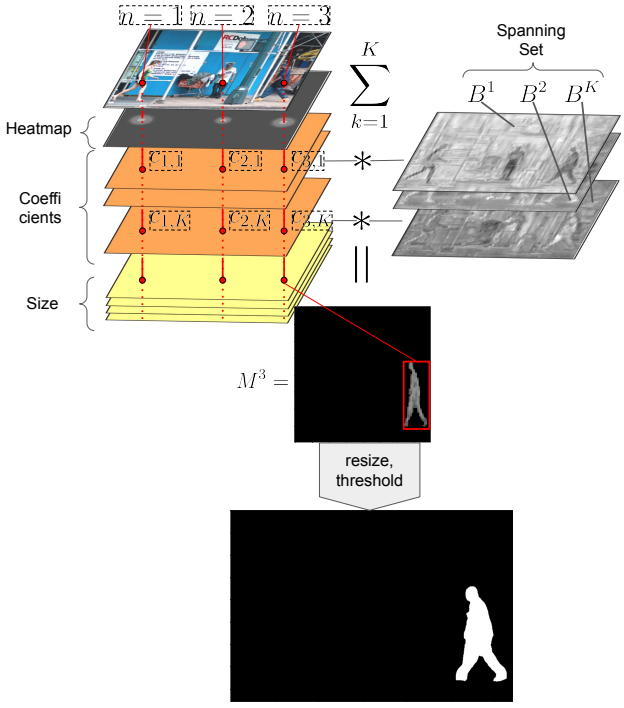

Figure 3: Instance mask reconstruction using Spanning Set B, Coefficients Λ, Heatmap Y and Size D for the case of $N = 3$ instance masks, class number $C = 1$. Operation "resize" means resize M^n, $n = 1, \ldots, N$ to original images size. The "threshold" operation assigns 1 to pixels for which $\sigma(M^n) > 0.5$ and 0, otherwise.

chosen threshold. Then, for each object center i_n, j_n, we compute bounding boxes $b^n = [x^n, y^n, w^n, h^n]$ using (1) and the corresponding instance mask heatmaps M^n, $n = 1, 2, \ldots, N$ using (5); we bi-linearly resize both the derived mask heatmaps M^n and the boxes b^n to the original image size. Lastly, we define the final instance masks as consisting of pixels that are both within the bounding boxes b^n and for which $\sigma(M^n) > 0.5$ (Fig 3).

4 EXPERIMENTS

4.1 Training Details

The instance segmentation models are trained on MS COCO (T.-Y. Lin, Maire, et al., 2014) train2017 dataset and evaluated on val2017. We follow the recommendations of Izmailov et al. (2018) and Smith (2018) to search for a flat (rather than sharp) minima on the loss surface, as a sharp minima determined on train data could result in a large loss function value on evaluation data. Our training process is a modification of cyclical learning rates approach (Smith, 2018), it uses Stochastic Gradient Descent (SGD) with momentum and has three phases: Ascent, Plateau, and Descent. We will refer to the training process as 'LRfinder' below.

At the Accent phase, LRfinder uses momentum $\mu = 0.9$ and linearly increases the learning rate as much as possible until the training loss starts deteriorating. Then we reduce the rate of the learning increasing twice and load previous epoch weights and repeat this procedure three times. The maximum achieved learning rate is then denoted as η. At the Plateau phase, the learning rate is fixed to η and we count "Patience", e.g. the number of epochs without training loss improvement. Each time the Patience increases by 5, the momentum value gets reduced to the following values $\mu = 0.63$, $\mu = 0.44$, $\mu = 0.3$, and then the Descent phase starts. At the Descent phase, the SGD momentum is $\mu = 0.3$, learning rate is reduced to $0.1 \cdot \eta$, $0.01 \cdot \eta$, $0.001 \cdot \eta$ each time the Patience increases by 5. Training is finished when learning rate becomes $0.001 \cdot \eta$. Notably that sometimes gradient explosion occurs in Plateau phase because of high learning rate value. In such case, we load previous epoch weights and switch to the next step (momentum reducing or Descent phase) and it always fixes the gradient explosion problem.

The method described by Smith (2018) gave us an idea how to modify the training process, but our implementation is conceptually different, specifically, we: 1) do not have any Cyclical Learning rates like described by Smith (2018), 2) have different rules for SGD momentum changing, and 3) follow a different approach to decrease the learning rate. In contrast to (Smith, 2018), we find the optimal learning rate and SGD momentum only during the training by creating model backups and "look forwards" to what happens if we change the learning rate or SGD momentum. Moreover, we use different learning rates ascent speeds and SGD momentum descent to get the best validation loss. It essentially differs from the method proposed by Smith (2018) and any other training process published in the literature.

We use random flip, scaling (between 0.25 to 4.0), cropping, photometric transformations and aspect ratio modification (from 0.5 to 2.0) as train data augmentation. All models are trained on a single Nvidia V100 GPU from scratch, without any pretrained weights, with input resolution 320×320 and batch size 64. Models are converted to TFlite and tested on S21 with Snapdragon 888 system on a chip with CPU Kryo 680, GPU Adreno 660. We use FLOAT16 quantization for S21 GPU and UINT8 for S21 DSP. To save training time, first, we train without mask upsampling in (9) and then tune with mask upsampling by a factor of 3.

4.2 Loss function parameters

To brute force parameters $\epsilon_1, \epsilon_2, \alpha_1, \alpha_2, \beta_1, \beta_2$ we use 'fast training' with modified backbone ResNet18, without segmentation mask upsampling by a factor of 3 in (9), a short "Patience" = 2 and switching of Accent phase to Plateau phase just after training loss starts deteriorating (i.e. without using different learning rates ascent speeds). The modified backbone is denoted as ResNet18** and does not contain first stride 2 and 3×3 MaxPool and has width $\alpha = 0.5$, i.e. only half of the convolution layer filter number. We found that BlitzMask with such backbone required only 120-140 epochs to train, and each epoch is near 11 minutes on a single Nvidia V100 GPU in contrast to near 350 epochs of full training with MobileNetV2 or EfficientNet-Lite backbone with near 50 minutes per epoch. Firstly, we get parameters $\epsilon_1, \epsilon_2, \alpha_1, \alpha_2$ used in loss functions L_{foc} (2) and L_{size} (3) for object detection (i.e. without Coefficients and Spanning Set outputs in Fig. 1) and then add additional head for Instance Segmentation to obtain parameters β_1, β_2 used in L_{mask} (9).

Our experiments show that parameter ϵ_1 has negligible influence on accuracy, therefore we fix $\epsilon_1 = 0$ and show some of our results for $\epsilon_2, \alpha_1, \alpha_2$ in Table A of Table 1 for object detection case only (i.e. only Heatmap and Size outputs in Fig. 1). The second column in Table A of Table 1 is object detection 'default case' which use the same loss as described by Zhou, D. Wang, and Krähenbühl (2019) and the best accuracy is reached with $\epsilon_2 = -0.05$, $\alpha_1 = 0.05$, $\alpha_2 = 1.5$. Chosen parameters give $+0.6AP^{box}$ in comparison to 'default case'. Then the best parameters for object detection case are fixed and β_1, β_2 are varying in Table B of Table 1. Then we add segmentation head (Spanning Set and Coefficients outputs in Fig. 1), use obtained parameters of the object detection loss and vary β_1, β_2 inside mask loss in Table B of Table 1

Table 2: BlitzMask speed and accuracy of the instance segmentation on Samsung S21 Snapdragon 888 **GPU** (FLOAT16 quantization) and **DSP** (UINT8 quantization) for input resolution 320×320, backbones MobileNetV1 (A. G. Howard et al., 2017), MobileNetV2 (Sandler et al., 2018a), EfficientNet-Lite B0-B4 (M. Tan and Le, 2019), (*EfficientNet-Lite* 2022) and modified ResNet-18 (He et al., 2015)

	GPU			DSP		
Backbone	Time(ms)	AP^{mask}	AP^{mask}_{50}	Time(ms)	AP^{mask}	AP^{mask}_{50}
MobileNetV1	21.3	21.0	34.3	10.4	20.2	33.1
MobileNetV2	21.8	21.9	35.8	9.2	21.0	34.7
EfficientNet-Lite B0	22.3	25.2	43.4	14.1	24.3	41.8
EfficientNet-Lite B1	24.3	26.2	45.0	16.0	25.4	43.5
EfficientNet-Lite B2	25.2	27.0	45.9	15.7	26.2	44.6
EfficientNet-Lite B3	28.2	27.9	47.3	18.1	27.0	45.9
EfficientNet-Lite B4	34.2	28.9	48.7	25.4	28.0	47.1
ResNet-18*	88.2	29.6	50.1	52.6	29.3	49.4

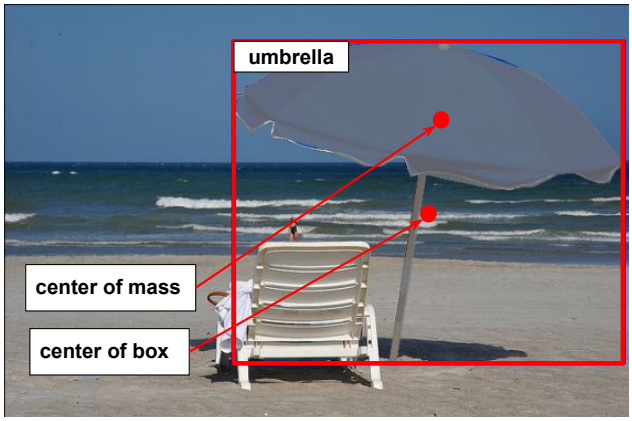

Figure 4: Difference between center of mass and center of box position

As shown in Table A of Table 1 mixing of Cross Entropy and Dice loss with $\beta_1 = 0.5$, $\beta_2 = 2.5$ gives $+0.8 AP^{mask}$ in comparison with 'default case' which is segmentation part of YOLACT loss (Bolya et al., 2019a).

Table 3: Instance segmentation accuracy for cases 1) Piece-wise constant learning rate schedule, 2) LRfinder, 3) 3x mask upsampling L_{mask}, 4) perturbation of the first polynomial coefficient of the Taylor's expansion $\epsilon_2 = -0.05$, 5) l_1 loss and UnitBox loss (Yu et al., 2016) with $\alpha_1 = 0.05$, $\beta_1 = 1.5$, 6) Cross Entropy loss and Dice loss (Milletari, Navab, and Ahmadi, 2016) with $\beta_1 = 0.5$, $\beta_2 = 2.5$, and 7) using mass center instead of box center.

1)	2)	3)	4)	5)	6)	7)	AP^{mask}
✓							18.7
	✓						19.8
	✓	✓					20.7
	✓	✓	✓				20.9
	✓	✓	✓	✓			21.2
	✓	✓	✓	✓	✓		21.6
	✓	✓	✓	✓	✓	✓	21.9

4.3 Impact of our improvements

Sometimes, the box center lies outsides the object, as shown in Fig. 4). We try to use mass center of instance mask as object centers for Heatmap and Coefficients instead of box center, and it improves accuracy a bit (see Table 3)

In Table 3 we show influence of our modification of "default" learning rate schedule, "default" loss functions and 'default' object centers position on Heatmap for the case of MobileNetV2 backbone. The best results for piece-wise constant learning rate schedule (which we called "default") we obtain using SGD with momentum μ= 0.9, learning rate $\eta = 0.02$ and standard "ReduceOnPlateau" technique, i.e. by $10x$ reducing learning rate each time "Patience" = 5 epochs without accuracy improvement. The cases 1) and 2) in Table 3 shows that LRfinder attains 1.1 AP gains in comparison to piece-wise constant learning rate. Then the mask upsampling in (9) (case 3) in Table 3) gives +0.9 AP, perturbation of the first polynomial coefficient of the Taylor's expansion (case 4)) adds 0.2 AP, mixing l_1 and UnitBox (Yu et al., 2016) losses with obtained coefficients in Table A of Table 1 instead of "default OD loss" adds 0.3 AP. Modification of "default" mask loss with $\beta_1 = 1$, $\beta_2 = 0$ by using obtained in Table B of Table 1 parameters achieved +0.4 AP improvement and using another Gaussian peak location in Ground Truth Heatmap adds 0.3 AP additionally. The whole improvement package achieves significant 3.2 AP instant segmentation accuracy improvement.

4.4 Results

As discussed in the Related work section, there are no published results for real-time instance segmentation on smartphones with adequate accuracy, and hence Table 2 sets a new benchmark for speed and accuracy on smartphones. As elaborated earlier, YolactEdge (H. Liu et al., 2020) when used with the lowest computing capability available in the literature - Jetson AGX Xavier - achieves $AP^{mask} = 20.8$ on MS COCO val2017 dataset with 35.7 FPS. However, AGX Xavier cannot be used in smartphones, and is superior to Samsung S21 GPUs (Ignatov et al., n.d.). Further, given that YolactEdge input resolution is 550×550, we achieve better accuracy of 21.9 with a smaller input resolution 320×320, but the same backbone MobileNetV2 (see Table 2). Our results could be compared with mobileYOLACT (J. Lee, S. Lee, and Ko, 2021) which achieves $AP^{mask}_{50} = 23.0$ at the speed of 21 fps on Samsung Galaxy S20. In contrast, our accuracy for the case of EfficientNet-Lite B4 backbone is $AP^{mask}_{50} = 47.1$ at the speed of 39.4 fps(see Table 2) on Samsung Galaxy S21. To allow comparison with mobile YOLACT, we additionally ran our method on Samsung S20 and achieved 36.8 fps (the same accuracy as on S21). This shows that our model has 2x better accuracy and 1.75x faster inference speed than mobileYOLACT (J. Lee, S. Lee, and Ko, 2021). Moreover, another approach EOLO (Zeng and Sabah, 2020) on Raspberry Pi4 with Google Coral USB, comparable with smartphone computing capability, achieves only $AP^{mask} = 11.7$.

Furthermore, we compare our box detection accuracy with state-of-the-art object detection on mobile YOLObite (Cai et al., 2020). In the case of EfficientNet-Lite B4 backbone, our approach achieves $AP^{mask} = 28.9$ and $AP^{box} = 31.4$ at 29.2 fps on Samsung S21 Adreno 660 GPU whereas YOLObile reaches $AP^{box} = 31.6$ at 19.1 fps on Samsung S20 Adreno 650 GPU. Mobile ranking (*Mobile Ranking* 2022) shows that S21 GPU speed is similar to S20 GPU speed, so that our approach reaches state-of-the-art in Object Detection on mobile (note, we did not target this capability). To utilize BlitzMask just for box detection, one can cut segmentation head (i.e. layers related to Spanning Set and Coefficient outputs in Fig. 1) from the model and get $AP^{box} = 31.4$ at 30.4 fps. Table 2) shows BlitzMask speed and accuracy on MobileNetV1 (A. G. Howard et al., 2017), MobileNetV2 (Sandler et al., 2018a) and EfficientNet-Lite B0-B4 (M. Tan and Le, 2019), (*EfficientNet-Lite* 2022). We found EfficientNet-Lite backbones are very efficient, and they reach 24.9 AP on 44.8 fps (B0 case) and 28.9 AP on 29.2 fps (B4 case) that is significantly better than MobileNet backbones. Lastly, Table 2) shows results for our heavier and more precise model with ResNet-18 backbone operating at 11.2 FPS on Samsung S21. While not sufficient for real-time instance segmentation, it still could be used for "near real-time" tasks. ResNet18* in Table 2) denotes ResNet-18 without 3×3, stride 2 MaxPool. This backbone is significantly heavier than the original ResNet-18 (He et al., 2015), but in our experiments, it has a better accuracy/speed trade-off.

5 CONCLUSIONS

We propose a new fast anchor-free instance segmentation approach BlitzMask whose mask head requires significantly less resources than the entire model. We use a modified YOLACT (Bolya et al., 2019a) loss function and a special training process to increase model accuracy. One of the main BlitzMask advantages is its simplicity, which is very important for a further method development or/and re-implementation from scratch. Our experiments show that mixing of l_1 and UnitBox losses for the box regression part and Cross-Entropy and Dice losses for the segmentation part improves model performance. In addition, using center mass of masks as location of Gaussian peaks in Heatmap improves model accuracy. Simple and fast post-processing allows achieving real-time speed on mobile devices with sufficient accuracy. BlitzMask achieves $AP^{mask} = 28.9$ at 29.2 fps on Samsung S21 GPU. This sets a new speed and accuracy benchmark for instance segmentation inference on mobile devices. In addition, BlitzMask reaches state-of-the-art object detection accuracy for real-time mobile applications.

Acknowledgments

Special thanks to Dr. Viktor Porokhonskyy (Samsung R&D Institute, Ukraine) for his expertise, assistance, helpful comments and wise advice in writing the manuscript.

References

Türkmen, Sercan and Janne Heikkilä (2019). "An efficient solution for semantic segmentation: Shufflenet v2 with atrous separable convolutions". In: *Scandinavian Conference on Image Analysis*. Springer, pp. 41–53 (cit. on p. 1).

Howard, Andrew et al. (2019). "Searching for mobilenetv3". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324 (cit. on p. 1).

Orsic, Marin et al. (2019). "In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12607–12616 (cit. on p. 1).

Cai, Yuxuan et al. (2020). "YOLObile: Real-time object detection on mobile devices via compression-compilation co-design". In: *arXiv preprint arXiv:2009.05697* (cit. on pp. 1, 9).

Vasu, Pavan Kumar Anasosalu et al. (2022). "An Improved One millisecond Mobile Backbone". In: *arXiv preprint arXiv:2206.04040* (cit. on p. 1).

Bazarevsky, V et al. (2020). "BlazePose: On-device real-time body pose tracking. arXiv". In: *arXiv preprint arXiv:2006.10204* (cit. on p. 1).

Minaee, Shervin et al. (2021). "Image segmentation using deep learning: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* (cit. on p. 1).

Liu, Haotian et al. (2020). "YolactEdge: Real-time Instance Segmentation on the Edge". In: *arXiv preprint arXiv:2012.12259* (cit. on pp. 1–3, 9).

Zeng, Longfei and Mohammed Sabah (2020). "EOLO: Embedded Object Segmentation only Look Once". In: *arXiv preprint arXiv:2004.00123* (cit. on pp. 1, 3, 9).

Dai, Jifeng et al. (2017). "Deformable convolutional networks". In: *Proceedings of the IEEE international conference on computer vision*, pp. 764–773 (cit. on pp. 1–3).

Wang, Xinlong, Rufeng Zhang, et al. (2020). "SOLOv2: Dynamic and fast instance segmentation". In: *arXiv preprint arXiv:2003.10152* (cit. on pp. 1–3).

Wang, Xinlong, Tao Kong, et al. (2020). "SOLO: Segmenting objects by locations". In: *European Conference on Computer Vision*. Springer, pp. 649–665 (cit. on pp. 1–3).

Wang, Yuqing et al. (2020). "CenterMask: single shot instance segmentation with point representation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9313–9321 (cit. on pp. 1–4).

Sandler, Mark et al. (2018a). "MobileNetV2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520 (cit. on pp. 1, 8, 9).

Bolya, Daniel et al. (2019a). "YOLACT: Real-time instance segmentation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9157–9166 (cit. on pp. 2, 3, 6, 8, 9).

– (2019b). "YOLACT++: Better real-time instance segmentation". In: *arXiv preprint arXiv:1912.06218* (cit. on p. 2).

Lee, Juwon, Seungjae Lee, and Jong Gook Ko (2021). "Mobile YOLACT: Toward Lightweight Instance Segmentation for Mobile Devices". In: *2021 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, pp. 1456–1460 (cit. on pp. 2, 3, 9).

Redmon, Joseph and Ali Farhadi (2018). "YOLOv3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (cit. on p. 2).

Zhou, Xingyi, Dequan Wang, and Philipp Krähenbühl (2019). "Objects as points". In: *arXiv preprint arXiv:1904.07850* (cit. on pp. 2–7).

Chen, Hao et al. (2020). "BlendMask: Top-down meets bottom-up for instance segmentation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8573–8581 (cit. on p. 2).

Xie, Enze et al. (2020). "PolarMask: Single Shot Instance Segmentation with Polar Representation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12193–12202 (cit. on pp. 2, 3).

Lin, Tsung-Yi, Michael Maire, et al. (2014). *Microsoft COCO: Common Objects in Context* (cit. on pp. 2, 7).

Tian, Zhi et al. (2019). "FCOS: Fully Convolutional One-Stage Object Detection". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9627–9636 (cit. on p. 3).

Sandler, Mark et al. (2018b). "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 3).

Ignatov, Andrey et al. (n.d.). "Ai benchmark: All about deep learning on smartphones in 2019. In 2019 IEEE". In: *CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3617–3635 (cit. on pp. 3, 9).

Jetson benchmarks (2022). Accessed: 2022-02-16. URL: https://developer.nvidia.com/embedded/jetson-benchmarks (cit. on p. 3).

Mobile Ranking (2022). Accessed: 2022-10-11. URL: https://ai-benchmark.com/ranking_detailed.html (cit. on pp. 3, 9).

Zhu, Lingyu et al. (2019). "Portrait instance segmentation for mobile devices". In: *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, pp. 1630–1635 (cit. on p. 3).

Lin, Tsung-Yi, Piotr Dollár, et al. (2017). "Feature Pyramid Networks for Object Detection". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944. DOI: 10.1109/CVPR.2017.106 (cit. on p. 3).

Howard, Andrew G et al. (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (cit. on pp. 3, 4, 8, 9).

Chiang, Cheng-Ming et al. (2020). "Deploying image deblurring across mobile devices: A perspective of quality and latency". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 502–503 (cit. on p. 3).

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR, pp. 448–456 (cit. on p. 3).

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on p. 3).

Law, Hei and Jia Deng (2018). "Cornernet: Detecting objects as paired keypoints". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 734–750 (cit. on pp. 4, 6).

Liu, Zili et al. (2020). “Training-time-friendly network for real-time object detection”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07, pp. 11685–11692 (cit. on p. 4).

Lin, Tsung-Yi, Priya Goyal, et al. (2017). “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988 (cit. on pp. 4, 5).

Leng, Zhaoqi et al. (2022). “PolyLoss: A Polynomial Expansion Perspective of Classification Loss Functions”. In: *arXiv preprint arXiv:2204.12511* (cit. on p. 4).

Yu, Jiahui et al. (2016). “Unitbox: An advanced object detection network”. In: *Proceedings of the 24th ACM international conference on Multimedia*, pp. 516–520 (cit. on pp. 5, 8).

Jadon, Shruti (2020). “A survey of loss functions for semantic segmentation”. In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. IEEE, pp. 1–7 (cit. on p. 5).

Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi (2016). “V-net: Fully convolutional neural networks for volumetric medical image segmentation”. In: *2016 fourth international conference on 3D vision (3DV)*. IEEE, pp. 565–571 (cit. on pp. 5, 8).

Tan, Mingxing and Quoc Le (2019). “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International conference on machine learning*. PMLR, pp. 6105–6114 (cit. on pp. 8, 9).

EfficientNet-Lite (2022). Accessed: 2022-10-03. URL: https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite (cit. on pp. 8, 9).

He, Kaiming et al. (2015). “Deep Residual Learning for Image Recognition”. In: *arXiv preprint arXiv:1512.03385*. arXiv: 1512.03385 (cit. on pp. 8, 9).

Izmailov, Pavel et al. (2018). “Averaging weights leads to wider optima and better generalization”. In: *arXiv preprint arXiv:1803.05407* (cit. on p. 7).

Smith, Leslie N (2018). “A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay”. In: *arXiv preprint arXiv:1803.09820* (cit. on p. 7).

A TRAINING DETAILS

Here, we describe the training details of how we achieved $AP^{mask} = 24.9$ with backbone EfficientNet-Lite B0. We use $\epsilon_1 = 0$, $\epsilon_2 = -0.05$, $\alpha_1 = 0.05$, $\beta_1 = 1.5$ inside loss function L_{box} and $\beta_1 = 0.5$, $\beta_2 = 2.5$ inside loss function L_{mask}, train without predicted mask upsampling using LRfinder and then tune with mask upsampling. These values are obtained by brute-forcing loss function parameters in 'fast training' described in Section 4.2. Other backbones show a similar training process, but for larger EfficientNet-Lite number ($B0 > B1 > B2 > \cdots$) training process is a bit longer. In the Accent phase (see Section 4) we use momentum $\mu = 0.9$ and the learning rate is increased each epoch on value 0.0025 linearly. After 49 epochs training loss starts deteriorating. Therefore, previous epoch weights are loaded, and increasing step is set to 0.00125. Analogically, the learning rate increasing step is reduced after 57 and 65 epoch and at 78 epoch the Plateau phase us started with learning rate $\eta = 0.1403125$, $AP^{mask} = 17.8$. The Plateau phase is continued till 261 epoch, e.g. it takes 183 epochs. The SGD momentum is reduced to $\mu = 0.63$ at 120 epoch, to $\mu = 0.44$ at 202 epoch, to $\mu = 0.3$ at 223 epoch. The criteria of momentum reducing or switching to Descent phase (if $\mu = 0.3$) is absence of training loss improvement till Patience = 5 epochs. The Descent phase begins with 10 time reduced learning rate $\eta = 0.01403125$, $\mu = 0.3$ and due to above-mentioned criteria learning rate becomes equal to $0.01 \cdot \eta$ at 282 epoch and to $0.001 \cdot \eta$ at 301 epoch. The training process is finished after 312 epoch and accuracy $AP^{mask} = 23.7$. Then we use upsampling by a factor of 3 for predicted and Ground Truth masks as it is described in Section 3.2.2 and tune trained model with momentum $\mu = 0.9$, learning rate $eta = 0.0001$ for 30 epoch and finally achieve $AP^{mask} = 24.9$. We just mention that EfficientNet-Lite B1 requires 332 epochs, B2 - 334 epochs, B3 - 358 epochs, B4 - 391 epochs. In each case, the model is tuned for a 30 epoch with learning rate $eta = 0.0001$ using predicted mask $3x$ upsampling in loss function.

B TRAINING FROM SCRATCH

Each type of mobile devices has its own computational specifics, e.g. some operations have either no or poor support, requiring model architecture modifications. Traditionally, each such model modification is pre-trained on ImageNet or another large dataset, making the model development process much more complex. To account for this, we train from scratch similarly to ScratchDet.

Using pre-trained weight such as ImageNet is not a big problem for research purposes only. But it could be a problem in the industry where a license for a large dataset may be too expensive and hence training from scratch using a smaller private dataset is in demand. Moreover, if one has a many configurations and backbones and needs to train many models "on the conveyor" then pretraining step makes the whole process more complex

C VISUALIZATION OF INSTANCE SEGMENTATION RESULTS

In Fig.5 we show inference of BlitzMask, input resolution 320×320 on 20 random images of MS COCO val2017 for MobileNetV2 backbone and EfficientNet-Lite B4. One can see that B4 backbone gives much better accuracy than MobileNetV2 backbone.

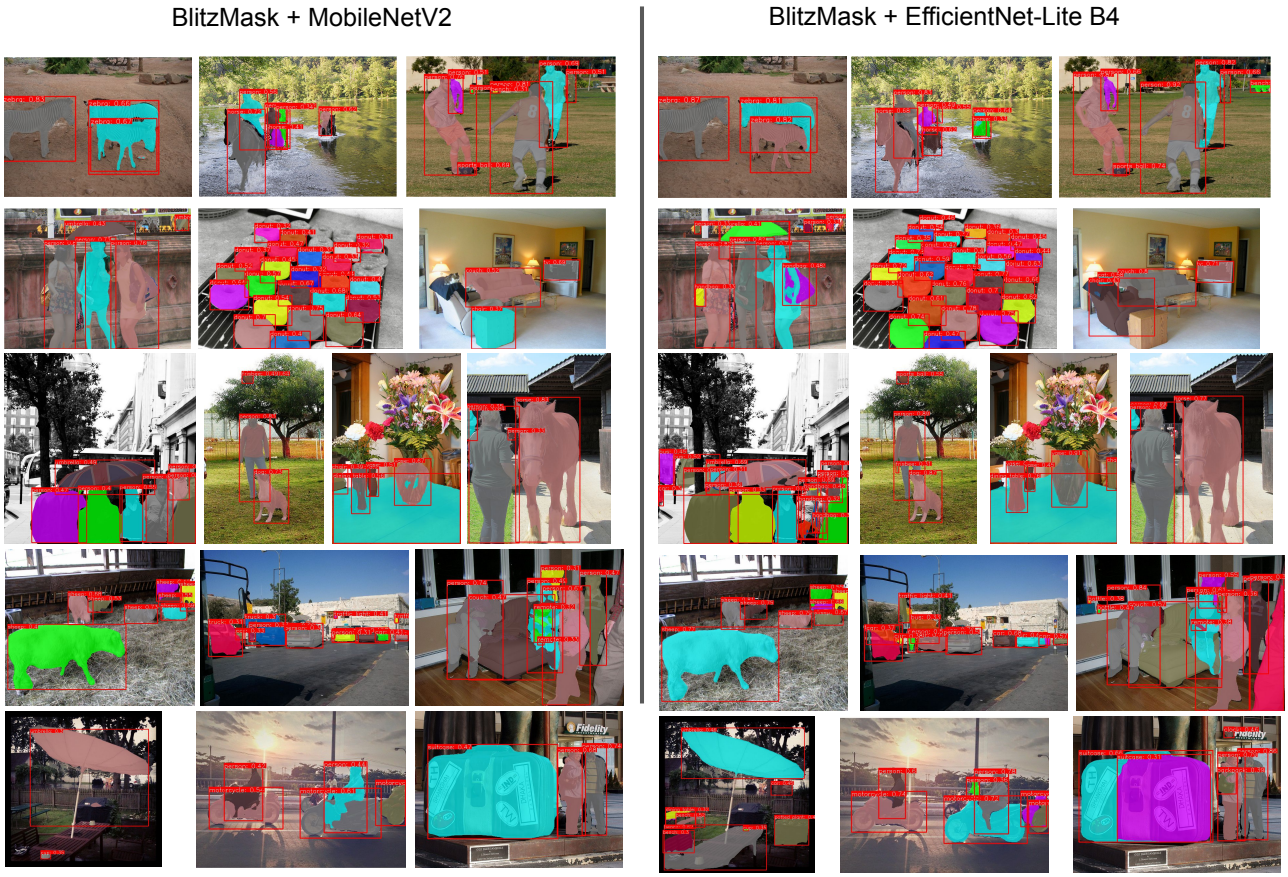

Figure 5: BlitzMask output with backbones MobileNetV2 (left) and EfficientNet-Lite B4(right) for input resolution 320×320, output resolution 80×80. Red rectangles denote predicted bounding boxes, the corresponding object type and confidence score are shown at the top of each box, different instance masks are shown by different colors on each image. All images have the confidence threshold at 0.3.