Towards Real-Time Automatic Portrait Matting on Mobile Devices

Seokjun Seo* [1], Seungwoo Choi* [1], Martin Kersner* [1], Beomjun Shin* [1], Hyungsuk Yoon† [2], Hyeongmin Byun [1], and Sungjoo Ha [1]

[1]Hyperconnect, Seoul, South Korea
[2]Unaffiliated, Seoul, South Korea
{seokjun.seo, seungwoo.choi, martin.kersner}@hpcnt.com
youhanmir@gmail.com
{hyeongmin.byun, shurain}hpcnt.com

Abstract

We tackle the problem of automatic portrait matting on mobile devices. The proposed model is aimed at attaining real-time inference on mobile devices with minimal degradation of model performance. Our model MMNet, based on multi-branch dilated convolution with linear bottleneck blocks, outperforms the state-of-the-art model and is orders of magnitude faster. The model can be accelerated four times to attain 30 FPS on Xiaomi Mi 5 device with moderate increase in the gradient error. Under the same conditions, our model has an order of magnitude less number of parameters and is faster than Mobile DeepLabv3 while maintaining comparable performance. The accompanied implementation can be found at https://github.com/hyperconnect/MMNet.

1. Introduction

Image matting, the task which predicts alpha values of foreground on every pixel, has been studied [8, 12–15]. Image matting system offers an opportunity for wide applications in computer vision such as color transformation, stylization, and background edits. It is well-known, however, that image matting is an ill-posed problem [19] since seven unknown values (three for foreground RGB, three for background RGB and one for alpha) should be inferred from three known RGB values. The most widely used method to alleviate the difficulties of the matting problem is to utilize an additional input which roughly separates an image such as trimap [12, 30] and scribbles [19]. A trimap splits an image into three parts: definite foreground, definite background, and ambiguous blended regions. Scribbles, on the other hand, indicate foreground and background with a few strokes. Even though some of the traditional methods [19, 27, 30, 34] work well if additional inputs are provided, it is hard to extend these methods to various image and video matting applications which require real-time performance due to their high computational complexity as well as the dependency on user-interactive inputs.

Other approaches have been studied to automate matting

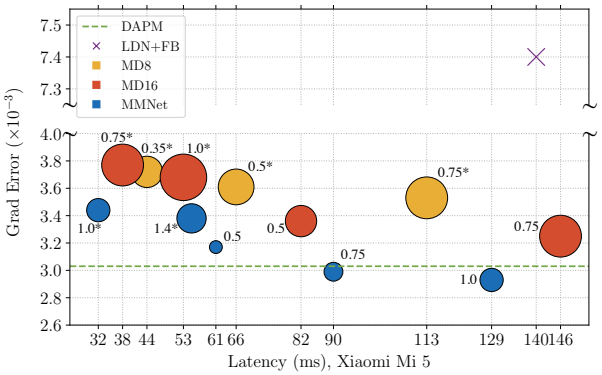


Figure 1. The trade-off between gradient error and latency on a mobile device. Latency is measured using a Qualcomm Snapdragon 820 MSM8996 CPU. Size of each circle is proportional to the logarithm of the number of parameters used by the model. Different circles of Mobile DeepLabv3 are created by varying the output stride and width multiplier. The circles are marked with their width multiplier. Results using 128×128 inputs are marked with $*$, otherwise, inputs are in 256×256. Notice that MMNet outperforms all other models forming a Pareto front. The number of parameters for LDN+FB is not reported in their paper. Best viewed in color.

*Equal contributions.
†Work done while at Hyperconnect.

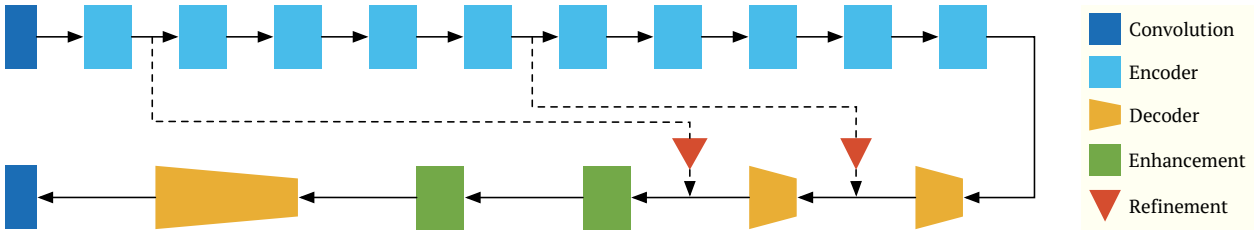


Figure 2. The overall structure of the proposed model. A standard encoder-decoder architecture is adopted. Successively applying encoder blocks summarize spatial information and capture higher semantic information. Decoding phase upsamples the image with decoder blocks and improves the result with enhancement blocks. Information from skip connections is concatenated with the upsampled information. Images are resized to target size before going through the network. The resulting alpha matte is converted back to its original resolution.

by specifying the object which has to be selected as a foreground [9, 29, 36], for example, portrait matting. Automatic portrait matting showed even better result than the other methods using trimap [29], but the latency is far too high to be used in a real-time application. Zhu et al. [39] released a lightweight model which can perform automatic matting relatively fast on mobile devices, attaining the latency of 62 ms per image on Xiaomi Mi 5. However, the gradient error of lightweight model was more than two times worse than that of the state-of-the-art, which made it less attractive in real-world applications.

In this paper, we propose a compact neural network model for automatic portrait matting which is fast enough to run on mobile devices. The proposed model adopts an encoder-decoder network structure [4] and focuses on devising efficient components of the network. We apply depthwise convolution [31] as the basic convolution operation to extract and downsample features. The depthwise convolution is considerably cheaper than other convolutions even if we take efficient convolutions such as 1×1 convolution [16] into account as well. The linear bottleneck structure [26] benefits from the efficiency of depthwise convolutions, boosting the performance while maintaining the latency. Building upon these observations, the encoder block of the proposed model, consists of multi-branch dilated convolution with linear bottleneck blocks which can reduce the model size with the linear bottleneck structure while aggregating multi-scale information with multi-branch dilated convolutions. We introduce the width multiplier, a global variable which enlarges or shrinks the number of channels of a convolution, to control the trade-off between the size and the latency of the model. We incorporate multiple losses into our loss function, including a gradient loss which we propose.

The proposed model shows better performance than the state-of-the-art method while achieving 30 FPS on iPhone 8 without GPU acceleration. We also evaluate the trade-offs between performance, the latency on mobile devices, and the size of the model. Our model can achieve 30 FPS on Google Pixel 1 and Xiaomi Mi 5 using a single core while suffering roughly 10% degradation of gradient error compared to the state-of-the-art.

Our contributions are as follows:

- We propose a compact network architecture for automatic portrait matting task which achieves a real-time latency on mobile devices.
- We explore multiple combinations of input resolution and width multiplier, which can beat strong baselines for automatic portrait matting on mobile devices.
- We demonstrate the capability of each component of the model, including the multi-branch dilated convolution with linear bottleneck blocks, the skip connection refinement block and the enhancement block, through ablation studies.

2. Methods

Image matting problems take input image I, which is a mixture of the foreground image F and the background image B. Each pixel at the i-th position is computed as follows:

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i, \tag{1}$$

where the foreground opacity determines α_i. Since all the quantities on the right-hand side of the Equation 1 are unknown, the problem is ill-posed.

However, we add an assumption that F_i and B_i are identical to I_i in order to reduce the complexity of the problem. Even though the assumption may decrease the performance substantially, the empirical result of our experiments show this assumption is reasonable considering the latency gain we get.

2.1. Model Architecture

Our model follows a standard encoder-decoder architecture that is widely used in semantic segmentation tasks [4, 20, 24]. Encoder successively reduces the size of the input by downsampling and summarizes the spatial information while capturing higher semantic information. Decoder, in

turn, upsamples the image to recover the detailed spatial information and restores the original input resolution. The whole network structure of our model, mobile matting network (MMNet), is depicted in Figure 2.

Many modern neural network architectures replace a regular convolution with a combination of several cheaper convolutions [11, 32, 35]. Depthwise separable convolution [11, 16] is one of the examples which consists of a depthwise convolution, applying a single convolutional filter per input channel, and a pointwise convolution (1×1 convolution) that accumulates the results. We not only use depthwise separable convolution for some blocks but also adopt the concept of depthwise separable convolution when designing our encoder block. Depthwise convolution is one such example which we use extensively. All convolution operations are followed by a batch normalization and a ReLU6 non-linearity except the linear projection operation that is placed at the end of the encoder block [26]. Due to the linear bottleneck structure, the information flow from an encoder block to another is projected to a low-dimensional representation.

In the encoder block, the information flowing from the lower layers is expanded by the first 1×1 multi-branch convolutions. The linear bottleneck compresses the processed image. The data upsampled by the decoder block is concatenated with the refined knowledge through a skip connection. The number of channels for each path are maintained to have the same value. Table 1 details how much each component expands and compresses the information flow.

To control the trade-off between model size and model performance, we adopt width multiplier [16]. The width multiplier α, is a global hyperparameter that is multiplied to the number of input and output channels to make the layers thinner or thicker depending on the computational budget.

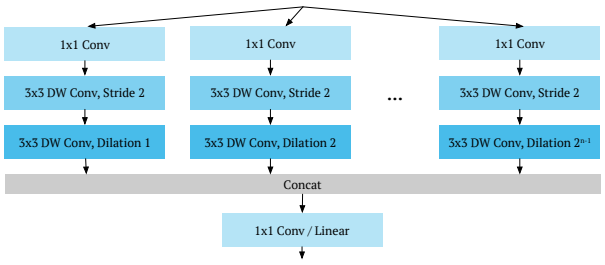


Figure 3. The encoder block. It employs a multi-branched dilated convolution with a linear bottleneck. The linear bottleneck compresses the information to a low-dimensional representation before handing it over to the next encoder block.

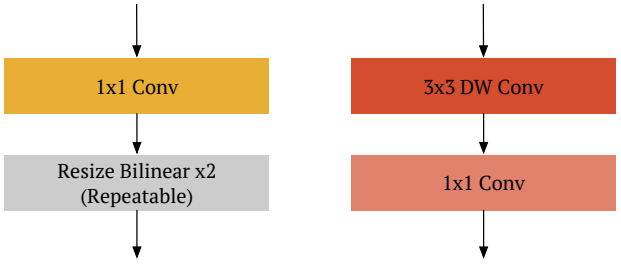
Name	Component Details	Output Size
Initial Block	Conv 3×3, S2	128×128, 32
Encoder 1	DR $[1, 2, 4, 8]$, S2	64×64, 16
Encoder 2	DR $[1, 2, 4, 8]$, S1	64×64, 24
Encoder 3	DR $[1, 2, 4, 8]$, S1	64×64, 24
Encoder 4	DR $[1, 2, 4, 8]$, S1	64×64, 24
Encoder 5	DR $[1, 2, 4]$, S2	32×32, 40
Encoder 6	DR $[1, 2, 4]$, S1	32×32, 40
Encoder 7	DR $[1, 2, 4]$, S1	32×32, 40
Encoder 8	DR $[1, 2, 4]$, S1	32×32, 40
Encoder 9	DR $[1, 2]$, S2	16×16, 80
Encoder 10	DR $[1, 2]$, S1	16×16, 80
Decoder 1	Upsample $\times 2$ (Skip 5)	32×32, 128
Decoder 2	Upsample $\times 2$ (Skip 1)	64×64, 80
Enhancement 1	DR $[1, 2, 4]$, S1	64×64, 40
Enhancement 2	DR $[1, 2, 4]$, S1	64×64, 40
Decoder 3	Upsample $\times 4$	256×256, 16
Final Block	Conv 1×1, Softmax	256×256, 2

Table 1. The model architecture of MMNet. We assume that width multiplier are set to 1.0. Decoder #1 and #2 are connected to encoder #5 and #1 with a skip connection and a refinement block, respectively. DR denotes the dilation rates in the multi-branch dilated convolutions. S represents the stride value in the strided convolution.

2.1.1 Encoder Block

MMNet encoder block has a multi-branched dilated convolution structure with a linear bottleneck. Input flows to multiple branches which undergo channel expansion followed by a strided convolution and a dilated convolution. The dilation rates are different for all branches following 2^{n-1} rates. Multi-branch dilated convolution amounts to sampling spatial information at different scales. The outputs of different branches are concatenated to form a tensor containing multi-scale information. Applying encoder blocks in succession allows the network to capture multi-level information increasingly. As the encoder blocks are consecutively applied, we decrease the number of branches in an encoder block, slowly changing the dilation rates from $[1, 2, 4, 8]$ to $[1, 2]$.

A linear bottleneck structure is imposed on the encoder block where the output of the encoder block is thinner than the intermediate representations. The final convolution after combining the multi-branch information projects the input to a low-dimensional compressed representation. The linear bottleneck is a decomposition of a regular convolution that connects two encoder blocks into two cheaper convolutions with reduced channels. The encoder block is illustrated in Figure 3.



(a) Decoder Block (b) Refinement Block

Figure 4. The decoder block (a) upsamples bilinearly which could be repeated multiple times to upsample by a larger factor. The refinement block (b) is added to each skip connection where the direct information from a lower level is refined before merging with the higher level information from a decoder block.

2.1.2 Decoder Block

The decoder performs multiple upsampling to restore the initial resolution of the input image. To help decoder with the restoration of low-level features from compressed spatial information, skip connections are employed to directly connect the output of the lower-layer encoder to its corresponding decoder [24]. Instead of using the information provided by the corresponding encoder blocks without any modifications we refine the information by performing a 3×3 depthwise separable convolution. The resulting refined information is concatenated with the upsampled information. This specific refinement technique is reminiscent of the refinement module proposed in SharpMask [22, 33]. A decoder block with a refinement block is illustrated in Figure 4. In this work, we connect the feature map of encoder #1 and encoder #5 to decoder #2 and decoder #1, respectively. In the final decoder block, we perform $4\times$ upsampling instead of the usual $2\times$ to shorten the decoding pipeline.

2.1.3 Enhancement Block

As the decoder block keeps upsampling the feature map, there is no way to enhance the predictions of neighboring values. To tackle this problem, we insert two enhancement blocks in the middle of the decoding phase. Rather than designing a new block, we share the same architecture with encoder block. The only difference between enhancement block and encoder block is that depthwise convolution with stride two is removed because the enhancement block should sustain the resolution of a feature map. In the ablation study, we show the effectiveness of the enhancement block.

2.2. Loss Functions

The alpha loss and the compositional loss are frequently used in matting tasks. The alpha loss L_α, measures the mean absolute difference between the ground truth mask and the mask predicted by the model. The compositional loss L_c, measures the mean absolute difference between the values of ground truth RGB foreground pixels and the model predicted RGB foreground pixels. The compositional loss penalizes the model when the model incorrectly predicts a pixel with high value.

$$L_\alpha = \frac{1}{K}\sum_{i=1}^{K}\left|\alpha_i - \alpha_i^{\text{gt}}\right| \tag{2}$$

$$L_\text{c} = \frac{1}{3K}\sum_{i=1}^{K}\sum_{j=1}^{3}\left|\alpha_{ij}I_{ij} - \alpha_{ij}^{\text{gt}}I_{ij}\right|, \tag{3}$$

where the K is equal to the width time height, $W \times H$, and α is a vectorized alpha matte where each pixel value is indexed by subscript i. The gt superscript denotes the alpha matte is from ground truth.

We use the KL divergence between the ground truth $A^{\text{gt}} \in \mathbb{R}^{W\times H}$ and the model predicted $A \in \mathbb{R}^{W\times H}$. The KL divergence is defined to be:

$$L_{\text{KL}} = -p(A^{\text{gt}}) \log \frac{p(A)}{p(A^{\text{gt}})} \tag{4}$$

$$= -p(A^{\text{gt}}) \log p(A) + p(A^{\text{gt}}) \log p(A^{\text{gt}}). \tag{5}$$

The second term is the entropy of the ground truth alpha matte, which is constant with respect to model predicted A. Removing the second term leads to optimization of the following loss:

$$\tilde{L}_{\text{KL}} = \sum_{i=1}^{K}\left(\alpha_i^{\text{gt}} \log \alpha_i + (1-\alpha_i^{\text{gt}}) \log(1-\alpha_i)\right). \tag{6}$$

Two additional loss terms are included in the loss function. An auxiliary loss [31] L_{aux}, helps with the gradient flow by including an additional KL divergence loss between the downsampled ground truth mask and the output of the encoder block #10. A gradient loss L_{grad}, guides the model to capture fine-grained details in the edges. We use Sobel-like filter $\mathbf{S}$:

$$\mathbf{S} = \begin{bmatrix} -\frac{1}{8} & 0 & \frac{1}{8} \\ -\frac{2}{8} & 0 & \frac{2}{8} \\ -\frac{1}{8} & 0 & \frac{1}{8} \end{bmatrix}, \tag{7}$$

to create a concatenation of two image derivatives $\mathbf{G}(A) = [\mathbf{S} * A, \mathbf{S}^T * A]$ where $*$ is a convolution. The resulting $\mathbf{G}(\cdot)$ yields a two-channel output that contains the gradient information along x-axis and y-axis. We apply $\mathbf{G}(\cdot)$ to both the ground truth mask and the model predicted mask to

compute the mean absolute differences. The gradient loss is computed as follows:

$$L_{\text{grad}} = \frac{1}{K}\sum_{i=1}^{K}\left|\mathbf{G}(A)_i - \mathbf{G}(A^{\text{gt}})_i\right| \tag{8}$$

$$= \frac{1}{2K}\sum_{i=1}^{K}\Big(\left|(\mathbf{S}*A - \mathbf{S}*A^{\text{gt}})_i\right| + \left|(\mathbf{S}^T*A - \mathbf{S}^T*A^{\text{gt}})_i\right|\Big). \tag{9}$$

The following Equation 10 depicts the loss function of our proposed network.

$$L = \beta_1 L_\alpha + \beta_2 L_{\text{c}} + \beta_3 \tilde{L}_{\text{KL}} + \beta_4 L_{\text{grad}} + \beta_5 L_{\text{aux}}, \tag{10}$$

where we set β values to control the influence of each loss terms. We set them to have equal values of one for the following experiments.

3. Experiments

Automatic portrait matting takes input image with a portrait and denotes each pixel with a linear mixture of the foreground and the background.

We use data provided by Shen et al. [29] which consists of 2,000 images of 600×800 resolution where 1,700 and 300 images are split as training and testing set respectively. To overcome the lack of training data, we augment images by utilizing scaling, rotation and left-right flip. First, an image is rescaled to the input size of the model and random scaling factor is selected from 1 to 1.15. The image is then scaled with the selected factor. Rotation by $[-15°, 15°]$ is applied with a probability of 0.5 which means that half of the augmented images are not rotated. Additional cropping is computed to make the size of the image to match the input size of the model. Finally, the left-right flip is also applied with a probability of 0.5.

To train our model, we optimize our proposed model with respect to the loss function in Equation 10 using Adam optimizer with a batch size of 32 and a fixed learning rate of 1×10^{-4}. Input images were resized to 128×128 and 256×256. The model trained on 128×128 images are faster but produces worse alpha mattes compared to the model trained on 256×256 images. Weight decays were set to 4×10^{-7}. All experiments are conducted using a TensorFlow [3] trained on a single Titan V GPU.

Following the work of Zhu et al. [39], we used gradient error to evaluate our model in portrait matting problem. The gradient error as a metric, which is different from gradient loss, is defined as:

$$\frac{1}{K}\sum_i \left\|\nabla\alpha_i - \nabla\alpha_i^{\text{gt}}\right\|, \tag{11}$$

where α is the alpha matte predicted by the model, and α^{gt} is the corresponding ground truth and K is equal to width $\times$ height. ∇ denotes the differential operator that is computed by convolving the alpha map with first-order Gaussian derivative filters with variance 1.4 [23].

Another metric we use to evaluate our model is the mean absolute differences (MAD). The MAD is defined as follows:

$$\frac{1}{K}\sum_i \left|\alpha_i - \alpha_i^{\text{gt}}\right|. \tag{12}$$

For a fair comparison with previous methods, we scale the predicted alpha matte to the original size of input images, 600×800 in this case, and calculate evaluation metrics.

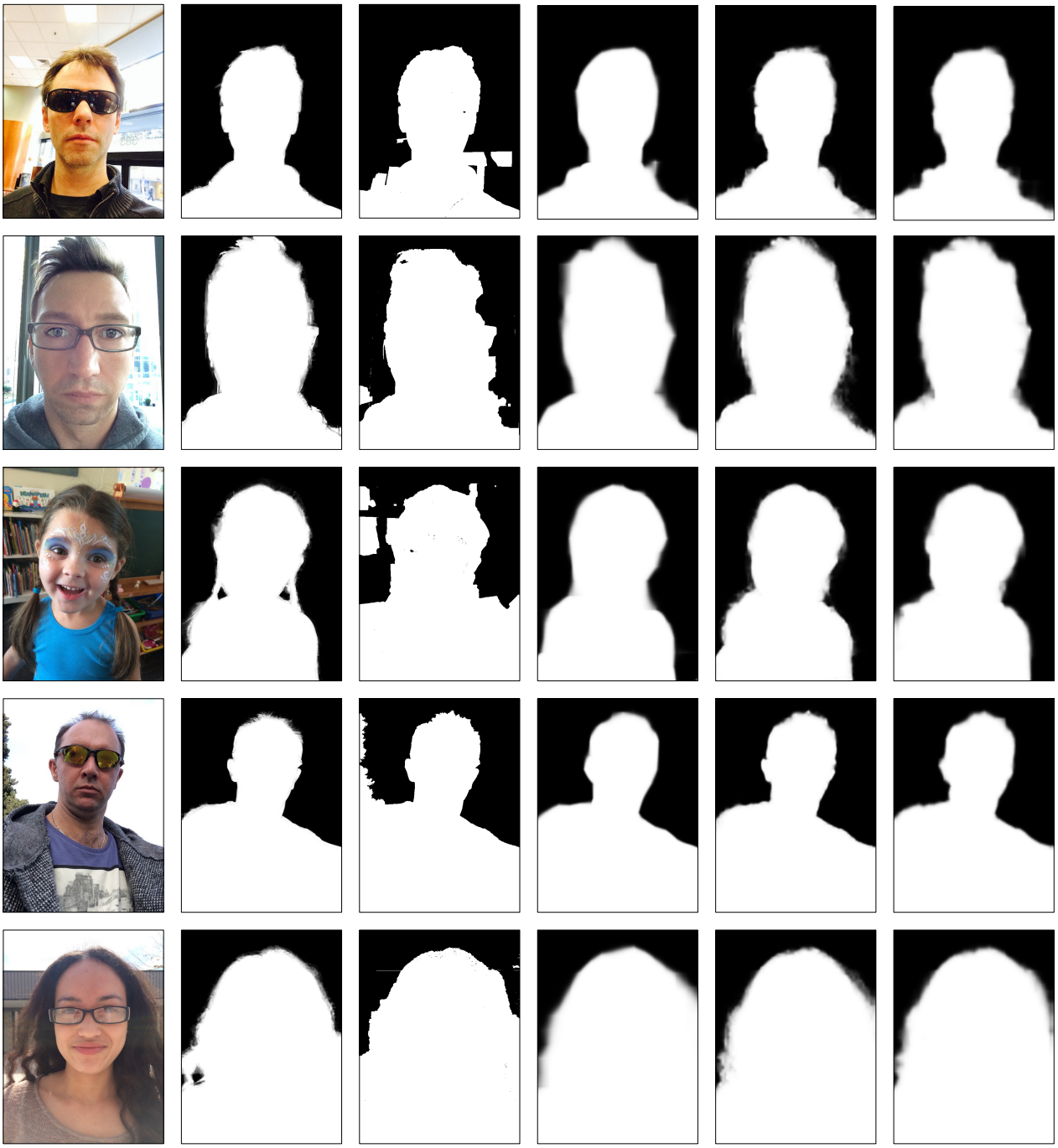
We compare our model to DAPM [29], LDN+FB[39], and Mobile DeepLabv3 [26]. Mobile DeepLabv3 exploits MobileNetV2 as its feature extractor and has its atrous spatial pyramid pooling (ASPP) module removed as suggested by Sandler et al. [26]. We use Equation 10 to optimize Mobile DeepLabv3 in equal footings as MMNet, but remove the auxiliary loss since it requires a modification to the network architecture.

4. Results

4.1. Matting Performance

Table 2 compares the result of DAPM [29], LDN+FB [39], Mobile DeepLabv3 [26], and the proposed method. Input images were scaled to 128×128 or 256×256, depending on the hyper-parameter. When smaller images are fed into the network, the latency drops considerably at the expense of the quality of the alpha matte. Input images were rescaled back to their original resolutions before evaluation. The gradient error and the latency for DAPM and LDN+FB were reported by Zhu et al. [39]. For a fair comparison, we compute the latency of the models on a Xiaomi Mi 5 device (Qualcomm Snapdragon 820 MSM8996 CPU), as suggested by Zhu et al. [39]. Since Zhu et al. [39] did not report how much CPU resources they used, we measure the latency by restricting the use to a single core. Specifically, we use TensorFlow Lite [2] benchmark tool to compute the latency of Mobile DeepLabv3 and MMNet by averaging 100 runs of the model inference on a Xiaomi Mi 5 device while restricting the models to use a single thread.

Zhu et al. [39] reports that DAPM takes 6 seconds on a computer with Core E5-2600 @2.60Ghz CPU. MMNet-1.0 outperforms DAPM while running orders of magnitude faster on a mobile CPU. When the input image is resized to 128×128 for faster inference, our model attains real-time inference, surpassing the rate of 30 frames per second. The real-time version of MMNet is still competitive against DAPM with a moderate increase in its gradient error.



(a) Input Image (b) Ground Truth (c) Graph Cut (d) MD16-0.75 (e) MMNet-1.00 (f) MMNet-1.00*

Figure 5. Visual comparison of different models. Graph Cut [25] results were obtained using OpenCV library [1]. The column marked with $*$ displays the result using 128 inputs. MMNet is better able to construct delicate details compared to other models. Note that MMNet with 128×128 input still outputs a reasonable alpha matte despite its reduced capacity.

The visual comparison of alpha matte in Figure 5 illustrates the qualitative differences of different models. MMNet is better able to construct the finer details compared to other models. Even the real-time version of MMNet produces a reasonable alpha matte regardless of its reduced capacity.

Method	Time (ms)	Gradient Error ($\times 10^{-3}$)
Graph-cut trimap†	-	4.93
Trimap by [28]†	-	4.61
Trimap by FCN [20]†	-	4.14
Trimap by DeepLab [6]†	-	3.91
Trimap by CRFasRNN [38]†	-	3.56
DAPM [29]	-	3.03
LDN+FB [39]	140	7.40
MD16-0.75	146	3.23
MD16-1.0	203	3.22
MD16-0.75∗	38	3.71
MMNet-1.0	129	2.93
MMNet-1.4	213	**2.86**
MMNet-1.0∗	**32**	3.38

Table 2. Model comparisons on the test split. Time is computed on Xiaomi Mi 5 phone. Mobile DeepLabv3 used output stride of 16. Floating point numbers in the method name indicate the width multiplier. The row marked with ∗ displays the result using 128 inputs. Our model outperforms other models while processing images at a faster rate. The experiments marked with † are copied from Shen et al. [29].

4.2. Real-Time Inference on Mobile Devices

To examine the trade-off between execution time and model performance, we explore the model space by varying the width multiplier values and the input resolution. We compare our model with Mobile DeepLabv3 suggested by Sandler et al. [26]. Table 3 details the result of the experiment. The results are sorted by the latency and models with comparable execution time are clustered using horizontal dividers. We see that our proposed model dominates Mobile DeepLabv3 in all clusters in terms of gradient error. Also, note that the number of parameters differs by an order of magnitude. Requiring a small number of parameters is especially appealing if we target a mobile device since end-users do not have to download a bulky model whenever there is an update of the model.

Figure 1 plots trade-off between gradient error and latency on a mobile device. Note that MMNet develops a Pareto-front in this space and outperforms other models. Latency comparison of Pixel 1 and iPhone 8 are included in the supplementary material.

4.3. Ablation Studies

Our proposed network owes its performance to several building blocks utilized in its model architecture. We analyze the impact of each design choices by performing ablation experiments.

Method	Time (ms)	Gradient (10^{-3})	MAD (10^{-2})	Params (M)
MD16-0.75	146	3.25	**2.31**	1.327
MMNet-1.00	**129**	**2.93**	2.48	0.199
MD8-0.75∗	113	3.53	2.61	1.327
MMNet-0.75	90	**2.99**	2.65	0.127
MD16-0.50	82	3.36	**2.53**	0.454
MD8-0.50∗	66	3.61	2.85	0.713
MMNet-0.50	**61**	3.17	2.83	0.069
MMNet-1.40∗	55	**3.38**	**2.72**	0.369
MD16-1.00∗	53	3.68	2.88	2.142
MD8-0.35∗	44	3.72	3.07	0.454
MD16-0.75∗	38	3.77	2.96	1.327
MMNet-1.00∗	**32**	3.44	2.97	0.199
MMNet-1.00Q	98	2.88	2.47	0.199

Table 3. Comparison of MMNet against Mobile DeepLabv3. Floating point numbers in the method name indicate the width multiplier. The row marked with ∗ displays the result using 128 inputs. Output strides of 8 and 16 were tested for Mobile DeepLabv3. Note that the proposed model dominates Mobile DeepLabv3 when the latency is less than 60. In slower regime, MMNet still outperforms Mobile DeepLabv3 in gradient error but are sometimes worse in MAD. Quantized model is included in the last row.

4.3.1 Network Component

Dilation Rates in Encoder Block We study the effect of different dilation rates in the encoder block. The proposed model contains a multi-branch dilated convolutions in the encoder block. We analyze the impact of this decision by fixing the dilation rates to one.

Refinement Block Whenever there is a skip connection, we have included a refinement block to improve the decoding quality. The refinement block enhances the result of the encoder block by performing 3×3 depthwise separable convolution followed by batch normalization and a ReLU6 non-linearity. We remove the refinement block and study its impact on the final result.

Enhancement Block The enhancement blocks are intended to give the network a layer to improve the final result before its resolutions are fully recovered. We study the effect of the enhancement block by removing it entirely from the network.

Table 4 illustrates the results when different components of the model architecture are modified. We see that all the components contribute to the final performance of the proposed model. When the dilation rate is fixed to one, the

Method	Gradient Error ($\times 10^{-3}$)
No dilation	3.25
No enhancement in decoding	3.04
No refinement in skip connection	3.07
Proposed model	2.93

Table 4. Ablation study on the test split of matting dataset. All experiments are performed using MMNet with width multiplier of 1.0.

network has a hard time generalizing due to its limited effective receptive field. Enhancement and refinement in the decoding phase also boost the network performance.

4.4. Quantization

We demonstrate the full pipeline for training a real-time portrait matting model targeting a mobile platform by incorporating quantization of our model. Quantization of model parameters and its activation reduces the bit-width required by the model. The reduction of bit-width allows one to exploit integer arithmetics in boosting the network inference speed.

The target model undergoes a quantization-aware training phase via fake quantization [17]. While maintaining full precision weights, tensors are downcasted to fewer bits during the forward pass. On a backward pass, the full precision weights are updated instead of downcasted tensors from which the gradients are computed. Once the training is complete, quantized models are executed using the TensorFlow Lite framework [2].

Table 3 contains the result of 8-bit quantized model. The model enjoys 25% decrease in latency and better gradient error. The details for quantization are included in the supplementary material.

5. Related Work

Image matting task has been mostly approached using sampling [12, 13, 15, 27, 34] or propagation-based [8, 14, 19, 30] ideas. Recently, with the success of convolutional neural networks (CNN) in computer vision tasks, there has been a growing number of works utilizing CNNs. Cho et al. [10] proposed end-to-end network which relies on other matting algorithms' outputs, such as the closed form matting [19] and the KNN matting [8], to produce the final alpha matte. Shen et al. [29] proposed an automatic image matting method leveraging CNN to create a trimap which is fed to closed form matting [19] by backpropagating the matting error back to the trimap network. Xu et al. [36] take the approach further by directly learning the alpha matte. Chen et al. [9] combine trimap generation and alpha matte generation using a fusion module.

Many works on image matting are mainly focused on achieving higher accuracy rather than the real-time inference of models. But recently, researchers are shifting the focus to networks that accommodate real-time inference [39]. Zhu et al. [39] studied real-time portrait matting on mobile devices which is directly comparable to our result.

Since the work of Long et al. [20], fully convolutional networks (FCN) have been widely used in various segmentation tasks [18, 37]. Many of the semantic segmentation networks adopt encoder-decoder structure [4]. The proposed model uses skip connections to concatenate the output of an encoder block to a decoder block which has been known to improve the result of semantic pixel-wise segmentation tasks [24].

Chen et al. [6] proposed DeepLab [5, 7] architecture which extensively uses the ASPP module. ASPP module aims to solve the problem of efficient upsampling and handling objects at multiple scales. Our model adopts a multi-branch structure from Inception network [31], together with the dilated convolution of different dilation rates, which resembles the ASPP module.

One of the most prominent light-weight neural networks is MobileNet and its variants [16, 26]. Depthwise separable convolution was shown to be extremely effective in creating a light-weight network while keeping the accuracy drop to a tolerable level.

ENet, an efficient neural network architecture designed with the intention of tackling a semantic segmentation task, was proposed by Paszke et al. [21]. Our work is inspired by the design choices detailed in their work for creating an efficient neural network.

6. Conclusions

In this work, we have proposed an efficient model for performing automatic portrait matting task on mobile devices. We were able to accelerate the model four times to achieve 30 FPS on Xiaomi Mi 5 device with only 15% increase in the gradient error. Comparison against Mobile DeepLabv3 showed that our model is not only faster when the performance is comparable, but also requires an order of magnitude less number of parameters. Through ablation studies, we have shown that our choice of the multi-branch dilated convolution with a linear bottleneck is essential in maintaining high performance. We also make our implementation available at https://github.com/hyperconnect/MMNet.

A general extension of our work is to handle general image matting problem, such as automatic saliency matting. Since we can already achieve real-time, it is natural to extend the work further by tackling the video matting problem as well. Pushing for real-time inference on mobile devices requires a carefully prepared pipeline for it to work in a real-world setting. Distillation to guide the mobile-friendly

model in training and even lower-bit quantization for added speedup is highly desired.

References

[1] OpenCV. https://opencv.org/. Accessed: 2018-10-23.

[2] TensorFlow Lite. https://www.tensorflow.org/lite/. Accessed: 2018-10-23.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[4] V. Badrinarayanan, A. Kendall, and R. Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[5] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.

[7] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European Conference on Computer Vision*, 2018.

[8] Q. Chen, D. Li, and C.-K. Tang. Knn matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(9): 2175–2188, Sept 2013.

[9] Q. Chen, T. Ge, Y. Xu, Z. Zhang, X. Yang, and K. Gai. Semantic human matting. In *Proceedings of the ACM Multimedia Conference*, pages 618–626, 2018.

[10] D. Cho, Y.-W. Tai, and I. Kweon. Natural image matting using deep convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*, 2016.

[11] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pages 1610–02357, 2017.

[12] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[13] E. S. L. Gastal and M. M. Oliveira. Shared sampling for real-time alpha matting. *Computer Graphics Forum*, 29(2): 575–584, May 2010. Proceedings of Eurographics.

[14] L. Grady, T. Schiwietz, S. Aharon, and R. Westermann. Random walks for interactive alpha-matting. In *Proceedings of Visualization, Imaging, and Image Processing*, 2005.

[15] K. He, C. Rhemann, C. Rother, X. Tang, and J. Sun.

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

[17] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[18] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio. The one hundred layers tiramisu: Fully convolutional DenseNets for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017.

[19] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):228–242, 2008.

[20] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[21] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[22] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollar. Learning to refine object segments. In *Proceedings of the European Conference on Computer Vision*, 2016.

[23] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[24] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention*, 2015.

[25] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut": Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3):309–314, August 2004.

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[27] E. Shahrian, D. Rajan, B. Price, and S. Cohen. Improving image matting using comprehensive sampling sets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[28] X. Shen, A. Hertzmann, J. Jia, S. Paris, B. Price, E. Shechtman, and I. Sachs. Automatic portrait segmentation for image stylization. In *Computer Graphics Forum*, volume 35, pages 93–102, 2016.

[29] X. Shen, X. Tao, H. Gao, C. Zhou, and J. Jia. Deep automatic portrait matting. In *Proceedings of the European Conference on Computer Vision*, 2016.

[30] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum. Poisson matting. In *ACM Transactions on Graphics*, volume 23, pages 315–321, 2004.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and*

pattern recognition, pages 2818–2826, 2016.

[33] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, B. Nessler, and S. Hochreiter. Speeding up semantic segmentation for autonomous driving. In *Advances in Neural Information Processing Systems*, 2016.

[34] J. Wang and M. F. Cohen. Optimized color sampling for robust matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[35] M. Wang, B. Liu, and H. Foroosh. Factorized convolutional neural networks. In *ICCV Workshops*, pages 545–553, 2017.

[36] N. Xu, B. L. Price, S. Cohen, and T. S. Huang. Deep image matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[37] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the International Conference on Computer Vision*, 2015.

[38] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the International Conference on Computer Vision*, 2015.

[39] B. Zhu, Y. Chen, J. Wang, S. Liu, B. Zhang, and M. Tang. Fast deep matting for portrait animation on mobile phone. In *Proceedings of the ACM Multimedia Conference*, 2017.

Supplemental Materials

A. Quantization

We used `tensorflow.contrib.quantize` to quantize our model. Custom implementation of `resize_bilinear` operation, optimized using SIMD instructions was deployed. Since we are using fake quantization [17] for quantization-aware training, additional fake quantization node was inserted after a `resize_bilinear` operation.

The quantized version of softmax provided by TensorFlow Lite is slow for our use case since it is optimized for a classification task. Our formulation allows us the make an assumption that the output has only two channels. Quantizing the values to 8-bits means that there are only 65,536 valid logit pairs. Instead of explicit computation of softmax, we precompute the values and substitute the calculation with a table lookup.

B. Latency

Table 5 depicts the latency of different models measured on Pixel 1, Xiaomi Mi 5, and iPhone 8. All measurements are performed with the TensorFlow Lite [2] benchmark tool on a mobile device while restricting the models to use a single thread. The mean and the standard deviation obtained from 100 runs are included in the table. The measurements were separated apart in time to give the device enough time to cool down. Demo video is available at https://github.com/hyperconnect/MMNet.

Method	Pixel 1	Mi 5	iPhone 8
MD16-0.75	142 ± 2	146 ± 1	32 ± 0
MMNet-1.00	127 ± 1	129 ± 1	30 ± 0
MD8-0.75$*$	111 ± 1	113 ± 1	22 ± 0
MMNet-0.75	87 ± 1	90 ± 1	22 ± 0
MD16-0.50	80 ± 1	82 ± 1	19 ± 0
MD8-0.50$*$	66 ± 1	66 ± 1	13 ± 0
MMNet-0.50	60 ± 0	61 ± 1	15 ± 0
MMNet-1.40$*$	53 ± 1	55 ± 1	12 ± 0
MD16-1.00$*$	53 ± 1	53 ± 1	12 ± 0
MD8-0.35$*$	45 ± 1	44 ± 0	9 ± 0
MD16-0.75$*$	38 ± 1	38 ± 1	8 ± 0
MMNet-1.00$*$	33 ± 1	32 ± 1	8 ± 0
MD16-0.75	104 ± 1	-	-
MMNet-1.00Q	98 ± 2	98 ± 1	-

Table 5. Latency of models on different mobile devices. All numbers are in milliseconds. The row marked with $*$ displays the result using 128 inputs. Quantized model is included in the last row.

C. Detailed Architectures

Table 6 illustrates the number of channels used in each component of MMNet. The initial block outputs a 32 channel feature map, as described in the first row. The numbers in the encoder/enhancement columns represent the number of channels returned by the multi-branch 1×1 convolutions and the final output of the encoder/enhancement block after the concatenation. For example, encoder #6 will receive a 40 channel input which the 1×1 convolutions in multiple branches each expand to 64 channels. After the multi-branch, the outputs are concatenated and convoled by a 1×1 convolution which compresses the number of channels back to 40. Whenever there is a skip connection, the output of a decoder block is concatenated with the output of a refinement block. Their respective number of channels are delineated in the decoder rows. The final block returns a two-channel output, each for foreground and background.

Name	Output channels of 1x1 convolution				
	First	Encoder/Enhancement	Decoder	Refinement	Final
Initial Block	32	–, –	–	–	–
Encoder 1	–	16, 16	–	–	–
Encoder 2	–	16, 24	–	–	–
Encoder 3	–	24, 24	–	–	–
Encoder 4	–	24, 24	–	–	–
Encoder 5	–	32, 40	–	–	–
Encoder 6	–	64, 40	–	–	–
Encoder 7	–	64, 40	–	–	–
Encoder 8	–	64, 40	–	–	–
Encoder 9	–	80, 80	–	–	–
Encoder 10	–	120, 80	–	–	–
Decoder 1	–	–	64	64	–
Decoder 2	–	–	40	40	–
Enhancement 1	–	40, 40	–	–	–
Enhancement 2	–	40, 40	–	–	–
Decoder 3	–	–	16	–	–
Final Block	–	–	–	–	2

Table 6. The number of channels in different components of the proposed network.