Report on Computer Vision project:
“Portrait blurring effect with Deep Learning methods”

Vladislav Pyzh
Igor Gogarev
M1 AI & ViC

1. Introduction

1.1. Problem statement

In this project we address the problem of creating portrait images with software-made blur effects. This idea has gained popularity in recent years. It is evident in features that different software and mobile device companies offer: for instance, there is a special “Portrait Mode” in iPhone cameras, which became a sought-after feature among users. We would like to tackle this challenge with deep learning by training different models from scratch. We separate the problem into three tasks:

1. **Implement a person segmentation pipeline.** This allows us to select the region that we would like to keep unchanged. From a more general perspective we can change the task from person segmentation to the segmentation of the main object in the picture. This is a direction for possible improvements of the model.
2. **Implement a monocular deep estimation.** We believe that depth estimation can enhance blur effects and thus we want to incorporate this in our pipeline. A deep estimation with several images is a problem that can be solved using disparity map and understanding how cameras are positioned in space. However, in our scenario we would like to estimate depth from only one picture. It is evident that a human has some intuition about monocular depth estimation (which sometimes can be deceived) so the overall idea of that is not meaningless. There are a lot of articles on this problem using deep learning techniques.
3. **Combine results from previous stages.** Even with very good models for segmentation and deep estimation, the problem of applying these models to the real image arises. Creating a seemingless combination with proper estimation of blurring parameters is a problem that we want to solve during this stage.

1.2. Used frameworks

The whole deep learning pipelines implemented with PyTorch framework with the version ≥ 2.0.1. During experiments on semantic segmentation models we use PyTorch Lighting with TensorBoard for metrics logging and WeightAndBiases for logging during training Depth estimation models. Experiments with semantic segmentation models were conducted on Nvidia 2080ti GPU with 11Gb of memory, while training depth estimation models were conducted using Kaggle provided resources. Training and experiments took approximately 40 hours.

For the work with images and matrix operations we use the open-cv and numpy frameworks for Python.

2. Method research and development

2.1. Person Segmentation

2.1.1. Literature overview

As a part of the image enhancement pipeline we do not want to distinguish different instances of persons. However, the problem of semantic segmentation of a person is not very popular in the academic sphere. More widespread problems are instance segmentation or human part segmentation. Both these tasks are well represented on papers with code website. In our work we decided to implement two different approaches proposed for semantic segmentation without focus on person segmentation. The first model that we implemented was a classic UNet model proposed for biomedical image segmentation in “U-Net: Convolutional Networks for Biomedical Image Segmentation”. Being very simple and at the same time proposing several important ideas this model is widely spread and used on many computer vision courses. The second approach that we implemented is obtaining an embedding from a pretrained complex model and after that doing decoding from this result. As a pretrained encoder we decided to use an EfficientNet, described in “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. We give a more thoughtful description of this model later in this paper.

Another important aspect of this problem is the evaluation metric and loss used during training. The most common approach that is used for this problem is optimizing cross-entropy loss since this problem can be seen as binary (in more general approach multi) label classification. At the same time Intersection-Over-Union is used as a standard metric for segmentation and detection problems, but there is no direct connection between cross-entropy and IoU. In our work we implemented the SoftIoU loss proposed in the “Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation” paper. More details about loss implementation and results can be seen in the next sections.

2.1.2. Dataset and data processing

For this task we used a public dataset published on hackernoon by DeepSystems. Dataset consists of 5678 images with 6884 high-quality annotated human instances. The dataset was annotated by only two persons, which makes annotations consistent and authors

claim that this dataset has a better annotation quality than others publicly open datasets. We split this dataset into train and validation parts with sizes 5110 and 568 correspondingly.

Comparison of different Dataset: DeepSystem's dataset (on the left) provides a significantly more accurate mask for person segmentation than COCO (on the right)

A widely used approach for image preprocessing is an image resizing to one fixed size, the most common settings are 512x512 or 224x224 sizes. After obtaining a corresponding image usually a simple upsampling is used to get an image of the original size. We believe that this approach can lower the model's quality for images with sizes ratio very different from squarelike. Changing sizes ratio drastically changes image geometry and objects' proportions. Therefore for our purposes we implemented a different approach. UNet and EfficientNet highly rely on the fact that edge size is divisible by 32, but this doesn't imply that sizes have to be equal. In our dataset we separated images into three different types: horizontal, vertical and squarelike. For every part we do a corresponding resizes: to 288x512 for horizontal, 384x384 for squarelike and 512x288 for vertical. After that we split the dataset so that in every batch we get images of only one type.

2.1.3 Loss and evaluation functions

Classic approaches for training semantic segmentation models use cross-entropy loss as for pixels classification problem, below C denotes number of classes and s_i are logits obtained from the model.:

$$L_{CE} = -\sum_{i}^{C} t_i \log\left(\frac{e^{s_i}}{\sum_{i}^{C} e^{s_i}}\right)$$

However, for evaluation purposes we chose the IoU function, which is not smooth (it means we can't optimize it).

$$IoU = \frac{Intersection}{Union} = \frac{\#TP}{\#TP + \#FP + \#FN}$$

As mentioned in the literature overview one of the approaches proposed to tackle this problem is using soft intersection over union from the paper "Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation". Here <> means element-wise matrix multiplication and summing results.

$$Intersection = < X, Y >$$

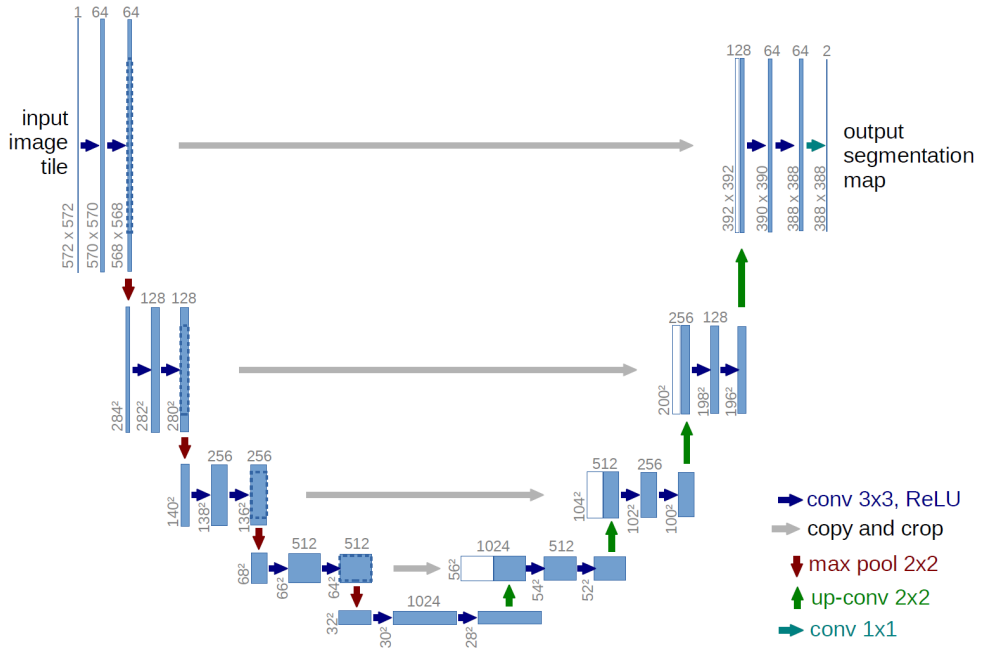
$$Union = X + Y - < X, Y >$$

$$L_{soft-IoU} = - \frac{<X,Y>}{X + Y - <X,Y>}$$

In our work we tested both loss functions, but as a possible way for improvement there can be conducted more experiments on class weights distribution for a cross-entropy loss, since usually on photos a person covers much less than a half of space.

2.1.4 Models

Classic UNet model consists of encoder and decoder with a passing and concatenation of intermediate results between corresponding levels of them. This approach is illustrated below. Despite the fact that the vanilla approach uses convolutions that decrease sizes of images, this problem can be easily mitigated by applying corresponding pooling parameters for every convolution. Therefore, on every level of encoder and decoder we apply two blocks of combination convolution – batch normalization – ReLU. After that on every encoder level we apply max pooling operation and on every decoder we apply transposed convolution (also known as fractionally-strided convolution or a deconvolution). One of the many disadvantages that can be seen in this model is that applying a concatenation operation for tensors takes a lot of time, which reduces the model's speed.



Second model that we implemented in our work was a model with a pretrained classification model for obtaining embeddings and after that decoding this into the image with the same size. We do not pass any intermediate results from embedder to decoder in this approach. As an encoder we decided to use EfficientNet-B3. Authors of the “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks” proposed several efficient features that provided state-of-the-art quality with respect to its time.

EfficientNet utilizes several MBConv blocks that originally were proposed in the MobileNetV2 paper. These blocks are key to the model's efficiency and effectiveness. MBConv, or Mobile Inverted Bottleneck Convolution, is an advanced block design that incorporates depthwise separable convolutions, allowing for a significant reduction in computational cost and model size while maintaining or even improving performance. At the same time authors added squeeze-and-excitation optimization that allows models to emphasize important channels and suppress useless ones.

Second important idea that was implemented in the EfficientNet is a proper scaling of dimensions. Authors noticed that previous works don't address the question of how to do scaling properly. Applying a systematic approach to that allowed to develop a more efficient architecture. This is illustrated below with a graph from the original paper.

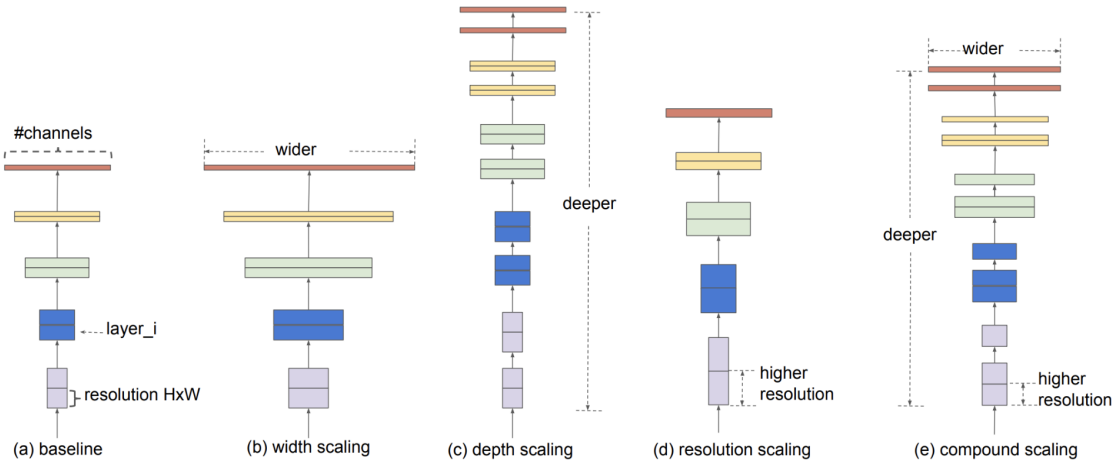


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

As a decoder network we use five transposed convolutions separated by three blocks of convolution – batch normalization – Mish layers. After the last upsampling we apply one block of convolution – batch normalization – Mish and a convolution into two classes, that provides the final logits. We use Mish, an activation function, that is claimed to provide a better performance than Swish and ReLU.

2.1.5 Experiments setup

As an optimizer, we used Adam with a learning rate set to 1e-3, beta parameters set to (0.9, 0.999), and no weight decay. To obtain the best possible results from our model, we implemented a learning rate scheduler, ReduceLROnPlateau, which monitors the validation loss. Our training stop criterion is early stopping, which halts training if there is no improvement in evaluation metrics during the last ten epochs. For both the UNet and

encoder-decoder models, we used a batch size of 8. As a baseline setup for augmentations, we employed horizontal flips and conservative rotations (maximum value = 15 degrees) because these augmentations do not significantly alter the data domain. We also have tried several augmentations like color changes and possible color normalization.

2.1.6 Results

Results can be seen in this table, plotting graphs can be found in the appendix. Conclusions we made based on the results of the experiments:

- Our implementation of UNet slightly outperforms a pretrained UNet imported from torch vision hub.
- Changing UNet to encoder-decoder model improves greatly both performance and speed
- Using Soft Intersection-over-Union provides a very slight improvement with comparison to cross-entropy. Perhaps tuning class weights for cross-entropy can bridge this gap.
- Using color augmentations and normalization didn't provide any improvement for validation IoU. This might be caused by the big size of our dataset, which already provides a sufficient amount of different examples.

Model	Validation IoU
Pretrained UNet from Torch Hub	0.811
Our UNet implementation	0.821
Decoder-Encoder with CE loss	0.883
Decoder-Encoder with Soft IoU loss	**0.885**
Decoder-Encoder with Soft IoU + normalization	0.879
Decoder-Encoder with Soft IoU + color augmentation	0.879

2.1.7 Results post-processing and model's mistakes

During evaluation of our model's quality we noticed two problems, described below. Examples of model predictions and illustrations for these problems can be found in the appendix.

1. Dataset provides a lot of examples with people holding something in their hands and the object in hand isn't covered by a mask. We believe that this factor encourages models to not mask people's hands even if they are visible. There are two possible solutions for this problem. First: filter these examples from a dataset which can be done with relatively cheap crowdsourcing. Second: create a model that will be an expert in classification of images containing hands/ not containing hands. Providing this information as a hint to the segmentation model can help it with better segmentation.
2. Dataset provides examples where complex human's pose cause empty gaps in the mask. This forces models to create false gaps like these in the images without any gap. To tackle this we provide a small script that uses a bfs algorithm to fill all the gaps inside of the masks. This problem is very rare and overall this approach decreases the model's performance, thus we leave use of it at the discretion of the user.

2.1.8 Possible improvements

We believe that small improvements can be obtained with further experiments on different augmentations and training parameters, but this direction requires a significant amount of time and computing resources.

Speaking from the architecture perspective it's meaningful to research how image classification models can be converted into semantic segmentation models with weights preserving. This can be done by changing fully connected layers with convolution layers, pooling and stride parameters to preserve spatial dimensions and using dilated convolutions. Although it is a challenging (or even impossible task) for EfficientNet, this is more realistic for small simple models such as ResNet18. Modification of encoder also allows passing features from encoder to decoder, which might be beneficial.

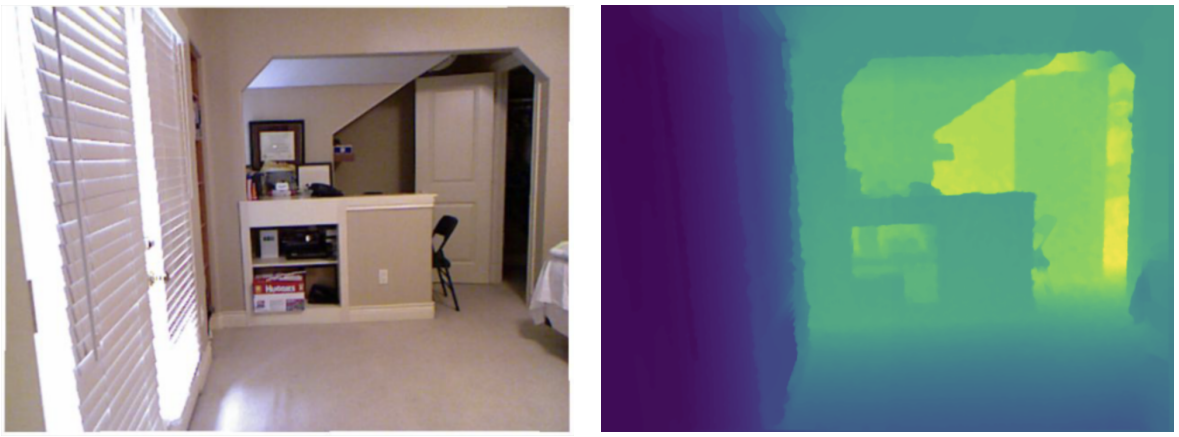
Together with this we believe that research in the direction of different non-transformers models like DeepLabV3 or ConvNeXt can be beneficial and at the same time feasible for modern PCs or Kaggle resources.

2.2. Depth map estimation

2.2.1. Literature overview

We faced the depth map estimation problem and, as it is a widespread problem, we used to do a literature review. As we were tackling the problem of estimating the depth of the object given only one image, we considered only the monocular depth map estimation solutions.

The main paper we studied and that we used for our depth map estimation implementation is High Quality Monocular Depth Map Estimation via Transfer Learning [12] - we used it as the proposed method shows high performance.



Depth Map Estimation example

2.2.2. Dataset review

We used the NYU Depth v2 dataset, which provides images and corresponding depth maps of indoor scenes. The resolution of images is set to be 640x480 pixels. As we were limited in our quantitative resources (we used Kaggle Free GPU), we decreased the number of samples from 120000 to 30000 with the training split fraction set to 0.7.

The depth value in the dataset has the upper bound of 10 meters.

The augmentations we used were the same as in the original paper and included the $Image.FLIP\_LEFT\_RIGHT$ rotation with the probability 0.5 and rearranging the channels with the probability 0.5.

2.2.3. Depth Map estimation model overview

The model architecture is an encoder-decoder one. This architecture has shown its ability to achieve high performance in different tasks, such as semantic segmentation from the previous section, also Autoencoders, which are the implementation of encoder-decoder architecture, are often used.

As an encoder the pretrained DenseNet-169 model was used. According to PyTorch official documentation, this model has 14.1 million parameters, which is also a big number considering our computing resources limitations. After reviewing available models, we decided to use a pre-trained MobileNet V2 model, as it consists only of 3.5 million parameters, and shows slightly worse performance on the baseline (on ImageNet-1k dataset accuracy-top1 is 72.154 against 75.6 for DenseNet-169). We used an implemented architecture from the repository [14].

The decoder part of model is a sequence of the following blocks:

1. Bilinear upsampling of the previous decoder output
2. Concatenation of the upsampled decoder output with the corresponding encoder layer output
3. Applying the convolution $Conv2d\ (kernel\ size\ =\ 3,\ stride\ =\ 1,\ padding\ =\ 1)$
4. Applying $LeakyReLU(0.2)$
5. Applying the convolution $Conv2d\ (kernel\ size\ =\ 3,\ stride\ =\ 1,\ padding\ =\ 1)$
6. Applying $LeakyReLU(0.2)$

In total, the decoder model consisted of 6 blocks.

2.2.4. Loss functions

The loss function we used was described in the paper. In general, it consisted of 3 terms, which are summed with different coefficients.

First of all, the point-wise $L1\ Loss$ is calculated. This is quite an obvious loss, which shows how good the model is predicting the depth values for certain pixels. In the original paper this term coefficient is set to 1.

$$L_{depth}(y, \hat{y}) = \frac{1}{n}\sum_{p}^{n}|y_p - \hat{y}_p|$$

The second term is also an application of $L1\ Loss$, but not with the original and predicted depth maps, but with its gradients over X and Y axes. This loss is useful, as we

want to control the change of the depth map predictions within a certain object, where the prediction should not change or should change smoothly, or between two separate objects, where the predictions change should be sharp.
In the original paper this term coefficient is set to 1.

$$L_{grad}(y, \hat{y}) = \frac{1}{n}\sum_{p}^{n}|\boldsymbol{g_x}(y_p, \hat{y}_p)| + |\boldsymbol{g_y}(y_p, \hat{y}_p)|$$

The last term is the Structural Similarity Loss (SSIM). This loss is based on the corresponding Structural Similarity metric that is used to quantify the structural similarity between two images. The idea behind this metric is to compare 3 main aspects of the images - luminance, contrast and structure. This metric is commonly used in the image reconstruction tasks, so it is supposed to be efficient in the depth map estimation task too.

$$L_{SSIM}(y, \hat{y}) = \frac{1 - SSIM(y, \hat{y})}{2}$$

As the loss functions tend to be higher for larger target and prediction values, the target depth map is normalized by the rule $maxValue \,/\, value$, were the $maxValue \;=\; 10\, meters$.

The losses changing plots are presented in the Appendix.

2.2.5. Model training

We trained our model for 10 epochs. The optimizer we used was Adam with the default parameter values, as it is used in the original paper and this is one of the most efficient optimizers for the models training. The learning rate was set to 0.0001.

The authors of the paper that we consider are proposing to take a mean of the original image depth map and on the mirrored one (as this augmentation was also provided during the model training process). We used the same technique in our method, assuming this leads to more stable results.

2.2.6. Model evaluation

We used the same metrics for the method evaluation as were used in the paper we consider. There are 3 following metrics:

1. Average Relative Error

$$\frac{1}{n}\sum_{p}^{n}\frac{|y_p - \hat{y}_p|}{y}$$

2. Root Mean Squared Error

$$\sqrt{\frac{1}{n}\sum_{p}^{n}(y_p - \hat{y}_p)^2)}$$

3. Average (log10) Error

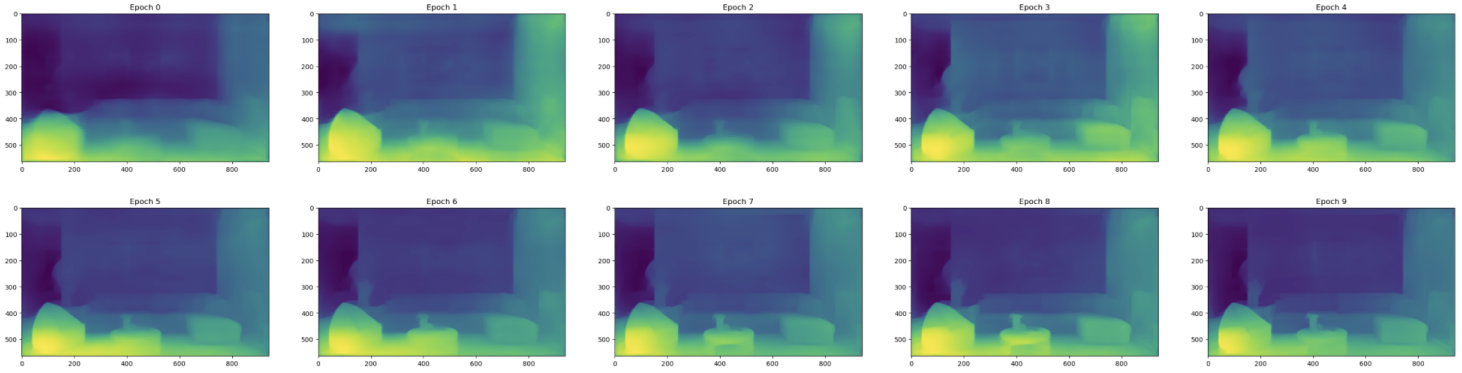
$$\frac{1}{n}\sum_{p}^{n}|\log_{10}(y_p) - \log_{10}(\hat{y}_p)|$$

The metrics plots are presented in the Appendix. The final metric values for our model after 10 epoch training are following:

Metric name	Value
Average Relative Error	0.072
Root Mean Squared Error	0.474
Average (log10) Error	0.031

2.2.7. Model Predictions Assessment

As the model was trained for 10 epochs, we were able to visually assess the model's performance on a certain sample. After first epochs the model is able only to strongly blurred areas without highlighting the objects borders and contours. As training continues, the model's predictions begin to be more precise. At first, the borders of big and most significant objects evolve, and depth values for these objects are predicted correctly. After further training, smaller objects' borders seem to evolve, making the overall depth map prediction more and more precise.



2.2.8. Possible Improvements

As was mentioned before, we were limited in our computing resources, so we tested smaller model architectures. Thus, one of possible ways to improve the method performance might be using larger encoder and decoder architectures.

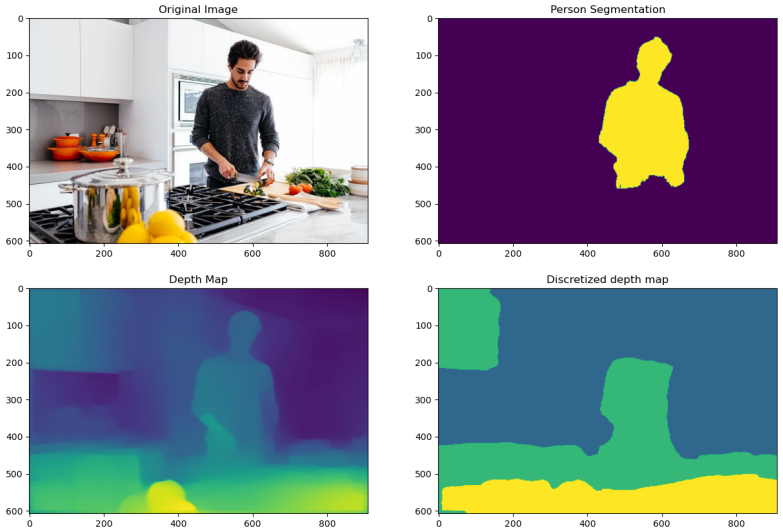
The other way to improve the algorithm is to use other datasets to be able to apply our blurring algorithm not only to the inside scenes, but also to the outside ones.

2.3. Aggregation of the segmentation and depth map estimation models

2.3.1. Baseline algorithm

As an input for the aggregation algorithm we consider binary person segmentation mask and depth map for the same image. The depth values were scaled to the range from 0 to 1. To provide a depth-dependent blurring 3 different kernel sizes (15, 9, 3) for blurring were used in the baseline. The intermediate results and the final image have revealed two key problems in the aggregation algorithm:

1. There were visible borders between different depth neighbors areas.
2. The area around the person was not blurred properly because of the sharp edges in the segmentation mask.

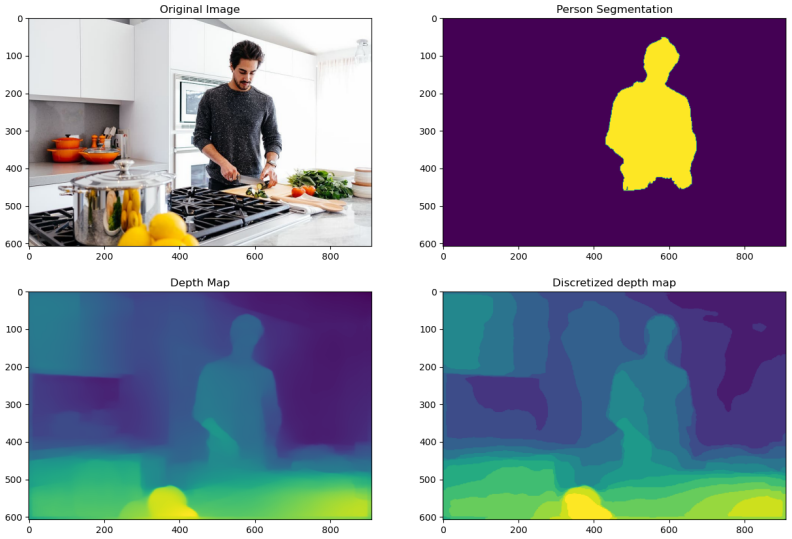


We provided solutions for both of the issues.

2.3.2. Increasing the number of blurring kernels

Our idea was to create smaller segments between 0 and 1 for the depth values and to assign each of them with a blurring kernel that size is closer to the next and previous kernel sizes compared to the baseline solution. For these purposes we created 13 kernels with sizes from 13 to 1 with step 1.

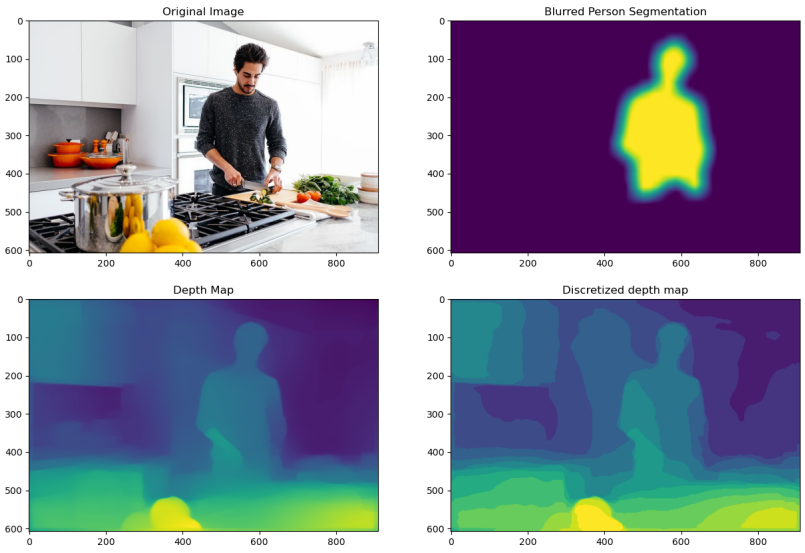
This improvement worked well and we got rid of the borders between different blurring areas.



2.3.3. Blurring the person segmentation mask

We tried to get a smooth transition between the area which was blurred (the background) and the area that was left without any changes (person segmentation area in our case). As we had a person segmentation mask with values from 0 and 1, we applied blurring of this mask with a high kernel size (45 in our case). The produced map (blurred segmentation mask) with values in range from 0 to 1 allowed us to take the weighted sum of two images: we took the person area with weight equal to corresponding blurred segmentation mask value, and sum it with the weight equal to (1 - blurred segmentation mask value) with the blurred background image (see Appendix).

The improvement was efficient and we finally obtained a smooth transition.



3. Conclusion

We have implemented the adaptive blurring method for the portrait type images and achieved the expected results. During the method research and development we have worked on two different Computer Vision problems: person segmentation and monocular depth map estimation.

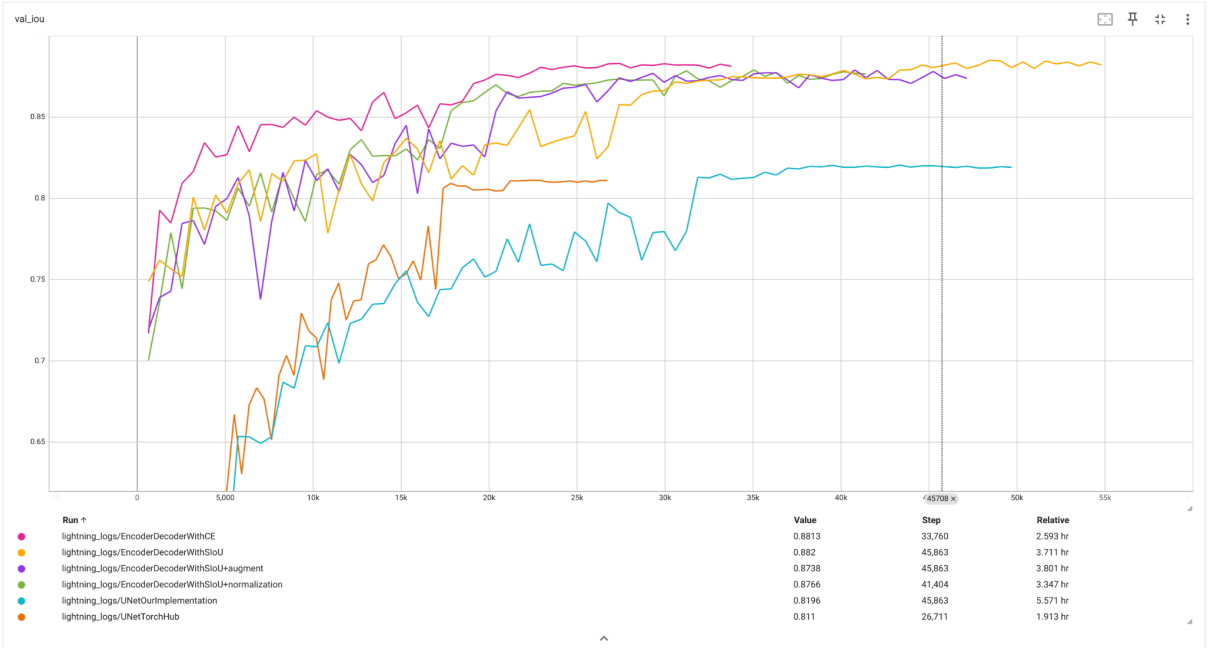
While developing an algorithm for a person segmentation, we tried two different models for semantic segmentation: classical well-known UNet and encoder-decoder approach. Throughout solving this task we implemented dataloader that preserves image sizes ratio, conducted several experiments, provided an efficient training pipeline and solved a problem with occurring gaps in the mask.

In the monocular depth map estimation task we used to analyze the existing solutions by considering the paper [12]. We also used MobileNet V2 based architecture and implemented the L1 gradient loss function and added it to the model training process.

While aggregating the results of the segmentation and depth map models, we faced an issue of having hard blurring transitions around the person and between two neighbors' depth map areas, and provided solutions for both of the cases. The combination of two models and Computer Vision image techniques allowed us to get the expected result of adaptive blurring.

Appendix

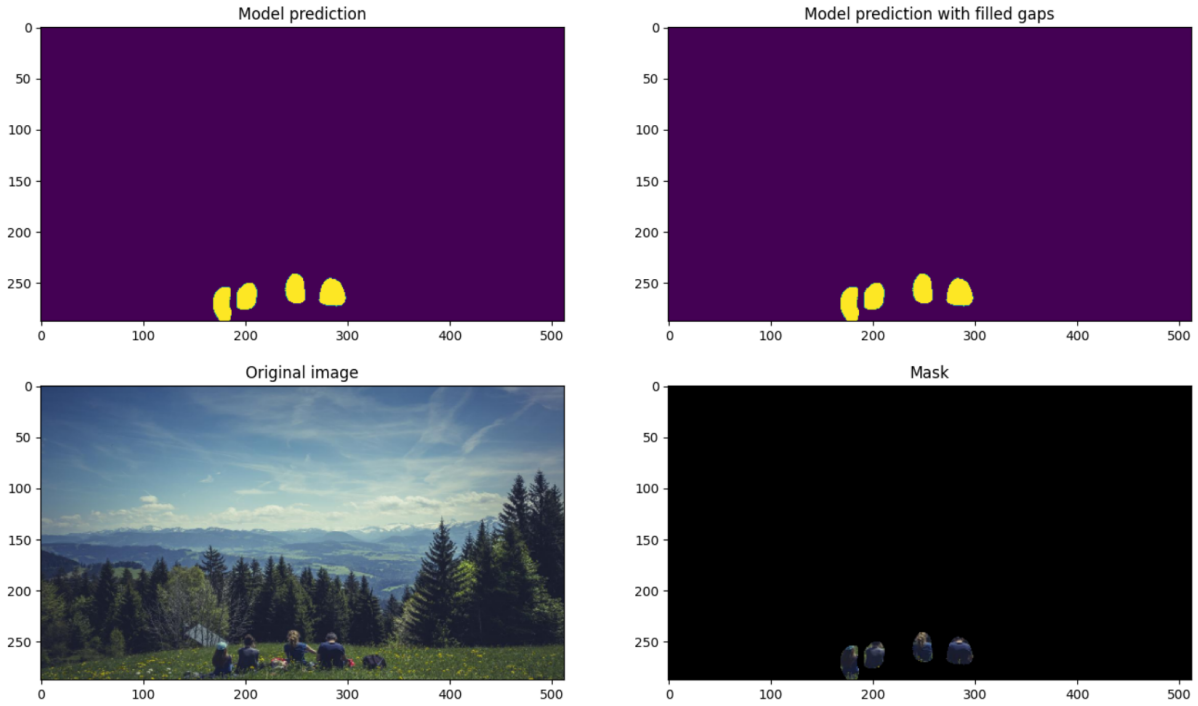
A plot illustrating training process for semantic segmentation models.



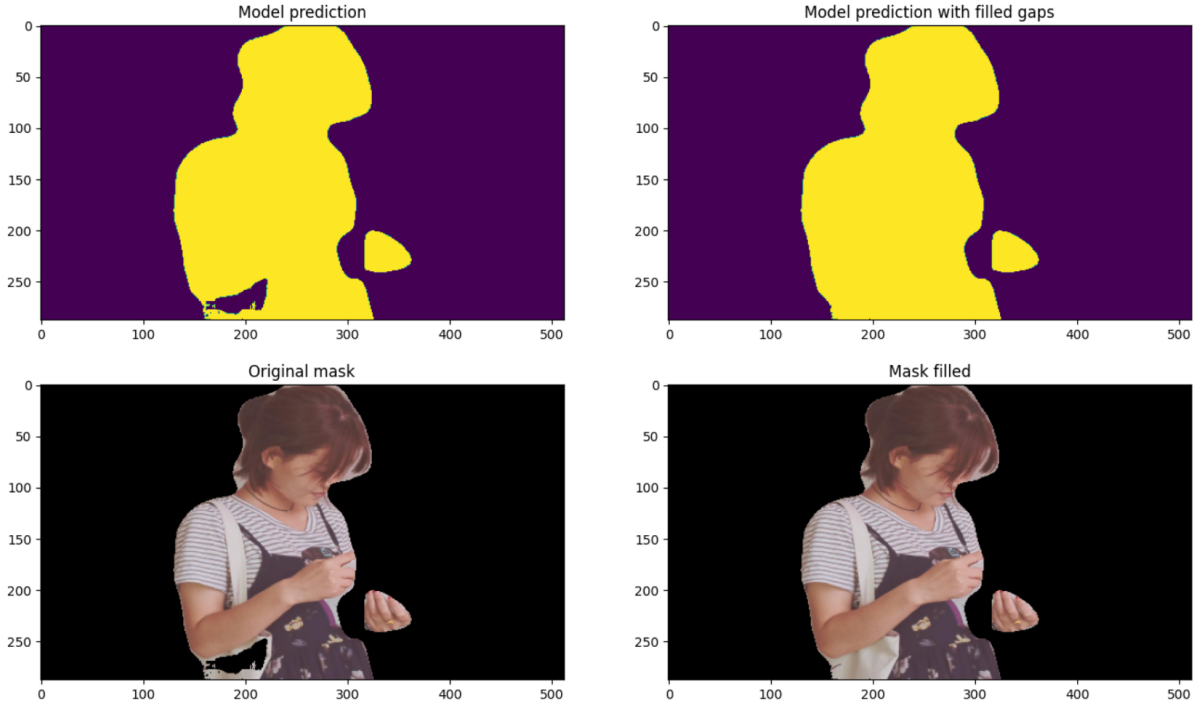
Run	Value	Step	Relative
lightning_logs/EncoderDecoderWithCE	0.8813	33,760	2.593 hr
lightning_logs/EncoderDecoderWithSIoU	0.882	45,863	3.711 hr
lightning_logs/EncoderDecoderWithSIoU+augment	0.8738	45,863	3.801 hr
lightning_logs/EncoderDecoderWithSIoU+normalization	0.8766	41,404	3.347 hr
lightning_logs/UNetOurImplementation	0.8196	45,863	5.571 hr
lightning_logs/UNetTorchHub	0.811	26,711	1.913 hr

An illustration of semantic segmentation model's performance, simple scenario.

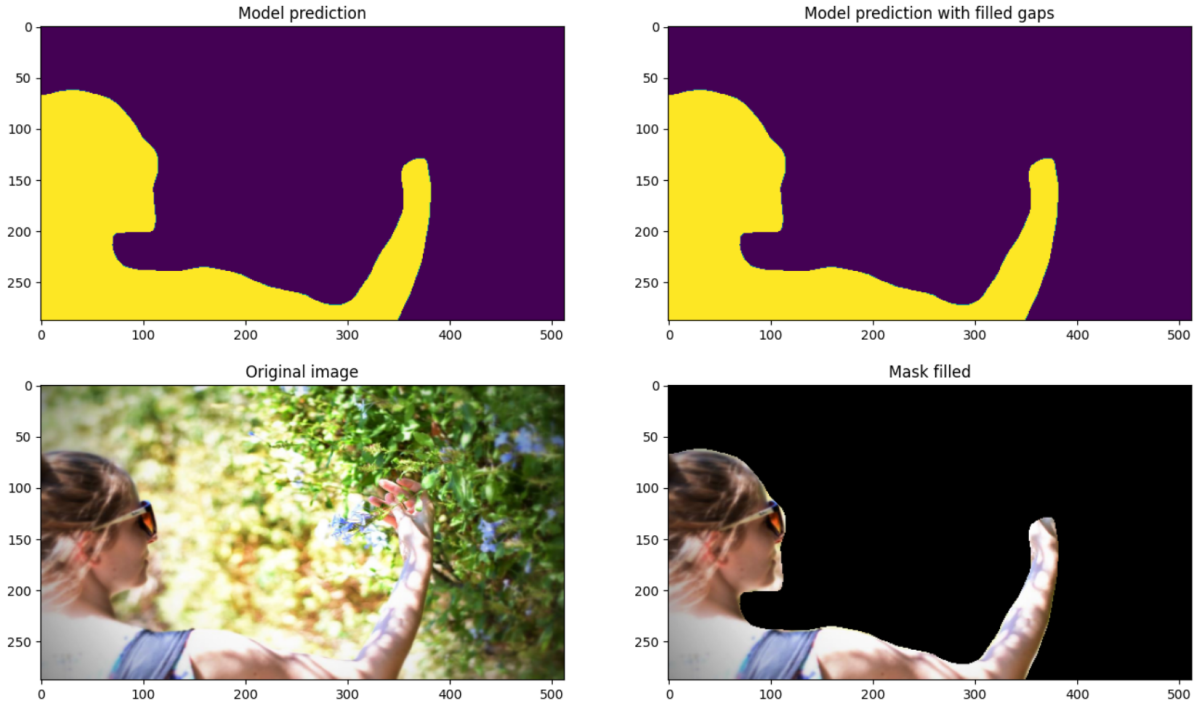
An illustration of semantic segmentation model’s performance, hard scenario.



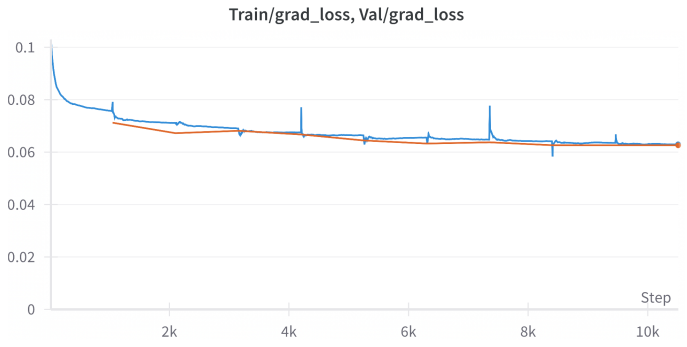
An illustration of the “gap inside the mask” problem.

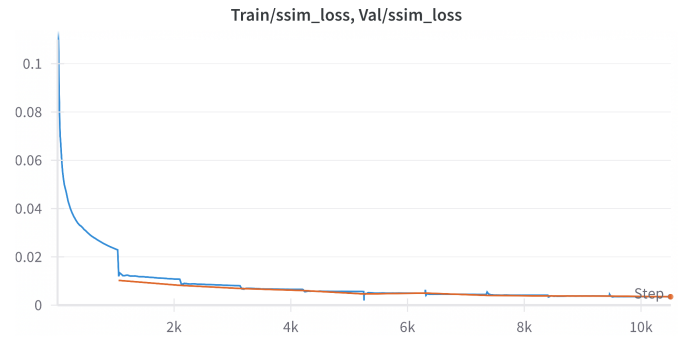


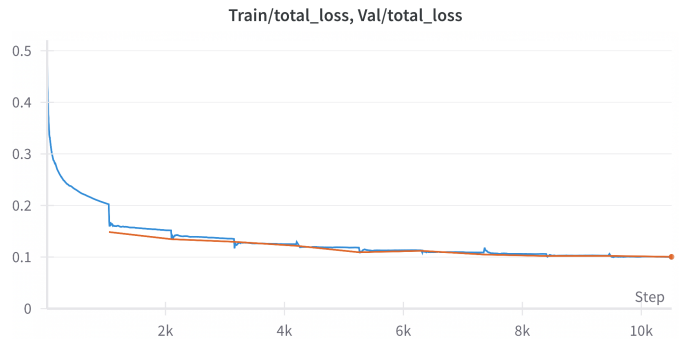
An illustration of the hand problem.



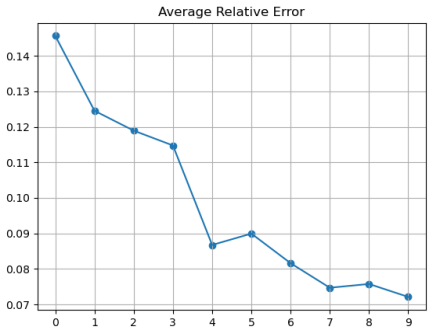
Loss functions for depth map estimation model training.

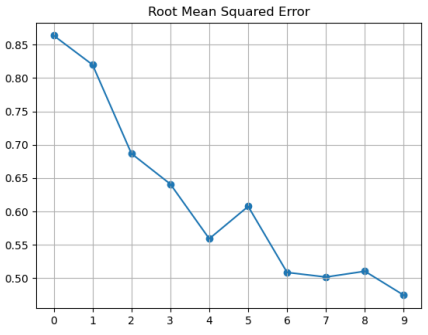


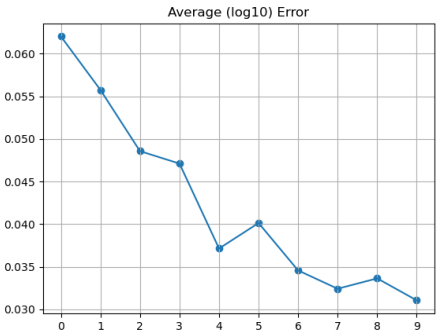




The depth map estimation metrics for epochs range from 0 to 10.







Three blurring methods proposed in our work on a certain example.

The original image

Baseline blurring algorithm

Increased number of blurring kernels

Blurring the person segmentation mask

References

1. Olaf Ronneberger, Philipp Fischer, Thomas Brox, 2015, “U-Net: Convolutional Networks for Biomedical Image Segmentation”
2. Md Atiqur Rahman and Yang Wang, 2016, “Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation”
3. Mingxing Tan, Quoc V. Le, 2019, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”
4. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, 2018, “MobileNetV2: Inverted Residuals and Linear Bottlenecks”
5. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015, “Deep Residual Learning for Image Recognition”
6. Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, Alan L. Yuille, 2016, “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”
7. Diganta Misra, 2019, “Mish: A Self Regularized Non-Monotonic Activation Function”
8. Prajit Ramachandran, Barret Zoph, Quoc V. Le, 2017, “Swish: a Self-Gated Activation Function”
9. DeepSystems, Person Segmentation Dataset, 2018, https://hackernoon.com/releasing-supervisely-person-dataset-for-teaching-machines-to-segment-humans-1f1fc1f28469
10. Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, Saining Xie, 2022, “A ConvNet for the 2020s”
11. Hu, J., Shen, L., and Sun, G., 2018, “Squeeze-and-excitation networks”
12. High Quality Monocular Depth Estimation via Transfer Learning - https://paperswithcode.com/paper/high-quality-monocular-depth-estimation-via
13. NYU Depth Dataset V2 - https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html
14. Monocular Depth Estimation with Transfer Learning pretrained MobileNetV2 - https://github.com/alinstein/Depth_estimation