NAZARBAYEV UNIVERSITY

School of Engineering and Digital Sciences

ELCE 457 Final Project

Interactive image segmentation using convolutional neural networks trained on a single image

Dair Ungarbayev

Abstract

This project investigates the effectiveness of applying convolutional neural networks to the task of interactive image segmentation. Importantly, the networks are trained on a single image instead of a large dataset. The central idea of the method is to get user input on a downscaled image and reconstruct the image of original resolution given the user input. The convolutional neural network is used to learn the mapping from downscaled images to the original image. The results show that the method has some potential, though the method does not work universally well on different images.

Background and motivation

Background

Broadly speaking, image segmentation means partitioning an image into meaningful regions. Here, the "meaningful" refers to application specificity. For instance, a common application is in the arts and graphics, where one may want to segment out some object in one scene and place it into another scene. Another important application area is in medicine, where there is a need for segmenting an image such as a CT scan in order to precisely locate a tumor and measure its dimensions. It would be challenging and error-prone to manually process each image, thus there is a need in computational methods of image segmentation.

Another way of approaching the problem of image segmentation is pixel classification. In other words, the goal is to assign a class to each pixel in an image and different methods are designed depending on the number of classes. For example, in background-foreground segmentation, each pixel is either classified as background or foreground. A more challenging is the task of semantic segmentation, where each pixel is assigned a semantic label, such as 'road', 'traffic sign' or 'pedestrian' (in the area of autonomous vehicles) [1, 2].

While a lot of effort is put into developing fully automated segmentation algorithms, it is also possible to have the human assist the computer in the task. Thus, interactive image segmentation refers to methods of image segmentation where some prior information is provided to the algorithm by means of user input.

Project objective

Initially, the proposed project sought to investigate the relationship between user input and the complexity as well as the effectiveness of a segmentation algorithm. Then, it was planned to design an adaptive interactive image segmentation system that could accept various user inputs. The main potential approach to this goal was to combine existing methods into a single system, which does not require much original work. It is more of a software engineering task rather than a computer vision problem. Thus, the revised goal was to develop an original interactive image segmentation method. The scope of the project was limited to foreground-background segmentation.

In order to design an interactive image segmentation method, the following requirement is set for the method: the user must have some margin of error in his inputs. In other words, the algorithm must produce the same output even if the user's input has errors. Otherwise, the user is too involved in the process and needs to put in considerable effort.

The main idea for an interactive image segmentation algorithm in this project that satisfies the requirement above is to ask the user to manually segment a downscaled version of the image. Since, the number of pixels is small, the user does not have to put in as much effort. Then, the downscaled and segmented image can be resized to its original size. The resulting image is then segmented. In other words, one needs to be able to map a downscaled image into its corresponding original image with segmentation applied.

Thus, the question is: how to map a downscaled image to the image of original size? The method that seems compelling is to use deep learning models, specifically, convolutional neural networks. Convolutional neural networks were already applied to the task of interactive image segmentation, for example in [3].

Typically, convolutional neural networks require large datasets for training (as is done in [3]). However, recent research efforts demonstrated the effectiveness of fitting a network to a single image, thus avoiding the expensive process of dataset creation. For example, Deep Image Prior [4] and SinGAN [5] successfully applied network fitted to a single image to such tasks as random sample generation, super-resolution, denoising, inpainting and harmonization. Thus, one may wonder if networks fitted to a single image can be applied to interactive image segmentation.

Therefore, **the objective of the project**:

To investigate the effectiveness of single image fitted convolutional neural network in the task of interactive image segmentation.

Algorithm description

Overview

Overall, the method can be briefly summarized as follows:

1. Generate training and validation data from the input image by creating random masks, that occlude the image except for some region.
2. Train a convolutional network on the generated data until the loss converges.
3. Get the user input on the image.
4. Create network input by applying user input on downscaled image.
5. Generate image of original resolution using the network.

The following subsections will describe the process in detail.

Mask generation

The main idea behind the training data is to occlude all image except for a certain portion. Therefore, masks are generated randomly via **`get_mask`** function. Fig. 1 displays several generate mask samples. The `get_mask` generates masks of two shapes: rectangles and circles. This can be extended to generate other shapes such as other polygons or arbitrary shapes in order to improve the diversity of training data. The masks are generated using the `randi` function, sampling integer values uniformly from a specified range. For rectangular mask, the upper-left corner coordinates as well as the height and width are sampled. For circular mask, the center coordinates and the radius are sampled. The range of values in `randi` is set so that very small shape dimensions are avoided. If most of the image is occluded, the network will simply learn to produce completely black images.

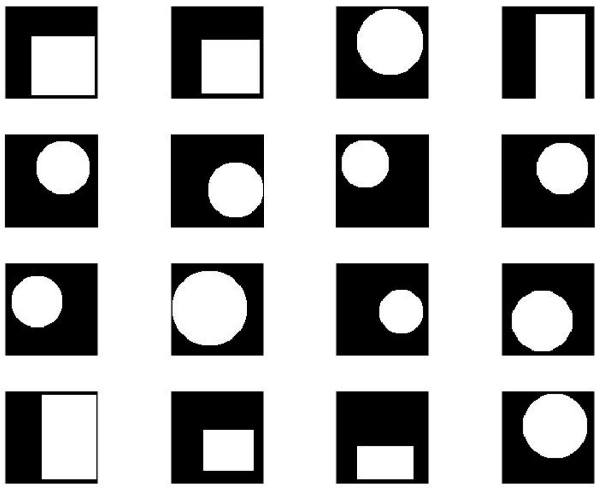


Figure 1. Several samples of the generated masks.

Data generation

Data is generated in the **`get_trainval_data`** function. The correct output training data is generated by applying the sampled masks to the image. These images are what the network must learn to produce from the input. Several samples are displayed in Fig. 2.

```
y_img = img.*uint8(mask);
```

Figure 2. Samples of training images that the network learns to reproduce.

The network inputs are generated by downscaling `y_img`, converting it to grayscale for convenience, and concatenating `y_img` to itself `inp_chan_num` times, thus generating a “string” (technically, a volume) of images:

```
x_full = x_img;
for l=1:inp_chan_num-1
    x_full = cat(3,x_full,x_img);
end
```

This is done so that the network has more opportunity to learn. Next, noise is added to the network input. The noise is sampled from normal distribution.

```
x_noise = normrnd(mu,sigma,[im_size(1)*scale im_size(2)*scale inp_chan_num]);
x_full = double(x_full)+x_noise;
```

This approach was adapted from [4] so that the network is robust to small perturbations in the input. Fig. 3 displays several channels of the `x_full` volume.

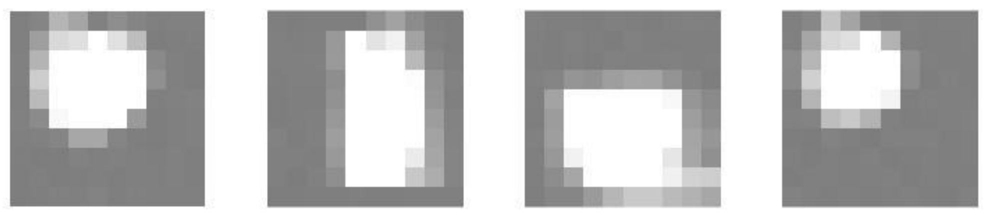


Figure 3. Several sample channels of the `x_full` volume.

Network

The network architecture used is as follows:

```
layers = [
  imageInputLayer(size(x_train(:,:,:,1)),'Normalization','none')
  transposedConv2dLayer(3,256)
  reluLayer
  transposedConv2dLayer(3,128,'Stride',2,'Cropping',[1 1 1 1])
  reluLayer
  transposedConv2dLayer(3,64)
  reluLayer
  transposedConv2dLayer(3,32,'Stride',2,'Cropping',[1 1 1 1])
  reluLayer
  transposedConv2dLayer(3,8)
  reluLayer
  transposedConv2dLayer(3,3,'Stride',2,'Cropping',[2 1 1 2])
  sigmoidLayer
  regressionLayer
];
```

Up-sampling is performed using transposed convolution layers. A stride of 2 helps to get images of larger dimensions much quicker. The final transposed convolution output is normalized to a [0, 1] range via a sigmoid function. The `regressionLayer` is used for training purposes. It computes a mean-square-error loss. Since we look to reproduce the output image, the goal of training is to minimize this loss. The specific architecture can be altered for experimentation, though input and output dimensions must be taken into account. The networks in this project are trained to reproduce 100x100 images from 10x10 downscaled images. This is done due to considerable computational cost that is incurred as image size becomes larger. Moreover, with larger images the network must be deeper to have the capacity to capture the image statistics.

Transposed convolution

In order to produce an output of higher dimensions from an input of lower dimensions, one may perform simple interpolation on an input image and perform regular convolution with padding to preserve the new dimensions. Another way is to use transposed convolutions. To understand the transposed convolution operation, it is first necessary to discuss matrix multiplication equivalent of a convolution. In order to perform convolution, the filter kernel must be represented as a sparse matrix as below:

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 \end{bmatrix}$$

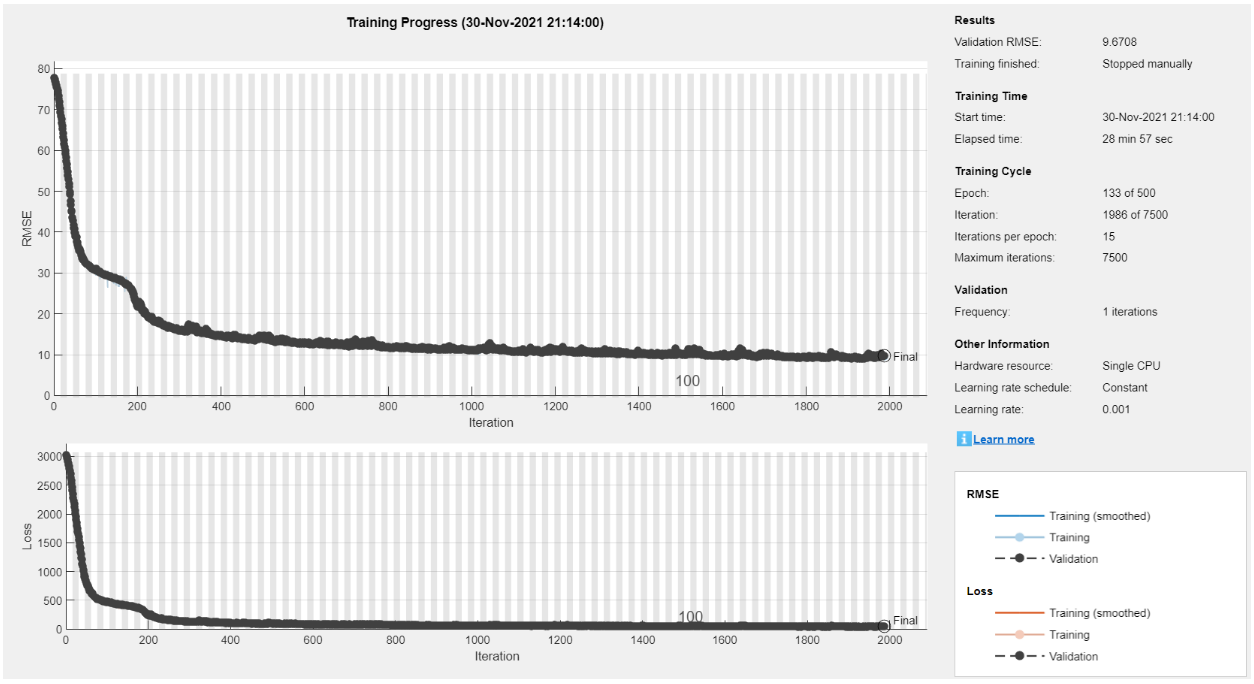
where $w_{m,n}$ is the element of a kernel in row m and column n. The input should be flattened to produce a vector and thus a convolution can be represented as follows:

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

where **y** is the flattened output, **W** is the sparse matrix form of the kernel and **x** is the input vector. Clearly, if one would expand this matrix-vector multiplication, the results would be the same as with a regular convolution. Thus, one could get produce an image of higher dimensionality **x** by calculating $\mathbf{W}^{\mathbf{T}}\mathbf{y}$.

Network training

The **`train_network`** script is used to train the network on an image. Both inputs x and targets y are divided by 255 to scale the pixel values to [0, 1] range. The network training options specify the number of epochs to 500 and the training is stopped manually when the loss function appears to converge. Fig. 4 displays the training progress of a network trained on the “cat” image. Fig. 5 displays the outputs of the network with validation images. It can be seen that the network learns to roughly reproduce the images.



Results

Validation RMSE:	9.6708
Training finished:	Stopped manually

Training Time

Start time:	30-Nov-2021 21:14:00
Elapsed time:	28 min 57 sec

Training Cycle

Epoch:	133 of 500
Iteration:	1986 of 7500
Iterations per epoch:	15
Maximum iterations:	7500

Validation

Frequency:	1 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

Learn more

RMSE
- Training (smoothed)
- Training
- Validation

Loss
- Training (smoothed)
- Training
- Validation

Figure 4. Training progress of the network on "cat" image.

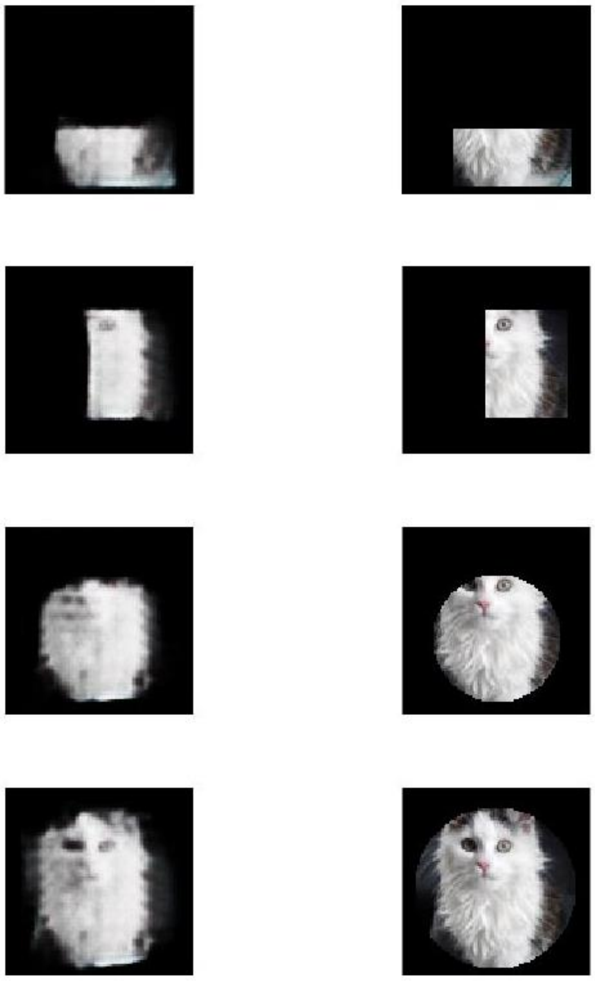


Figure 5. The network outputs (left column) on validation images (right column).

Applying the trained network

The user input in **`segment_image`** script is taken by overlaying downscaled image over the original image. The `drawfreehand` function allows the user to draw the rough outline of the desired foreground region. Crucially, the user is not required to draw an accurate outline. This can be seen in Fig. 6. Since the mask is applied to the downscaled image, the user has a margin of error when outlining the object. The image in the original resolution is displayed for the convenience of the user, while the downscaled image is overlayed to see how the network input will be generated.

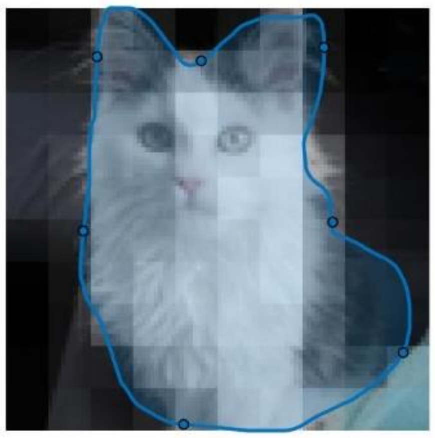


Figure 6. Getting the user input.

The input to the network is generated by applying the user specified mask to the downscaled image:

```
im_input = im_small.*imresize(uint8(mask_fg),scale);
```

The standard procedure of converting to grayscale, concatenating, and adding noise is performed as with training data generation. Finally, prior to feeding the input to the network, the pixel values are scaled to the [0, 1] range. Fig. 7 displays the resulting input to the network. The user could directly edit such a downscaled version of the image, and this would not take much work since the number of pixels is small. The result generated by the network is displayed in Fig. 8. It illustrates how the network has learned the mapping from downscaled images to images of the original size.

```
prediction = predict(net,x_full);
```

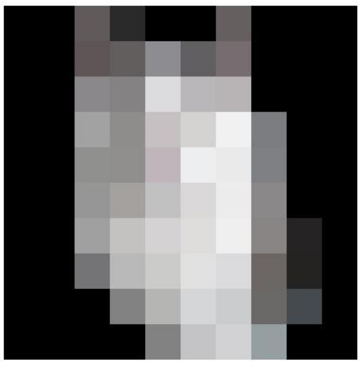


Figure 7. The input to the network with user specified mask applied.

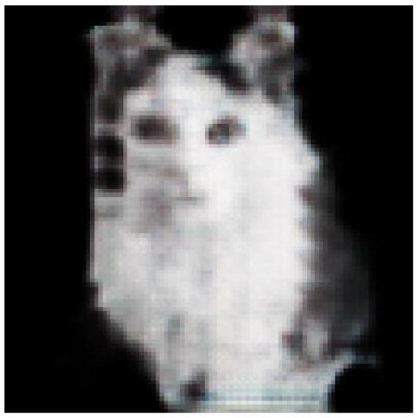


Figure 8. The generated segmented image having the original dimensions.

Results

“Cat” image

Fig. 9 displays the “cat” image. The network is trained on this image and the results are shown in Fig. 10. As can be seen on Fig. 10, given the varying and sloppy user input (leftmost column), the input to the network (center column) is roughly the same in each case. Thus, the produced reconstructions (rightmost column) are also roughly the same. This shows that the presented method has the potential to satisfy the requirement for interactive image segmentation.

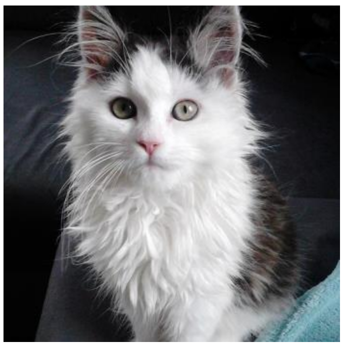


Figure 9. “cat” image.

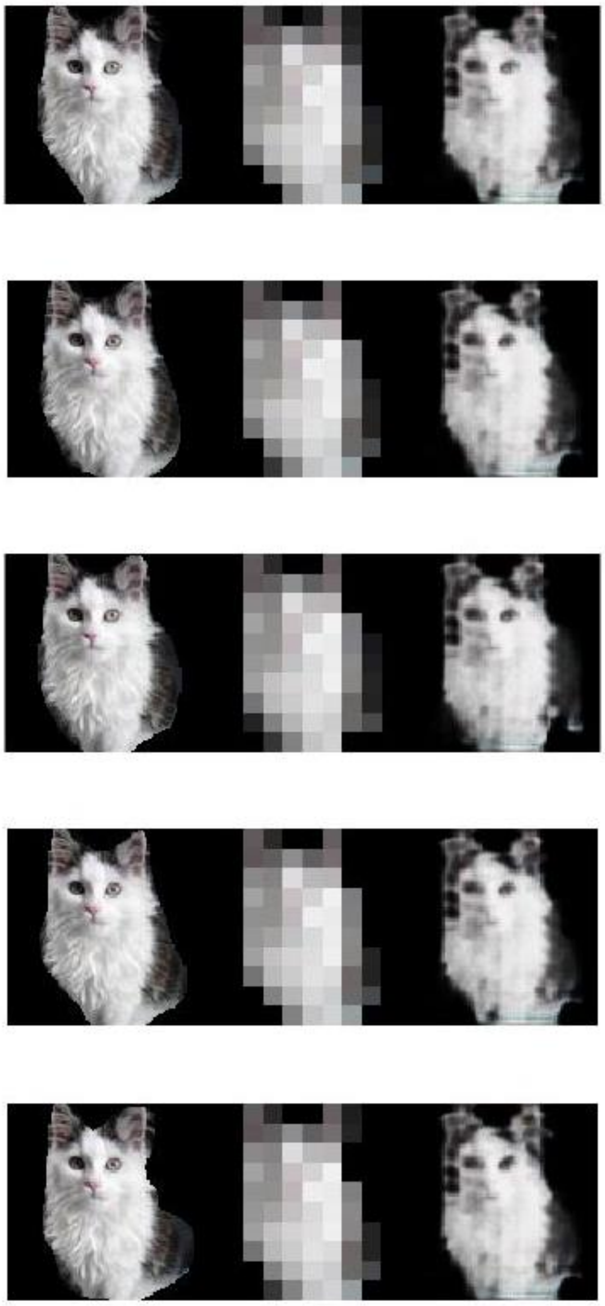


Figure 10. Results on the “cat” image. Leftmost column – original image with mask applied for reference. Center column – corresponding network input. Rightmost column – images produced by the network.

“Dog” image

The same process is applied to “dog” image (Fig. 11). The results (Figs. 12 and 13) show that the network in this case fails to accurately reconstruct the image. The reconstructions of segmented images in Fig. 13 especially look poor. This may be due to insufficient network depth as it clearly lacks the capacity to fit the image. The network clearly cannot reconstruct small details such as fur of the dog.

Figure 11. “dog” image.

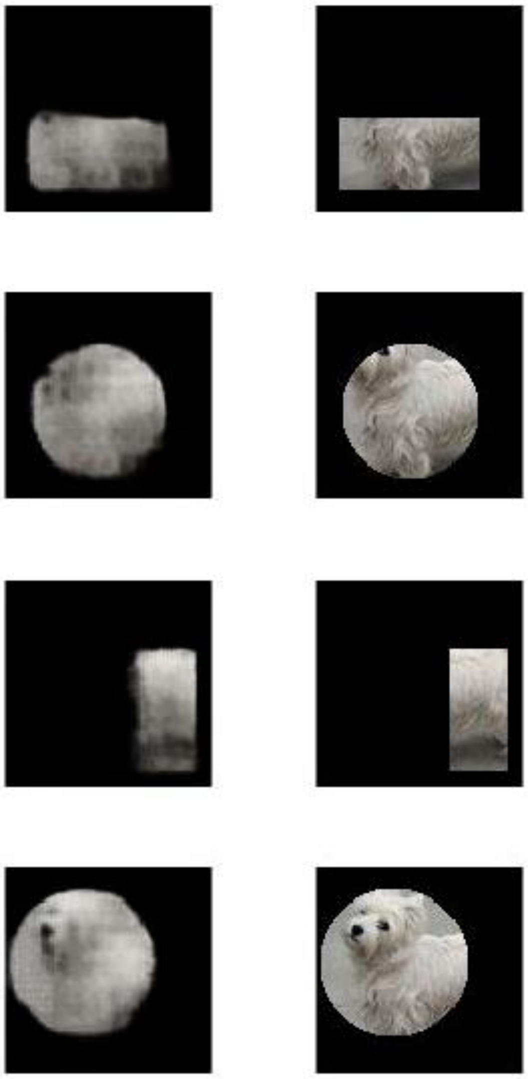


Figure 12. The trained network outputs (left column) on validation images (right column).

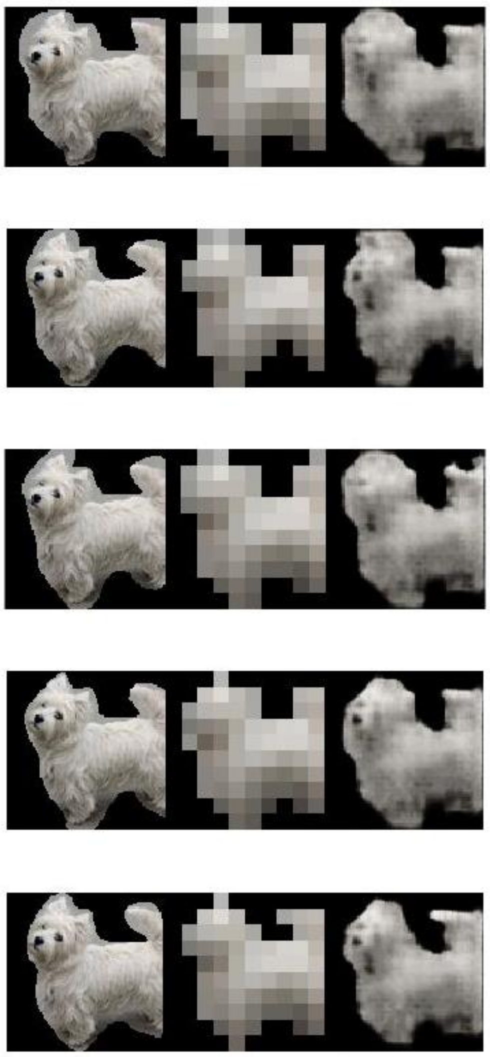


Figure 13. Results on the “dog” image. Leftmost column – original image with mask applied for reference. Center column – corresponding network input. Rightmost column – images produced by the network.

“Car” image

In case of the “car” image (Fig. 14), the network is larger by doubling the number of channels in each layer including the input layer. It was done to see if a larger network will result in better reconstruction. Fig. 15 shows that validation set data is reconstructed well. However, Fig. 16 shows that when user input is applied, the quality of reconstructions is worse.

Figure 14. “car” image.

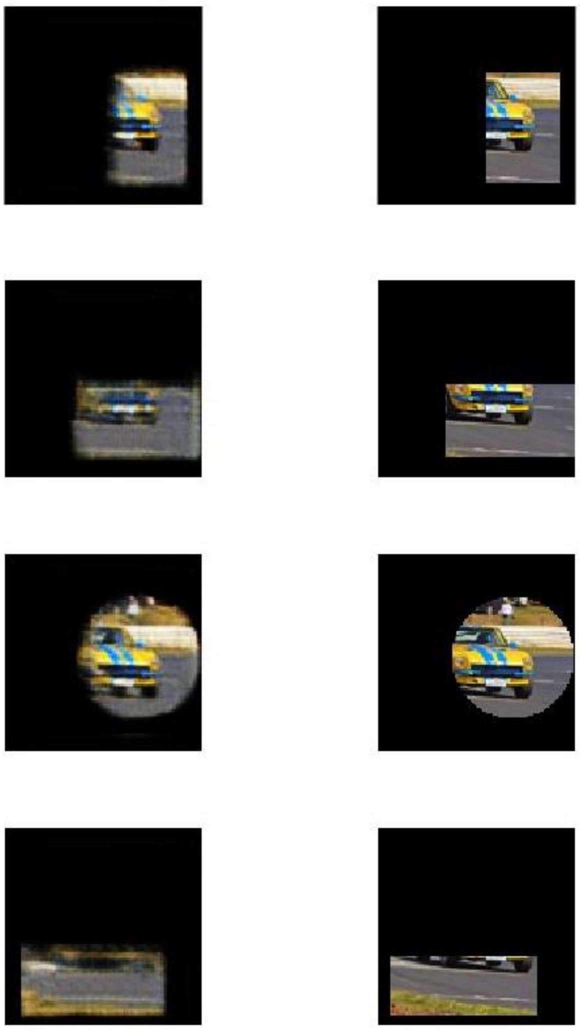


Figure 15. The trained network outputs (left column) on validation images (right column).

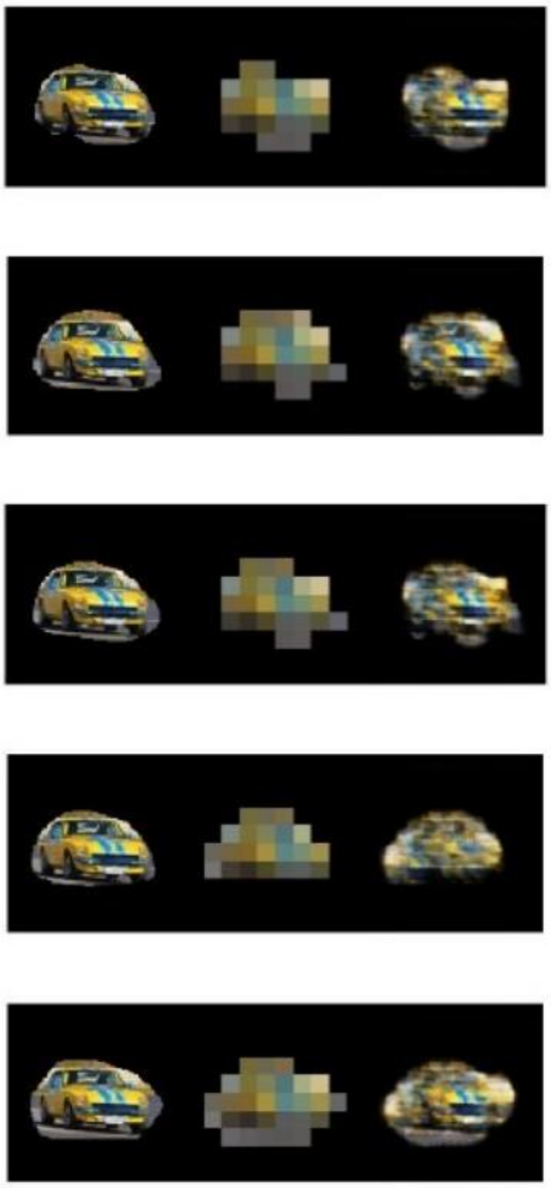


Figure 16. Results on the “car” image. Leftmost column – original image with mask applied for reference. Center column – corresponding network input. Rightmost column – images produced by the network.

Link to trained networks:

https://drive.google.com/drive/folders/136U7W0Z_Pq-0e9xJe46OP16Qc-KPT_kR?usp=sharing

Conclusion and future work

Applicability of a convolutional neural network (fitted to a single image) to the task of interactive image segmentation was explored. The presented interactive segmentation method is clearly still at the concept level and requires considerable work in order to improve the results. However, given the general trend of deep learning models outperforming traditional algorithms, it might be a method with some potential. Perhaps, a similar approach to image segmentation will find its use when the hardware is powerful enough to learn very deep networks much faster. However, clearly the traditional segmentation methods such as graph cuts will result in better segmentation with much smaller computational cost. Thus, one may conclude that convolutional neural networks fitted to a single image in a way that is done in this project is not suitable for interactive image segmentation.

It might be the case that the approach of applying user input on downscaled images and reconstructing from it is inadequate for the task of interactive image segmentation. Therefore, the future work on this topic might involve reconsideration of the approach.

One simple improvement upon this method is to extend it to work on images of varying resolutions. This would also require dynamic creation of a neural network to satisfy the given image dimensions as well as the depth to have the capacity to learn the image statistics. However, larger images clearly would require considerable hardware power.

One potential area of future work is to explore different neural network architectures to test which ones are better suited for this task. For example, one can investigate the effectiveness of encoder-decoder models. Since a simply downscaled image is a poor representation of the image, encoder models can be used to learn a better representation, which will then be used to get user input.

Another worthwhile direction of future work is to implement this method on a large image dataset instead of fitting a network to a single image. Such an approach could potentially be helpful in getting a generic network that can accurately process any instance that is representative of the training dataset. The method presented in this project works on a single instance and thus does not generalize to other instance of the same image classes.

One could also explore annotated datasets such as COCO [6] or Pascal VOC [7]. In these datasets, each image has a corresponding segmented ground-truth image. It would be interesting to see whether such supervision yields better results.

References

[1] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[2] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481-2495, 2017.

[3] J. H. Liew, Y. Wei, W. Xiong, S. H. Ong and J. Feng, "Regional interactive image segmentation networks," in *2017 IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017.

[4] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Deep image prior," in *CVPR*, 2018.

[5] T. R. Shaham, T. Dekel, and T. Michaeli, "SinGAN: learning a generative model from a single natural image," in *ICCV*, 2019.

[6] T. Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft COCO: common objects in context,” in *European Conference on Computer Vision (ECCV)*, 2014.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL visual object classes (VOC) challenge,” in *International Journal of Computer Vision*, 2009.