Deep Image Matting

CV Project Final Evaluation
Team-30(Mission Impossible)
MNS Subramanyam(20161190)
Sowrya Deep Thayi (20161029)
Appari Lalith (20161038)

Introduction

- Image matting involves extracting foreground object from an image.
- It is a key technology in image editing and film production and effective natural image matting methods can greatly improve current professional workflows.
- Matting tasks usually produces a "matte" that can be used to separate foreground from the background in a given **image**.

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i \quad \alpha_i \in [0, 1].$$

Conventional Methods

- Previous algorithms have poor performance when an image has similar foreground and background colors or complicated textures.
- The main reasons are prior methods
  - Only use low-level features.
  - Lack high-level context.
- These approaches rely largely on color as the distinguishing feature (often along with the spatial position of the pixels), making them incredibly sensitive to situations where the foreground and background color distributions overlap which is a drawback.
- Even other deep learning methods are highly dependent on color-dependent propagation methods.

Approach to overcome these problems

- To overcome these problems a new approach which considers structural and semantic features more into account for matting.
- Neural networks is capable of capturing such high order features and applying them to compute improved matting results.

Deep learning approach

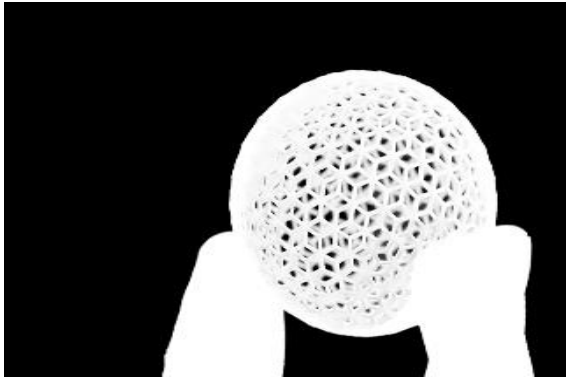
- This deep learning model has two parts.
- The first part is a deep convolutional encoder-decoder network that takes an image and the corresponding trimap as inputs and predict the alpha matte of the image.
- The second part is a small convolutional network that refines the alpha matte predictions of the first network to have more accurate alpha values and sharper edges.

Trimap

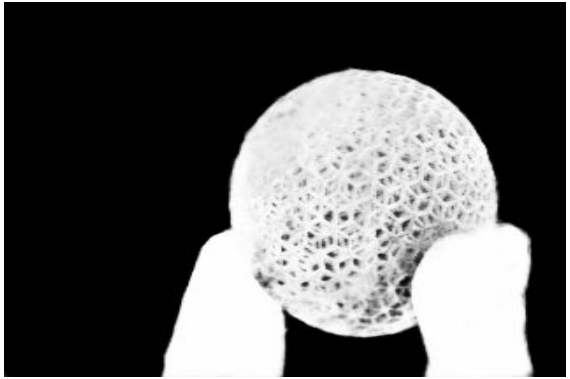
- A pre-segmented image consisting of three regions of foreground (what you want to cut out), background and unknown.
- Our algorithm tries to predict the pixel values in the unknown region.
- In the below image the unknown region is indicated in gray color.

Trimap

Original Alpha Matte



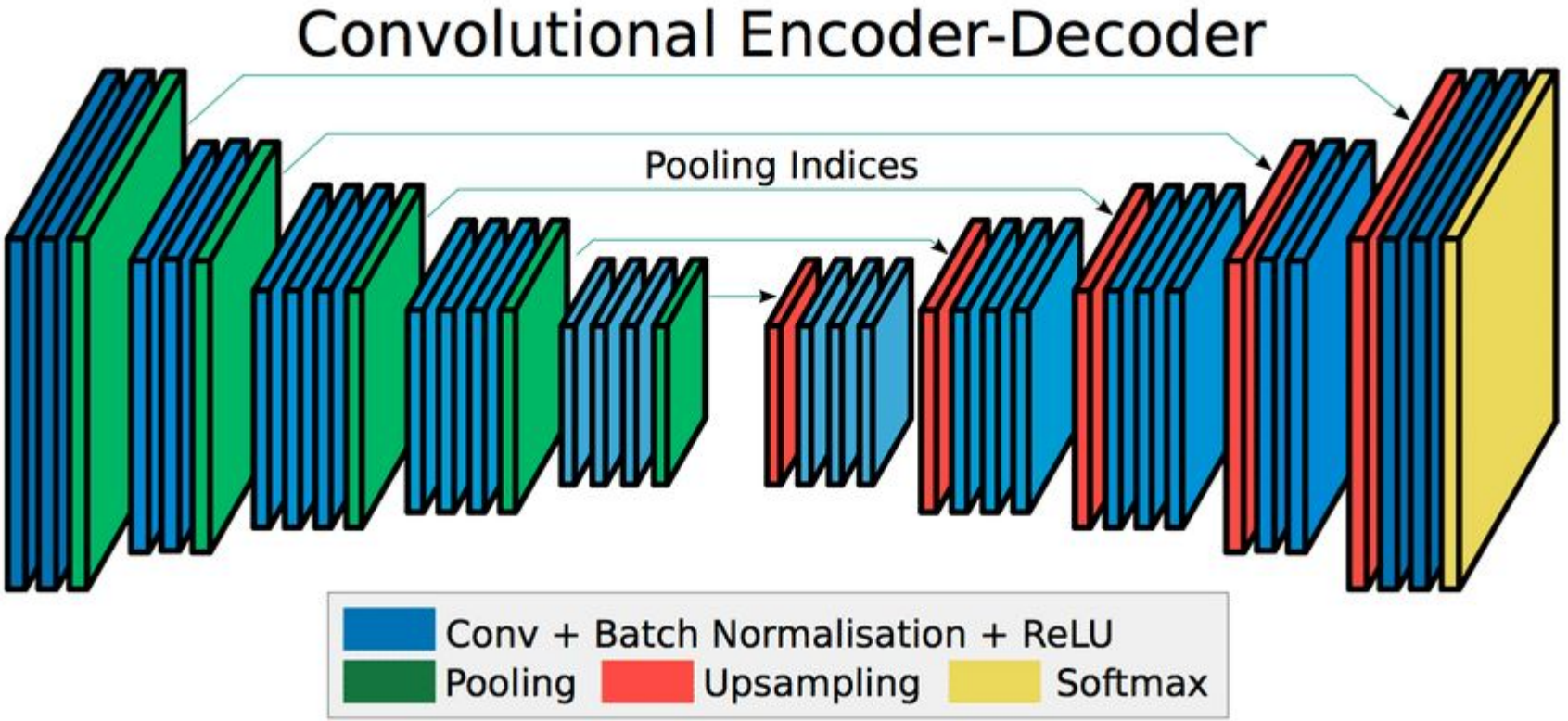
Predicted alpha matte



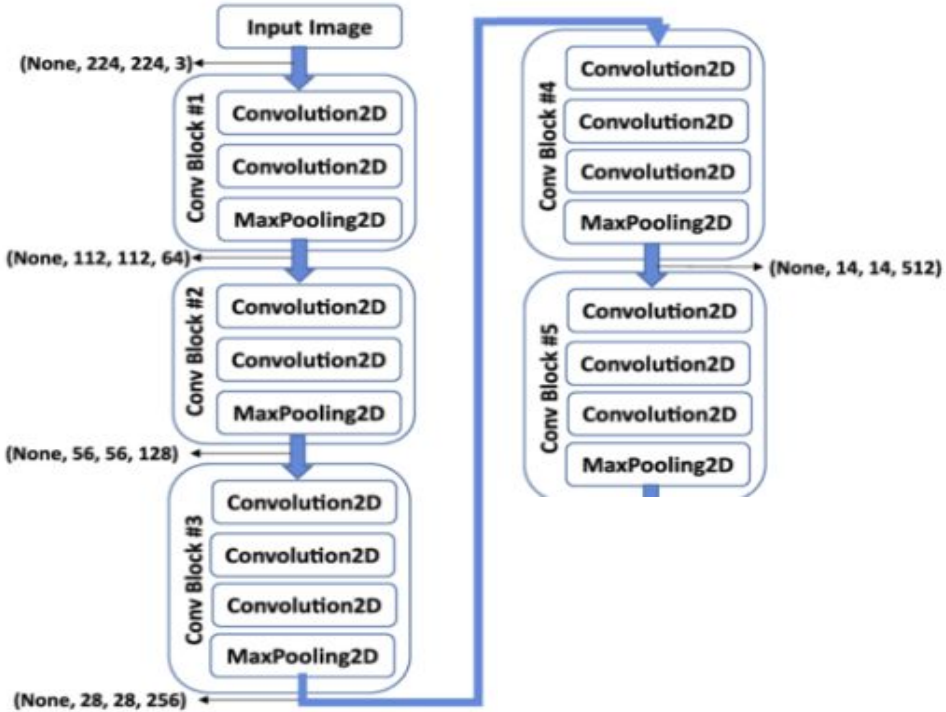
Encoder-decoder network

- Takes the input image (3-channel) and trimap and concatenates to form a 4-channel dimensional input.
- Encoder involves downsampling the image. (14 conv layers and 5-max pooling layers).
- Decoder involves upsampling the image. (6 conv layers and 5-unpooling layers).
- The decoder network has a small structure to reduce the parameters and amount of computation required.

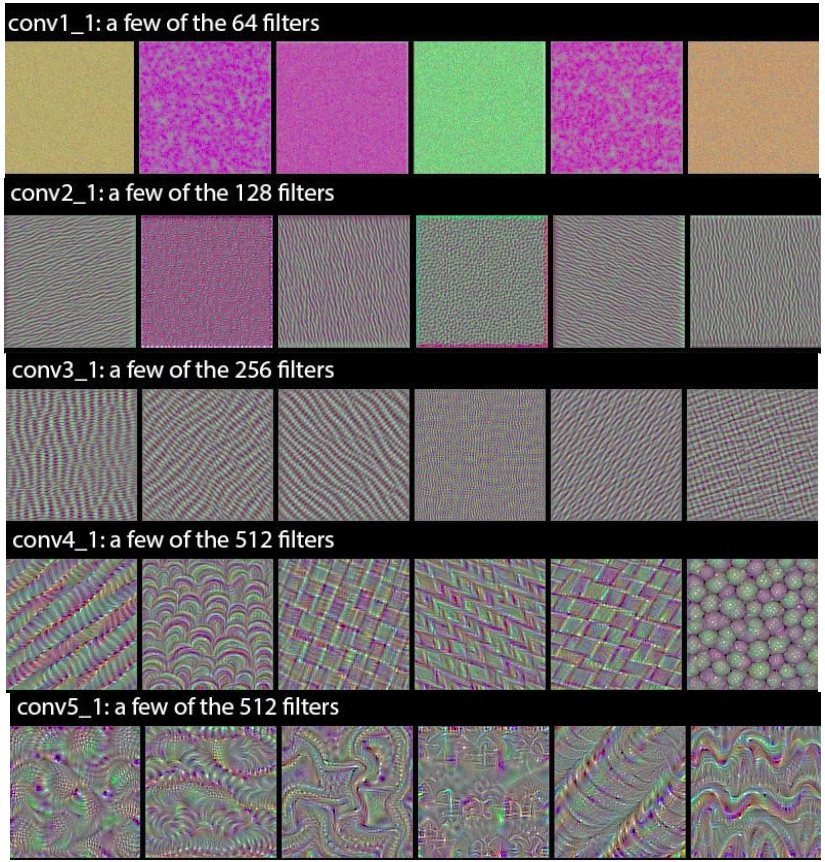
What is Encoder-Decoder network



Encoder Network



Filters in Encoder Network

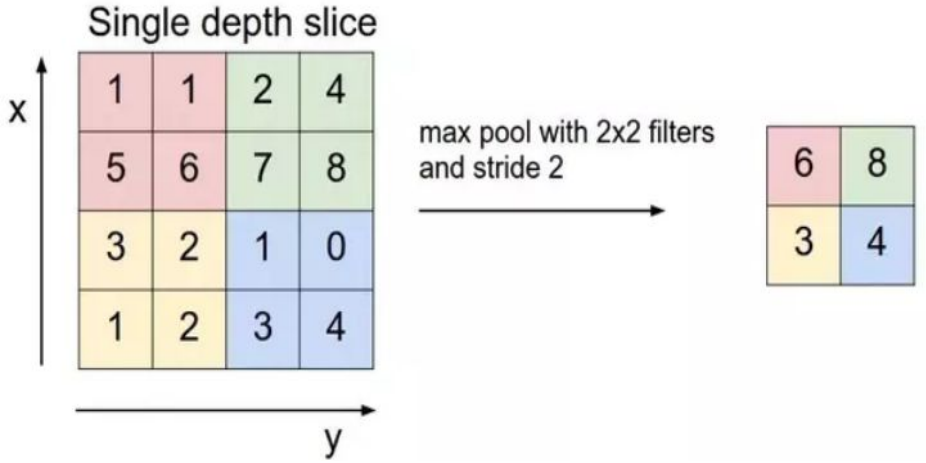


- A VGG16 network is used for the deep encoder network.
- These are some of the visualizations of the filters used in encoder network.
- The first layers basically just encode direction and color. These direction and color filters then get combined into basic grid and spot textures.
- The textures gradually get combined into increasingly complex patterns.
- One of the key observations is that a lot of these filters are identical but rotated at different angles, this helps in achieving rotational invariance

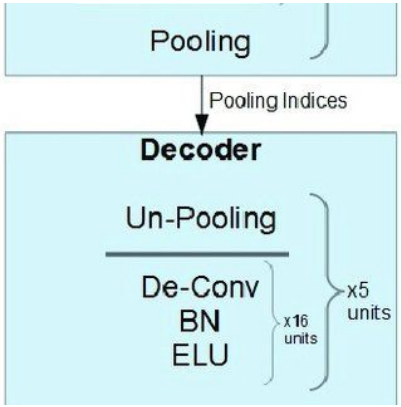
Max-Pooling Layers

- The objective is to down-sample an input representation (image) reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.
- This also helps to prevent overfitting and providing an abstracted version of the original image.
- This also reduces the computational cost, since now we have to deal with less number of parameters.
- Max pooling is done by applying a *max filter* to (usually) non-overlapping subregions of the initial representation.

Max Pooling - Pictorial Explanation



Decoder Network

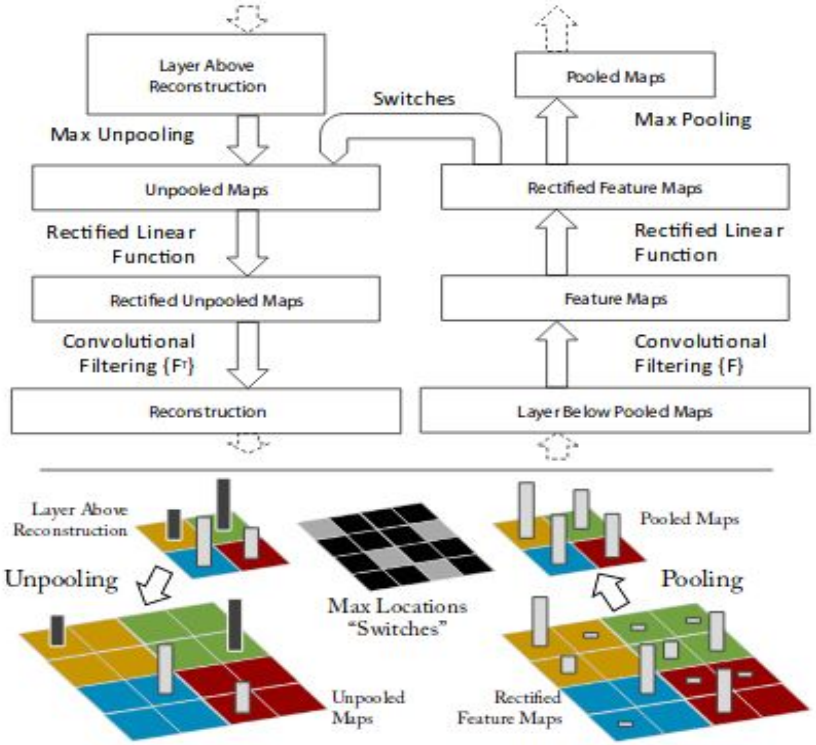


- The decoder CNN is used to map the low resolution features (extracted from pooling layers) to the final feature-maps of high resolution.
- The decoder model that is composed of upsampling, de-convolution, ReLU activation units. The decoder up-samples the encoder's convolution layer indices produced from pooling layers.
- Each decoder upsamples the activations generated by the corresponding encoder.

Un-Pooling Layers

- In the Convolution Network, the max pooling operation is non-invertible, however we can obtain an approximate inverse by recording the locations of the maxima within each pooling region.
- In the deconvnet, the unpooling operation then places the reconstructions from the layer above into appropriate locations, preserving the structure of the stimulus.

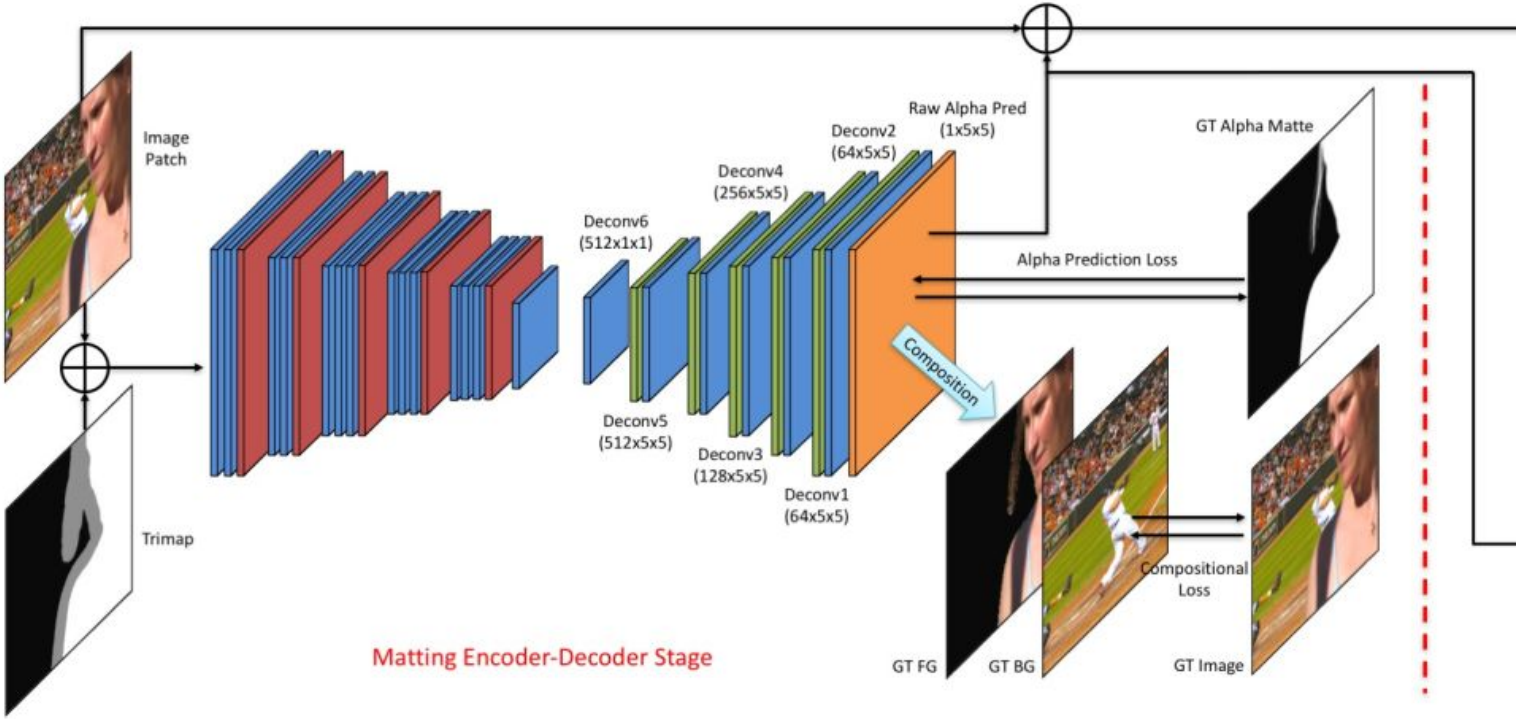
Max pooling and Un-Pooling Layers



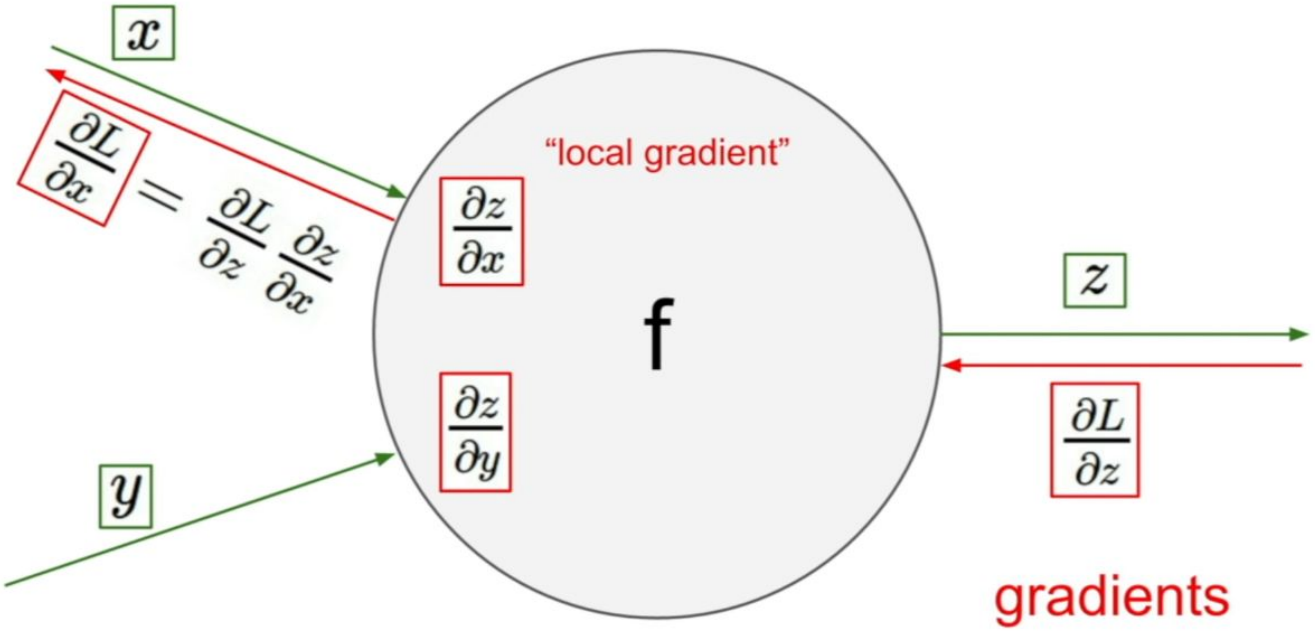
Convolutional Layers in the encoder decoder net

```
EDNet(
  (conv1_1): Conv2d(4, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv1_2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2_1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2_2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3_1): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3_2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3_3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4_1): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4_2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4_3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5_1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5_2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5_3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv6_1): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (deconv6_1): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (deconv5_1): Conv2d(512, 512, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (deconv4_1): Conv2d(512, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (deconv3_1): Conv2d(256, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (deconv2_1): Conv2d(128, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (deconv1_1): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (deconv1): Conv2d(64, 1, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
)
```

Matting Encoder-Decoder



Back propagation method



First stage procedure

- Encoder-decoder network takes the image concatenated with the trimap as input and penalized by the alpha prediction loss and a novel compositional loss.
- The network has two losses, the first loss is the alpha prediction loss which is the absolute squared difference between the ground truth value and predicted alpha values.

$$\mathcal{L}_{\alpha}^{i} = \sqrt{(\alpha_{p}^{i} - \alpha_{g}^{i})^{2} + \epsilon^{2}}, \quad \alpha_{p}^{i}, \alpha_{g}^{i} \in [0, 1].$$

- The second loss is the compositional loss which is squared difference between original RGB image and predicted RGB image using the new alpha loss.

$$\mathcal{L}_{c}^{i} = \sqrt{(c_{p}^{i} - c_{g}^{i})^{2} + \epsilon^{2}}$$

First stage procedure continued...

- The compositional loss constrains the network to follow compositional operation, leading to more accurate alpha predictions.
- The overall loss is a weighted combination of the two individual losses.

$$\mathcal{L}_{overall} = w_l \cdot \mathcal{L}_\alpha + (1 - w_l) \cdot \mathcal{L}_c$$

- While training our network we took w_l to be 0.5.
- We train this model with 431,000 train images obtained by combining 431 foreground images with 1000 background images.
- After training we tested our model with 1000 test images obtained by combining 20 foreground test images with 50 background images.

Model Analysis

- The training of the model took approximately 2-3 days for 30 epochs on 431K images.
- Next we evaluated our model on 1000 test images and got an MSE test error of 0.02
- We have saved the model at intermediate epochs to get results of intermediate images while training.
- Some of the results on test images are shown in the next slides.

Results

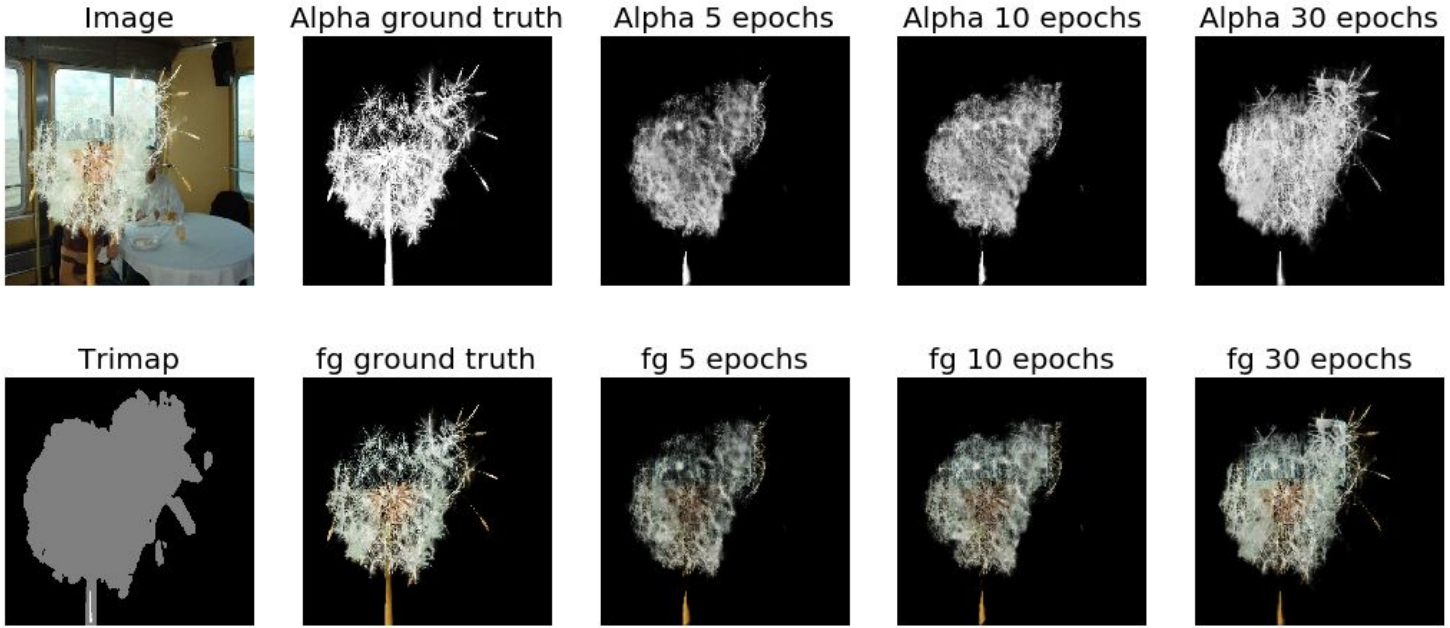


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

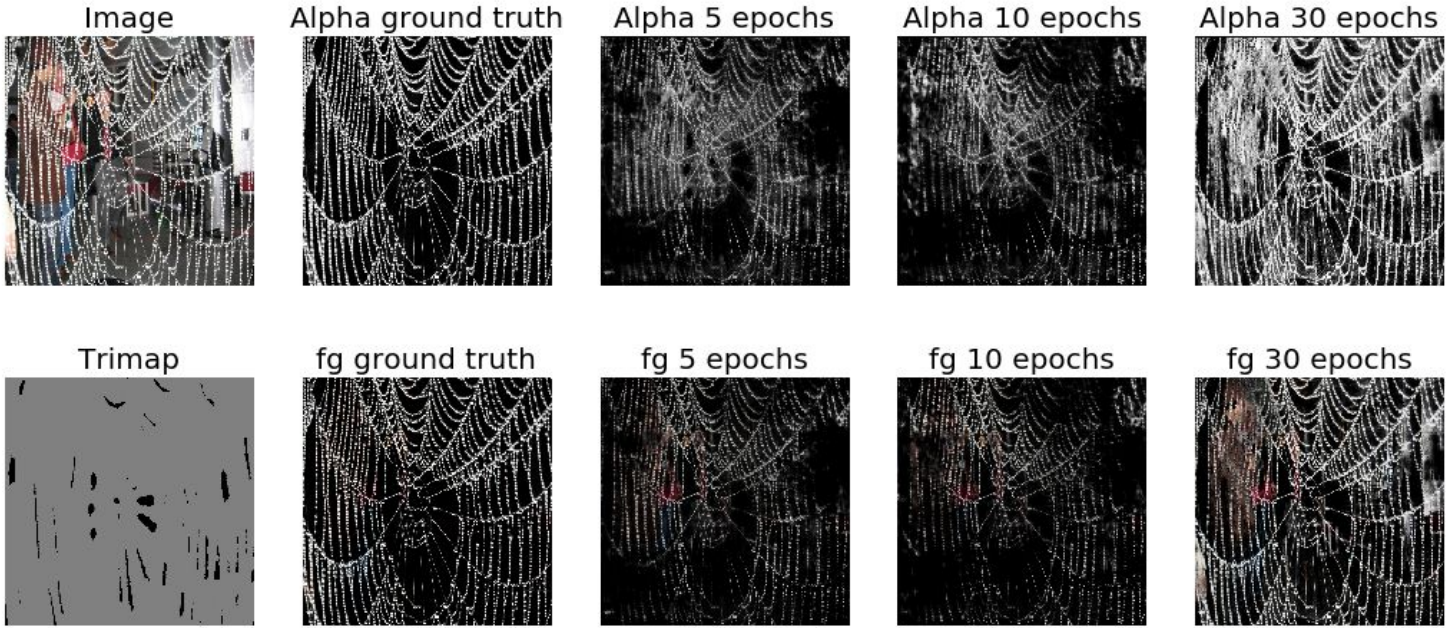


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

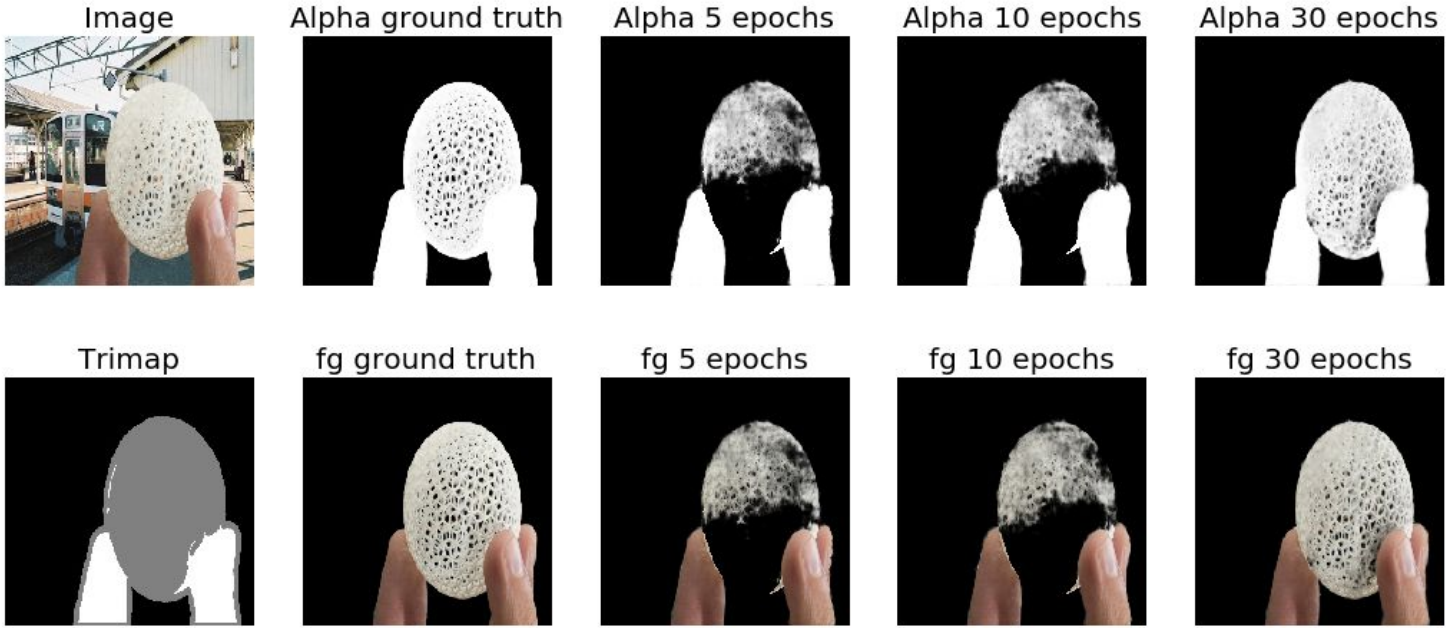


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

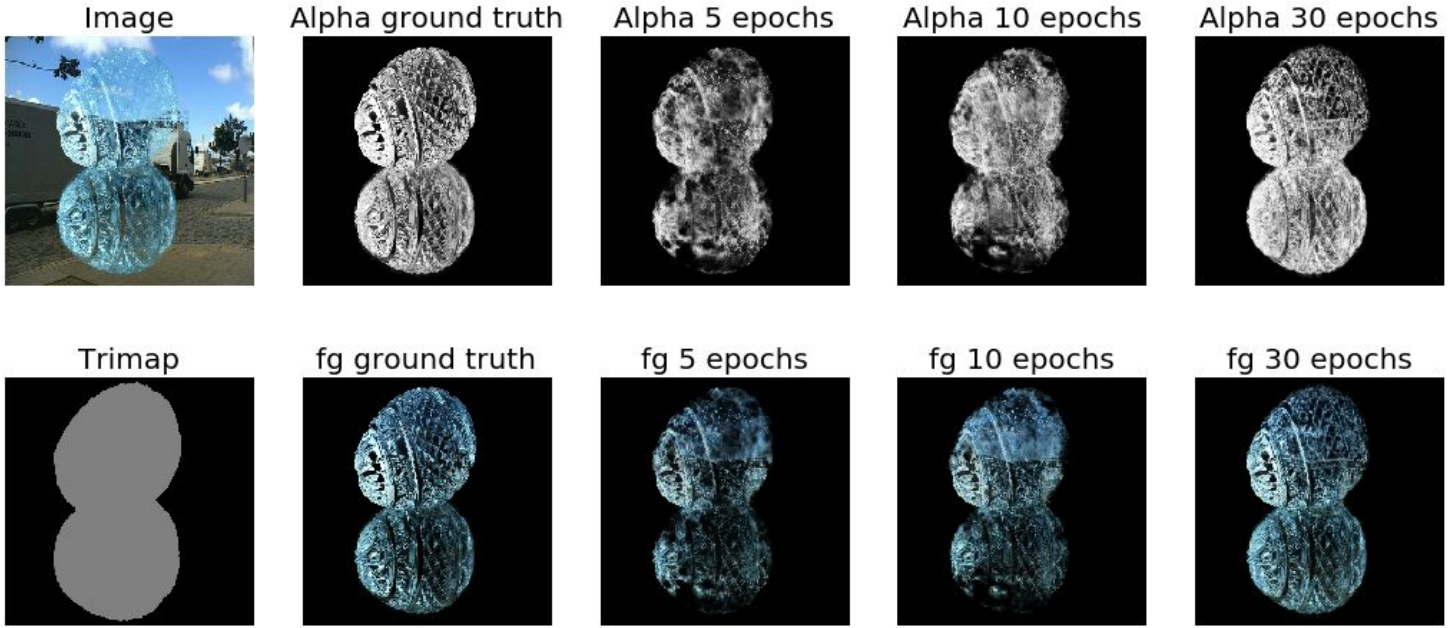


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

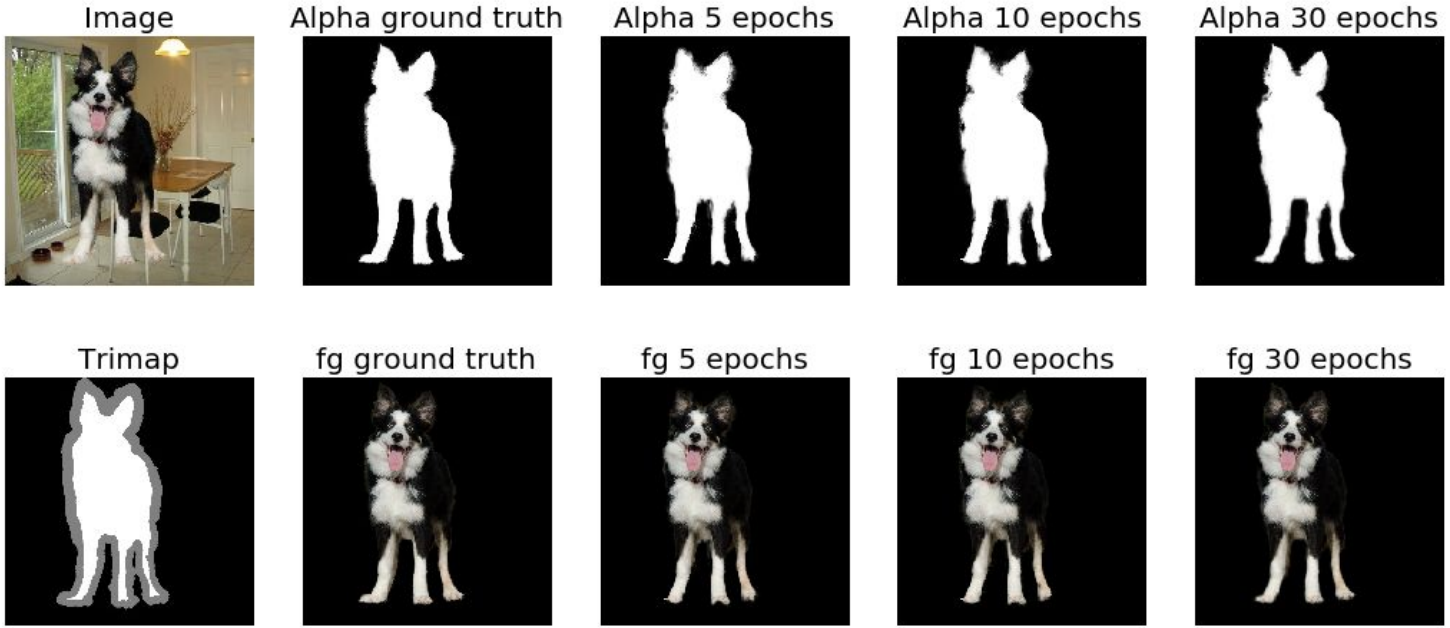


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

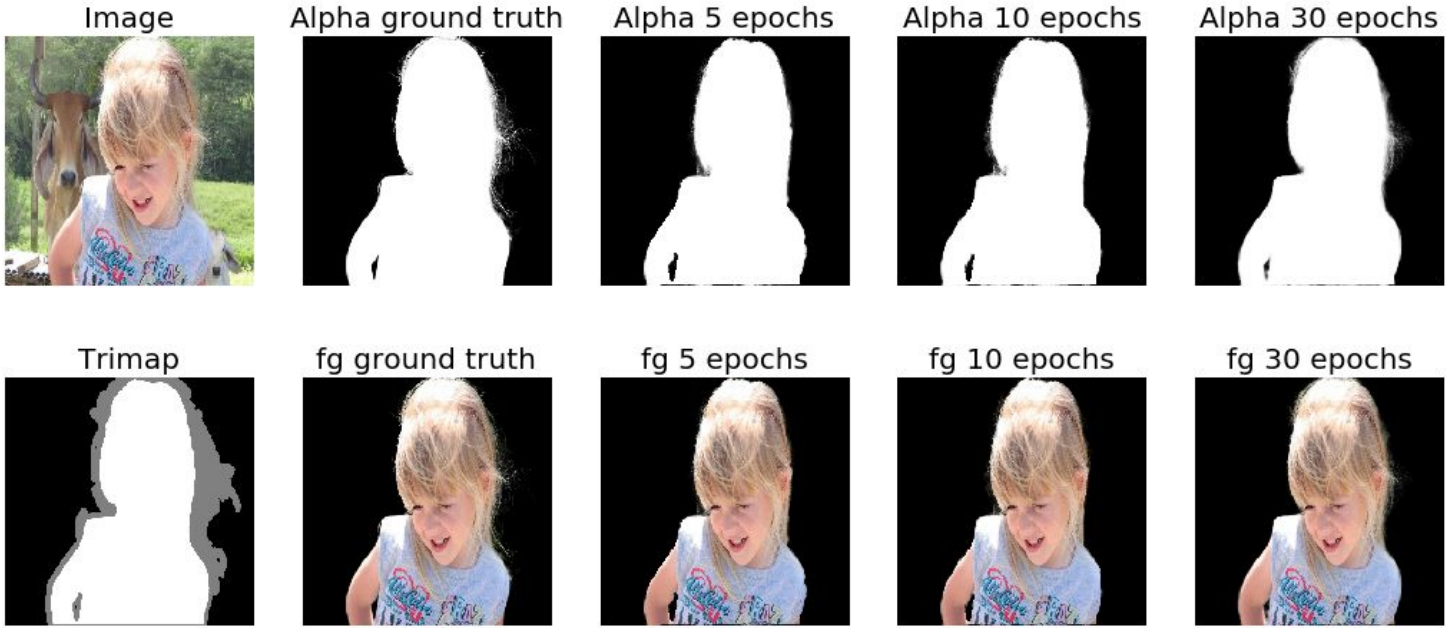


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

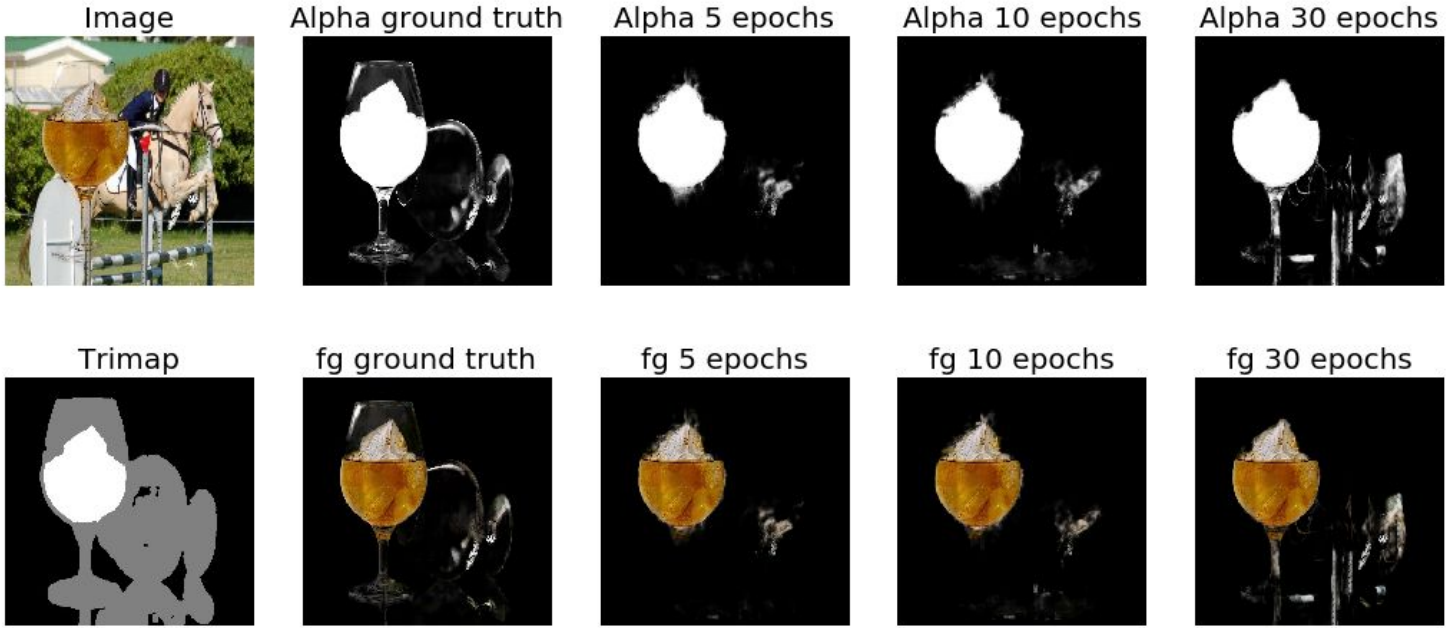


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

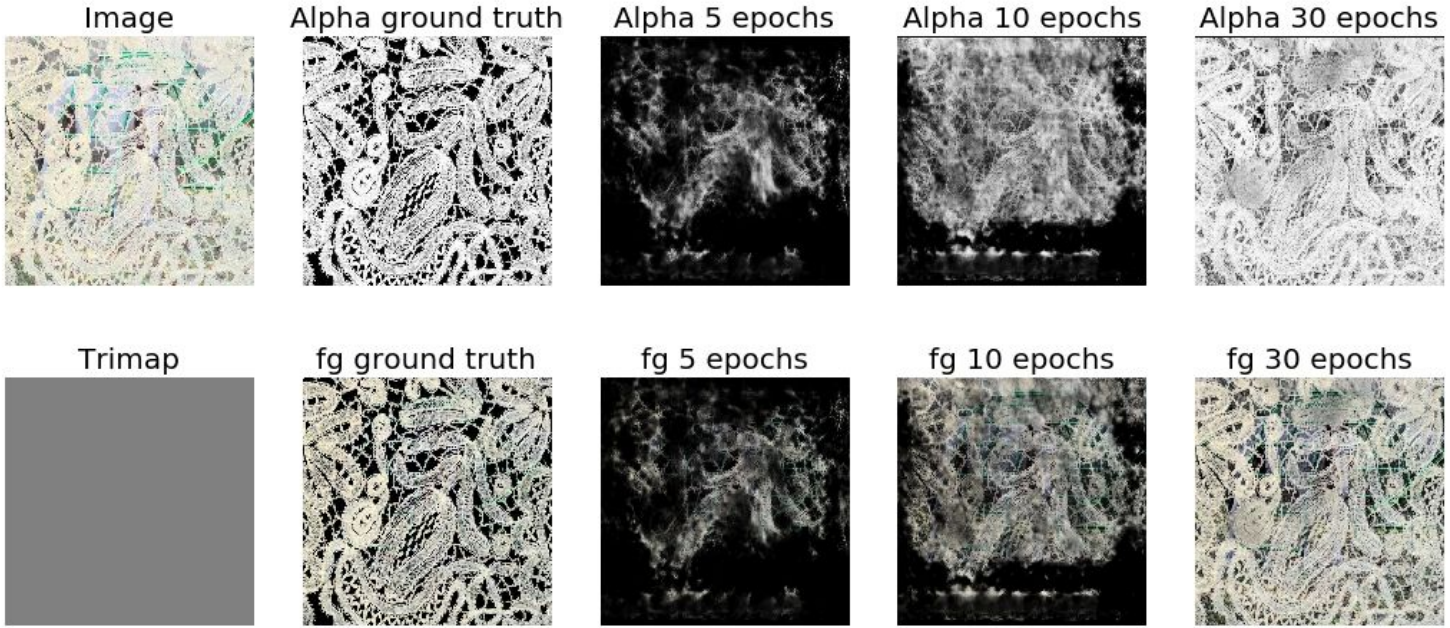


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

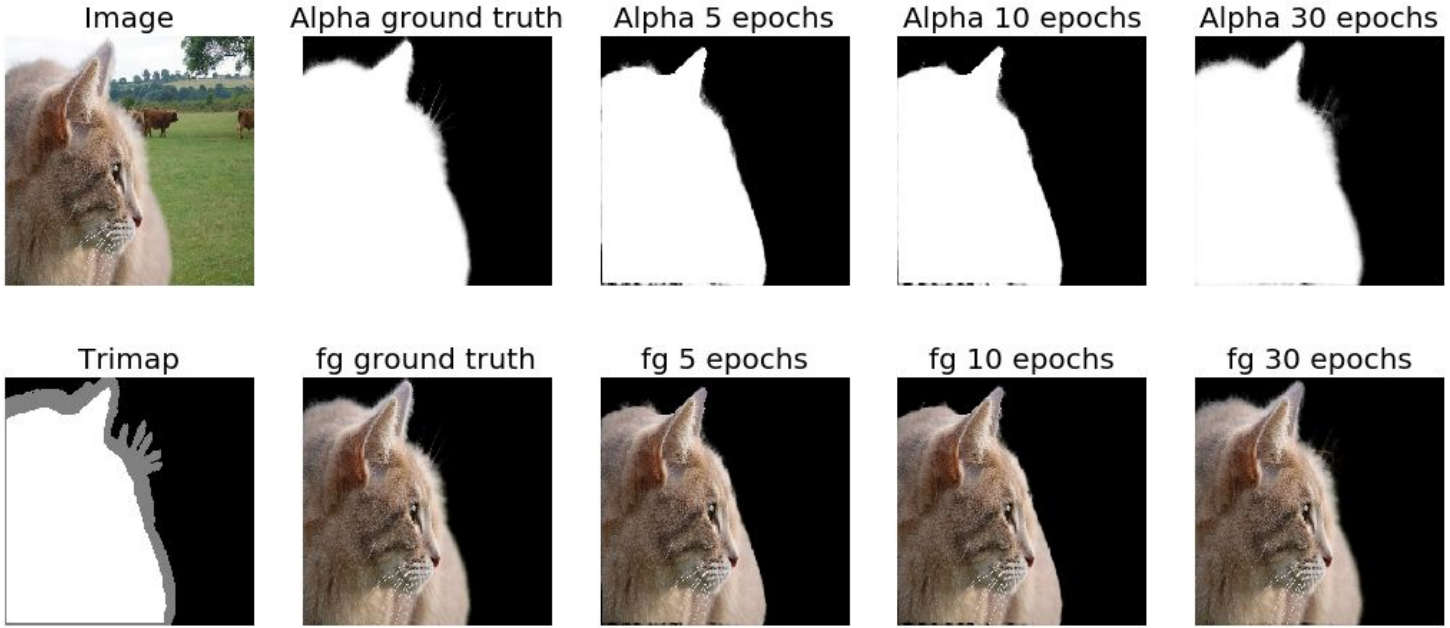


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

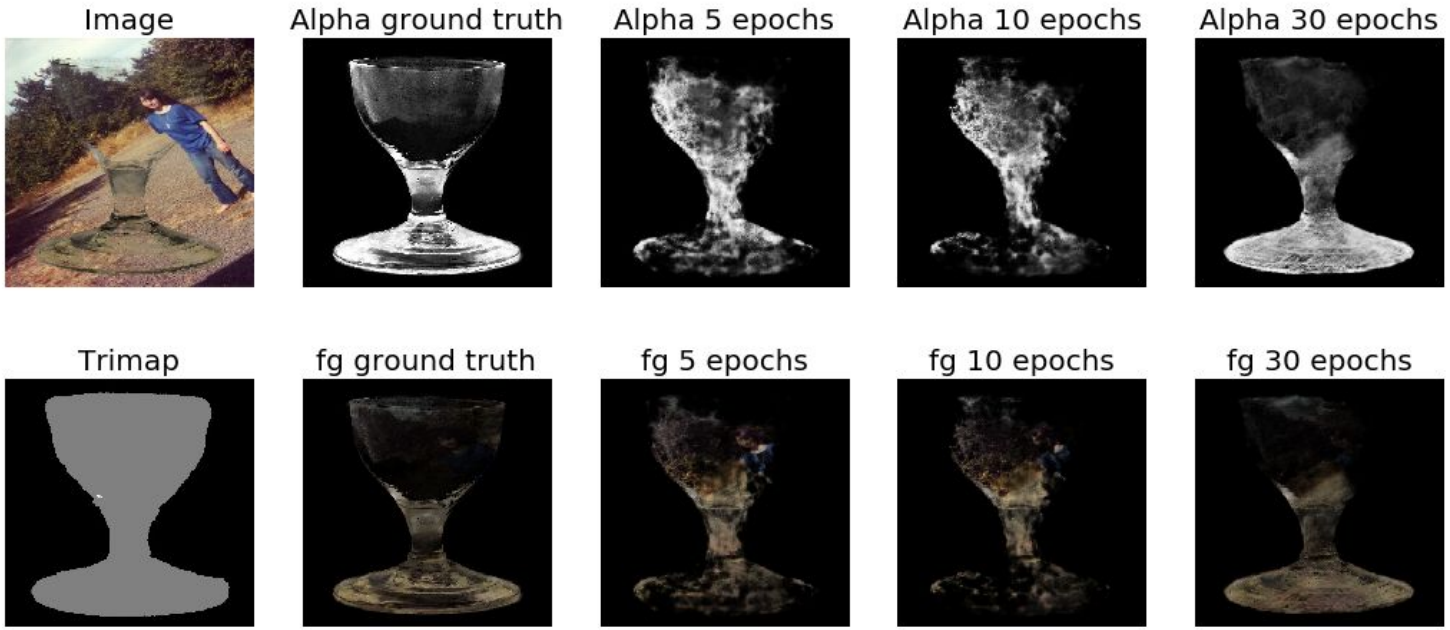


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

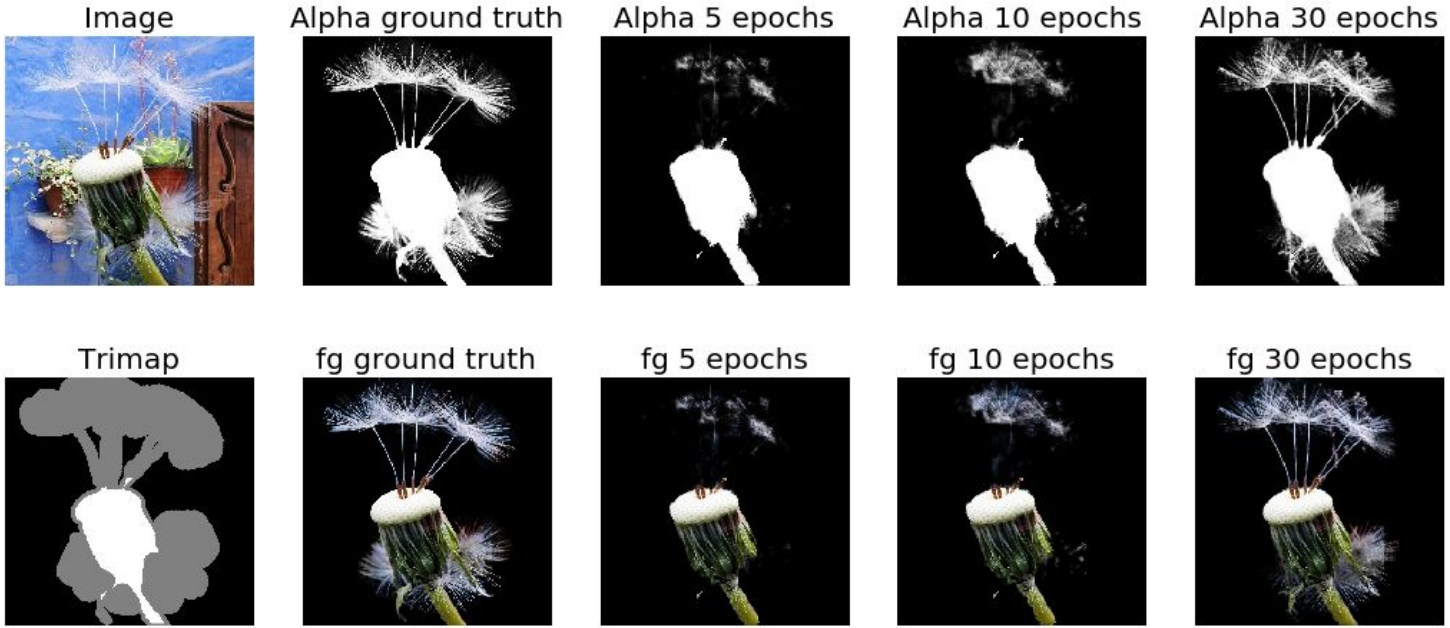


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

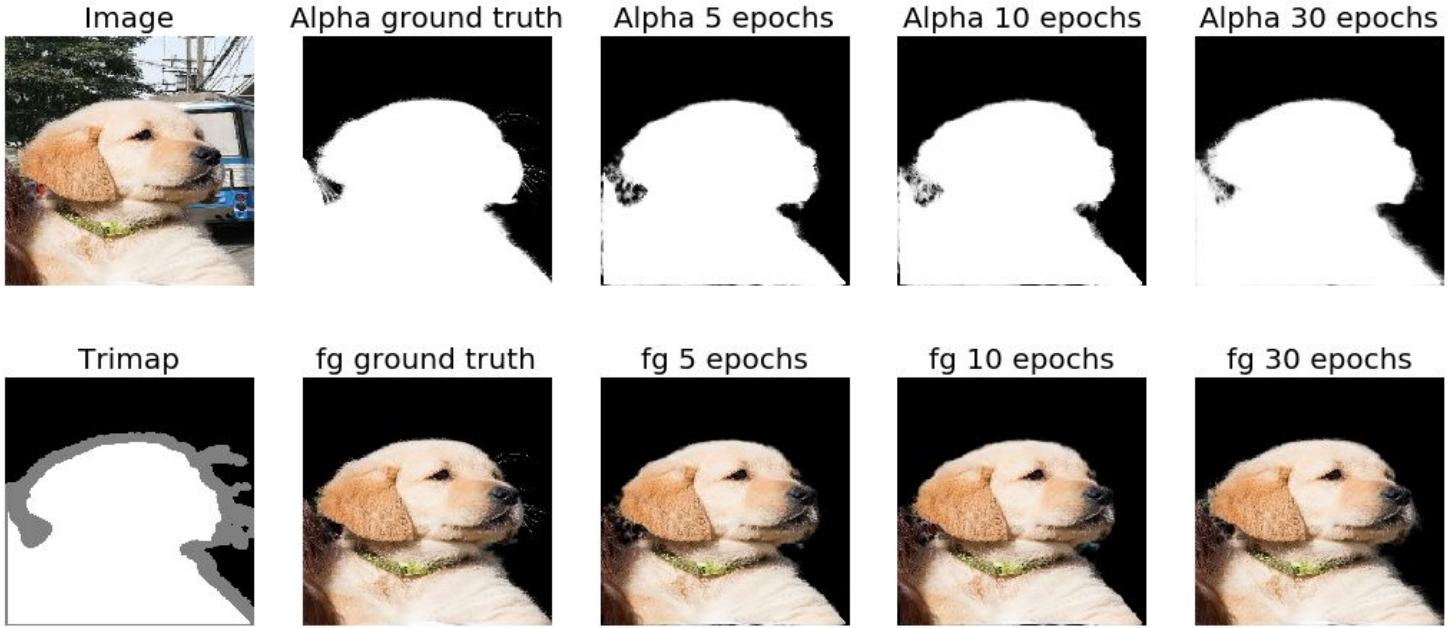


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

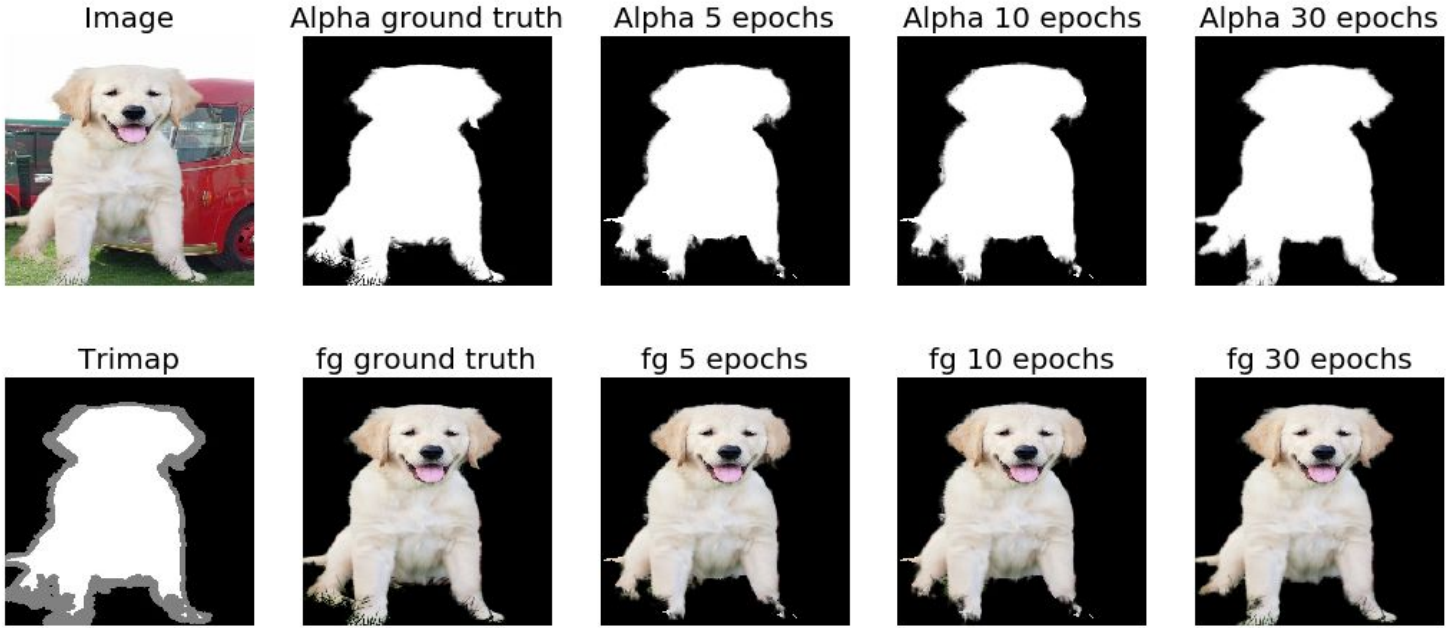


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

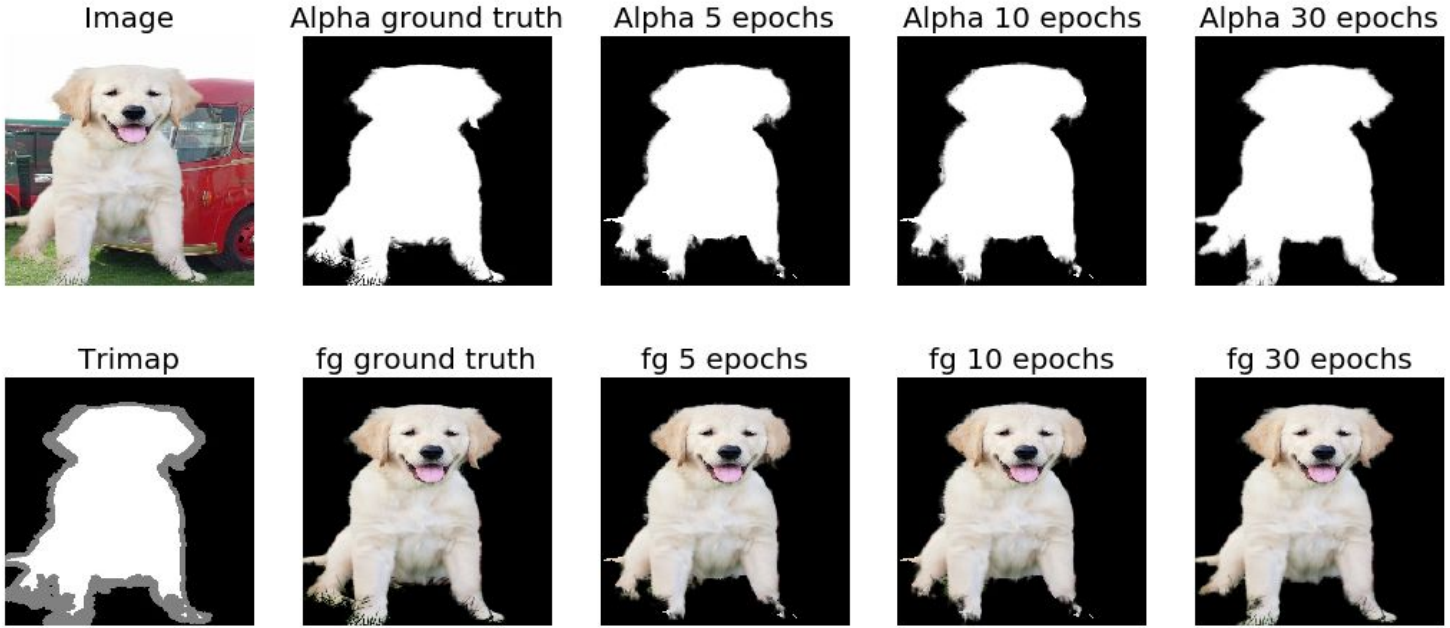


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

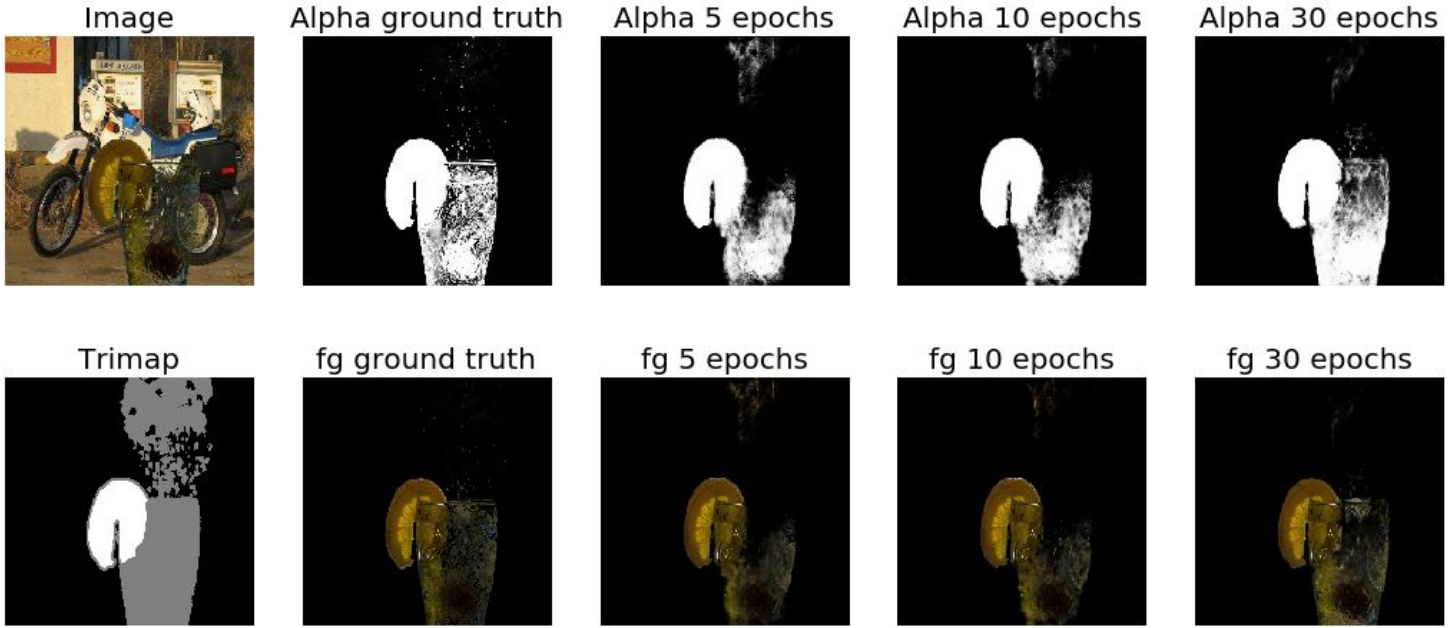


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Results

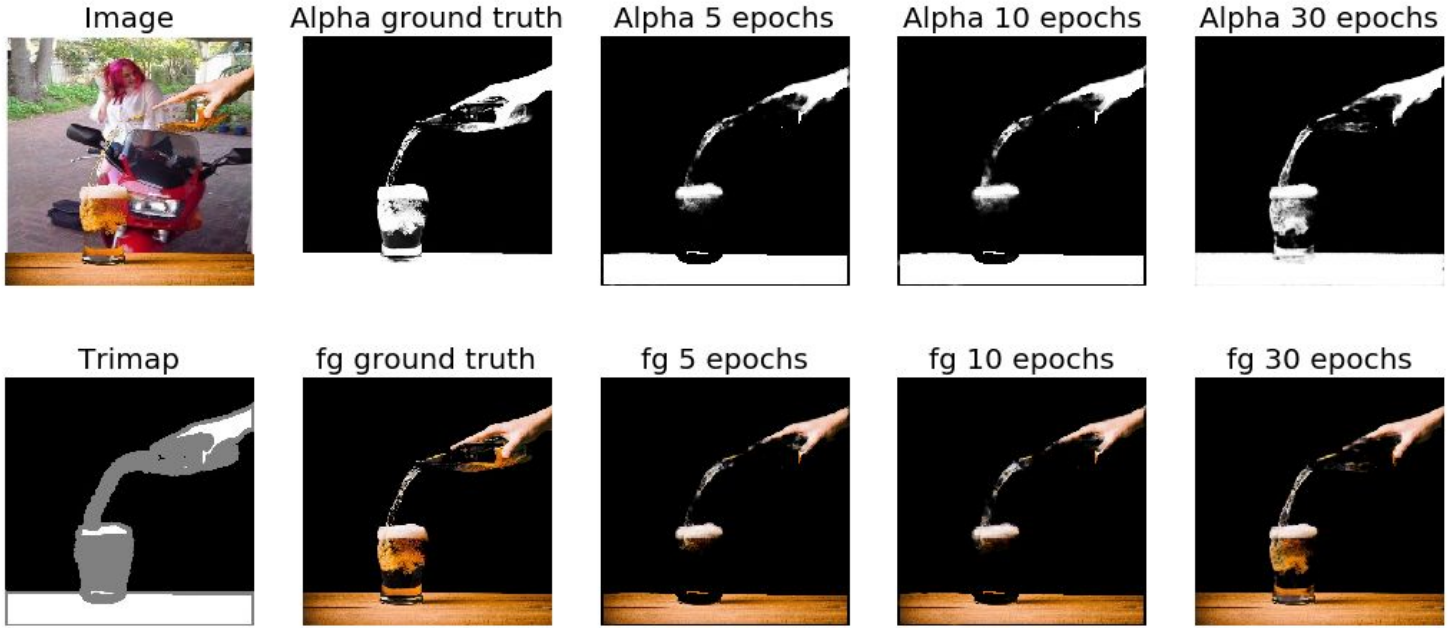
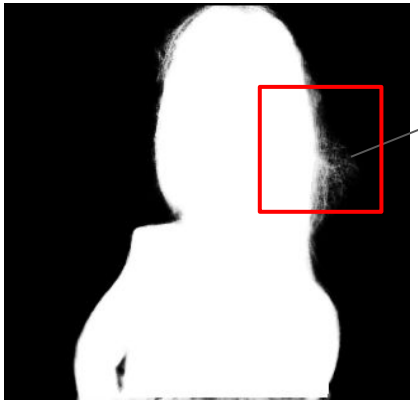


Image
Alpha ground truth
Alpha 5 epochs
Alpha 10 epochs
Alpha 30 epochs
Trimap
fg ground truth
fg 5 epochs
fg 10 epochs
fg 30 epochs

Refining the output from first part.

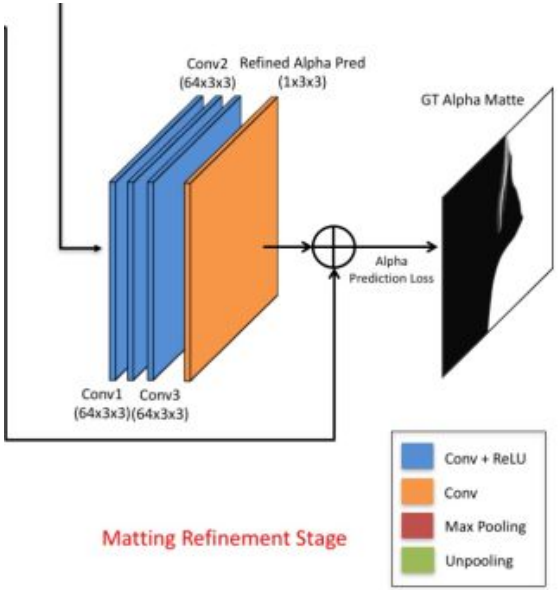


Smoothened area
(ours after first stage)

Sharpened area after
refinement stage.

- Although the alpha predictions from the first part of our network are already much better than existing matting algorithms, because of the encoder-decoder structure, the results are sometimes overly smooth.
- Therefore, we extend our network to further refine the results from the first part.
- This extended network usually predicts more accurate alpha mattes and sharper edges.

Refinement network



- This network has 4 conv layers each of them containing ReLU as activation function.
- After the refinement part the overall network is fine-tuned. Adam optimizer is used to update both parts.

```
(refine_conv1): Conv2d(4, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(refine_conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(refine_conv3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(refine_pred): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

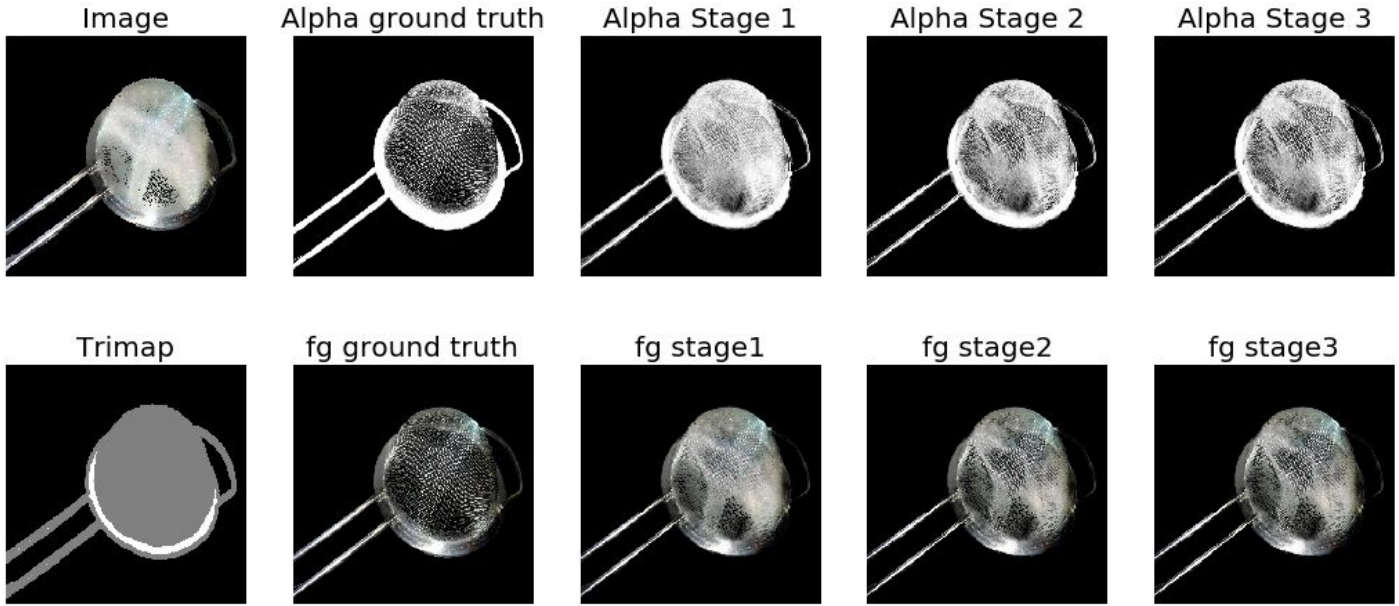
Refinement (Second stage)

- The input to the second stage of the network is the concatenation of an image patch and its alpha prediction from the first stage (scaled between 0 and 255), resulting in a 4-channel input.
- Each of the first 3 convolutional layers is followed by a non-linear “ReLU” layer.
- The 4th convolutional layer is followed by a sigmoid activation function which takes the 4th conv layer output added with output of the first stage before applying sigmoid activation.
- Since we want to retain very minute structure in the image there is no downsampling in the network.

Second and Third stage implementation

- First we train the encoder-decoder network without including the refinement net until for around 30 epochs. (Stage-1)
- After it is converged we fix the encoder-decoder network parameters and train the refinement net. (Stage-2)
- Back propagation takes place only through the refinement part of the network.
- After the second stage we fine-tune the whole network with the same training dataset (Stage-3).
- Loss for Stage-2 is only alpha compositional loss.
- Loss for Stage-3 is alpha compositional loss + novel compositional loss similar to Stage-1

Results for various stages



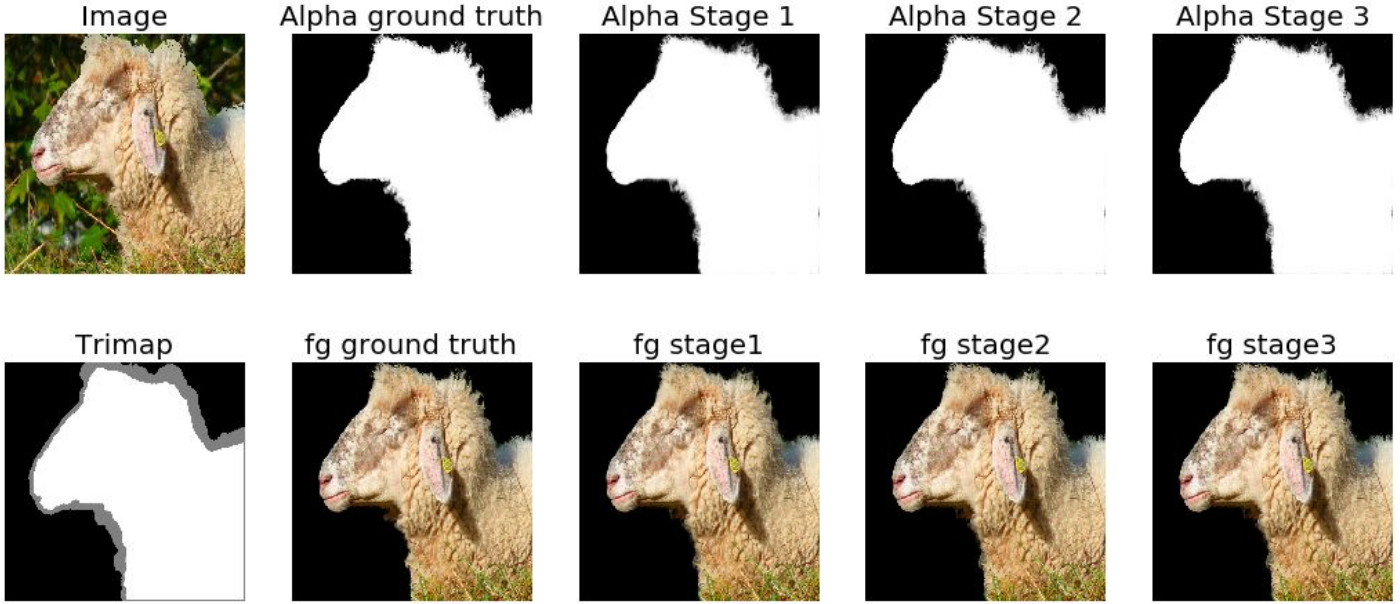


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

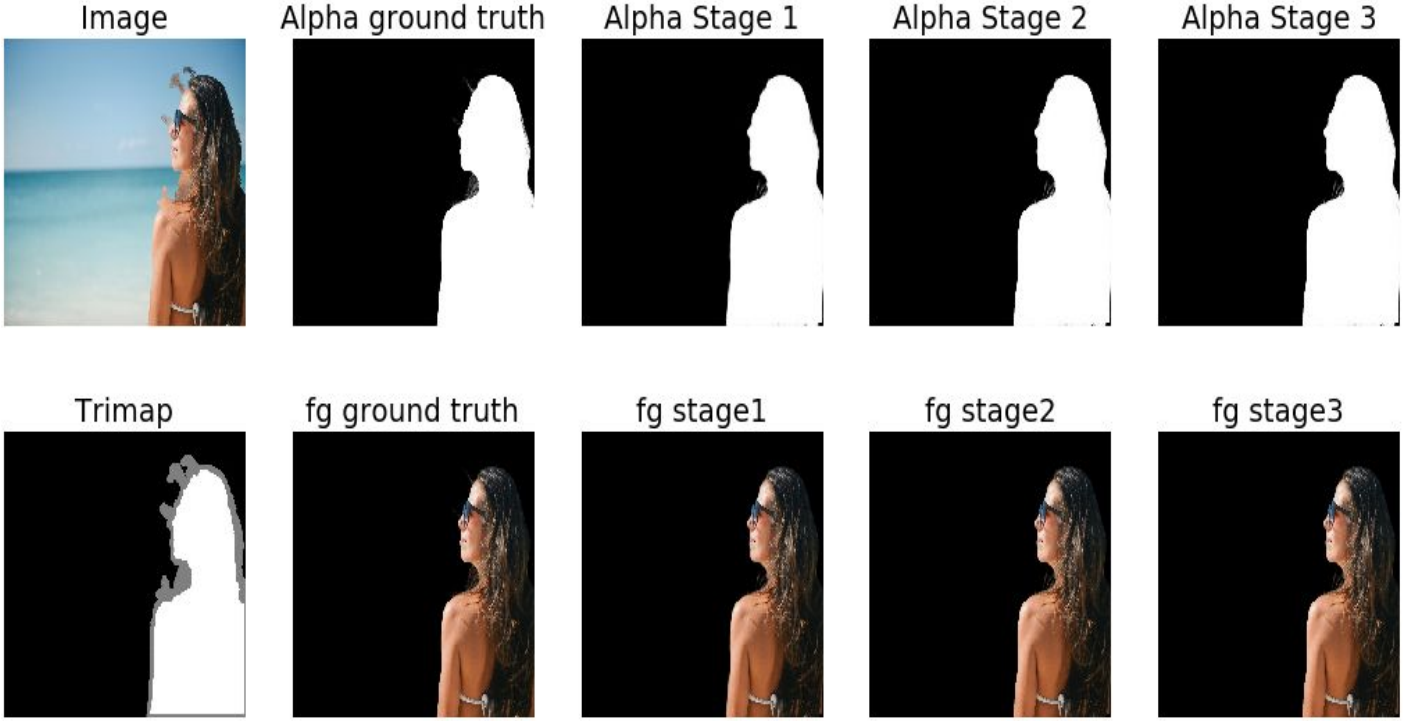


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

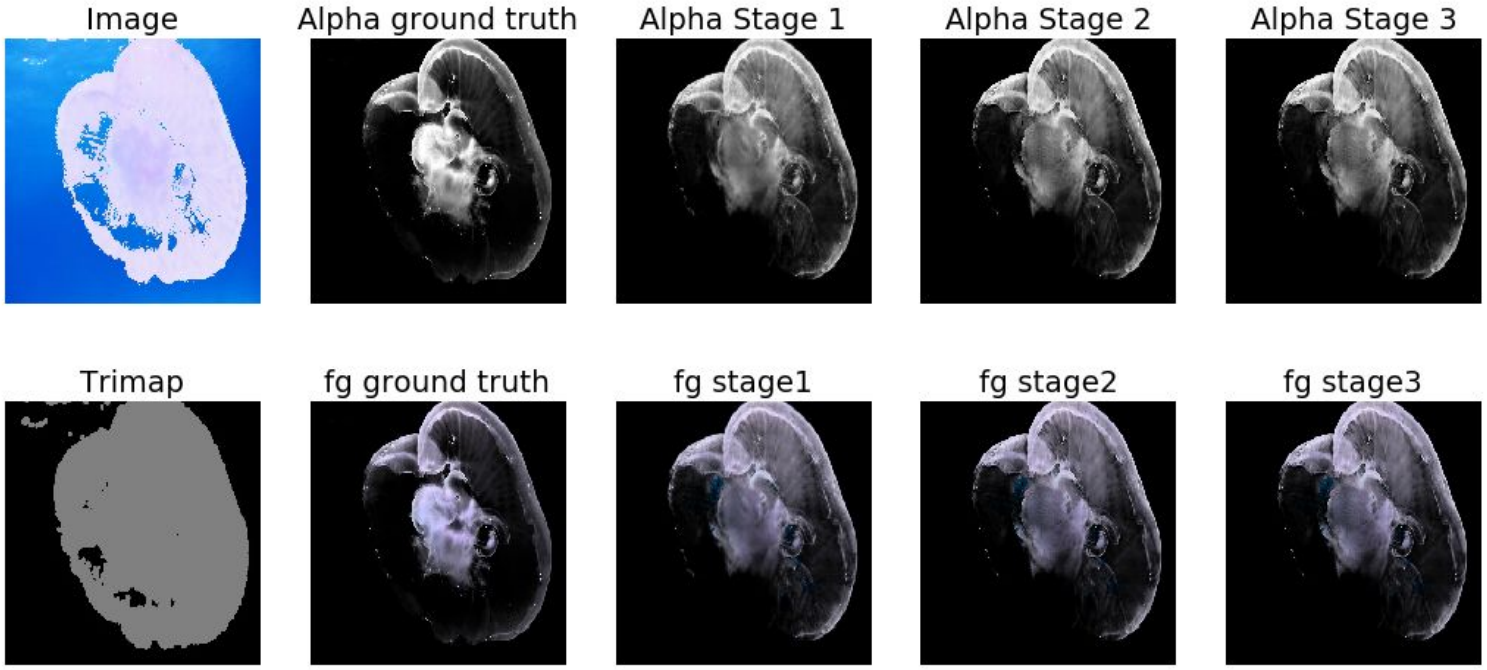


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

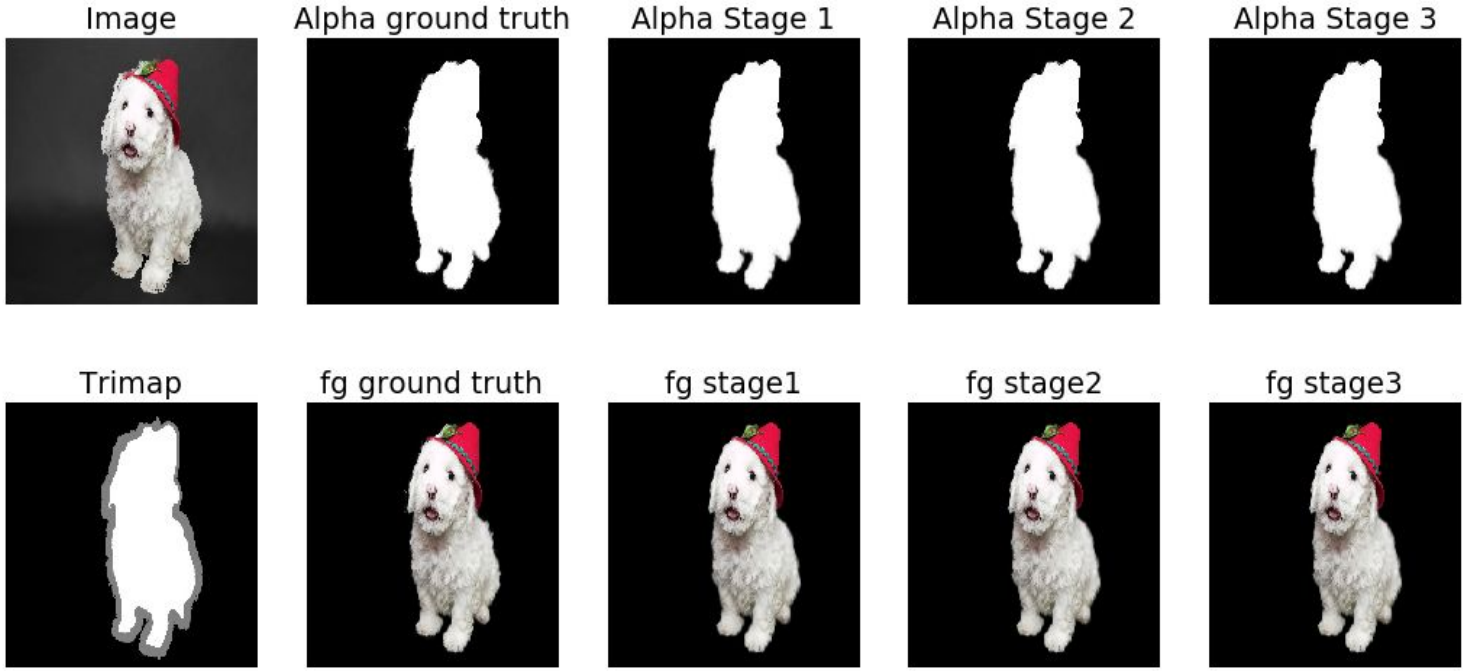


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

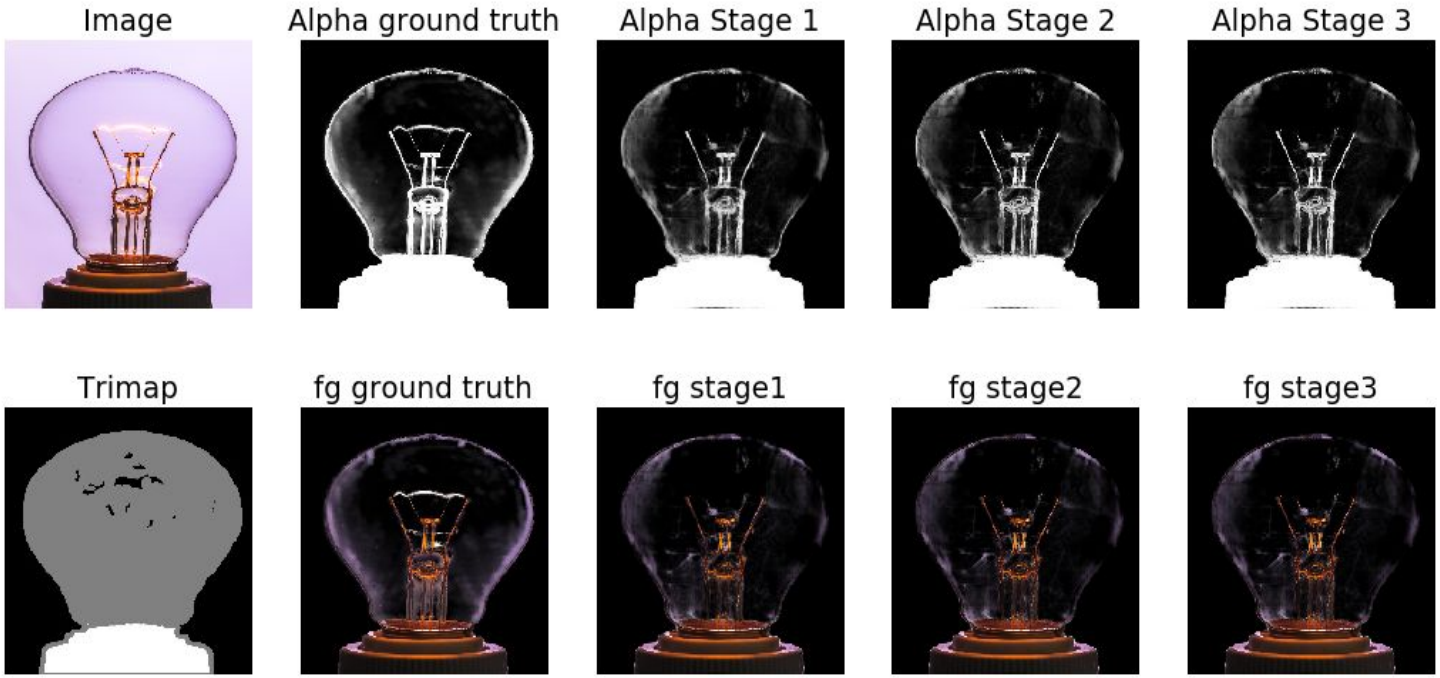


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

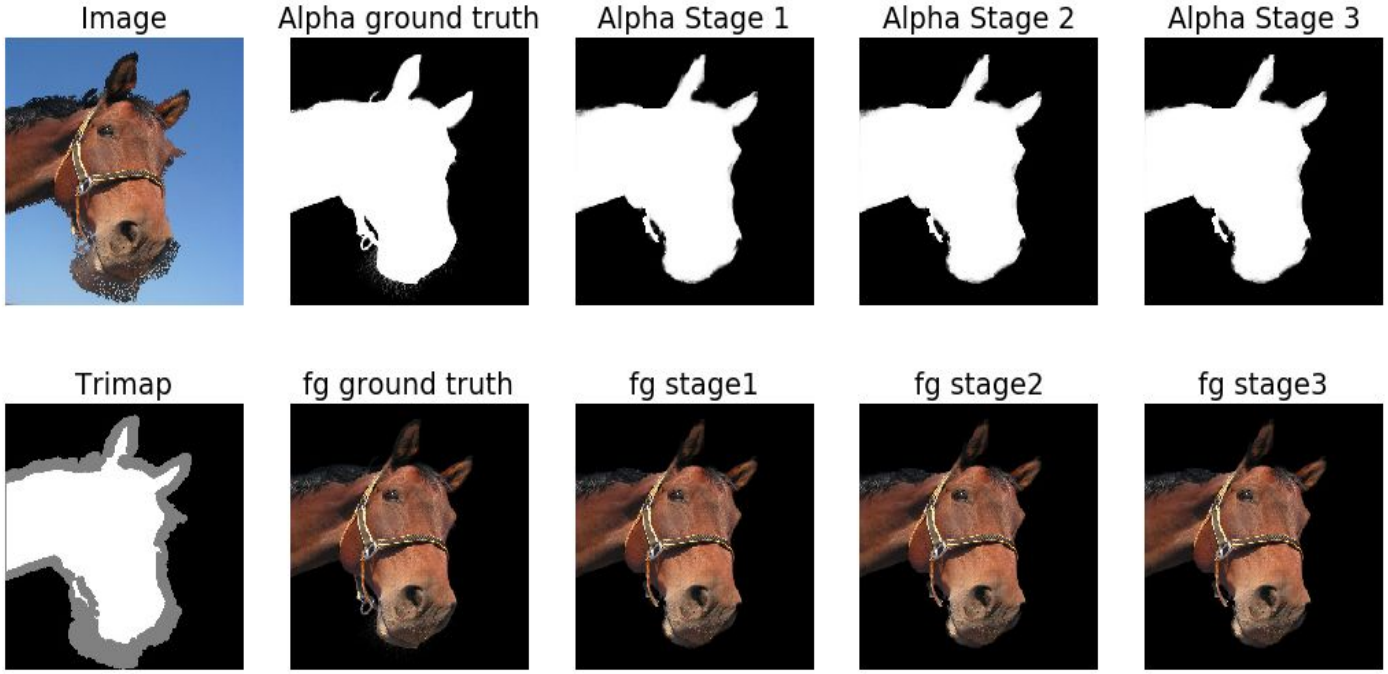


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

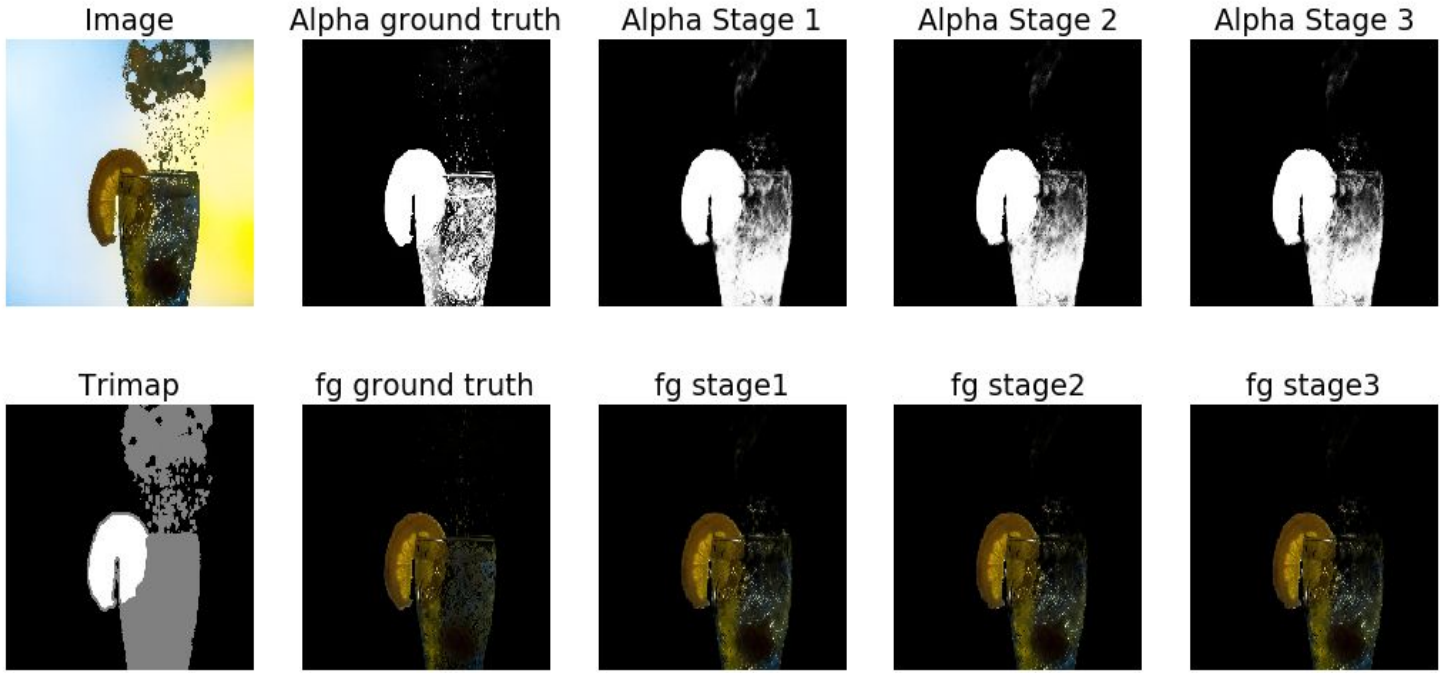


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

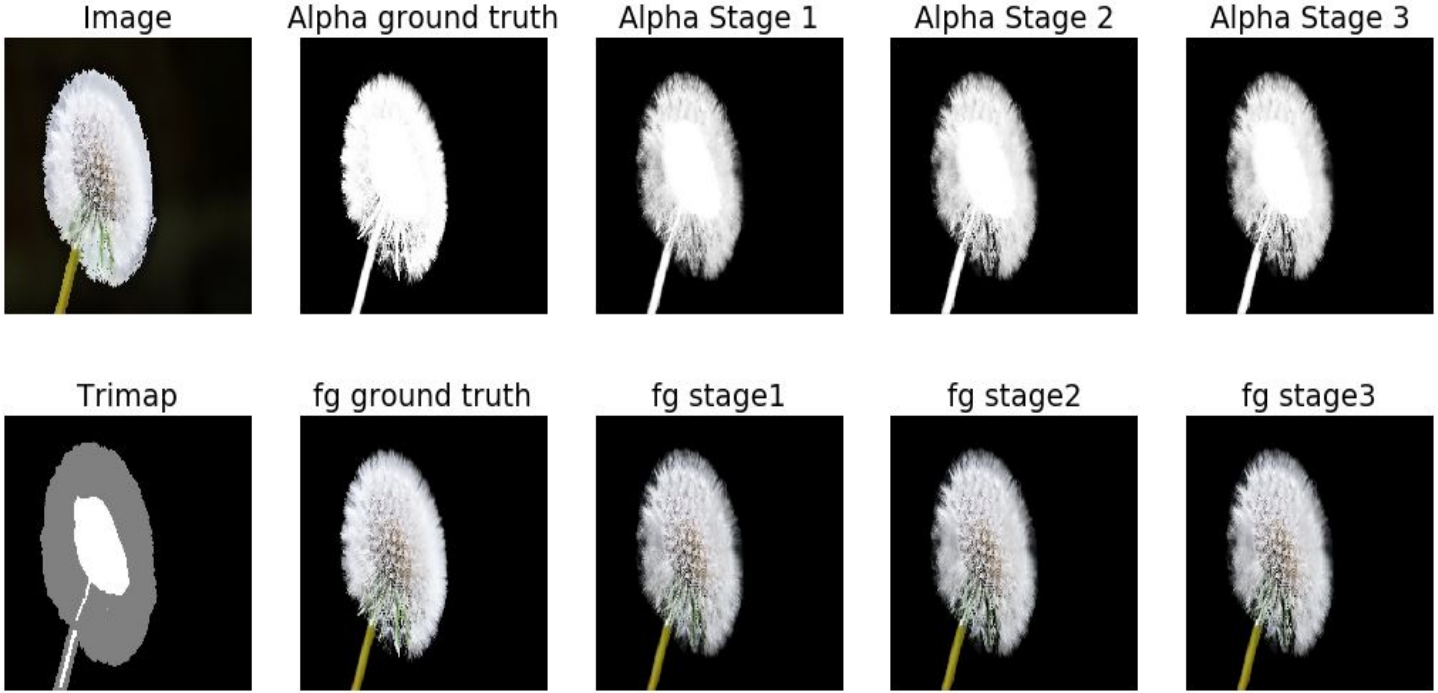


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

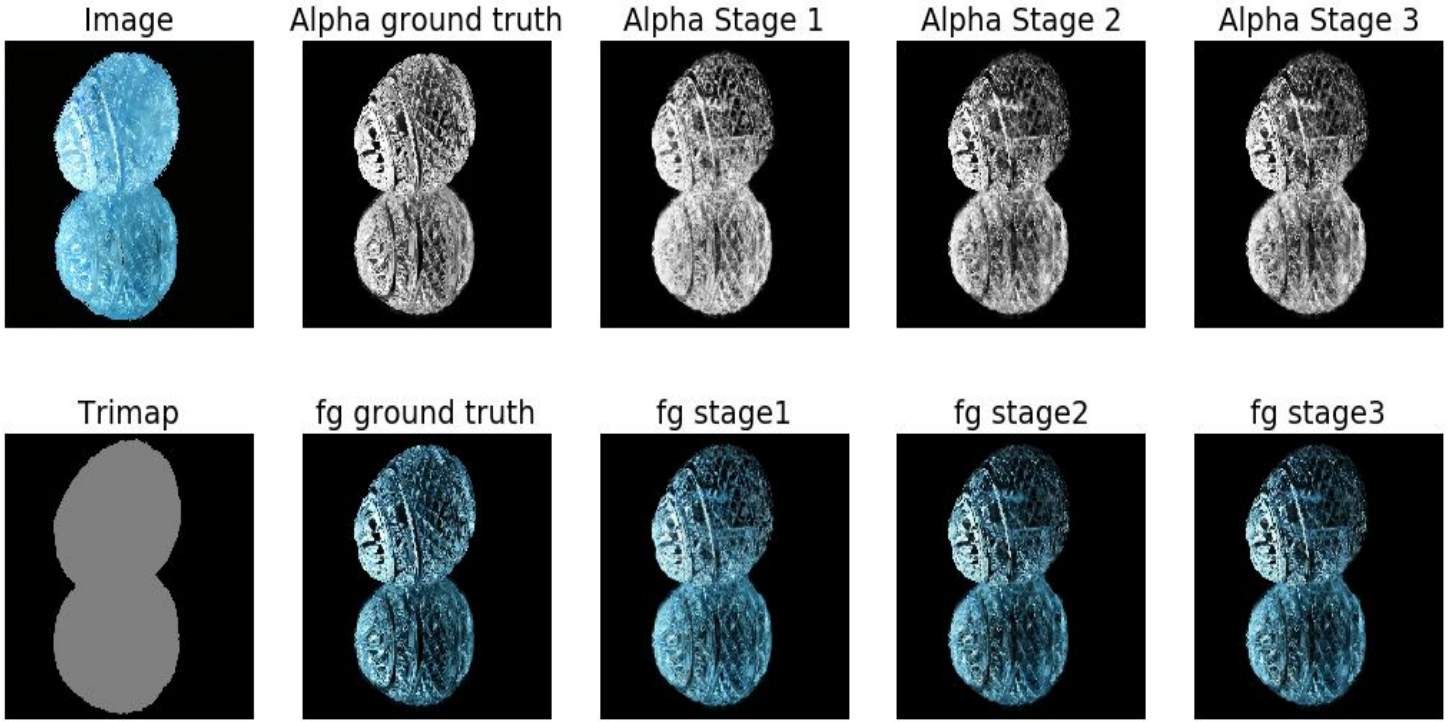


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

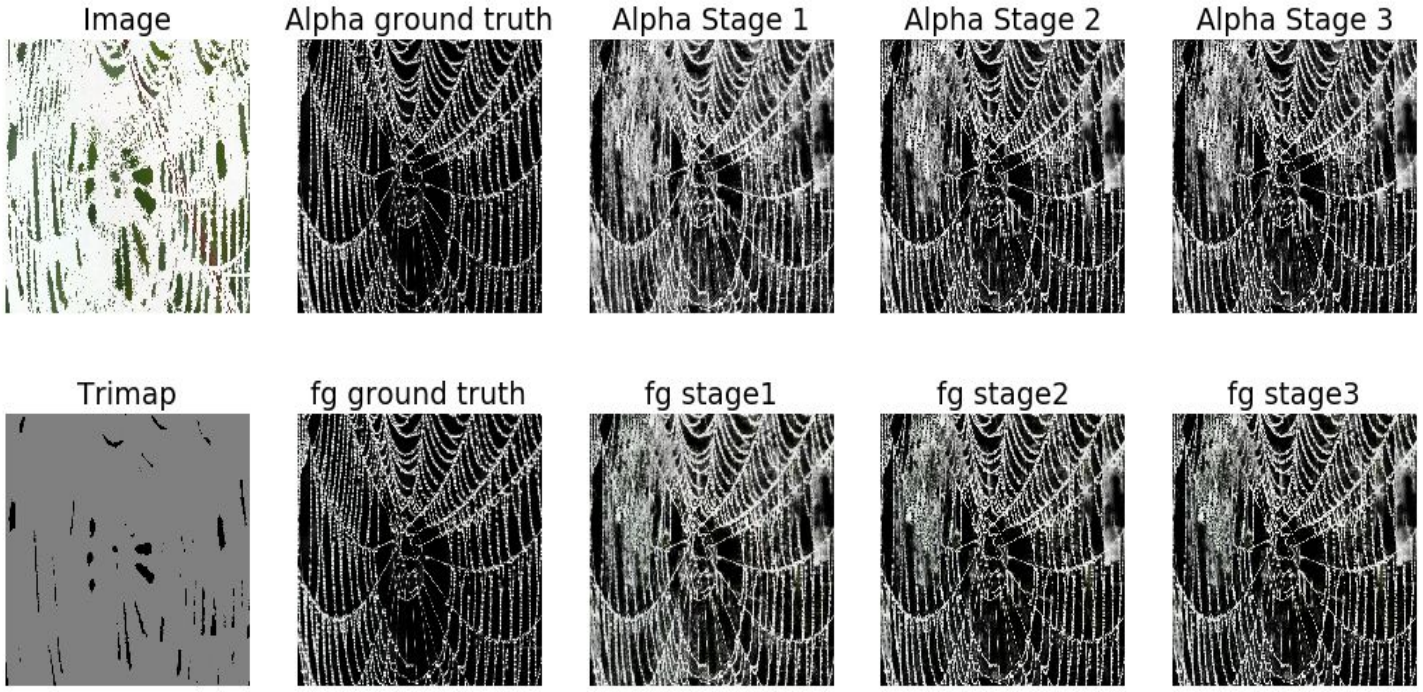


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

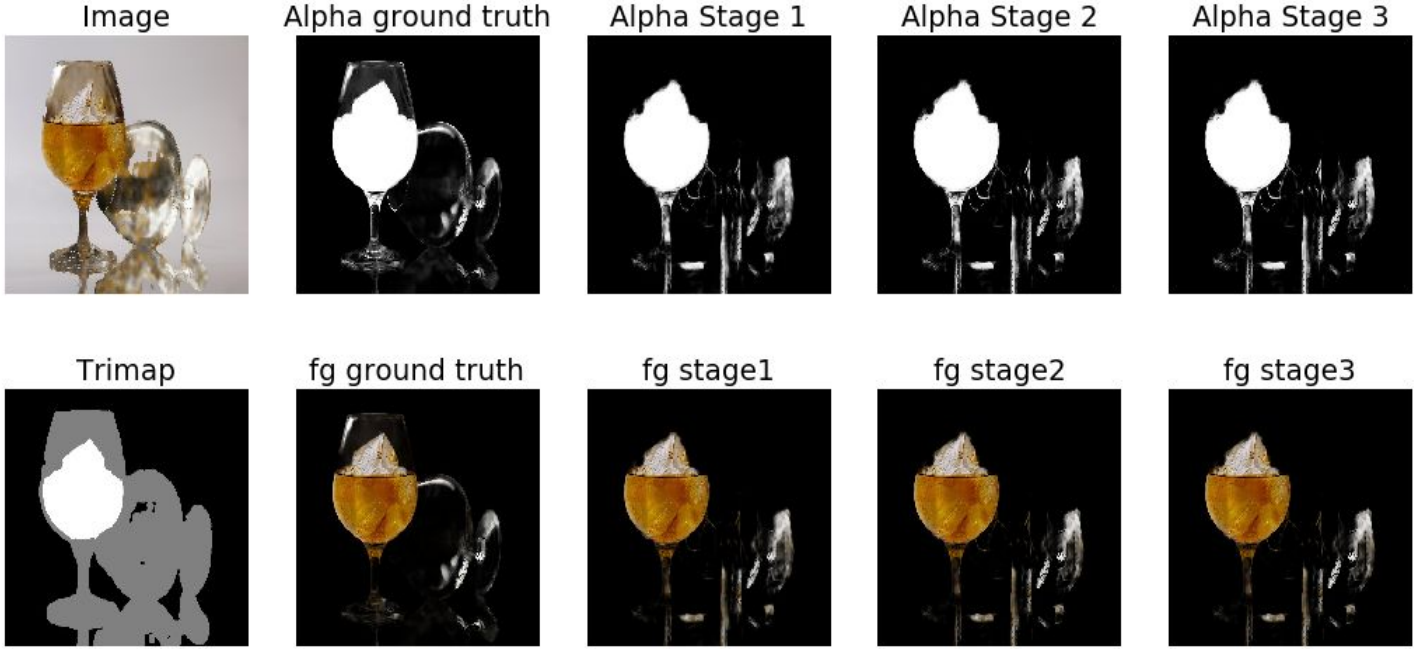


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

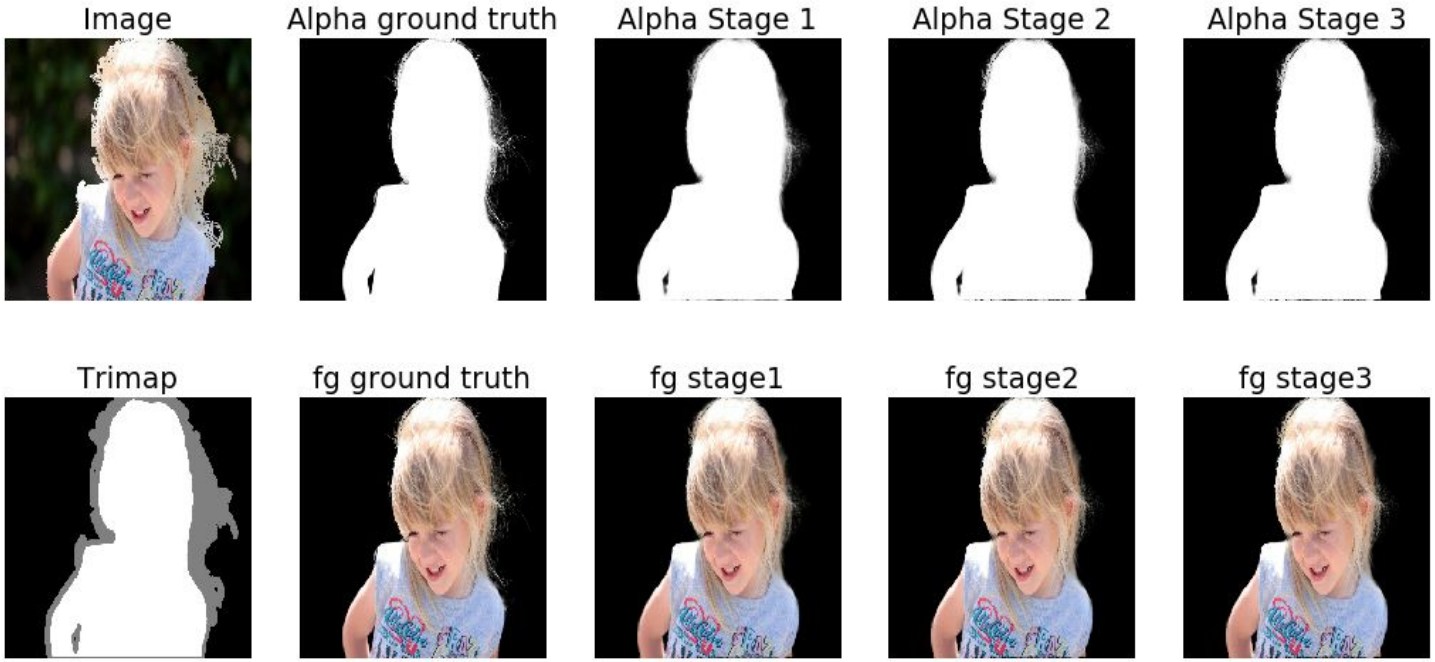
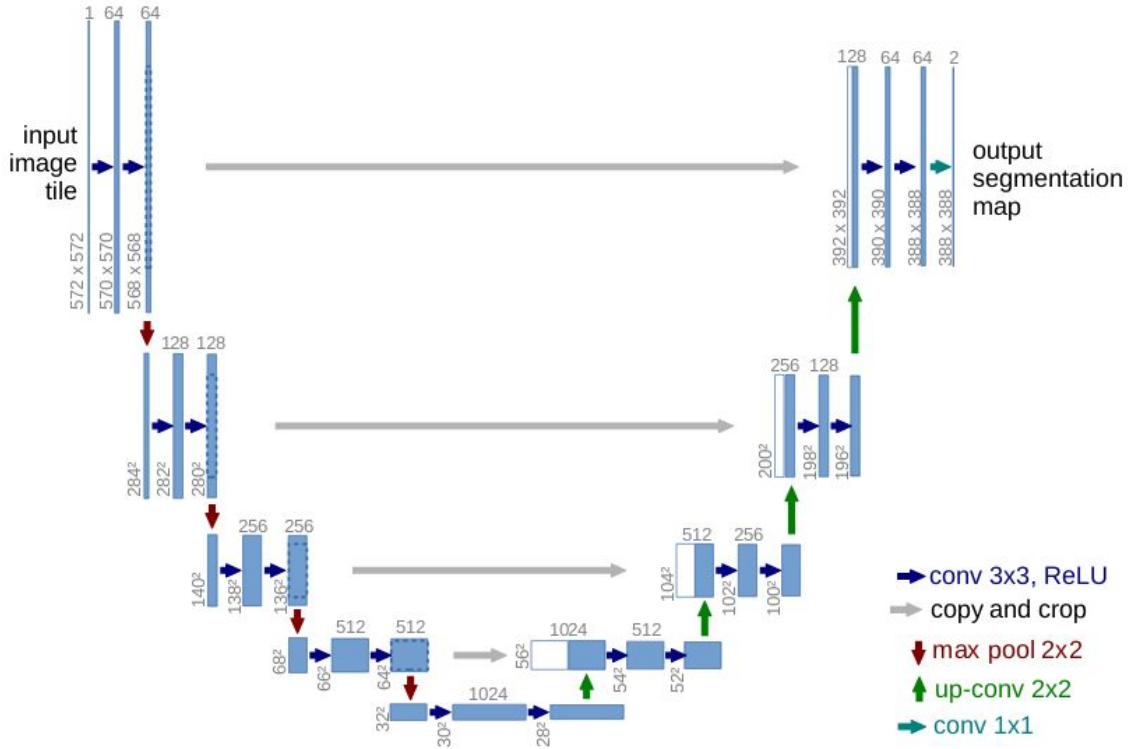


Image
Alpha ground truth
Alpha Stage 1
Alpha Stage 2
Alpha Stage 3
Trimap
fg ground truth
fg stage1
fg stage2
fg stage3

Experiments with other methods and networks

U-Net : Convolutional Neural Network



U-Net : Convolutional Neural Network

- U-Net is a convolutional neural network used for image segmentation in the case of having few training samples.
- This network relies on the strong use of data augmentation to use the available annotated samples more efficiently.

PROS:

- The localization of this network i.e this network can localize. (ability to assign label to each pixel)
- The training data in terms of patches is much larger.(than the training images)

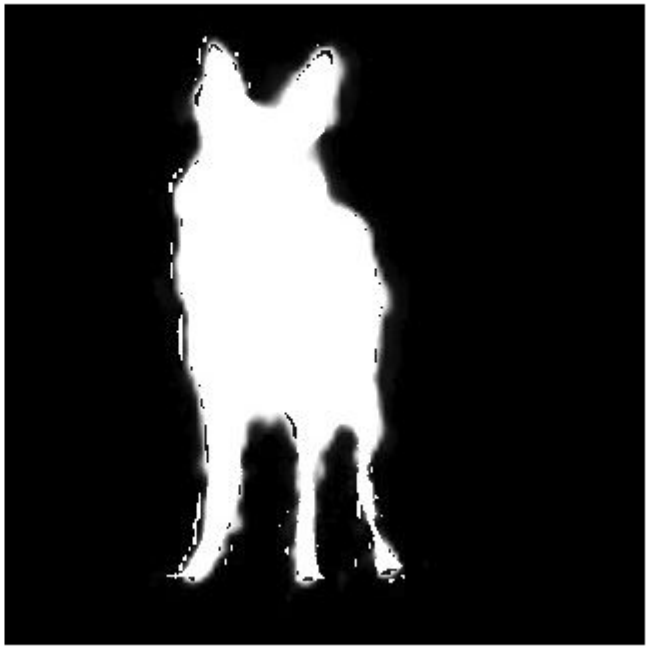
CONS:

- It is quite slow as the network is run separately for each patch.
- There is a lot of redundancy due to overlapping of patches.

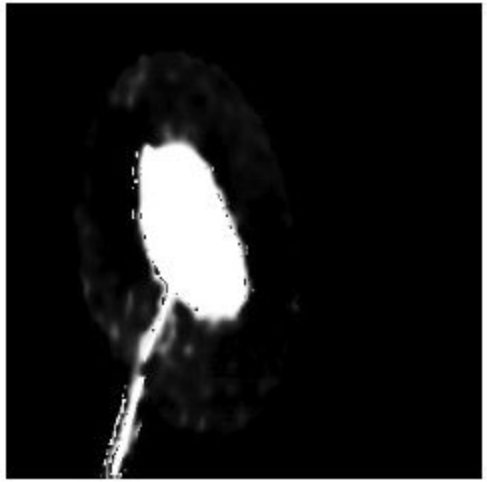
Our Encoder-Decoder VS Unet Encoder-Decoder

Using our Encoder-Decoder Network

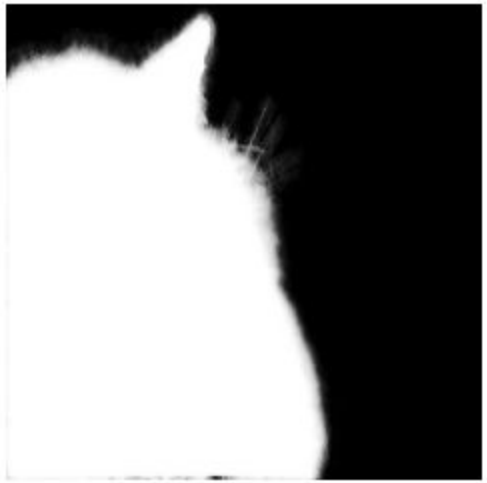
U-Net encoder-decoder network

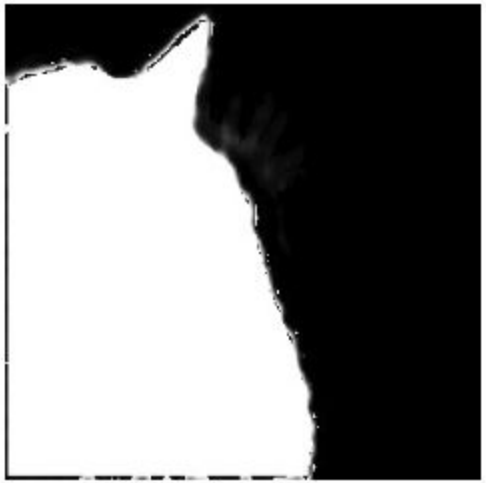


OURS vs UNet



Ours Vs Unet



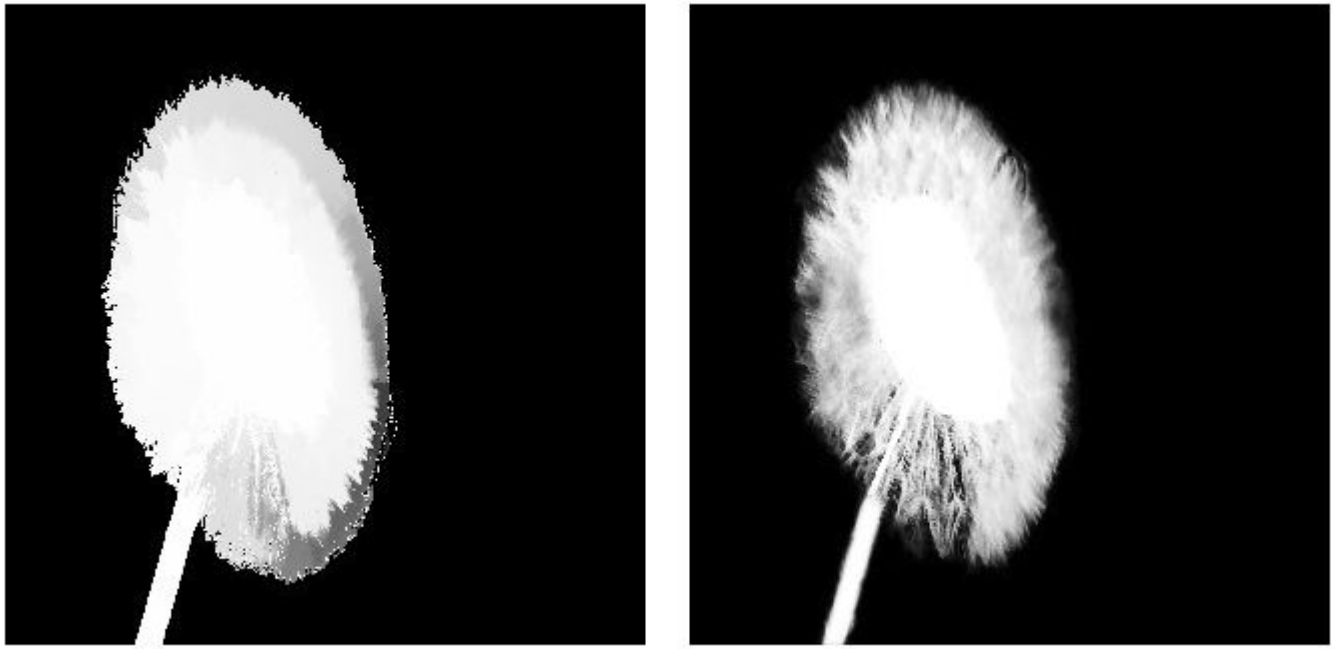


KNN-Matting

- This approach uses K-nearest neighbors (KNN) searching in the feature space for matching, and uses an improved matching metric to achieve good results with a simpler algorithm than other architectures.
- It does not require good foreground and background sample pairs for prediction.

KNN Matting vs Ours

KNN Matting vs Ours



Generalized Blue Screen Matting

- This method addresses matting problem using Triangulation method.
- Triangulation method takes two composite images and their respective backgrounds separately. Then the alpha values are obtained using these.
- It forms a matrix equation with alpha values of the image as an unknown, next it uses the concept of pseudo inverse to compute the alpha matte.
- The following equation finds the alpha using only blue-channel, this can be generalized using all the channels.

$$\begin{bmatrix} c_{f_1} \\ c_{f_2} \end{bmatrix} = \begin{bmatrix} R_o & G_o & B_o + (1-\alpha_o)B_{k_1} \\ R_o & G_o & B_o + (1-\alpha_o)B_{k_2} \end{bmatrix}.$$

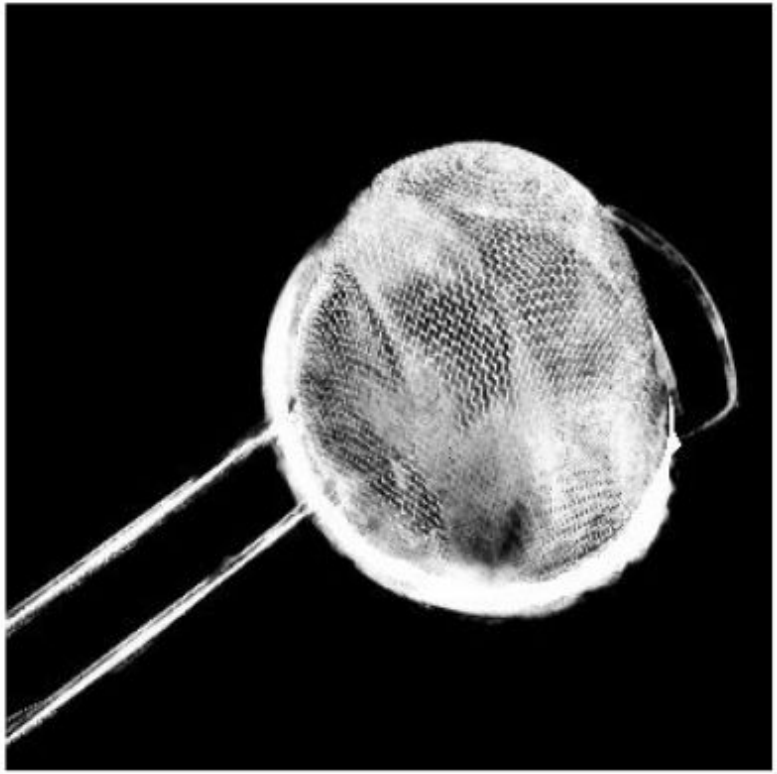
(using only blue channel)

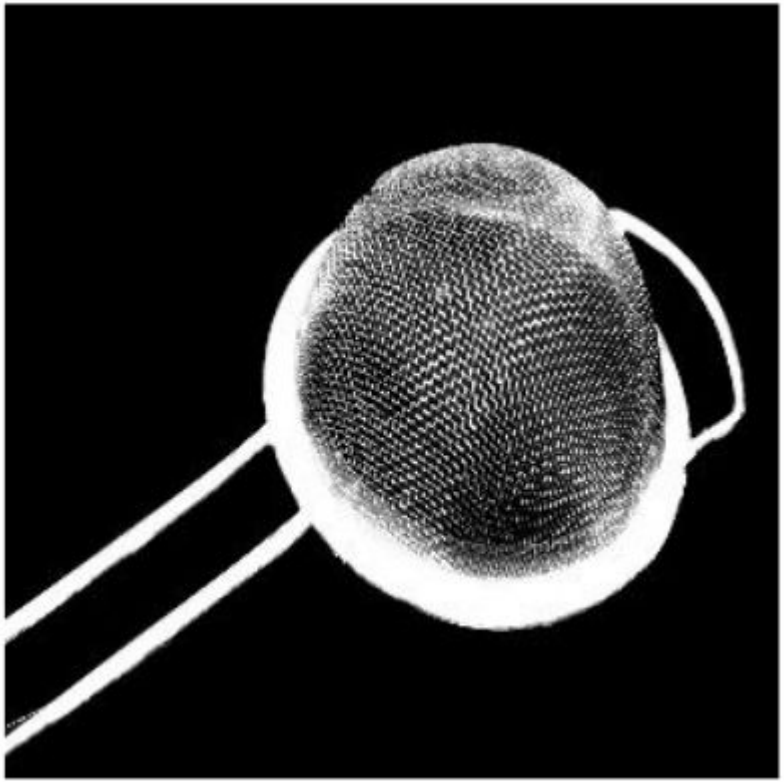
$$\alpha_o = 1 - \frac{B_{f_1} - B_{f_2}}{B_{k_1} - B_{k_2}}$$

(Generalized form)

$$1 - \frac{(R_{f_1} - R_{f_2}) + (G_{f_1} - G_{f_2}) + (B_{f_1} - B_{f_2})}{(R_{k_1} - R_{k_2}) + (G_{k_1} - G_{k_2}) + (B_{k_1} - B_{k_2})}$$

Ours vs Generalized Blue screen matting





Ours vs Generalized Blue Screen Matting

Observations and Goodness of our algorithm

● Blue screen matting has almost close to 100 accuracy. It can be seen that our method is able to achieve results just closer to this.

● Deep-image-matting is able to get significant results even without the use of background images which is the case in Blue-screen matting.

Datasets and dependencies

- https://sites.google.com/view/deepimagematting dataset containing 43100 training images with 431 unique objects and 1000 background images for each foreground image. 1000 testing images with 50 unique objects and 20 background images.
- Training dataset background images: COCO
- Testing dataset background images: PASCAL
- Dependencies: pytorch, GPUs (already have ada accounts) and other python packages.

Thank You