Final Project: Image Background Removal

1. Introduction

In this project, we develop an image background removal system using machine learning techniques. Image background removal is an essential component of image processing with numerous practical applications in the real world. Here, we have trained an image in-painting model using the Partial Convolution to restore or reconstruct an image by filling in the missing parts with plausible information. We integrated the trained image in-painting model with a pre-trained image segmentation model which removes a particular object of an image.

In this final project report, we present the methodology and results of our image background removal project. We describe in detail the steps involved in training and evaluating the model, as well as the integration of the pre-trained image segmentation model. We also discuss the challenges encountered and the solutions implemented to optimize the system's accuracy and speed. Finally, we provide a comprehensive evaluation of our system's performance.

2. Image inpainting

Source: https://www.nvidia.com/research/inpainting/index.html

Image in-painting is the task to fill in parts of the image that is masked out in order to seamlessly delete unwanted elements from a picture. The technique that we are going to use in particular is utilizing deep learning models to perform the deletion and adjacent scene interpolation for seamless resulting image. Previously GAN-based approaches were studied [2, 3]. More recently, in 2018 Liu et al. [4] studied partial convolution-based UNet architecture to perform the image in-painting task resulting in substantial improvements from the previous approach.

3. PConvUNet

The PConv (Partial Convolution) network is a deep learning-based approach for image in-painting. The network is designed to handle images with irregular holes, where the missing regions have arbitrary shapes. The PConv network architecture is based on the U-Net architecture, which is a popular architecture for image segmentation. The U-Net architecture consists of an encoder and decoder network, with connections between the corresponding layers of the encoder and decoder. The PConv network extends the U-Net architecture by introducing partial convolutions in the decoder network.

3.1. Partial Convolution

This chapter will cover the important details of partial convolutions and how they are used to handle irregular holes in images. In [4], partial convolutions were suggested to separately compute the mask and the image. To provide some context, in the previous approaches, masked parts were blacked out then the conv filters were applied with no differentiation between the "real" pixels and the masked pixels.

Below provides the equation for partial convolution as defined in [4].

$$x' = \begin{cases} \mathbf{W}^T(\mathbf{X} \odot \mathbf{M})\frac{\text{sum}(\mathbf{1})}{\text{sum}(\mathbf{M})} + b, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$m' = \begin{cases} 1, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The relevant variables are:

- W= Kernel weight
- X= Feature values in the filter region
- M= Mask
- 1= Matrix with all ones
- b= bias

As shown in the equations, while traditional convolutional filters apply a fixed-size kernel to all regions of an image, PConv applies a variable-size kernel that depends on the amount of non-masked data in the region. The key idea behind PConv is to process only the valid (non-masked) parts of an image and to use a weighted average of the valid parts to fill in the masked (invalid) parts. This allows PConv to effectively handle missing or incomplete data in an image while preserving the spatial information of the image.

3.2 Network architecture

The PConv network is based on the UNet architecture and extends it with partial convolutions in the decoder. In the PConv network, as in the UNet network, the encoder network performs a series of convolution and pooling operations to reduce the spatial resolution of the input image while increasing the number of feature maps. The decoder network, however, consists of a series of partial convolution layers followed by upsampling layers. The partial convolution layers take as input the feature maps from the corresponding encoder layer and a binary mask that indicates which pixels are missing. The convolution operation is only applied to the non-missing pixels, and the missing pixels are set to zero in the output feature maps. The output of each partial convolution layer is concatenated with the corresponding encoder layer's feature maps, and this concatenated feature map is used as input to the next partial convolution layer. This allows the PConv network to generate plausible content for missing regions in the input image.

3.3 Perceptual loss and style loss

Interestingly, [4] adopts a loss based on VGG16 model results. The VGG16-based loss is computed using the feature representations of the VGG16 network, which is a pre-trained deep convolutional neural network commonly used for image classification tasks. Specifically, the loss is computed as the mean squared error between the feature representations of PConvUNet in the first three layers and the feature representation at the first three pooling layers of the VGG16 network.

This VGG16 loss has two parts: perceptual loss and style loss. For exact equations please refer to the original article. This two-part loss is implemented to help training filters in learning visual features and stylistic features correctly

4. Overall Implementation & training

4.1 Image segmentation module

For the image segmentation, we have used a pre-trained Mask RCNN model from OpenMMLabs detect. This model is trained on the coco instance segmentation dataset and is capable of identifying 80 different object types including categories like people,car,dog etc. The model's output consists of bounding box information along with the instance segment masks for the objects and labels.
For the purpose of this project, we also added an interactive method that requires the user to select the object to remove from the final image. The input reads the x and y coordinates and returns the bounding box which contains the selected coordinates. The bounding box and mask information is used to save the mask file for the selected object and the image. The created mask files and the images are provided as input to the inpainting model.

4.2 Image in-painting module

As discussed earlier, we implement PConvUNet as our image in-painting module. We implemented the model and training code that adheres to the standards in Tensorflow2. We take reference of an open source version of TF1.0 implementation [6] in implementing model layers, but most of the code including the layers, model, data input pipeline is re-wrote to incorporate Tensorflow 2 functionality. Based on the code base we fully trained the model from scratch.

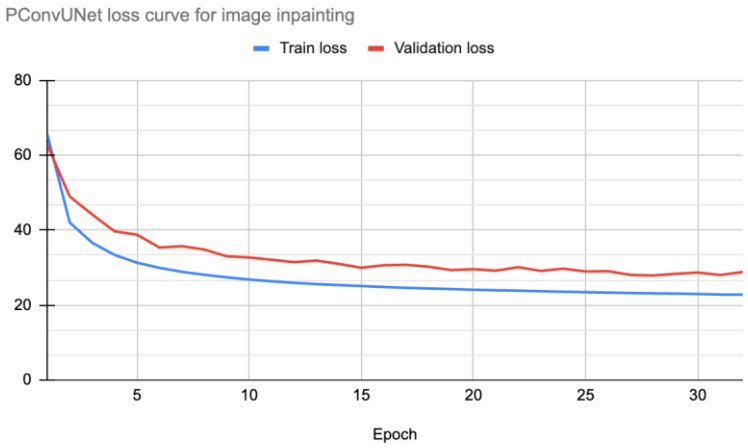
5. Training

5.1 Data

(Source: https://paperswithcode.com/dataset/imagenet)

While the original paper utilizes all 150+GB of ImageNet data (1.2M images), we take a subset comprising 10% of the training data (~17GB, ~100K images) to train the model due to the limited capacity of training resources. Similarly validation loss is computed on 5% of the original dataset.

5.2 Training

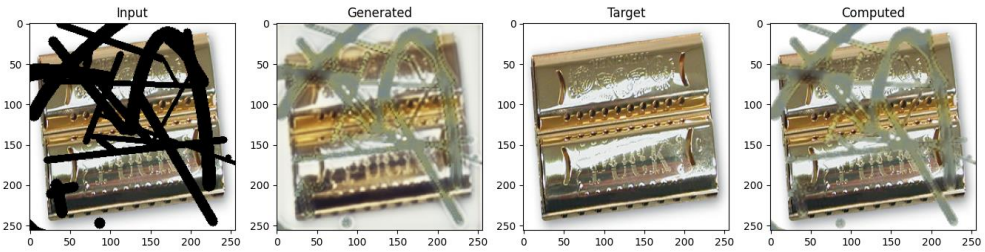
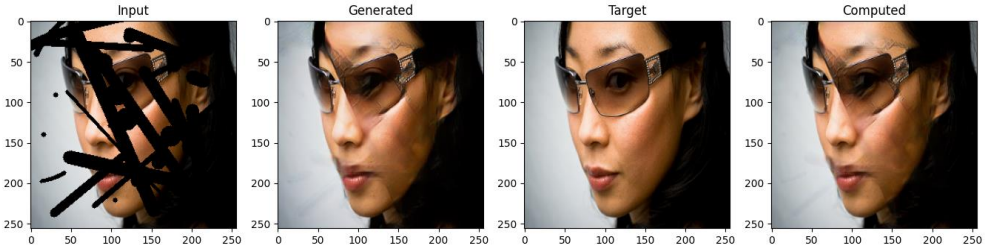
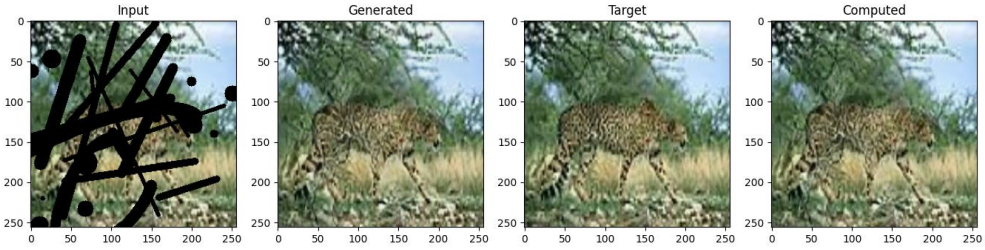


We train the model in TAMU's HRPC infrastructure [7] on an A100 GPU for 24 hours. As shown in the above plot, both the training loss and validation loss steadily decreases. But due to the time

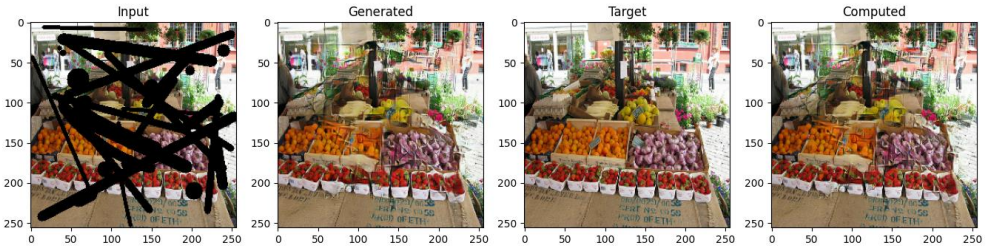
constraints we stopped the training early at 32 epochs. Each epoch consisted of 10,000 training steps and 1,00 validation steps. The plotted loss is the mean of all training steps (and validation steps).

5.3 Validation plots

For every epoch, we validated the checkpoints in 100 of the validation set images. Below are sampled validation plots. Following the plots, you can see that the model gradually gets better in filling the randomly created masks.

Epoch 1

Epoch 5

Epoch 10

Epoch 20

Epoch 30



6. Results

6.1. Mask creation using image segmentation

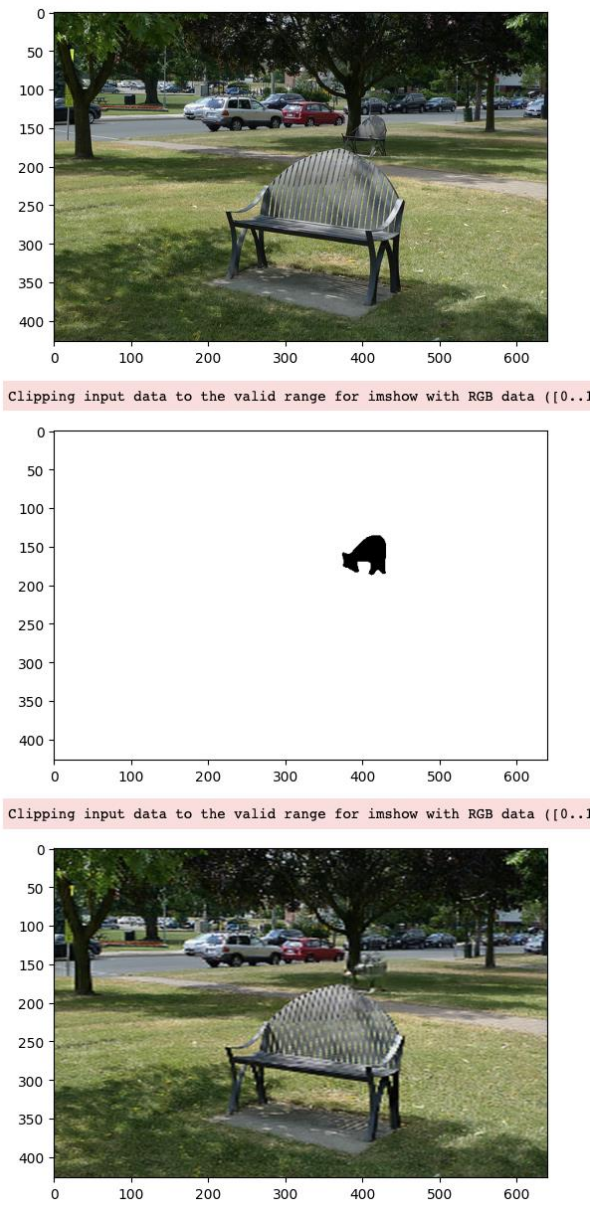
(Above) test images, (below) masks extracted using image segmentation module.

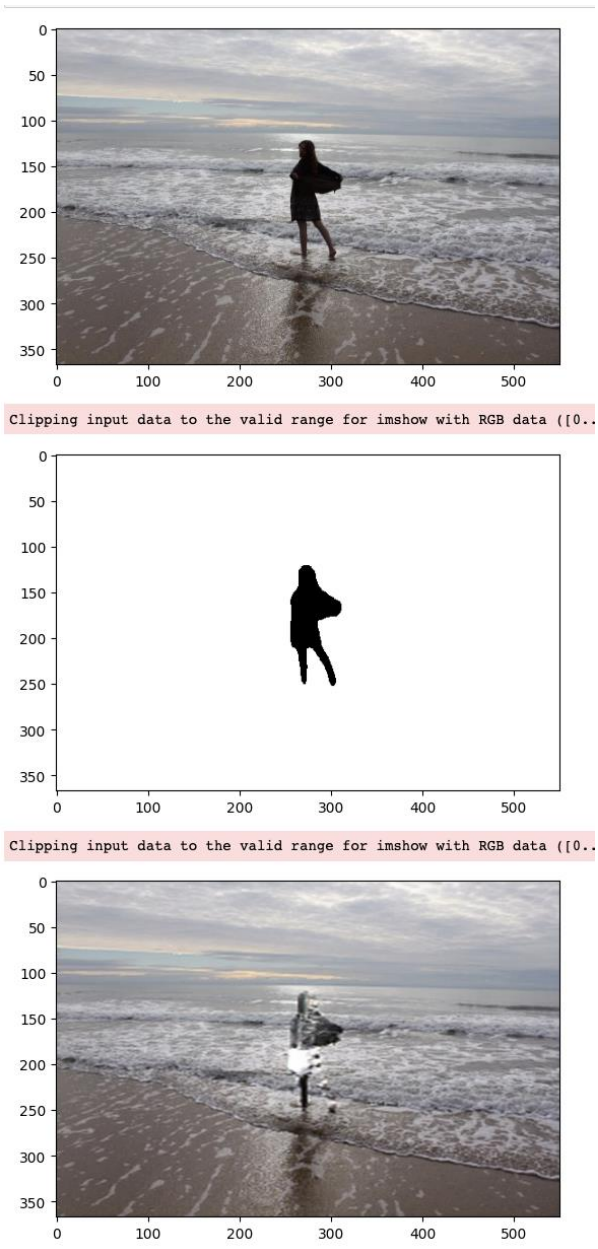
To select the object to delete, we utilize an image segmentation module discussed in previous sections to 1) identify objects in the scene, and 2) create masks to remove a portion of the image and convert the mask into suitable format to feed into the inpainting model. The above figure depicts a few examples of the masks that were generated using this process.

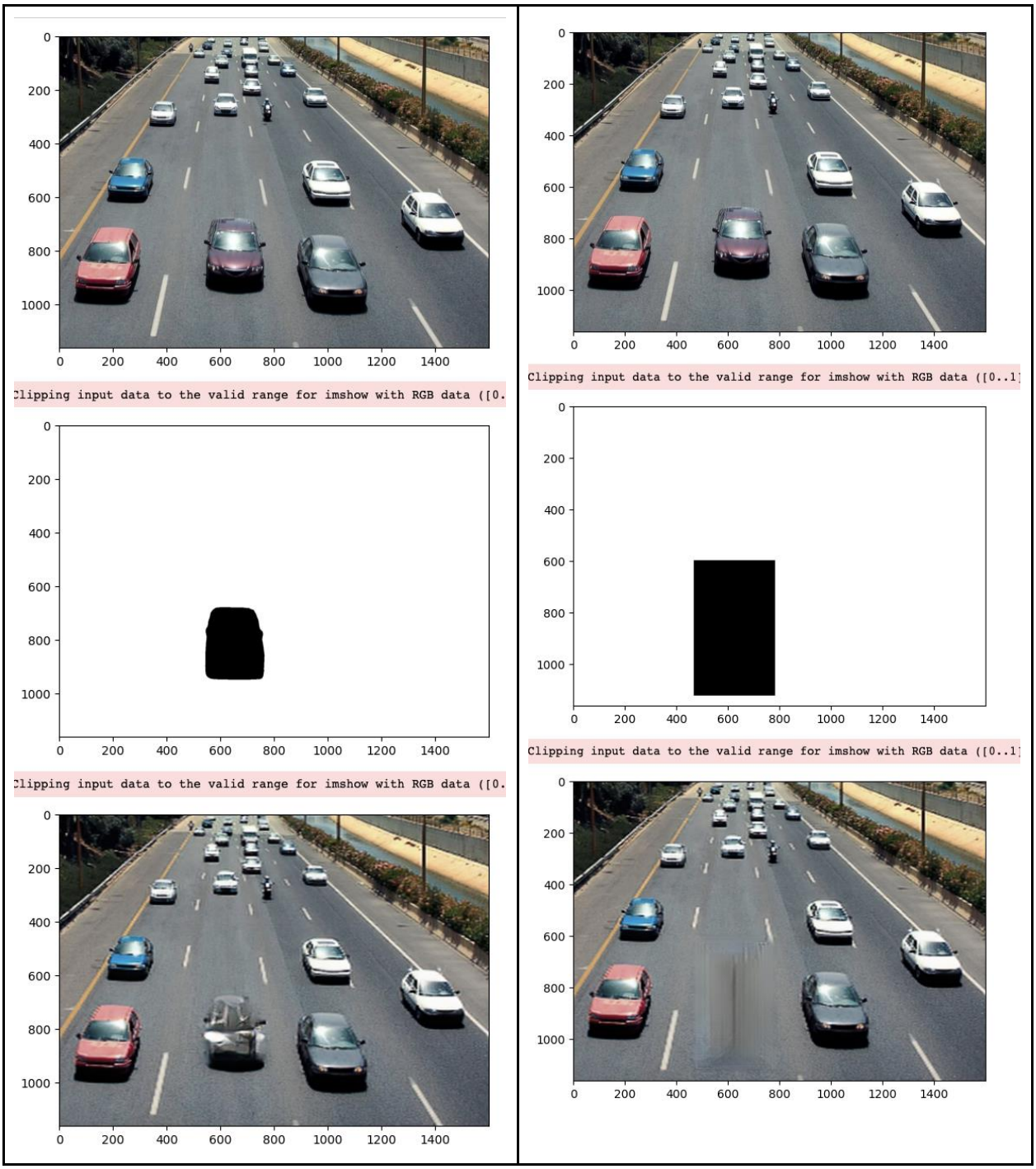
6.2. Image inpainting

The test image, along with the masks generated with image segmentation module is fed to image inpainting module for the final experiment.

Results







The top-left example removes a bench in the back of the scene. We do see a small black artifact, but overall, the bench is quite properly erased. On the other hand in the top-right example where the mask removes a person in the middle of the scene, the resulting patch acts like a ghost of the original person, not fully removing the segment of the person. The same goes for the bottom-left example, where the gray artifact still resembles a car.

We conjecture that this shortcoming is due to the mismatch between the generated masks used during training and validation (see Section 5.3 for examples). Hence, we employ a more primitive masking technique as shown in the bottom right example. We see that the masked object is better hidden.

6.3. Analysis

The performance of the model was satisfactory for the cases with smaller masked objects, however when the mask size was bigger, the inpainting results couldn't fill the entire masked space coherently. This can be attributed to the training performed on the smaller subset of the ImageNet, with approximately using only 10% of the original dataset. This decision was made due to lack of enough computing power and space. Additionally, the mask patterns used for training the in-painting model were more abstract as compared to the masks that we obtained from the instance segmentation For reference, mask used during training is presented in Section 5.3 resembling random pen strokes, but the mask used during inference as shown in Section 6.1 traces the exact shape of the selected object.

7. Conclusion

In conclusion, our image in-painting system has shown promising results for filling in missing objects with fully-covered masks. The results obtained by using more uniform masks were significantly better with an almost perfectly blended image.

However, we witnessed some shortcomings that resulted in non-organic artifacts in the resulting image. As discussed above, we believe that some of the found lacking image quality arose from the lack of data and short-cut training. Hence, the performance of the whole model can be improved further with

- Using the complete ImageNet dataset to train the Inpainting model.
- Fine tuning the model with masks similar to those obtained from the instance segmentation.
- Training the model with a larger batch size and more number of epochs.

8. Source code

The source code for this project can be found here.

References

1. https://towardsdatascience.com/10-papers-you-must-read-for-deep-image-inpainting-2e41c589ced0
2. Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros, "Context Encoders: Feature Learning by Inpainting," *Proc. International Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2016.
3. Ugur Demir, and Gozde Unal, "Patch-Based Image Inpainting with Generative Adversarial Networks," https://arxiv.org/pdf/1803.07422.pdf.
4. Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro, "Image Inpainting for Irregular Holes Using Partial Convolution," *Proc. European Conference on Computer Vision* (*ECCV*), 2018.
5. https://medium.datadriveninvestor.com/inpainting-of-irregular-holes-using-partial-convolution-paper-summary-e836cd2c44ae
6. https://github.com/MathiasGruber/PConv-Keras
7. https://hprc.tamu.edu/