University of Southern California
EE599–Deep Learning Systems (Fall2020)

Realtime Background Replacement and Super Resolution for Video Conferencing Applications

Adityan Jothi
Aditi Hoskere Deepak
Srujana Subramanya

December 10, 2020

Abstract

Background Replacement with Super Resolution enhances video quality in video conferencing Applications i.e., virtual backgrounds. In this project, we implement a deep-learning architecture based on GAN and U-Net for image matting that estimates the foreground and outputs a realistic image and an SRGAN to enhance the resolution of these background replaced images. Following this, we also propose a new architecture from concepts based on the above two implementations, the Background Replacement Super Resolution GAN (BGRSR-GAN) for frames in video conferencing applications.

Contents

Chapter 1

Introduction

In this project, our main goal is to replace the background of an image/video frame and then perform the operation of Super Resolution for the entire image. While the background replacement task can be performed using applications such as Zoom, they tend to fail in cases of poor/excess lighting, camera quality. Our model attempts to overcome these issues and generate realistic looking background replaced super resolution images.

Convolutional Neural Networks have proved to perform exceptionally well on images/videos as it is capable of learning complex functions. Automated Background Replacement and Super Resolution requires sophisticated algorithms like CNNs to handle this operation.

Some of the steps we followed during the implementation and the report structure is as follows:

1. In Section 2, we implemented Instance Segmentation for Background Replacement using the UNET-GAN architecture.
2. In Section 3, Super Resolution using SRGAN is applied to the Background Replaced image/video frame.
3. In Section 4, we propose a new architecture (BGRSRGAN) which performs both Background Replacement and Super Resolution at one shot.
4. In Section 5, we discuss about some of the shortcomings of the model and how it can be improved in the future.

The mentor for our project is Professor Brandon Franzke and code for our project including model implementation can be found here: Code Link

Chapter 2

Instance Segmentation for Background Replacement

2.1 Introduction

In this section, we tackle the problem of background removal through image matting. It mainly revolves around predicting the foreground of an image or a video frame. Unlike other instance segmentation methods[4],matting takes into account the transparency of an object. And as presented in[5], image matting can be formulated as:

$$I_i = a_i * F_i + (1 - a_i)B_i$$

where i indicates which pixel is concerned, α_i is the pixel's matte estimate, F_i is the foreground color, B_i is the background color and I_i is the pixel's color.The is therefore, not only to estimate α_i but also F_i and B_i.

Image matting uses the concept of trimaps, where trimaps represent the mask of the instance to segment the foreground from the background. A trimap typically comprises of three regions. The sure foreground pixels, sure background pixels and the unsure gray pixels. For instance, these unsure pixels can be the result of partially transparent foreground or the mixing of foreground and background colors in the on the foreground's borders.

In this project, we have written a script to extract trimaps from the images provided by the COCO dataset.

2.2 Dataset

The primary dataset used for instance segmentation is the COCO-2014 dataset[6]. The COCO dataset is one of largest datasets used for object recognition, instance segmentation tasks. We limited the dataset to 6660 images considering the limited amount of time and resources available to us. Trimaps are generated using the script we wrote for these images and all images are resized to a standard (320,240) size resolution to perform super resolution later.

For the background images that were used for blending the segmented foreground with new background, we randomly downloaded 10 different background images by crawling the web with search criteria 'background images'. Our script randomly chooses one of the 10 images to replace background with.

2.3 Architecture

Our project utilises two different architectures for the generator network and the discriminator network of the GAN. Our project is implemented using the UNet architecture. The network was suggested by Ronneberger, Fischer and Brox in [7]. They perform image segmentation using a U-shaped fully convolutional architecture. The down slope of the U mainly comprises of the regular convolutions and max-pooling layers. The upward slope comprises of the upsampling and convolutions. Information from the down slope convolutions are sent forward to the up slope in order to pass along more global information the upsampling steps.

From [1], it was seen that U-nets perform extremely well for image matting. And it can be said that a fully convolutional step applied to image matting is a generative model and generates an image of its own. We use a pre-trained Resnet-18 model as our discriminator network. The pre-trained Resnet-18 is used due to its ability to classify images at a high accuracy thereby forcing the generator to learn better samples to fool the discriminator.

2.3.1 Generative Network

Since U-nets are proven to perform well on image segmentation tasks, we implement the generative network in the same fashion as [1]. The network takes in 4 channels as input, the RGB and the trimap as the 4th channel. It also outputs a 4 channel image RGBA, where A accounts for the opacity to represent the foreground in the form the generated image. The U-net architecture is illustrated in Figure 2.1.

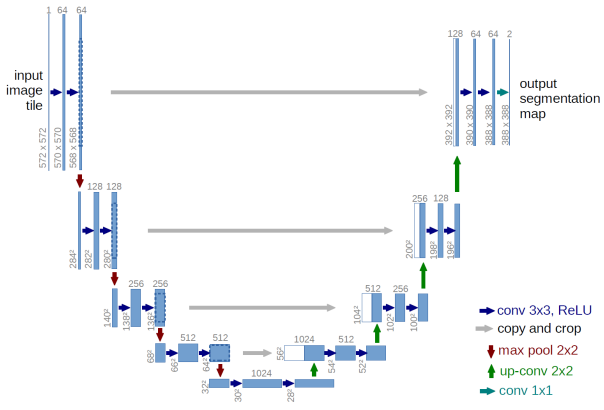

Figure 2.1: Illustration of the U-net architecture used as the generative model.

2.3.2 Discriminative Network

For the discriminator architecture, we used the pre-trained Resnet-18 architecture. The pre-trained Resnet helps in reducing training time and weights of the discriminator architecture.

2.4 Training

The overall architecture of our GAN can be illustrated as shown in Figure 2.3. Let $E(x, M_T)$ denote the generative network; where x represents the input image and M_T represent the trimap associated with it. Both the input image and the trimap are of the same size. Let $D(x)$ represent the discriminative network.

For Generative network, we use the MSE loss function given as:

$$L_G(x, M_T) = ||E(x, M_T)||^2$$

The MSE can be accounted as the foreground reconstruction error.

For the Discriminative Network, we use the Wasserstein Loss, Let $G(z)$ be the output of the Generative Network, then the Wasserstein Loss is given as:

$$L_D(x, G(z)) = D(x) - D(G(z))$$

The discriminator tries to maximize this function. In other words, it tries to maximize the difference between its output on real instances and its output on fake instances.

The overall training presentation of the GAN network is shown in Figure 2.3.

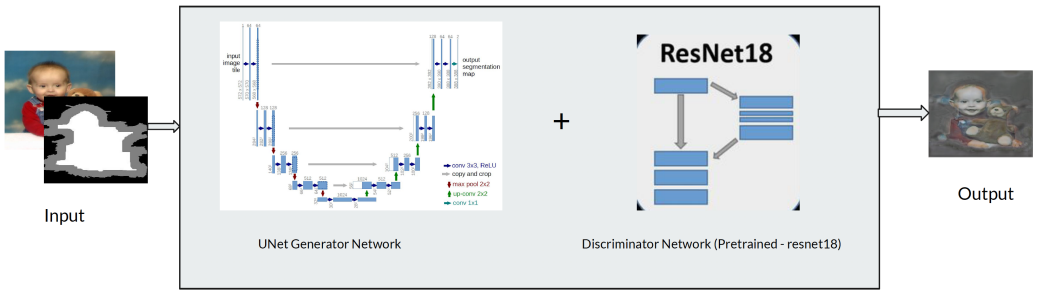

Figure 2.2: Illustration of the overall GAN architecture used for training.

Trainable Paramaters	Forward/Backward Pass (Mb)	Batch Size	Training Time on p3.2xlarge (mins/epoch)
2,89,57,481	1200.33	10	12

Figure 2.3: Model Summary of U-net GAN.

2.5 Results

The results of the U-net GAN are as shown below.

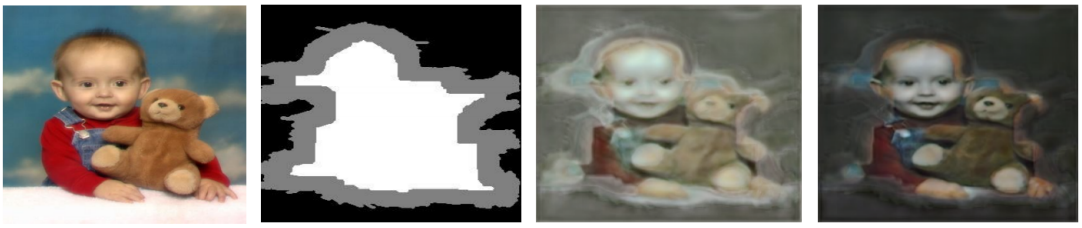

Figure 2.4: *(from left to right:)* Input Image, Trimap associated with input image, Intermediate results of our model after 8 epochs, Final Result of our model after 50 epochs.

Figure 2.5: *(from left to right:)* Input Image, Trimap associated with input image, Intermediate results of our model after 8 epochs, Final Result of our model after 50 epochs.

2.6 Performance Characteristics

The performance characteristics of our model is evaluated on the loss functions that was described earlier. The generative loss, discriminative loss and a total loss of the model are calculated. The performance of our model based on these three loss values was captured through the training epochs.

Figure 2.6: *(from left to right:)* The generative loss, the discriminative loss and the total loss.

2.7 Conclusion

The extracted foreground was blend on a randomly chosen background image using the blend function from the pymatting toolkit. The overall performance of this model was not to the standard expected due to inaccurate pre-processing of the trimap image. We hence, proceeded on to integrate the super resolution GAN and the UNet GAN into one single BGRSRGAN and observed promising results. This will be discussed in further chapters.

Chapter 3

Super Resolution

3.1 Introduction

Super Resolution is the process of recovering High Resolution (HR) image from a Low Resolution (LR) image. To perform Super Resolution, we made use of SRGAN wherein the generator upsamples LR images to Super Resolution images and the discriminator distinguishes the HR images and backpropagates the GAN loss to train the discriminator and the generator networks.

3.2 Dataset

We made use of 9500 samples of Youtube videos from game streamers, podcasters (Video call-esque nature of data), movie audition tapes and converted the video frames from a low resolution of (320,240) to a high resolution of (640,480).

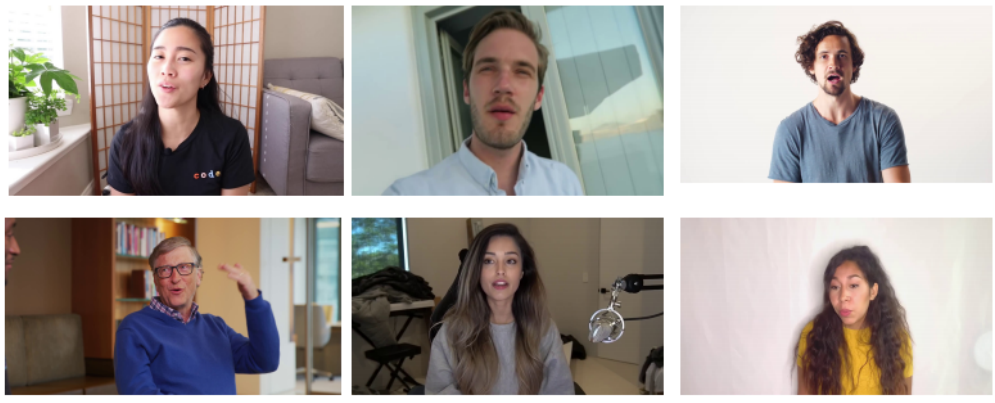

Figure 3.1: Dataset samples

3.3 Architecture

3.3.1 Generator Architecture

Our initial design is inspired by Bee Lim et al.[6]. The SRGAN model architecture that we used consists of six Residual Blocks and an Upsample block. The Residual Block has a Convolution layer, Batch Normalization layer, Depthwise Convolution and Swish activation layer. The Upsample block consists of a Convolution layer, Pixel shuffle layer and Swish activation layer. We have used Depthwise Convolutions to reduce parameter size for decreasing inference time and Swish activation instead of ReLU for better performance. Also, the generator network uses Perceptual Loss and Adversarial Loss and the we utilized Resnet-50 as feature extractor for computing perceptual/content loss.

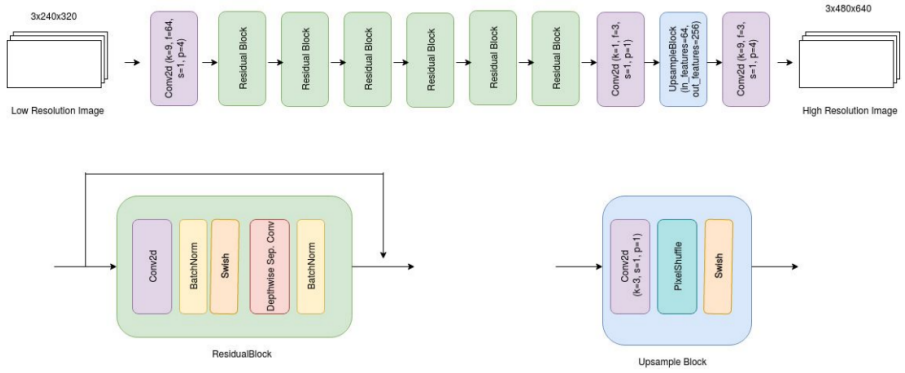

Figure 3.2: Illustration of the SRGAN architecture used as the generative model.

3.3.2 Discriminator Architecture

We have used the pre-trained Resnet-18 for the discriminator. In the case of pre-trained networks, we perform transfer learning and hence, the model reaches convergence earlier compared to training a custom network from scratch.

3.4 Training

In order to train our SRGAN, we performed LR transformation by down-sampling the image to 320x240 and HR transformation by down-sampling to (640x480). The LR image is fed as input to the model and the HR image is used as the ground truth to compute loss against the SR image produced by the generator.

For the Generative Network, we use Content Loss and Adversarial Loss. They can be illustrated as follows:

Content Loss: This loss helps to apprehend the error with the generated and the content image.

$$I = (1/2) * \sum_{(A(g)-A(c))^2}$$

Adversarial Loss: This loss is used in GANs. The generator network generates images so as to fool the discriminator. With each iteration, the generator tries to generate better images and hence, results in a more photo-realistic image.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(x)))] \tag{3.1}$$

For the Discriminator Network, we made use of Binary Cross Entropy (BCE) Loss and that can be illustrated as follows:

$$Hp(q) = -(1/N) * \sum_{i=0}^{i=N} y_i * log(p(y_i)) + (1 - y_i * log(1 - p(y_i)))$$

The combination of the cross-entropy Loss and the sigmoid activation results in a BCE Loss. It is a logarithmic loss function. This loss function is utilized because the output of our model is a single super resolution image.

We can see the overview of the Super Resolution GAN operation as follows:

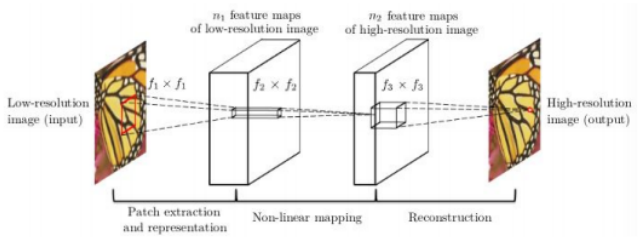

Figure 3.3: SRGAN Overview

The SRGAN model has about 467,843 trainable parameters and took about 26 mins per epoch. This model was comparatively heavy in terms of forward/backward pass size and required more training time.

Trainable Parameters	Forward/Backward pass size (Mb)	Batch Size	Training time on g4dn.4xlarge (mins/epoch)	Best PSNR (dB)
467,843	2144.53	8 (pretrained), 4 (trained)	26.25	61.3

Figure 3.4: SRGAN Train Model Summary

3.5 Results

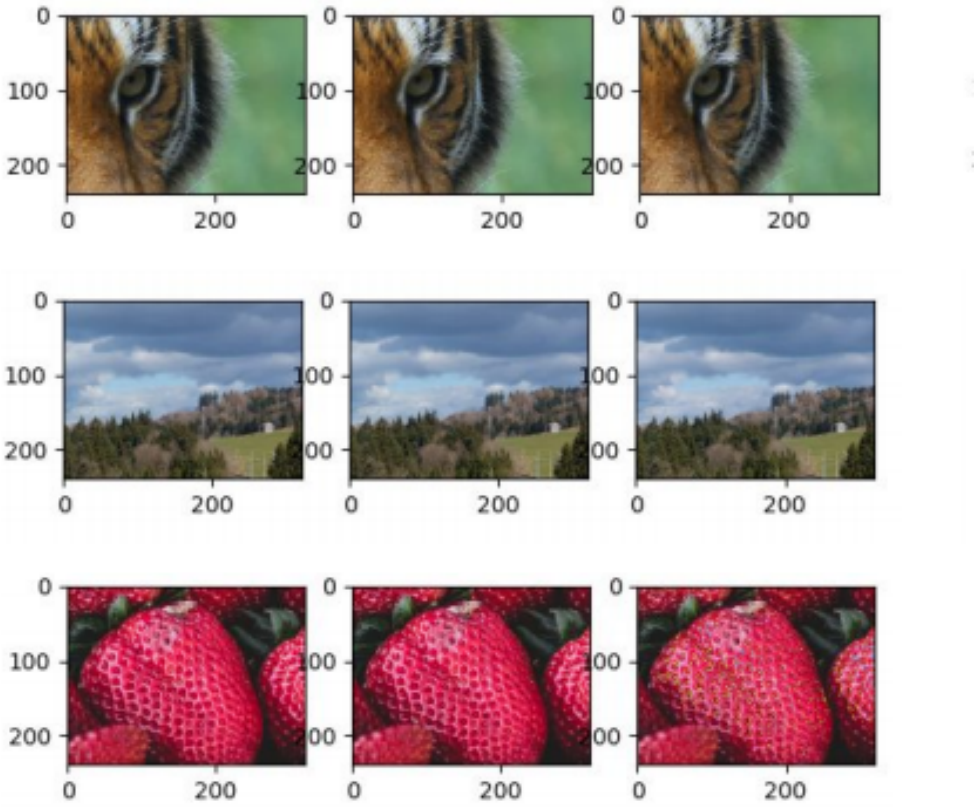

Figure 3.5: Low Resolution

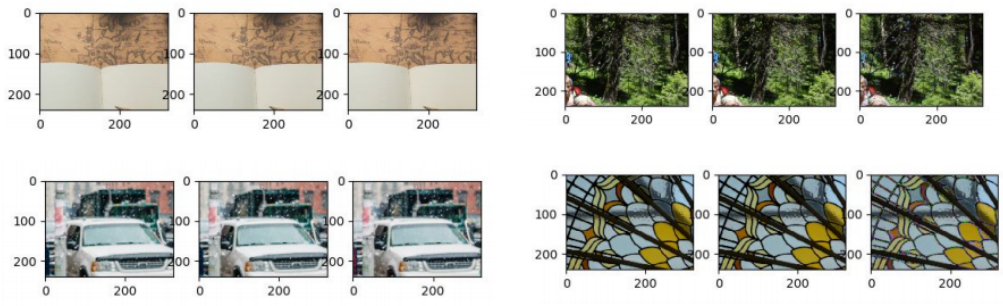

Figure 3.6: High Resolution (Left), Super Resolution (Right)

3.6 Performance Characteristics

The performance of our Generator Network is evaluated based on Generator Content Loss, Adversarial Loss and BCELoss for the Discriminator Network. This is shown below.

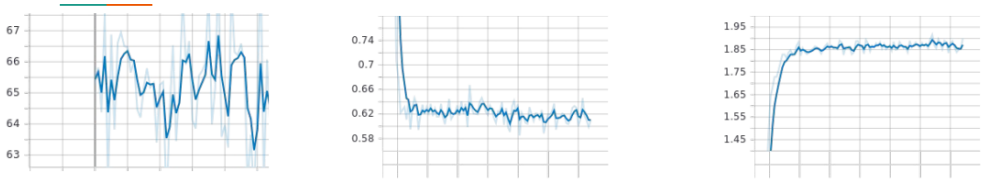

Figure 3.7: PSNR(dB)(Left), Discriminator Loss (Middle), Generator Adversarial Loss (Right)

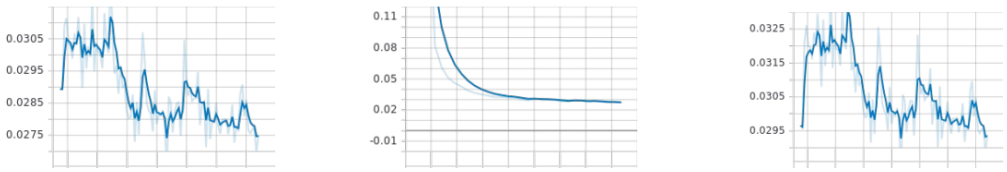

Figure 3.8: Generator Content Loss(Left), Generator Discriminator Loss (Middle), Generator Total Loss (Right)

3.7 Conclusion

The SRGAN was successfully able to transform a low resolution image to a super resolution image. However, the combination of UNet-GAN (used for Background Replacement) and SRGAN gave a Structural Similarity Index value less then 60. Hence, we moved forward to the implementation of the BGRSRGAN in order to achieve a better SSIM value for the two tasks together.

Chapter 4

BGRSRGAN

4.1 Introduction

Based on our experiments in building Background Replacement GANs and Super Resolution GANs, we realized using two separately trained models would not suit our purpose of achieving real-time background replacement and super resolution. As a result, we propose a new architecture built from concepts from background replacement using U-Net and super resolution. The proposed architecture is end-to-end trainable and performs well on low compute edge devices in real-time. The proposed architecture has multi-part loss for guiding the network towards achieving background replacement at the end of our modified U-Net backbone followed by a super resolution loss task at the end of the network.

4.2 Dataset

In order to train our network, we had to come up with techniques to build a ground truth dataset which is similar to video conference calls. The requirements for the dataset were primarily to be of high resolution and easy to swap backgrounds for a primary "speaker(s)" in the video. We choose to use green screen videos which had settings similar to video calls from YouTube and Videezy websites. We then built a background replacement script based on OpenCV to build a ground truth dataset from the extracted frames in the video using chroma keying [8].

4.3 Generator Architecture

The generator modifies the U-net backbone from the background replacement task to use group convolutions with weight standardization. We modified the residual blocks used in super resolution task to be lighter by using grouped convolutions for first layer and depthwise separable convolution for the next convolutional layer. We also modified the activation function from ReLU to Swish in several layers.

4.3.1 Swish Activation

The swish function $y = x * sigmoid(x)$ is a smooth, non-monotonic activation function proposed by Google Brain team in [4]. The swish function has a smooth bump in the range

of -5 < x < 0 and the paper showed that on a training of Resnet-32 a large percentage of the preactivations fall under this domain [4]. We have used these activation functions in building our residual blocks for the super resolution part of the network.

4.3.2 Weight Standardization

We made use of weight standardization in order to accelerate microbatch training as proposed in [5]. We use this in concert with grouped convolutions to reduce the model size.

4.3.3 DoubleConv Block

The DoubleConv blocks comprise of a grouped convolution layer of size 2 with weight normalization followed by batch normalization, ReLU activation and another grouped convolution layer of size 2 with weight normalization followed by batch normalization. We choose to use Swish activation in several layers and ReLU in others.

4.3.4 Down and Up Blocks

The Down block is similar to the U-net's downsampling which comprises of a Maxpooling layer of pool size 2 followed by a DoubleConv block. The Up block performs upsampling by a factor of 2 using Conv2dTranspose followed by a DoubleConv block.

4.3.5 Residual Block

The residual block comprises of a grouped convolutional layer of size 2 with weight normalization followed by batch normalization, swish activation, a depthwise separable convolutional layer and a final batch normalization. The residual block is lighter in comparison to the original SRGAN architecture. We use 3 residual blocks in our architecture in contrast to the 6 residual blocks in the reference architecture and achieve realtime performance with SSIM of close to 73.

4.3.6 Upsample Block

The upsample block from the SRGAN architecture performs upsampling by a factor of 2 and performs pixel shuffling we modified the activation function of this layer to be Swish instead of ReLU for improving performance of the model in the super resolution task.

4.4 Discriminator Architecture

Similar to the previous tasks, we made use of the Resnet-18 model as our discriminator. In order to perform well against a this discriminator we run a few epochs of "pre-training" with generator content loss to bring the generator to produce realistic samples before making it compete with the Resnet-18 model. The Resnet-18 being a more deeper and better performing model than a traditional CNN learns to discriminate samples better and therefore forcing our generator to perform a better task at replacing background and performing super resolution.

4.5 Losses

We made use of several loss functions to perform multiple tasks using the same network. For a given input ((input_img, target_bg, low_res_bg_real, high_res_bg_real)) and given outputs from the generator network G(low_res_bg_fake, high_res_bg_fake) and a discriminator network D. We used the following losses to train the network.

1. Discriminator Loss :
   BCELoss(D(high_res_bg_real), target_real) + BCELoss(D(high_res_bg_fake), target_fake) where target_real is a tensor with range [0.7,1.0) and target_fake is a tensor with range [0,0.3)

2. Generator Content Loss:
   MSELoss(high_res_bg_fake, high_res_bg_real) + 0.006 * MSELoss(hr_bg_fake_ftrs, hr_bg_real_ftrs)
   where hr_bg_fake_ftrs, hr_bg_real_ftrs are the feature vector extracted using a Resnet34 for the high_res_bg_fake and high_res_bg_real images respectively used for perceptual loss.

3. Background Replacement Loss:
   MSELoss(low_res_bg_fake, low_res_bg_real) used to guide the layer corresponding to U-net output to learn a background replacement task.

4. Generator Adversarial Loss:
   BCELoss(D(high_res_bg_fake), target_fake)

4.6 Training

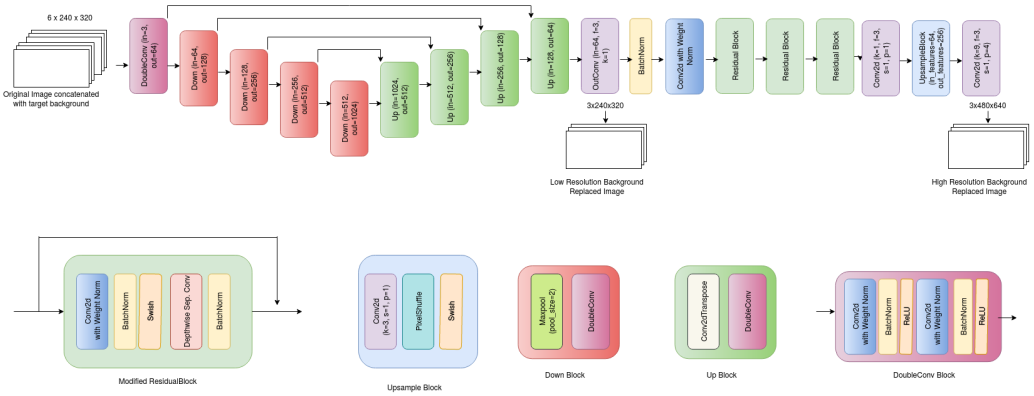

Figure 4.1: BGRSRGAN Architecture

Trainable Parameters	Forward/Backward pass size (Mb)	Batch Size	Training time on g4dn.4xlarge (mins/epoch)	Best PSNR (dB)
17,192,908	2556.45	32 (pretrain), 16(trained)	~4	73

Figure 4.2: BGRSRGAN model summary

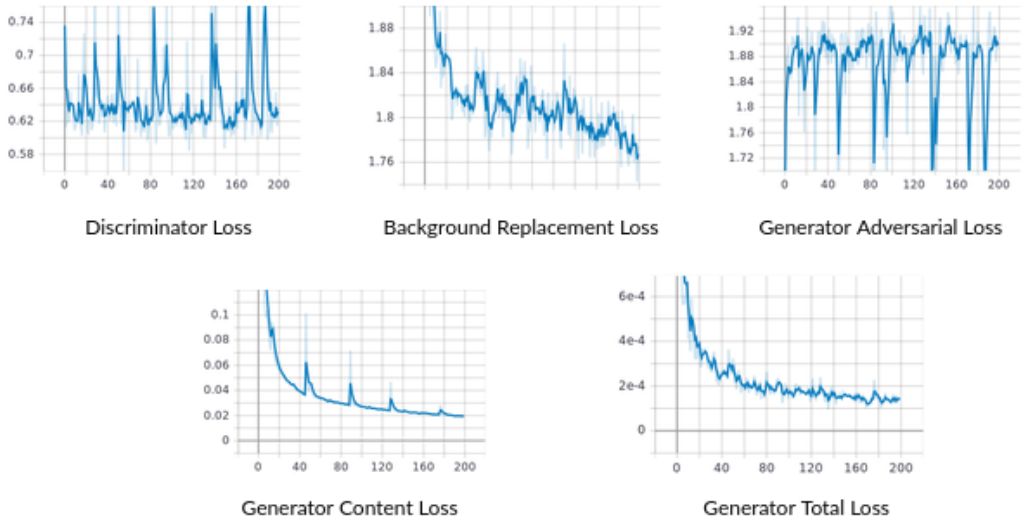

Figure 4.3: BGRSRGAN Initial Training

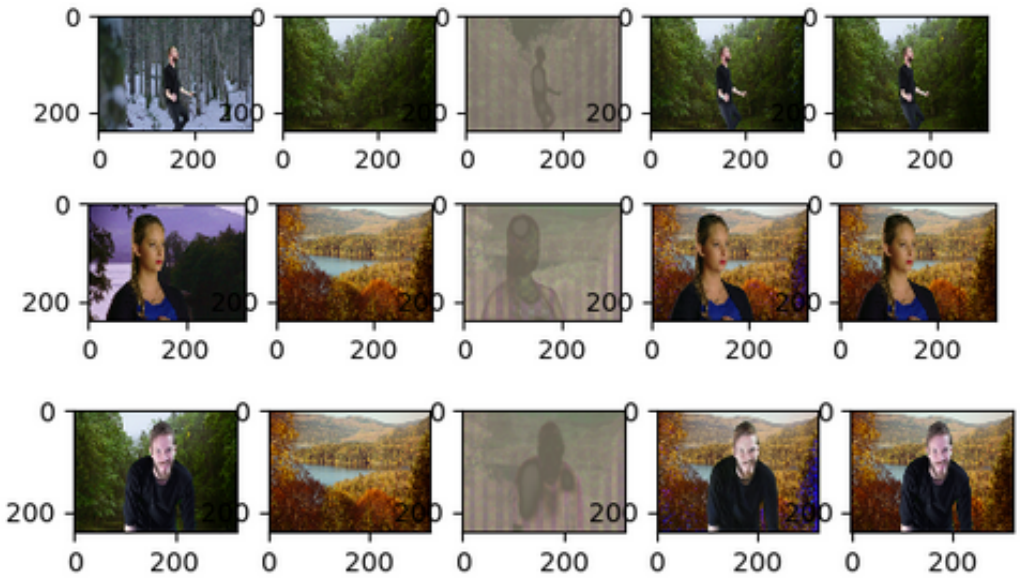

Figure 4.4: BGRSRGAN Initial Results (From Left to Right: Original BG Image, Target BG, LR BG Replacement Layer, Output SR Image, HR Ground Truth)

We trained the BGRSRGAN on a g4dn.4xlarge instance with a batch size of 8 for pre-training phase and batch size of 4 for the training phase due to the large forward/backward pass size. We were able to achieve realistic looking results after 50 epochs of training and were able to achieve an SSIM of 63. The model training took approximately 26 minutes for an epoch. After which we wanted to train with a larger batch size to get a better accuracy and we added more diverse backgrounds to increase generalization capability of the model. The final training was performed with a batch size of 32 for pre-training phase and 16 for the training phase on a g4dn.12xlarge instance. The training time was approximately 4 minutes for each epoch. This brought us to the best SSIM value of 73.

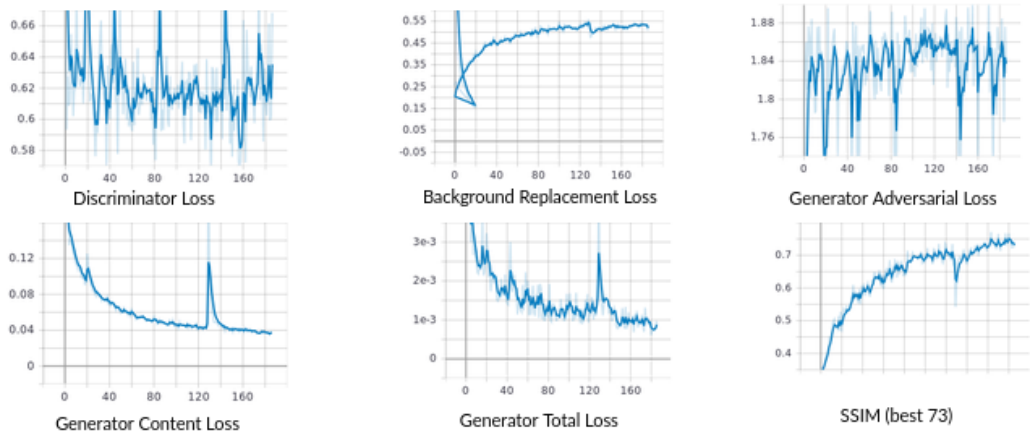

Figure 4.5: BGRSRGAN Final Training

Figure 4.6: BGRSRGAN Final Results (From Left to Right: Original BG Image, Target BG, LR BG Replacement Layer, Output SR Image, HR Ground Truth)

4.7 Real-time Inference

The design options that we choose for the model architecture allowed us to attain a good frame rate of upto 20 fps consistenly on a Jetson Nano device with 4GB of GPU memory, approximately 0.05 seconds per frame. We also tested the model inference time on an EC2 instance with 8GB of GPU memory where it ran at close to 100 fps. With further model compression techniques such as pruning, quantization, lower precision inference, layer fusion. our model would be able to perform even better on low compute edge devices.

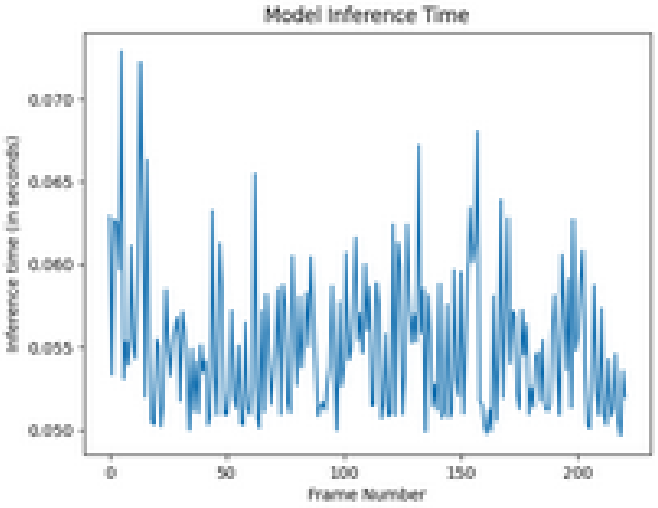

Figure 4.7: Inference Time on Jetson Nano)

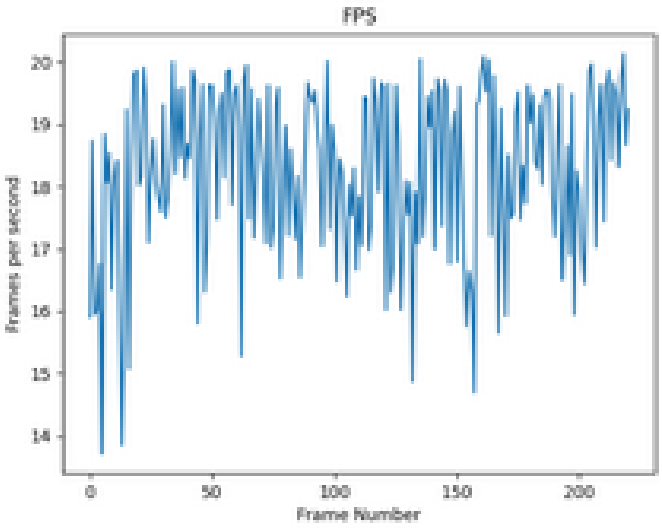

Figure 4.8: FPS on Jetson Nano)

Chapter 5

Conclusion

Our model performed well under single image bgrsr task. However in video acquisition and performing frame-by-frame inference, due to motion blur there were discoloration issues which came up which weren't present in static/single image background replacement + super resolution. This could be resolved by doing some pre-processing and motion blur reduction for frames from videos.

The green screen approach worked well for this particular task but in order to create a more robust model, would need diverse data with high quality labeling for building model with high SSIM and PSNR metrics. Based on the model's ability to perform real-time background replacement and super resolution with a small dataset with an SSIM of 73, with further research on high quality datasets for this task and video related input to the model would be improvements that can be made to make our model more robust and generalize well.

Bibliography

[1] https://github.com/eti-p-doray/unet-gan-matting/blob/master/exploringImageMattingReport.pdf

[2] https://github.com/eti-p-doray/unet-gan-matting

[3] https://pytorch.org/docs/stable/index.html

[4] "Searching for Activation Functions", P. Ramanchandran, B. Zoph, Q. V. Le, 2017

[5] "Micro-Batch Training with Batch-Channel Normalization and Weight Standardization", S. Qiao, H. Wang, C. Liu, et al., 2020

[6] "Enhanced Deep Residual Networks for Single Image Super-Resolution",Bee Lim, Sanghyun Son Heewon Kim, Seungjun Nah, Kyoung Mu Lee,2017

[7] https://www.videezy.com

[8] https://medium.com/fnplus/blue-or-green-screen-effect-with-open-cv-chroma-keying-94d4a6ab2743

[9] https://github.com/joe-siyuan-qiao/WeightStandardization

[10] https://github.com/HasnainRaz/Fast-SRGAN

[11] "ESRGAN: Enhanced Super-Resolution Generative Adversarial networks", X. Wang, K. Yu, S. Wu et al., 2018

[12] "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", C. Ledig, L. Theis, F. Huszar, et al., 2017

[13] "Perceptual Losses for Real-Time Style Transfer and Super-Resolution" J. Johnson, A. Alahi, Fei-Fei Li, 2016

[14] "Towards Real-Time Image Enhancement GANs", L. Galteri, L. Seidenari, M. Bertini, A.D. Bimbo, 2019

[15] "Image Super-Resolution using Deep Convolutional Networks", C. Dong, C. C. Loy, K. He, et al., 2015