Image Enhancement (Low Resolution to High Resolution)

Afifa Tariq

Ryerson University

Toronto, Ontario, Canada

afifa.tariq@ryerson.ca

Abstract

Image enhancement is the technique used to construct a high-resolution image from a low resolution image. This paper aims to provide a review into what are the possible techniques that can be used to construct the high resolution images. I have used Generative Adversarial Networks (GANs) for this task, mainly SRGAN (Super Resolution Generative Adversarial Network) and DCGAN (Deep Convolution Generative Adversarial Network).

I. Introduction

As a classical task in computer vision, image enhancement aims to restore a high-resolution image from a degraded low-resolution one. There are several methods that are possible to achieve this task. I have tried out 2 of these methods which use GANs. Different techniques give us different types of results.

Super-Resolution Generative Adversarial Network, or SRGAN, is a Generative Adversarial Network (GAN) that can generate super-resolution images from low-resolution images, with finer details and higher quality. CNNs were earlier used to produce high-resolution images that train quicker and achieve high-level accuracy. However, in some cases, they are incapable of recovering finer details and often generate blurry images.

SRGAN has three neural networks, a generator, a discriminator, and a pre-trained VGG19 network on the Imagenet dataset. The first step in building a SRGAN is to import the data and preprocess it. I have used the CelebFaces Attributes Dataset (CelebA). We need to train on 2 sets of images, low resolution images and high resolution images because our goal is to go from the low resolution image to the high resolution image. The CelebA dataset contains the high resolution image. By resizing these images, we can get their low resolution images as well.

The generator is fed the low resolution images and is tasked to create high resolution images. The discriminator is fed both the fake generated image from generator and high resolution image from dataset for training. It is then taught to guess which image is fake and which is real and that's how it trains. The GAN loss is sent back to the generator to train it. The generator learns from the discriminators output.

The most important part of the architecture is the generator loss function. Generator loss is actually the sum of the content Loss and the adversarial loss. To calculate the content loss, the generated images and HR images are passed through the pretrained vgg19 (for matching the features) and then the mean loss is calculated. This allows us to recover the photo realistic textures/features which were lost when the image was downsampled. This in turn gives us a result which is close to the reality of a face.

After getting good results from implementing SRGAN, I decided to try another method to enhance images as well which is using Deep Convolution Generative Adversarial Network. DCGANs are usually fed random noise vector and then an image is generated. We need to modify that technique for our system so that the input is the low resolution image instead of the noise vector.

We also need to change the architecture a bit so that instead of the normal DCGAN architecture, we need to add 5 residual layers to it. The loss function for the generator composes of the content loss and the adversarial loss. The content loss is calculated by the difference in the generated image and the original image calculated as l1 norm.

II. Motivation

The high-definition displays in recent years have reached a new level but the need for resolution enhancement cannot be ignored in many applications. For instance, to guarantee the long-term stable operation of the recording devices, as well as the appropriate frame rate for dynamic scenes, digital surveillance products tend to sacrifice resolution to some degree. The current techniques cannot yet satisfy all the demands. Resolution enhancement is therefore still necessary, especially in fields such as video surveillance, medical diagnosis, and remote sensing applications.

Considering the high cost and the limitations of resolution enhancement through "hardware" techniques, especially for large-scale imaging devices, signal processing methods, which are known as super-resolution (SR), have become a potential way to obtain high-resolution (HR) images. With SR methods, we can go beyond the limit of the low-resolution (LR) observations, rather than improving the hardware devices. This is the reason why I have chosen this project to outline the techniques which can be used for this purpose.

III. Prior and Related work

Before neural networks, this problem was dealt with by using bicubic scaling where extra pixels were added to the picture and their values calculated by comparing with the neighbor pixels as shown in figure 1. This did not give out such great results. The resulting image was blurrier and still too hard to see the actual features of the face as shown in figure 2.

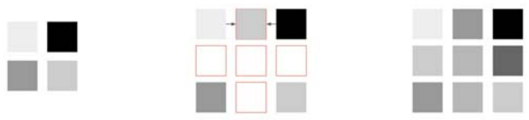


Figure 1

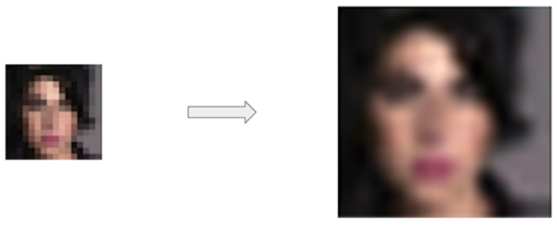


Figure 2

More sophisticated methods than bicubic exploit internal similarities of a given image or, use data-sets of low-resolution images and their high-resolution counterparts to effectively learn a mapping between them. Examples of these are SR algorithms and the Sparse-Coding-Based method.

Before GANs, other deep learning methods had been tried on doing super resolution. SRCNN was the first deep learning method to outperform traditional ones. It is a Convolutional Neural Network consisting of only 3 convolution layers: patch extraction and representation, non-linear mapping and reconstruction. This method gives the result as shown in figure 3. This image is taken from the paper published on SRCNN.

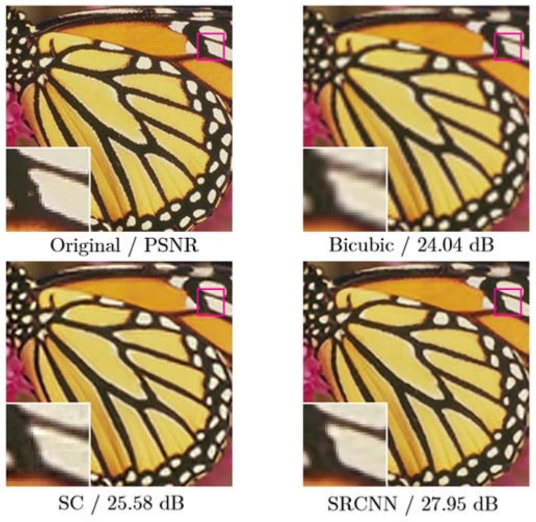


Figure 3

Another deep learning method is the Very Deep Super Resolution (VDSR). It employs a similar structure as SRCNN, but goes deeper to achieve higher accuracy. Both SRCNN and VDSR apply bi-cubic up-sampling at the input stage and deal with the feature maps at the same scale as output.

These deep learning techniques produce high-resolution images that train quicker and achieve high-level accuracy. But still there is one problem which is not solved. That is, how can we recover finer texture details from low resolution image so that the image is not distorted. Recent work has largely focused on minimizing the mean squared reconstruction error. The results have high peak signal-to-noise ratios (PSNR) which means we have good image quality results, but they are often lacking high-frequency details and are perceptually unsatisfying as they are not able to match the fidelity expected in high resolution images. Previous ways try to see similarity in pixel space which led to perceptually unsatisfying results or they produce blurry images. So we need a stable model which can capture the perceptual differences between the model's output and the ground truth image.

IV. SRGAN

Super resolution generative adversarial network applies a deep network in combination with an adversary network to produce higher resolution images. The whole process of the network is that we pass the low resolution image into the generator and get it to convert the image into high resolution. We pass this fake generated image and the real high resolution image to the discriminator and then pass the GAN Loss back to the generator to train it. On each iteration, we tell the discriminator which image is real and which is fake and thus train the discriminator to distinguish between the two.

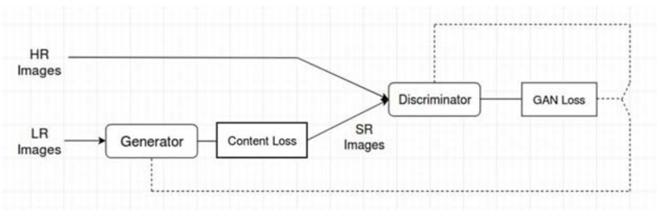


Figure 4

The most important part of the architecture is the generator loss function. Generator loss is actually the sum of the content Loss and the adversarial loss. To calculate the content loss, the generated images and HR images are passed through the pretrained vgg19 (for matching the features) and then the mean loss is calculated. This allows us to recover the photo realistic textures/features which were lost when the image was down sampled. This in turn gives us a result which is close to the reality of a face.

The method for developing Super Resolution Generative Adversarial Network is following:

A. *Dataset and Preprocessing*

The first step in building a SRGAN is to import the data and preprocess it. I am going to be using the CelebFaces Attributes Dataset (CelebA). This is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter. We need to train on 2 sets of images, low resolution images and high resolution images because our goal is to go from the low resolution image to the high resolution image. The CelebA dataset contains the high resolution image. By resizing these images, we can get their low resolution images as well.

High Resolution Image Size – 64
Low Resolution Image Size – 16

For this purpose, I have written a custom dataset class so that it caters to both low resolution and high resolution images. The transform that is applied to images does the following:

1. Resizes them – 16 and 64 respectively
2. Converts them to a tensor
3. Applies normalization with the mean and standard deviation values I got from pre-trained pytorch modules

B. *Network Architecture*

The model for SRGAN consists of Generator and Discriminator. The network architecture is shown in the following figure 5 taken from the SRGAN research paper.

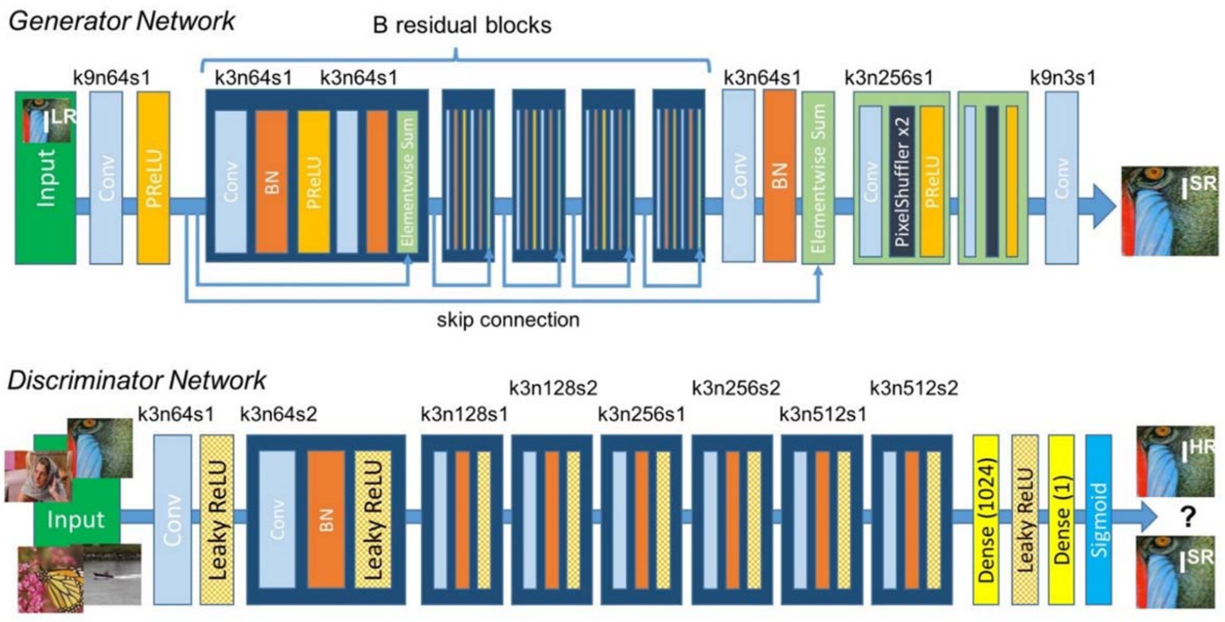


Figure 5

a) Generator:

It mostly composes of convolution layers, batch normalization and parameterized ReLU (PRelU). PRelu is a kind of leakyRelu where instead of a predefined slope of 0.01, it makes it parameter for the neural network to itself decide the value of slope. The generator also implements skip connections similar to ResNet.

The details are as follows:

1. Convolutional Layer:
    a. Filters: 64
    b. Kernel size: 9
    c. Strides: 1
    d. Padding: same
    e. Activation: relu

2. Residual Block:
    a. Convolutional Layer
    b. Batch Normalization
    c. Activation: relu
    d. Convolutional Layer
    e. Batch Normalization

    We add 16 of these residual blocks. The output of the pre-residual block goes to the first residual block. The output of the first residual block goes to the second residual block, and so on, up to the 16th residual block.

3. Convolutional Layer
    a. Convolution Layer:
        i. Filters: 64
        ii. Kernel size: 3
        iii. Strides: 1
        iv. Padding: same
    b. Batch Normalization

4. Add 2 Upsampling Blocks:
    a. Upsampling size: 2
    b. Filers: 256
    c. Kernel size: 3
    d. Strides: 1
    e. Padding: same
    f. Activation: PReLU

5. Ouput Convolutional Layer:
    a. Filters: 3 (equal to number of channels)
    b. Kernel size: 9
    c. Strides: 1
    d. Padding: same
    e. Activation: tanh

b) Discriminator:

The details for the discriminator are as follows:

1. Convolutional Layer:
    a. Filters: 64
    b. Kernel size: 3
    c. Strides: 1
    d. Padding: same
    e. Activation: LeakyReLU with alpha equal to 0.2

2. Another 7 Convolutional Layer:
    a. Filters: 64, 128, 128, 256, 256, 512, 512
    b. Kernel size: 3, 3, 3, 3, 3, 3, 3
    c. Strides: 2, 1, 2, 1, 2, 1, 2
    d. Padding: same for each convolution layer
    e. Activation: LealyReLU with alpha equal to 0.2 for each convolution layer:
3. Dense Layer:
    a. Nodes: 1024
    b. Activation: LeakyReLU with alpha equal to 0.2:

c) Feature Extractor:

We use the VGG pretrained model for feature extraction. We use this for the calculation of generator loss.

C. Loss Functions

a) Generator:

The loss function for the generator composes of the content loss (reconstruction loss) and the adversarial loss.

$$l^{SR} = \underbrace{\underbrace{l^{SR}_{\mathrm{X}}}_{\text{content loss}} + \underbrace{10^{-3} l^{SR}_{Gen}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

The adversarial loss pushes our solution to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolution images and original high resolution images. The Adversarial loss is defined by:

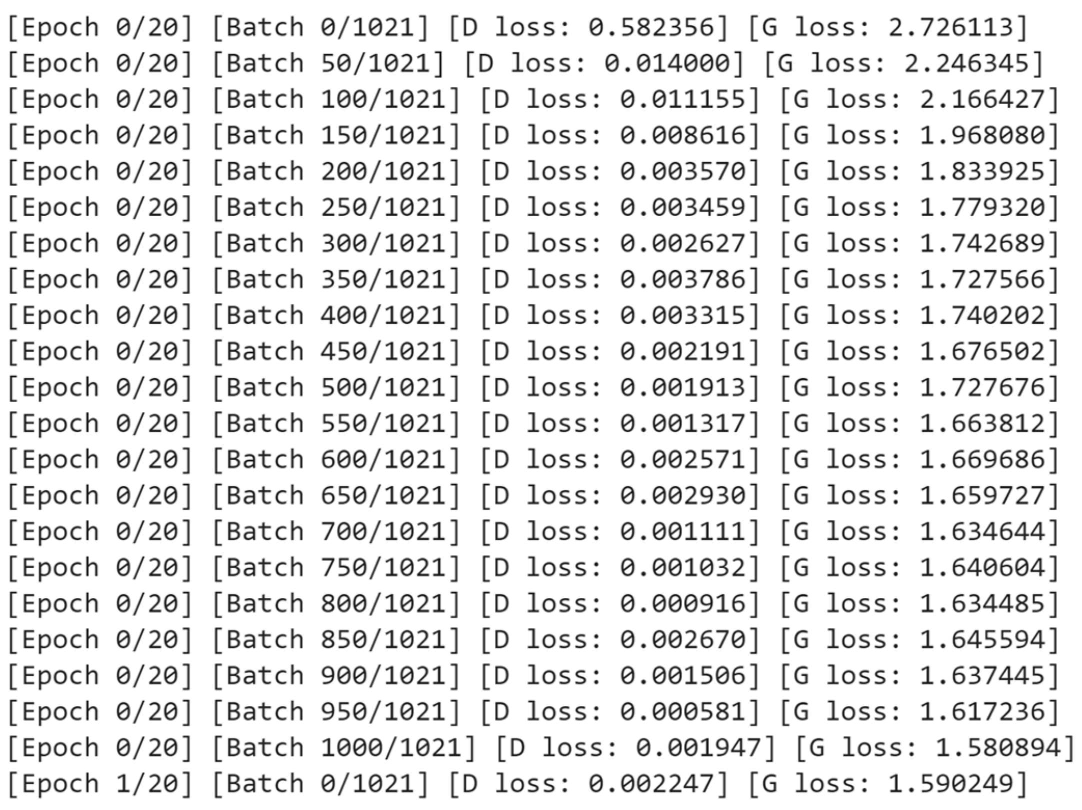
$$l^{SR}_{Gen} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

The content loss helps keep perceptual similarity instead of pixel wise similarity. This will allow us to recover photo-realistic textures from heavily down sampled images.

b) Discriminator:

The loss function for the discriminator is the original GAN loss for discriminator which is as shown in figure.

$$\min_{\theta_G}\max_{\theta_D} \mathbb{E}_{I^{HR}\sim p_{\text{train}}(I^{HR})}[\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))]$$



```
[Epoch 0/20] [Batch 0/1021] [D loss: 0.582356] [G loss: 2.726113]
[Epoch 0/20] [Batch 50/1021] [D loss: 0.014000] [G loss: 2.246345]
[Epoch 0/20] [Batch 100/1021] [D loss: 0.011155] [G loss: 2.166427]
[Epoch 0/20] [Batch 150/1021] [D loss: 0.008616] [G loss: 1.968080]
[Epoch 0/20] [Batch 200/1021] [D loss: 0.003570] [G loss: 1.833925]
[Epoch 0/20] [Batch 250/1021] [D loss: 0.003459] [G loss: 1.779320]
[Epoch 0/20] [Batch 300/1021] [D loss: 0.002627] [G loss: 1.742689]
[Epoch 0/20] [Batch 350/1021] [D loss: 0.003786] [G loss: 1.727566]
[Epoch 0/20] [Batch 400/1021] [D loss: 0.003315] [G loss: 1.740202]
[Epoch 0/20] [Batch 450/1021] [D loss: 0.002191] [G loss: 1.676502]
[Epoch 0/20] [Batch 500/1021] [D loss: 0.001913] [G loss: 1.727676]
[Epoch 0/20] [Batch 550/1021] [D loss: 0.001317] [G loss: 1.663812]
[Epoch 0/20] [Batch 600/1021] [D loss: 0.002571] [G loss: 1.669686]
[Epoch 0/20] [Batch 650/1021] [D loss: 0.002930] [G loss: 1.659727]
[Epoch 0/20] [Batch 700/1021] [D loss: 0.001111] [G loss: 1.634644]
[Epoch 0/20] [Batch 750/1021] [D loss: 0.001032] [G loss: 1.640604]
[Epoch 0/20] [Batch 800/1021] [D loss: 0.000916] [G loss: 1.634485]
[Epoch 0/20] [Batch 850/1021] [D loss: 0.002670] [G loss: 1.645594]
[Epoch 0/20] [Batch 900/1021] [D loss: 0.001506] [G loss: 1.637445]
[Epoch 0/20] [Batch 950/1021] [D loss: 0.000581] [G loss: 1.617236]
[Epoch 0/20] [Batch 1000/1021] [D loss: 0.001947] [G loss: 1.580894]
[Epoch 1/20] [Batch 0/1021] [D loss: 0.002247] [G loss: 1.590249]
```

Figure 6

D. Training

After setting up the neural network, we can start the training process. The figure 6 shows the initial epochs of the network.

The loss curve during training is shown in figure 7.

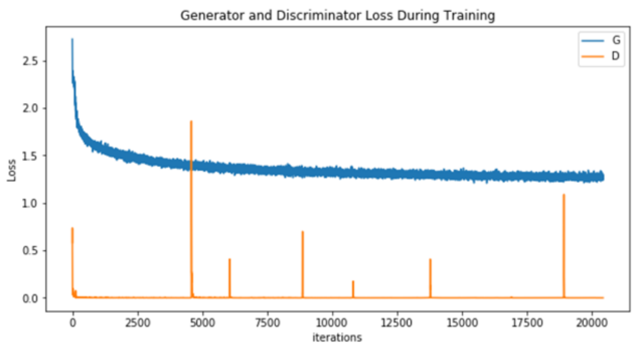


Figure 7

E. Training Results

The results from running this are as shown in figure. The results will get better if I increase the number of epochs and let it train for longer. But because I am using the GPU from google colaboratory, it has to run for a long time to complete these 20 epochs and I have had a lot of problem with the GPU disconnecting from the server and making me start everything again. The disconnection even loses the variables saved.

I trained my model on Google Colaboratory GPU. It took me around 10 hours to do 20 epochs.

As you can see the results get better with more training, the network learns the different features of the faces and makes them more prominent.

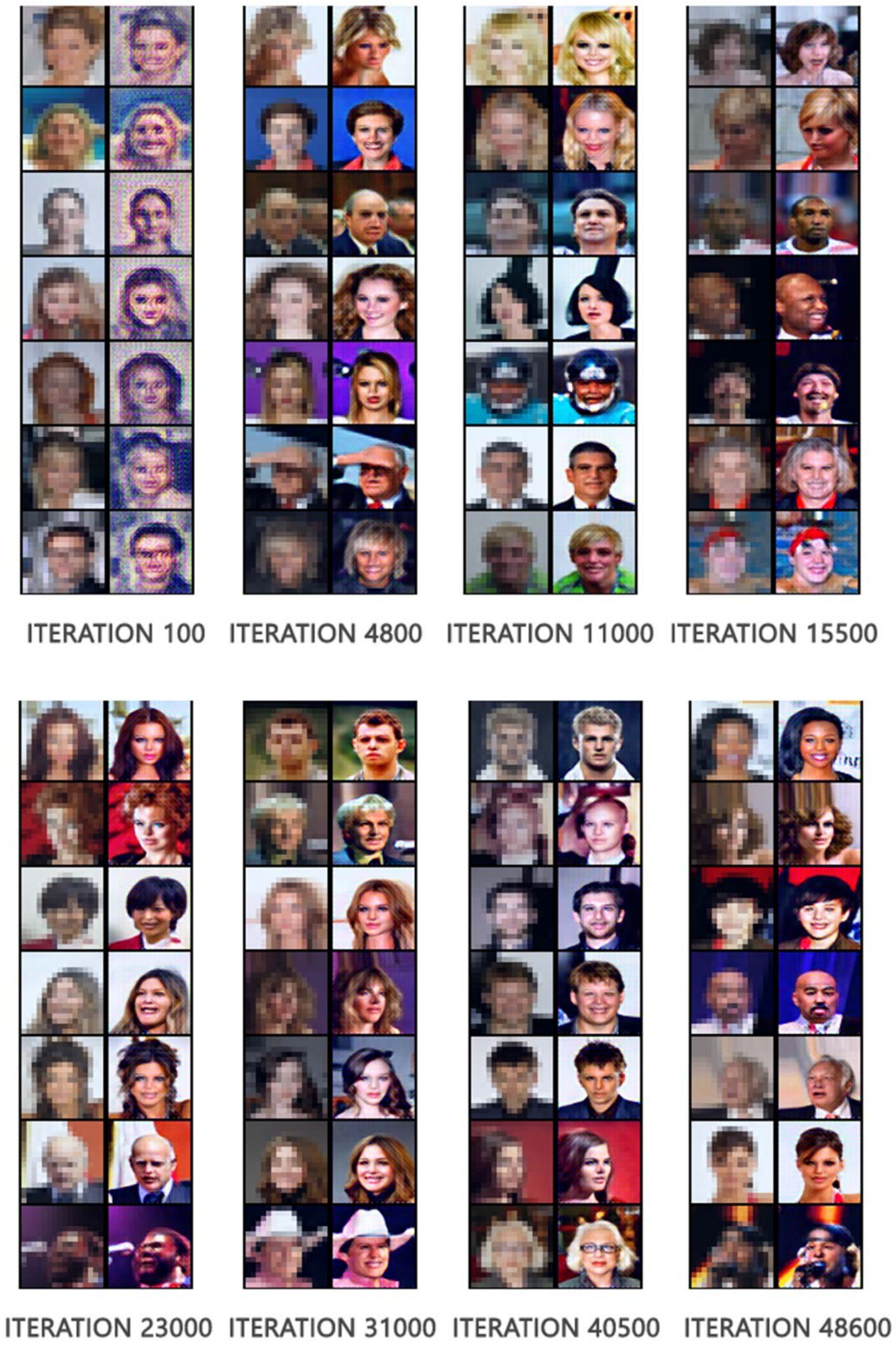

ITERATION 100
ITERATION 4800
ITERATION 11000
ITERATION 15500
ITERATION 23000
ITERATION 31000
ITERATION 40500
ITERATION 48600

Figure 8

F. *Testing my model*

After the training is done, I can now use the generator model to generate high resolution images from the low resolution images I feed into it. The results for the testing are as shown in figures 9, 10, and 11. These results are done after 20 epochs. The results from the iterations shown was up to 50 epochs which is why they were much better than these. Because of the limitation of my GPU, I was unable to reproduce that result.

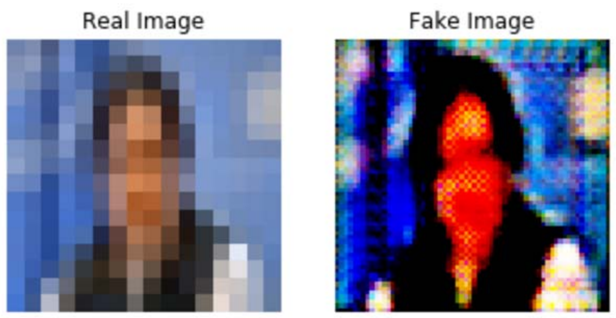


Figure 9

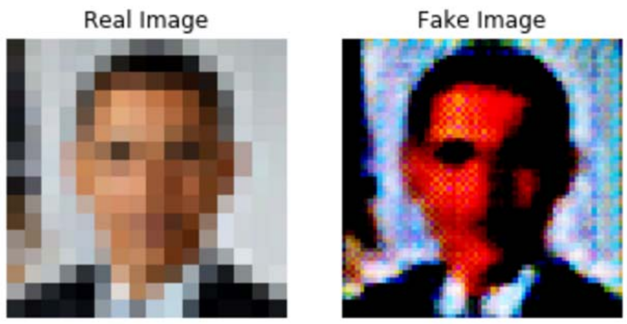


Figure 10

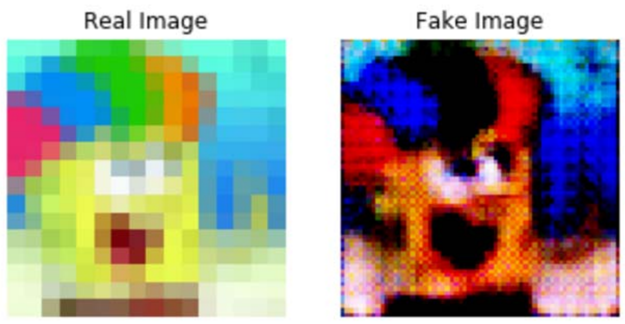


Figure 9

V. DCGAN

The second option I wanted to try for implementing enhancement techniques is to use a Deep Convolution Generative Adversarial Network (DCGAN). DCGAN mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling of images. The DCGAN original model is as shown in figure 12.

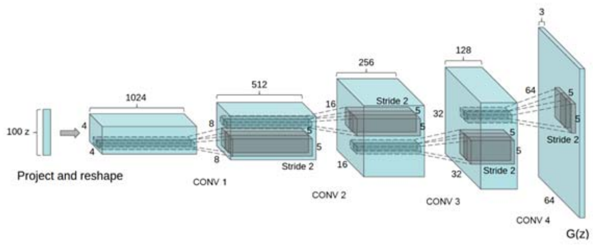


Figure 12

The paper that I followed to develop a DCGAN for image enhancement is this [1]. According to this paper, DCGAN generates more realistic and appealing images. The same architecture can be used to do different image processing tasks such as; image enhancement, image denoising, and image deconvolution.

The idea of DCGAN is to have a generator, which is trained to generate the desired image from downsampled or noisy input, and a discriminator, which is trained to discriminate between the original image and the generated image. The generator and discriminator are adversarial and they are trained together so that after training, generator would be good at generating images that look authentic.

For image enhancement, we feed in low resolution images to the model and the generator generates high resolution images.

A. *Dataset and Preprocessing*

The first step in building a DCGAN is the same as in SRGAN. It is to import the data and preprocess it. I am going to be using the CelebFaces Attributes Dataset (CelebA). This is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter. We need to train on 2 sets of images, low resolution images and high resolution images because our goal is to go from the low resolution image to the high resolution image. The CelebA dataset contains the high resolution image. By resizing these images, we can get their low resolution images as well.

High Resolution Image Size – 64
Low Resolution Image Size – 16

For this purpose, I have written a custom dataset class so that it caters to both low resolution and high resolution images. The transform that is applied to images does the following:

4. Resizes them – 16 and 64 respectively
5. Converts them to a tensor
6. Applies normalization with the mean and standard deviation values I got from pre-trained pytorch modules

B. *Network Architecture*

The model for DCGAN consists of Generator and Discriminator. The network architecture is shown in the following figure 13 taken from the research paper.

a) Generator:

It mostly composes of convolution layers, batch normalization and ReLU. The generator also implements skip connections similar to ResNet. The details are as follows:

1. Convolutional Layer:
   a. Filters: 128
   b. Kernel size: 4
   c. Strides: 1
   d. Padding: 1
   e. Activation: relu

2. Residual Block:
   a. Convolutional Layer
   b. Batch Normalization
   c. Activation: relu
   d. Convolutional Layer
   e. Batch Normalization
   f. Activation: relu
   g. Upsample(factor=2)
   h. Batch Normalization
   i. Activation: Relu
   j. Convolution Transpose Layer

We add 5 of these residual blocks.

3. Convolutional Layer
   a. Conv Layer:
      i. Filters: 128
      ii. Kernel size: 3
      iii. Strides: 1
      iv. Padding: 1
   b. Activation: Relu

4. Convolutional Layer
   a. Conv Layer:
      i. Filters: 64
      ii. Kernel size: 3
      iii. Strides: 1
      iv. Padding: 1
   b. Activation: Relu

5. Ouput Convolutional Layer:
   a. Filters: 3 (equal to number of channels)
   b. Kernel size: 3
   c. Strides: 1
   d. Padding: 1
   e. Activation: Sigmoid

d) Discriminator:

The details for the discriminator are as follows:

1. Convolutional Layer:
   f. Filters: 64
   g. Kernel size: 3
   h. Strides: 1
   i. Padding: 1
   j. Activation: LeakyReLU with alpha equal to 0.2
2. Another 4 Convolutional Layer:
   f. Filters: 64, 128, 256, 512
   g. Kernel size: 3, 3, 3, 3
   h. Strides: 2, 2, 2, 2
   i. Padding: 1
   j. Activation: LeakyReLU
   k. Batch Normalization

3. Convolutional Layer:
   k. Filters: 512
   l. Kernel size: 3
   m. Strides: 1
   n. Padding: 1
   o. Batch Norm
   p. Activation: ReLU
4. Convolutional Layer:
   q. Filters: 512
   r. Kernel size: 3
   s. Strides: 1
   t. Padding: 1
   u. Batch Norm
   v. Activation: ReLU

5. Dense Layer:
   a. Nodes: 1024
   b. Activation: LeakyReLU with alpha equal to 0.2:

6. Dense Layer:
   a. Nodes: 1
   b. Activation: Sigmoid

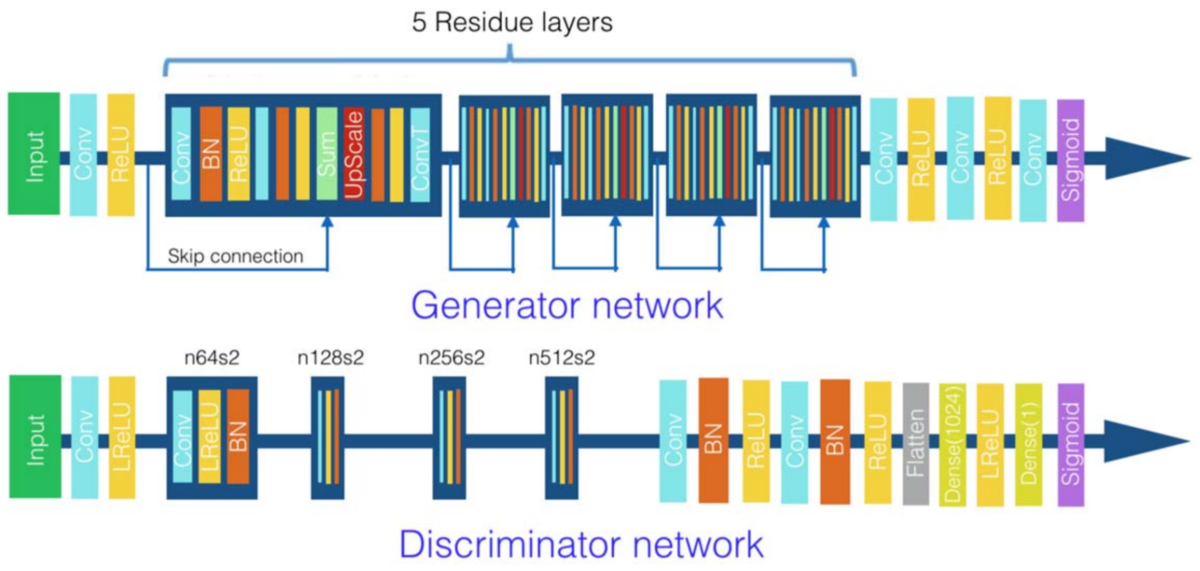


Figure 13

C. *Loss Functions*

a) Generator:

The loss function for the generator composes of the content loss and the adversarial loss.

$$l_G = 0.9 * l_{content} + 0.1 * l_{G,adv}$$

The content loss is calculated by the difference in the generated image and the original image calculated as l1 norm.:

$$l_{content} = \|I^{generated} - I^{original}\|_1$$

The adversarial loss is calculated by:

$$l_{G,adv} = \sum_{n=1}^{N} -\log D(G(I^{input}))$$

b) Discriminator:

The loss function for the discriminator is as shown:

$$l_D = l_{D,adv} = \sum_{n=1}^{N} (\log D(G(I^{input})) + \log(1 - D(I^{original})))$$

D. *Comments*

I have not been able to train this network yet. I am still trying to figure out the correct network layers values. This paper is the only source of information that I have for this network. I haven't been able to find anything related to this DCGAN architecture. This paper itself doesn't give the exact values of the layers as the paper for SRGAN did which is why I am having more difficulty with this.

VI. Comment on Results

We could never 100% reconstruct the original image from the low resolution version because we are making up the image of how it might look like judging from where the pupils of the eyes are, or the contour of the jawline is for example. We could only hope to try to teach a neural network what a face is supposed to look like, and let it imprint that knowledge onto a low res image, thereby 'enhancing' it.

VII. Conclusion

My implementation of SRGAN does a pretty good job of enhancing an image. The resulting image is not blurry and you can make out most of the features correctly. I hope to achieve the same kind of results or even better results with the DCGAN as well.

References

[1] http://stanford.edu/class/ee367/Winter2017/yan_wang_ee367_win17_report.pdf

[2] http://stanford.edu/class/ee367/Winter2017/yan_wang_ee367_win17_report.pdf

[3] https://medium.com/@jonathan_hui/gan-super-resolution-gan-srgan-b471da7270ec

[4] https://arxiv.org/pdf/1609.04802.pdf

[5] https://medium.com/@birla.deepak26/single-image-super-resolution-using-gans-keras-aca310f33112

[6] https://hub.packtpub.com/using-srgans-to-generate-photo-realistic-images-tutorial/

[7] https://github.com/soumith/ganhacks

[8] https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d

[9] https://www.geoffreylitt.com/2017/06/04/enhance-upscaling-images-with-generative-adversarial-neural-networks.html

[10] https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

[11] https://github.com/david-gpu/srez

[12] https://www.kaggle.com/ashishpatel26/gan-beginner-tutorial-for-pytorch-celeba-dataset

[13] https://github.com/bellchenx/DCGAN-with-SE-Residual-Blocks

[14] https://medium.com/@patrickhk/experience-super-resolution-gan-srgan-with-pytorch-f52df6eb06b9