# Super Resolution General Adversarial Network

Mayur Bhise
bhise.m@northeastern.edu

Derek Koechlin
koechlin.d@northeastern.edu

***Abstract*— Recently there has been some advancement in the field of upscaling images to a higher resolution to detect features that were ambiguous earlier, several methods and network architectures were proposed such as PIX2PIX, CYCLEGAN, SRGAN[1], ESRGAN. In these methods, a low resolution (LR) input image is upscaled to the high resolution (HR) space using a variety of deep neural network architectures, which are useful in many applications, such as recovering information from old footages, upscaling video games, images with natively low resolution. SRGAN[1] is a latest innovation in this area, where it uses a perceptual loss function and a pretrained vgg model, to generate far better results than the earlier implementations.**

## Introduction

Shifting and Image from High to Low Resolution is termed as Super resolution. Earlier, it was done using bicubic-interpolation, CNNs, and then GANs were implemented; which caused an increase in the quality of the images produced. GANs consist of two models; a generator which produces images from random noise when trained; and a discriminator which determines whether the generated image is fake or not. The task of upscaling lies with the CNN RESnet. The objective is achieved when there is a 0.5 probability for discriminating between the generated and the original image. The GAN is a type supervised learning algorithm which is trained to be able to successfully generate unseen upscaled images. In our SRGAN model, we used a VGG19 network to assist in the discriminator's loss score. The VGG19 is a network developed by a group at OXFORD where they pretrained an image classification network on the ImageNet dataset. Using the VGG19 network helps to decrease the time it takes to train the discriminator network as it has help from a pre-trained model.

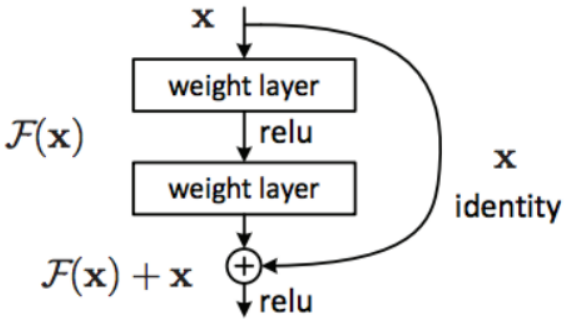
## Architecture

### A. CNNs

Convolutional Neural Networks is a type of neural network in which was developed to tackle image classification, it works on the principle of convolving a kernel on the image thereby generating a number of feature maps according to the complexity which are then passed through various activations which in turn give us accurate results for classification, the terms used in the CNNs are **padding**, which is the addition of extra zeros to the edges of the images to concentrate on the features which lie at the edges. **Stride,** it is how much the kernel slides while convolving on the image, it reduces the dimension of the image for higher values, Pooling, it is used to downsample the image by keeping the dominant features in an image, generally we downsample the image and increase the number of channels in an image which going deeper in the network to be able to detect most of the features in an image, which is proved to be beneficial.

### B. Batch Normalization & Skip Connections

It is an operation performed after a convolution operation to avoid the covariate shift much like the scaling we perform in Machine Learning, i.e. min-max scaler, Standardization, to keep the weights in a limit to avoid exploding and vanishing gradients, With batch normalization each element of a layer in a neural network is normalized to zero mean and unit variance, based on its statistics within a mini-batch As the networks get deeper, the model starts saturation which results in high training error, overfitting is not the cause for this, but the topic is highly in research, residual blocks with Skip connections tend to alleviate this problem to some extent by adding a previous block by element wise sum.




## II. Loss Functions

The loss function for a standard general adversarial network proposed by Ian Goodfellow et al. is as follows

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

It is termed as a value function which is a min-max game in which the generator tries to fool the discriminator in passing the fake image as real. The value function first tries to solve the maximizing problem on the discriminator and then minimizing with respect to generator with global optimum at

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right)$$

**While** not converged **do**

1. **For** k steps **do**
   1.1 Draw $B$ training samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_B\}$ from $p_{data}(\mathbf{x})$
   1.2 Draw $B$ latent samples $\{\mathbf{z}_1, \ldots, \mathbf{z}_B\}$ from $p(\mathbf{z})$
   1.3 Update the **discriminator** $D$ by **ascending** its stochastic gradier

$$\nabla_{\mathbf{w}_D} \frac{1}{B} \sum_{b=1}^{B} \log D(\mathbf{x}_b) + \log(1 - D(G(\mathbf{z}_b)))$$

2. Draw $B$ latent samples $\{\mathbf{z}_1, \ldots, \mathbf{z}_B\}$ from $p(\mathbf{z})$
3. Update the **generator** $G$ by **descending** its stochastic gradient:

$$\nabla_{\mathbf{w}_G} \frac{1}{B} \sum_{b=1}^{B} \log(1 - D(G(\mathbf{z}_b)))$$

### A. Perceptual Loss Function

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

Here the content loss is,

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

which is adopted from Gatys et al. Bruna et al. and Johnson et al. for achieving perceptual similarity which is not so focused by the traditional MSE loss used in earlier networks.

The generative loss lSR Gen is defined based on the probabilities of the discriminator over all training samples as:

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

Here the log term is the discrimination probability, whether the reconstructed image is a fake or real, here we minimize the log function as -log(disc) instead of log(1- disc) to avoid the problem of mode collapse.

This surprisingly simple idea just combines the content loss (VGG) with the appropriately weighted adversarial loss at a ratio of 1000:1. This is enough to encourage the generator to find solutions that lie within the PDF of natural images without overly conditioning the network to reproduce rather than generate.

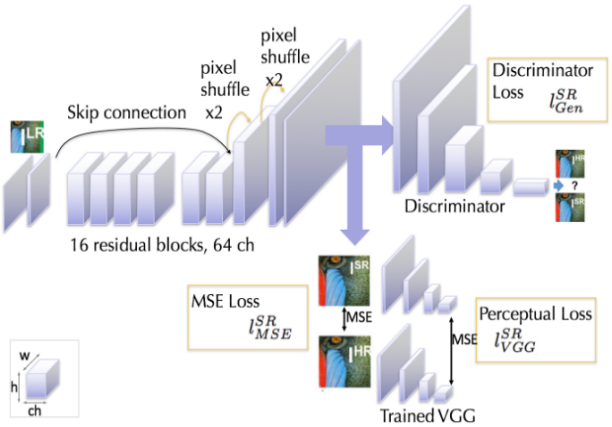
While it's important to reproduce is the correct pixels, learning this representation through MSE lacks context, the idea of using the VGG network is that it has an excellent 'feel' for features in general and the spatial relation of pixels to each other carries more weight, so by comparing the latent space of feature representations at each layer of the VGG network both high and low level features are encouraged in a realistic way and can guide the style of the generated image.

### B. *Contribution of the SRGAN authors*

- A new state of the art for image SR with high upscaling factors (4×) as measured by PSNR and structural similarity (SSIM) with 16 blocks deep ResNet (SRResNet) optimized for MSE.
- **SRGAN which is a GAN-based** network optimized for a **new perceptual loss**. Here MSE-based content loss is replaced with a loss calculated on **feature maps of the VGG network**, which are more invariant to changes in pixel space.
- Evaluation with an extensive **mean opinion score (MOS)** test on images from three public benchmark datasets to confirm that SRGAN is the new state of the art, by a large margin, for the estimation of photo-realistic SR images with high upscaling factors (4×)

[1], https://medium.com/@ramyahrgowda/srgan-paper-explained-3d2d575d09ff

### C. *Our Implementation*




The Generator model consists of four stages. The first stage is the input layer which uses a 2D convolution and a parametric rectified linear unit (PReLU) activation function. The purpose of this input layer is to ingest the image which has a feature space of size 3 for the color channels red, green and blue channels; and produce a convolution feature space of size 64 using a 9-pixel x 9-pixel kernel. The PRelu follows the first 2D convolution to enforce non-linearity to the feature space before passing it to the second stage. It is important to note that the ReLU is not used here because it can cause neurons to die out through training.

The second stage is comprised of a recurrent residual block. From our research, it is recommended to repeat this residual block at least 15 times in order to learn the feature space of the image. This residual block consists of 6 methods starting with an 2D convolution maintaining the feature size of 64; however, the kernel size is the root of the input stage convolution equating to 3-pixel x 3-pixel. By using a smaller kernel size, the residual block is able to maintain many of the features of the original image without

overfitting. The next layer is a batch normalization which, as mentioned previously normalizes the feature data to a zero mean with a unit variance. A PRelu activation follows before the block repeats the 2D convolution and batch normalization. The last layer in the residual block is important as it involves a short skip connection. This layer is an element-wise sum which, as the layer name suggests, sums up the elements from the second batch normalization layer (layer 4) and the skip connection elements. The skip connection is imperative in this deep convolutional network to stabilize the gradient updates in this deep network [3]. Each of the 15 residual blocks use a skip connection using spatial information from the previous residual block.

The third stage is made up of only a few layers but incorporates an important skip connection. This stage performs an additional 2D convolution and batch normalization using the same hyperparameters from the residual block. The following layer is an element-wise sum using a skip connection from the input layer. This skip connection specifically is important as it is a long skip connection. Long skip connections pass features acquired from previous layers and recover information that may have been lost in the convolutions.

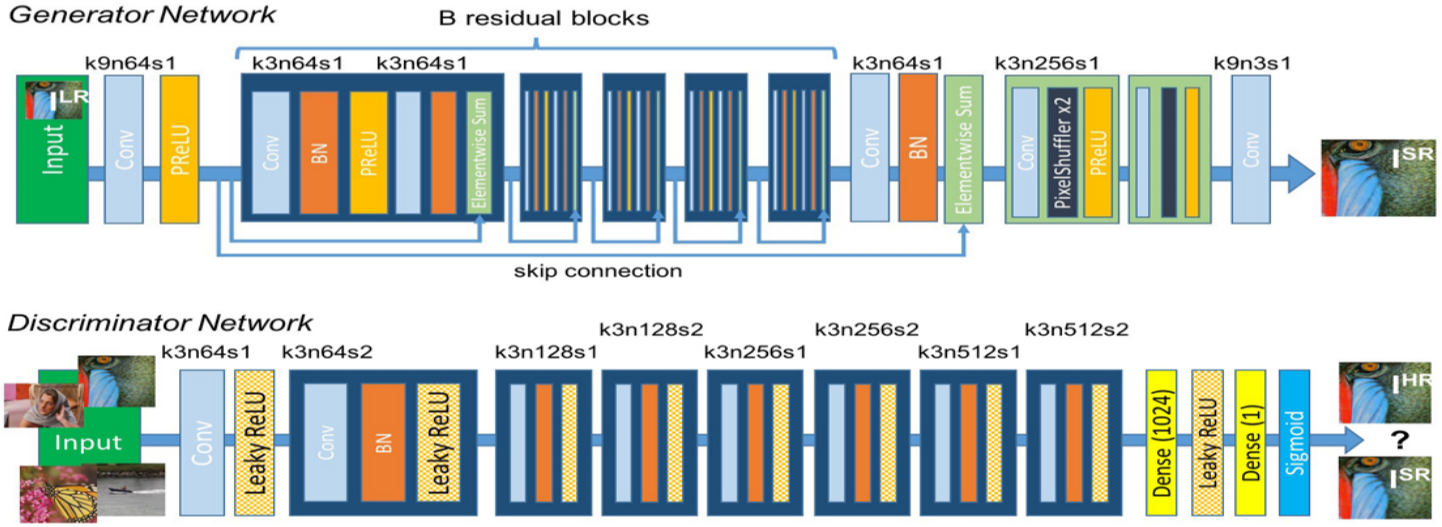
The last stage is known as the up-sample block. Our first layer in the up-sample block is a 2D convolution where we use the same stride and kernel sizes from the previous two stages, but increase the feature size to 256. After this convolution is performed, the data is passed through a pixel shuffler layer of size 2. The pixel shuffle layer rearranges elements in a tensor of shape (*,C×r2,H,W) to a tensor of shape (*,C,H×r,W×r), where r is an upscale factor [4]. Our size two is reference to the "r" meaning we are upscaling the image by a factor of two. The result from this is passed through a PReLU non-linear activation function. The 2D convolution, pixel shuffler and PReLU layers are repeated equating to a total upscale by a factor of 4. Lastly, the data is passed through a 2D convolution using a feature space of size 3, returning the image to the RGB color space.

The Discriminator follows a similar structure in its architecture. The network has an input layer followed by a deep convolutional neural network and ending with a dense layer. The input layer for the discriminator is virtually the same as the generator; however, it swaps out the PReLU activation function for a leaky ReLU (LReLU) activation function. The LReLU is slightly different from the PReLU which learns the coefficient alpha value during training, and instead hard sets this coefficient to .01. The second stage as stated earlier involves a deep convolutional neural network. Similar to the generator network, this stage starts with a convolution taking the input image with a feature space of 3 for RGB, to 64. Following this convolution is a batch normalization. Lastly, a LReLU activation function is applied to the network. A series of 6 more convolution blocks are used after the first one, doubling the feature space by a factor of 2 every other convolution block. The final convolution block contains a feature space of 512. It is important to note that a stride of 2 is also used in every other convolution block to smooth out the feature space before doubling in size.

The dense layer follows last which starts by linearly transforming the feature space to a size of 1024 from 512. Once the feature space is at the final size of 1024, a LReLU is applied to the feature space before the final linear activation function is applied. The last linear activation takes the 1024 x (16*16*512) down to 1024 x 1. This is a crucial part in the discriminator network because the data can be passed into the sigmoid activation function where a single prediction can be made on the entire feature space. Without the final Linear layer, the model would make 16*16*512 predictions which is not what we are looking for.

### Hyperparameters & Approach

Hyperparameter selection plays a significant role in the quality of the images generated by the model. Hyperparameters fall into two main categories, model parameters and training parameters. Most of the model hyperparameters can be seen in the image below. These



include feature size for convolutions, strides, kernel sizes and residual block size. Throughout our research, we found that we would have the best success using the model hyperparameters recommended by our sources. Training parameters were something we were able to pick and we used a couple different setups when training. These parameters include input image size, number of epochs, batch size and learning rate. The two that we experimented with were the number of epochs and batch size. Our first test training, we trained the SRGAN using an input image size of 64x64 pixels, 4000 epochs and a batch size of 1. With a smaller batch size the time per each epoch is smaller than a larger batch size. Our second round of training from scratch we used 750 epochs with a batch size of 8 images. It's important to note that we were interested in using a larger batch size but we hit the hardware limit of our graphics card which only had 10GB DRAM. After training completed, we generated 4-times upscaled images and compared them to the original image, the results are captured in the next section.

## Results

We achieved our results using the following hardware specs: 24 Core CPU, 256GB RAM, Nvidia 3080 GPU. Using the two hyperparameters setups mentioned in the previous section we were able to produce the following two images. The original low resolution input image is shown on the right. This image is a 128 x 128 pixel image with a gaussian blur applied. On the right we have our generated images from our two models. The first generated image shown in the middle is a 512 x 512 pixel image generated by our model that ran with 750 epochs and batch size of 8. The second generated image shown on the bottom far right, used the hyperparameters of 4000 epochs with a batch size of 1.

While our results rely mostly on objective reasoning, we believe that both of our implementations were able to accomplish our goal successfully. We were able to produce images that had 4 times the pixel count as the input image while also restoring resolution to be comparable to the original image. Some of the observations we notice with the first generated image in the middle is that the model was able to smooth out edges and make them sharper as seen in the shoulder, nose and hat regions. One thing this model missed was replicating the same color space. As seen in the picture, the color appears dull with less contrast. Our second model on the bottom right performed better on this image and was able to keep the same color space as well as produce a sharper image. Some of the additional sharpness can be seen in the background of the image where there are shadows on the wall.

| METRICS | SR-ResNet-L2 | SRGAN (authors) | SRGAN (ours) |
|---|---|---|---|
| **MSE** | 280 | N/A | 68.16 |
| **PSNR** | 21.97 | 29.4 | 29.75 |
| **SSIM** | 0.47 | 0.84 | 0.72 |

## References


[1] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi (Twitter) - Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, https://arxiv.org/abs/1609.04802v5

[2] Ian J. Goodfellow, Jean Pouget-Abadie*, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio D´epartement d'informatique et de recherche opérationnelle Université de Montréal arXiv:1406.2661v1 [stat.ML] 10 Jun 2014 Montreal, QC H3C 3J7, https://arxiv.org/abs/1406.2661

[3] https://theaisummer.com/skip-connections/

[4] https://pytorch.org/docs/stable/generated/torch.nn.PixelShuffle.html?highlight=pixel%20shuffle#torch.nn.PixelShuffle

[5] https://arxiv.org/abs/1609.07009

[6] https://towardsdatascience.com/srgan-a-tensorflow-implementation-49b959267c60

[7] https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/autonomous-vision/lectures/deep-learning/

[8] https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[9] https://medium.com/@hirotoschwert/introduction-to-deep-super-resolution-c052d84ce8cf