Super Resolution/Image DeBlurring using GAN

Rachit Shah (rshah25) Sourabh Sandanshi (ssandan) Swaraj Meshram (smeshram)

I. Introduction

Super Resolution finds application in various domains ranging from photography, image compression, Astronomy/Satellite and medical image processing. Manually upscaling images via photo-editing software is a tedious task but still may result in inaccurate results. It also finds commercial importance in image compression where rather than storing high quality images, we can save space via storing compressed images which can later be upscaled. Through image super resolution, we denoise, deblur and upscale lower resolution images to generate finer details and make higher resolution clean images.

II. Dataset

DIV2K [2] dataset consists of 2K high resolution RGB images with a large diversity of contents. It was used for the NTIRE 2017 and PIRM 2018 competitions. It provides various types of lower resolution sets with various downgrading operations applied to the high resolution images. We use the lower resolution set with an unknown degradation applied. This degradation consists of gaussian blur, random noise, and many more which are kept unknown to the users for the sake of competition. The lower resolution are downscaled 4 times from the high resolution images. The training set consists of 800 images while the validation set consists of 100 images.

III. Methodology

A. Preprocessing

We cache and store our training and validation images using Tensorflow Dataset. It acts as a generator for our model and follows the logical plan or preprocessing steps that we predefine and executes it at runtime so that the dataset doesn't take too much memory. We follow the following preprocessing steps:

- **Random Crop** - Super Resolution of an entire low res image of 512x512 to 2048x2048 super res image will make our model too big with too many parameters to train. Instead we randomly crop parts of the image into 96x96 blocks such that the entire image is now a combination of these individually super resolved images. This allows us to reduce our number of parameters by a great extent.
- **Random Flip** - We do a horizontal flip to the images randomly to mirror the images with a 50% chance.
- **Random Rotate** - We generate a number between 0-4 randomly to rotate the image randomly by 0, 90, 180 or 270 degrees.

These preprocessing steps help to save memory, augment data and help to create a more generalized model.

```
Model: "simple_cnn"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
normalize (Lambda)           (None, None, None, 3)     0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, None, None, 64)    1792
_________________________________________________________________
conv2d_9 (Conv2D)            (None, None, None, 64)    36928
_________________________________________________________________
conv2d_10 (Conv2D)           (None, None, None, 256)   147712
_________________________________________________________________
lambda_3 (Lambda)            (None, None, None, 64)    0
_________________________________________________________________
activation_2 (Activation)    (None, None, None, 64)    0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, None, None, 256)   147712
_________________________________________________________________
lambda_4 (Lambda)            (None, None, None, 64)    0
_________________________________________________________________
activation_3 (Activation)    (None, None, None, 64)    0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, None, None, 3)     1731
_________________________________________________________________
denormalize (Lambda)         (None, None, None, 3)     0
=================================================================
Total params: 335,875
Trainable params: 335,875
Non-trainable params: 0
_________________________________________________________________
```

Fig. 1. CNN Architecture

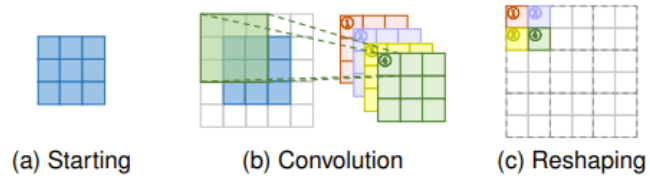

Fig. 2. Upsampling - Sub-Pixel Convolution

B. Models

*1) **Convolutional Neural Network**:* We make a simple baseline for this project using a standard CNN architecture along with some upsampling blocks. The architecture for this can be seen in figure 1. The model consists of the following major parts:

- **Standardize and Destandardize** - We get the RGB mean of the entire dataset. We then standardize image input to the model using formula $(X - RGB\_MEAN)/127.5$. At the end of the model we undo this standardization to get our super resolved image using the formula $X * 127.5 + RGB\_MEAN$. This helps to center the data and help make the model learn faster as the gradients will act uniformly for each channel.
- **Convolution Layers** - We have two convolution layers in the middle with 64 number of filters of kernel size

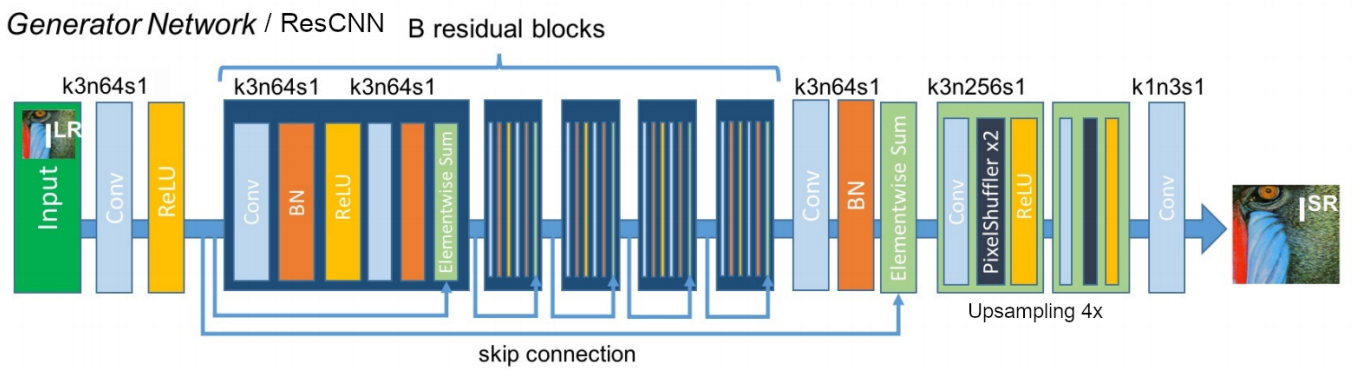

Fig. 3. ResCNN or Generator for GAN Architecture

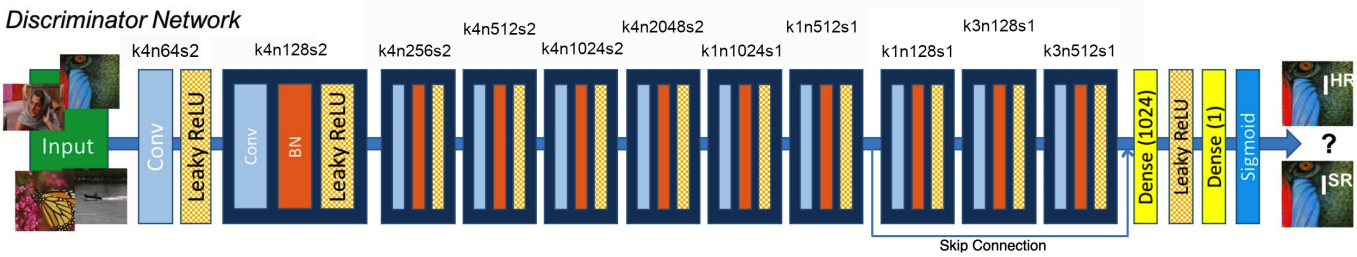

Fig. 4. Discriminator for GAN Architecture

3x3. We also have one final convolution at the end with 3 filters of kernel size 3x3.

- **Upsampling Blocks** - As the name suggests, this layer helps to upsample our feature space using Sub-Pixel Convolution [6]. Suppose the input has shape $(Height, Width, Channels)$ and we want to upsample it by a factor s, sub-pixel convolution first does a convolution to make the features $(Height, Width, s^2Channels)$ shape and then reshape it to $(sHeight, sWidth, Channels)$, essentially upscaling it by s. We do this 2 times to upscale it 4 times since our low res image was downscaled 4 time. The output of this layer would be the super resolved image. Sub-Pixel Convolution is visualized in figure 2.

*2) **Residual Convolutional Neural Network**:* We also create another model for CNN called Residual Convolutional Neural Network. It is a deep CNN model which makes use of skip connections to solve the problem of vanishing gradients in very deep neural networks. Skip connections add the output of a previous layer to the output of a layer much later. As you can see in figure 3, we have skip connections (denoted by arrows) between each residual blocks and one overlying skip connection from the first convolution layer to the last convolution layer before upsampling blocks. This allows our neural network to be very deep while solving problem of vanishing gradient.The architecture consists of the following major blocks:

- **Normalizing and Denormalizing** - We normalize the input by dividing it by 255 and denormalize the output at the end of the model by multiplying with 255. Normalizing changes the input within the range [0,1] which makes gradient calculation faster.
- **Convolution Layers** - We have one convolution layer of 64 filters, kernel size 3x3 and stride 1 at the start, one convolution layer after the residual blocks of the same size and one convolution layer at the end of the model of 3 filters, 1x1 kernel size and stride 1. There is a skip connection after the first convolution to the middle convolution.
- **Residual Blocks** - We have 16 residual blocks in the middle, each having skip connections in between. Each Residual blocks consists of 2 batches of Convolution Layer (64 filters, 3x3 kernel size, 1 stride) and Batch Normalization Layer activated with ReLu.
- **Upsampling Blocks** - same as in CNN model

*3) **Generator Adversarial Network**:* Generative Adversarial Networks (GANs) are a family of Deep Neural Networks which focus on generating data (in the form of images/test) by training two models in parallel. These two models, the Generator and Discriminator complement each other where the generator creates fake images from the input data which the discriminator aims to distinguish the fake image from the real one. This allows both of them to increase their accuracy. We referred the SRGAN [1] paper to create the generator and discriminator models with some changes to filters, kernel sizes, activation functions and skip connections. Figure 3 and 4 shows the architecture that we follow. You can refer the original paper for the original architecture.

We use the same model as ResCNN detailed in previous section for the generator part of GAN. We use the trained weights of ResCNN as pretrained weights to the generator. This pretraining helps to mitigate the problem of vanishing gradient as the generator often creates very unstructured data at the start of training due to random initialization which causes the discriminator to classify all images as fake making the gradients zero and hence causing vanishing gradient problem. Pretraining helps the generator to start with a good initialization of weights.

The Discriminator model is shown in figure 4. It consists of the following major components:

- **Normalize** - Like the generator we normalize the input to make the gradient calculation faster and to converge better. But unlike generator we normalize the input between [-1,1] by dividing it with 127.5 and subtracting by 1. We also don't need to denormalize since discriminator classifies image to real or fake.
- **Discriminator Blocks** - We have 10 discriminator blocks with one skip connection in our model unlike the original paper which only had 7 with no skip connections. Each discriminator block consists of a convolution layer with variable number of filters, kernel size and strides as described in the figure, a Batch Normalization Layer and LeakyReLu activation. There is one skip connection between the 7^{th} and 10^{th} block.
- **Fully Connected Layer** - We have one Dense Fully Connected Layer of 1024 units activated with LeakyReLu and a final sigmoid Dense Layer to classify image as real or fake.

IV. Model Training and Hyper-parameter Selection

A. Evaluation Criteria

Since we have unsupervised learning models where we don't provide the model with output label, traditional loss and accuracy metrics are not sufficient to evaluate our model as the output will be subjective. There are certain noise evaluation metrics that are often used to evaluate output of generative models like: SSIM [3] (Structural Similarity Index Metric) and PSNR [4] (Peak Signal-to-Noise Ratio). We use SSIM and PSNR when evaluating the model but while training we make use of loss functions tailored to each model.

- **SSIM** is a reference metric that evaluates noise from a reference image (our original high resolution image) and a processed image (the output of GAN model).
- **PSNR** is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of a image's representation.
- **User Opinions** - Since output is very subjective, each of our team member gives opinion about the quality of image.
- **Loss** - We use a different loss function for each of our model:
  - **CNN** - Mean Absolute Error
  - **ResCNN** - Mean Squared Error
  - **GAN** - For Generator we use Perceptual Loss which is a combination of Content Loss and Adversarial Loss as shown in equation 1. Content Loss is also called VGG Loss where we use a subset of the VGG19 model upto the 54th layer and compare the output of the SR and HR image to get the loss as shown in equation 2. The adversarial loss is the generator loss implemented as a binary cross entropy loss. The discriminator loss is also implemented as a binary cross entropy loss.

$$L_{perceptual} = L_{content} + 10^{-3} * L_{generator} \quad (1)$$

$$L_{content}(I^{HR}, I^{SR}; \phi, l) = \frac{||\phi_l(I^{HR})\phi_l(I^{SR})||_2^2}{H_l W_l C_l} \quad (2)$$

$$L_{generator}(I^{LR}; G, D) = logD(G(I^{LR})) \quad (3)$$

SSIMs are considered a more reliable indicator than PSNR since they are based on visible structures in the image (noise). These metrics actually measures the perceptual difference between two similar images but cannot judge which of the two is better in actual.

TABLE I
Hyper Parameter Tuning

Parameter	Value	Avg_Perceptual_Loss	Avg_Discriminator_Loss
Learning Rate	0.001	0.132	0.121
Learning Rate	0.0001	0.189	0.132
Hidden Layers	0	0.173	0.156
Hidden Layers	1 (1024)	0.127	0.109
Activation	ReLu	0.211	0.187
Activation	LeakyReLu	0.110	0.097

B. Hyper-parameter Tuning

One of the limitations of GANs is that they are effectively a lazy approach as their loss function is trained as part of the process and not specifically engineered for this purpose. This makes training them much harder than other deep learning models. Another technical difficulty faced is the long time taken to train these models. This also makes hyper-parameter tuning a difficult task. As a result, we only tune certain parameters like learning rate, hidden layers and activation. The moving average of the Perceptual Loss and Discriminator Loss for the last 250 epochs is taken as a criteria for selecting the better parameter. We run each parameter stepwise for 10000 epochs and have shown the results in table I. We finalize our parameters as learning rate 0.001, 1 hidden layer of size 1024 and LeakyReLu activation.

C. Learning Curves

The Learning Curves for each model can be seen in figure 5, 6 and 7. For CNN, you can see that the loss and psnr curve plateaus after about 5000 epochs. For ResCNN, the curve is steadily decreasing even uptil 100000 epochs as evidenced by the moving average line. We could train further but stopped

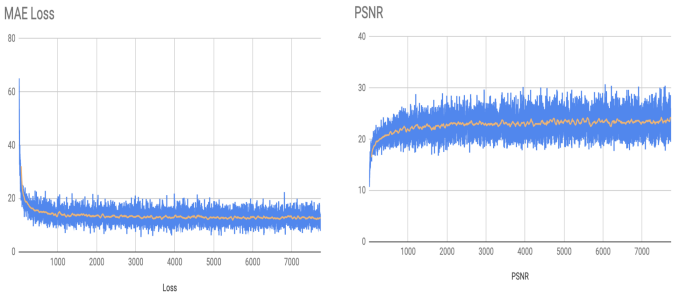

Fig. 5. CNN Learning Curve - Loss and PSNR

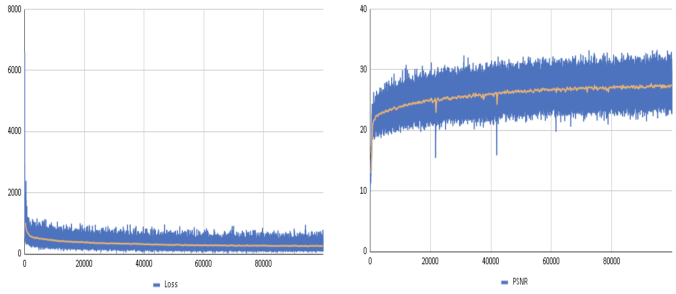

Fig. 6. ResCNN Learning Curve - Loss and PSNR

because of time constraints. PSNR for ResCNN is also steadily increasing. For GAN, since we used ResCNN as pretrained model, the generator loss doesn't decrease too much. However, there is a downward trend in the dicriminator loss as you can see in figure 7.

V. Evaluation and Results

Figures 8, 9, 10 and 11 show the side by side comparison of some super resolution images from the validation set for each model compared with the upscaled low resolution image on the far left and high resolution image on the far right. Each image has a PSNR and SSIM value by comparing it with the high resolution image. For both PSNR and SSIM, higher means better. The difference might not be visible due to the low quality in PDF, but you can see the zoomed in cross-section of the images for better comparison. For figure 8, you can see

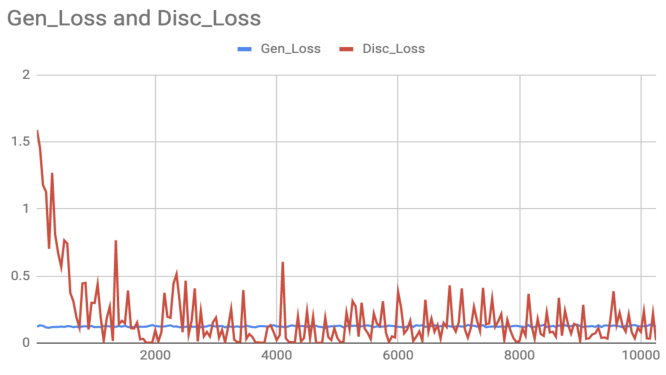

Fig. 7. GAN Learning Curve - Perceptual Loss and Discriminator Loss

that none of the models performed well. ResCNN and GAN performed significantly better than low res and CNN image but the rays of light are more thicker than actual. ResCNN illuminated the tower more than what it actually is. SSIM of CNN is higher than that of others which contradicts our visual analysis. This can be seen in all other images. SSIM is not a good indicator for this test as it tends to prefer CNN. For figure 9, GAN image has a lot of noise and artifacts (especially the sky). ResCNN is better in that respect but the color is way more dull than actual. This is also seen in other images where ResCNN has a more greyish look. For figure 10, ResCNN and GAN performed really well and almost matches the high resolution image. CNN on the other hand doesn't perform as well. ResCNN again has a greyish look while GAN is very close to high resolution in detail and color. PSNR seems to again prefer ResCNN though. For figure 11, though the cross section of CNN looks better than ResCNN, if you look at the entire image, ResCNN is better but still has lot less detail and color than HR. GAN looks very good and in fact added more detail than what HR has.

To conclude, ResCNN and GAN recreates the original image very well considering that we manually added a lot of degradation and noise in the low resolution images. Since, these are generator models, they tend to add a lot of data which might not be in the original image. Both ResCNN and GAN, sometimes messes up the image by adding artifacts, noise and blank spaces. SSIM is not a good indicator for quality. PSNR gives a higher score for ResCNN and GAN but it is biased towards ResCNN as it always gives it a higher score even when it is visually worse. GAN tends to add more details and preserves the color balance like the original. ResCNN tends to add less details in comparison and also tends to have a greyish image as if a filter was applied to it. This might be different if we train with different parameters or train the models more. The results are also very subjective as we had disagreements between ourselves to determine which model result was better. To see all our results in high resolution and source code, have a look at our GitHub repository at https://github.com/ssandanshi/Super-Resolution-GAN.

References

[1] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[2] Lai, Wei-Sheng, et al. "Fast and accurate image super-resolution with deep laplacian pyramid networks." IEEE transactions on pattern analysis and machine intelligence (2018).

[3] "SSIM: Structural Similarity Index — imatest", Imatest.com, 2019. [Online]. Available: http://www.imatest.com/docs/ssim/. [Accessed: 03- Oct- 2019].

[4] "Peak Signal-to-Noise Ratio as an Image Quality Metric - National Instruments", Ni.com, 2019. [Online]. Available: https://www.ni.com/en-us/innovations/white-papers/11/peak-signal-to-noise-ratio-as-an-image-quality-metric.html. [Accessed: 03- Oct- 2019].

[5] C. Thomas, "Deep learning based super resolution, without using a GAN," Medium, 24-Feb-2019. [Online]. Available: https://towardsdatascience.com/deep-learning-based-super-resolution-without-using-a-gan-11c9bb5b6cd5.

[6] W. Shi et al., "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," Sep. 2016.

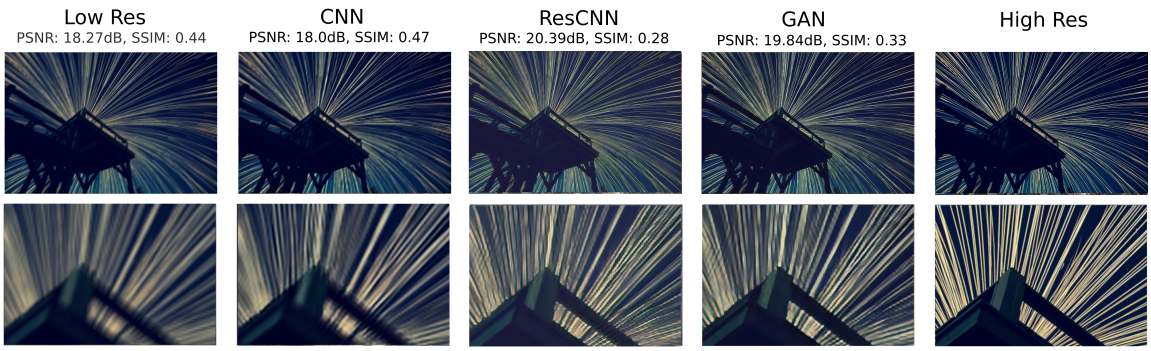

Fig. 8. Results Comparison 1

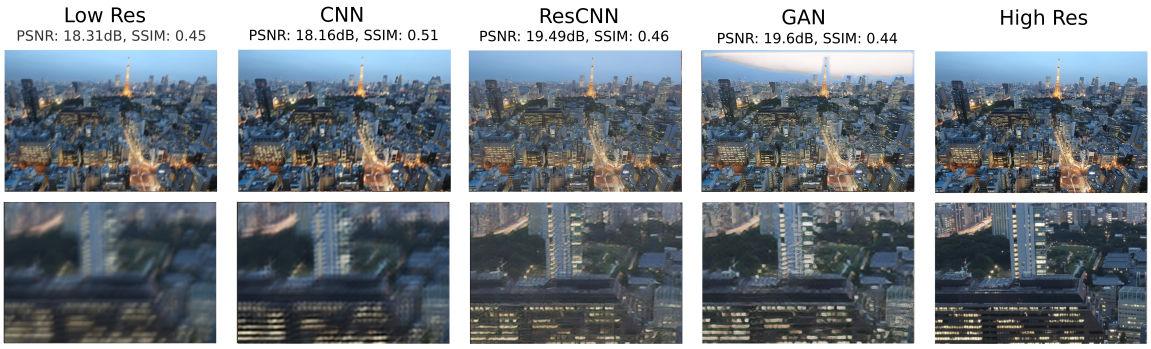

Fig. 9. Results Comparison 2

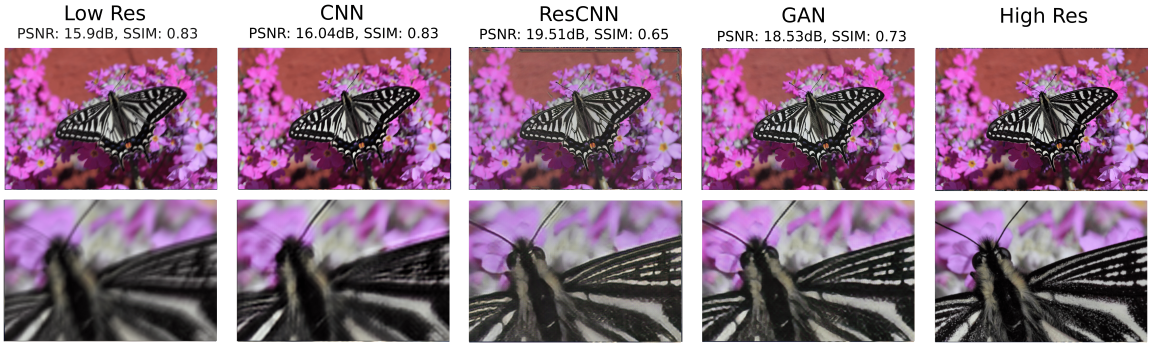

Fig. 10. Results Comparison 3

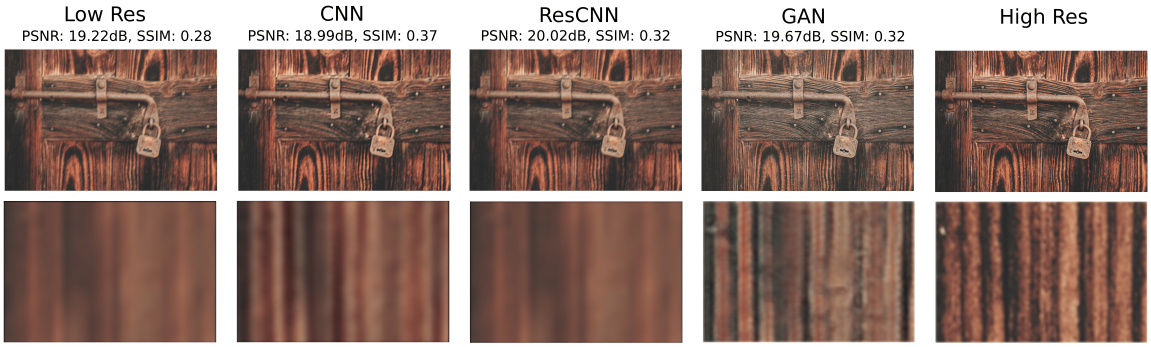

Fig. 11. Results Comparison 4