Implementing Super Resolution GAN: Upsampling Images

Raghvendra Pratap Singh - 50289128[a)]

(Dated: 16 May 2019)

This project aims to implement the original architecture of SR-GANS (Super Resolution Generative Adversarial Network) trained on DIV2k and FLIKR2k dataset. We attempt to replicate the orignal results and show that with slight modification in the residual layers, we can achieve better quality of super sampled images in same number of epochs.

I. REVISIT GANS

The GAN can be seen as a game between two players, the generator and the discriminator:

Generator: the generator is the generative model, and performs a maping x=G(z), where z is some random noise. Its goal is to produce samples, x, from the distribution of the training data p(x)

Discriminator: The discriminator is the generators opponent, and performs a mapping D(x)(0,1). Its goal is to look at sample images(that could be real or synthetic from the generator), and determine if they are real samples (D(x)closer to1) or synthetic samples from the generator (D(x)closer to 1). D(x) can be interpreted as the probability that the image is a real training example

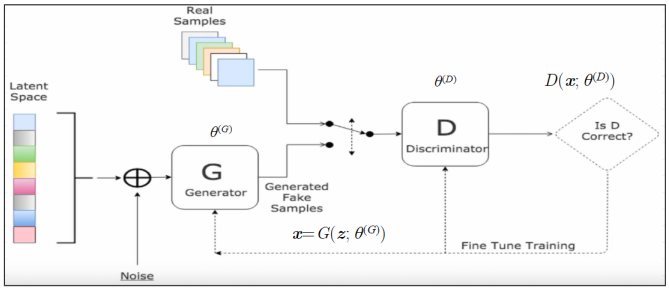

FIG. 1. Vanilla GANs process

Because of this setting, it's common to refer to them as an 'adversarial' game, but I like to think of it as a 'cooperative' network.

The objective function of our original GAN is essentially the minimization of the **Jensen Shannon Divergence** (JSD). Specifically it is:

$$\frac{1}{2}KL(P||M) + \frac{1}{2}KL(Q||m) \tag{1}$$

where $KL(A||B) = \sum^{i} Na_i \ln \frac{a_i}{b_i}$ and $M = \frac{1}{2}(P+Q)$

A. Loss Function in Vanilla GANs

Loss function in GANs is essentially a min max game between the generator and discriminator where we are opitmizing:

$$min_G max_D [\frac{1}{2}E_{x\ p_data} \log D(x) + \frac{1}{2}E_z \log(1 - D(G(z)))] \tag{2}$$

1. The discriminator wants to maximize the objective (i.e., its payoff) such that D(x)is close to 1 and D(G(z))is close to zero.
2. The generator wants to minimize the objective (i.e., its loss) so that D(G(z)) is close to 1

If D(G(z))=0, as may happen early on in training when the discriminator can tell the difference between real and synthetic examples, the gradient is close to zero. This results in little learning for(G), and thus in practice the generator cost function

$$E_z \log(1 - D(G(z^(i))))] \tag{3}$$

is rarely ever used.

Instead, we opt for a cost function that has a large gradient when D(G(z))=0, so that the generator is encouraged to learn much more early in training. This is the cost function

$$-E_z \log(D(G(z))) \tag{4}$$

This still obtains the same overall goal of being minimized when D(G(z)) = 1,but now admits far more learning when the generator performs poorly.

We will now see how all this ties in with our Super Resolution GANs:

II. HOW SUPER RESOLUTION GANS ARE DIFFERENT

At it's core, SR-GANs follow the same intuition of min max game between generator and discriminator.

However, the major difference lies in the training input and cost function of the SR-GANS architecture:

1. The generator instead of feeding on random noise is provided with the low resolution variants of the images and is asked to generate SR images
2. Cumulative cost function consists of Content loss and a weighed down adversarial loss rather than MSE as we saw in vanilla GANs

We discuss more about the loss functions in detail below:

[a)] Also at CSE Dept., UB.

A. Loss Function in SR-GANs

Instead of plain old MSE loss, we use what is called as Perceptual Loss:

$$l^{SR=} \underbrace{l_{\mathrm{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}} \tag{5}$$

a. *Content Loss* :

While the MSE does fine job as a cost function, the results are not very eye-pleasing to a human. It's this reason that we use a content loss which is basically in our implementation euclidean distance between the features extracted from a VGG-19 network which is pre-trained on 'imageNet'. For a specific layer within VGG-19, we want their features to be matched (Minimum MSE for features):

$$l_{VGG/i,j}^{SR} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j} \left(I^{HR} \right)_{x,y} - \phi_{i,j} \left(G_{\theta_G} \left(I^{LR} \right) \right)_{x,y} \right)^2 \tag{6}$$

where $\phi_{i,j}$ refers to the feature map for the j-th convolution (after activation) and before the i-th max pooling layer.

b. *Adversarial Loss* :

This pushes our solution to the natural image manifold using a discriminator network that is trained to differentiate between the super resolved images and original photo realistic images:

$$l_{Gen}^{SR} = \sum_{n=1}^{N} - \log D_{\theta_D} \left(G_{\theta_G} \left(I^{LR} \right) \right) \tag{7}$$

Here, $D_{\theta_D} G_{\theta_G}(I^{LR}$ is the probability that the reconstructed image $G_{\theta_G}(I^{LR}$ is a natural image. For better behaviour we minimize $\log D_{\theta_D} G_{\theta_G}(I^{LR}$ instead of $[\log 1 - D_{\theta_D} G_{\theta_G}(I^{LR}]$ drawbacks of which were discussed in the previous section II A.

III. TRAINING PROCESS

Now let's discuss how I went with the training process: Broken down as follows:

1. Training Data Used
2. Generator and Discriminator Parameters
3. Batch Training

A. Training Data Used

DIV2K AND Flickr2k dataset were used to train the model. Div2k consists of 800 2k resolution images and flickr2k consists of 2000 2k resolution images.

The images read are first shuffled to mix in both the datasets.
Data is re-sized to 56, 56, 3 shape to represent lower resolution image 2k Higher resolution images are reshaped to (224, 224, 3). A better alternative would be to crop images from the center, however as a result the images are sometimes just shades of color. Hence I went forward with the resize method. The HR images do lose a ton of information, but it's not too bad from a relative perspective. Another important pre-processing of image is normalize the pixel values before fedding to our VGG extractor, a value of **127.5** is subtracted and divided to normalize the training and validation images. Once the model is trained the generated images are denormalized before plotting and saving.

B. Generator and Discriminator Models

We can see below the architecture of our original GAN as proposed in the paper. The number of trainable parameters for our generator model is 2,039,555, and for our discriminator model is 107,451,585

a. *Arch of Generator* : Our generator consists of:

Conv2d (filter:9 feature:64 stride:1), followed by PReLU activation

16 residual blocks consisting of Conv2d, BN, PReLU and BN with an element wise sum, all skipping a single subsequent residual block

Another conv2d, BN and element wise sum

2 up-sampling blocks produing the image from 56, 56 to 224, 224 res

One last conv2d layer with (filter:9 feature:64 stride:1) and activated on **tanh**

b. *Arch of Discriminator* : The discriminator is more traditional in it's architecture,it consists of:

Conv2d (filter:3 feature:64 stride:1), followed by LeakyRElu activation

6 more conv2d blocks of 1 and 2 strides extracting 64, 128, 256, and 512 features respectively

Followed by a Dense(1024) layer and activated on LeakyRely

The output layer is a single Dense(1) layer activated on sigmoid function

We compile both our generator using VGG and cross-entropy loss function as discussed in previous section and optimized using Adam with a learning rate of 1e-4, a beta value of 0.9 and beta 2 value of 0.999

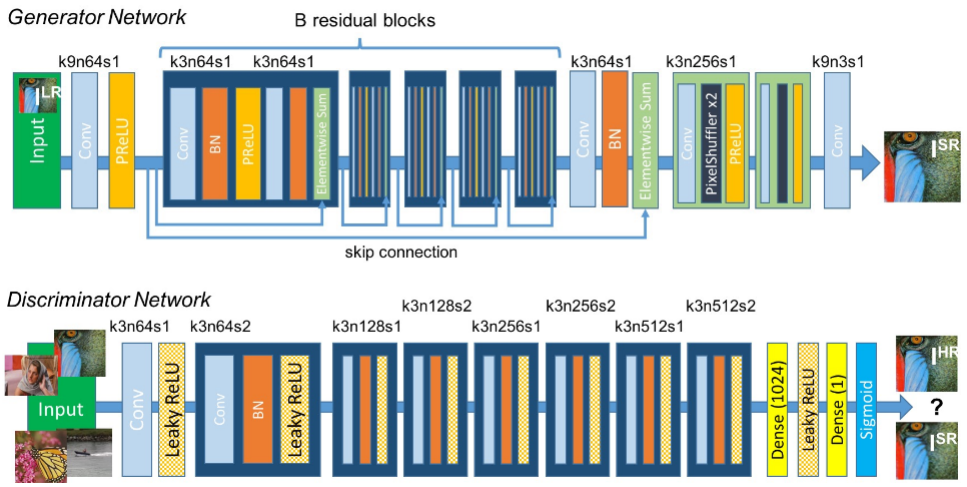

FIG. 2. Original Architecture used in SR GAN paper and in my vanilla implementation

C. Batch Training Process

Since it is impossible to train upto 50k epochs let alone 99k trained in the original paper, I choose to save the model every 50 epochs and train iteratively. Internally, the training process is similar to the original GANs.

A batch size of 8 LR and HR images is used The generator is asked to predict on some random samples of 8 images. These samples are fed to the discriminator to train on and try distinguish the generated image from the real one. The discriminator is then frozen and the perceptual loss is passed to the GAN network to be trained on the random samples of 8 images. This perpetuates the cycle and let our generator utilize the perceptual loss to generate better images. This training cycle is shown below:

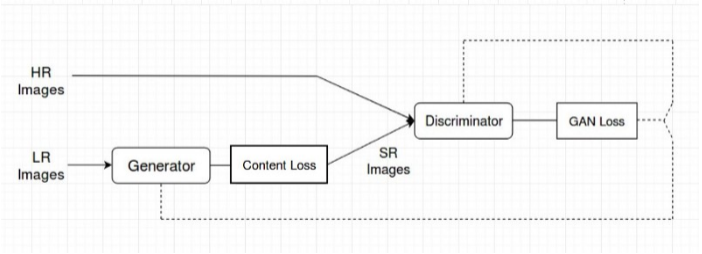

FIG. 3. Feature extraction from Conv5-4 layer of VGG

Every 10 epochs I'm plotting the generated test images and storing the loss values to evaluate model performance.

As in the original paper, the content loss is extracted after the 5th convolution layer and before the 4th maxpool layer, it is basically the last batch of convolution before the top layer as we can see from below image:

In the next section, I discuss how the images generated from our vanilla SR-GAN looks like:

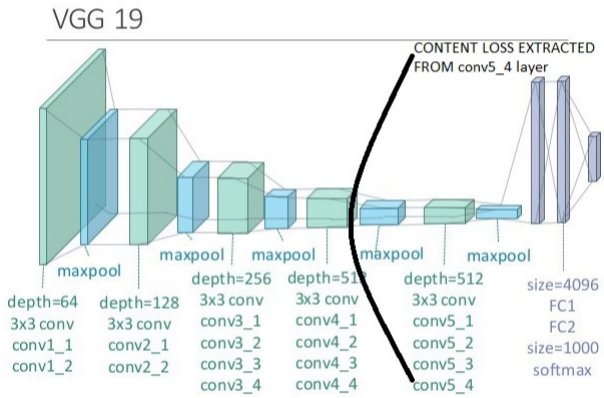

FIG. 4. SR-Gans Training process

IV. RESULTS FROM SR-GAN

I showcase the results for epoch 50, 100, 500 and 1000 epochs for the vanilla SR-GAN implementation, all images belwo are 224*224 resolution.

We can see the image quality progressively increases especially the information gained in textures improves considerably from 50 to 100 and from 500 to 1000 epochs.

FIG. 5. Images after 50 epochs

FIG. 6. Images after 100 epochs

FIG. 7. Images after 500 epochs

FIG. 8. Images after 1000 epochs

We also see the comparison between bicubic, generated image after 1000 epoch and the original image. We can see the quality is improved from biocubic version, however the generated images are no where as good as shown in the original paper.The color reproduction is a little hazy near the borders, but the textures reproduced are fairly consistent. Probably because I don't have the resources to train it for 100k epochs. Maybe one day, but there's no promise that the error will go on converging. The resolution of 224*224, I think is not too great to be considered as a high res image. We also have the problem of information loss by re-sizing the images from 1024 res to 224. The images generated are fine for now and I'll try some more changes in the architecture described in the following section.

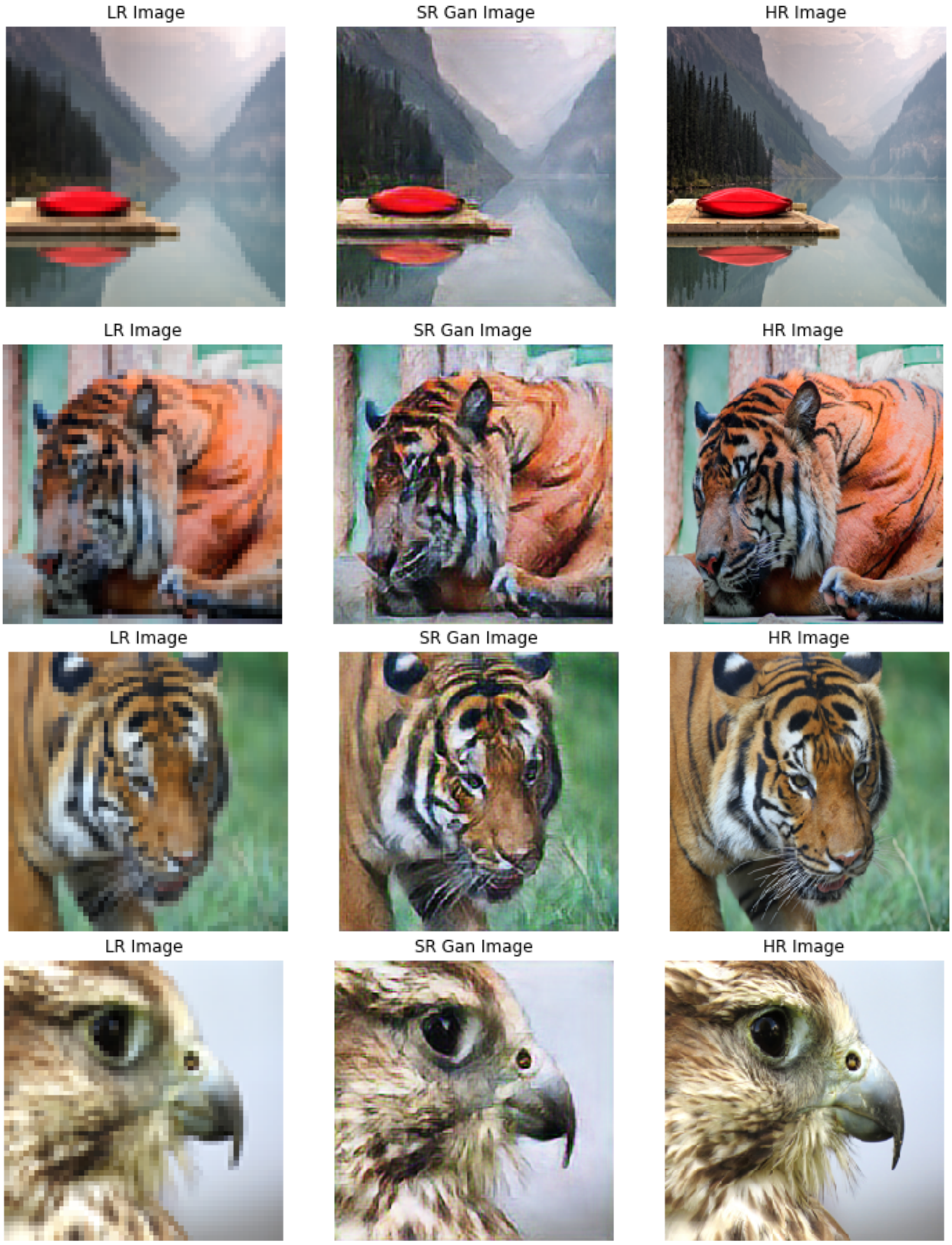

FIG. 9. Images comparison after 1000 epochs

V. SR-GAN WITH INCEPTION-A AS BACKBONE

After the modest performance of vanilla SR-GAN, I tried changing the architecture particularly in the residual blocks upgrading from a linear stack of 16 residual k3n64s1 block to what is called an Inception-A block. Below is the image depicting the architecture changes:

In it's essence, Inception-A block captures what al-

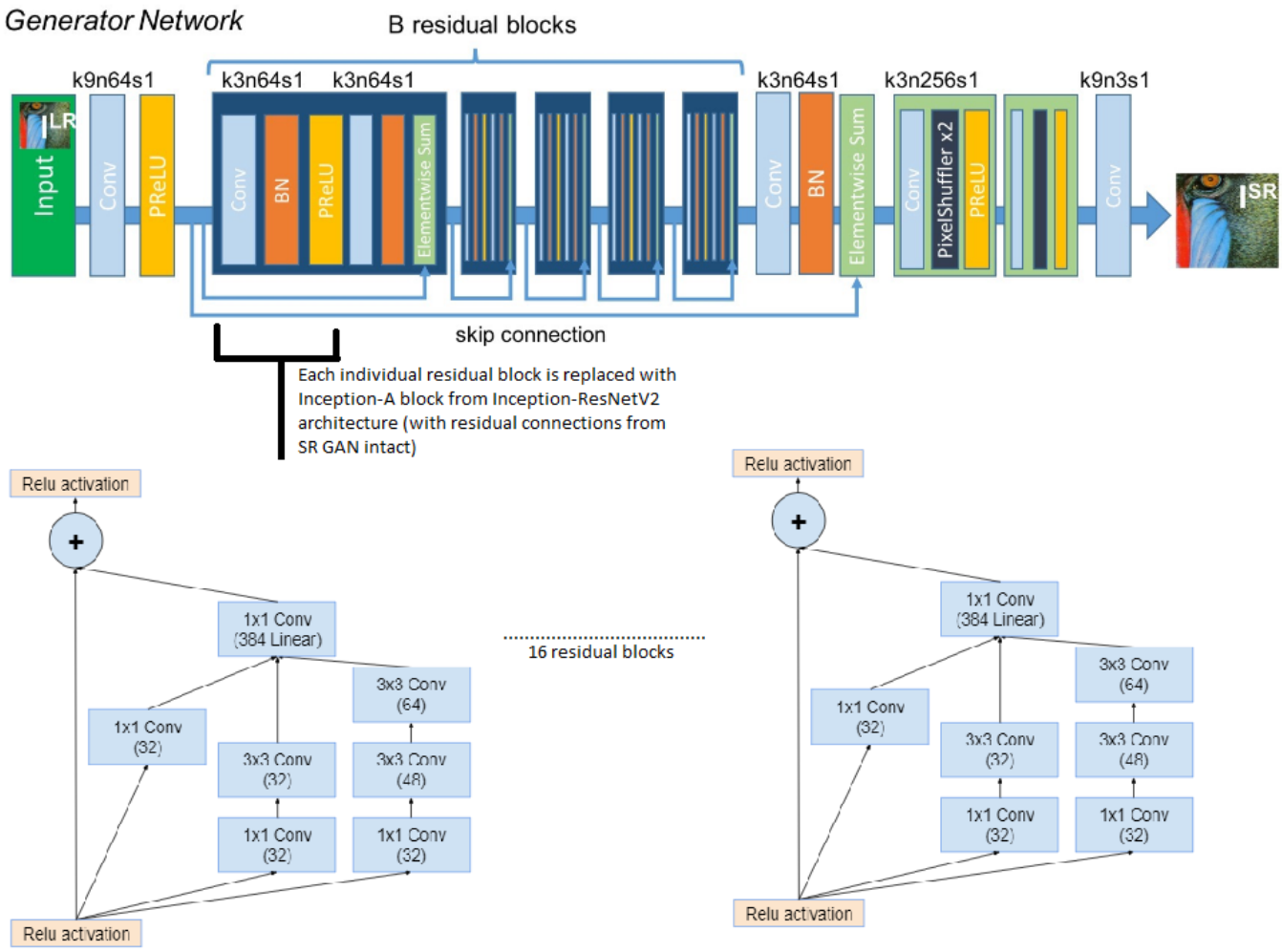

FIG. 10. Basic k3n64s1 residual block replaced with Inception-A block

lowed the use of multiple filters using factorization with significant less cost. This block is stacked 16 times each having a residual connection. We are thus able to extract more features from the image and also decrease the number of parameters by 100k. Extracting more features while reducing the number of parameters is a win-win situation and should lead us to better generated images.

Also, to note, no changes are made to my discriminator architecture and the perceptual loss is being extracted from VGG19 architecture.

As is demonstrated in the original Inception paper, when the number of filters is exceeding 1000, the generator network simply dies, as a result, they are scaling down the residuals before adding them to the previous layers activation seems to stabilize the training. Since, I'm using 16 blocks of residuals made up of Inception-A block, I have gone ahead and scaled my residuals conenctions as well, using the value **0.17** used in the native implementation of InceptionResNetV2 in keras.

Below are the same images generated using the modified architecture:

As we can see, images have progressively become sharper and have gained textures even more so than the vanilla SR-GANs. Especially the region near eyes and forehead of the tiger and eagle has gained sharper edges.

FIG. 11. Images with Inception-A residual generator after 50 epochs

FIG. 12. Images with Inception-A residual generator after 100 epochs

FIG. 13. Images with Inception-A residual generator after 500 epochs

FIG. 14. Images with Inception-A residual generator after 1000 epochs

Let's compare the images side by side with it's HR and bicubic variation:

As we saw in our original implementation, the SR images are much similar visually to the HR images, the interesting aspect is that textures around edges are more robust and color reproduction is more visually appealing that the SR images of original architecture 9

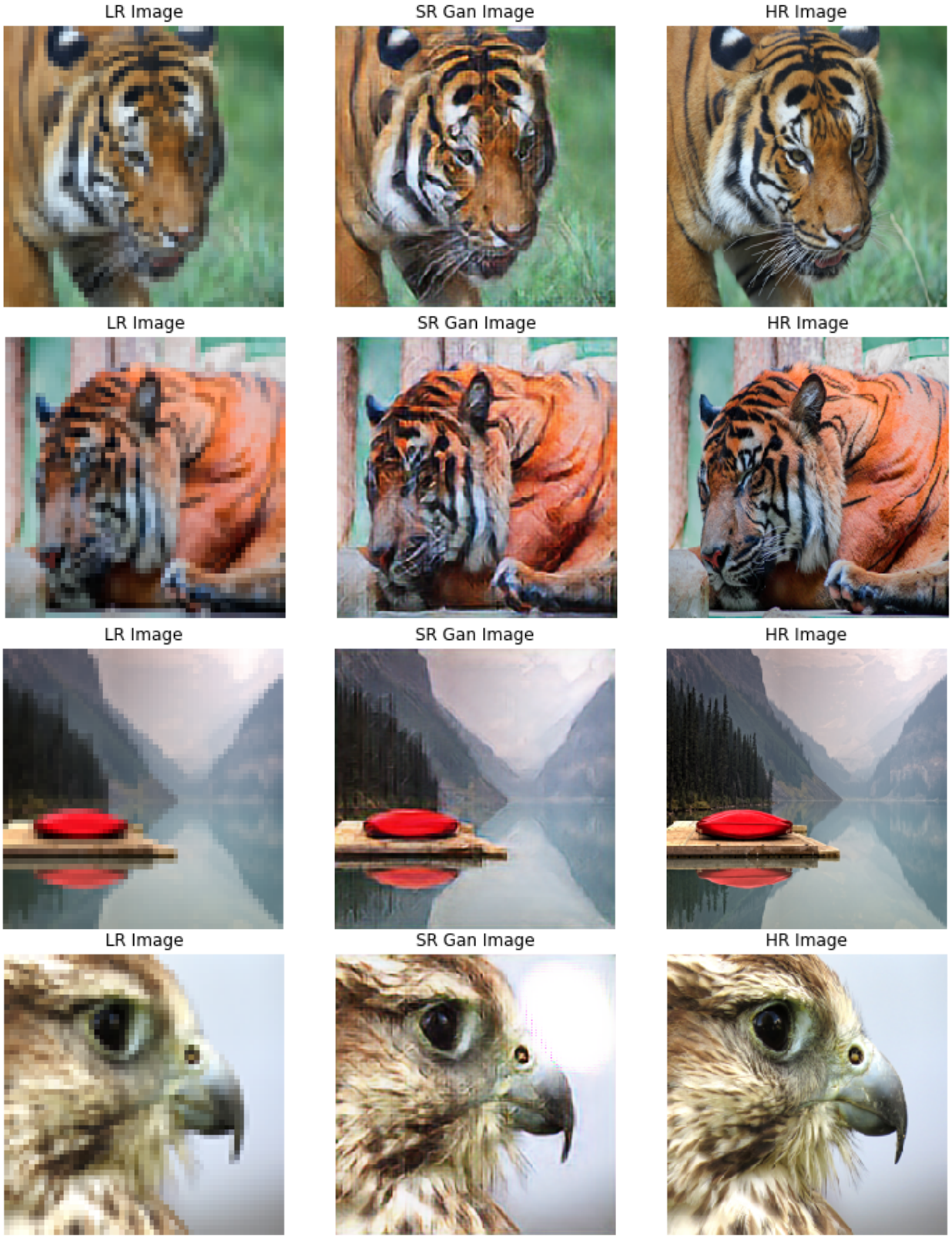

FIG. 15. Modified GAN arch - Images comparison after 1000 epochs

VI. WHAT I TRIED, BUT DIDN'T WORK

I figured, a natural extension to the Inception-Resnet backbone would be to extract the loss from one of it's cousin architectures pretrained in Keras. So, instead of extracting the loss from the conv5-4 layer of VGG19, I am now extracting it from the Inception-ResNetV2 architecture especially from the middle of the 35th block as shown in 17

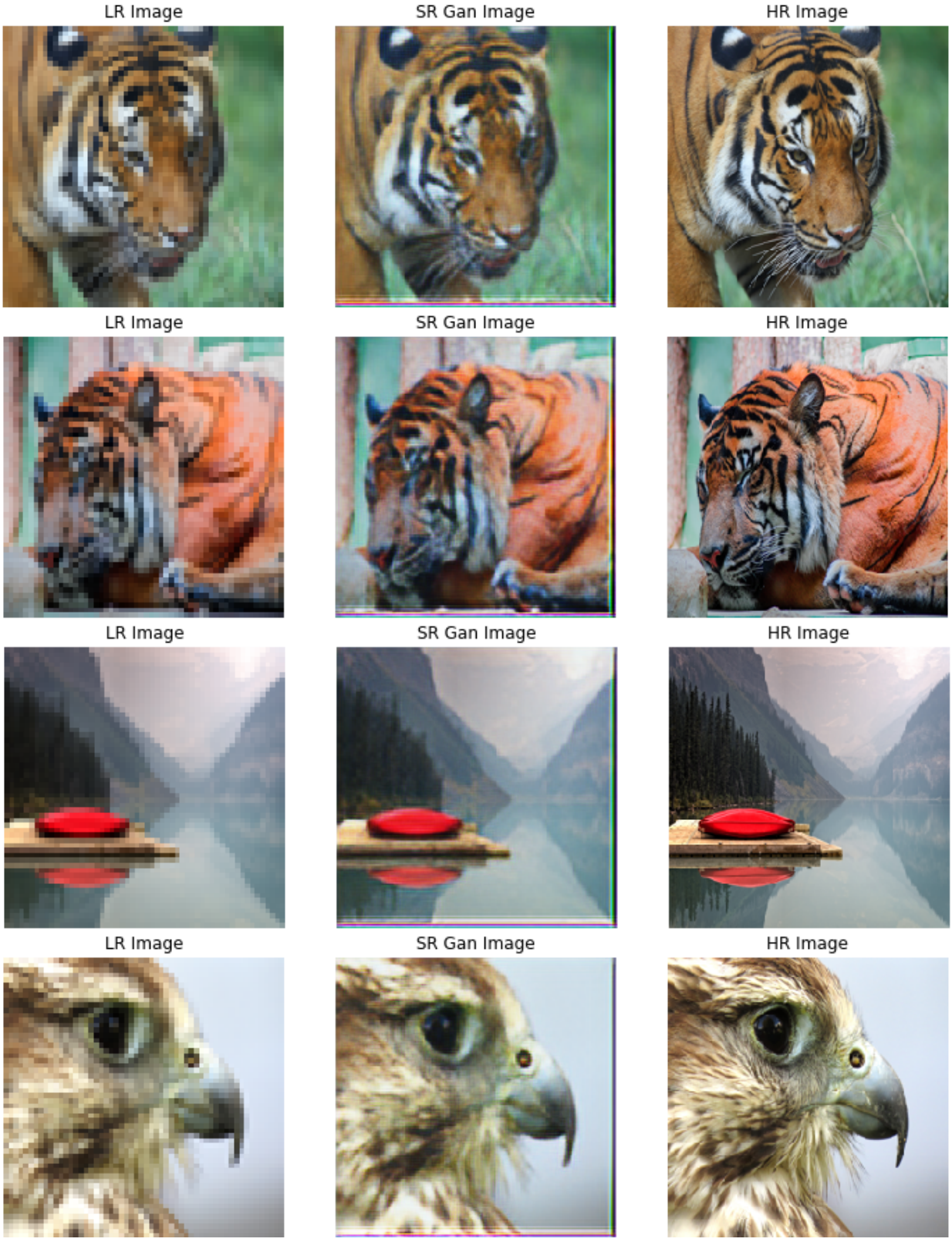

FIG. 16. ResNetV2-feat-extracted - Image comparison after 500 epochs

Rest of the architecture is kept same as in previous section. The results generated are, however, not as good as when we used VGG network to extract features. Below are the results comparison after 500 epochs on the same set of images, the reason I stopped at 500 epochs is because there was no converging just after 100 epochs on the same batch size of 8 and even the images generated were having the same flaws.

As we can see the color reporoduction is decent, however, the images generated are blurry adn do not cap-

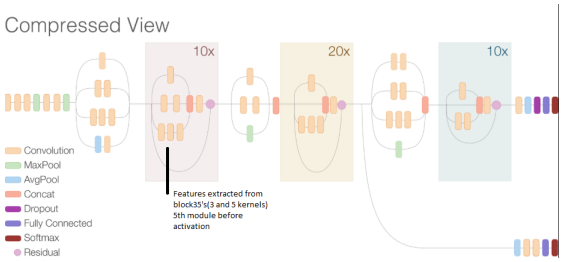

FIG. 17. Perceptual Features extracted from pre trained InceptionResNetv2 network

ture the textures properly. THis is most certainly due to poor features being given by the pre-trained resNetV2 arch particularly from block35 i.e Inception-A block.
i tried changing the blocks from where to extract the feature first with mixed5b(before the block35) and from later blocks as well but, the results only worsened. the decision to extract features before the activation is inspired by the ESR-GAN paper, but as we can see images are not upto par with what we have already tried.

Below we compare the GAN loss values among the three variations of the model in this project:

Epochs	SR-GAN	Incep[a]	Incep[b]
10	0.04	0.07	0.05
50	0.02	0.02	0.03
100	0.012	0.001	0.022
500	0.007	0.005	0.009
1000	0.00045	0.003	–

[a] SR-GAN with Inception-A Residual and VGG extraction
[b] SR-GAN with Inception-A Residual and InceptionResNetV2 extraction

From the table we can see the second variant converges better than the other two while the third variant starts better but does not continue the trend.

VII. CONCLUSIONS

In conclusion, I would say that the Inception ResNet backbone produces slightly better images (visually) than the vanilla SR-GANs and that VGG feature extractor is better than the other pre trained feature extractor in terms of learning the texture details of any image. HOwever, the images produced in a mere 1000 epochs are not up to par to what we saw in the orginal paper and a much more exhaustive training set as well as epochs trained on has to increase to reach the desired quality.