CSE 674 Project 3: H-Single Image Super Resolution Using GANs

Aishwarya Manoharan

AM355

Department of Computer Science

State University of New York, University at Buffalo

Buffalo, NY 14260

am355@buffalo.edu

Abstract

The aim of this project is to recover finer texture details when we resolve a low-resolution image to a high-resolution image. Reconstructing the image pixels by finding the mean of the neighboring pixels leads to mean square loss. There are recent work focusing on reconstruction of root mean square loss. But the results still lack in high-frequency details. In this project, we do not consider MSE as the prime optimization target. We are going to implement the Super Resolution Generative Adversarial Network(SRGAN). This method makes use of a proposed idea of a perceptual loss function, which has adversarial loss and content loss. The idea behind content loss is to use perceptual similarity of surrounding pixels instead of taking mean of neighboring pixels. The adversarial loss makes use of a discriminator network that is trained to take the original image and the super-resolved image and differentiate between the two images. In a recent study it was found that the Mean Opinion Score(MOS) calculated from SRGAN image was closer to original image than image obtained by any other leading image resolution techniques. In our project we use deep Inception Networks in the generator. Also, we do not consider MSE as the prime optimization target.

1 Base paper content

1.1 Peak Signal to noise ratio PSNR

The signal to noise ratio is used in audio recording where signal is the desired sound and noise is the unnecessary sound. Here in our project it refers to the desired part of image i.e., the pixel that replicate more closely to actual image is the signal and the unexpected pixel results are the noise.

1.2 Super Resolution

The task to reconstructing a high resolution(HR) image from a low resolution(LR) image is called Super Resolution(SR).

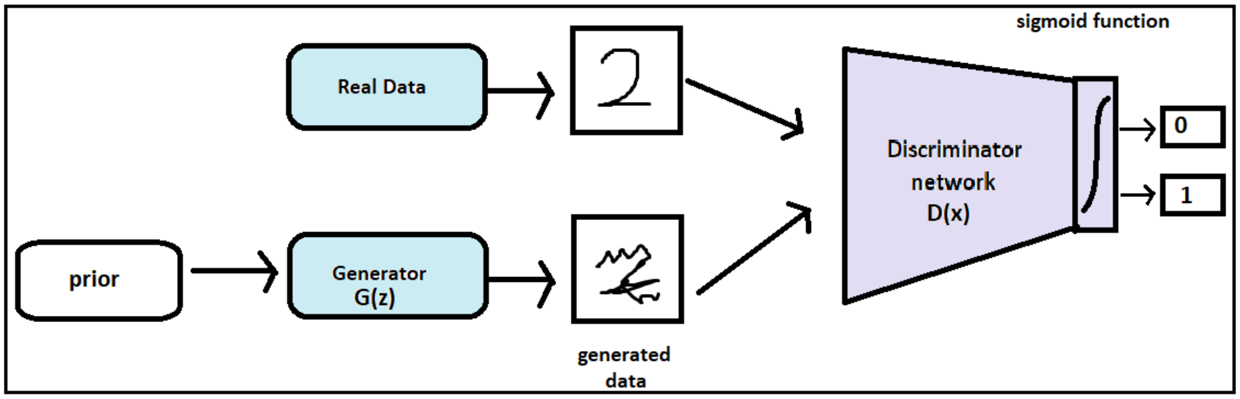
1.3 VGG network

It usually refers to a deep convolutional network for object recognition developed and trained by Oxford's renowned Visual Geometry Group (VGG), which achieved very good performance on the ImageNet dataset. It is used in the base paper implementation and this project implementation.

1.4 GAN – Generative Adversarial Network

Generative model-G: given X and Y, where X → observable variable and Y → target variable, a generative model is a model of joint probability distribution. A generative network directly produces samples. A discriminative model-D attempts to distinguish between the data drawn from actual training data and the data from generative model using its adversary. Once the model has distinguished it produces a probability value to indicate if the data was from G or training data. Both Generative model and Discriminative models are deep neural network. The weights are updated to increase the probability of correctly identifying a sample as belonging to read training data and minimize the probability of classifying it as belonging to Generative model. Technically stating, loss function maximizes the D(x), where x is the data samples, and minimizes G(D(z)).

1.4.1 Discriminator Input and output



The real data and generated data is given as input to the discriminator and it tries to distinguish between the two by giving a probability of 0, for fake images a.k.a. generated Super Resolution Images and 1, for Real image a.k.a. High Resolution Images.

1.5 Losses

1.5.1 Content loss

The Euclidean distance between the feature representations of a reconstructed image and the reference image gives the VGG loss. We make use of the feature map obtained by jth convolution in ith max pooling layer in calculating the VGG loss. Instead of calculating pixel-wise MSE loss, which gives high PSNR but lacks content of high frequency, we use loss function closer to a perceptual similarity.

$$l^{SR}_{VGG/\,I,j} = \left(1/W_{i,j}H_{i,j} \right) \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j}\,(I^{HR})x,y - \phi_{i,j}\left(G_{\theta G}\left(I^{LR} \right) \right)_{x,y} \right)^2$$

1.5.2 Adversarial loss

In addition to content loss, adversarial loss is a generative component added to the perceptual loss. It represents the generative component of our GAN. The generative loss is defined based on the probabilities of the discriminator over all the training samples.

$$l^{SR}_{gn} = \sum_{n=1}^{N} -\log D_{\theta D} (G_{\theta G} (I^{LR}))$$

where, the probability that the reconstructed super resolution image is the real high-resolution image is given by $D_{\theta D} (G_{\theta G} (I^{LR}))$.

1.5.3 Perceptual loss function

The sum of content loss and adversarial loss is formulated as perceptual loss.

1.6 Deep residual network

1.6.1 Vanishing gradient problem

An update proportional to the partial derivative of the error function w.r.t. current weight's is given to each layer's weight. When the gradient is vanishingly small it becomes an issue as the weights are not updated significantly. This might even prevent the neural network from training further. This led to bad performance. Since this is not related to overfitting, we cannot use the dropouts to come to the rescue. Kaiming He and his colleagues at Microsoft Research Asia came up with the best solution. Hence the residual network was introduced.

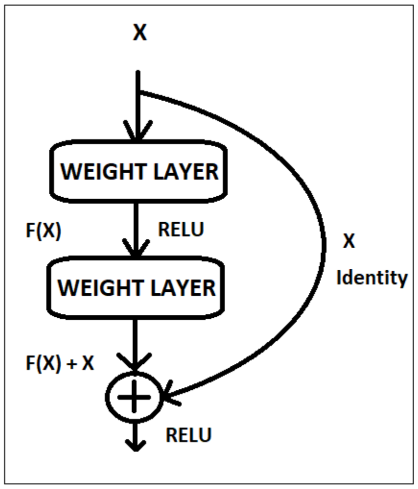
1.6.2 Residual Network

Batch-normalization: Here each layer is enabled to learn a little independent of other layers by itself. VGG does not have a batch normalization because the concept came into existence after VGG. Adding batch-norm to VGG however, can improve the results. Inserting batch normalization into a pre-trained network will change the pre-trained weights by subtracting the mean and dividing by the standard deviation, which is not what we want. We want the pre-trained weights to remain unchanged but normalize the output of the previous layer. The batch-norm is turned off during testing to obtain a output that depends deterministically only on the input.

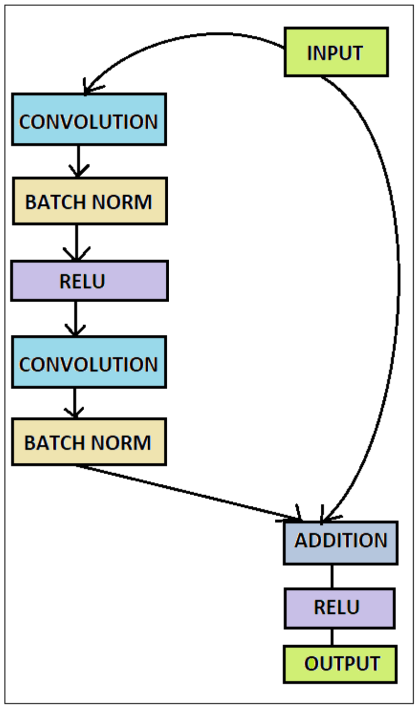
Discussing the possibility of increasing the accuracy by simply stacking more layers of convolutional-batch normalization relu network, the accuracy was found to drop after certain number of layers were added , due to vanishing gradients issue. The solution was residual network. This means to simply connect the output of the previous layers to the output of the new layers. In residual model, we not only pass the output of layer i to layer (i+1), but also, we add up the outputs of layer i to layer (i+1).

For instance, if each layer is denoted by f(x), then in a standard network each layer is denoted by $y = f(x)$, whereas in a resnet each layer is denoted by $y = f(x) + x$.

$F(x) = H(x) – x \Rightarrow H(x) = F(x) + x$. Works when both x and F(x) have same dimensions. Dimension mismatch was handled by applying zero-padding and linear projections. Linear projection(LP) increases the complexity while zero-padding keeps it comparatively simple. No dropouts used. Residual network use skip-connection i.e., short-cuts to jump over layers. Skip connection is used to avoid the problem of vanishing gradients, the previous layer activations are reused until the adjoint layer learns its weights. This allows weight adaptation of adjacent layers without explicit weights from the upstream. This approach needs the intermediate layers to be linear. The skipped layers are gradually restored when the network learns in the feature space.



Residual Network, Skip connections



1.7 ADVERSARIAL ARCHITECTURE

The main idea is that we want to create a generative model G that can produce image to fool the discriminator model D. We do this so that the generator model can learn to produce image that closely resemble the real images. We have residual blocks with identical layout at the core of our deep generative model.

1.8 METHOD

The given training images are high resolution images, I^{HR}. We want to estimate from a low-resolution image I^{LR}, generated by passing through gaussian filter and then down sampling by a r factor, a super resolution image $I^{SR.}$ During training we get the low- resolution image $I^{LR.}$ An image with C color channels requires I^{LR} by a real-valued tensor of size W * H * C and I^{HR}, I^{SR} by rW * rH * C respectively.

Now we want to train a generating function that for a given low resolution image I^{LR} can come up with a corresponding high-resolution image I^{HR}. Our generator function is trained as a feed-forward CNN, $G_{\theta G}$ parametrized by θ_G, where for L layer in the deep network, W1 is the weight and B1 is the bias, and is obtained by optimizing a SR – specific loss function.

$$\theta_G = \underset{\theta G}{\text{argmin}}\ 1/N \left(\sum_{n=1}^{N} l^{sr}(G_{\theta G}(I^{LR}n), I^{HR}{}_n\right)$$

where, l^{sr} is the perceptual loss, which is given the generator function passed with low resolution image, the High-resolution image.

Now that we have discussed the generator function, we need a function to ensure the images generated by the generator is closer to the real high resolution I^{HR} image. To perform this, we use a discriminator function $D_{\theta D}$ that attempts to differentiate between the image that was generated by our function G and the real high resolution I^{HR} image. $D_{\theta D}$ is optimized in an alternative manner to $G_{\theta G}$ to resolve the adversarial min-max issue. We aim to train a generator function so well that it can fool the discriminator function, which is trained to differentiate the high-resolution image and generated super resolution image, with its generated super resolution images as the high-resolution images. We do so to enable our generator to learn to create images close to real image.

1.9 Base paper implementation

1.9.1 Generator

Pre- residual block

A convolution layer with kernel size as 9, number of filters as 64 and strides as 1.

Residual block architecture

The network has 16 identical residual bocks. At the core of our deep neural network we have residual blocks of identical layout. 2 convolution layers with 3*3 kernels are used with 64 feature maps followed by a batch-normalization layer. The activation function used is parametric ReLU.

Image resolution is increased by using 2 trained sub-pixel convolution layer.

Post-residual block

A convolution layer with kernel size as 3, with 64 filters and stride as 1, followed by a de-convolution block with kernel size as 3 with 256 filters and number of strides as 1.

1.9.2 Discriminator network

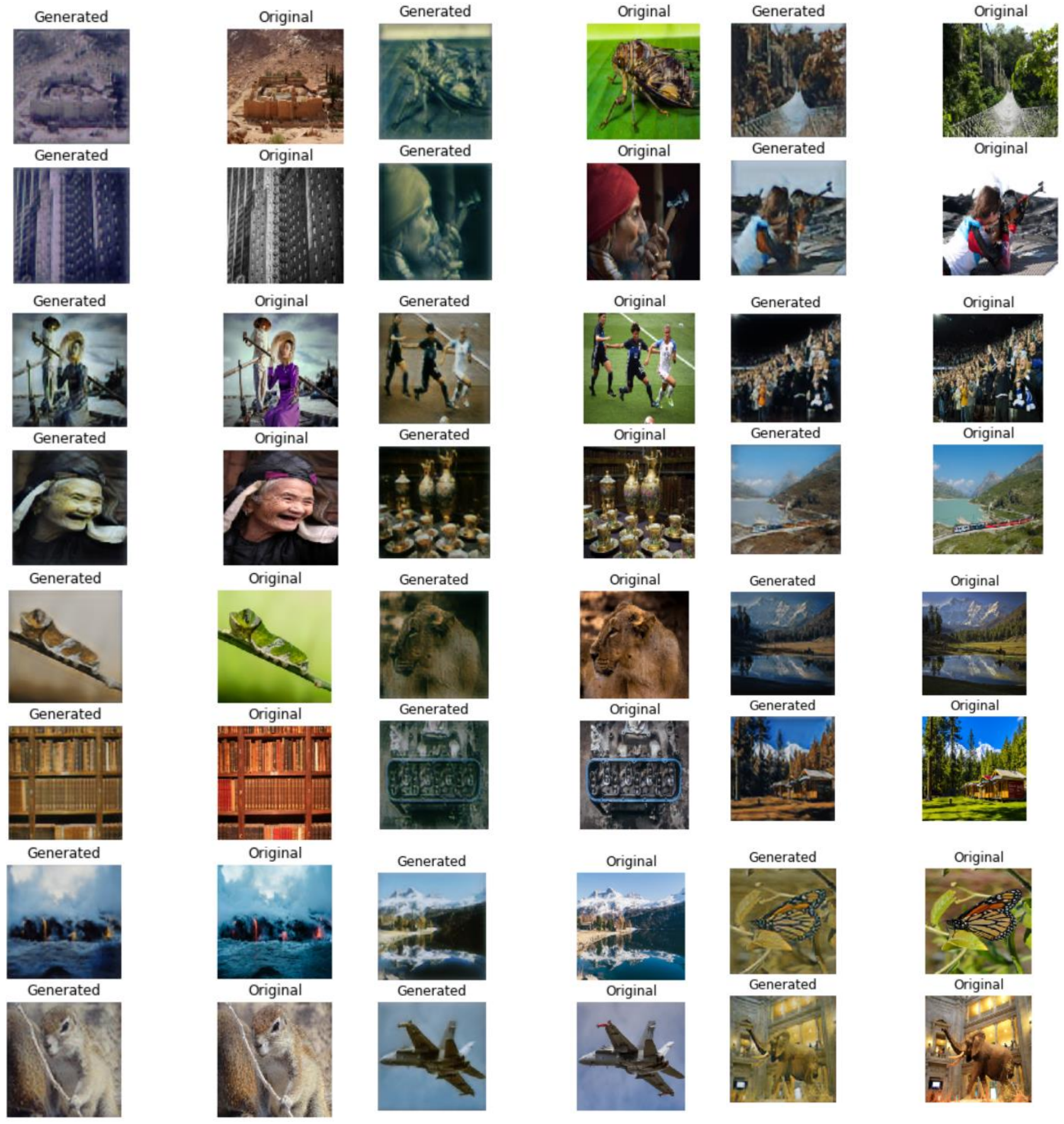
Used activation- LeakyReLU with (α = 0.2).

We also avoid max pooling throughout the network, as the discriminator is designed to solve maximization problem.

There are 8 convolution layers with 3*3 filter kernels. Each of the convolution layer has a 1 stride and 2 stride-convolution. So, 64 map features will have 2 layers, 128 has 2 layers, 256 has 2 layers and 512 has 2 layers with 1 stride and 2 strides respectively. Here number of map-features the number of filters used. The 8 convolution layers are followed by 2 dense layers. The activation function is sigmoid for the dense layers to obtain the probability for sample classification.

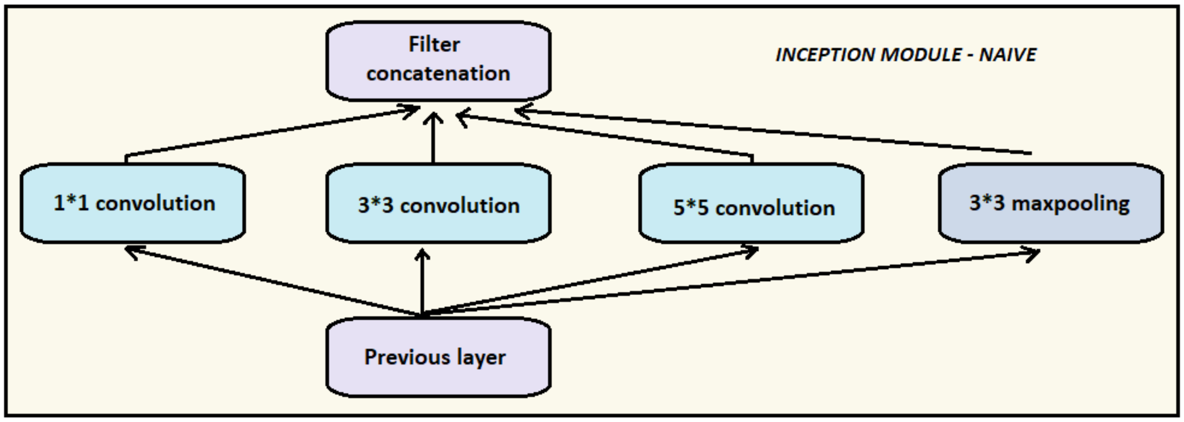
1.9.3 Base paper implementation results

The code from base paper was run and the following results were obtained on DIV2k dataset across increasing epochs.



1.10 INCEPTION NETWORK

Before inception network, in CNN layers were simply stacked over each other. But different images showing same object, say a cat, held information in varying locations. These variations were large. So, a large kernel that held more information was globally used, while for local use a smaller kernel was used. Deep networks that are too deep are prone to overfitting and passing gradient updates through the entire network becomes hard. One simple solution was to have multiple size filers applied on same level. This increases the width of the network instead of its depth. So, an inception module was designed to do this. This also reduces the overall cost of performing the operations. An inception module with reduced dimensions called GoogLeNet was built. This was the first version of inception network. The network was pretty deep and in order to prevent the middle part of the network from fading away, two auxiliary classifiers were introduced. Softmax was applied to the output of both the modules and an auxiliary loss over the same labels were computed. The total loss was the weighted sum of auxiliary loss and the actual loss. Below is the block diagram of the inception net used in the project.



1.11 Project implementation

In this project the base paper implementation was used along with some modification to the generator block.

1.11.1 Generator

Pre-Inception block

A convolution layer with kernel size as 9, number of filters as 64 and strides as 1.

Inception block

The first inception layer, d1 is a convolution layer and receives the output of pre inception layer as its input, and the number of filters is 64, kernel size is 1, the number of strides is set to 1, padding is kept as same and the activation is 'relu'.

The second inception layer, d1 is a convolution layer and receives the output of d1 layer as its input, and the number of filters is 64, kernel size is 3, the number of strides is set to 1, padding is kept as same and the activation is 'relu'.

The third inception layer, d2 is a convolution layer and receives the output of pre inception layer as its input, and the number of filters is 64, kernel size is 1, the number of strides is set to 1, padding is kept as same and the activation is 'relu'.

The fourth inception layer, d2 is a convolution layer and receives the output of d2 layer as its input, and the number of filters is 64, kernel size is 5, the number of strides is set to 1, padding is kept as same and the activation is 'relu'.

The next layer, d3 is a max-pooling layer and receives the output of pre inception layer as its input, and the number of filters is 64, kernel size is 3, the number of strides is set to 1, padding is kept as same.

The last inception layer, d3 is a convolution layer and receives the output of d2 layer as its input, and the number of filters is 64, kernel size is 1, the number of strides is set to 1, padding is kept as same and the activation is 'relu'.

Finally, the layer d takes the merged version of d1, d2 and d3 using the concatenate function of keras.layer and this is the layer that is returned from the inception network.

Post-Inception block

A convolution layer with kernel size as 3, with 64 filters and stride as 1, followed by a de-convolution block with kernel size as 3 with 256 filters and number of strides as 1.

1.11.2 Discriminator network

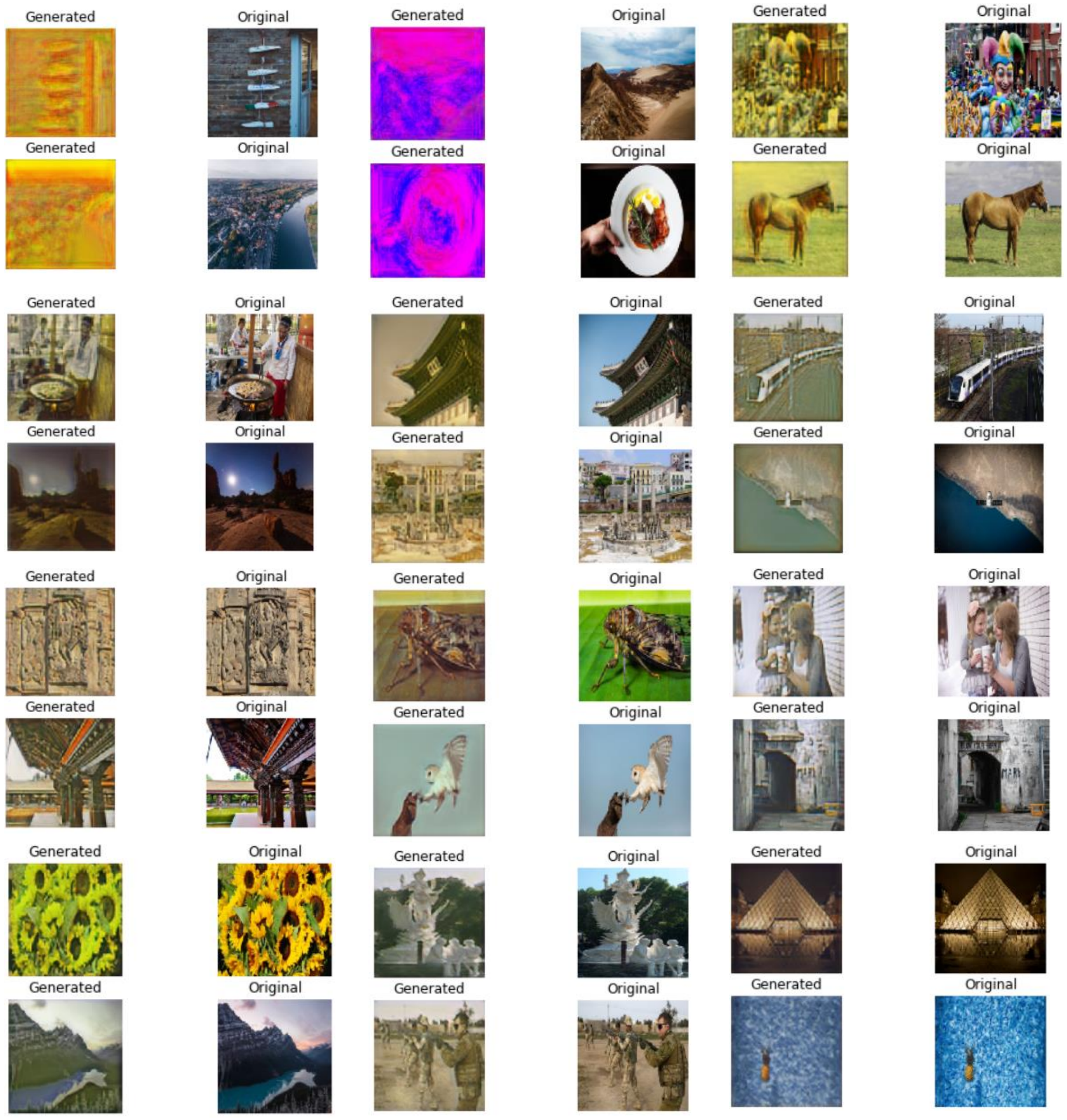
Used activation- LeakyReLU with ($\alpha = 0.2$).

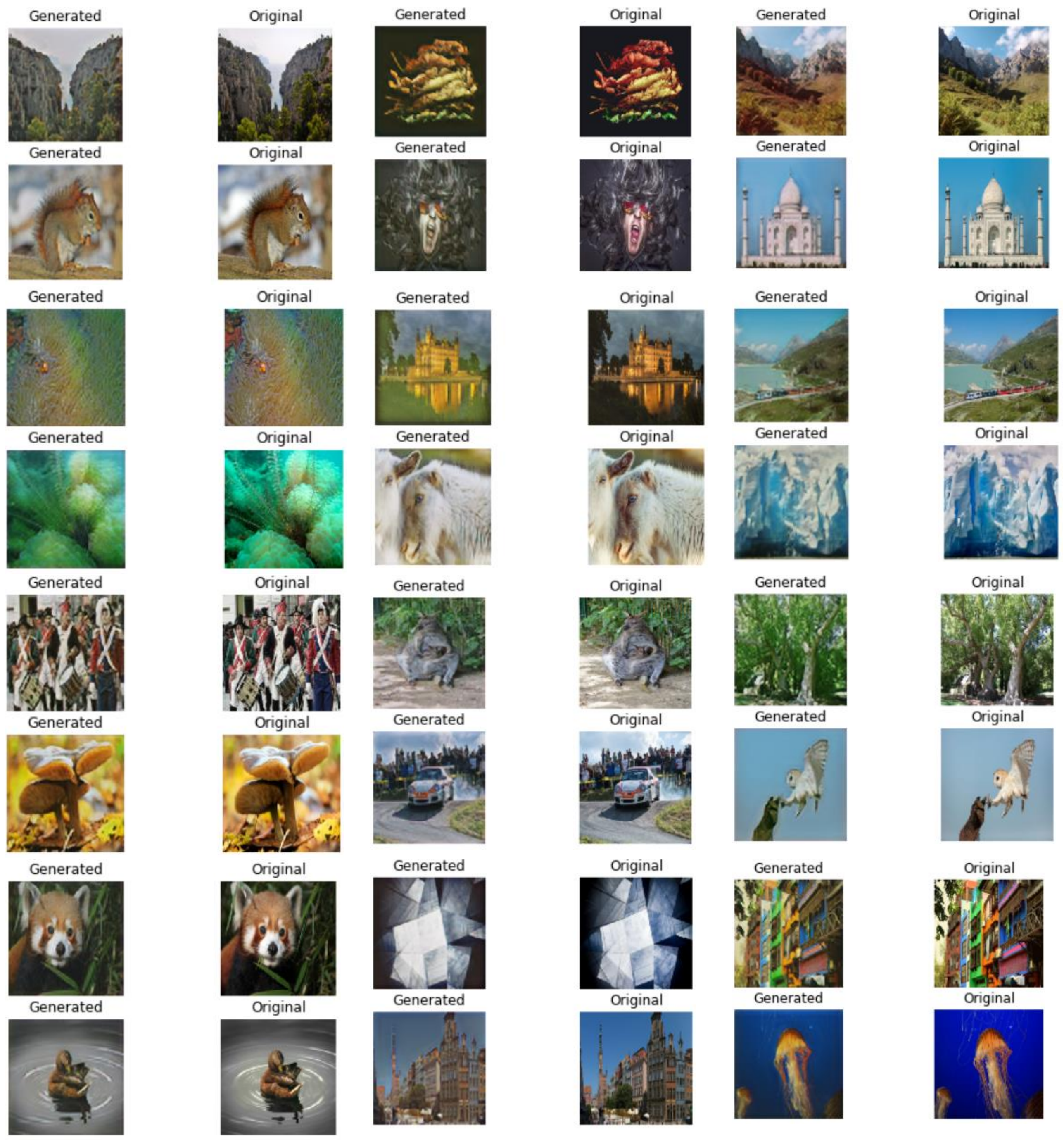
We also avoid max pooling throughout the network, as the discriminator is designed to solve maximization problem.

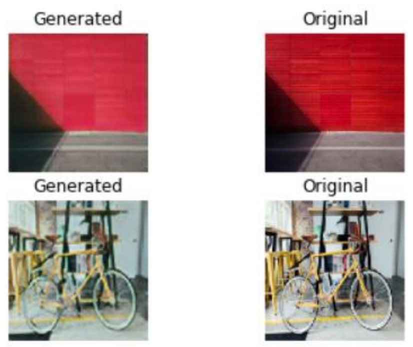
There are 8 convolution layers with 3*3 filter kernels. Each of the convolution layer has a 1 stride and 2 stride-convolution. So, 64 map features will have 2 layers, 128 has 2 layers, 256 has 2 layers and 512 has 2 layers with 1 stride and 2 strides respectively. Here number of map-features the number of filters used. The 8 convolution layers are followed by 2 dense layers. The activation function is sigmoid for the dense layers to obtain the probability for sample classification.

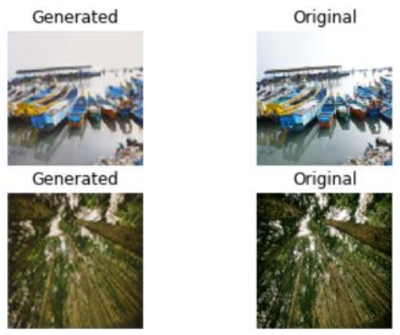
1.11.3 Our model with Inception Network implementation results

The code based on the base paper was modified by replacing the residual block by inception block with the architecture mentioned above and the following results were obtained across various epochs.




Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original
Generated
Original





1.12 Observation and Conclusion

Both inception net and residual network, capable of generating images that resembles the real image with enough details for humans to identify the image generated. In residual net we use skip connections. In inception net we use multiple size kernels at each layer attempting to capture finer details at different locations in an image using varying kernel sizes.

1.13 References

1) Project code available in the following link: https://github.com/AishwaryaMano/SRGAN-Inception-net/blob/master/SRGAN_Inceptionnet.ipynb
2) Base paper used: https://arxiv.org/pdf/1609.04802.pdf%20(CVPR%202017)
3) https://towardsdatascience.com/understanding-residual-networks-9add4b664b03
4) https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-networ
5) https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c
6) https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/
7) Quora
8) Wikipedia
9) Youtube
10) Kaggle
11) machinelearningmastery.com
12) Elitedatascience.com
13) stats.stackexchange.com/
14) nextjournal.com
15) docs.scipy.org