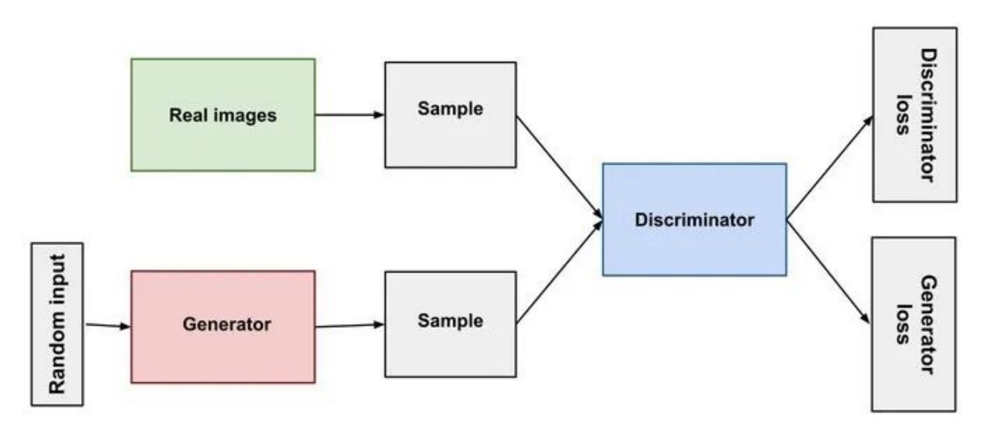
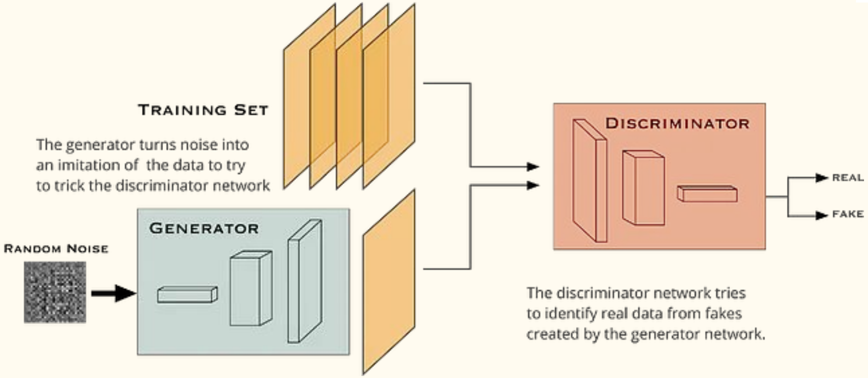
Single Image SRGAN

By: James Cheung

Generative Adversarial Networks

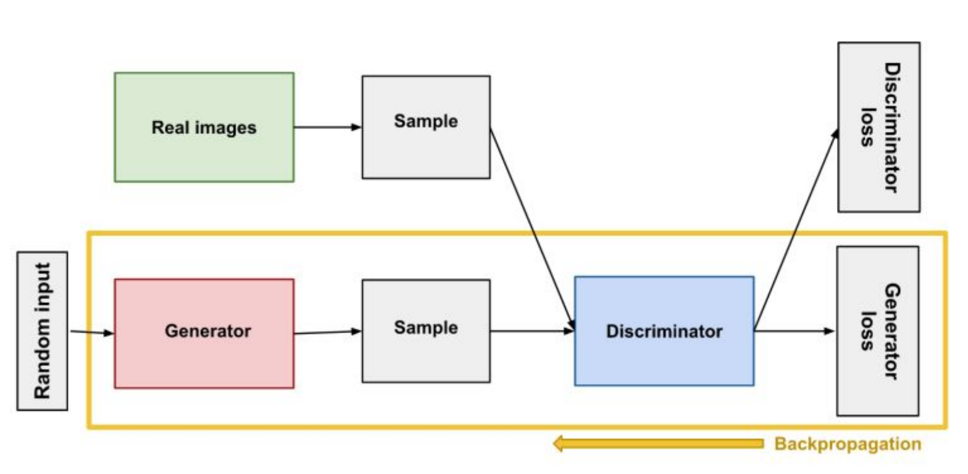




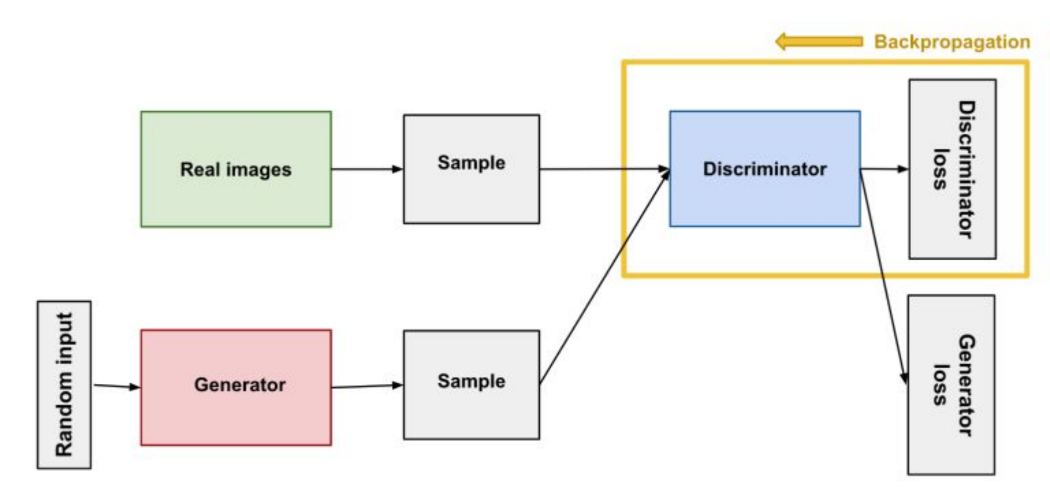
Generative Adversarial Networks Steps

1. From random Noise seed (latent vector) a Generator Network creates fake images
2. Real images are loaded
3. Images both real and fake are past through a Discriminator Network
4. Discriminator Network makes a binary classification decision trying to distinguish between real and fake images
5. Step 1-4 is looped and repeated: Re-trains the Generator Network / Discriminator Network depending on who wins and goes on for user-determined number of epochs
6. Generator model is saved to create new, realistic fake data

Generator Network

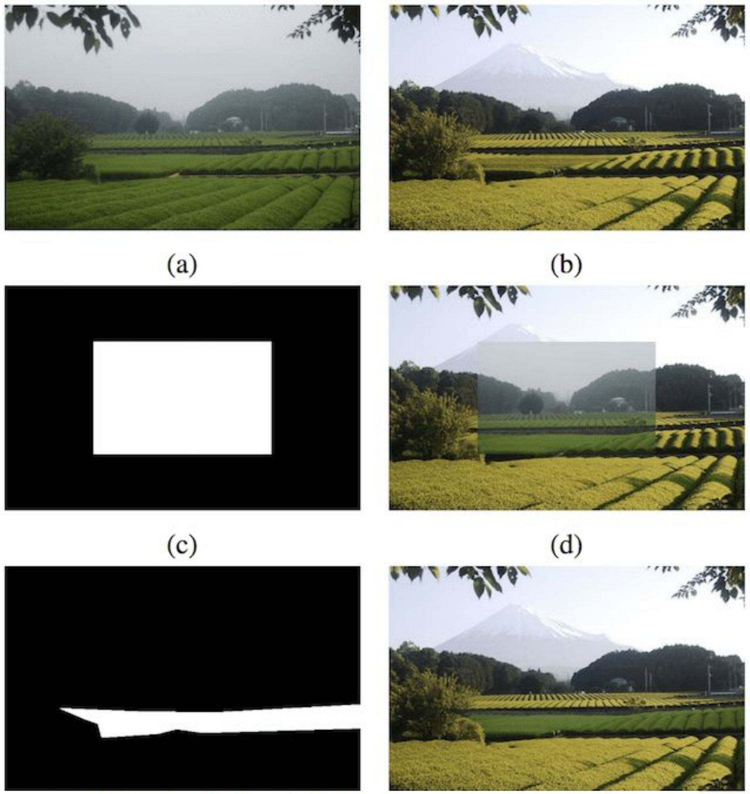


Discriminator Network



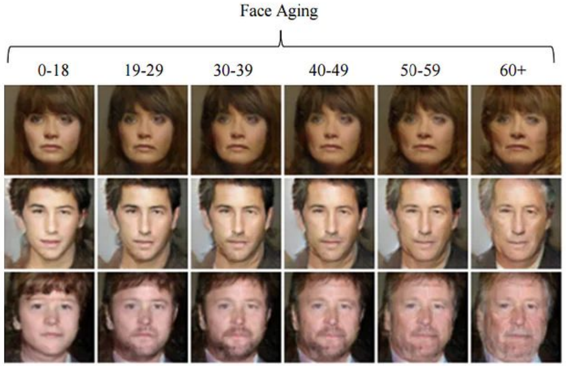
Applications of GAN:

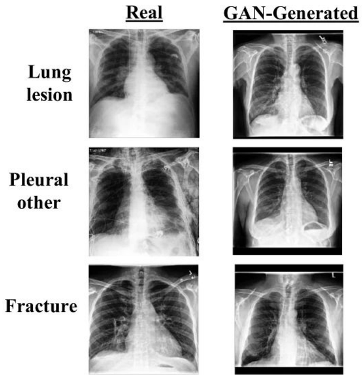
- StackGAN - text-to-image synthesis (e.g. DALL-E)
- DCGAN - generate fake human faces
- Conditional GANs - generate images of people aging
- CycleGAN - Photoblending
- CycleGAN - Image Style Transfer
- Medical Image Synthesis using GANs for Pulmonary Chest X-rays


(a)
(b)
(c)
(d)


Input labels
Synthesized image


this bird is red with white and has a very short beak


Face Aging
0-18
19-29
30-39
40-49
50-59
60+


Real
GAN-Generated
Lung lesion
Pleural other
Fracture

Super Resolution Generative Adversarial Network

- Using CNN and VGG19 in the model
- Used 250 pictures and 40 epochs
- Image from 32x32 to upscale to 128x128 i.e. 4x magnification
- Instead of calculating loss with Mean-Squared-Error which results in overly smooth textures, peceptiual loss function is used:
  - Content loss - uses Eucldean distance between feature representations
  - Adversarial loss - based on probability of the discriminator over all trainig samples

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

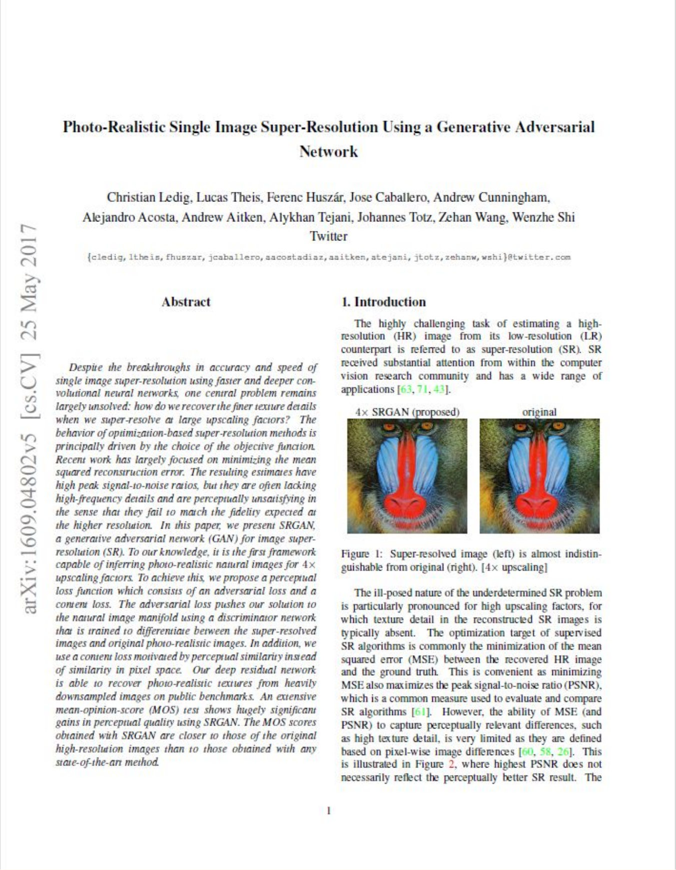


Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham,
Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi
Twitter
{cledig, ltheis, fhuszar, jcaballero, aacostadiaz, aaitken, atejani, jtotz, zehanw, wshi}@twitter.com

arXiv:1609.04802v5 [cs.CV] 25 May 2017

Abstract

Despite the breakthroughs in accuracy and speed of single image super-resolution using faster and deeper convolutional neural networks, one central problem remains largely unsolved: how do we recover the finer texture details when we super-resolve at large upscaling factors? The behavior of optimization-based super-resolution methods is principally driven by the choice of the objective function. Recent work has largely focused on minimizing the mean squared reconstruction error. The resulting estimates have high peak signal-to-noise ratios, but they are often lacking high-frequency details and are perceptually unsatisfying in the sense that they fail to match the fidelity expected at the higher resolution. In this paper, we present SRGAN, a generative adversarial network (GAN) for image super-resolution (SR). To our knowledge, it is the first framework capable of inferring photo-realistic natural images for 4× upscaling factors. To achieve this, we propose a perceptual loss function which consists of an adversarial loss and a content loss. The adversarial loss 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. In addition, we use a content loss motivated by perceptual similarity instead of similarity in pixel space. Our deep residual network is able to recover photo-realistic textures from heavily downsampled images on public benchmarks. An extensive mean-opinion-score (MOS) test shows hugely significant gains in perceptual quality using SRGAN. The MOS scores obtained with SRGAN are closer to those of the original high-resolution images than to those obtained with any state-of-the-art method.

1. Introduction

The highly challenging task of estimating a high-resolution (HR) image from its low-resolution (LR) counterpart is referred to as super-resolution (SR). SR received substantial attention from within the computer vision research community and has a wide range of applications [63, 71, 43].

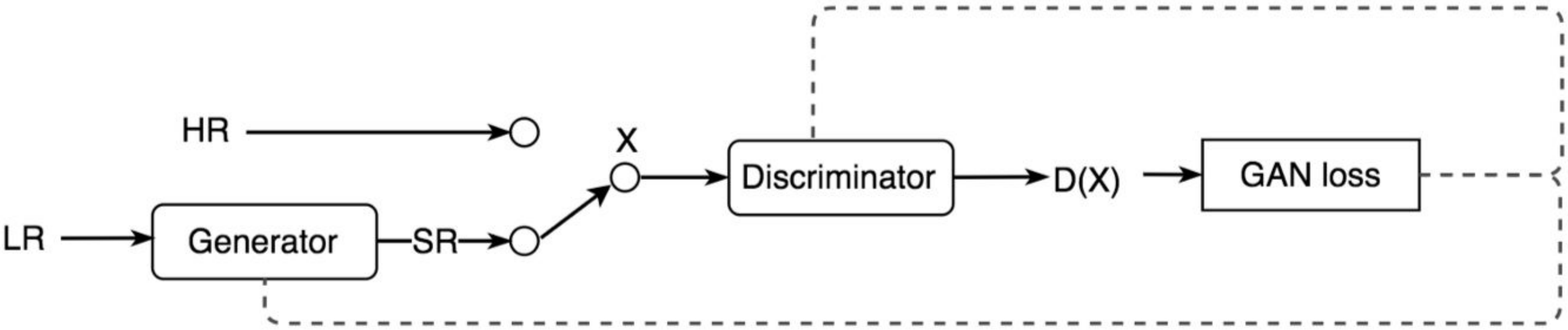
4× SRGAN (proposed) original

Figure 1: Super-resolved image (left) is almost indistinguishable from original (right). [4× upscaling]

The ill-posed nature of the underdetermined SR problem is particularly pronounced for high upscaling factors, for which texture detail in the reconstructed SR images is typically absent. The optimization target of supervised SR algorithms is commonly the minimization of the mean squared error (MSE) between the recovered HR image and the ground truth. This is convenient as minimizing MSE also maximizes the peak signal-to-noise ratio (PSNR), which is a common measure used to evaluate and compare SR algorithms [61]. However, the ability of MSE (and PSNR) to capture perceptually relevant differences, such as high texture detail, is very limited as they are defined based on pixel-wise image differences [60, 58, 26]. This is illustrated in Figure 2, where highest PSNR does not necessarily reflect the perceptually better SR result. The

Steps of SRGAN

1. A high-resolution image (HR) is downsampled to a low-resolution image (LR)
2. A GAN generator upsamples LR images to super-resolution images (SR)
3. We use a discriminator to distinguish the HR images and backpropagate the GAN loss to train the discriminator and the generator



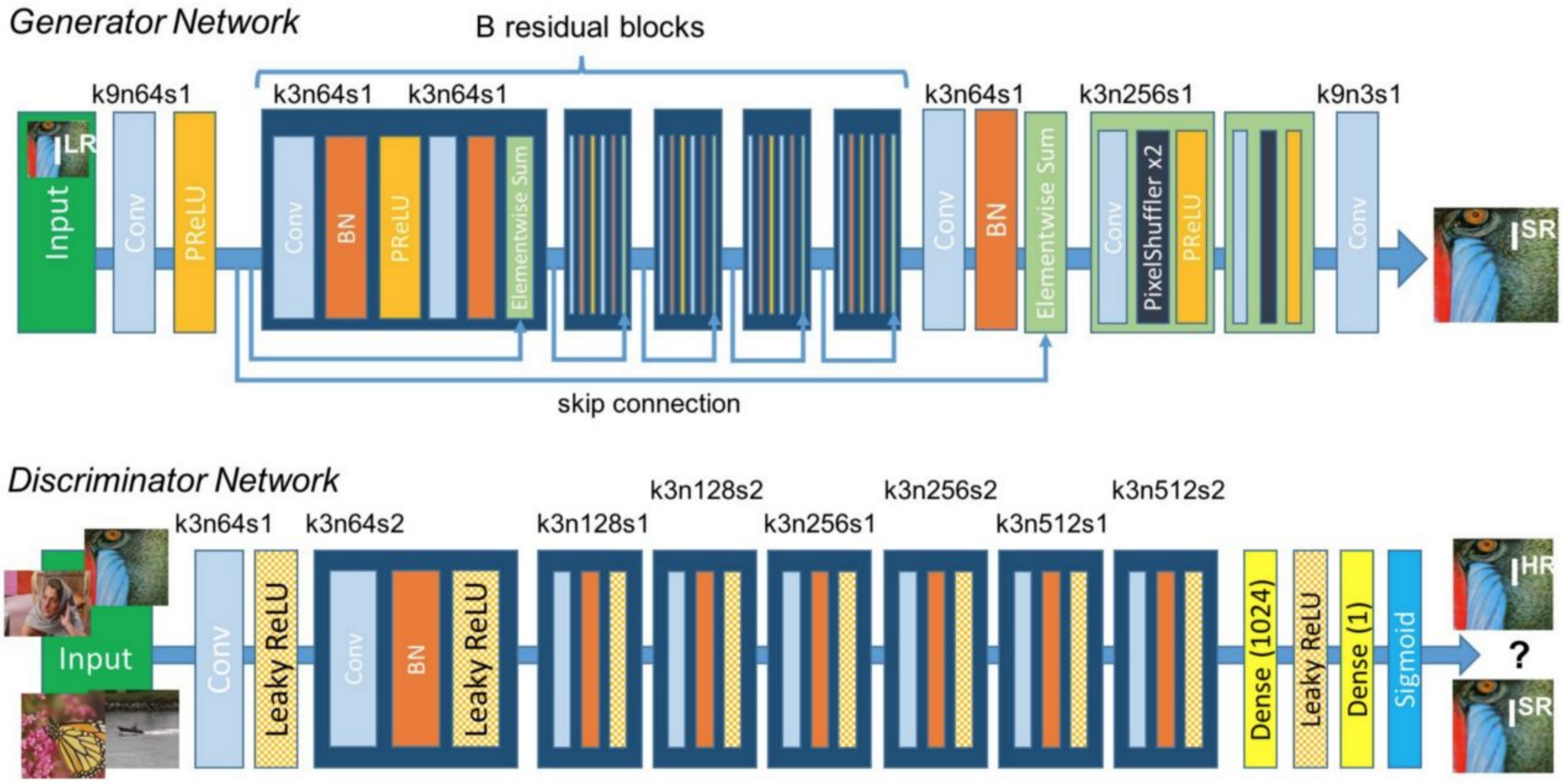


Figure 4: Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

The Generator Network

```
#Define blocks to build the generator
def res_block(ip):

    res_model = Conv2D(64, (3,3), padding = "same")(ip)
    res_model = BatchNormalization(momentum = 0.5)(res_model)
    res_model = PReLU(shared_axes = [1,2])(res_model)

    res_model = Conv2D(64, (3,3), padding = "same")(res_model)
    res_model = BatchNormalization(momentum = 0.5)(res_model)

    return add([ip,res_model])
```

```
def upscale_block(ip):

    up_model = Conv2D(256, (3,3), padding="same")(ip)
    up_model = UpSampling2D( size = 2 )(up_model)
    up_model = PReLU(shared_axes=[1,2])(up_model)

    return up_model
```

```
num_res_block=16

#Generator model
def create_gen(gen_ip, num_res_block):
    layers = Conv2D(64, (9,9), padding="same")(gen_ip)
    layers = PReLU(shared_axes=[1,2])(layers)

    temp = layers

    for i in range(num_res_block):
        layers = res_block(layers)

    layers = Conv2D(64, (3,3), padding="same")(layers)
    layers = BatchNormalization(momentum=0.5)(layers)
    layers = add([layers,temp])

    layers = upscale_block(layers)
    layers = upscale_block(layers)

    op = Conv2D(3, (9,9), padding="same")(layers)

    return Model(inputs=gen_ip, outputs=op)
```

Discriminator Network

```python
def discriminator_block(ip, filters, strides=1, bn=True):

    disc_model = Conv2D(filters, (3,3), strides = strides, padding="same")(ip)

    if bn:
        disc_model = BatchNormalization( momentum=0.8 )(disc_model)

    disc_model = LeakyReLU( alpha=0.2 )(disc_model)

    return disc_model
```

```python
#Descriminartor, as described in the original paper
def create_disc(disc_ip):

    df = 64

    d1 = discriminator_block(disc_ip, df, bn=False)
    d2 = discriminator_block(d1, df, strides=2)
    d3 = discriminator_block(d2, df*2)
    d4 = discriminator_block(d3, df*2, strides=2)
    d5 = discriminator_block(d4, df*4)
    d6 = discriminator_block(d5, df*4, strides=2)
    d7 = discriminator_block(d6, df*8)
    d8 = discriminator_block(d7, df*8, strides=2)

    d8_5 = Flatten()(d8)
    d9 = Dense(df*16)(d8_5)
    d10 = LeakyReLU(alpha=0.2)(d9)
    validity = Dense(1, activation='sigmoid')(d10)

    return Model(disc_ip, validity)
```

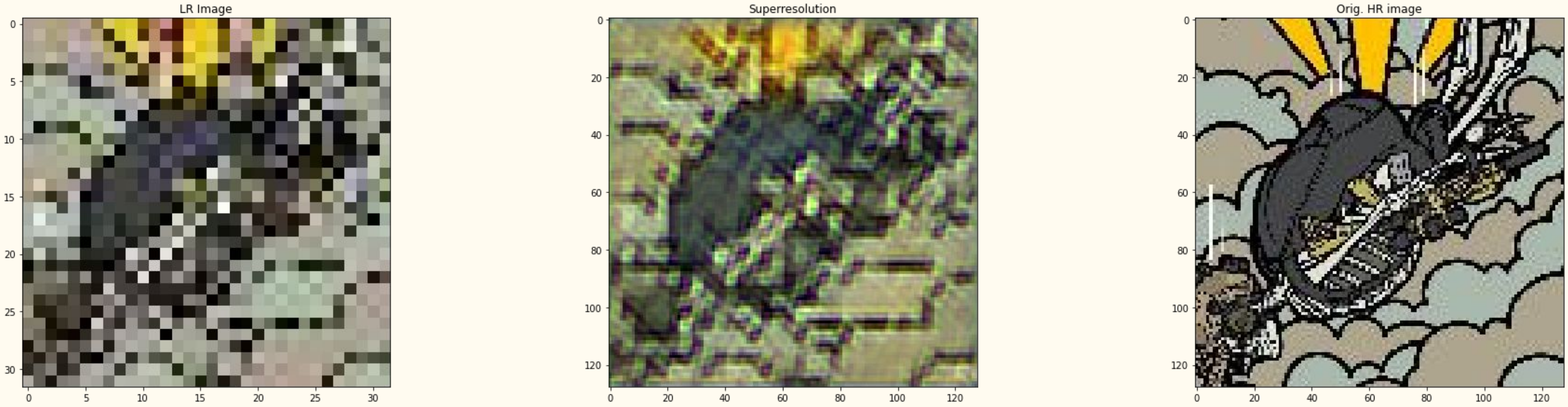
Compile Model

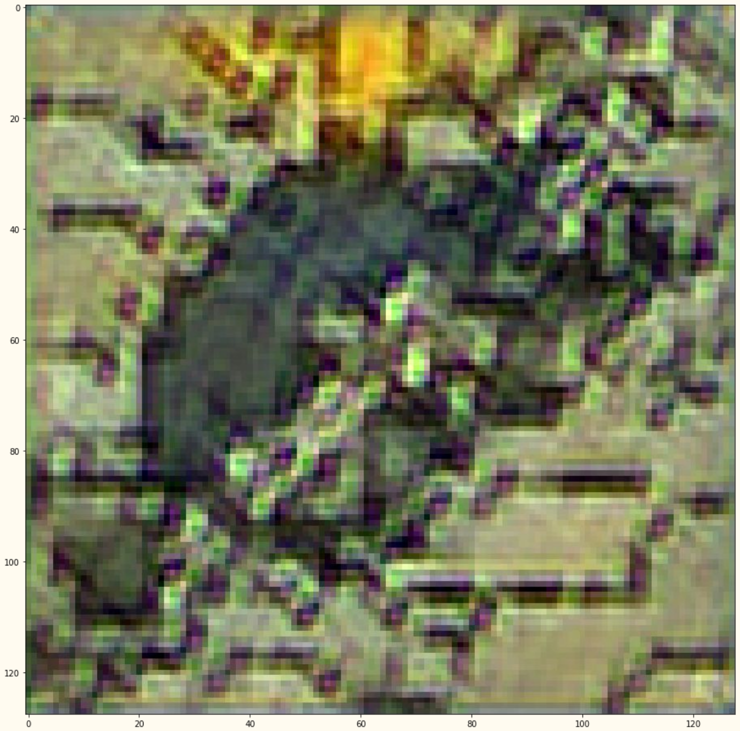
- VGG19 - weights using “imagenet” & selected layers
- low resolution image 32x32
- high resolution image 128x128
- compile using “binary crossentropy”
- For each epoch:
    - Train Discriminator - two losses are captured after supplying mix of fake lr images and real lr to the generator, and the the two losses are averaged
    - Train Generator - halt the discriminator and get the image features from vgg prediction on hr images to get the generative loss

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv4 (Conv2D)       (None, 32, 32, 256)       590080

=================================================================
Total params: 2,325,568
Trainable params: 2,325,568
Non-trainable params: 0
_________________________________________________________________
None
```

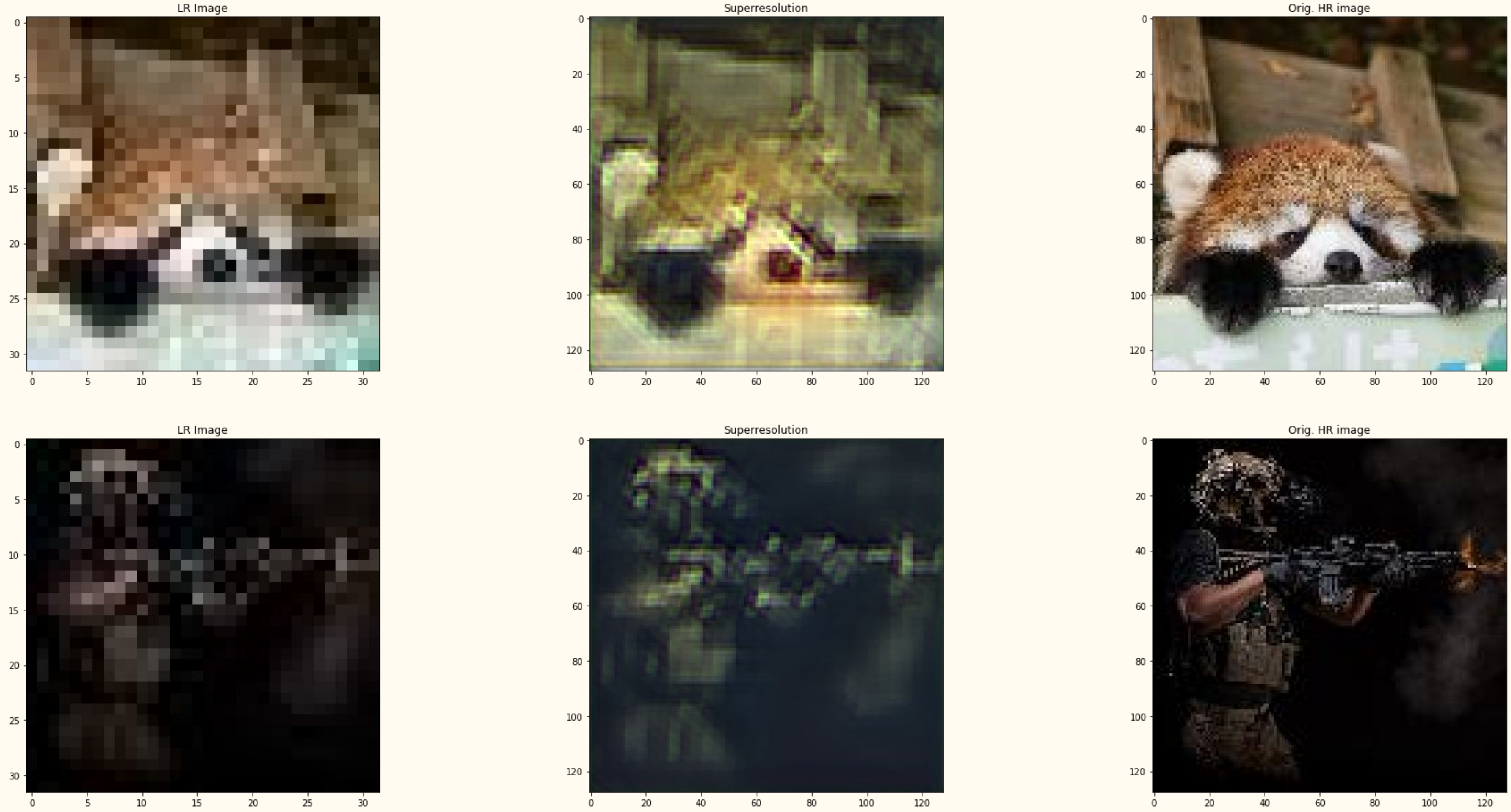
```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 32, 32, 3)]  0           []

 model (Functional)             (None, 128, 128, 3)  2044291     ['input_1[0][0]']

 input_2 (InputLayer)           [(None, 128, 128, 3  0           []
                                )]

 model_1 (Functional)           (None, 1)            38249281    ['model[0][0]']

 model_2 (Functional)           (None, 32, 32, 256)  2325568     ['model[0][0]']

==================================================================================================
Total params: 42,619,140
Trainable params: 2,040,067
Non-trainable params: 40,579,073
__________________________________________________________________________________________________
100%|██████████████| 167/167 [12:00<00:00,  4.32s/it]
epoch: 1 g_loss: 127.59544988163931 d_loss: [6.02951705 0.66167665]
100%|██████████████| 167/167 [11:46<00:00,  4.23s/it]
epoch: 2 g_loss: 86.76631530030758 d_loss: [0.6692774  0.89221557]
100%|██████████████| 167/167 [11:56<00:00,  4.29s/it]
epoch: 3 g_loss: 81.03504640042425 d_loss: [0.77218723 0.88323353]
100%|██████████████| 167/167 [11:57<00:00,  4.30s/it]
epoch: 4 g_loss: 74.73745766942372 d_loss: [2.22943178 0.75149701]
100%|██████████████| 167/167 [12:02<00:00,  4.33s/it]
epoch: 5 g_loss: 71.64772969091723 d_loss: [0.80506992 0.86526946]
100%|██████████████| 167/167 [12:14<00:00,  4.40s/it]
epoch: 6 g_loss: 69.24715004424135 d_loss: [0.86259081 0.84431138]
100%|██████████████| 167/167 [11:58<00:00,  4.30s/it]
epoch: 7 g_loss: 68.03483305148735 d_loss: [0.47137236 0.87724551]
100%|██████████████| 167/167 [12:06<00:00,  4.35s/it]
epoch: 8 g_loss: 66.65545580629816 d_loss: [0.52786384 0.9011976 ]
100%|██████████████| 167/167 [11:38<00:00,  4.18s/it]
epoch: 9 g_loss: 67.89887893128538 d_loss: [0.98622361 0.87125749]
100%|██████████████| 167/167 [11:21<00:00,  4.08s/it]epoch: 10 g_loss: 64.490
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to
```

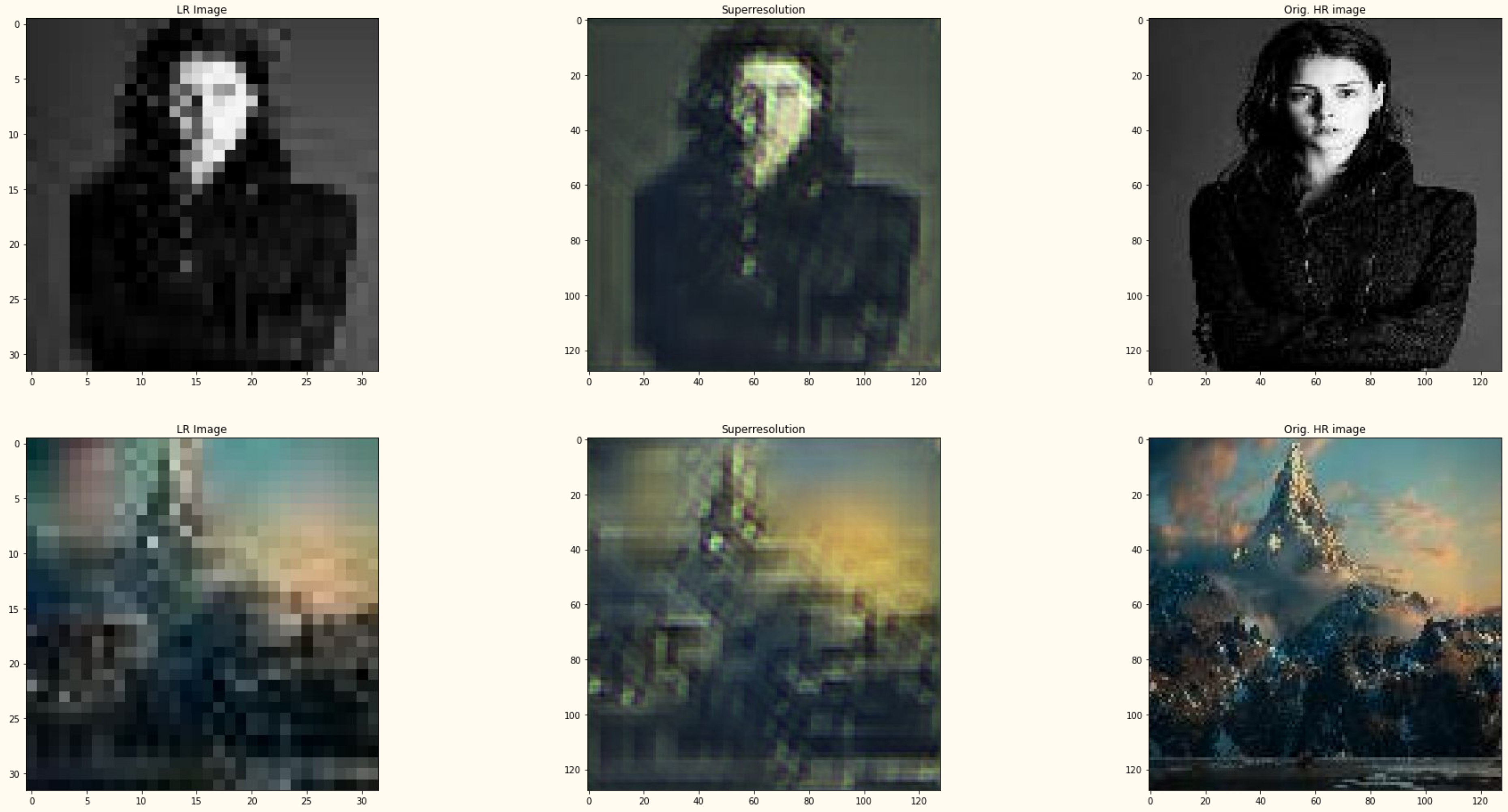
Results 1:




0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120


LR Image
Superresolution
Orig. HR image
LR Image
Superresolution
Orig. HR image


LR Image
Superresolution
Orig. HR image
LR Image
Superresolution
Orig. HR image

~The End~

Questions?