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

Project submitted by

Abhinav Pavithran 17EC202
Nadeem Roshan 17EC226
Saurav Vinil 17EC246

Under the guidance of

Dr. Sandeep Kumar

In partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL, MANGALORE - 575025

April , 2021

Abstract

Single image super-resolution (SR) , which aims at recovering a high-resolution image from a single lowresolution image, is a classical problem in computer vision. This problem is inherently ill-posed since a multiplicity of solutions exist for any given low-resolution pixel. In other words, it is an underdetermined inverse problem, of which solution is not unique. Such a problem is typically mitigated by constraining the solution space by strong prior information. To learn the prior, recent state-of-the-art methods mostly adopt the example-based strategy

In this paper, we present SRGAN, a Generative Adversarial Network (GAN) for Photo Realistic Single Image Super-resolution (SR). To our knowledge, it is the first framework capable of inferring photo-realistic natural images for a 4× upscaling factor. 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. We also look into ESRGAN (Enhanced Super Resolution Generative Adversarial Network) which is a more improved version of SRGAN and is known to give better results.

keyword = **Keras, SRGAN , ESRGAN , Neural network**

Contents

List of Figures

1 INTRODUCTION

In this project we have looked into different architectures for image super resolution and compare the models on the dataset that we have used to train and test the algorithms. By using Tensorflow and Keras framework, we are able to obtain a working code with the help of some references. The algorithm we used has a pixel accuracy of roughly 95% and we were able to get favourable outcomes for most of the cases. By further trying new models we can compare and deduce which architecture has the best result.

1.1 Problem definition

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. 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. We can observe this loss in detail when switching from 2x to 4x upscaling factor on CNN models without GAN. 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.

1.2 Motivation

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.

1.3 Overview

In this report we will show the results obtained by 6 architectures, i)SRCNN ii)ESRCNN iii)DSRCNN iv)DDSRCNN v)SRResNet vi)SRGAN and compare each model. Our objective is to obtain the best model to obtain a photo-realistic output rather than one with high pixel-accuracy but missing details.

2 Related work

2.1 Image super-resolution

Super-resolution is based on the idea that a combination of low resolution (noisy) sequence of images of a scene can be used to generate a high resolution image or image sequence. Thus it attempts to reconstruct the original scene image with high resolution given a set of observed images at lower resolution. The general approach considers the low resolution images as resulting from resampling of a high resolution image. The goal is then to recover the high resolution image which when resampled based on the input images and the imaging model, will produce the low resolution observed images. Thus the accuracy of imaging model is vital for super-resolution and an incorrect modeling, say of motion, can actually degrade the image further.

Single image super-resolution (SISR) is a notoriously challenging ill-posed problem that aims to obtain a highresolution (HR) output from one of its low-resolution (LR) versions. Recently, powerful deep learning algorithms have been applied to SISR and have achieved state-of-the-art performance.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNN) date back decades and deep CNNs have recently shown an explosive popularity partially due to its success in image classification . They have also been successfully applied to other computer vision fields, such as object detection , face recognition and pedestrian detection . Several factors are of central importance in this progress:
(i) the efficient training implementation on modern powerful GPUs
(ii) the proposal of the Rectified Linear Unit (ReLU) which makes convergence much faster while still presents good quality
(iii) the easy access to an abundance of data (like ImageNet) for training larger models.
Our method also benefits from these progresses.

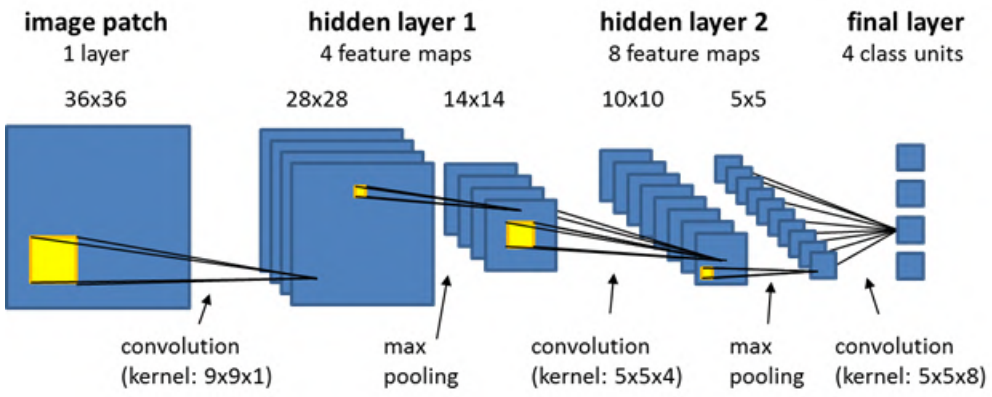

Figure 1: Conolutional Neural Network

3 Methodology

For this project we have done it in Google colab which is an open source platform that allows any user to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. We have made use of tensorflow and keras library in order to execute the program.

Traditional super-resolution network models such as SRCNN often employ 2D convolution layers to extract the spatial features of RGB images. These layers pay little attention to the correlation between image bands, thus ignoring spectral information. Therefore, this type of network is suitable only for images with few bands, such as RGB images or single-channel images. If we want to preserve spectral information when dealing with HSI SR problems, we must not only consider the spatial relation of the neighboring pixels, but also the spectral similarity of the neighboring bands. The idea of generating confrontation in SRGAN is very conducive to the detailed reconstruction of images.

3.1 Architectures of proposed models

We will be looking into the six architectures each increasing in its complexity and whose results will provide insight into the motivation behind GAN for SIngle Image Super-Resolution.

3.1.1 Super Resolution CNN (SRCNN)

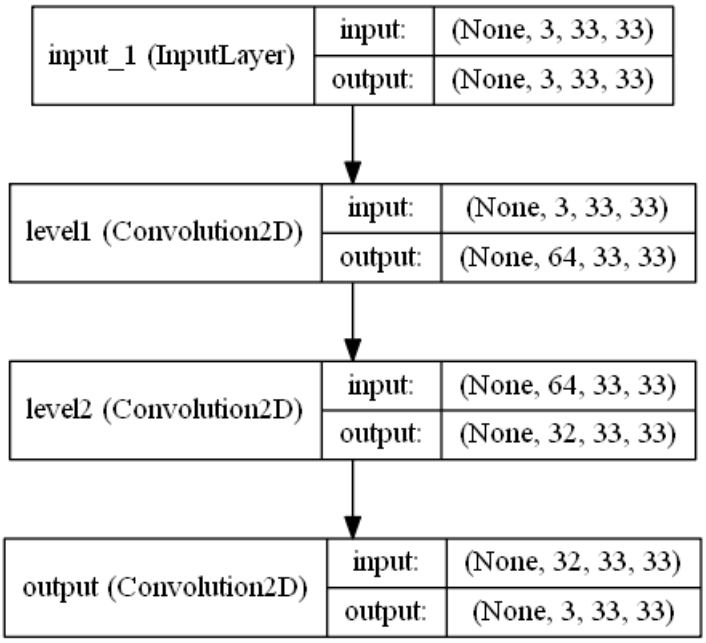

Figure 2: SRCNN Architecture

The model above is the simplest model of the ones implemented, consisting of the 9-1-5 model. Larger architectures can be easily made, but come at the cost of execution time, especially on CPU. This model contains some 21,000 parameters, more than the 8,400 of the original paper.It performs well, however images are slightly noisy at 4x upscaling.

3.1.2 Expanded Super Resolution CNN (ESRCNN)

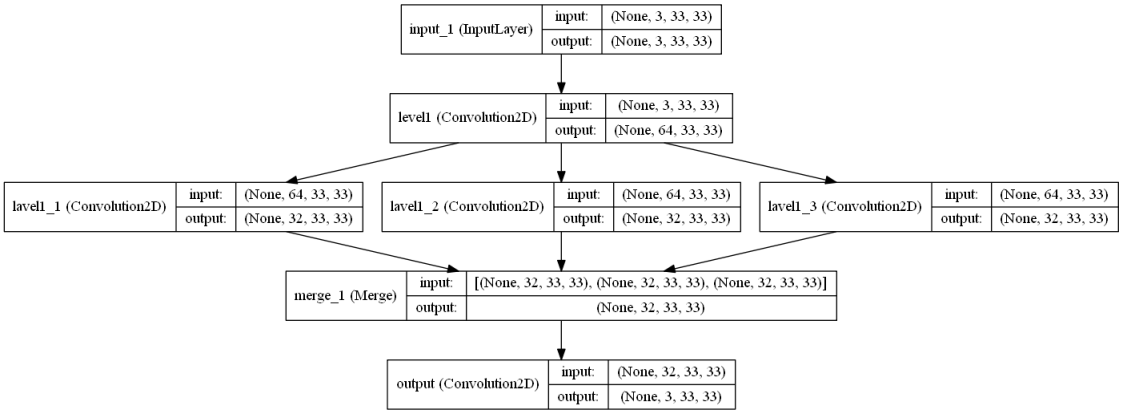

Figure 3: ESRCNN Architecture

The above is called "Expanded SRCNN", which performs slightly worse than the default SRCNN model on the dataset (PSNR 25.94 dB vs 27 dB for 2x upscaling). The "Expansion" occurs in the intermediate hidden layer. The outputs of this layer are then averaged, in order to construct more robust upscaled images.

3.1.3 Denoiseing (Auto Encoder) Super Resolution CNN (DSRCNN)

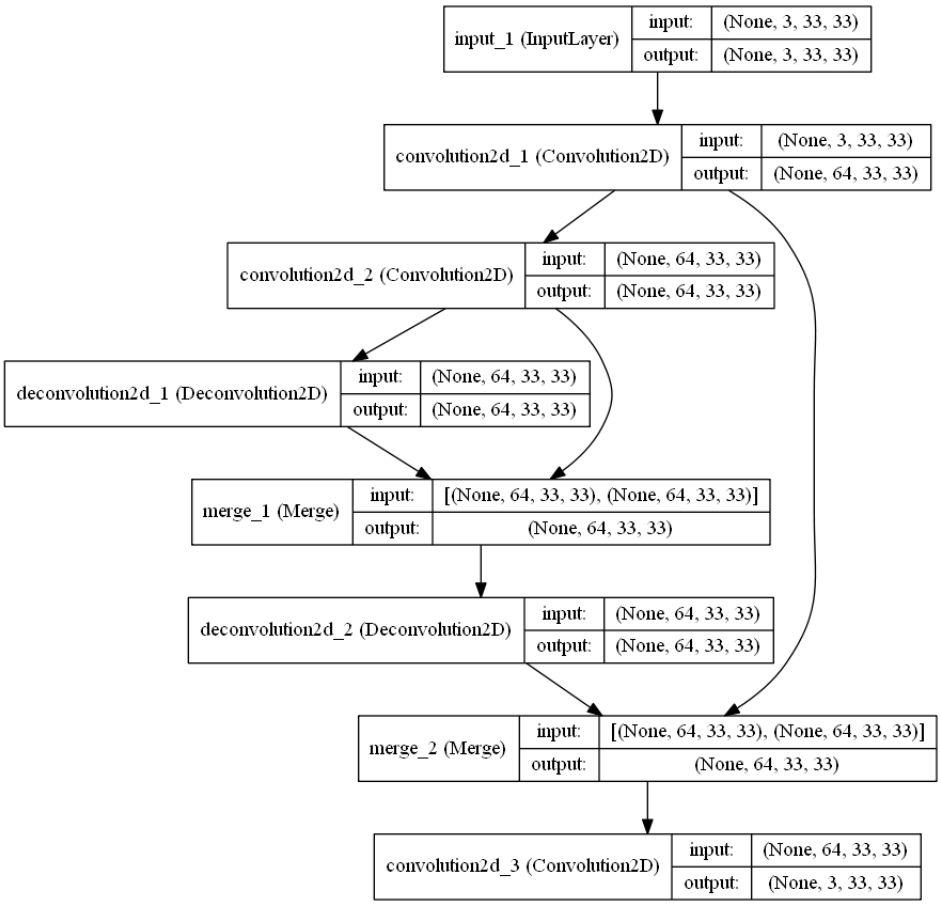

Figure 4: DSRCNN Architecture

The above is the "Denoiseing Auto Encoder SRCNN".

This model uses bridge connections between the convolutional layers of the same level in order to speed up convergence and improve output results. The bridge connections are averaged to be more robust.

Since the training images are passed through a gausian filter (sigma = 0.5), then downscaled to 1/4th the size, then upscaled to the original size images, the images can be considered "noisy". Thus, this auto encoder quickly improves on the earlier results, and reduces the noisy output image problem faced by the simpler SRCNN model.

3.1.4 Deep Denoiseing Super Resolution (DDSRCNN)

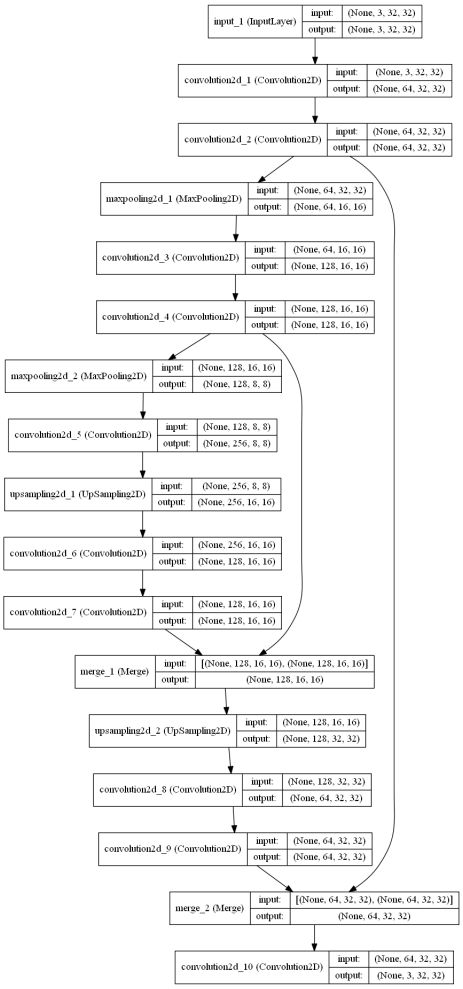

Figure 5: DDSRCNN Architecture

The above is the "Deep Denoiseing SRCNN", which is a modified form of the architecture described in the paper "Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections" applied to image super-resolution. It can perform far better than even the Denoiseing SRCNN.

Similar to the paper Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections, this can be considered a highly simplified and shallow model compared to the 30 layer architecture used in the above paper.

3.1.5 ResNet Super Resolution (SRResNet)

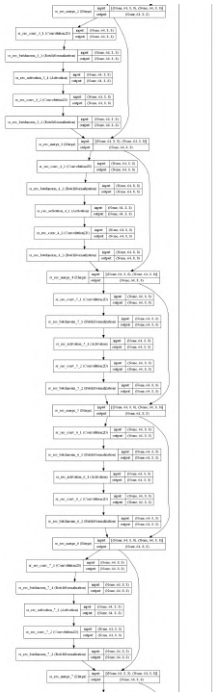

Figure 6: SRResNet Architecture

The above is the "ResNet SR" model, derived from the "SRResNet" model of the paper [Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network]
Currently uses only 6 residual blocks.

3.1.6 Super Resolution Generative Adversarial Network (SRGAN)

Similar to GAN architectures, the Super Resolution GAN also contains two parts Generator and Discriminator where generator produces some data based on the probability distribution and discriminator tries to guess weather data coming from input dataset or generator. Generator than tries to optimize the genrated data so that it can fool the discriminator. Below are the generator and discriminator architectural details:

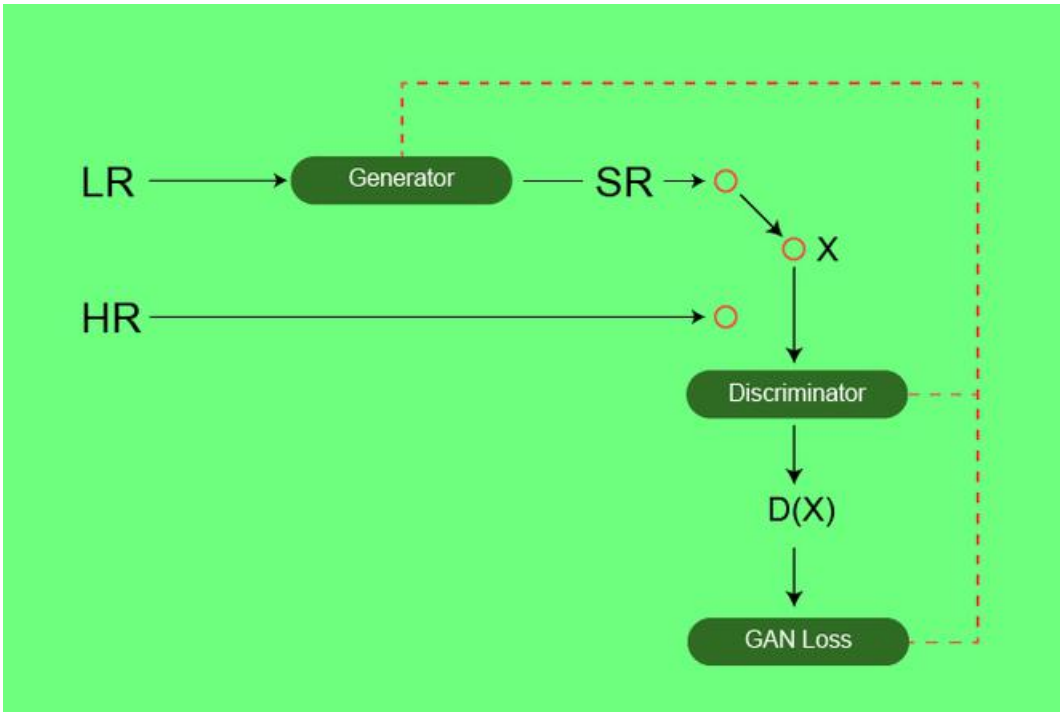

Figure 7: SRGAN Architecture

Generator Architecture:

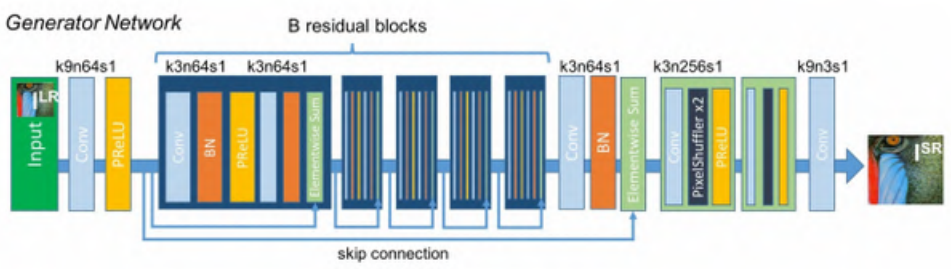

Figure 8: SRGAN Generator Architecture

Discriminator Architecture:

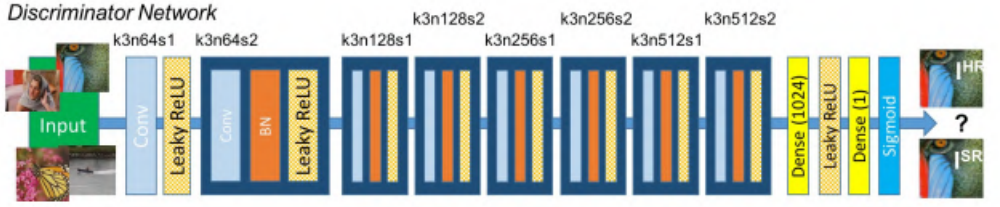

Figure 9: SRGAN Discriminator Architecture

3.1.7 Enhanced Super Resolution Generative Adversarial Network (ESRGAN)

To further enhance the visual quality, we look into three key components of SRGAN - network architecture, adversarial loss and perceptual loss, and improve each of them to derive an Enhanced SRGAN (ESRGAN).
How ESRGAN actually functions is by looking for colour variation between an image's pixels, and then inserting the detail that it thinks should exist based on the images it's seen before. When trained on photographs, ESRGAN inserts tons of details into the large pixels typically seen in older photos.
The SRGAN is improved from three aspects:

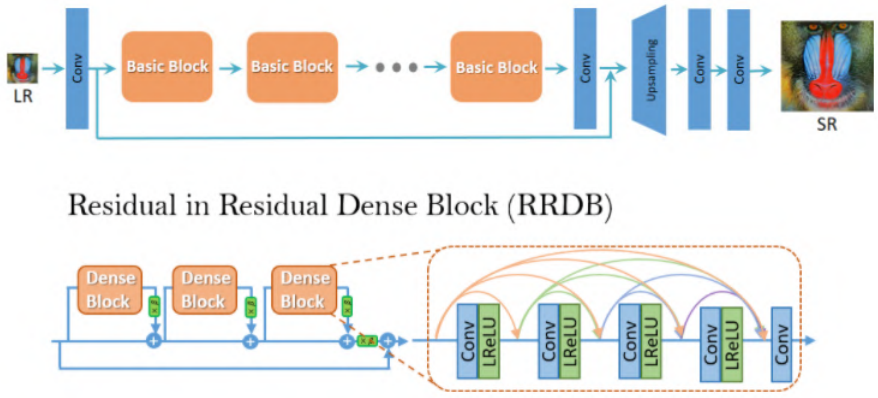

Figure 10: ESRGAN Architecture

1.Adopt a deeper model using Residual-in-Residual Dense Block (RRDB) without batch normalization layers.
2.Employ Relativistic average GAN instead of the original GAN.
3.Improve the perceptual loss by using the features before activation.

3.2 Training

All the models were trained on the BSD100 dataset using tensorflow for 150 epochs. Training was done on GoogleColab utilizing GPU. The SRGAN model was trained for around 38000 iterations after which the generated images looked fairly photo-realistic. The models were trained in two scenarios of Super-Resolution: 1) 2x Upscaling factor and 2) 4x upscaling factor.
We compare the SSIM values between the models. Note that we can apply the generator model to images of arbitrary size as it is fully convolutional. For optimization we use Adam with beta1 = 0.9.
Some of the images from BSD100:

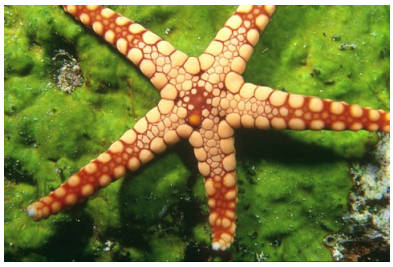

Figure 11: Image 1

Figure 12: Image 2

Figure 13: Image 3

Figure 14: Image 4

4 Results

We have obtained results for two cases:-
i) 2x-Upscaling - Where a 240x160 low resolution input images is converted to 480x320 high resolution images.
ii) 4x-Upscaling - Where a 120x80 low resolution input images is converted to 480x320 high resolution images.

4.1 SRCNN

4.1.1 2x

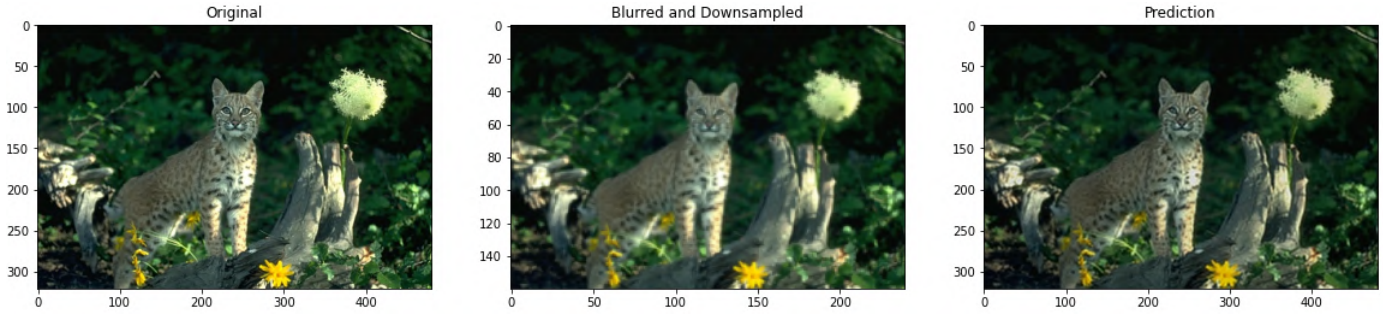

Figure 15: SRCNN Image results

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 160, 240, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 160, 240, 32)      18464
_________________________________________________________________
resizing (Resizing)          (None, 321, 481, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 321, 481, 3)       867
=================================================================
Total params: 21,123
Trainable params: 21,123
Non-trainable params: 0
_________________________________________________________________
```

Figure 16: SRCNN Model summary

Average PSNR	27
Average MSE loss	117.0433
Accuracy	0.9280

4.1.2 4x

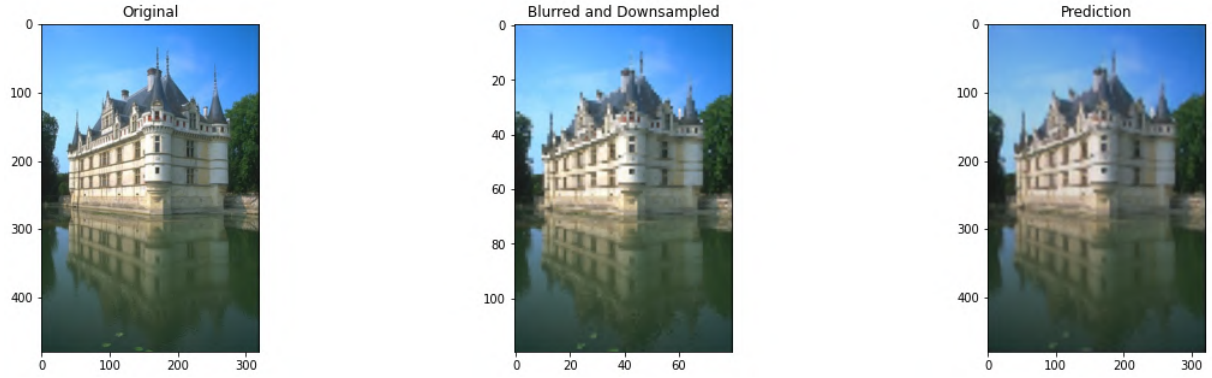

Figure 17: SRCNN Image results

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_32 (Conv2D)           (None, 120, 80, 64)       1792
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 120, 80, 32)       18464
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 120, 80, 3)        867
_________________________________________________________________
up_sampling2d_8 (UpSampling2 (None, 240, 160, 3)       0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 240, 160, 64)      1792
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 240, 160, 32)      18464
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 240, 160, 3)       867
_________________________________________________________________
up_sampling2d_9 (UpSampling2 (None, 480, 320, 3)       0
=================================================================
Total params: 42,246
Trainable params: 42,246
Non-trainable params: 0
_________________________________________________________________
```

Figure 18: SRCNN Model summary

Average PSNR	23.12
Average MSE loss	257.8263
SSIM	0.6255

4.2 ESRCNN

4.2.1 2x

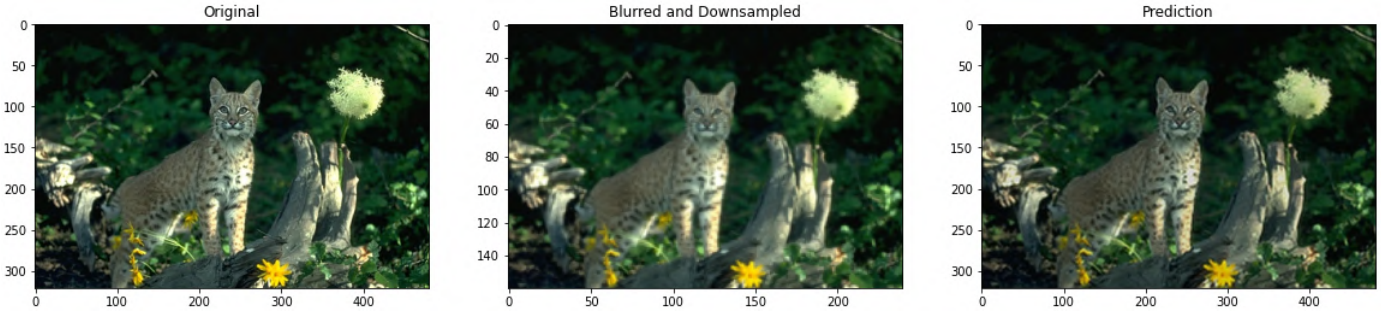

Figure 19: ESRCNN Image results

```
Model: "ESRCNN"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================================================
Img (InputLayer)             [(None, 160, 240, 3) 0
_________________________________________________________________________________________________
conv2d_3 (Conv2D)            (None, 160, 240, 64) 1792      Img[0][0]
_________________________________________________________________________________________________
conv2d_4 (Conv2D)            (None, 160, 240, 32) 18464     conv2d_3[0][0]
_________________________________________________________________________________________________
conv2d_5 (Conv2D)            (None, 160, 240, 32) 18464     conv2d_3[0][0]
_________________________________________________________________________________________________
conv2d_6 (Conv2D)            (None, 160, 240, 32) 18464     conv2d_3[0][0]
_________________________________________________________________________________________________
add (Add)                    (None, 160, 240, 32) 0         conv2d_4[0][0]
                                                            conv2d_5[0][0]
                                                            conv2d_6[0][0]
_________________________________________________________________________________________________
conv2d_7 (Conv2D)            (None, 160, 240, 3)  867       add[0][0]
_________________________________________________________________________________________________
resizing_1 (Resizing)        (None, 321, 481, 3)  0         conv2d_7[0][0]
=================================================================================================
Total params: 58,051
Trainable params: 58,051
Non-trainable params: 0
_________________________________________________________________________________________________
```

Figure 20: ESRCNN Model summary

Average PSNR	25.943
Average MSE loss	139.8332
Accuracy	0.9396

4.2.2 4x

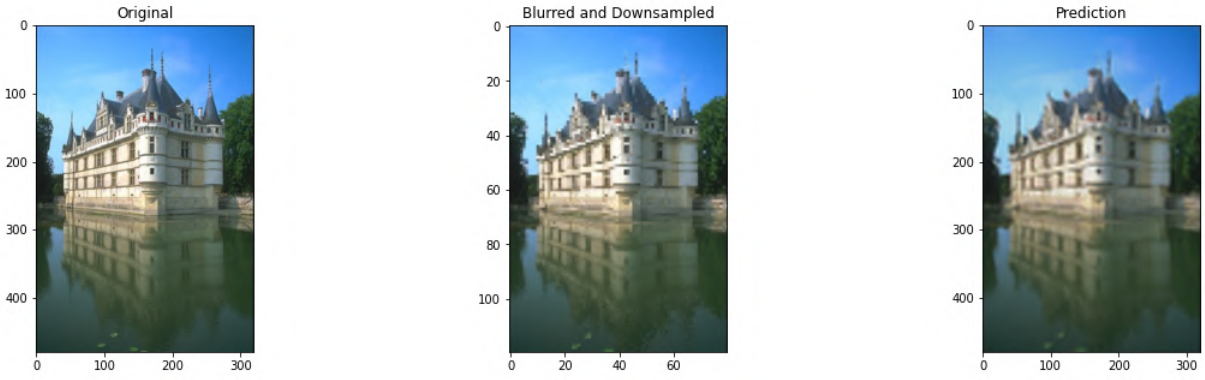

Figure 21: ESRCNN Image results

```
Model: "ESRCNN"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
Img (InputLayer)             [(None, 120, 80, 3)] 0
_________________________________________________________________________________________________
conv2d_43 (Conv2D)           (None, 120, 80, 64)  1792        Img[0][0]
_________________________________________________________________________________________________
conv2d_44 (Conv2D)           (None, 120, 80, 32)  51232       conv2d_43[0][0]
_________________________________________________________________________________________________
conv2d_45 (Conv2D)           (None, 120, 80, 32)  51232       conv2d_43[0][0]
_________________________________________________________________________________________________
conv2d_46 (Conv2D)           (None, 120, 80, 32)  51232       conv2d_43[0][0]
_________________________________________________________________________________________________
add_5 (Add)                  (None, 120, 80, 32)  0           conv2d_44[0][0]
                                                              conv2d_45[0][0]
                                                              conv2d_46[0][0]
_________________________________________________________________________________________________
conv2d_47 (Conv2D)           (None, 120, 80, 3)   867         add_5[0][0]
_________________________________________________________________________________________________
up_sampling2d_11 (UpSampling2D) (None, 240, 160, 3)  0           conv2d_47[0][0]
_________________________________________________________________________________________________
up_sampling2d_12 (UpSampling2D) (None, 480, 320, 3)  0           up_sampling2d_11[0][0]
=================================================================================================
Total params: 156,355
Trainable params: 156,355
Non-trainable params: 0
_________________________________________________________________________________________________
```

Figure 22: ESRCNN Model summary

Average PSNR	23.86
Average MSE loss	272.0003
SSIM	0.5996

4.3 DSRCNN

4.3.1 2x

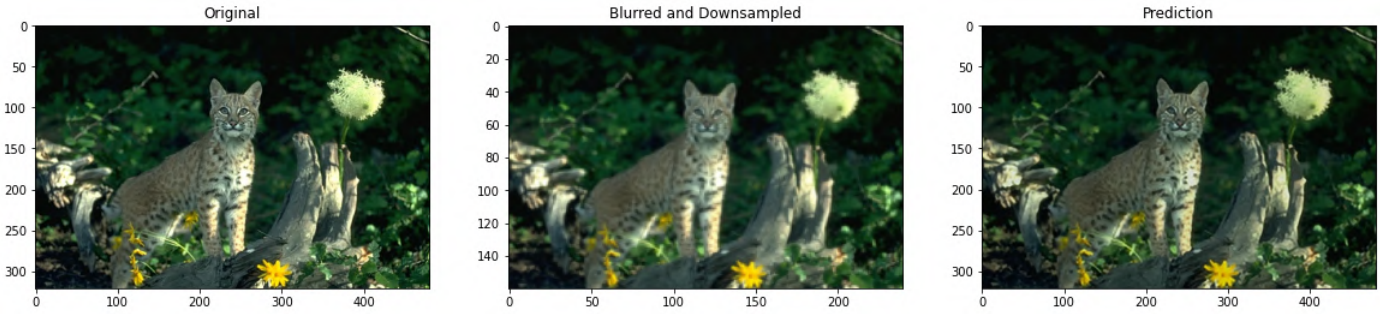

Figure 23: DSRCNN Image results

```
Model: "DSRCNN"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
Img (InputLayer)             [(None, 160, 240, 3)      0
_________________________________________________________________________________________________
resizing_2 (Resizing)        (None, 321, 481, 3)       0           Img[0][0]
_________________________________________________________________________________________________
conv2d_8 (Conv2D)            (None, 321, 481, 64)      1792        resizing_2[0][0]
_________________________________________________________________________________________________
conv2d_9 (Conv2D)            (None, 321, 481, 64)      36928       conv2d_8[0][0]
_________________________________________________________________________________________________
conv2d_transpose (Conv2DTranspo (None, 321, 481, 64)   36928       conv2d_9[0][0]
_________________________________________________________________________________________________
add_1 (Add)                  (None, 321, 481, 64)      0           conv2d_transpose[0][0]
                                                                   conv2d_9[0][0]
_________________________________________________________________________________________________
conv2d_transpose_1 (Conv2DTrans (None, 321, 481, 64)   36928       add_1[0][0]
_________________________________________________________________________________________________
add_2 (Add)                  (None, 321, 481, 64)      0           conv2d_transpose_1[0][0]
                                                                   conv2d_8[0][0]
_________________________________________________________________________________________________
conv2d_10 (Conv2D)           (None, 321, 481, 3)       1731        add_2[0][0]
=================================================================================================
Total params: 114,307
Trainable params: 114,307
Non-trainable params: 0
_________________________________________________________________________________________________
```

Figure 24: DSRCNN Model summary

Average PSNR	26.503
Average MSE loss	102.9427
Accuracy	0.9406

4.3.2 4x

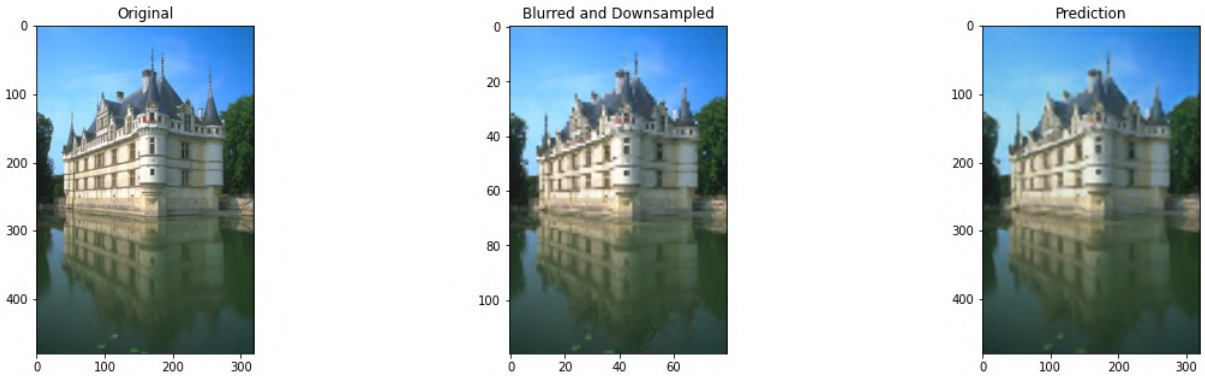

Figure 25: DSRCNN Image results

Model: "DSRCNN"

Layer (type)	Output Shape	Param #	Connected to
Img (InputLayer)	[(None, 120, 80, 3)]	0	
conv2d_54 (Conv2D)	(None, 120, 80, 64)	1792	Img[0][0]
conv2d_55 (Conv2D)	(None, 120, 80, 64)	36928	conv2d_54[0][0]
conv2d_transpose_4 (Conv2DTrans	(None, 120, 80, 64)	36928	conv2d_55[0][0]
add_8 (Add)	(None, 120, 80, 64)	0	conv2d_transpose_4[0][0] conv2d_55[0][0]
conv2d_transpose_5 (Conv2DTrans	(None, 120, 80, 64)	36928	add_8[0][0]
add_9 (Add)	(None, 120, 80, 64)	0	conv2d_transpose_5[0][0] conv2d_54[0][0]
conv2d_56 (Conv2D)	(None, 120, 80, 3)	1731	add_9[0][0]
up_sampling2d_14 (UpSampling2D)	(None, 240, 160, 3)	0	conv2d_56[0][0]
conv2d_57 (Conv2D)	(None, 240, 160, 64)	1792	up_sampling2d_14[0][0]
conv2d_58 (Conv2D)	(None, 240, 160, 32)	18464	conv2d_57[0][0]
conv2d_59 (Conv2D)	(None, 240, 160, 3)	867	conv2d_58[0][0]
up_sampling2d_15 (UpSampling2D)	(None, 480, 320, 3)	0	conv2d_59[0][0]

Total params: 135,430
Trainable params: 135,430
Non-trainable params: 0

Figure 26: DSRCNN Model summary

Average PSNR	24.23
Average MSE loss	249.0088
SSIM	0.6206

.

4.4 DDSRCNN

4.4.1 2x

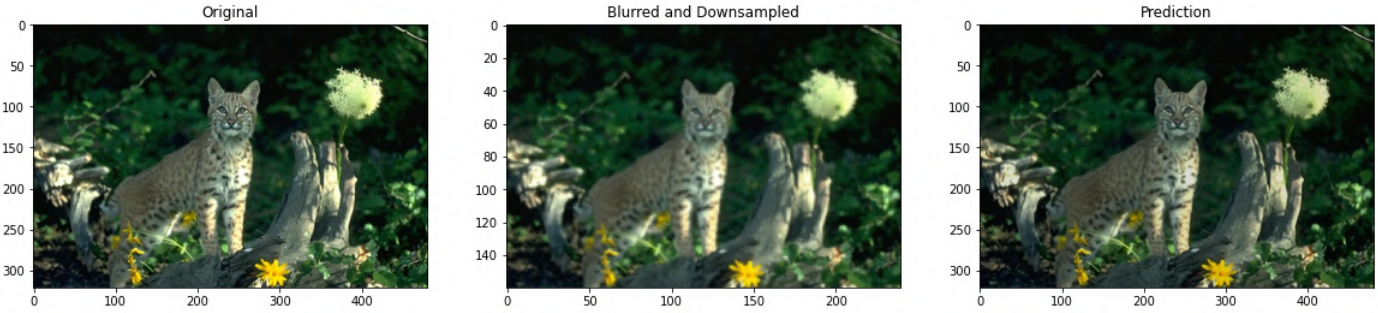

Figure 27: DDSRCNN Image results

Model: "DDSRCNN"

Layer (type)	Output Shape	Param #	Connected to
Img (InputLayer)	[(None, 160, 240, 3)	0	
resizing_3 (Resizing)	(None, 320, 480, 3)	0	Img[0][0]
conv2d_11 (Conv2D)	(None, 320, 480, 64)	1792	resizing_3[0][0]
conv2d_12 (Conv2D)	(None, 320, 480, 64)	36928	conv2d_11[0][0]
max_pooling2d (MaxPooling2D)	(None, 160, 240, 64)	0	conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, 160, 240, 128	73856	max_pooling2d[0][0]
conv2d_14 (Conv2D)	(None, 160, 240, 128	147584	conv2d_13[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 80, 120, 128)	0	conv2d_14[0][0]
conv2d_15 (Conv2D)	(None, 80, 120, 256)	295168	max_pooling2d_1[0][0]
up_sampling2d (UpSampling2D)	(None, 160, 240, 256	0	conv2d_15[0][0]
conv2d_16 (Conv2D)	(None, 160, 240, 128	295040	up_sampling2d[0][0]
conv2d_17 (Conv2D)	(None, 160, 240, 128	147584	conv2d_16[0][0]
add_3 (Add)	(None, 160, 240, 128	0	conv2d_17[0][0] conv2d_14[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 320, 480, 128	0	add_3[0][0]
conv2d_18 (Conv2D)	(None, 320, 480, 64)	73792	up_sampling2d_1[0][0]
conv2d_19 (Conv2D)	(None, 320, 480, 64)	36928	conv2d_18[0][0]
add_4 (Add)	(None, 320, 480, 64)	0	conv2d_19[0][0] conv2d_12[0][0]
resizing_4 (Resizing)	(None, 321, 481, 64)	0	add_4[0][0]
conv2d_20 (Conv2D)	(None, 321, 481, 3)	1731	resizing_4[0][0]

Total params: 1,110,403
Trainable params: 1,110,403
Non-trainable params: 0

Figure 28: DDSRCNN Model summary

Average PSNR	27.29
Average MSE loss	104.1706
Accuracy	0.9216

4.4.2 4x

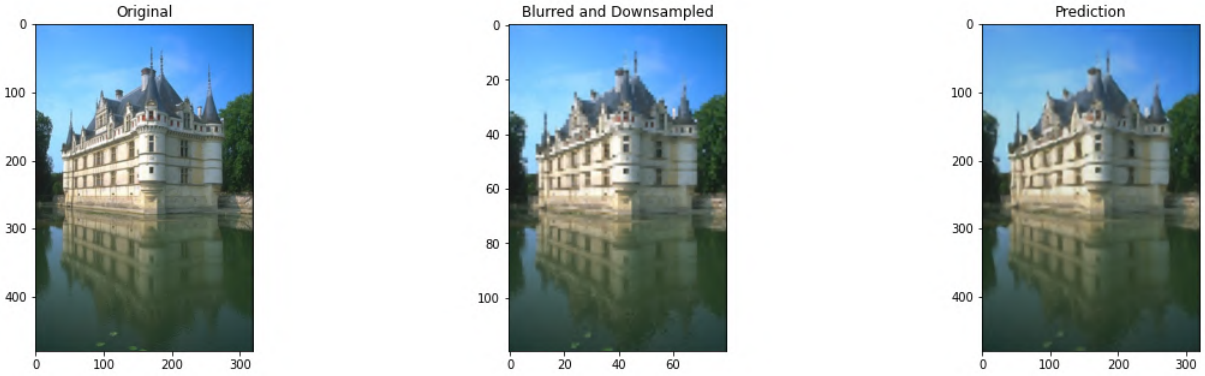

Figure 29: DDSRCNN Image results

```
Model: "DDSRCNN"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Img (InputLayer)                [(None, 120, 80, 3)] 0
__________________________________________________________________________________________________
conv2d_149 (Conv2D)             (None, 120, 80, 64)  1792        Img[0][0]
__________________________________________________________________________________________________
conv2d_150 (Conv2D)             (None, 120, 80, 64)  36928       conv2d_149[0][0]
__________________________________________________________________________________________________
max_pooling2d_16 (MaxPooling2D) (None, 60, 40, 64)   0           conv2d_150[0][0]
__________________________________________________________________________________________________
conv2d_151 (Conv2D)             (None, 60, 40, 128)  73856       max_pooling2d_16[0][0]
__________________________________________________________________________________________________
conv2d_152 (Conv2D)             (None, 60, 40, 128)  147584      conv2d_151[0][0]
__________________________________________________________________________________________________
max_pooling2d_17 (MaxPooling2D) (None, 30, 20, 128)  0           conv2d_152[0][0]
__________________________________________________________________________________________________
conv2d_153 (Conv2D)             (None, 30, 20, 256)  295168      max_pooling2d_17[0][0]
__________________________________________________________________________________________________
up_sampling2d_36 (UpSampling2D) (None, 60, 40, 256)  0           conv2d_153[0][0]
__________________________________________________________________________________________________
conv2d_154 (Conv2D)             (None, 60, 40, 128)  295040      up_sampling2d_36[0][0]
__________________________________________________________________________________________________
conv2d_155 (Conv2D)             (None, 60, 40, 128)  147584      conv2d_154[0][0]
__________________________________________________________________________________________________
add_26 (Add)                    (None, 60, 40, 128)  0           conv2d_155[0][0]
                                                                 conv2d_152[0][0]
__________________________________________________________________________________________________
up_sampling2d_37 (UpSampling2D) (None, 120, 80, 128) 0           add_26[0][0]
__________________________________________________________________________________________________
conv2d_156 (Conv2D)             (None, 120, 80, 64)  73792       up_sampling2d_37[0][0]
__________________________________________________________________________________________________
conv2d_157 (Conv2D)             (None, 120, 80, 64)  36928       conv2d_156[0][0]
__________________________________________________________________________________________________
add_27 (Add)                    (None, 120, 80, 64)  0           conv2d_157[0][0]
                                                                 conv2d_150[0][0]
__________________________________________________________________________________________________
up_sampling2d_38 (UpSampling2D) (None, 240, 160, 64) 0           add_27[0][0]
__________________________________________________________________________________________________
conv2d_158 (Conv2D)             (None, 240, 160, 128 73856       up_sampling2d_38[0][0]
__________________________________________________________________________________________________
up_sampling2d_39 (UpSampling2D) (None, 480, 320, 128 0           conv2d_158[0][0]
__________________________________________________________________________________________________
conv2d_159 (Conv2D)             (None, 480, 320, 64) 73792       up_sampling2d_39[0][0]
__________________________________________________________________________________________________
conv2d_160 (Conv2D)             (None, 480, 320, 32) 18464       conv2d_159[0][0]
__________________________________________________________________________________________________
conv2d_161 (Conv2D)             (None, 480, 320, 3)  867         conv2d_160[0][0]
==================================================================================================
Total params: 1,275,651
Trainable params: 1,275,651
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 30: DDSRCNN Model summary

Average PSNR	24.18
Average MSE loss	198.1032
SSIM	0.6315

4.5 SRResNet

4.5.1 2x

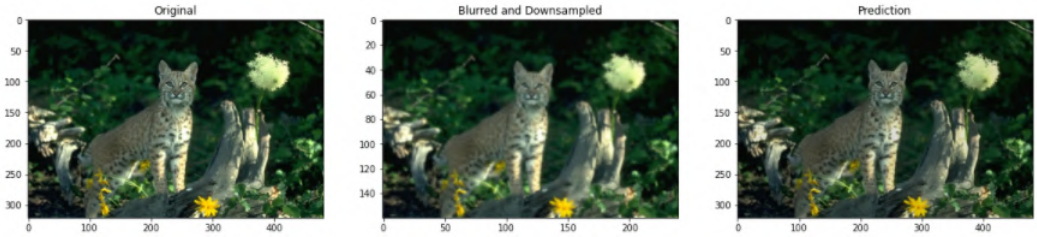

Figure 31: SRResNet Image results

```
Model: "functional_13"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_7 (InputLayer)            [(None, 160, 240, 3) 0
__________________________________________________________________________________________________
sr_res_conv1 (Conv2D)           (None, 160, 240, 64) 1792        input_7[0][0]
__________________________________________________________________________________________________
sr_res_conv_1_1 (Conv2D)        (None, 160, 240, 64) 36928       sr_res_conv1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_1_1 (BatchNorm (None, 160, 240, 64) 256         sr_res_conv_1_1[0][0]
__________________________________________________________________________________________________
sr_res_activation_1_1 (Activati (None, 160, 240, 64) 0           sr_res_batchnorm_1_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_1_2 (Conv2D)        (None, 160, 240, 64) 36928       sr_res_activation_1_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_1_2 (BatchNorm (None, 160, 240, 64) 256         sr_res_conv_1_2[0][0]
__________________________________________________________________________________________________
sr_res_merge_1 (Add)            (None, 160, 240, 64) 0           sr_res_batchnorm_1_2[0][0]
                                                                 sr_res_conv1[0][0]
__________________________________________________________________________________________________
sr_res_conv_2_1 (Conv2D)        (None, 160, 240, 64) 36928       sr_res_merge_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_2_1 (BatchNorm (None, 160, 240, 64) 256         sr_res_conv_2_1[0][0]
__________________________________________________________________________________________________
sr_res_activation_2_1 (Activati (None, 160, 240, 64) 0           sr_res_batchnorm_2_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_2_2 (Conv2D)        (None, 160, 240, 64) 36928       sr_res_activation_2_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_2_2 (BatchNorm (None, 160, 240, 64) 256         sr_res_conv_2_2[0][0]
__________________________________________________________________________________________________
sr_res_merge_2 (Add)            (None, 160, 240, 64) 0           sr_res_batchnorm_2_2[0][0]
                                                                 sr_res_merge_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_3_1 (Conv2D)        (None, 160, 240, 64) 36928       sr_res_merge_2[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_3_1 (BatchNorm (None, 160, 240, 64) 256         sr_res_conv_3_1[0][0]
__________________________________________________________________________________________________
sr_res_activation_3_1 (Activati (None, 160, 240, 64) 0           sr_res_batchnorm_3_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_3_2 (Conv2D)        (None, 160, 240, 64) 36928       sr_res_activation_3_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_3_2 (BatchNorm (None, 160, 240, 64) 256         sr_res_conv_3_2[0][0]
__________________________________________________________________________________________________
sr_res_merge_3 (Add)            (None, 160, 240, 64) 0           sr_res_batchnorm_3_2[0][0]
                                                                 sr_res_merge_2[0][0]
__________________________________________________________________________________________________
sr_res_conv_4_1 (Conv2D)        (None, 160, 240, 64) 36928       sr_res_merge_3[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_4_1 (BatchNorm (None, 160, 240, 64) 256         sr_res_conv_4_1[0][0]
__________________________________________________________________________________________________
sr_res_activation_4_1 (Activati (None, 160, 240, 64) 0           sr_res_batchnorm_4_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_4_2 (Conv2D)        (None, 160, 240, 64) 36928       sr_res_activation_4_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_4_2 (BatchNorm (None, 160, 240, 64) 256         sr_res_conv_4_2[0][0]
__________________________________________________________________________________________________
sr_res_merge_4 (Add)            (None, 160, 240, 64) 0           sr_res_batchnorm_4_2[0][0]
```

Figure 32: SRResNet Model summary

Average PSNR	27.503
Average MSE loss	105.9868
Accuracy	0.9406

4.5.2 4x

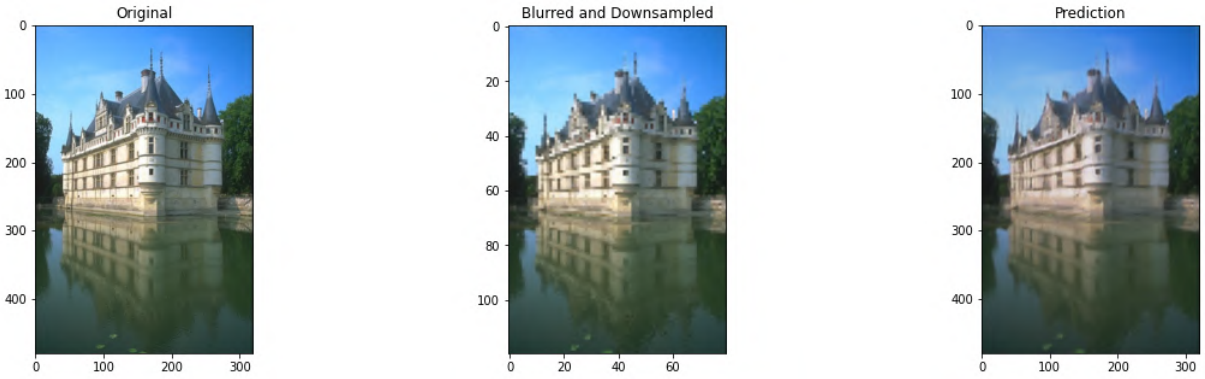

Figure 33: SRResNet Image results

```
Model: "functional_7"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_8 (InputLayer)            [(None, 120, 80, 3)] 0
__________________________________________________________________________________________________
sr_res_conv1 (Conv2D)           (None, 120, 80, 64)  1792        input_8[0][0]
__________________________________________________________________________________________________
sr_res_conv_1_1 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_conv1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_1_1 (BatchNorm (None, 120, 80, 64)  256         sr_res_conv_1_1[0][0]
__________________________________________________________________________________________________
sr_res_activation_1_1 (Activati (None, 120, 80, 64)  0           sr_res_batchnorm_1_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_1_2 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_activation_1_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_1_2 (BatchNorm (None, 120, 80, 64)  256         sr_res_conv_1_2[0][0]
__________________________________________________________________________________________________
sr_res_merge_1 (Add)            (None, 120, 80, 64)  0           sr_res_batchnorm_1_2[0][0]
                                                                 sr_res_conv1[0][0]
__________________________________________________________________________________________________
sr_res_conv_2_1 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_merge_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_2_1 (BatchNorm (None, 120, 80, 64)  256         sr_res_conv_2_1[0][0]
__________________________________________________________________________________________________
sr_res_activation_2_1 (Activati (None, 120, 80, 64)  0           sr_res_batchnorm_2_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_2_2 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_activation_2_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_2_2 (BatchNorm (None, 120, 80, 64)  256         sr_res_conv_2_2[0][0]
__________________________________________________________________________________________________
sr_res_merge_2 (Add)            (None, 120, 80, 64)  0           sr_res_batchnorm_2_2[0][0]
                                                                 sr_res_merge_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_3_1 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_merge_2[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_3_1 (BatchNorm (None, 120, 80, 64)  256         sr_res_conv_3_1[0][0]
__________________________________________________________________________________________________
sr_res_activation_3_1 (Activati (None, 120, 80, 64)  0           sr_res_batchnorm_3_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_3_2 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_activation_3_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_3_2 (BatchNorm (None, 120, 80, 64)  256         sr_res_conv_3_2[0][0]
__________________________________________________________________________________________________
sr_res_merge_3 (Add)            (None, 120, 80, 64)  0           sr_res_batchnorm_3_2[0][0]
                                                                 sr_res_merge_2[0][0]
__________________________________________________________________________________________________
sr_res_conv_4_1 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_merge_3[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_4_1 (BatchNorm (None, 120, 80, 64)  256         sr_res_conv_4_1[0][0]
__________________________________________________________________________________________________
sr_res_activation_4_1 (Activati (None, 120, 80, 64)  0           sr_res_batchnorm_4_1[0][0]
__________________________________________________________________________________________________
sr_res_conv_4_2 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_activation_4_1[0][0]
__________________________________________________________________________________________________
sr_res_batchnorm_4_2 (BatchNorm (None, 120, 80, 64)  256         sr_res_conv_4_2[0][0]
__________________________________________________________________________________________________
sr_res_merge_4 (Add)            (None, 120, 80, 64)  0           sr_res_batchnorm_4_2[0][0]
                                                                 sr_res_merge_3[0][0]
__________________________________________________________________________________________________
sr_res_conv_5_1 (Conv2D)        (None, 120, 80, 64)  36928       sr_res_merge_4[0][0]
__________________________________________________________________________________________________
```

Figure 34: SRResNet Model summary

Average PSNR	25.21
Average MSE loss	211.5460
SSIM	0.6294

4.6 SRGAN

For the SRGAN model we input a low resolution image of dimensions 120x80 and after training the Generator and Discriminator models through 38000 iterations, the output can be visualized as a step by step reconstruction of the image enhancing its quality.
The motive of this architecture is to recover finer textures from the image when we upscale it so that it's quality cannot be compromised. The other methods proposed suffer from image information loss and smoothing when moving from 2x-Upscaling to 4x-Upscaling. In our results we confirm that SRGAN,although having comparable PSNR values, generates image more pleasing to eyes as compared to previous proposed models.
below is a visualization of our GAN learning to reconstruct a high resolution image of an input image from the BSD100 dataset:

Figure 35: Original image

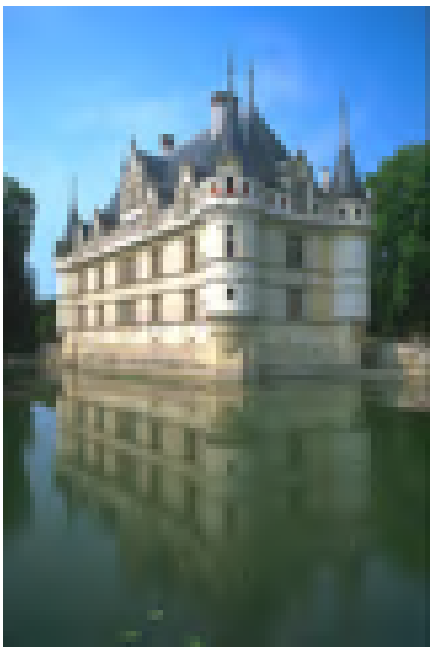

Figure 36: LowResolution Input

Figure 37: Result

.

Figure 38: Output

Average PSNR	25.36
Average MSE loss	102.9427
SSIM	0.6021

.

4.7 ESRGAN

Even though SRGAN was able to achieve in accuracy and speed of single image super-resolution using faster and deeper convolutional neural networks, one central problem remains how do we recover the finer texture details How ESRGAN actually functions is by looking for colour variation between an image's pixels, and then inserting the detail that it thinks should exist based on the images it's seen before. When trained on photographs, ESRGAN inserts tons of details into the large pixels typically seen in older photos.

Figure 39: Original image

Figure 40: Low Resolution Input

Figure 41: Output

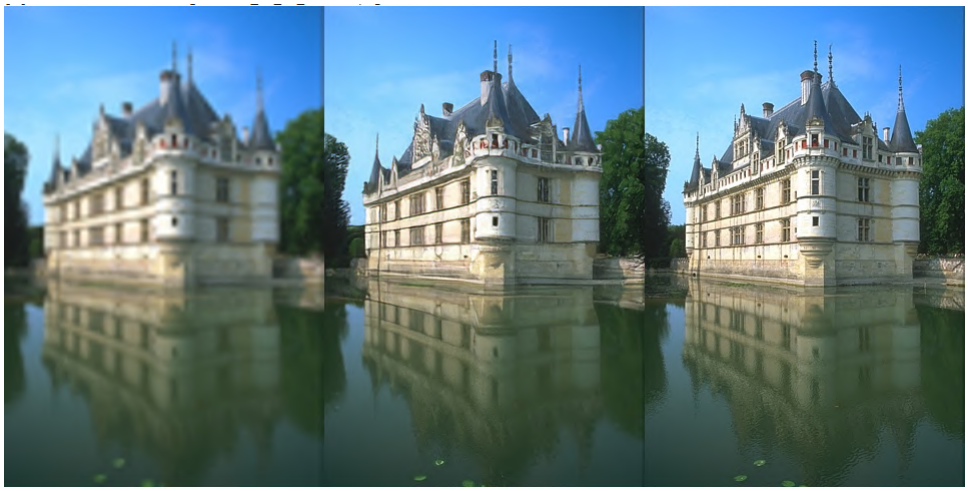

Figure 42: Comparison of the input, the original image and the output

Average PSNR	25.24
Average MSE loss	98.9538
SSIM	0.7801

5 Conclusion

From the following 7 Architecture we are able to obtain near to perfect outcomes for 2x Upscaling factors but at 4x upscaling, the first 5 proposed models fail to preserve texture details and suffer from detail loss and smoothening. In this scenario, ESRGAN is the most favourable as it regenerates fine grain textures for an overall more pleasing image.We can also conclude that all 7 Architecture's have similar PSNR and SSIM values based on the image upscaling factors. Our results are summarized for 4x upscaling on the BSD100 dataset:

Model	PSNR	SSIM
SRCNN	24.14	0.6255
ESRCNN	23.86	0.5996
DSRCNN	24.29	0.6206
DDSRCNN	24.17	0.6315
SRResNet	24.21	0.6294
SRGAN	24.64	0.6021
ESRGAN	25.24	0.7801

5.1 Future Work

For our future work we will be looking into further improving our proposed ESRGAN architecture so as to achieve a better SSIM value. We will be using the following paper for making further improvements 'ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks' (Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, Xiaoou Tang) [arXiv:1809.00219] .

References

[1] Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network - 25 May 2017
Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, ´ Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi Twitter

[2] Keras Conv2D and Convolutional Layers by Adrian Rosebrock on December 31, 2018
Source: https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

[3] Image Super-Resolution Using Deep Convolutional Networks Chao Dong, Chen Change Loy, Member, IEEE, Kaiming He, Member, IEEE, and Xiaoou Tang, Fellow, IEEE - 31 Jul 2015

[4] ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks - 17 Sep 2018 Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, Xiaoou Tang