IMAGE SUPER PIXELATION USING AI

A THESIS REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF TECHNOLOGY

IN

DIVISION OF COMPUTER ENGINEERING

Submitted by:

Vaibhav Sharan (2017UCO1677)
S.M. Aadithya (2017UIT2520)
Aman Hembram (2017UIT2507)

Under the Supervision of
Dr. Devender Kumar
Dr. Veenu

DIVISION OF COMPUTER ENGINEERING
NETAJI SUBHAS INSTITUTE OF TECHNOLOGY
UNIVERSITY OF DELHI
DECEMBER, 2020

ACKNOWLEDGEMENT

It is near impossible to work on a project without the encouragement & assistance of other people. Our project is no exception to this. We would like to express our sincere gratitude to our supervisors Prof. Devender Kumar and Prof. Veenu for providing their invaluable guidance, comments and suggestions throughout the course of the project. They were the ones who constantly motivated and encouraged us to work harder. We thank NSIT for providing us with such a good opportunity. We thank our parents, friends and family members who have always supported us. We would also give our thanks to our colleagues in directly or indirectly helping us develop the project and people who have willingly helped us out with their abilities. The project was an extremely productive & enriching experience, not only technically but also in the matter of providing some practical skills.

Department of Computer Engieering
University of Delhi
Delhi-110007, India

CERTIFICATE OF ORIGINALITY

We, Vaibhav Sharan (2017UCO1677) of B.Tech Department of Computer Enginerring, S.M. Aadithya (2017UIT2520), Aman Hembram (2017UIT2507) of B.Tech, Department of Information Technology. Hereby declare that the Project-Thesis titled "Image Super Pixelation using AI" which is submitted by us to the Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi (University of Delhi) in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology, is original and not copied from source without proper citation. The manuscript has been subjected to plagiarism check by Tutnitin software. This work has not previously formed the basis for the award of any Degree.

Place: Delhi

Date: 31/12/20

Vaibhav Sharan

S.M. Aadithya

Aman Hembram

Department of Computer Engineering

Netaji Subhas University of Technology

Delhi-11007, India

CERTIFICATE OF DECLARATION

This is to certify that the work embodied in the Project-thesis titled "Super Pixelation using AI" has been completed by S.M. Aadithya (2017UIT2520), Aman Hembram (2017UIT2507) of B.Tech, Department of Information Technology, and Vaibhav Sharan (2017UCO1677) of B.Tech, Department of Computer Sciences and Engineering, under the guidance of Dr. Devender Kumar and Dr. Veenu towards fulfillment of the requirements for the award of the degree of Bachelor of Technology. This work is based on original research and has not been submitted in full or in part of any other diploma or degree of any university.

Place: Delhi

Date: 31/12/20

Dr. Veenu

Dr. Devender Kumar

ABSTRACT

Recently, Image Processing has been a very interesting and rapidly evolving technology. Image processing is used to perform some operations on an image, in order to get an enhanced version of the image as well as to extract some useful information from it. It forms a core research area within engineering and computer science disciplines as well.

It involves tasks like importing the image through image acquisition tools, analysing and manipulating the image to get output in which result can be altered image or even a report that is based on image analysis.

Despite the breakthroughs in accuracy and speed of single image super-resolution using faster and deeper CNNs, 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.

A low-resolution image is processed and upscaled to a higher resolution output image using artificial intelligence models trained utilizing deep learning techniques like GANs (Generative Adversarial Networks), CNNs (Convolutional Neural Networks). Our aim is to create a model which, given an input image, will output a higher resolution image by filling in the unknown details by itself. We use both GAN components (generator and discriminator) which will compete and thus with training will return a model which will be capable enough where the generator can generate upsampled, good quality images from low quality images.

INDEX

LIST OF FIGURES:

CHAPTER 1

INTRODUCTION

1.1 Thesis Introduction

Super Pixelation is the process of upscaling and improving the details inside an image. The details in the output image are guessed/estimated using AI models trained using Generative Adversarial Networks (GAN), 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. Our deep residual network will be able to recover photo-realistic textures from heavily down sampled images.

The project consists of four main steps:

1. Review and study recent works on the topic.
2. Implement one or more of the solutions and possibly compare pre-trained versions.
3. Get some data, train and test the model.
4. Improve the model and the data to get better results.

1.2 Motivation

Prediction-based methods were among the first methods to tackle SISR (Single Image Super Resolution). While these filtering approaches, e.g. linear, bicubic filtering, can be very fast, they oversimplify the SISR problem and usually yield solutions with amped up anti-aliasing or overly smooth textures. Methods that put particularly focus on edge-preservation have been proposed. More powerful approaches aim to establish a complex mapping between low- and high-resolution image information and usually rely on training data. Many methods that are based on example-pairs rely on LR training patches for which the corresponding HR counterparts are known. Deeper network architectures can be difficult to train but have the potential to substantially increase the network's accuracy as they allow modelling mappings of very high complexity.

Now, in our daily lives whenever we come across an old image or a zoomed image that appears very low quality, we would want to enhance it to higher resolution. Also, we have a limited storage in our devices and online storage comes with a cost. So we need to minimize the amount of storage used. Although our image capturing devices have limited capabilities, by using SRGAN (Super Resolution Generative Adversarial Networks) we can retain every detail of our images.

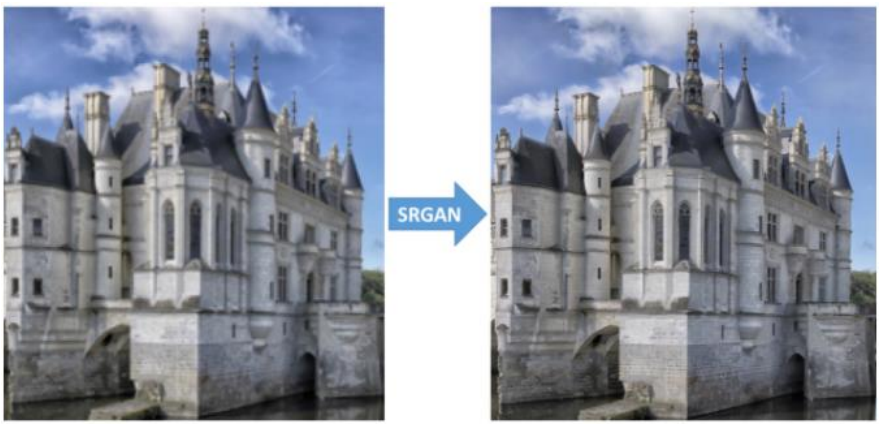

Fig 1. Image Super Resolution Example

The ultimate goal of our project is to train a generating function G that estimates for a given LR input image its corresponding HR counterpart. To achieve this, we train a generator network as a feed-forward CNN.

CHAPTER 2

RELATED WORK

2.1 Generative Adversarial Network

GANs are a class of AI algorithms based on Unsupervised Machine Learning. GANs are deep neural network architectures comprised of two networks (Generator and Discriminator) competing with each other (thus the “adversarial”). GANs is about creating, like drawing a portrait or composing a symphony. The main focus for GANs is to generate data from scratch.

To understand GANs, first we need to understand what a generative model is. In machine learning, the two main classes of models are generative and discriminative. A discriminative model is one that discriminates between two (or more) different classes of data, for example a convolutional neural network that is trained to output 1 given an image of a car and otherwise 0. A generative model on the other hand doesn’t know anything about classes of data. Instead, its purpose is to generate new data which fits the distribution of the training data.

2.2 Basic Architecture

GANs consist of two main components which are the Generator and the Discriminator. Think it like a game where Generator tries to produce some data from probability distribution and Discriminator acts like a judge. Basically, Generator generates fake data which are really close to the realistic data in

order to fool the discriminator. While the discriminator tries to learn by distinguishing which data is fake and which data is real.

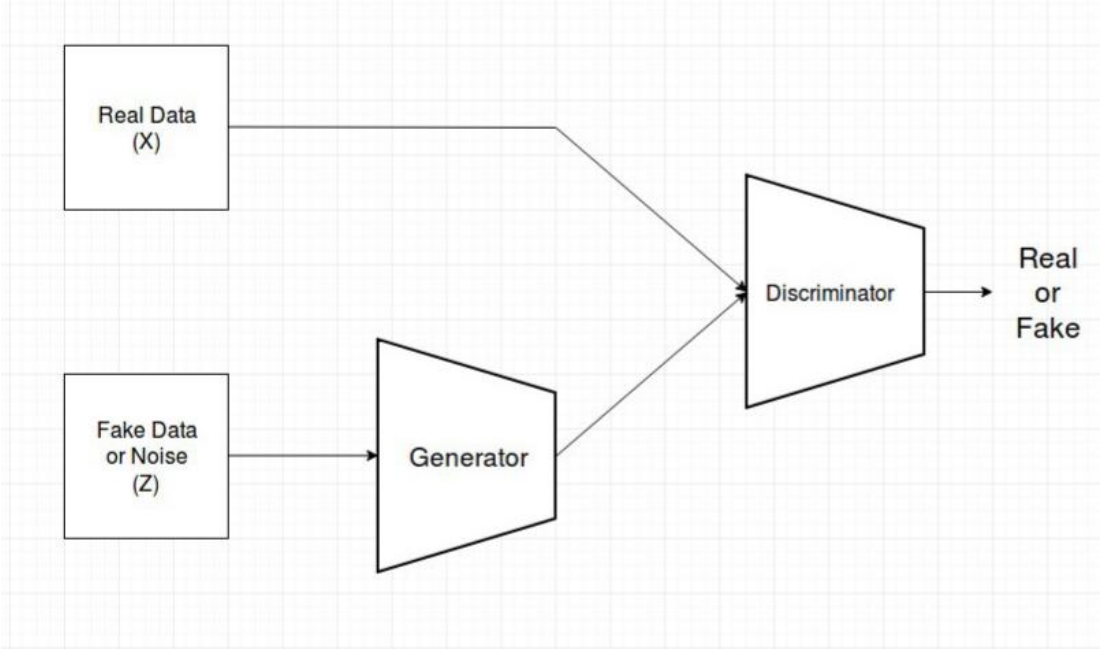

Fig 2. GAN basic principle

We pass low resolution (LR) images through the Generator by unsampling and giving Super Resolution Images.

Discriminator distinguishes the high resolution (HR) images and backpropagates loss (GAN loss) to train both the discriminator and the generator.

After that, the generator learns and again tries to produce more realistic images with High Resolution as during the training.

2.2.1 Adversarial Network Architecture

The general idea behind this formulation is that it allows one to train a generative model G with the goal of fooling a differentiable discriminator D that is trained to distinguish super-resolved images from real images. With this approach our generator can learn to create solutions that are highly similar to real images and thus difficult to classify by D. At the core of our very deep generator network G are B residual blocks with identical layout. We use two convolutional layers with small 3x3 kernels and 64 feature maps followed by batch-normalization layers and ParametricReLU as the activation function. We increase the resolution of the input image with two trained sub-pixel convolution layers.

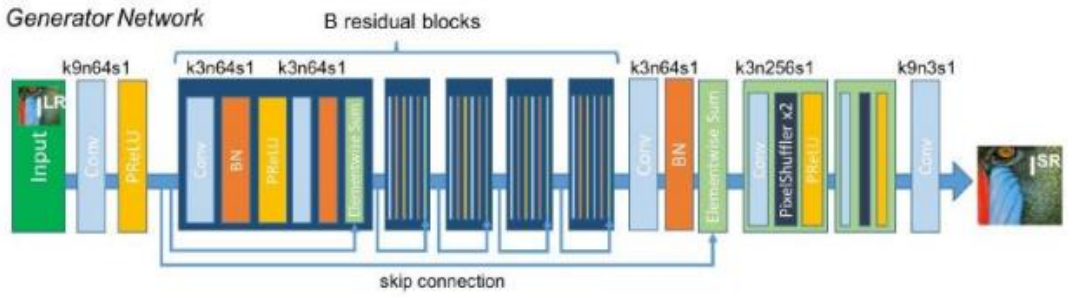

Fig 3. Generator Model Architecture

To discriminate real HR images from generated SR samples we train a discriminator network. The architecture. We follow the architectural guidelines summarized by Radford et al. and use LeakyReLUactivation (= 0:2) and avoid max-pooling throughout the network. The discriminator network is trained to solve the maximization problem. It contains eight convolutional layers with an increasing number of 3 3 filter kernels, increasing by a factor of 2 from 64 to 512 kernels as in the VGG network. Strided convolutions are used to reduce

the image resolution each time the number of features is doubled. The resulting 512 feature maps are followed by two dense layers and a final sigmoid activation function to obtain a probability for sample classification.

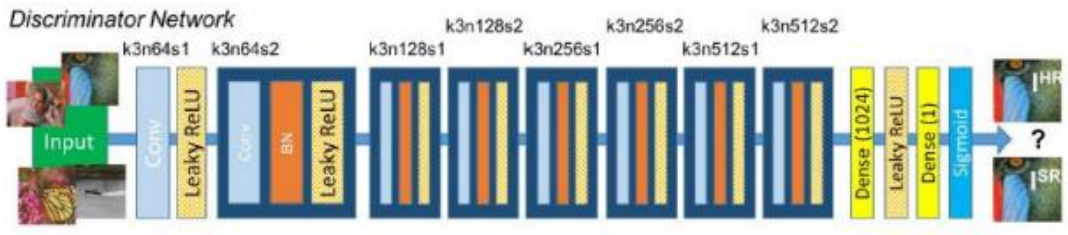

Fig 4. Discriminator Model Architecture

2.3 Components of GAN

The components of GAN, Generator and Discriminator, are CNNs (Convolutional Neural Networks) as they process the image.

2.3.1 Convolutional Neural Networks

A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data. A convolution is essentially sliding a filter over the input.

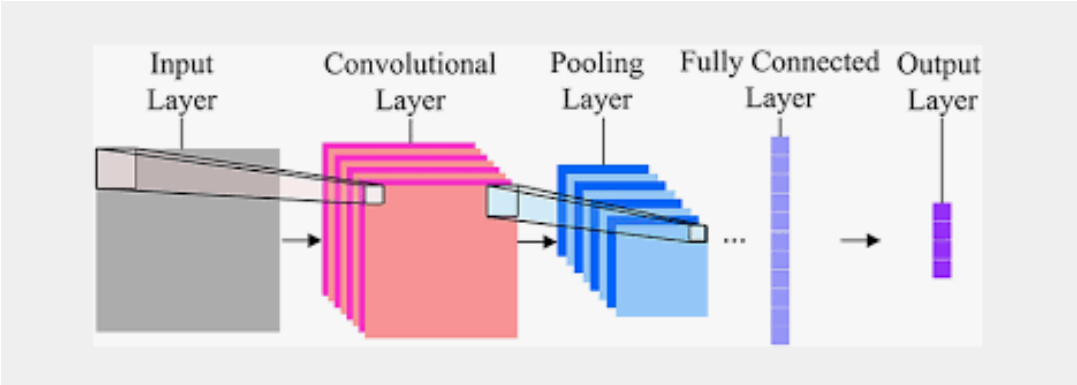

Fig 5. CNN Components

The state of the art for many computer vision problems is meanwhile set by specifically designed CNN architectures. It was shown that deeper network architectures can be difficult to train but have the potential to substantially increase the network's accuracy as they allow modelling mappings of very high complexity.

To efficiently train these deeper network architectures, batch normalization is often used to counteract the internal co-variate shift. Deeper network architectures have also been shown to increase performance for SISR.

2.3.2 Loss Functions

This is the most important part. As discussed we will be using Perceptual loss. It comprises of Content (Reconstruction) loss and Adversarial loss.

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

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.

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

Content Loss: Content loss we are using so that we can keep perceptual similarity instead of pixel wise similarity. This will allow us to recover photo-realistic textures from heavily down sampled images. Instead of relying on pixel-wise losses we will and use a loss function that is closer to perceptual similarity. We define the VGG loss based on the ReLU activation layers of the per-trained 19 layer VGG network. VGG loss is defined as the euclidean distance between the feature representations of a reconstructed image and the reference image.

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

Pixel-wise loss functions such as MSE (Mean Square Error) struggle to handle the uncertainty inherent in recovering lost high-frequency details such as texture: minimizing MSE encourages finding pixel-wise averages of plausible solutions which are typically overly-smooth and thus have poor perceptual quality. Reconstructions of varying perceptual quality the problem of minimizing MSE where multiple potential solutions with high texture details are averaged to create a smooth reconstruction.

MSE (Mean Square Error) is the loss/cost function most ordinarily used regression loss function. MSE is that the sum of squared distances between our target variable and predicted values Mean squared error may be a loss function used for regression once you don't need outliers to play an enormous role. Used as a performance metric, it is easy to interpret.

2.3.3 Activation Function

Activation functions take any real number as input, also known as its domain, and outputs a number in a certain range using a non-linear differentiable function. We typically use them for classification between certain layers in deep neural networks and more specifically in games.

Activation functions should be differentiable and non-linear because it is used for backpropagation for training neural network and updating its parameters. It needs to be able to differentiate and provide gradients to the previous layer. It also needs to be non-linear. The features that are computed within the neutral networks can be complex. If we didn't use non-linear activations, a neural network like this one with multiple hidden layers and neurons could actually be collapsed into a simple linear regression.

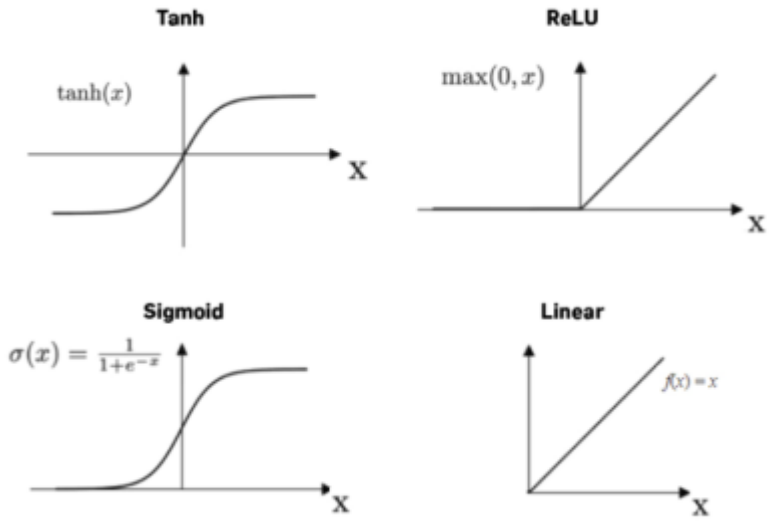

Fig 6. Activation Functions

2.3.4 Backpropagation

Backpropagation is a very common technique for preparing artificial neural organizations and it is based on an inclination based improvement strategy, for example, stochastic angle plunge. In backpropagation, we initially figure the forecast and its related loss for each preparation model. At that point we proliferate the mistake to register the fractional subsidiaries of the cost work C for all loads w and predispositions b in the fake neural organizations.

2.4 Challenges of a CNN

The main challenges for a neural network are learning speed and training set size. In order to generalize well, a neural network must have enough training data to learn from. It also needs to be able to learn quickly enough so that it can adapt to changing conditions as it is deployed.

The other main challenge is being able to provide the network with sufficient computing power to train itself. A modern deep neural network is a massive parallel computation, and as such requires a large amount of processing power to train.

The major challenges of a neural network and deep neural network are:

1. Data hungry: the models tend to overfit if you don't have a lot of data to train them on.
2. Compute hungry: they tend to be very computationally intensive and as such mere mortals like me with just a laptop cannot do

much, or at least we cannot try out complex state of the art models like the researchers and more wealthy can. Runtime has the same problem though not as pronounced.

3. Training time: even on multiple GPUs some models take a long time to train.
4. Interpretability: they have millions, or even billions of parameters with orders of magnitude more relationships between said parameters. We therefore cannot know what relationships or parameters led to a conclusion, or why exactly it came to such a conclusion.

CHAPTER 3

APPROACH

3.1 Our Approach

In SISR, the aim is to estimate a high-resolution, super resolved image ISR from a low-resolution input image ILR. Here ILR is the low-resolution version of its high-resolution counterpart IHR. The high-resolution images are only available during training. In training, ILR is obtained by applying a Gaussian filter to HR followed by a down sampling operation with down sampling factor 4.

Our ultimate goal is to train a generating function G that estimates for a given LR input image its corresponding HR counterpart. To achieve this, we train a generator network as a feed-forward CNN

In this work we will specifically design a perceptual loss lSR as a weighted combination of several loss components that model distinct desirable characteristics of the recovered SR image.

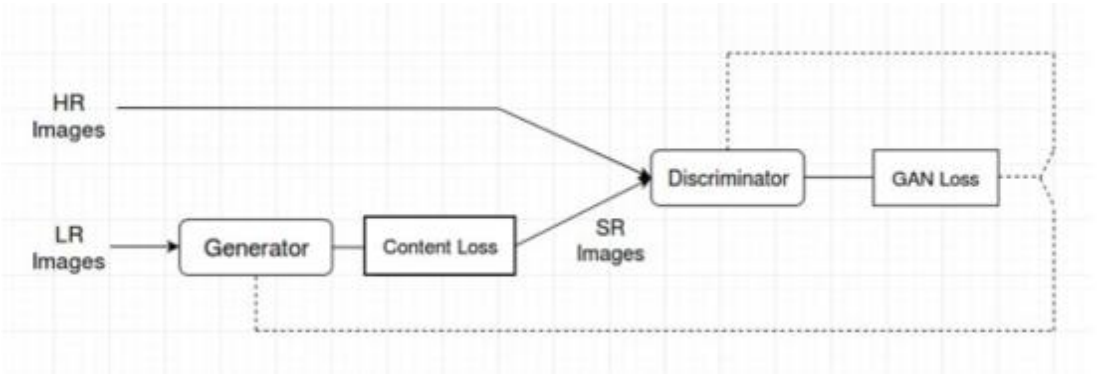

Fig 7. Our GAN's approach

Since GAN is basically ConvNet, so it has fixed input and output dimensions. But we may have varying (width X height) images. For training Generator, we resize every image into (96 x 96) size and this acts as Low resolution image and the output of Generator is (384 x 384) dimension. But say after training, if we have an image of size 1024 x 720, then The output (384 x 384) will be of lower resolution than our input image, which defeats our purpose. For this we will divide the images into smaller cells of size 96x96 and get results of 384x384 from our generator, we will then recombine these and get out whole upscaled image.

3.2 Design of The Model

The general idea behind this formulation is that it allows one to train a generative model G with the goal of fooling a differentiable discriminator D that is trained to distinguish super-resolved images from real images. With this approach our generator can learn to create solutions that are highly similar to real images and thus difficult to classify by D. At the core of our very deep generator network G are B residual blocks with identical layout. We use two convolutional layers with small 3x3 kernels and 64 feature maps followed by batch-normalization layers and ParametricReLU as the activation function. We increase the resolution of the input image with two trained sub-pixel convolution layers.

To discriminate real HR images from generated SR samples we train a discriminator network. The architecture. We follow the architectural guidelines summarized by Radford et al. and use LeakyReLUactivation (= 0:2) and avoid

max-pooling throughout the network. The discriminator network is trained to solve the maximization problem. It contains eight convolutional layers with an increasing number of 3 3 filter kernels, increasing by a factor of 2 from 64 to 512 kernels as in the VGG network. Strided convolutions are used to reduce the image resolution each time the number of features is doubled. The resulting 512 feature maps are followed by two dense layers and a final sigmoid activation function to obtain a probability for sample classification.

Generator Model:

Model: "model_3"

Layer (type)	Output Shape	Param #	Connected to
input_6 (InputLayer)	[(None, 384, 384, 3)	0	
conv2d_40 (Conv2D)	(None, 384, 384, 64)	15616	input_6[0][0]
p_re_lu (PReLU)	(None, 384, 384, 64)	64	conv2d_40[0][0]
conv2d_41 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu[0][0]
batch_normalization_35 (BatchNo	(None, 384, 384, 64)	256	conv2d_41[0][0]
p_re_lu_1 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_35[0][0]
conv2d_42 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_1[0][0]
batch_normalization_36 (BatchNo	(None, 384, 384, 64)	256	conv2d_42[0][0]
add (Add)	(None, 384, 384, 64)	0	p_re_lu[0][0] batch_normalization_36[0][0]
conv2d_43 (Conv2D)	(None, 384, 384, 64)	36928	add[0][0]
batch_normalization_37 (BatchNo	(None, 384, 384, 64)	256	conv2d_43[0][0]
p_re_lu_2 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_37[0][0]
conv2d_44 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_2[0][0]
batch_normalization_38 (BatchNo	(None, 384, 384, 64)	256	conv2d_44[0][0]
add_1 (Add)	(None, 384, 384, 64)	0	add[0][0] batch_normalization_38[0][0]
conv2d_45 (Conv2D)	(None, 384, 384, 64)	36928	add_1[0][0]
batch_normalization_39 (BatchNo	(None, 384, 384, 64)	256	conv2d_45[0][0]
p_re_lu_3 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_39[0][0]
conv2d_46 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_3[0][0]
batch_normalization_40 (BatchNo	(None, 384, 384, 64)	256	conv2d_46[0][0]
add_2 (Add)	(None, 384, 384, 64)	0	add_1[0][0] batch_normalization_40[0][0]
conv2d_47 (Conv2D)	(None, 384, 384, 64)	36928	add_2[0][0]

batch_normalization_41 (BatchNo	(None, 384, 384, 64)	256	conv2d_47[0][0]
p_re_lu_4 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_41[0][0]
conv2d_48 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_4[0][0]
batch_normalization_42 (BatchNo	(None, 384, 384, 64)	256	conv2d_48[0][0]
add_3 (Add)	(None, 384, 384, 64)	0	add_2[0][0] batch_normalization_42[0][0]
conv2d_49 (Conv2D)	(None, 384, 384, 64)	36928	add_3[0][0]
batch_normalization_43 (BatchNo	(None, 384, 384, 64)	256	conv2d_49[0][0]
p_re_lu_5 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_43[0][0]
conv2d_50 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_5[0][0]
batch_normalization_44 (BatchNo	(None, 384, 384, 64)	256	conv2d_50[0][0]
add_4 (Add)	(None, 384, 384, 64)	0	add_3[0][0] batch_normalization_44[0][0]
conv2d_51 (Conv2D)	(None, 384, 384, 64)	36928	add_4[0][0]
batch_normalization_45 (BatchNo	(None, 384, 384, 64)	256	conv2d_51[0][0]
p_re_lu_6 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_45[0][0]
conv2d_52 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_6[0][0]
batch_normalization_46 (BatchNo	(None, 384, 384, 64)	256	conv2d_52[0][0]
add_5 (Add)	(None, 384, 384, 64)	0	add_4[0][0] batch_normalization_46[0][0]
conv2d_53 (Conv2D)	(None, 384, 384, 64)	36928	add_5[0][0]
batch_normalization_47 (BatchNo	(None, 384, 384, 64)	256	conv2d_53[0][0]
p_re_lu_7 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_47[0][0]
conv2d_54 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_7[0][0]
batch_normalization_48 (BatchNo	(None, 384, 384, 64)	256	conv2d_54[0][0]
add_6 (Add)	(None, 384, 384, 64)	0	add_5[0][0] batch_normalization_48[0][0]

batch_normalization_41 (BatchNo	(None, 384, 384, 64)	256	conv2d_47[0][0]
p_re_lu_4 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_41[0][0]
conv2d_48 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_4[0][0]
batch_normalization_42 (BatchNo	(None, 384, 384, 64)	256	conv2d_48[0][0]
add_3 (Add)	(None, 384, 384, 64)	0	add_2[0][0] batch_normalization_42[0][0]
conv2d_49 (Conv2D)	(None, 384, 384, 64)	36928	add_3[0][0]
batch_normalization_43 (BatchNo	(None, 384, 384, 64)	256	conv2d_49[0][0]
p_re_lu_5 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_43[0][0]
conv2d_50 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_5[0][0]
batch_normalization_44 (BatchNo	(None, 384, 384, 64)	256	conv2d_50[0][0]
add_4 (Add)	(None, 384, 384, 64)	0	add_3[0][0] batch_normalization_44[0][0]
conv2d_51 (Conv2D)	(None, 384, 384, 64)	36928	add_4[0][0]

batch_normalization_45 (BatchNo	(None, 384, 384, 64)	256	conv2d_51[0][0]
p_re_lu_6 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_45[0][0]
conv2d_52 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_6[0][0]
batch_normalization_46 (BatchNo	(None, 384, 384, 64)	256	conv2d_52[0][0]
add_5 (Add)	(None, 384, 384, 64)	0	add_4[0][0] batch_normalization_46[0][0]
conv2d_53 (Conv2D)	(None, 384, 384, 64)	36928	add_5[0][0]
batch_normalization_47 (BatchNo	(None, 384, 384, 64)	256	conv2d_53[0][0]
p_re_lu_7 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_47[0][0]
conv2d_54 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_7[0][0]
batch_normalization_48 (BatchNo	(None, 384, 384, 64)	256	conv2d_54[0][0]
add_6 (Add)	(None, 384, 384, 64)	0	add_5[0][0] batch_normalization_48[0][0]
conv2d_55 (Conv2D)	(None, 384, 384, 64)	36928	add_6[0][0]
batch_normalization_49 (BatchNo	(None, 384, 384, 64)	256	conv2d_55[0][0]
p_re_lu_8 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_49[0][0]
conv2d_56 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_8[0][0]
batch_normalization_50 (BatchNo	(None, 384, 384, 64)	256	conv2d_56[0][0]
add_7 (Add)	(None, 384, 384, 64)	0	add_6[0][0] batch_normalization_50[0][0]
conv2d_57 (Conv2D)	(None, 384, 384, 64)	36928	add_7[0][0]
batch_normalization_51 (BatchNo	(None, 384, 384, 64)	256	conv2d_57[0][0]
p_re_lu_9 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_51[0][0]
conv2d_58 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_9[0][0]
batch_normalization_52 (BatchNo	(None, 384, 384, 64)	256	conv2d_58[0][0]
add_8 (Add)	(None, 384, 384, 64)	0	add_7[0][0] batch_normalization_52[0][0]
conv2d_59 (Conv2D)	(None, 384, 384, 64)	36928	add_8[0][0]
batch_normalization_53 (BatchNo	(None, 384, 384, 64)	256	conv2d_59[0][0]
p_re_lu_10 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_53[0][0]
conv2d_60 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_10[0][0]
batch_normalization_54 (BatchNo	(None, 384, 384, 64)	256	conv2d_60[0][0]
add_9 (Add)	(None, 384, 384, 64)	0	add_8[0][0] batch_normalization_54[0][0]
conv2d_61 (Conv2D)	(None, 384, 384, 64)	36928	add_9[0][0]
batch_normalization_55 (BatchNo	(None, 384, 384, 64)	256	conv2d_61[0][0]
p_re_lu_11 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_55[0][0]
conv2d_62 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_11[0][0]
batch_normalization_56 (BatchNo	(None, 384, 384, 64)	256	conv2d_62[0][0]

add_10 (Add)	(None, 384, 384, 64)	0	add_9[0][0] batch_normalization_56[0][0]
conv2d_63 (Conv2D)	(None, 384, 384, 64)	36928	add_10[0][0]
batch_normalization_57 (BatchNo	(None, 384, 384, 64)	256	conv2d_63[0][0]
p_re_lu_12 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_57[0][0]
conv2d_64 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_12[0][0]
batch_normalization_58 (BatchNo	(None, 384, 384, 64)	256	conv2d_64[0][0]
add_11 (Add)	(None, 384, 384, 64)	0	add_10[0][0] batch_normalization_58[0][0]
conv2d_65 (Conv2D)	(None, 384, 384, 64)	36928	add_11[0][0]
batch_normalization_59 (BatchNo	(None, 384, 384, 64)	256	conv2d_65[0][0]
p_re_lu_13 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_59[0][0]
conv2d_66 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_13[0][0]
batch_normalization_60 (BatchNo	(None, 384, 384, 64)	256	conv2d_66[0][0]
add_12 (Add)	(None, 384, 384, 64)	0	add_11[0][0] batch_normalization_60[0][0]
conv2d_67 (Conv2D)	(None, 384, 384, 64)	36928	add_12[0][0]
batch_normalization_61 (BatchNo	(None, 384, 384, 64)	256	conv2d_67[0][0]
p_re_lu_14 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_61[0][0]
conv2d_68 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_14[0][0]
batch_normalization_62 (BatchNo	(None, 384, 384, 64)	256	conv2d_68[0][0]
add_13 (Add)	(None, 384, 384, 64)	0	add_12[0][0] batch_normalization_62[0][0]
conv2d_69 (Conv2D)	(None, 384, 384, 64)	36928	add_13[0][0]
batch_normalization_63 (BatchNo	(None, 384, 384, 64)	256	conv2d_69[0][0]
p_re_lu_15 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_63[0][0]

conv2d_70 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_15[0][0]
batch_normalization_64 (BatchNo	(None, 384, 384, 64)	256	conv2d_70[0][0]
add_14 (Add)	(None, 384, 384, 64)	0	add_13[0][0] batch_normalization_64[0][0]
conv2d_71 (Conv2D)	(None, 384, 384, 64)	36928	add_14[0][0]
batch_normalization_65 (BatchNo	(None, 384, 384, 64)	256	conv2d_71[0][0]
p_re_lu_16 (PReLU)	(None, 384, 384, 64)	64	batch_normalization_65[0][0]
conv2d_72 (Conv2D)	(None, 384, 384, 64)	36928	p_re_lu_16[0][0]
batch_normalization_66 (BatchNo	(None, 384, 384, 64)	256	conv2d_72[0][0]
add_15 (Add)	(None, 384, 384, 64)	0	add_14[0][0] batch_normalization_66[0][0]
conv2d_73 (Conv2D)	(None, 384, 384, 64)	36928	add_15[0][0]
batch_normalization_67 (BatchNo	(None, 384, 384, 64)	256	conv2d_73[0][0]
add_16 (Add)	(None, 384, 384, 64)	0	p_re_lu[0][0] batch_normalization_67[0][0]
conv2d_74 (Conv2D)	(None, 384, 384, 256	147712	add_16[0][0]
up_sampling2d (UpSampling2D)	(None, 768, 768, 256	0	conv2d_74[0][0]
leaky_re_lu_45 (LeakyReLU)	(None, 768, 768, 256	0	up_sampling2d[0][0]
conv2d_75 (Conv2D)	(None, 768, 768, 256	590080	leaky_re_lu_45[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 1536, 1536, 2	0	conv2d_75[0][0]
leaky_re_lu_46 (LeakyReLU)	(None, 1536, 1536, 2	0	up_sampling2d_1[0][0]
conv2d_76 (Conv2D)	(None, 1536, 1536, 3	62211	leaky_re_lu_46[0][0]
activation_5 (Activation)	(None, 1536, 1536, 3	0	conv2d_76[0][0]

Total params: 2,043,779
Trainable params: 2,039,555
Non-trainable params: 4,224

Discriminator model:

Model: "model_2"

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 384, 384, 3)]	0
conv2d_32 (Conv2D)	(None, 384, 384, 64)	1792
leaky_re_lu_36 (LeakyReLU)	(None, 384, 384, 64)	0
conv2d_33 (Conv2D)	(None, 192, 192, 64)	36928
batch_normalization_28 (Batc	(None, 192, 192, 64)	256
leaky_re_lu_37 (LeakyReLU)	(None, 192, 192, 64)	0
conv2d_34 (Conv2D)	(None, 192, 192, 128)	73856
batch_normalization_29 (Batc	(None, 192, 192, 128)	512

leaky_re_lu_38 (LeakyReLU)	(None, 192, 192, 128)	0
conv2d_35 (Conv2D)	(None, 96, 96, 128)	147584
batch_normalization_30 (Batc	(None, 96, 96, 128)	512
leaky_re_lu_39 (LeakyReLU)	(None, 96, 96, 128)	0
conv2d_36 (Conv2D)	(None, 96, 96, 256)	295168
batch_normalization_31 (Batc	(None, 96, 96, 256)	1024
leaky_re_lu_40 (LeakyReLU)	(None, 96, 96, 256)	0
conv2d_37 (Conv2D)	(None, 48, 48, 256)	590080
batch_normalization_32 (Batc	(None, 48, 48, 256)	1024
leaky_re_lu_41 (LeakyReLU)	(None, 48, 48, 256)	0
conv2d_38 (Conv2D)	(None, 48, 48, 512)	1180160
batch_normalization_33 (Batc	(None, 48, 48, 512)	2048
leaky_re_lu_42 (LeakyReLU)	(None, 48, 48, 512)	0
conv2d_39 (Conv2D)	(None, 24, 24, 512)	2359808
batch_normalization_34 (Batc	(None, 24, 24, 512)	2048
leaky_re_lu_43 (LeakyReLU)	(None, 24, 24, 512)	0
flatten_4 (Flatten)	(None, 294912)	0
dense_10 (Dense)	(None, 1024)	301990912
leaky_re_lu_44 (LeakyReLU)	(None, 1024)	0
dense_11 (Dense)	(None, 1)	1025
activation_4 (Activation)	(None, 1)	0

Total params: 306,684,737
Trainable params: 306,681,025
Non-trainable params: 3,712

3.3 Dats Set and Experimental Tools Setup

1. Pretrained VGG19 model

2. DIV2K dataset: DIVerse 2K resolution high quality images as used for the challenges @ NTIRE (CVPR 2017 and CVPR 2018) and @ PIRM (ECCV 2018).

Experimental Tools used

1. TensorFlow
2. Keras
3. Jupyter Notebook
4. Spyder

Data Overview

A large newly collected dataset -DIV2K- of RGB images with a large diversity of contents.

The DIV2K dataset is divided into:

Train data: starting from 800 high definition high resolution images we obtain corresponding low resolution images and provide both high and low resolution images for 2, 3, and 4 downscaling factors

Validation data: 100 high definition high resolution images are used for generating low resolution corresponding images.

DIV2K dataset has the following structure:

1000 2K resolution images divided into: 800 images for training, 100 images for validation, 100 images for testing.

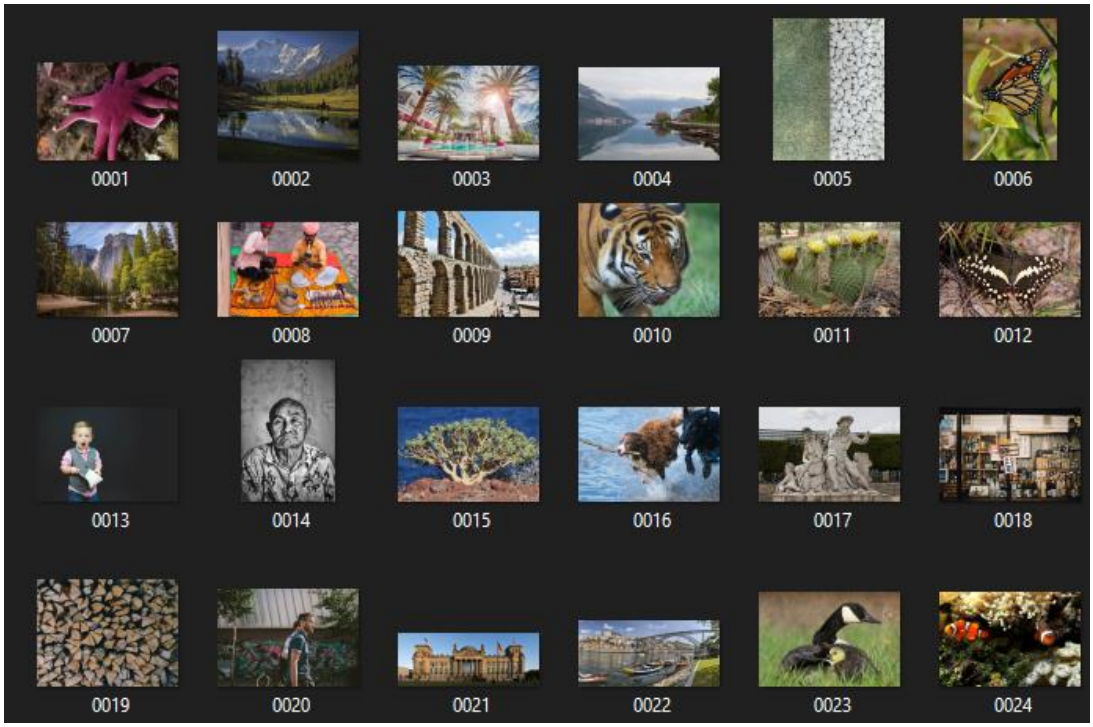

Fig 8. High resolution images for training

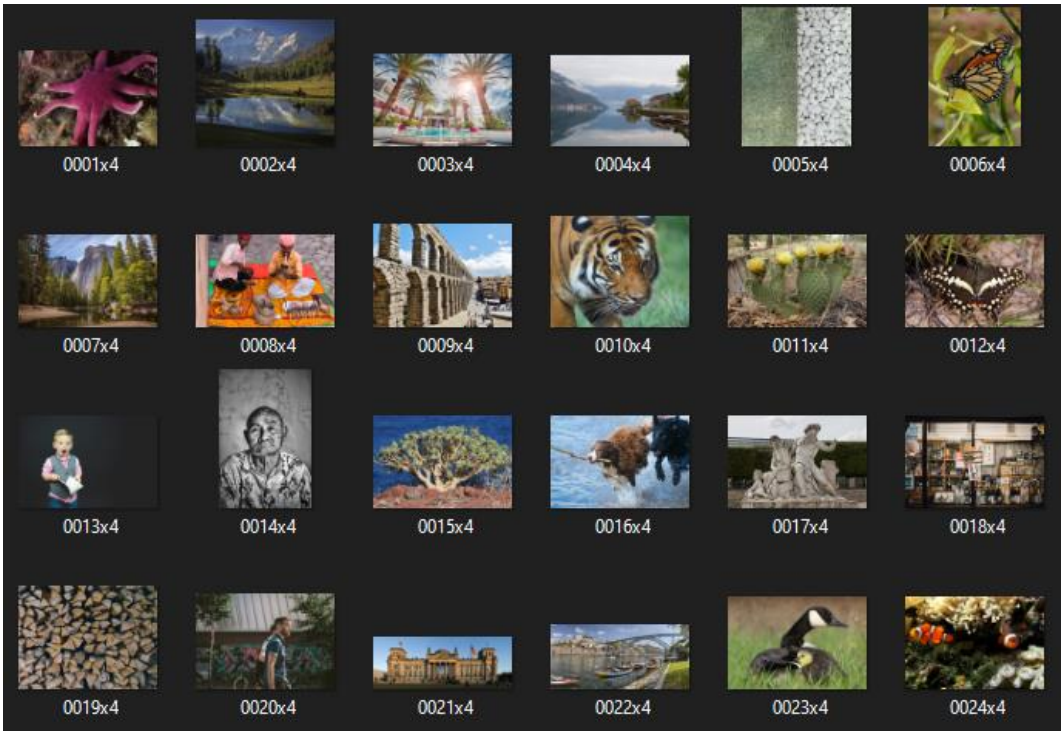

Fig 9. Low Resolution images 4x bicubic downsampled for training

CHAPTER 4

RESULTS AND APPLICATIONS

4.1 Results

Super pixelation is achieved. A low-resolution image after processing with SRGAN yields a super resolution image.

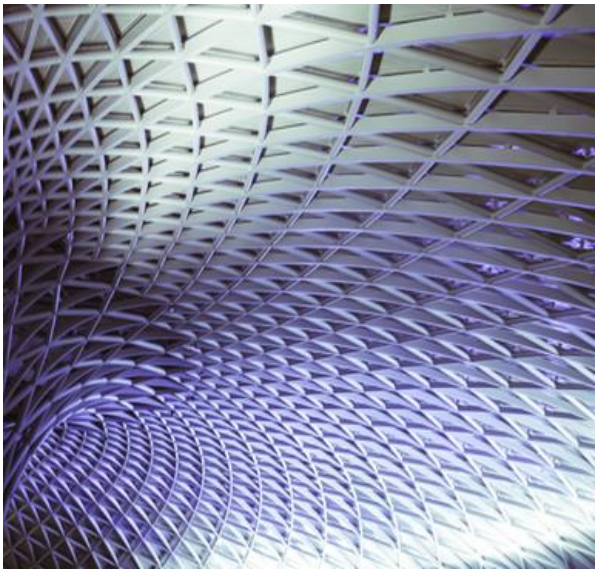

Fig 10. Results of the Generator Model

4.2 Applications

- Improving medical imaging systems
- Astronomical imaging
- Satellite imaging
- Super-resolution microscopy (SRM) encompasses multiple techniques that achieve higher resolution than traditional light microscopy.
- We can apply the same concept to low resolution videos, where video is treated as collection of images and then low resolution image will yield high resolution image and again collecting those output images will result in a high resolution video.

Applying different Activation functions and modifying truncation of noise and activation, results varying. quality of output.

Also for already lower quality images, the model works great. For images which are in high . resolution, while dividing grid, since each block will not contain great individual features so it is trade off between more quality and features, and sometimes, some important features may get distorted due to this. Also, with GAN, some new features can be introduced, and if it were not extracted by feature map in input, it will still generate that extra features.

REFERENCES

1. Yulun Zhang, Yapeng Tian, et al. 2018. Residual Dense Network for Image SuperResolution. arXiv:1802.08797.

2. Christian Ledig, Lucas Theis, Ferenc Huszar, et al. 2016. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. arXiv:1609.04802.

3. H. A. Aly and E. Dubois. Image up-sampling using total-variation regularization with a new observation model. IEEE Transactions on Image Processing, 14(10):1647–1659, 2005.

4. C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(2):295–307, 2016.

5. J. Johnson, A. Alahi, and F. Li. Perceptual losses for real-time style transfer and superresolution. In European Conference on Computer Vision (ECCV), pages 694–711. Springer, 2016

6. W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient SubPixel Convolutional Neural Network. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1874–1883, 2016.

7. https://medium.com/idealo-tech-blog/zoom-in-enhance-a-deep-learningbasedmagnifying-glass-part-2-c021f98ebede

8. https://github.com/idealo/image-super-resolution