Generation of Super Resolution Images using Deep Neural Networks

Final Report on Study and Development of Satellite Image Super-Resolution Techniques for the fulfilment of Summer Research Internship Project

by

Abhimanyu Bhowmik

REGIONAL REMOTE SENSING CENTER, EAST

National Remote Sensing Centre (ISRO)

Certificate

It is certified that the work contained in this thesis entitled **"Generation of Super Resolution Images using Deep Neural Networks"** by **Abhimanyu Bhowmik** has been carried out under my supervision and that it has not been submitted elsewhere for any purposes.

Dr. Arati Paul
Project Supervisor
Regional Remote Sensing Centre, East
National Remote Sensing Centre (ISRO)

Declaration

This is to certify that the thesis titled **"Generation of Super Resolution Images using Deep Neural Networks"** has been authored by me. It presents the research conducted by me under the supervision of **Dr. Arati Paul**.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for any purposes. Further, due credit has been attributed to the relevant state-of-the-art collaborations with appropriate citations and acknowledgements, in line with established norms and practices.

Abhimanyu Bhowmik
Summer Research Intern
Regional Remote Sensing Centre, East
National Remote Sensing Centre (ISRO)

Abstract

Name of the author: **Abhimanyu Bhowmik**

Organization: **Regional Remote Sensing Centre - East, NRSC(ISRO)**

Title: **Generation of Super Resolution Images using Deep Neural Networks**

Name of the supervisor: **Dr. Arati Paul**

Month and year of submission: **April 2023**

Super Resolution Images are required to properly perceive the intricacies of any given image. Satellite imaging is one such domain where details of an image must be preserved extremely carefully since image quality decreases drastically at high magnification. Due to developments in the disciplines of computer vision and deep learning, super-resolution which tries to increase image resolution by computational means has advanced recently. Convolutional neural networks built on a range of architectures, such as autoencoders, generative adversarial networks, and residual networks, have been used to tackle the issue. Few studies concentrate on single or multi-band analytic satellite imaging, whereas the majority of research focuses on the processing of images with simply RGB colour channels. Super resolution is a highly important and significant operation that must be carried out carefully in the realm of remote sensing. This work proposes a cutting-edge architecture AutoEn-GAN for the super-resolution of satellite images by blending autoencoders with an adversarial setting. All of the models' output is compared to the recently developed SR GAN, SR-ResNet, and EDSR models, and the traditional super-resolution benchmark using bicubic interpolation. Results of the AutoEn-GAN super-resolution method show a significant improvement over other state of the art methodologies such as SR-GAN.

Acknowledgements

Words cannot express my gratitude to my professor and senior scientist Dr. Arati Paul for her invaluable patience and feedback. Without the assistance of Amity University Kolkata, which kindly gave me the knowledge and experience I needed to conduct study in this specific area, I would not have been able to go on this adventure. Additionally, this endeavour would not have been possible without the generous support from the Regional Remote Sensing Centre- East, NRSC, ISRO who provided me with this research opportunity.

I am also grateful to the professors of Amity University Kolkata, especially Dr. Semati Chakraborty and my classmates from my University, for their help, feedback sessions, and moral support. Lastly, I would be remiss in not mentioning my family, especially my parents. Their belief in me has kept my spirits and motivation high during this process.

Contents

List of Figures

List of Tables

Abbreviations

HR	High Resolution
LR	Low Resolution
SR	Super Resolution
AE	AutoEncoder
CNN	Convolutional Neural Network
GAN	Generative Adversarial Networks
VAE	Variational Auto-Encoder
MSE	Mean Squared Error
MAE	Mean Absolute Error
EDSR	Enhanced Deep Residual Networks
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index Measure
RMSE	Root Mean Squared Error
SRGAN	Super Resolution Generative Adversarial Networks
VDVAE	Very Deep Variational Auto-Encoder
SRResNet	Super Resolution Residual Neural Networks

Chapter 1

Introduction

Super-resolution imaging is the process of up-scaling or enhancing the resolution of a lower resolution (LR) image to a higher resolution (HR) image. A high-resolution image has a higher pixel density and provides much more detailed information about the image. This is crucial for extracting minute details from satellite images and amplifying their zooming capacities with minimal distortion to the image. Images with high resolution are often desired in a vast number of digital image applications ranging from satellite imagery to medical imaging.

1.1 Image super-resolution

The resolution of an image may be raised in a number of ways. Hardware-wise, the easiest way to increase pixel sensors on a chipset is by shrinking the pixel size. Reduced pixel size results in shot noise and a reduction in photons hitting the sensor, deteriorating image quality. Therefore, the size of the image sensor can only be shrunk so far physically[11]. It would also be feasible to create bigger image sensors, but doing so would result in bulkier and costlier cameras. Further possibilities include expanding the sensor chip's size at the expense of raising the system's capacitance, leading to heavy and large hardware. Super-resolution involves the use of signal-processing techniques to construct HR images with better attributes than compared to a sampling grid[12]. This overcomes the constraints of sensor technology and imaging techniques. In order to eliminate signal loss in fused HR pictures, early research in the area of super-resolution was conducted in the 1980s in the frequency domain utilizing a number of under-sampled LR images[13].

1.2 Key Contributions

In this paper, several SR techniques have been examined and contrasted on the basis of numerous standard matrices. Deep learning techniques like GAN and CNN-based methods as well as conventional non-Deep learning approaches are examples of different methodologies. The project's primary contributions are:

- Our model is able to reduce artefacts while preserving the clarity of Images using Autoencoder with GAN architecture especially trained on satellite Images.
- Our framework can handle multiple types of files including .TIFF, .PNG, .JPEG and so on.
- It can preserve and project GEO - referencing information to the Super Resolution Images.
- It can handle Single as well as multiband LR Images which state-of-the-art SR models are not able to perform as those are build for RGB Images only.
- Our framework enhances Image for better feature extraction.
- Our framework can accurately match the colour of LR while generating the SR Images.
- It includes multiple dropout layers to prevent overfitting, thus increasing the accuracy and efficiency of the proposed framework.

1.3 Applications

As Deep Learning has grown in popularity, upscaling techniques utilizing Deep Learning have begun to appear in recent years. In the context of satellite imagery, there are several applications that can be considered. For example, it allows objects to be easily identified, made detectable, and more evident for manual or automated object detection techniques.

By upscaling satellite images, we can monitor land, sea, forest, and many other graphical properties of the earth's surface in minute details. The specio-temporal information contained in satellite images is very helpful for urban planning, observing changes in land usage, transportation and environmental planning.

Satellite imagery data and inferences are essential in the areas of disaster management, defence, agriculture, telecommunications, and aviation. Enhancing the images used for these purposes will undoubtedly improve the inference's accuracy and make prediction much more efficient.

1.4 Challenges

Since satellite images are very different from normal day-to-day images we get, handling satellite data is quite challenging. A few challenges faced while handling satellite images are as follows[14].

- The greater dimension of multi-spectral and hyperspectral data ranges from one to many bands. Each observation contains a significant amount of data due to the wide field of view and distance between the sensor and the scene. Encapsulating and retaining most of the features is one of the challenging tasks to solve.
- Making models that can be used with both types of satellite photos, which might be single or multiband, is a challenging problem. Additionally, for multiband pictures, the suggested model may exhibit some degree of bias toward a particular band that has to be eliminated.
- The images are in *.tiff* format and are huge in size. Processing the entire image in an instance requires tremendous resources and high-capacity computing instances. To make it computationally less exhaustive, the images must be shredded down into smaller patches before super-resolution.
- The satellite images are encoded in multiple different formats, and handling the mismatch of the bit depth, such as 8-bit encoded low-resolution or high-resolution image with a 16-bit encoded counterpart, is another challenge.
- Another major problem is adjusting the georeferencing of images after super-resolution imaging is performed on them. This adjustment of geo-referencing is crucial for map analysis, such as determining the exact coordinates of a point or image.
- A digital number (DN) is a numerical value assigned to each pixel in a satellite picture that records the intensity of electromagnetic radiation detected for the ground resolution cell represented by that pixel. The preservation of DN values during image super-resolution is required since it is a critical aspect of remote sensing for image classification.

1.5 Organization of the Report

The structure of the report is as follows: Chapter II incorporates the past state-of-the-art solutions to Super-resolution problems while Chapter III confers the theoretical background behind the project. Chapter IV and Chapter V consist of the methodology and experimentation procedures whereas, Chapter VI showcases the results and outcome of the study. Chapter VII provides a glimpse of the AutoEn-GAN desktop application and finally, Chapter VIII concludes the paper, highlighting the future research directions.

Chapter 2

Literature Review

The image reflects the overall perspective of image super-resolution and the common techniques used to achieve it. There are a number of techniques, including traditional classical techniques as well as machine and deep learning models. Deep learning techniques stand out among them as they have the It has the potential to deliver outcomes with consistently high accuracy. A few cutting-edge models are explained in further depth.

2.1 CNN-based model

The SRCNN[15], based of a 3-layered CNN framework[16] and employing a bicubic interpolated[17] low-resolution image as input to the model, is one of the earliest research that obtained decent results in the domain of superresolution. Since the Residual Neural Network(ResNet)[18] was developed, several studies have embraced it because it provides rapid training and improved performance for deep designs[19]. A few such state-of-the-art models like SRResNet[10], used in GAN implementation, and SRDenseNet[20] in which the size of the model is dilated for better results.EDSR(Enhanced Deep Residual Networks)[9] is another popular model for Single Image Super-Resolution.

2.2 GAN-based model

The Generative Adversarial Networks[21] for Super Resolution (SRGAN)[10] are entirely predicated on gated adversarial networks with the theory that the LR image is handed on to the generator network as input and the discriminator network attempts to make a

distinction between the original HR image and the SR output developed by the generator network[22]. Implementing the game theory method, GANs[23] execute an implicit density estimation and learn to produce from the genuine data distribution. GANs have historically been seen as very unstable and difficult to train, however, once training is done correctly, they are capable of producing very excellent sample quality[24]. Among the top performing image superresolution algorithms are the ESRGAN[25], which is an enhanced version of the SRGAN and Enhanced Net (Single Image Super-Resolution Through Automated Texture Synthesis)[26].

2.3 VAE-based model

In order to produce new data, the VAEs[27] sample from a learned latent space. They simulate an explicit, difficult-to-optimize density function, so the lower bound of likelihood is optimised in its place. Most recently, research has shown a significant improvement by using more flexible approximations and giving the latent variables structure[28]. These VAE have historically been known for their poor sample quality. Even though a recent study found that these models performed fairly well, GAN models continue to be the superior option for creating SR images. The SR VAE[29] is a recent example of an image superresolution system that uses a VAE architecture. To create a richer prior distribution in this work, the authors use a bijective model corresponding to RealNVP[30]. Another recent model that produces noticeably good results is the VDVAE SR model[8], which is VAE-based (Image Super-Resolution With Deep Variational Autoencoders).

2.4 Transformer based model

Transformers and attention-based architectures have dominated the area of Natural Language Processing since their debut by Vaswani et al. 2017[31], easily surpassing other techniques employed at the time like LSTMs or RNNs in terms of performance. BERT (Bidirectional Encoder Representations from Transformers) by Devlin et al. 2019[32], GPT (Generative Pretrained Transformer) by Radford et al.[33], T5 (Text to Text Transfer Transformer), and other popular adaptations are among the most widely used (Raffel et al. 2020)[34]. Transformers entered the field of vision as a result of the impressive outcomes in other fields[35]. They are now used for a variety of tasks including classification problems, object recognition, and even superresolution, with modelling techniques like Yang et al.[36] achieving excellent outcomes in superresolution, more particularly in

pattern recovery. A Texture Transformer made up of a learnable pattern extractor, a relevance embedding module, a hard attention unit for feature transfer, and a soft attention subsystem for feature synthesizing work together in the final model to accomplish this. Another feature of the model is that it uses 4 pictures as input to obtain further details about the low-resolution textures: the reduced image, the high-quality reference image, a 4x bicubic upsampled low-res image, and a reference image that has undergone consecutive down- and upsampling.

Chapter 3

Theoretical Background

This chapter covers all the theoretical groundwork required for understanding artificial intelligence, with a particular emphasis on deep learning and neural networks. The section contains information on a number of different models, including ANN, CNN, and GAN, as well as the prerequisites needed to use them. For a better understanding of the data given in the next chapter, section 3.5 is also addressed in relation to satellite imaging in the context of remote sensing.

3.1 Artificial Neural Networks

In the domains of AI, machine learning, and deep learning, neural networks enable computer programs to identify patterns and address common issues. They are additionally referred to as simulated neural networks (SNNs). Their structure and nomenclature are modelled after the human brain, mirroring the communication between organic neurons.

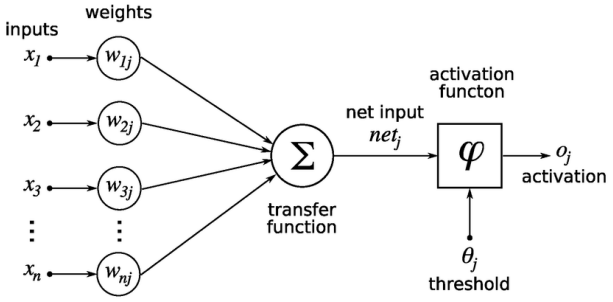

FIGURE 3.1: Architecture of a Single Neuron. [1]

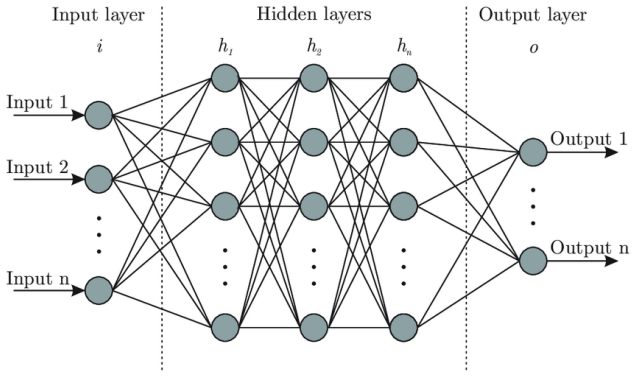

FIGURE 3.2: Figure of the architecture of a Multilayer Perceptron neural network. [2]

These networks mimic the function of the human brain. Deep learning techniques are based on neural networks, a branch of machine learning.

Combinations of fundamental neurons, also known as perceptrons (a fundamental unit depicted in the above figure 3.1), are placed in a network of layers to form a neural network (figure 3.2). An input layer, a processing layer, and an output layer are the three primary parts of a single neuron. Data is fed into the network through the input layer. The weighted summation of these inputs along with a bias (β) value is delivered to the transfer function as net input 3.1. The obtained output from the transfer function(3.2) is represented by the output layer (3.1.2).

$$y_{net} = \sum_i x_i w_i + \beta \tag{3.1}$$

w: Weight, β: Bias, y_{net}: Net input, x: Input

Equation 3.2, which includes a weighted summation and a non-linear activation function3.1.2 that resembles the threshold set by the cell body, may be used to describe an artificial neuron (known as ANN). Layers can be created by joining many neurons together.

$$y = f(\sum_i \omega_i x_i + \beta_i) \tag{3.2}$$

x_i: Network Input, ω_i: Weight, β_i: Bias, f: Activation function,y: Network output

Simple logical processes can be carried out by interconnecting layers in sequence. The network can carry out increasingly complicated jobs as the layers get more sophisticated. A multilayer perceptron (MLP), which consists of multiple completely linked layers, is illustrated in Figure 3.2. Each neuron in a neural network is linked to every other neuron present in the previous layer.

3.1.1 Perceptron Learning Rule

After Rosenblatt (1958) originally put out this concept, a single-layer feed-forward network is referred to as a perceptron. The sensory unit (input unit), the associator unit (hidden unit), and the response unit (output unit) are the three components that make up the perceptron network. The associator units that connect to the sensory units have fixed weights that are randomly given values of 1, 0, or -1. Both the sensory unit and the associator unit employ the binary activation function. The activation of the response unit might be 1, 0, or -1. The associator is activated using a binary step with a predefined threshold θ.

The output of a perceptron network is given by $y = f(y_{net})$ where $f(y - net)$ is described in the equation 3.3.

$$f(y_{net}) = \begin{cases} 1, & \text{if } y_{net} > \theta \\ 0, & \text{if } -\theta \leq y_{net} \leq \theta \\ -1, & \text{if } y_{net} < -\theta \end{cases} \tag{3.3}$$

The weight updating between the associator unit and the response unit uses the perceptron learning rule. The net output will compute the response for each training input and identify whether or not an error has occurred. In the instance of perceptron learning, the weight update is as displayed in the equations 3.4,3.5.

$$w_{new} = \begin{cases} w_{old} + \alpha t x, & \text{if } y \neq t \\ w_{old}, & \text{otherwise} \end{cases} \tag{3.4}$$

$$b_{new} = \begin{cases} b_{old} + \alpha t, & \text{if } y \neq t \\ b_{old}, & \text{otherwise} \end{cases} \tag{3.5}$$

Here in equation 3.4 and 3.5 , w and b are the weights and biases of the perceptron network. t is the target variable or the actual given label and α is the learning rate. A learning rate determines how fast or slow the algorithm converges to the optimum value. If the value of α is very high it takes very little time but it may not converge to the optimum solution, however a lower value of α increases the chance of convergence but takes more time to converge. The Learning Rate is a hyper-parameter which set by the user.

3.1.2 Activation Function

For neural networks, there are several activation functions that may be used. Their key characteristic is non-linearity. Convolution is a linear process. This the result that if several convolutions are applied successively to create an output, only one convolution can also provide the exact same result. As a result, a neural network with no non-linearities can essentially handle problems involving only linear regression. As a result, non-linearities are important in neural networks.

3.1.2.1 Sigmoid Activation

A popular activation function with a distinctive "S" shaped graph is the sigmoid activation, often known as the logistic function. It is limited between 0 and 1 and has continuous derivatives. Thus, it is used for the mapping of arbitrary values to this range (as demonstrated in eq. 3.6). The sigmoid function, for instance, has the benefit of being easy to compute for its first derivative and may be employed as the final activation in binary classification tasks though never for categorical classification.

$$S_x = \frac{1}{1 + e^{-x}} \tag{3.6}$$

$$S'_x = S_x(1 - S_x)$$

where x: Network Input, S_x: Sigmoid of input x

3.1.2.2 Softmax Activation

A sigmoid function adaptation that can handle inputs with multiple dimensions is the softmax activation function. Let z be the function's C-dimensional input vector. The softmax function returns C values in the range of zero to one such that all the values sum up to one. Equation 3.7 shows the mathematical representation of the function. The maximum value of the input vector gets reduced and ends up being the nearest to one. Because of this, categorical classifications where the network must "decide" which output value is most pertinent are perfect applications for the softmax function.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_j}} \qquad for\ j = 1, 2, ..., C \tag{3.7}$$

3.1.2.3 ReLU Activation

As early as 1975[37], the rectifying linear unit (ReLU) activation was employed to simulate the activity of biological neurons. When their total input surpasses a specific threshold, biological neurons begin to fire; otherwise, they stay essentially dormant. This behaviour is mimicked by the ReLU process, which maps all negative inputs to zeros while allowing all positive inputs to pass through, yielding zero as a threshold value(as in eq. 3.8). It has been shown to increase the efficiency of model training[38]. The derivative of the ReLU function is the Heaviside function, commonly referred to as the step function.

$$ReLU = \max(0, x) \tag{3.8}$$

where x: Positive Network Input

3.1.2.4 LeakyReLU and PReLU Activation

One of the factors in recent deep learning achievements has been ReLU. When used, it has shown better results than sigmoid. This is partly caused by the issue of vanishing gradients that arises with sigmoid activations. ReLU still has room for improvement, though. LeakyReLU [39] was invented, which does not zero off the negative inputs the way ReLU does. Instead, it leaves the positive input at its current value and multiplies the negative input by a small number (such as 0.02) [Figure: 3.3]. However, this has only slightly improved the accuracy of our models. By learning the small value (parameter) during training, we may enhance it and make our activation function more tolerant of other factors (like weights and biases). Herein lies the role of PReLU [3]. With a little increase in training costs, backpropagation allows us to learn the slope parameter.

$$f(x_i) = max(0, x_i) + a_i min(0, x_i) \tag{3.9}$$

In the equation 3.9, x_i is any input on the ith channel, and a_i is the negative slope, a parameter that may be learned.

- If $a_i = 0$, f transform into ReLU.
- If $a_i > 0$, f starts to leak. ReLU
- f becomes PReLU if a_i is a learnable parameter.

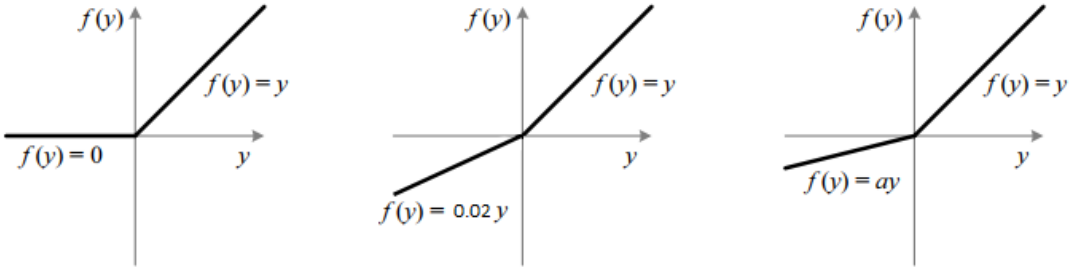

FIGURE 3.3: The figure shows the comparison graph of (Left) ReLU, (Middle), LeakyReLU and (Last) PReLU [3]

3.1.3 Optimizer

In order to train a neural network, we must adjust the weights for each epoch and reduce the loss function. An optimizer is a procedure or method that alters neural network properties like weights and learning rates. When mapping inputs to outputs, an optimization method determines the value of the parameters (weights) that minimizes the error. As a result, it aids in decreasing total loss and increasing precision. They also have a deep impact on the model's speed training. A deep learning model often has millions of parameters, making the process of selecting the proper weights for the model challenging. It highlights the importance of selecting an optimization algorithm that is appropriate for the model.

3.1.3.1 Gradient Descent

By scaling the present location with a learning rate and deducting the resulting value from the current position to move on, the gradient or steepest descent algorithm continuously determines the very next point. It will deduct the value from the objective function in order to minimise the function rather than the other way around. This process is carried out as follows(Eq. 3.10):

$$x_{n+1} = x_n - \eta \nabla f(x_n) \tag{3.10}$$

The gradient is scaled by a key parameter η, which in turn determines the step size. Performance is significantly impacted by it, which is known as the learning rate in machine learning.

- The gradient descent process will take significantly longer to optimise if the learning rate is very low, or it may approach its maximum number of iterations before discovering the best solution.
- If the learning rate is too high, the approach could bounce about the solution instead of converging, or it might even completely diverge.

3.1.3.2 Stochastic Gradient Descent

In order to overcome the limitations of gradient descent, stochastic gradient descent (SGD) is an iterative approach for maximising an objective function. Since it estimates the gradient using a randomly chosen subset of the data rather than the true gradient, which is determined from the complete data set, it may be thought of as a stochastic approximation of gradient descent optimization. This minimises the extremely high computational complexity, especially in high-dimensional optimization problems, allowing for quicker iterations at the expense of a reduced convergence rate.

3.1.3.3 Adam Optimizer

Along with its advantage of being simple, the SGD method has a number of drawbacks. One big issue is that it can be quite stagnant in some circumstances and become stuck in sub-optimal minima, like saddle points. The Adam Optimizer [40] was developed in order to address this issue along with many others using SGD. It maintains a very simple design and implementation while achieving faster convergence than the bulk of other optimizers in a variety of circumstances. The optimizer computes exponential moving averages of the first and second moments, or mean and uncentered variance of the gradients, to find updated steps specific to each individual weight.

3.1.4 Loss Function

To reduce algorithmic error, neural networks employ optimizing techniques such as stochastic gradient descent. We really use a Loss Function to calculate this mistake. It is used to express how well or poorly the model is working. Depending on the issue, there are several functions available to determine the loss based on the expected and actual value.

3.1.4.1 Perceptual loss function

When evaluating two distinct photographs that appear similar, such as the identical photo that has been displaced by one pixel, perceptual loss functions are utilised. The tool compares important discrepancies between photographs, such as those in content and appearance. The performance of the generator network depends heavily on the specification of the perceptual loss function l^{SR} utilised in this study. While l^{SR} is frequently modelled using the MSE 3.12, the enhanced version of the loss discussed in Johnson et al. [41] and Bruna et al. [42] has been created as a loss function that evaluates a solution with regard to perceptually important properties. The weighted sum of a content loss (l_X^{SR}) and an adversarial loss component is[10] and is used to define the perceptual loss as shown in Equation 3.11:

$$l^{SR} = \overbrace{\underbrace{l_X^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}}^{\text{perceptual loss (for VGG based content losses)}} \tag{3.11}$$

3.1.4.2 Content Loss

The pixel-wise MSE loss is described as follows:

$$l_{MSE}^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2 \tag{3.12}$$

The majority of cutting-edge methods for image SR depend on this as their primary optimization objective. The solutions to MSE (Eq. 3.12)optimization problems frequently lack high-frequency content, resulting in perceptually unpleasant solutions with excessively smooth textures, even if they frequently achieve extremely high PSNR (Eq. 5.3). The content loss is built on the principles of Gatys et al. [43], Bruna et al. [42], and Johnson et al.[41] and employs a loss function that is more closely related to perceptual similarity rather than depending on pixel-wise losses. The pre-trained VGG 19 model described in Simonyan and Zisserman [44]'s description of the ReLU (Eq. 3.8) activation layers serves as the foundation for the VGG loss. With $\phi_{i,j}$ denoting the feature map produced by the j-th convolution (after activation) in the VGG19 network before the i-th max-pooling

layer. The Euclidean distance between both the feature representations of a reconstructed image, $G_{\theta_G}(I^{LR})$, and the reference image, I^{HR}, is known as the VGG loss:

$$l^{SR}_{VGG/i,j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2 \tag{3.13}$$

Here, $W_{i,j}$ and $H_{i,j}$ outline the specific feature maps' dimensions within the VGG network.

3.1.4.3 Adversarial Loss

Along with the previously described content losses, the perceptual loss in GAN base models like SRGAN also includes the generative GAN element. The discriminator network is encouraged to prefer responses that exist on the diversity of natural images in an attempt to deceive the network. The generative loss l^{SR}_{Gen} is stated as follows based on the probabilities of the discriminator $D_{\theta_D}(G_{\theta_G}(I^{LR}))$ across all training instances:

$$l^{SR}_{Gen} = \sum_{n=1}^{N} -log(D_{\theta_D}(G_{\theta_G}(I^{LR}))) \tag{3.14}$$

The likelihood that the reconstructed image,$G_{\theta_G}(I^{LR})$, is a genuine HR image is given by $D_{\theta_D}(G_{\theta_G}(I^{LR}))$. Rather than minimising $log[1 - D_{\theta_D}(G_{\theta_G}(I^{LR}))]$ [21], we minimise $-log(D_{\theta_D}(G_{\theta_G}(I^{LR})))$ for improved gradient behaviour.

3.2 Convolutional Neural Network

An artificial neural network[3.1] and a convolutional neural network (CNN) vary primarily in a way that an ANN connects every neuron of a layer to every other neuron in the layer before it, but a CNN only allows input from just a restricted selection of units called the unit's receptive field.

CNN, as in figure 3.5, is performed using kernels(e.g. 1x1, 3x3, etc.) that filter out all but the horizontal and vertical variations and use the remaining information to generate features by performing element-wise products. Moreover, a nonlinear activation function takes place over all the elements. This layer filters the remaining variations, resulting in a more distinct set of features than before. The kernels used have learning weights that change after each iteration for better performance.

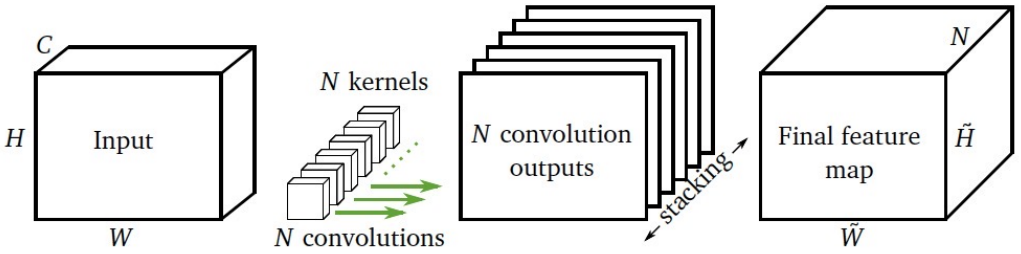

FIGURE 3.4: Figure capturing the working architecture of a CNN model [4].

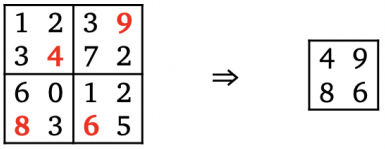

FIGURE 3.5: Figure showcasing the max-pooling operation in CNN [4].

By adjusting the size of a window that is slid over the input in such procedures, it is easy to vary the dimension of the data. The two most common pooling strategies are max and average pooling, where max pooling outputs each window at its maximum value. The result of the pooling procedure for each window in average pooling corresponds to determining the mean value.

3.3 Generative Adversarial Networks

A revolutionary generative technique in the field of deep learning is generative adversarial networks. GANs are two competing neural networks that were first developed by Goodfellow,et al. [21] in 2014.

A random input noise vector, z, is translated into a data space by one of the networks, called a generator. The generator aims to map z to the actual data x as nearly as it can. The function G (z, g), where G is some modified version of a multi-layer convolutional neural network and g specifies its parameters, does this. The discriminators' job in the second network is to calculate the likelihood that the x genuinely originated from the input rather than as an outcome of G. (z). The function D(x, d), where D is also a multi-layer convolutional neural network while d stands for its parameters [21], does this. The objective of training the discriminators is to ensure that they can distinguish between inputs that are created data (G) and genuine data (X) with accuracy (z). When supplied with synthetic results, the generator is taught to minimise equation 3.15. To put it another

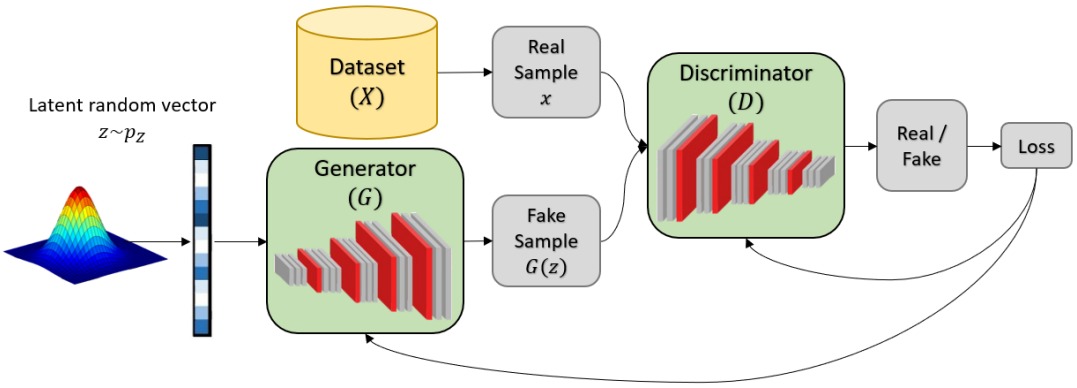

FIGURE 3.6: A schematic view of a GAN. The generator takes a noise vector z as input and maps this to G(z). The discriminator then receives inputs that are either x (real data) or G(z) (generated data). The discriminator then outputs a prediction if the input data is either fake (0) or real (1). These outputs are used to train the network.[5]

way, the generator is taught to make the discriminator anticipate the created data as actual data.

$$log(1 - D(G(z))) \tag{3.15}$$

When two networks are used for this purpose, they engage in a minimax game as shown in equation 3.16. Argumentative training is the name given to this type of instruction [45].

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(x)))] \tag{3.16}$$

By using the loss function, the GAN's goal is to get the synthesized data distribution closer to the genuine distribution. The loss function in equation 3.17 might be determined to be equivalent to the Jensen-Shannon divergence which is stated in equation 3.17 when the discriminator operates at its best.

$$JS_{div}(P \parallel Q) = \frac{1}{2}K(P \parallel M) + \frac{1}{2}K(Q \parallel M) \tag{3.17}$$

where K is the KL divergence and $M = \frac{1}{2}(P + Q)$ [46]. The KL divergence may be determined using the formula 3.18:

The maximum likelihood estimator, known as KL divergence, will approach infinite if the bases for the two distributions do not coincide. The logarithm of two is produced

$$K(P \parallel Q) = \sum_i P(i) log \left(\frac{P(i)}{Q(i)} \right) \tag{3.18}$$

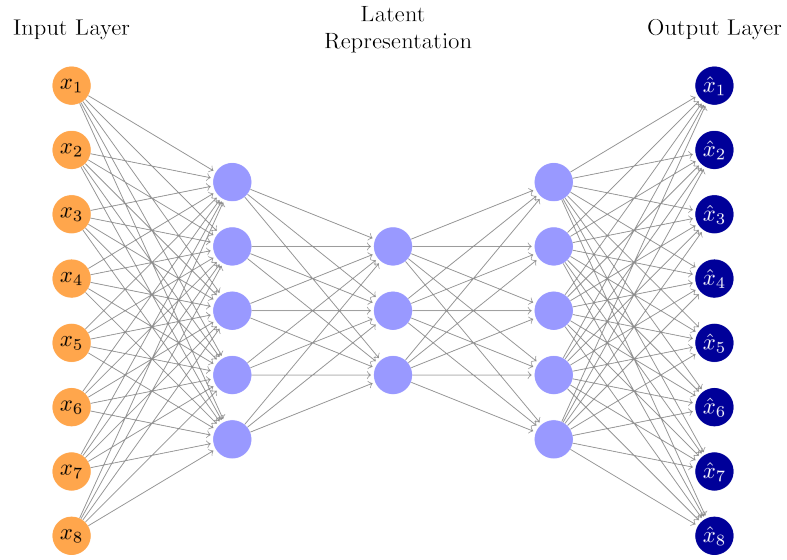

FIGURE 3.7: An architecture of Autoencoder [6]

while combining several KL divergences with the Jensen-Shannon divergence in the same situation with no overlap of the distributions. This results in certain difficulties during adversarial training with the loss function proposed in 3.17.

3.4 Autoencoder

An autoencoder (AutoEn) is a method for unsupervised learning that identifies a mapping from high-dimensional inputs to low-dimensional representations, such as latent vectors [47]. The dimensions of this hidden or latent layer are less than the input data in order to avoid the autoencoder from learning the identity transform. Encoders (Enc) and decoders (Dec) are two symmetric neural networks that makeup autoencoders. The latent space z, also known as the bottleneck layer because of its lower dimensions compared to the input data, is created when the encoder maps the input x into the latent space. Following that, the decoder will synthesise and reconstruct the input data x using this latent space. The aim of the reconstruction, also known as reconstruction loss, appears to be to minimise the difference between x and x. The reconstruction loss is described in the equation 3.19.

$$\min_{\theta,\phi} L(\theta,\phi) = \frac{1}{N}\sum_{i=1}^{N} ||x_i - Dec_\theta(Enc_\phi(x_i))||_2^2 \tag{3.19}$$

In order to guarantee that the reconstruction will be sufficiently comparable to the original data, the latent space must be taught the most significant feature changes of the original data. The whole network (encoder and decoder) is jointly trained to obtain a low reconstruction loss. In both the encoder and the decoder, autoencoders can contain a single layer, although two or more layers are more typical in these networks. The autoencoders in this situation are deep autoencoders 3.7.

Autoencoders may be divided into four categories: denoising autoencoders, contractive autoencoders, sparse autoencoders, and variational autoencoders.

Denoising autoencoders [48] attempt to denoise or recover a clear picture from a randomly partly distorted input, as their name indicates. These autoencoders fundamentally work on the principle of forcing the hidden layer to acquire robust features rather than merely the identity function, which would lead to yet another malformed picture.

Contractive autoencoders, often known as CAEs [49], is a little more complicated since they add a new component to their loss function to make the model more resistant to minute changes in the input values.

The hidden layer in sparse autoencoders [50] typically has dimensions greater than the input. Only permitting a few of the hidden neurons to be active at any given time solves the issue of avoiding the network to learn the identity function.

Lastly, variational autoencoders, or VAEs, have an autoencoder-like design (Fig. 3.7). Variational autoencoders use a variational strategy for latent representation learning, which is the primary distinction between them and other types of autoencoders.

3.5 Remote Sensing

Remote sensing is the acquisition of information from a distance. Space agencies observe Earth and other planetary bodies via remote sensors on satellites and aircraft that detect and record emitted and reflected energy. Remote sensors, which provide a global perspective and a wealth of data about Earth systems, enable data-informed decision-making

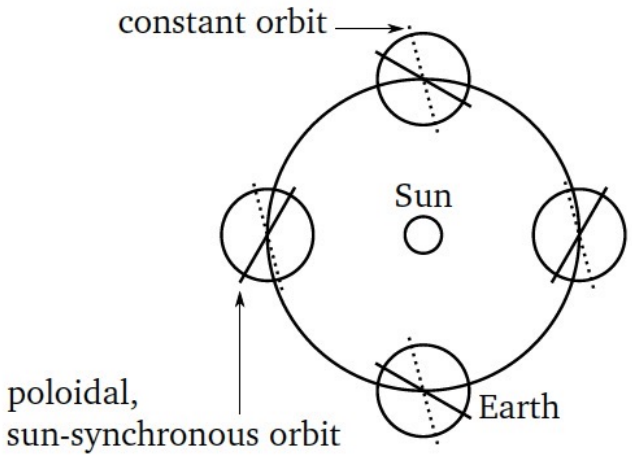

FIGURE 3.8: In the sun-synchronous orbit (solid), the earth's rotation plane revolves once a year. The non-spherical form of the earth imparts intrinsic angular momentum that propels this revolution. The rotational plane of an unaltered orbit, on the other hand, preserves its alignment [4].

based on the current and future state of our planet. Some common methods followed by space agencies for remote sensing are mentioned below:

- **Observing with the Electromagnetic Spectrum:** The oscillation of charged particles creates electromagnetic energy, which propagates as waves across the atmosphere and the emptiness of space. The amount of energy that each object on Earth reflects, absorbs, or transmits varies depending on its wavelength, and each object has a distinct spectral fingerprint.

- **Sensors:** To measure the energy which is bounced back, sensors or equipment on board satellites and aeroplanes either utilize the Sun as a light source or create their stream of illumination. Passive sensors are those that rely on solar energy from the Sun, and active sensors are those that have their internal energy source. Altimeters, scatterometers, and other radio detection and ranging (radar) sensors are examples of active sensors.

- **Resolution:** The utilisation of sensor data depends in part on resolution. Depending on the satellite's orbit and sensor configuration, the resolution can change. For each dataset, there are four different forms of resolution to take into account: radiometric, spatial, spectral, and temporal.

- **Data Processing and Analysis:** Before the majority of researchers and consumers in applied science can make use of remote sensing data from instrumentation

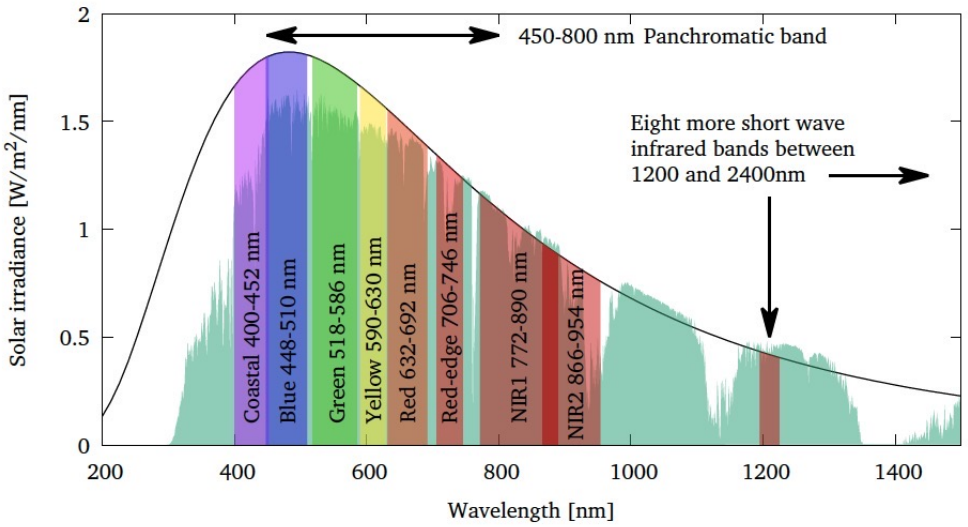

FIGURE 3.9: The wavelengths of the WorldView-3 imaging bands [3, 4], the solar radiation spectrum, and the spectral response of a pure black body at 5500 K [4].

onboard satellites, processing is necessary. They may be processed and used for several purposes, including agriculture, water resources, health, and air quality.

We will use satellite-sourced pictures for this project. Knowing the origins of the data is necessary in order to comprehend and analyse it. We will go into more detail about the difficulties in using the data source and discuss it in this part.

3.5.1 Satellite Imagery

There are several distinct remote sensing satellites in orbit now, each with a particular purpose. Mainly optical sensors will be the subject of our attention since they are the most accessible and relevant for deep learning image processing jobs. The Landsat satellites, for example, have been in orbit for several decades, providing the chance to analyse the temporal history of land cover. The majority of optical remote sensing satellites travel in poloidal, sun-synchronous lower earth orbit, with mean altitudes ranging from 450 km to 800 km.

Two key benefits of the sun-synchronous orbit 3.8 are as follows: Firstly, the satellite may be positioned in continuous sunshine, and secondly, with the exception of seasonal variations, a certain location on the planet is always shot in the same lighting circumstances. This makes captured photos more comparable. These satellites can return to each region of interest within a few days or even once every day because the sun-synchronous

orbital plane and the planet both spin slowly. Since the satellite's movement already covers one imaging dimension, many satellites scan the globe using line sensors, although two-dimensional sensors are also used. A zone of a specific width is captured by the satellite-based on elevation, focal length, and sensor (the swath width) at ground level.

A specific range of wavelengths is used by each satellite to acquire photographs. The WorldView-3 satellite's imaging bands, which we shall employ subsequently, are summarised in Figure 3.9.

The near-infrared, red, green, and blue bands are crucial for classifying land use patterns. Longer wavelengths are typically employed for geological surveys, military purposes, or fire detection. The quantity of vegetation covering the ground may be determined, in part, by comparing the red and red-edge or NIR bands.

Chapter 4

Methodology

In this section, we will take a closer look at the models employed in this work. The models used are EDSR, SRGAN and SR(PRE) or SRResNet. Because in the Deep Learning landscape, all of them are prominent state-of-the-art methods. Other very new methods, including VAE and transformers, are not performing well enough till now. Though very few recent studies show some promising results, these models need robust testing and improvement to perform better consistently than GAN and CNN-based models. All of the models mentioned here were compared against each other and with the Bi-cubic interpolation technique which is a classical SR method.

4.1 Bi-cubic Interpolation

Bicubic interpolation is a two-dimensional approach for enhancing and expanding digital photographs using cubic splines as well as other polynomial techniques. When upscaling or resampling an image, retouchers and editors frequently employ it in computer image manipulation. The 2 interpolation methods that are often used are adaptive and non-adaptive. While non-adaptive approaches treat all pixels identically, adaptive techniques vary depending on what they will be interpolating. While non-adaptive algorithms include the nearest neighbour, bilinear, bicubic, spline, etc., adaptive strategies are utilised in many proprietary approaches in specialist professional picture editing software. Lagrange polynomials, cubic splines, or cubic convolution techniques can be used for bicubic interpolation. Bicubic interpolation produces nicer interpolated surfaces than either bilinear interpolation or nearest-neighbour interpolation. Both Lagrange polynomials, cubic splines, or the cubic convolution technique can be used for bicubic interpolation.

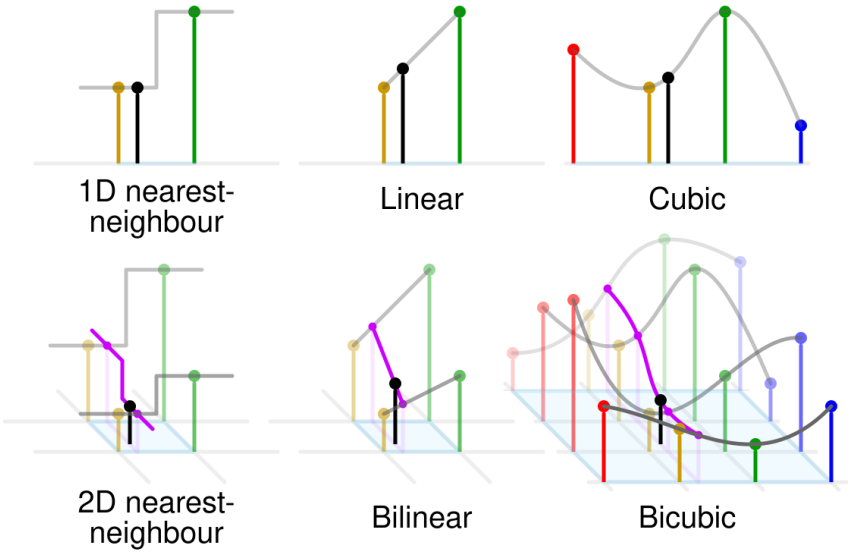

FIGURE 4.1: Bicubic interpolation is compared to a few 1- and 2-dimensional interpolations. Red, yellow, green, and blue dots represent the adjacent samples, while black dots represent the interpolated position. Their values are in line with their heights above the surface. [7]

When speed is not a concern, bicubic interpolation in image processing is frequently preferred to nearest-neighbour or bilinear interpolation in photo resampling. Bicubic interpolation considers 16 pixels (4 x 4) as opposed to bilinear interpolation's consideration of just 4 pixels (2 x 2). Bicubic interpolation produces smoother and less erratic images when resampling images. The Bicubic Interpolation methodology is applicable to both the upsampling and downsampling of pictures for various applications. It has been used in several works as a common benchmark to measure all other machine or deep learning models against because it is one of the most well-known classical super-resolution techniques. It is also used in the same way in this research. In accordance with Gavade and Sane's 2013 [51], the bicubic interpolation is as follows. Assume that the values of the function f and its derivatives f_x, f_y, and f_{xy} are known at the unit square's four corners (0, 0), (0, 1), (1, 0), and (1, 1). Consequently, the interpolated surface may be expressed as Equation 4.1. The interpolation problem then consists of determining 16 coefficients

$$p(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j. \tag{4.1}$$

a_{ij}.

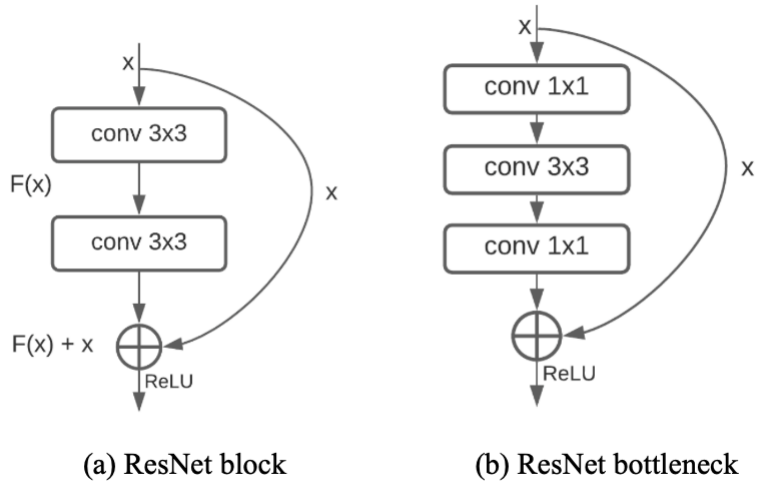

FIGURE 4.2: The ResNet architecture [8]

4.2 SR(PRE)

The SR-ResNet model is referred to as SR (PRE) in this study as it only uses and trains the Generator Network of the SRGAN model (described in section 4.4). A Resnet or residual network called SR-ResNet is utilised for Super Resolution tasks. In the subsequent section 4.2.1, the ResNet's architecture and workings are explained. The weights from this pre-trained generator, SR (PRE), are further employed as input weights in the SRGAN model for experimentation-related GAN fine-tuning.

4.2.1 ResNet

Residual networks, commonly known as ResNets, were initially developed by He et al. 2016 [18] to address the degradation problem in deep neural networks, in which the efficiency of network systems with several layers quickly declines just before the model is ready for convergence. As illustrated in Fig. 4.2, which shows one residual block, the ResNet is made up of neural networks with skip connections.

The outcome of the stacked layers is combined with identity mapping performed by the skip connections to produce $F(x) + x$. In this manner, rather than learning via $F(x)$ in the usual situation without the skip connection, the identity mapping x may easily be constructed in the scenario when it is the best solution by simply setting $F(x)$ to zero. 34 convolutional layers, 3x3 filters, and a ReLU activation function are the components

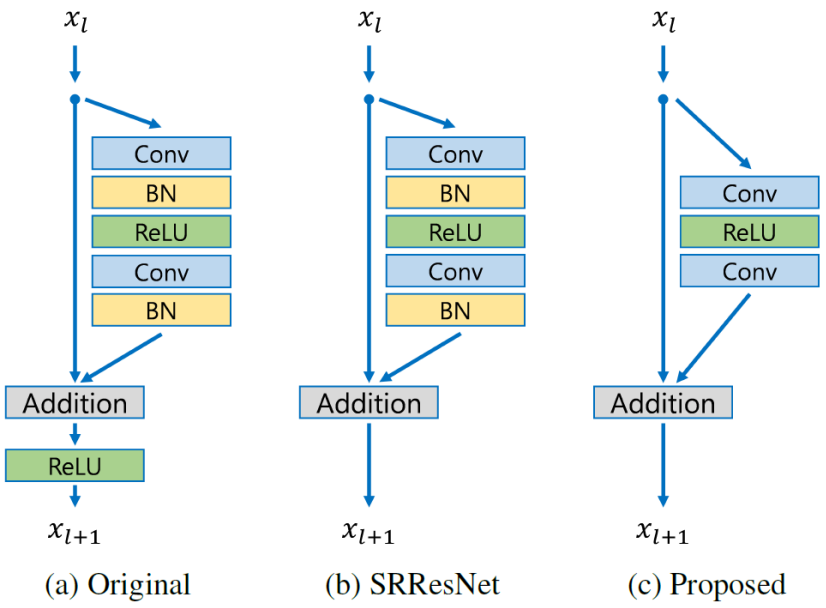

FIGURE 4.3: The EDSR architecture compared with SRResNet and Original ResNet structure [9]

of the original ResNet(as shown in fig. 4.2.a). Another variation of the ResNet is the bottleneck architecture designed for deeper networks, which has a structural component that has three layers rather than two, the centre one of which acts as a bottleneck with 3x3 convolutions, and the other two of which function as the rest of the network (as portrayed in fig. 4.2.b).

4.3 EDSR

EDSR, or Enhances Deep Super Resolution Model, is a deep neural network used for image super-resolution. The network design of EDSR was invented by Bee Lim et al. [9] in 2017. EDSR is based on the SRResNet architecture (4.2) and is simplified by analyzing and deleting unnecessary modules. Numerous super-resolution approaches have been compared to EDSR. SRResNet-like topologies are employed. Two layers of Convolution and one layer of ReLU activation function make up EDSR, which was heavily influenced by SRResnet. However, the batch normalisation of the residual block has been removed by the authors for increased effectiveness and performance. The architecture of residual block and single scale EDSR is given in figure 4.3.

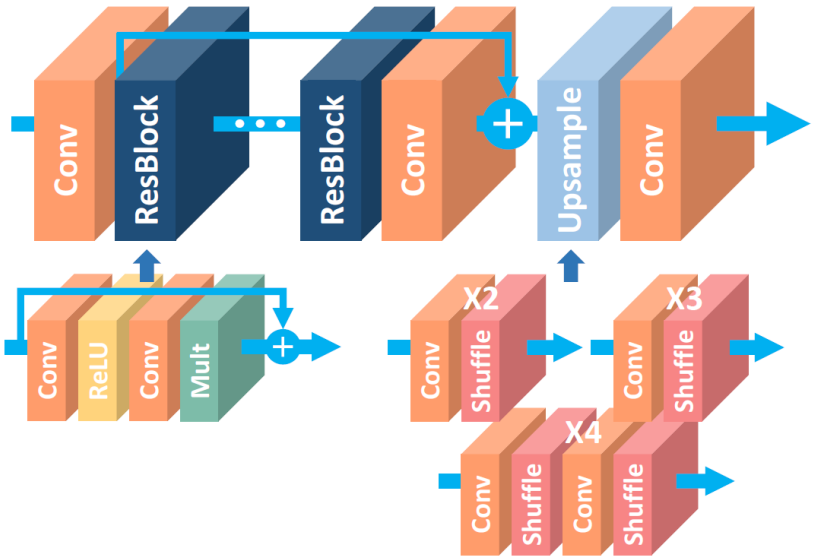

FIGURE 4.4: The architecture of single scale EDSR residual block [9]

4.3.1 Single-Scale EDSR Model Architecture

- The depth (number of layers) B and width (number of feature channels) F of a typical CNN architecture occupy around O(BF) memory with O(BF2) parameters.
- Each layer consists of a residual block and convolution layers. Outside of the residual blocks, ReLU activation layers don't exist.
- Thus, while taking into account restricted computational resources, raising F rather than B can optimize the model's capability.
- The learning becomes inconsistent even as number of feature maps rises, though.
- As recommended by Inception-v4 by Szegedy et.al [52], residual scaling with a factor of 0.1 is done at the residual path prior putting back to convolutional path. Constant scaling layers are positioned after the final convolution layers in each residual block. For improved computing efficiency, this layer can be combined with the prior convolution layer during the testing phase.
- Outside of the residual blocks, ReLU activation layers don't exist.
- Baseline model: Since each convolution layer only uses 64 feature maps, residual scaling layers really aren't needed.
- EDSR: By setting B = 32 and F = 256 with a scaling factor of 0.1, the basic model is extended.

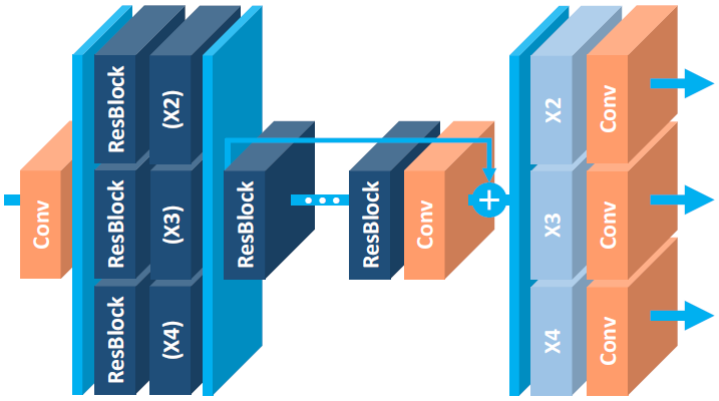

FIGURE 4.5: Multi-Scale EDSR Model [9]

4.3.2 Multi-Scale EDSR Model Architecture(MDSR)

- Super resolution at various scales is a challenge that is connected to others.
- Baseline Model: As seen in figure 4.5, the basic model is a single main branch with B = 16 residual blocks allows for the majority of the parameters to be shared across various sizes.
- In order to limit the variance resulting from input pictures of various scales, pre-processing modules are first placed at the head of networks. Two leftover blocks with 55 kernels make up each pre-processing module. The scale-specific portion can be shallow while the wider receptive field is covered in the early stages of networks by using larger kernels for pre-processing modules.
- Scale-specific upsampling components are placed in parallel at the multi-scale model's endpoint for performing multi-scale reconstruction.
- The basic multi-scale models has only 3.2M parameters, with comparable performance to the single-scale models, while the baseline single-scale models for the three distinct scales have around 1.5M parameters apiece, totalling 4.5M.
- As the scale-specific components are lighter than residual blocks components, the MDSR requires just 2.5 times as many parameters as the basic multi-scale model, with B = 80 and F = 64, or around 5 times greater depth.

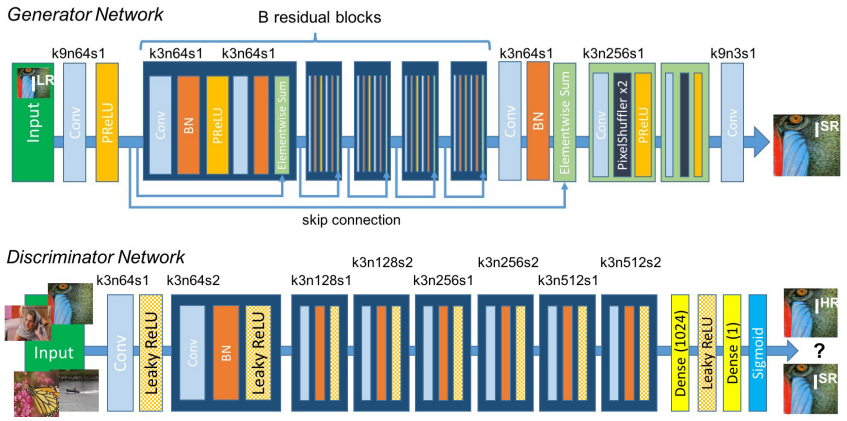

FIGURE 4.6: Architecture of the Generator and Discriminator Networks with the associated kernel size (k), number of feature mappings (n), and stride (s) for each convolutional layer. [10]

4.4 SRGAN

Based on ResNet architecture and tuned to create a new perpetual loss function that comprises of an adversarial loss and a content loss, SRGAN is a generative adversarial network for single picture super-resolution. Utilizing a discriminator network that has been trained to distinguish between the super-resolved pictures and the original photo-realistic images, the adversarial loss drives the output to the natural image manifold. The authors also employ a content loss that is driven by perceptual similarity rather than similarity in pixel space. It is a state-of-art network to estimate a super-resolved image from a low-resolution image with high upscaling factors(4x). The diagram 4.6 illustrates the architecture of the SRGAN Model for various feature maps, kernel sizes, and strides.

4.4.1 Generator Network

The Generator Network seeks to produce SR images that closely resemble the actual images, making it challenging for the Discriminator to categorise them. It basically comprises of an SR-ResNet generating network, where the low-resolution input is routed via a Parametric ReLU [section: 3.1.2.4]layer after first passing through a convolutional layer made up of 9x9 kernels and 64 feature maps. It is clear that the Parametric ReLU serves as the main activation function across the whole generator design. Since it is one of the finest non-linear functions for this specific purpose of translating low-resolution pictures to high-resolution images, the Parametric ReLU was chosen.

Several B residual blocks are used in the subsequent layer of the feed-forward fully convolutional SR-ResNet model. Following a batch normalisation layer, a Parametric ReLU [section: 3.1.2.4] activation function, a convolutional layer with batch normalisation, another convolutional layer with batch normalisation, and a final elementwise sum technique, each residual block has a convolutional layer with 3x3 kernels and 64 feature maps. The final result produced from the elementwise sum technique combines the feed-forward and skip connection outputs.

The remainder of the generator model is produced once the remaining blocks have been created, as seen in the picture 4.6. In this generator model architecture, the convolutional layer is 4x upsampled before being utilised to create the super-resolution pictures. The pixel shufflers insert values into the height and width dimensions using values from the channel dimension. With the help of two pre-trained sub-pixel convolution layers, the image's resolution is enhanced.

4.4.2 Discriminator Network

To differentiate between the produced SR pictures and the raw HR image, the discriminator architecture is utilised. It is designed to assist a standard GAN method [Fig:3.6] in the best possible way. The discriminator and the generator are concurrently getting better as they compete with one another. The generator tries to create realistic pictures so that it can avoid detection by the discriminator network while the discriminator network searches for bogus images. Similar principles apply to how SRGANs operate, where the generative model G aims to deceive a differentiable discriminator D that has been taught to discern between genuine pictures and super-resolved images.

4.5 Proposed Methodology

Variational Autoencoders(VAE) or Autoencoders(AE) in general tend to generate a bit blurry images as they use Gaussian distribution to sample from. On the other hand, GAN typically tends to generate more sharp images. But in the case of satellite images, this increase in sharpness is one of the reasons for artefact generation. To combat these issues, the proposed AutoEn-GAN framework is a combination of Auto-encoding with a traditional SR-GAN network. Although very deep VAE has been used in the past for SR images, the concept of AutoEn-GAN remains novel and is created by the author. Moreover, all the existing models are trained on RGB images and can't handle any other number of

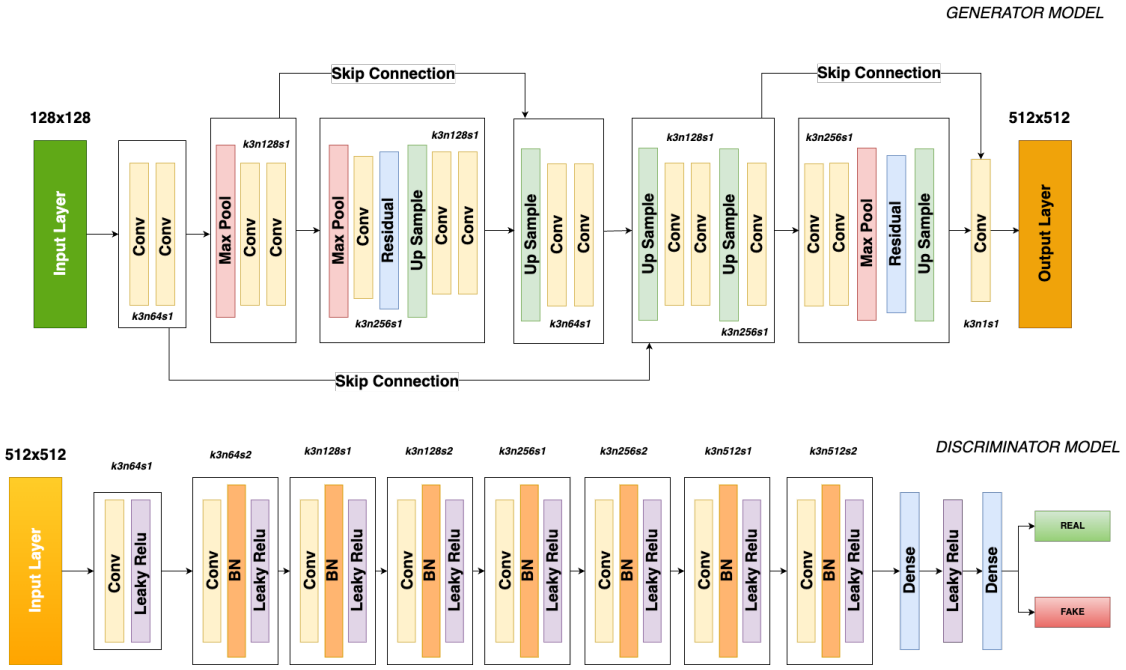

FIGURE 4.7: The Generator model and Discriminator model of the proposed AutoEn-GAN framework

bands as input images. The AutoEn-GAN model takes the input images and converts them in greyscale and the Generator Network seeks to produce SR images that closely resemble the actual images, making it challenging for the Discriminator to categorise them. The novel contributions of this method can be found in Section 1.2.

4.5.1 The Generator Model

The Generator Network tries to develop SR images that are remarkably similar to the original images in order to deceive the Discriminator into classifying them as real images. The Generator model takes low-resolution inputs and feeds them through two convolution networks with 3x3 kernels and 64 channels to generate the super-resolution output. Before reaching the Residual block, the image is passed through several max-pooling and convolution layers. The structure of the Residual block is similar to that of the SR-Resnet block; however, it uses LeakyReLU as its activation function instead of PReLU since the former provided better results while being more computationally efficient as shown in Figure 4.8. The output of the Residual block is then 3x upscaled and convoluted, which increases the resolution of the generated images. The skip connections between the blocks help to reduce the amount of information lost in the network, improving the performance of the GAN and ensuring that more details are included in the generated images.

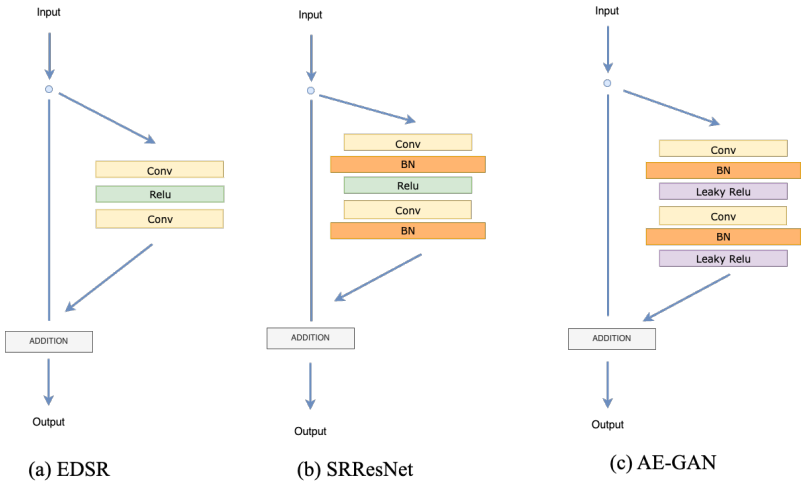

FIGURE 4.8: Comparison of Residual block among EDSR, SRResnet (SR-GAN & SR-PRE) and AutoEn-GAN

On careful examination of the generator model, one can observe that the feature map undergoes repeated reduction and enlargement in size. Specifically, the feature map is reduced from 128x128 to 32x32 after the 5th convolution block, through the Convolution and Max Pool Layer. Later, the feature map is incrementally enlarged using Up-sample layers, resulting in an output image size of 512x512. Additionally, the transfer of network information between layers is facilitated by skip connections, which are crucial in this process. These characteristics resemble those of an Autoencoder architecture, and as a result, we have named this network AutoEn-GAN, or an Autoencoder-like GAN framework.

4.5.2 The Discriminator Model

The proposed discriminator model is the same as that of the SRGAN model. The discriminator architecture is highly logical and simple to comprehend. It employs a Leaky ReLU activation function after the first convolutional layer. For this architecture, the α value in Leaky ReLU's [section:3.1.2.4] is set at 0.2. The batch normalisation layer as well as the Leaky ReLU activation function are then followed by a number of convolutional layers recurrent blocks. After five of these repeating blocks, the dense layers and sigmoid activation function are used to carry out the classification activity. The baseline convolutional size is 64 x 64, which is doubled by 2 after two full blocks of each until we reach the 8x upscaling factor of 512 x 512. The generator learns more efficiently and generates improved outcomes thanks to this discriminator model.

4.5.3 Residual Block

The Generator network includes multiple Residual Blocks, shown as light blue blocks in figure 5.1. AutoEn-GAN's Residual Block is similar to SR(GAN), but with some improvements inspired by SR(GAN)'s success. Figure 4.8 illustrates the differences and similarities between the Residual Blocks of EDSR, SR(GAN), and AutoEn-GAN. AutoEn-GAN uses Leaky Relu instead of Relu, which is less expensive than Parametric-Relu but better than Relu. Additionally, AutoEn-GAN has an extra layer of Relu activation compared to SR-Resnet. These modifications have impacted AutoEn-GAN's performance.

The AutoEn-GAN architecture is unique and distinguishable from the existing VAE solutions since it uses residuals like blocks, skip connections and dropout regularisation and nested residual blocks are used for creating the latent space/bottleneck layer.

4.5.4 Advantages

- The proposed AutoEn-GAN method is a lightweight deep learning model without trading off the output quality.
- The proposed framework can accept input in any number of bands and provide an output in grayscale format.
- The output SR-image produced by AutoEn-GAN is much cleaner and artefact free than existing classic models.
- The accuracy is much higher when compared to other Super-resolution models.

  Further direct comparison with the existing methodologies and AutoEn-GAN based on the obtained results is discussed in section 8.1

Chapter 5

Experimentation

This chapter discusses the experimental process, dataset, and data preparation and post-processing with all the relevant details as in figure 5.1. The data description as well as georeferencing and meta-data information is mentioned in detail in the section 5.1. As the dataset contains georeferencing information, the data pre-processing is very different from regular images as mentioned in the section 5.2. The data is split into training and testing data [Section: 5.3] for the learning of the models in the subsequent stages of the

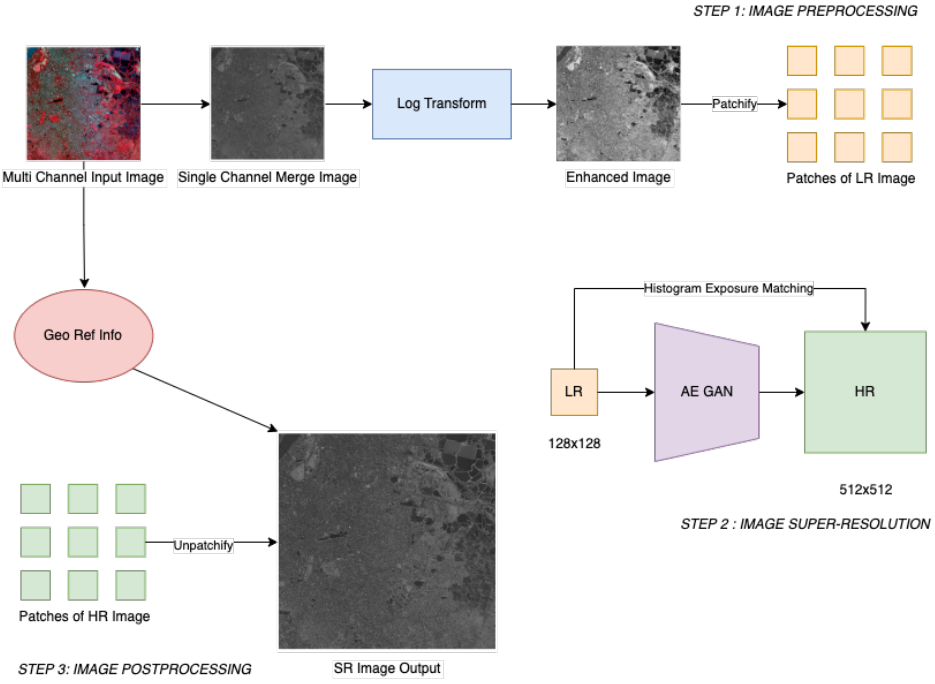

FIGURE 5.1: The overall framework of the proposed model

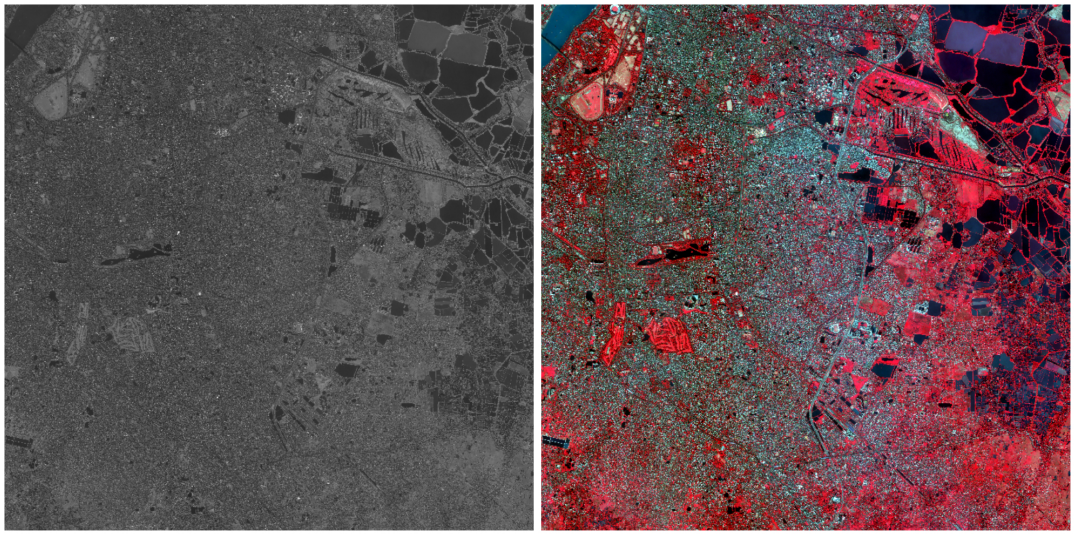

FIGURE 5.2: Data is displayed in the figure. The HR picture with one band is exhibited on the left, while the LR image with three bands is shown on the right.

experimentation. The study uses relevant pre-trained deep learning models for super-resolution based on which a novel methodology has been proposed. The paper implements a GAN framework which is customised to fit in the satellite images of a single channel 5.4. The results of all pre-trained, trained (transfer learning) as well as the proposed model were compared to a benchmark classical SR approach (bicubic interpolation) as well as with each other. The standard evaluation matrices which are used to measure the performance of the models with their theoretical formulation are shown in section 5.6.

5.1 Dataset

Two satellite images were used for the training and testing of the models. Among them, one is a High Resolution (HR) image and the other one is a Low Resolution (LR) image as portrayed in fig 5.3. The HR image is in greyscale since it contains only 1 band. On the contrary, the LR image, having 3 bands, is in false colour RGB format. As both are satellite images, they also contain projection and Geo-Transformation information. The projection details of both images are the same since they have been captured at the same location and time. The project details are given in table 5.1 and code snippet 5.1.

```
Image Metadata (Coordinate System):
```

```
PROJCS["WGS 84 / UTM zone 45N",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",87],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","32645"]]
Corner Coordinates:
Upper Left  (  635809.989, 2496067.008) ( 88d19'15.40"E, 22d33'57.23"N)
Lower Left  (  635809.989, 2482963.008) ( 88d19'11.36"E, 22d26'51.17"N)
Upper Right (  649945.089, 2496067.008) ( 88d27'30.23"E, 22d33'52.96"N)
Lower Right (  649945.089, 2482963.008) ( 88d27'25.76"E, 22d26'46.92"N)
Center      (  642877.539, 2489515.008) ( 88d23'20.69"E, 22d30'22.13"N)
```

5.2 Data Pre-processing

- At first, all the geo-transformation and the coordinate information are separated from the image and stored securely to be utilised during the data post-processing step. According to table 5.2 the Origin X & Y coordinates are used to identify the

TABLE 5.1: The Meta-Data information of the LR and HR images

Properties	**Low Resolution**	**High Resolution**
Driver	GTiff	GTiff
Dtype	uint16	uint16
No data	None	None
Width	5049	20193
Height	4681	18720
Count	3	1
CRS	$CRS.from\_epsg(32645)$	$CRS.from\_epsg(32645)$

TABLE 5.2: Geo Transformation information of the LR and HR images

Geo Transformation	**Low Resolution**	**High Resolution**
Origin X Co-Ordinate	635809.9893442297	635809.9893442297
Pixel Width	2.8	0.7
X Pixel Rotation	0	0
Origin Y Co-Ordinate	2496068.408324141	2496068.408324141
X Pixel Rotation	0	0
Pixel Height	-2.8	-0.7

top-left point in the image. Pixel width and height are the ratios of pixels to geo-coordinates in the X and Y directions. The rotation of X & Y coordinates signifies how much the satellite images are tilted from their actual geo-position.

- For the proposed model, the single channel is log-transformed in order to enhance the features and alter the colours of the images for better results.

- Before loading two input images for further processing, the images are divided into small patches using the patchify [53] library in Python to make them easier to work with. LR images are divided into 128x128 patches, whereas patches of HR images are of size 512x512. The above-mentioned geo-transformation information is necessary to identify the geo-location of each patch and their respective coordinates on the earth's surface. This information is extracted using the GDAL[54] library in Python.

- LR image contains 3 bands, which must be separated before slicing into smaller patches and re-assembled before further preprocessing. Since the HR image contains only 1 band, the slicing can be done right away.

- The proposed model can work with only 1-dimensional images for prediction purposes. Thus, all the image patches are converted to grayscale, i.e., in the range of (0, 255), as the network is trained to perform well on grey-scaled images.
- Also, all the images are converted into Uint8 from Uint16 as the networks are designed to work with Uint8 values.
- Model-based Pre-processing: However, in the training phase of the transfer learning-based SRGAN model needs images of a specific size. To achieve this, all the LR training images are divided into 32x32 patches and the HR training images are chopped up into 92x92 patches.

5.3 Train-test Setup

As previously mentioned in section 5.2, LR images are divided into 32x32 training sets whereas HR images are in 92x92 training sets before feeding to the SR models for Transfer Learning. A total of 37,560 images are produced in the training set with batch size = 3, which is further separated into train-validation sets as **Training:** 26292 and **Validation:** 7512.

The train-validation split consists of (70% + 20%) of data, while the remaining 10% is manoeuvred for testing purposes. Training of the 3 models using transfer learning is performed in the train set with batch size = 3 and buffer size = AUTOTUNE. The models are trained for 1000 Epoches which takes approx *1.5 hour* for SRGAN and *1 hour* for both EDSR and SR(PRE) each. For the bi-cubic interpolation, since it is a classic model, no further transfer learning or training has been performed over it.

All the models are tested with pre-trained weights which have been trained on the DIV2K dataset (with PSNR on the DIV2K validation set = 28.89 dB (images 801 - 900, 6 + 4-pixel border included for EDSR and for SRGAN 1.55M parameters, trained with VGG54 content loss).

5.4 Image super-resolution using AutoEn-GAN

Before being used as input into the model for the suggested AutoEn-GAN methodology, the LR and HR images are divided into 128x128 and 512x512 patches, respectively. The model

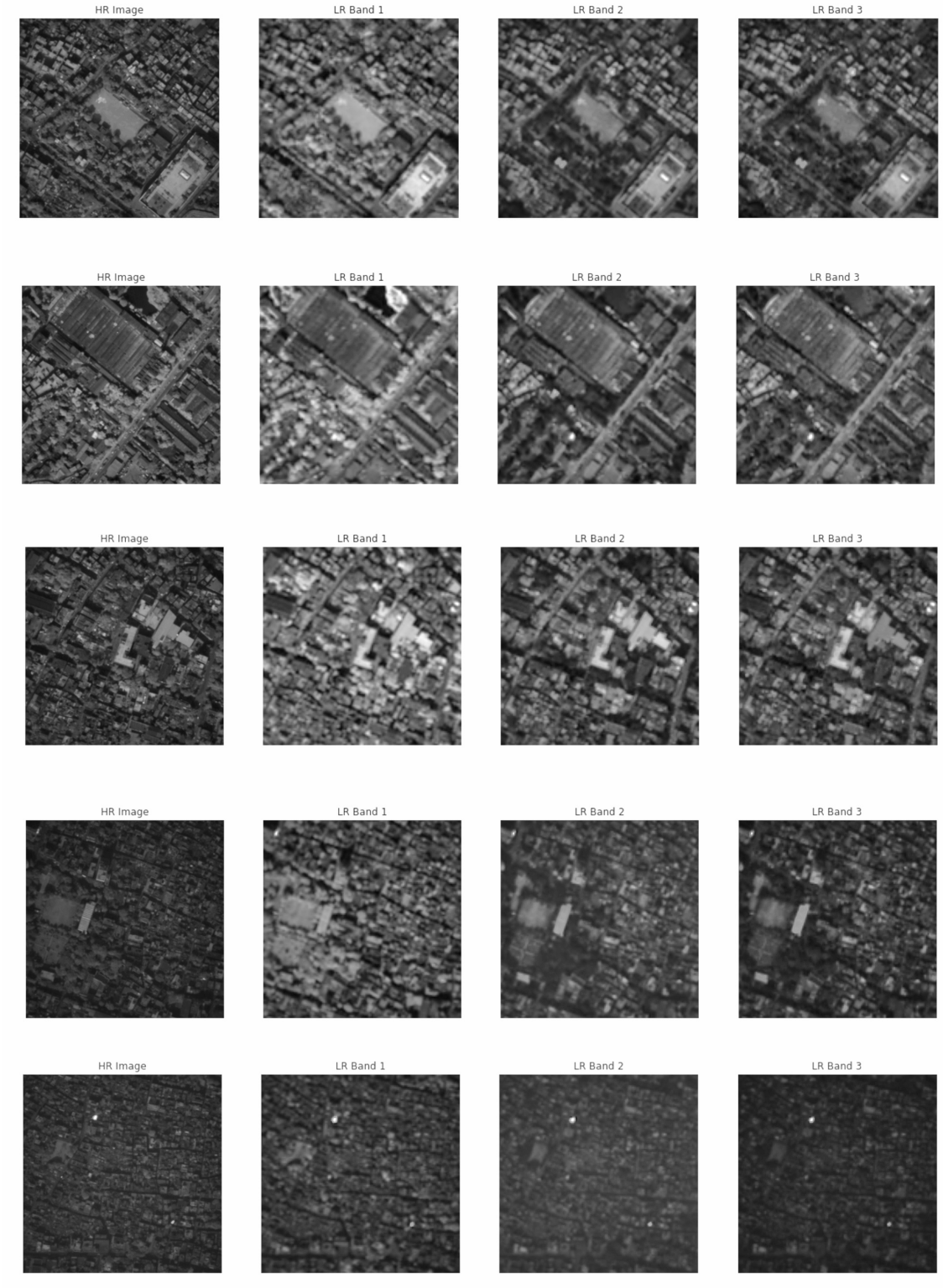

FIGURE 5.3: Sample patches with 3 LR and 1 HR band

is trained on an entire satellite-generated image that was scraped into patches, with 70% of the patches used for training and the remaining 30% used for validation. In the training loop, the LR image enters the Generator network as input and an SR image are received as output. This output image is fed into the Discriminator for prediction, which categorises it as real (1) or fake(0). Furthermore, the adversarial and content loss is calculated as portrayed in figure 5.4 and described in section 5.5. During this Generator training phase, the Discriminator remains non-trainable and the perceptual loss is minimised using the Adam optimiser. While training the Discriminator network, HR and SR images are used with appropriate labels. Here the average discriminator loss is minimised using the same optimiser.

ReLU was used as the activation function. In the proprietary system without a GPU, it took about 4 days to train the entire framework over 50 epochs. Further details of system configuration can be found in Appendix A. Another satellite image was used for testing as can be seen in Appendix B. Similar to the training image, the image is pre-processed and divided into smaller patches, before the testing phase in which it was evaluated using various parameters for quality control.

5.5 Loss Function

A loss function, also known as a cost function, is a mathematical function that calculates the difference between the predicted output and the actual output in a machine-learning model. Adversarial loss and content loss are two common loss functions used in deep learning models, particularly in the training of generative models. The adversarial loss trains the generator to generate samples that can fool the discriminator. This work defines adversarial loss based on the binary cross-entropy between the predicted probabilities of the discriminator and the actual image (real or generated) with the Adam optimiser for optimisation. The objective here is to maximise the loss while minimising the accuracy, which is precisely the opposite goal of discrimination training and this is the reason for the naming ‘adversarial loss’.

The content loss, on the other hand, measures the difference between the feature representations of a reconstructed image and a reference image. The feature representations are usually obtained from a pre-trained network; in this work, a pre-trained VGG19 network is used upto the 6th layer as illustrated in figure 5.4. The difference between the two representations is measured using the Euclidean distance or Mean Square Error(MSE).

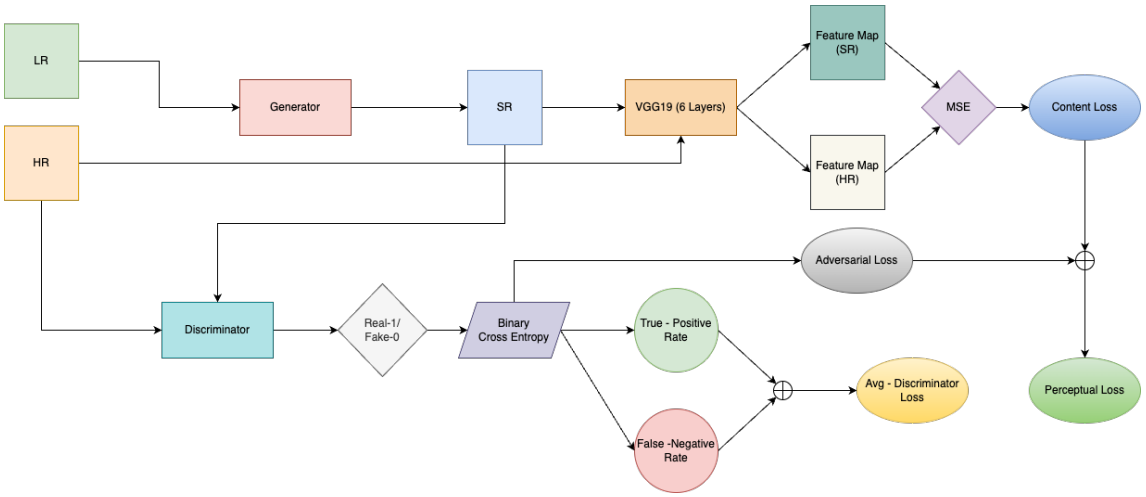

FIGURE 5.4: Generator and Discriminator Training Loop in AutoEn-GAN

The content loss helps the generator produce samples that are indistinguishable from the actual samples and retain the reference image's content information. The adversarial loss and content loss together make up the perceptual loss which is used to train the generator model.

During the training of the discriminator model, A batch of the generated image is labelled with fake (0) and the original HR image is labelled as real (1) and fed into the discriminator network. Here the binary-cross entropy is also used with the Adam optimizer to maximise the accuracy of the prediction. This is done by True-True and False-False discriminator loss which are the measurement of True-Positive and False-Negative rates respectively. The average of these losses is minimised which is known as Average Discriminator Loss as demonstrated in figure 5.4.

5.6 Evaluation

Subjective and objective approaches can be used to evaluate image quality [55]. Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) are examples of objective approaches that compare quantitative parameters with underlying data, in this instance the form of a reference picture [56]. Although PSNR and SSIM have evolved into common assessment techniques for computer vision applications, their outcomes can be rather deceptive when evaluating the quality enhancements of a picture [57]. When

evaluating a picture, the Human Visual System (HVS) takes additional factors into consideration. When viewed by the HVS, the visual quality of image pairs with nearly equivalent PSNR values might vary substantially.

5.6.1 Root Mean Squared Error (RMSE)

As the square root of MSE(eq. 5.2), the Root Mean Square Error (RMSE) measures the amount of change per pixel caused by image processing. The most used estimator of an image quality measuring parameter is MSE. It is a complete reference metric, and the closer the number is to zero, the better the performance.

$$MSE = \frac{1}{mn}\sum_{x=0}^{m}\sum_{y=0}^{n}[f(x,y) - h(x,y)]^2 \tag{5.1}$$

$f(x,y)$: HR Image, $h(x,y)$:LR Image, m,n: No.of pixels in rows and columns

$$RMSE = \sqrt{MSE} \tag{5.2}$$

5.6.2 Peak Signal-to-Noise Ratio (PSNR)

The PSNR block computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed or reconstructed image. In order to calculate the PSNR, the MSE (mean square error) of the HR reference image, $f(x,y)$, and the processed image(LR Images), $h(x,y)$, is used as in Equation 5.1. The MSE will indeed be near to 0 if the transformed picture resembles the reference image. As MSE approaches zero, the PSNR, which has the mathematical formula 5.3 will reach infinite. The similarity between h and f is strong when the PSNR value is high. A low

$$PSNR = 10 * \log_{10}\left(\frac{MAX_f^2}{MSE}\right) \tag{5.3}$$

PSNR value suggests that the pictures have numerical disparities. PSNR is measured in dB.(decibel).

5.6.3 The Structural Similarity Index (SSIM)

SSIM is a method for quantifying how similar two images are to one another. The SSIM index is a complete reference metric, meaning that the initial uncompressed or distortion-free picture serves as the baseline for the measurement or prediction of image quality. A transformed picture (LR Image), h, and an HR reference image, f, are both used by SSIM in its evaluation procedure. However, it evaluates the resemblance of the photos using the known quality perception of the HVS rather than computing the difference between the two images. As a result, SSIM is the result of three assessments established between the two pictures.

The *luminosity* is the very first parameter represented by equation 5.4.

$$l(f,h) = \frac{(2\mu_f\mu_h + c_1)}{({\mu_f}^2 + {\mu_h}^2 + c_1)} \tag{5.4}$$

The *contrast* is the second parameter represented by equation 5.5.

$$c(f,h) = \frac{(2\sigma_f\sigma_h + c_2)}{({\sigma_f}^2 + {\sigma_h}^2 + c_2)} \tag{5.5}$$

The *structure* is the third parameter represented by equation 5.6.

$$s(f,h) = \frac{(2\sigma_{fh} + c_3)}{({\sigma_f}^2{\sigma_h}^2 + c_3)} \tag{5.6}$$

Here, the positive constants c_1, c_2, and c_3 are employed to prevent division by zero. The final equation 5.7 is obtained when $c_3 = c_2/2$:

$$SSIM(f,h) = l(f,h)c(f,h)s(f,h) = \frac{(2\mu_f\mu_h + c_1)(2\sigma_{fh} + c_2)}{({\mu_f}^2 + {\mu_h}^2 + c_1)({\sigma_f}^2 + {\sigma_h}^2 + c_2)} \tag{5.7}$$

here μ_f is the mean of f, μ_h is the mean of h, ${\sigma_f}^2$ is f's variance, ${\sigma_h}^2$ is h's variance, and σ_{fh} is f and h's covariance. The Structural similarity value ranges from 1 to 1, where a value of 1 denotes complete identity between the pictures and a value of 0 denotes no structural similarity.

5.6.4 Execution Time

Execution time is the time taken by a computer program to run from start to finish. It is the amount of time that elapses between the start of a program and its completion. This time is measured in seconds, milliseconds, or microseconds, depending on the precision of the system clock. Execution time of a program depends on several factors, such as the complexity of the program, the speed of the computer, the amount of memory available, and the type of operating system being used. The execution time can be measured in different ways, including wall clock time, CPU time, and user time.

Wall clock time, also known as elapsed time, is the time that elapses on a real-world clock from the start of the program to its completion. It includes all the time spent waiting for input and output operations to complete, as well as any time spent waiting for other processes to complete. This is the most common way of measuring execution time, and it provides a good indication of the program's overall performance.

Execution time can be an important evaluation parameter for deep learning models, particularly in real-time applications where fast inference is crucial. In addition to accuracy and performance metrics, measuring the execution time can help determine the efficiency and practicality of a model. The shorter the execution time, the more practical the model is for deployment in real-world applications. However, it's important to note that execution time alone shouldn't be the sole evaluation parameter, and should be considered alongside other relevant metrics.

5.6.5 Visual inspection

After being subjected to super-resolution processing, certain features could surface that are challenging for objective assessment techniques to evaluate. When comparing before-and-after photographs, some subjective parameters such as visual inspection are required because, in certain conditions, in spite of giving better performance in terms of objective measurement, the generated image is not quite a good representation of the actual image.

5.7 Data Post-processing

After the generation of the testing image, it is found that each patch has its own colour balance. As a result, after connecting the patches, the photo had highly distinct colour

profiles with noticeable variances. In order to ensure uniformity, the colour-matching method utilised in this situation is called "Histogram Exposure Matching." [58] The goal of exposure matching is to transform the bin edges of the two images X and Y such that their resulting histograms (i.e., the distribution of each feature independent of the other features) become more similar. The greyscale input image X has a probability density function $p_r(r)$, where r represents the greyscale value, and $p_r(r)$ represents the probability of that particular value. This probability can be easily calculated from the image histogram by [59]:

$$p_r(r) = n_j/n$$

Here, the value n_j represents the frequency of the greyscale value r_j, while n denotes the total number of pixels present in the image. Suppose we have a desired output probability density function $p_z(z)$. In that case, we require a transformation of the probability density function $p_r(r)$ to convert it into the desired probability density function $p_z(z)$. Now, each probability density function (pdf) can be simply mapped to its cumulative distribution function (CDF) by:

$$S(r_k) = \sum_{j=0}^{k} p_r(r_j)$$

$$G(z_k) = \sum_{j=0}^{k} p_z(z_j)$$

where, $k = 0, 1, 2, 3, ..., L-1$ and L = total number of grey-level (a standard image has 256)

The concept involves mapping each greyscale value r in the input image X to the corresponding greyscale value z in the desired pdf, which has the same probability as that of r.

Due to the SR image's greyscale format, it only has one colour profile that can be altered by exposure. The exposure of the output patches is matched with that of the input ones using corresponding LR image patches and the SKLearn Exposure matching package. The Scikit-learn's Exposure matching package offers the ability to match datasets based on their marginal distributions, with the aim of minimising bias and enhancing generalisation

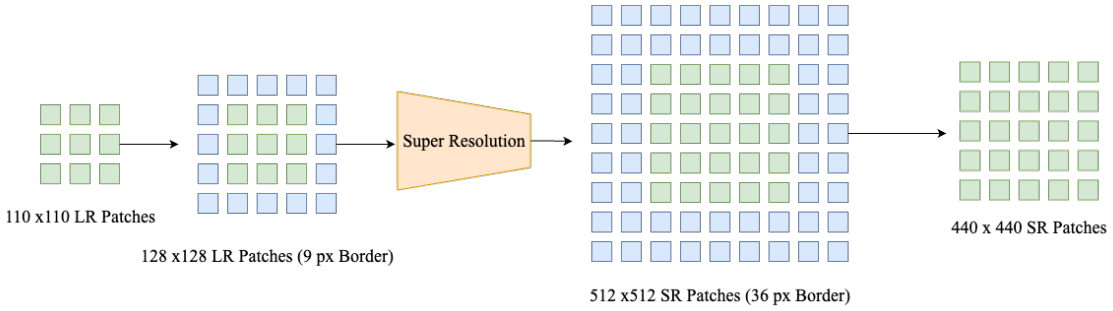

FIGURE 5.5: Post-processing for 4x Image Super-resolution

capabilities. As a result, the resulting patches are much more blended with each other than before.

The 2x super-resolution turned out to be smooth and synthesised beautifully, however, artefacts were observed at the edges of the output image patches for the 4x super-resolution of the input image. To address this issue, 110 x 110 px patches (step size 110 px) with 9 px edge padding are created instead of 128 x 128 px patches as portrayed in Fig 5.5. After SR, the outer margins of each patch had 4x9 = 36 px removed from them. All the sub-patches are then combined using Unpatchify. This removes the harsh boundary lines and smooths down the artefacts in the patch borders.

The satellite image Geo-transformation information is added after processing the images. After the small patched images are created, the final large Super Resolution image is produced by assembling and unpatchifying the small images. The Geo-referencing to the large image is attached after changing the scales accordingly, i.e., the pixel width and height information are transformed accordingly (i.e. Pixel-width / Scaling-factor). Let's consider an input image with a resolution of 128 x 128 px going through 4x Super Resolution, the output image resolution will be 512 x 512 px. Here, each pixel of the input image represents 4 pixels of the output image. However, the ground truth of the satellite image is the same as before. If, for example, the pixel size along the X and Y axis of the input image is 28.0, then the output image will have a pixel size of $28.0/4 = 7.0$ along the X and Y axis.

Chapter 6

Results

In this chapter the proposed AutoEn-GAN framework, the pre-trained models along with the ones after transfer learning with the pre-trained weights are compared against each other visually as well as through various metrics. Image patches from the original image were applied to the model for visualization before and after transfer learning. The performance was compared using image quality metrics for quantitative measurements such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and Root Mean Squared Error (RMSE).

6.1 Overview of Results

The brief overview of sample patches can be visually identified. The quality of the outputs after transfer learning gets significantly better as seen in figures [fig:6.1] [fig: 6.2]. The proposed model performs significantly better than other methodologies whereas SR(PRE) performs the worst among them. The output produced by the benchmark model after bicubic interpolation is more blurry. On the other hand, the output from the deep learning models consists of artefacts. It could be due to the different outcomes produced by each of the 3 layers, which couldn't be adequately assembled.

To solve this, the proposed model takes greyscale images as input and produces only a single-layer output AutoEn-GAN model is trained on satellite images only, whereas other models are trained on public datasets like DIV2K. The gaussian nature of Auto Encoder for artefacts removal with GAN architecture produces visually sharper images compared to other traditional frameworks. The new architecture also includes many dropout layers

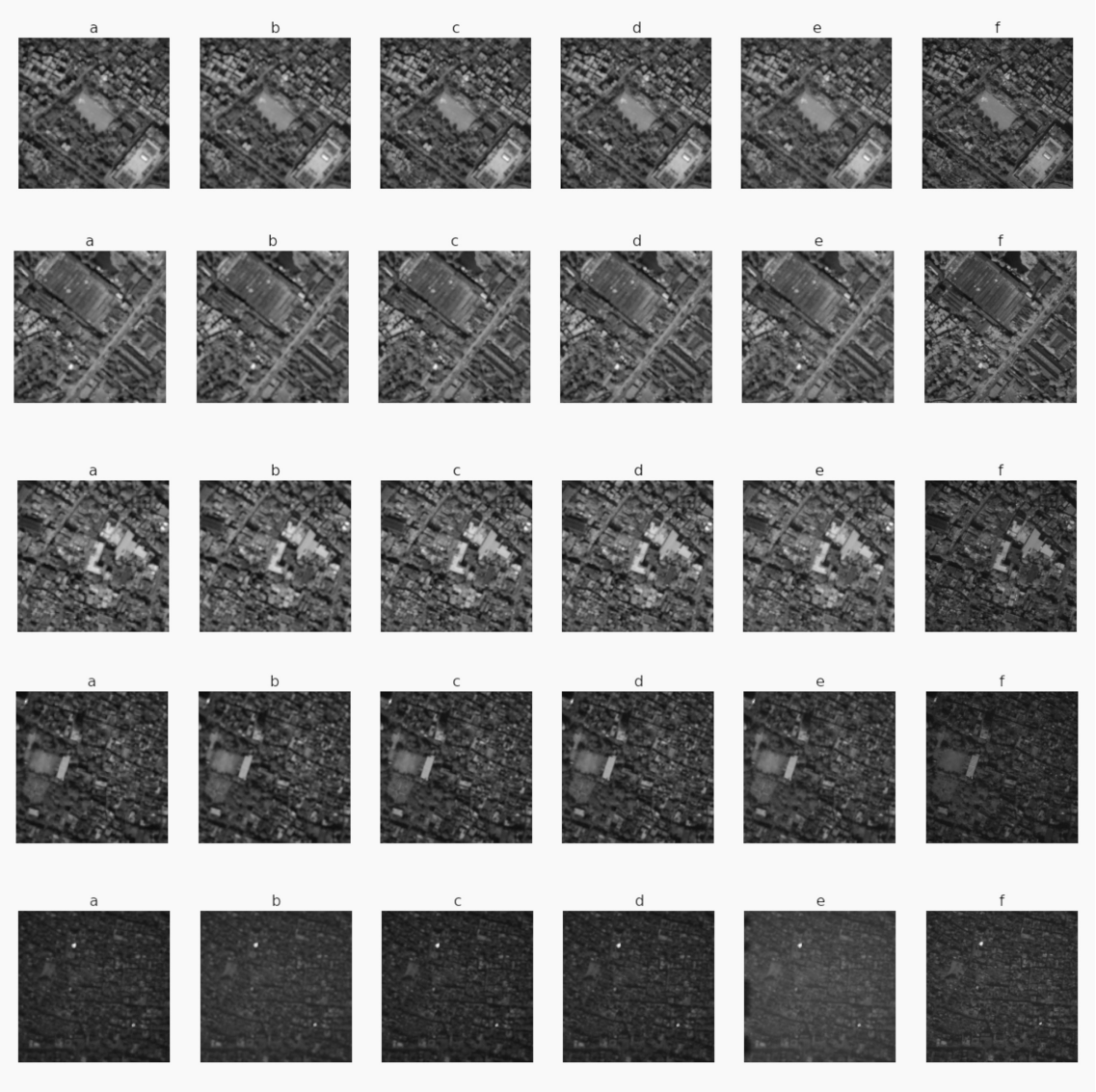

FIGURE 6.1: Sample patches of all images before transfer learning (a) Input Image, (b) Bicubic Interpolation, (c) EDSR, (d) SR(PRE), (e) SR(GAN), (f) Original Image

for the prevention of overfitting which is a major problem while dealing with small-size domain-specific datasets.

The problem of sharp image generation while using Autoencoder-based methodologies is usually solved by increasing the depth of VAE and creating a Very Deep Variational Autoencoder. However, in this proposed methodology we have trained Autoencoder style Neural Networks in a Generative Adversarial framework. Uses of residuals like blocks, skip connections and dropout regularisation are key features for achieving structurally and visually crisp images. Here, the residual blocks create bottleneck layers and skip connections between them maintains the structural integrity of images with the LR sample.

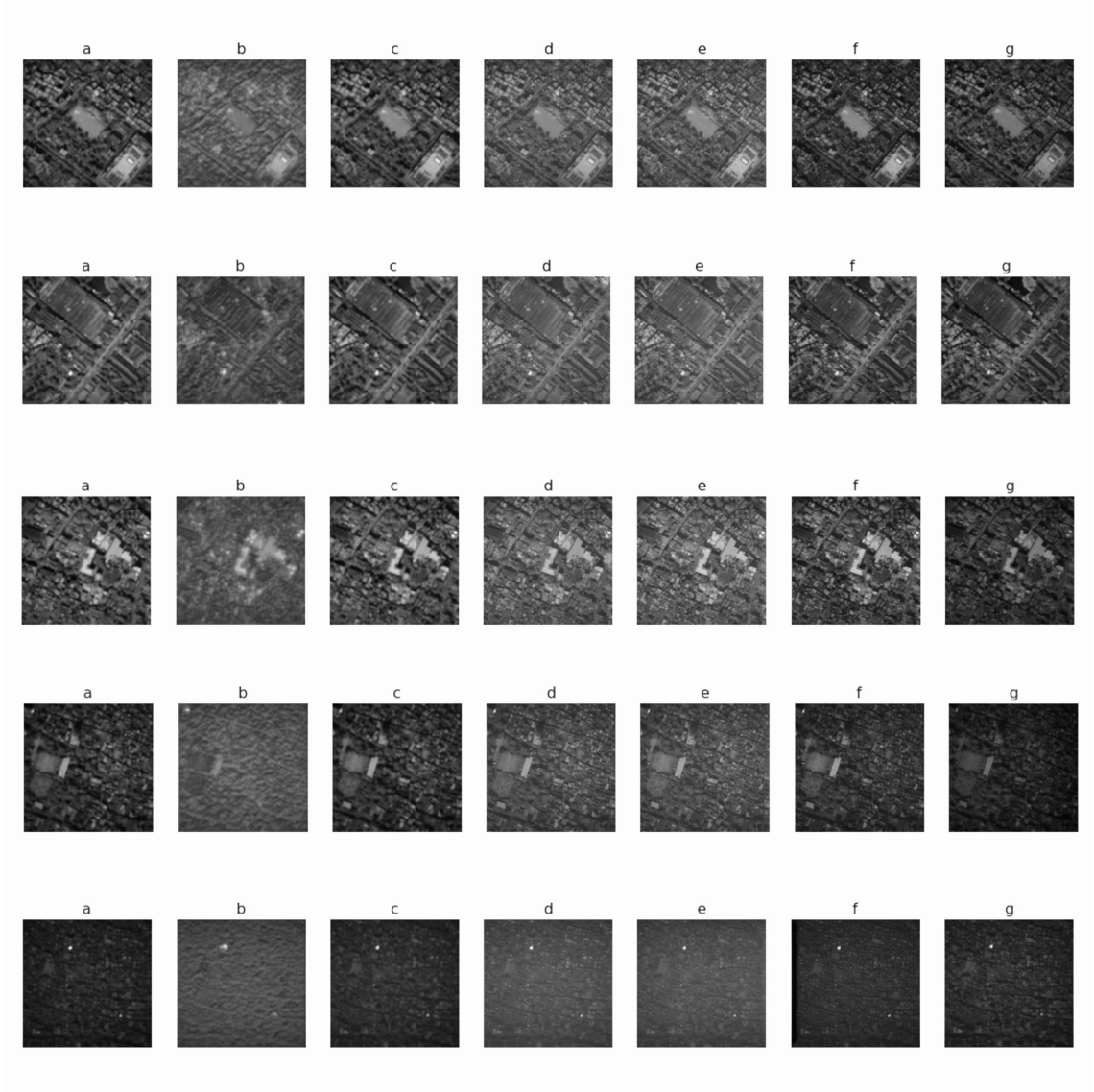

FIGURE 6.2: Sample patches of all images after transfer learning and training of the model(a) Input Image, (b) Bicubic Interpolation, (c) EDSR, (d) SR(PRE), (e) SR(GAN), (f) Proposed Model, (g) Original Image

6.2 Quantitative results

For the quantitative estimation purposes, the results of the CNN and GAN networks along with the novel AutoEn-GAN framework, PSNR, SSIM, and RMSE matrices are used for comparison. All of them are already discussed theoretically in the section 5.6. The results of the sample of five images were taken into account. Among them, the first three images are shown in the figures 6.1 and 6.2. For visualisation purposes, each of the matrices is

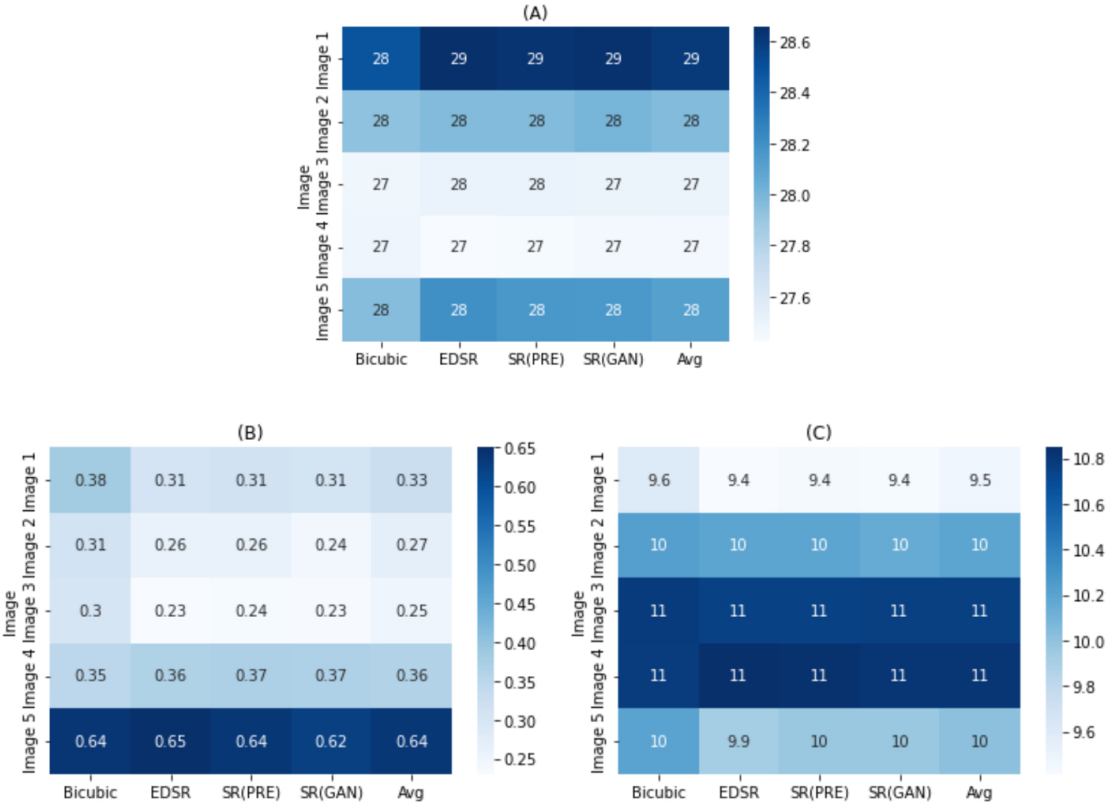

FIGURE 6.3: Comparison of the matrices before transfer learning (A) PSNR Values, (B) SSIM Values, (C) RMSE Values

compared and plotted using a heatmap. Both the cases, i.e., before and after transfer learning, are studied this way in Figures 6.3 and 6.4.

It is evident from figure 6.3 that the PSNR values for image 1 are higher than their counterparts. However, Image 4 has the lowest PSNR out of all of them. The outputs of Image 3 and the HR image, on the other hand, exhibit very little structural resemblance according to the SSIM findings, whereas Image 5 and its SR image exhibit significant structural similarity. The RMSE value, on the other hand, proposes that Image 3 and 4 has a significantly higher error, which is in accordance with the results produced by the PSNR metric. However, considering each model individually, we can understand that all of the models perform almost similarly to each other. This is because none of the models is trained on our specific dataset. This has significantly improved after transfer learning.

The quantitative outcomes of SR images after transfer learning, however, draw an improved comparison among the different frameworks. It is evident from the figures 6.4 that the PSNR and SSIM values of the bi-cubic interpolation are significantly lower with

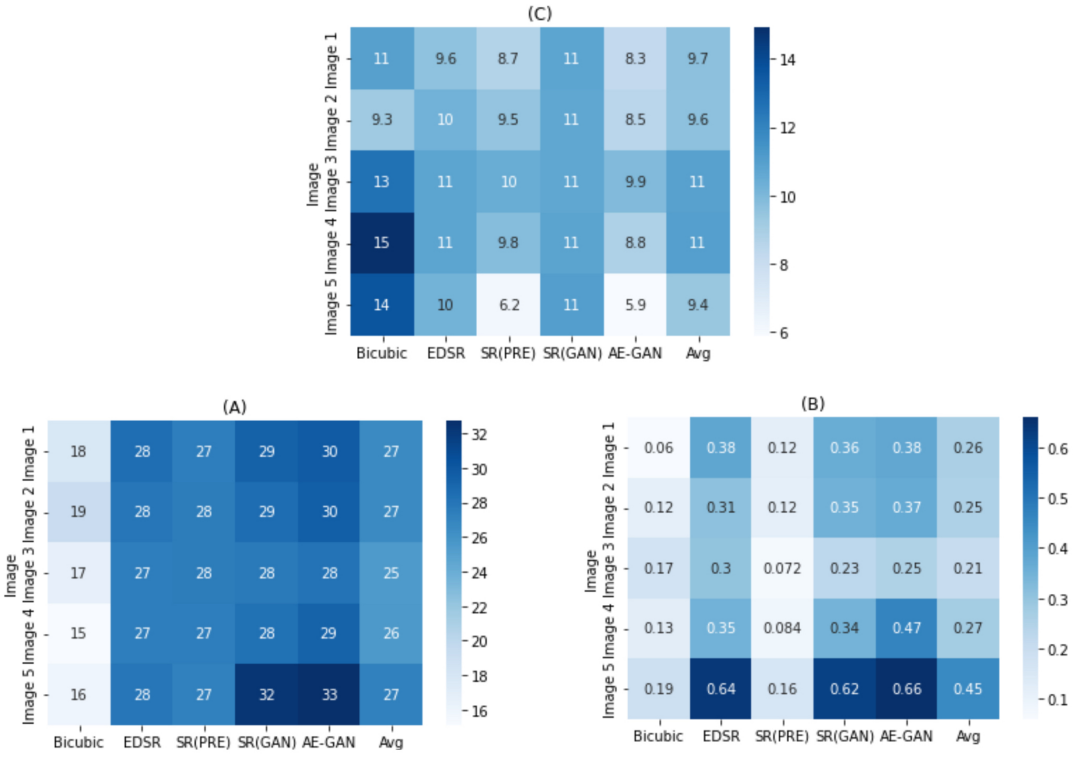

FIGURE 6.4: Comparison of the matrices after transfer learning (A) PSNR Values, (B) SSIM Values, (C) RMSE Values

higher RMSE values, suggesting the presence of noise, thus distorting the output images. Contrary to this, the AutoEn-GAN structure gives the best outcome among other models and appears to have a higher degree of structural resemblance to the original HR image as illustrated in Figure 6.5.

The training loss mentioned in section 3.1.4.1 is also taken into consideration when comparing the results. The training loss considered in the paper acts as the perceptual loss in the SR-GAN model, Mean Squared Error in the SR(PRE) and Mean Absolute Error for EDSR 6.1. The results are taken before and after 1000 epochs of training of each of the 3 Deep Learning models. During transfer learning of these models, only their last layers were trained with the new weights and biases. Figure 6.6 clearly shows the decrement of the training loss after training. As training loss is created to depict human visual responses, it matches our visual intuition. A significant boost in the performance of the models after transfer learning was anticipated, according to the human perception of images and training loss.

The graph in figure 6.7 shows a slight increase in the efficient performance of models after transfer learning. However, the dramatic improvement as shown by the pictures 6.1 and

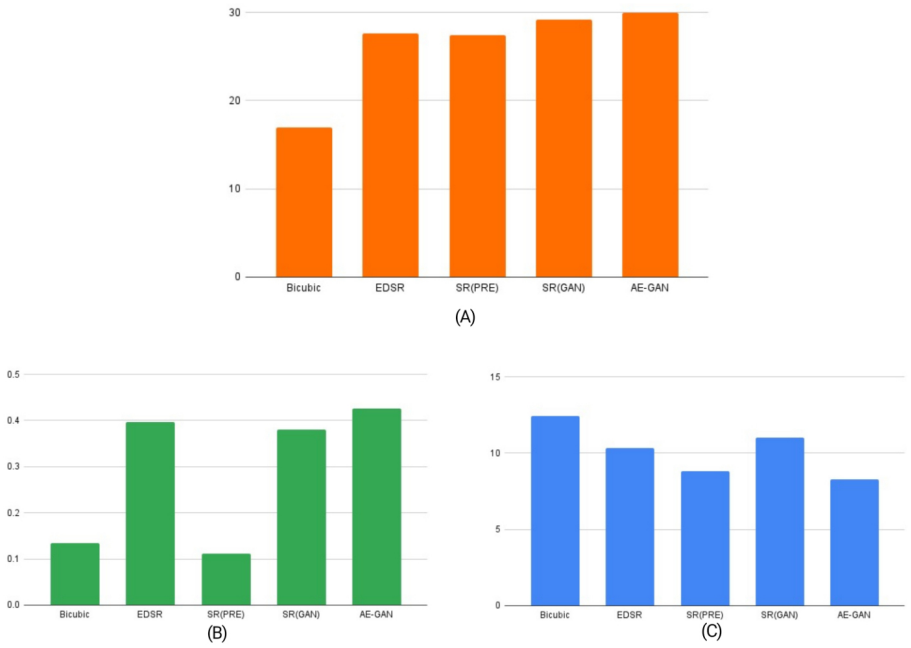

FIGURE 6.5: Column chart depicting mean (A) PSNR (B) SSIM and (C) RMSE of all the 5 images across all methods

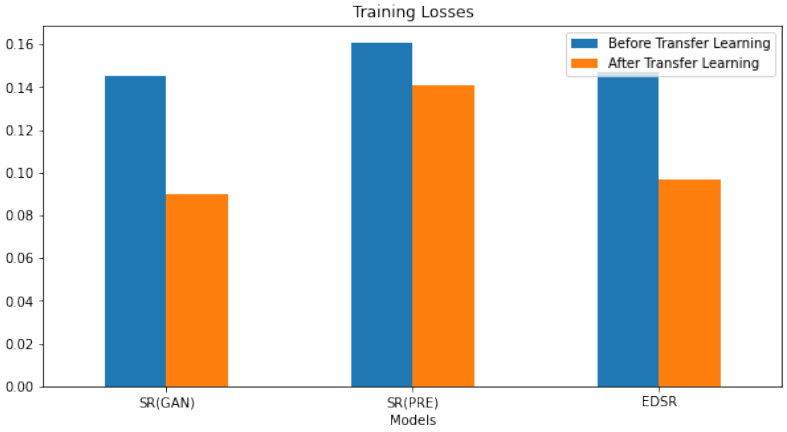

FIGURE 6.6: Comparison of the training loss before and after transfer learning

6.2 is not reflected in the figure 6.7. The new PSNR and new SSIM(after transfer learning) showcase a slight boost in performance. However, the old RMSE(before transfer learning) and the new RMSE(after transfer learning) exhibit unexpected results. In 3 instances, the new RMSE shows better results than the old RMSE. In the other 2 instances, the old RMSE shows similar or slightly better results than the new RMSE.

The poor performance with respect to the evaluation matrices can be because of the artefacts which are generated during SR processing after transfer learning of the discussed models as the artefacts can destroy the image's structural and textural consistency.

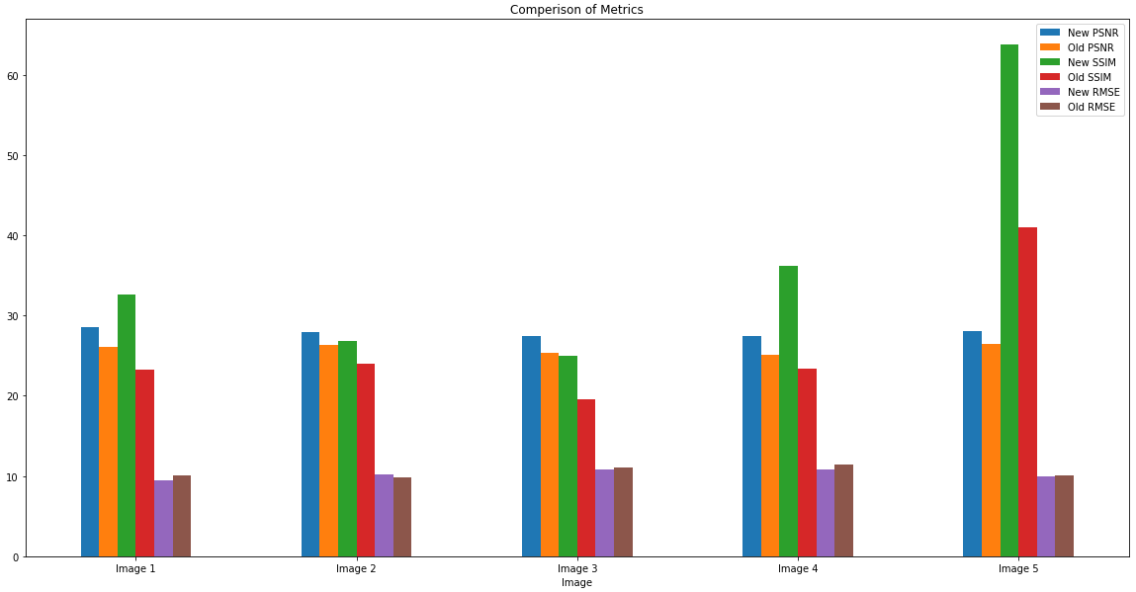

FIGURE 6.7: Comparison of all the matrices before and after transfer learning (Here SSIM is scaled 100x for visual comparison and Average signifies the average matrices of all the images in different models)

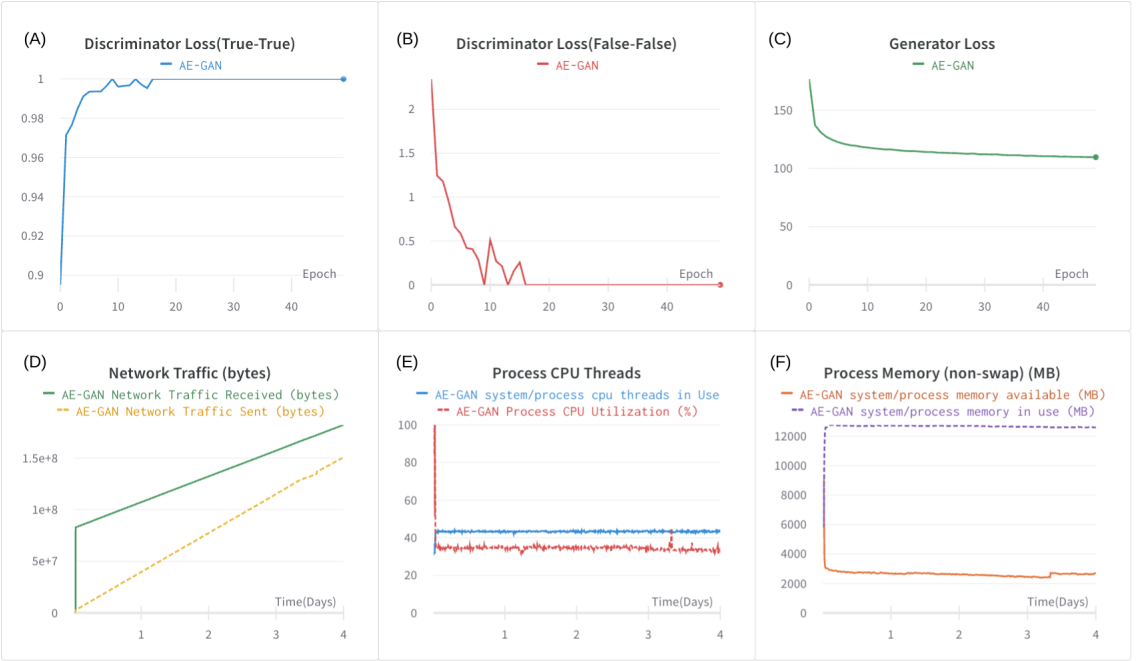

FIGURE 6.8: All the training conditions of AutoEn-GAN (Proposed Model). Generator and Discriminator Loss are shown in graphs (A),(B) & (C). Network, CPU threads and Memory used by the system while training is shown in graphs (D),(E) & (F).

TABLE 6.1: Comparison of Proposed Model with Existing Solutions

	EDSR	SR-PRE	SRGAN	***AEGAN***
No. of trainable parameters	43 million (4X)	2 million (4X)	140 million (4X)	544 million (4X)
Optimisation function	Stochastic Gradient descend	Adam	Adam	Adam
Loss function	Mean Absolute Error	Mean Squared Error	Perceptual loss	Adversarial loss and content loss Images
Speed of convergence / training time	1.5 Hours (Transfer Learning on GPU)	0.5 Hours (Transfer Learning on GPU)	3 Hours (Transfer Learning on GPU)	4 days (training from scratch without GPU)
Suitable for which kind of image	32 x 32px coloured image with a 4-pixel border	64 x 64 px coloured image	64 x 64 px coloured image	128 x 128px and accurate greyscale image
Advantages	Comparatively lightweight model while giving accurate enough output	Very light weight while producing acceptable results	Produces better results compared to SR-PRE(SR-Resnet)	Light weight compared to SR-GAN while producing better results with 128x128 patches
Disadvantages	Due to Not accurate overlapping of all 3 RGB channels it generates artefacts	Can not generate very sharp and accurate Images	Due to the discriminator it has a very high number of parameters to train	Takes Huge time to train accurate outputs for 4x compared to 2x

For the proposed AutoEn-GAN framework, the detailed report is portrayed in figure 6.8. The first 2 charts (a and b) show the Discriminator loss in form of True-True and False-False. Both these losses remained fairly inconsistent for the first 16 epochs and eventually got stabilised at 1 and 0 respectively. This shows that after about 20 epochs, the discriminator was unable to distinguish between the original HR images and the generated SR images. This indicates the improvement in the Generator network in producing original-like outputs. The third graph depicts the Generator loss, which gets stabilised after a few epochs portraying that the differences between the RMSE values of the original and generated image have declined and finally got stabilised in just a few epochs. This generator loss is actually a combination of adversarial and content loss i.e. perceptual loss which unlike other generator networks has very unique scale. As a result, the generator loss converges at around 100 which is not very common among other GAN models.

The graphs d,e, and f illustrate the Network Traffic (in bytes), Process memory, and Process CPU Threads used while training (using training data-set5.1) and executing the AutoEn-GAN architecture. The number of process CPU threads was quite high at the beginning of the execution. However, it got constant over some time and remains like it throughout the execution. The system also started with 6000MB of memory utilisation which then rose to about 12000 MB, before gaining stability. This shows that during the training phase in the proprietary system (reference:A) after initialisation, the program utilises the CPU and Memory in almost a constant manner throughout the period of training. No sudden changes are encountered during the training of the proposed framework in resource utilisation. A gradual increase is observed in the network traffics which is essential to communicate the log information with Weight and Biases (online MLOps Platform). The table 6.1 further compares the state-of-the-art model with the existing traditional solutions.

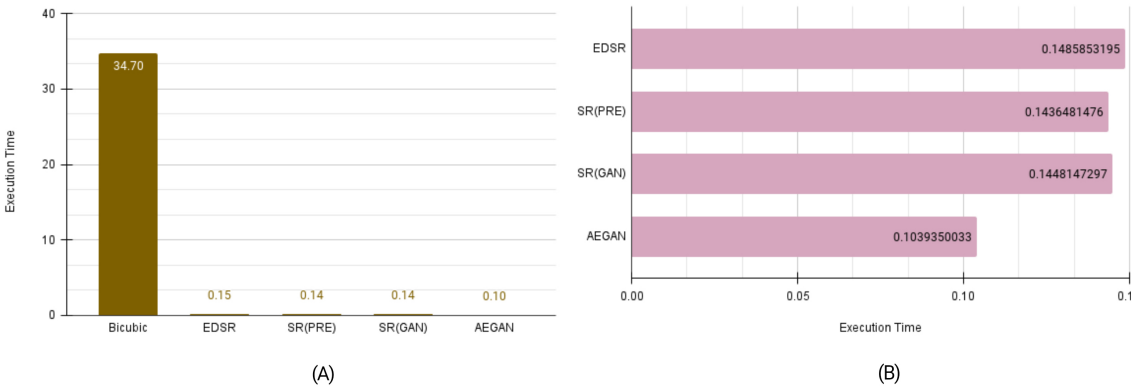

FIGURE 6.9: Execution time (in Seconds) of (A) All 5 frameworks used for comparison in the research work, (B) All 5 models, except bicubic for precise portrayal

6.3 Execution Time

In terms of execution time, the Bi-cubic framework had the longest duration compared to all the other models, whereas AutoEn-GAN had the shortest execution time, as illustrated in Figure 6.9(A). The reason for this is that bicubic interpolation involves classical super-resolution methods that require a large number of matrix operations, unlike deep learning models. Deep learning models, on the other hand, take longer during training but have a much shorter execution time after the training is completed and the weights and biases are adjusted. Specifically, the bi-cubic model took about 35 seconds to complete execution, while all other models took significantly less time, as shown in Figure 6.9(B). The EDSR, SR(PRE), and SRGAN models took approximately 0.148, 0.143, and 0.144 seconds, respectively. The proposed AutoEn-GAN model was the fastest in terms of execution time, taking only around 0.1 seconds.

Chapter 7

AutoEn-GAN Application

This section defines the interface of the proposed model and its app's implementation details. The model employs AutoEn-GAN to automatically enhance low-resolution satellite images From the beginning of the project, the main objective was to create a user-friendly application that non-technical users can use this program using a secure interface. As a desktop is the most reliable and secure option for this project, a fully platform-independent desktop application is developed for this project work. Due to the cross-platform nature and the fact that most code included in this project is in Python3, the authors chose to build the desktop interface in Python3 itself for the purpose.

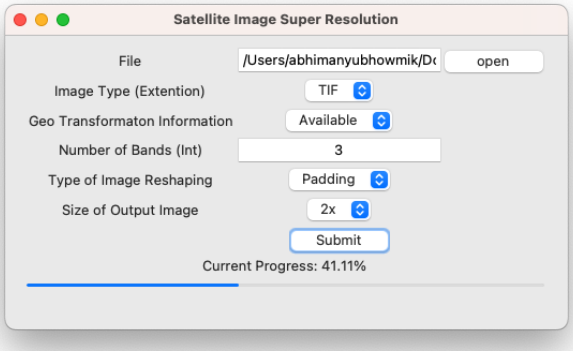

FIGURE 7.1: Desktop application interface while Processing.

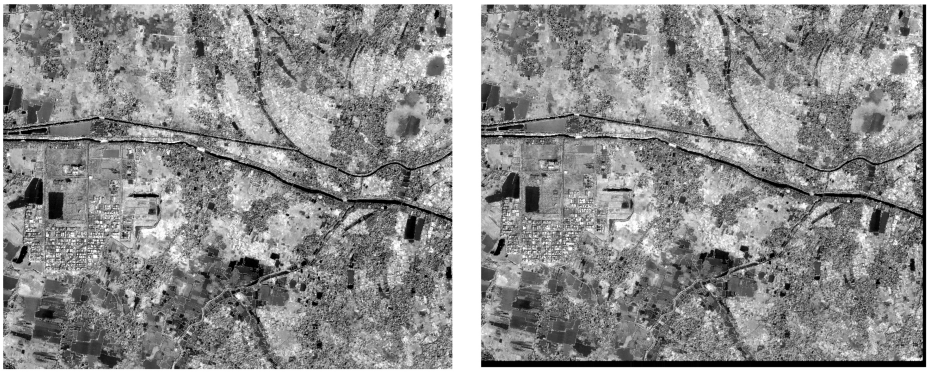

FIGURE 7.2: Comparison of Cropped(Left) vs Padded(Right) Images.

7.1 Implementation Details

The entire system has a desktop architecture which can be used only with physical systems. The user end has a simple easy-to-read interface to access the AutoEn-GAN framework. When a user opens the application, an interface as in figure 7.1 will appear on the screen. Here, one can choose from a variety of options, depending on their input. To begin with, the user chooses the input image location in the local system. Then comes the type and geo-referencing details of it. This image is expected to be satellite imagery and can be of various types, i.e., TIF, JPEG, JPG, PNG, GIF, etc. The proposed model is capable of processing all these different types of input data. In case when the image is geo-referenced (true for most satellite images), the geo-reference information will be processed separately by the model and will be added to the output image. The subsequent data is the number of bands, which can also be variable since most satellite images are multiband in nature. All of these bands will be merged and transformed to greyscale with proper normalisation before applying the AutoEn-GAN model to the images. The next option is for the type of image reshaping. Since the patches are divided into some predefined sizes (128x128 or 110x110), the input image need to integer multiple of the patch size. In most scenarios, it will not be the case. To encounter this issue the app will either crop the image or add padding to it. In figure 7.2 comparison between padded and cropped output is displayed. The final option is the desired size of the output image, which may be either 2x or 4x.

As the user submits all these pieces of information, the application will start preprocessing, followed by super-resolution and geo-referencing of the input image to deliver the output. The output image will be saved at the same location as the input image with the name “2xoutput” or “4xoutput” at the end of the image name, depending on its size, as depicted in figure 7.3. The data type of the output remains the same as the input image throughout

FIGURE 7.3: (A) Input Image with 2x and 4x SR Image, (B) Success Message, (C) Application icon in the doc.

the process. After the whole process is over, the user will receive a pop-up, letting them know that the super-resolution of the image has been successful (fig 7.3). If any issue occurs during the process or any misinformation about the dataset given by the user is detected during execution, the program will automatically generate relevant warnings or error messages for the user.

To accomplish all of these features the app is developed in synchronous multi-threaded architecture. All the backend processes such as image preprocessing, super-resolution and geo-transformation are performed in separate threads while communication between the threads is established via Python Queue data structure. The Independence of UI thread (Main thread) is crucial for smooth application performance. Various type-checking and exception-handling techniques have been implemented throughout the application. Object-oriented modular programming is used to build robust applications while maintaining readable code.

7.2 Technology Stack

- **Tkinter**: Tkinter is a standard Python library used for creating graphical user interfaces (GUIs) for desktop applications. Tkinter provides various widgets and tools for designing interactive and user-friendly interfaces. These widgets include buttons, text boxes, labels, menus, and more.

- **OpenCV & PIL**: PIL (Python Imaging Library) and OpenCV (Open Source Computer Vision Library) are two popular libraries used for image processing in Python. PIL is a library that allows performing various image operations such as image resizing, cropping, conversion, filtering, and more. One of the strengths of PIL is its simplicity and ease of use. OpenCV, on the flip side, is a more complex library designed specifically for high-speed computer vision applications. It is a powerful library that provides various tools for image and video processing, including object detection, facial recognition, and motion tracking.

- **RasterIO & GDAL**: RasterIO and GDAL (Geospatial Data Abstraction Library) are Python libraries used for reading and writing geospatial raster data in various formats, including GeoTIFF, netCDF, HDF5, and more. GDAL supports over 150 geospatial data formats, including vector, raster, and database formats. RasterIO also provides a flexible and extensible architecture, making it easy to extend and customize for specific use cases. Additionally, GDAL is highly optimized for performance, making it suitable for processing large datasets.

- **Patchify-Unpatchify**: Patchify is a function that takes an input image and breaks it down into smaller patches or tiles. Unpatchify, on the other hand, takes the patches produced by Patchify and reconstructs the original image. The strength and importance of Patchify and Unpatchify lie in their ability to break down larger images into smaller, more manageable pieces, allowing for more efficient processing.

- **Tensorflow & Keras**: TensorFlow is a popular open-source platform for building and training machine learning models. It is designed to handle large-scale neural networks with large datasets and complex models. Keras, on the other hand, is a high-level neural network API that runs on top of TensorFlow, providing a user-friendly interface for training deep learning models.

- **Scikit-Image**: Scikit-Image is an image processing library in Python that is built on top of the scientific computing package NumPy. It provides a collection of algorithms for image processing and computer vision tasks, such as image restoration, geometric transformations, colour manipulation, and exposure matching.

- **Numpy**: NumPy (Numerical Python) is a powerful Python library used for numerical computing. It is one of the most widely used libraries in scientific computing, data analysis, and machine learning applications. NumPy provides fast and efficient multidimensional array operations and also offers a wide range of mathematical functions to work with these arrays.

- **Threading**: The threading library in Python is used for creating and managing lightweight threads or concurrent execution units that can run in parallel with the main program. It is a Python tool for parallel processing, resource utilization, responsiveness, and simplification of programming.

The application consists of several classes and modules. Many of the functions of the applications are meant to run parallelly or consecutively. There are 4 main classes in the entire app.

aeganApp: This is the main class which is responsible for UI elements and processing. The constructor of this class is responsible for all the Tkinter UI elements. 'processFile' function is triggered while pressing the submit button. This method is responsible for all the processing of the app. When it is triggered it will check the inputs and depending upon the input it creates 4-5 threads for image preprocessing, reshaping, super-resolution, georeference and UI progress update.

preProcess: This class consists of 3 methods i.e. 'img_preprocessing', 'img_cropping' and 'img_padding'. The first method is for merging all the image bands into a single band and reshaping the image matrix which is suitable for further processing. The next 2 functions are executed depending on the user's choice. If the user selects the method as cropping, the 2nd function crops the image into the nearest size divisible by patch size. While the third function pads the image, making it divisible by patch size.

supRes: This is the main super-resolution class which initialises the patch size and the model, depending upon the user's choice. There are 3 methods inside this class which are responsible for (a) Log Transformation of preprocessed image; (b) Super-resolution of Log Transformed image in 2x and 4x resolution depending upon the user's selection.

geoInf: This class is located in the geoTransform module and executes 3 tasks. First, the constructor extracts the geo-information from the given image. Then the 'geo_scaling' function optimises the geo-information according to the output image. Lastly, the 'geo_out' method attaches the geo-information with the image and saves the image in the appropriate location.

Progress: This class is meant for updating the current situation in the main UI thread. It initialises a progress bar, stopping and updating the text and the value of the progress bar is handled by independent functions. It can obtain the state information of the application using the Python Queue data structure from all of the different threads.

Chapter 8

Final Remarks

Satellite image super-resolution is a growing field of study which needs to be addressed with respect to their specific requirements and desired outcomes in the domain of satellite imagery and signal processing. In this chapter, the outcomes of the results are analysed with reasoning along with a deep discussion on the factors that can be improved upon (in section 8.1). The overall work and future aspects of this work are also mentioned in sections 8.2 and 8.3.

8.1 Discussion

- The proposed AutoEn-GAN network is inspired by the existing SRGAN architecture. However, it is a novel contribution of its kind since it blends in Autoencoders with the SR-GAN adversarial training framework, thus producing both artefact-free and sharp SR images. Although inspired by SRGAN, the proposed AutoEn-GAN has a much lighter framework compared to SRGAN. For extracting better features from the smaller filter kernels 8.3, the AE GAN uses dropout regularisation which can be clearly witnessed in the feature maps of the images 8.6.

- Among the pre-existing Super-resolution models, SRGAN keeps performing the best. One of the major reasons is that SRGAN has larger filter kernels compared to EDSR (as portrayed in the figures 8.1 and 8.2), thus it can extract much more information prominently when compared to EDSR as illustrated in the feature maps of the respective models. Some improvements in the EDSR model are expected if its filter kernel's size is increased. Nevertheless, transfer learning won't be possible since any

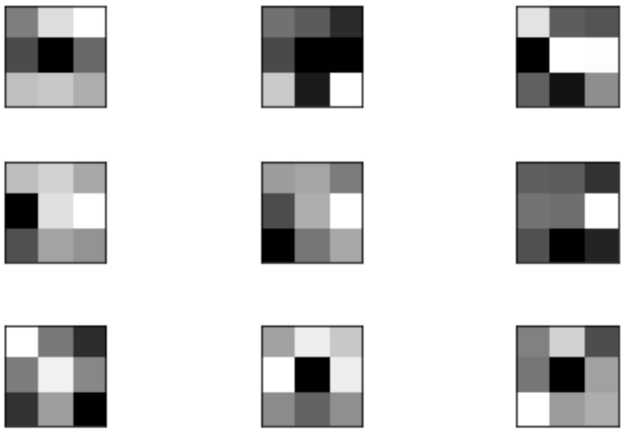

FIGURE 8.1: 3rd Layer EDSR Filter Kernels.

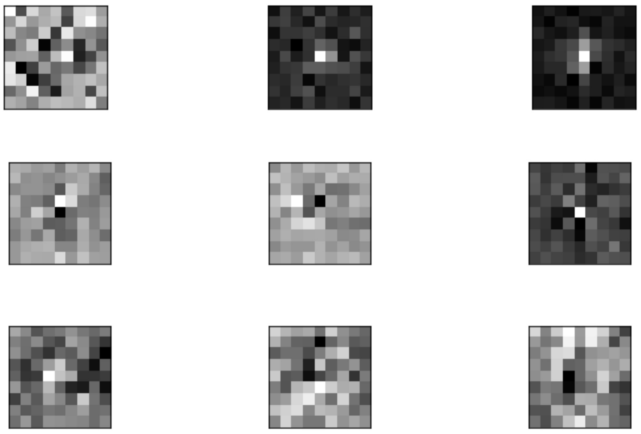

FIGURE 8.2: 2nd Layer SRGAN Filter Kernels.

changes to the architecture will completely destroy the orientation and significance of weights and biases.

- Furthermore, SRGAN and SR(PRE) have the same CNN architecture. However, SRGAN uses the adversarial setting for training with images whereas SR(PRE) has a simple ResNet framework. The weights and biases from SR(PRE) are taken as inputs by SRGAN and are used for finetuning the generator network thus, providing better results.

- Most of the feature maps created by EDSR are dark, meaning that only a small amount of feature information can be gleaned from them. The results might improve if somehow one can enhance the feature maps. However, feature maps are

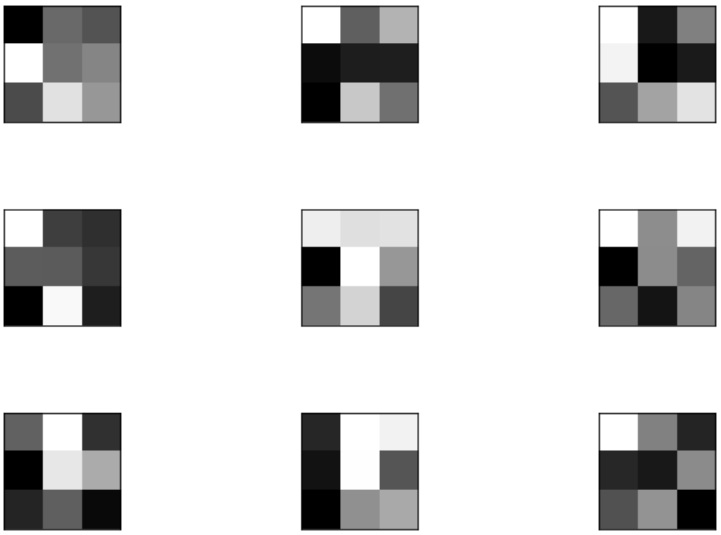

FIGURE 8.3: 2nd Layer AutoEn-GAN Filter Kernels.

the representation of the learnt weight and biases or the filter kernels, so we can't enhance the feature maps without interfering with the filter kernels of the particular layer.

Additionally, there is not much difference in the extracted features due to the similarity of the several resultant feature maps, as seen in figure 8.4. It can be the result of the model overfitting the available data.

- To improve the model's efficiency and overall complexity, Dropout layers are introduced to the cutting-edge AutoEn-GAN model. Unlike the EDSR network, the feature matrix produced by AutoEn-GAN has many dark-pitch black outcomes [fig 8.6]. This is because of the dropout of the repetitive and excessive layers. This makes sure that the extracted feature maps contain loads of information packed into them, thus making it computationally less expensive while maintaining a high image quality.

- The feature maps of the SRGAN model extract various features. As seen in figure 8.5, the maps show different features with variations in contrast, sharpness, colour, brightness, noise, etc. However, some features are duplicated multiple times among most feature maps, thus making them redundant and unnecessarily resource-extensive.

- One may also examine the artefacts that have been formed in the picture by zooming into the network-generated image as shown in Figure 8.7. It could be because the

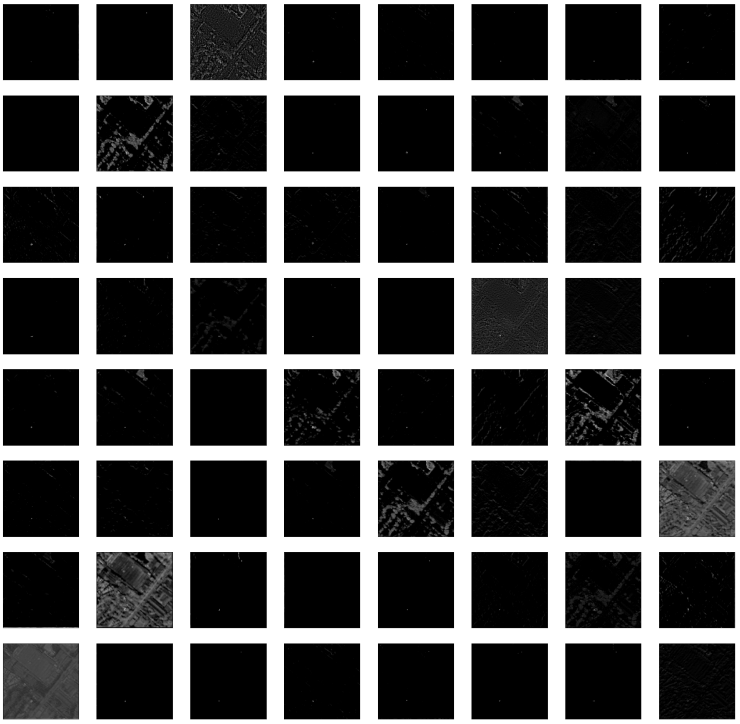

FIGURE 8.4: 3rd Layer Feature Map of EDSR.

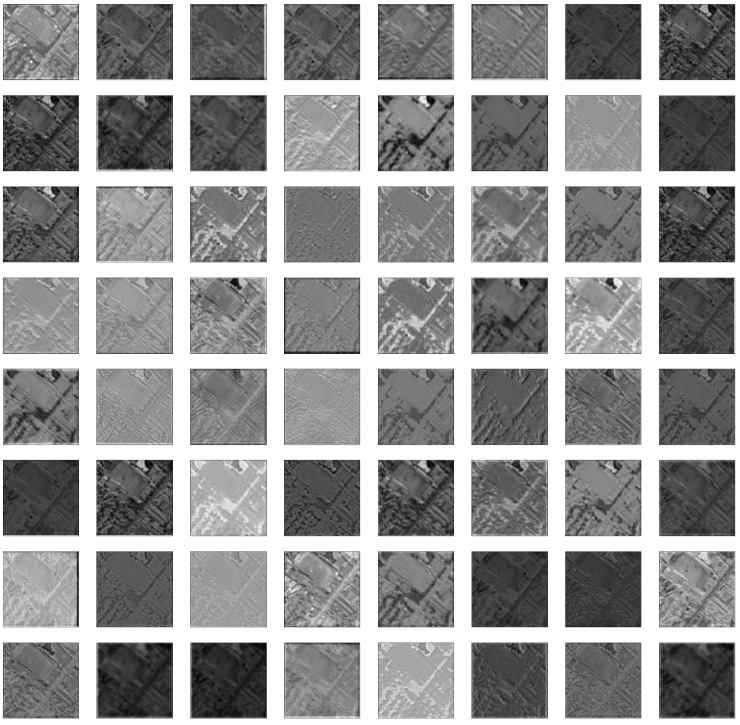

FIGURE 8.5: 2nd Layer Feature Map of SRGAN.

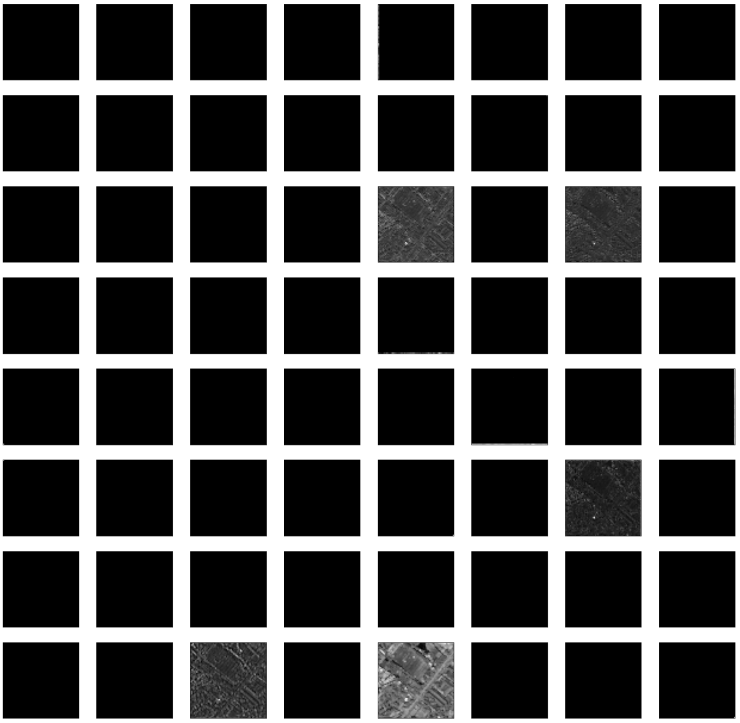

FIGURE 8.6: 2nd Layer Feature Map of Proposed AEGAN

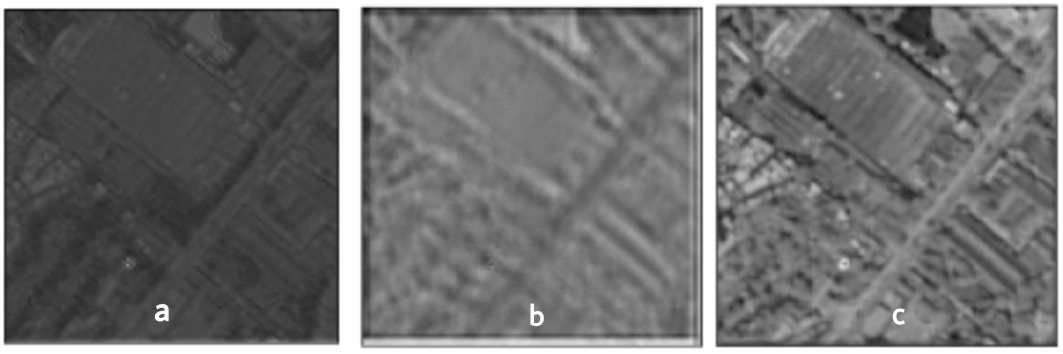

FIGURE 8.7: Network-generated images of (a) EDSR, (b) SRGAN, (c) AutoEn-GAN

features that the three layers extracted have different properties, and when they overlap, they produce RGB artefacts.

- To resolve this cause, the AutoEn-GAN network merges all the bands of the image, thus taking a greyscale input for further processing. This reduces artefact generation in the network-generated images when compared to SRGAN and EDSR framework as depicted in figure 8.7.

8.2 Conclusion

Image Super-Resolution in a very specialized context (e.g., satellite imagery) is a quite new problem that has not been the focal point of deep learning research before. The issues with the super-resolution of satellite images face many challenges, such as dealing with multi bands, processing very large data, joining small patches of SR efficiently, and dealing with artefacts and georeferencing, among others. Besides, the traditional SR frameworks are generally huge and not well-trained in the context of satellite images which is a big obstacle in the remote sensing industry. This work suggests a novel lightweight GAN-based framework to overcome many of these issues.

AutoEn-GAN is a promising model for the super-resolution of Satellite images and has the potential to make deep learning accessible for super-resolution to non-technical individuals and industry professionals. An interactive desktop has been developed using Python and Tkinter, which is both user-friendly and lightweight. The incorporation of autoencoder architecture in the GAN framework is one of the novel contributions of this work. Our findings suggest that GAN-based models can produce even better outcomes in the field of superresolution for satellite image processing in the future.

8.3 Future Works

Although the current work is more or less satisfactory, further work can be conducted in the future to explore the prospect of producing better results.

1. The AutoEn-GAN model is not perfect; it has its flaws as well. One of them being the generation of artefacts at the borders of the images. In future, it can be addressed by alternating or adding up layers to the framework.

2. Also, the pre and post-processing methods can be improved for better and higher-quality super-resolution image generation. Some systems can be developed in the future that can accommodate the complete dynamic range as well as the varying number of bands in satellite images.

3. To speed up image processing, a secure web interface can be created. Web servers with Graphics and storage capacity are needed for this. The interfaces for application interfaces can be built using a web-based Python framework like Streamlit, Flask, or Dash, and Docker or Kebernets can be used for cluster deployment.

4. A distributed system with online training can be implemented for more robust use cases. Major security measures could be implemented using federated learning or blockchain-enabled web-based technologies.

5. Moreover, some AI-integrated fault-tolerant server-based distributed systems may be developed for very large (Big Data) processing on the server without any human intervention.

6. Based on GAN, newly created VAE, or transformer architectures, several new models can be assembled that should be particularly trained on satellite imaging data and capable of extracting considerably more features within a huge dynamic range. Additionally, quicker training and testing times should be anticipated.

8.4 Conflict of interest

The authors affirm that they do not have any conflicts of interest. To the best of our knowledge, the proposed approach is the first one to introduce an autoencoder in an adversarial setting.

Appendix A

The integrated development environment used for the coding of the model, pre and post-transfer learning, is Kaggle with features as follows:

- CPU: Kaggle kernel
- CPU count: 2
- GPU: 15.9 GB
- RAM: 13 GB
- Disk: 73.1 GB
- OS - Linux-5.15.65+-x86_64-with-debian-bullseye-sid
- Python version - 3.7.12
- Python executable -

  ```
  /opt/conda/bin/python
  ```

- GPU count: 1
- GPU type: Tesla P100-PCIE-16GB
- Site: https://www.kaggle.com

Google Colab has been used for Geo-referencing experimentation purposes, with features such as:

- CPU: Python3 Google Compute Engine backend

- CPU count: 1
- GPU: 15.9 GB
- RAM: 12.68 GB
- Disk: 107.72 GB
- OS - Linux-5.10.147+-x86_64 - with -glibc2.31
- Python version - 3.9.16
- Python executable -

  ```
  /usr/bin/python3
  ```

- GPU count: 1
- GPU type: Tesla T4
- Site: http://colab.research.google.com

A Proprietary System provided by RRSC - East has been used for training the proposed AE-GAN model.

- OS - Windows-10-10.0.19044-SP0
- Python version - 3.9.13
- Python executable -

  ```
  c:\Users\Administrator\Anaconda3\envs\supres\python.exe
  ```

- CPU count: 24
- GPU count: 2
- GPU type: Quadro P4000

Appendix B

Here are the extra samples of Images and their respective Geo-transformation information which has been used for testing and validation purposes of the proposed AE-GAN model.

Geo-Information for Image 1:

```
Extent 632400.8399999999674037,2497648.4799999999813735
    :646683.6400000000139698,2510696.4799999999813735
    Width 5101
    Height 4660
Data type UInt16 - Sixteen bit unsigned integer
GDAL Driver Description GTiff
GDAL Driver Metadata GeoTIFF
Compression PACKBITS
```

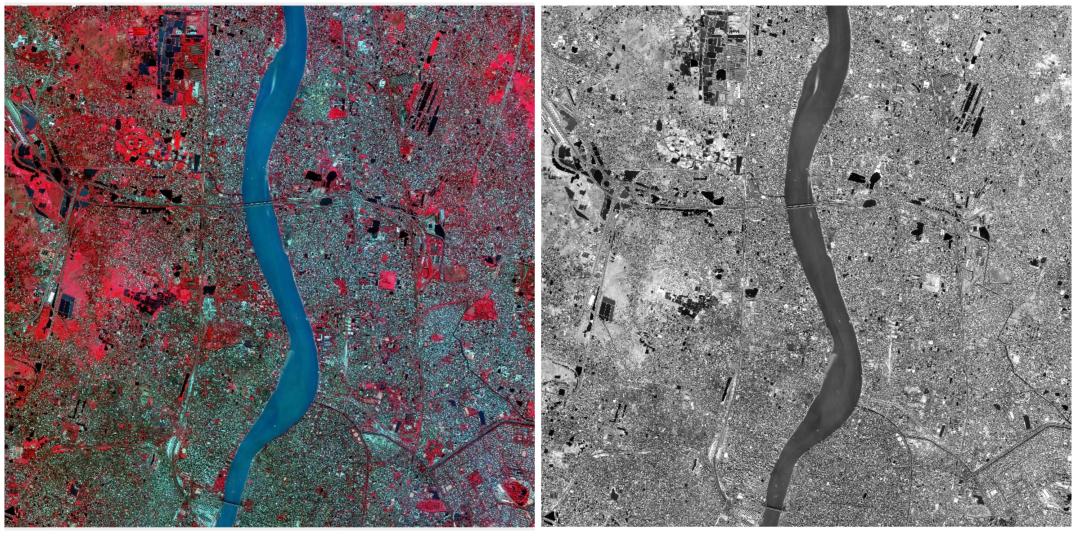

FIGURE B.1: Low-Resolution and High-resolution of Image 1

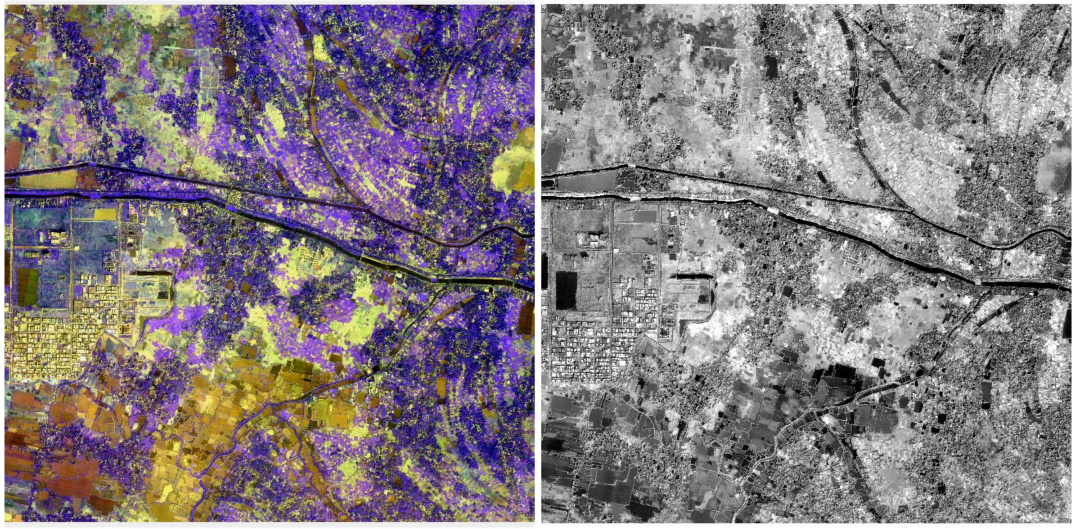

FIGURE B.2: Low-Resolution and High-resolution of Image 2

```
Band 1
STATISTICS_APPROXIMATE=YES
STATISTICS_MAXIMUM=16383
STATISTICS_MEAN=2160.4164735095
STATISTICS_MINIMUM=0
STATISTICS_STDDEV=847.23461236968
STATISTICS_VALID_PERCENT=100
Scale: 1
Offset: 0
Dimensions X: 5101 Y: 4660 Bands: 3
Origin 632400.8399999999674037,2510696.4799999999813735
Pixel Size 2.799999999999999822,-2.799999999999999822
```

Geo-Information for Image 2:

```
Extent 654625.7099999999627471,2485631.799999999813735
    :662974.7099999999627471,2492407.7999999998137355
    Width 30360
    Height 24640
Data type Float32 - Thirty two bit floating point
GDAL Driver Description - GTiff
GDAL Driver Metadata -GeoTIFF
Compression
```

```
Band 1
STATISTICS_APPROXIMATE=YES
STATISTICS_MAXIMUM=43.885677337646
STATISTICS_MEAN=13.742198767828
STATISTICS_MINIMUM=7.7095885276794
STATISTICS_STDDEV=2.172739588685
STATISTICS_VALID_PERCENT=100
Scale: 1
Offset: 0
More information
AREA_OR_POINT=Area
Dimensions X: 30360 Y: 24640 Bands: 1
Origin 654625.7099999999627471,2492407.7999999998137355
Pixel Size 0.2750000000000000222,-0.2750000000000000222
```

Bibliography

[1] G. Keilbar, *Modelling Systemic Risk using Neural Network Quantile Regression*. PhD thesis, 08 2018.

[2] F. Bre, J. Gimenez, and V. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks," *Energy and Buildings*, vol. 158, 11 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[4] M. U. Freudenberg, "Tree detection in remote sensing imagery baumerkennung in fernerkundungs- bildmaterial." https://bit.ly/3LpMUIx.

[5] M. BEN-YOSEF, *Multi-Modal Generative Adversarial Networks*. PhD thesis, Hebrew University of Jerusalem, 2018.

[6] J. Riebesell, "Autoencoder." https://tikz.net/autoencoder/.

[7] E. Dahlström, "Super-resolution using dynamic cameras," 2020.

[8] D. Chira, I. Haralampiev, O. Winther, A. Dittadi, and V. Liévin, "Image super-resolution with deep variational autoencoders," *arXiv preprint arXiv:2203.09445*, 2022.

[9] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 136–144, 2017.

[10] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

[11] S. C. Park, M. K. Park, and M. G. Kang, "Super-resolution image reconstruction: a technical overview," *IEEE signal processing magazine*, vol. 20, no. 3, pp. 21–36, 2003.

[12] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super-resolution," *IEEE Computer graphics and Applications*, vol. 22, no. 2, pp. 56–65, 2002.

[13] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar, "Advances and challenges in super-resolution," *International Journal of Imaging Systems and Technology*, vol. 14, no. 2, pp. 47–57, 2004.

[14] G. Tsagkatakis, A. Aidini, K. Fotiadou, M. Giannopoulos, A. Pentari, and P. Tsakalides, "Survey of deep-learning approaches for remote sensing observation enhancement," *Sensors*, vol. 19, no. 18, p. 3929, 2019.

[15] 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*, vol. 38, no. 2, pp. 295–307, 2015.

[16] M. U. Müller, N. Ekhtiari, R. M. Almeida, and C. Rieke, "Super-resolution of multispectral satellite images using convolutional neural networks," *arXiv preprint arXiv:2002.00580*, 2020.

[17] M. A. Nuño-Maganda and M. O. Arias-Estrada, "Real-time fpga-based architecture for bicubic interpolation: an application for digital image scaling," in *2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig'05)*, pp. 8–pp, IEEE, 2005.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, pp. 630–645, Springer, 2016.

[20] T. Tong, G. Li, X. Liu, and Q. Gao, "Image super-resolution using dense skip connections," in *Proceedings of the IEEE international conference on computer vision*, pp. 4799–4807, 2017.

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[22] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," *arXiv preprint arXiv:1611.02163*, 2016.

[23] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[24] D. Mahapatra, B. Bozorgtabar, and R. Garnavi, "Image super-resolution using progressive generative adversarial networks for medical image analysis," *Computerized Medical Imaging and Graphics*, vol. 71, pp. 30–39, 2019.

[25] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy, "Esrgan: Enhanced super-resolution generative adversarial networks," in *Proceedings of the European conference on computer vision (ECCV) workshops*, pp. 0–0, 2018.

[26] M. S. Sajjadi, B. Schölkopf, and M. Hirsch, "Enhancenet: Single image super-resolution through automated texture synthesis," *arXiv preprint arXiv:1612.07919*, 2016.

[27] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[28] Z.-S. Liu, W.-C. Siu, and Y.-L. Chan, "Photo-realistic image super-resolution via variational autoencoders," *IEEE Transactions on Circuits and Systems for video Technology*, vol. 31, no. 4, pp. 1351–1365, 2020.

[29] I. Gatopoulos, M. Stol, and J. M. Tomczak, "Super-resolution variational auto-encoders," *arXiv preprint arXiv:2006.05218*, 2020.

[30] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[33] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[34] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer.," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[35] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[36] F. Yang, H. Yang, J. Fu, H. Lu, and B. Guo, "Learning texture transformer network for image super-resolution," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5791–5800, 2020.

[37] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological cybernetics*, vol. 20, no. 3, pp. 121–136, 1975.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[39] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, Citeseer, 2013.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European conference on computer vision*, pp. 694–711, Springer, 2016.

[42] J. Bruna, P. Sprechmann, and Y. LeCun, "Super-resolution with deep convolutional sufficient statistics," *arXiv preprint arXiv:1511.05666*, 2015.

[43] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," *Advances in neural information processing systems*, vol. 28, 2015.

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[45] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[46] M. Menéndez, J. Pardo, L. Pardo, and M. Pardo, "The jensen-shannon divergence," *Journal of the Franklin Institute*, vol. 334, no. 2, pp. 307–318, 1997.

[47] M. Tschannen, O. Bachem, and M. Lucic, "Recent advances in autoencoder-based representation learning," *arXiv preprint arXiv:1812.05069*, 2018.

[48] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.," *Journal of machine learning research*, vol. 11, no. 12, 2010.

[49] S. Rifai, G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. Dauphin, and X. Glorot, "Higher order contractive auto-encoder," in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 645–660, Springer, 2011.

[50] A. Makhzani and B. Frey, "K-sparse autoencoders," *arXiv preprint arXiv:1312.5663*, 2013.

[51] A. Gavade and P. Sane, "Super resolution image reconstruction by using bicubic interpolation," 10 2013.

[52] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[53] W. Wu, "patchify 0.2.3." https://pypi.org/project/patchify/.

[54] O. S. G. Foundation, "Gdal." https://gdal.org/.

[55] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th international conference on pattern recognition*, pp. 2366–2369, IEEE, 2010.

[56] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[57] K. Nelson, A. Bhatti, and S. Nahavandi, "Performance evaluation of multi-frame super-resolution algorithms," 12 2012.

[58] M. Nikolova and G. Steidl, "Fast ordering algorithm for exact histogram specification," *IEEE Transactions on Image Processing*, vol. 23, no. 12, pp. 5274–5283, 2014.

[59] A. N. Avanaki, "Exact global histogram specification optimized for structural similarity," *Optical review*, vol. 16, pp. 613–621, 2009.

[60] X. Liu, P. He, W. Chen, and J. Gao, "Improving multi-task deep neural networks via knowledge distillation for natural language understanding," *arXiv preprint arXiv:1904.09482*, 2019.

[61] NASA, "What is remote sensing?." https://www.earthdata.nasa.gov/learn/backgrounders/remote-sensing.