Deep Learning for Super-Resolution of Meteorological Data

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

Submitted by

Kaustubh Atey
Indian Institute of Science Education and Research, Pune

Under the supervision of

Dr. Bipin Kumar (IITM Pune)
Co - Supervisior: Dr. Manmeet Singh (IITM Pune)
TAC Member: Dr. Anupam Kumar Singh (IISER Pune)

Certificate

This is to certify that this dissertation entitled **Deep Learning for Super-Resolution of Meteorological Data** towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Kaustubh Atey at Indian Institute of Tropical Meteorology, Pune under the supervision of Dr. Bipin Kumar and Dr. Manmeet Singh during the academic year 2021-2022

Dr. Bipin Kumar
Scientist E, HPCS
IITM Pune

Committee:

Supervisor: Dr. Bipin Kumar
Co-Supervisor: Dr. Manmeet Singh
TAC Member: Dr. Anupam Kumar Singh

This thesis is dedicated to my Mother
I pray for her joyful and healthy life ahead.

Declaration

I hereby declare that the matter embodied in the report entitled **Deep Learning for Super-Resolution of Meteorological Data** are the results of the work carried out by me at the Indian Institute of Tropical Meteorology, Pune, under the supervision of Dr. Bipin Kumar and Dr. Manmeet Singh and the same has not been submitted elsewhere for any other degree.

Kaustubh Atey

Acknowledgments

I'm very grateful to have been granted the opportunity to work on my masters project under the supervision of Dr Bipin Kumar at IITM Pune. I wish to thank him for always bestowing faith and patience towards me. His guidance and constant motivation helped me throughout the project. Sir has provided me with exposure to excellent research and work environment.

I want to express sincere gratitude to Dr Manmeet Singh, Dr Rajib Chattopadhyay and Mr Bhupendra Bahadur Singh for providing constant guidance and insightful discussions throughout the project. Additionally, I would like to thank Dr Anupam Kumar Singh for always showing a keen interest in the project and giving valuable suggestions and comments.

My heartfelt appreciation goes to my elder brother Dr Saurabh Atey for always inspiring and motivating me. He was a constant support pillar in this journey. I look up to him in the hope to be a better and more humble person like him.

Next, I wish to acknowledge my dear friends Avinash Verma, Gitesh Masram, Aditya Singh Bagri and Ankur Prakash. I owe them a debt of gratitude for taking care and looking after me in my tough times. Their company was an absolute delight. I wish you all the best in all your future endeavours.

Machine learning is one of the things in my life that always excites me and burns curiosity within me. Finally, a big shoutout to all the researchers and open source contributors who made the deep learning revolution possible and accessible for all. *"If I have seen further it is by standing on the shoulders of Giants."*

Abstract

For any country, understanding and interpreting precipitation dynamics is of great importance. In India, rainfall patterns profoundly influence daily livelihoods and economic development. Thus, it is essential to get cognizance of local rainfall for better policy making. Often, downscaling methods are used to produce high-resolution projections from low-resolution GCM outputs or observation data.

This study applies a deep generative model called SRGAN to statistically downscale precipitation data from IMD over the Indian region. Our analysis shows that SRGAN performs comparatively better than other deep learning methods used for downscaling. We used SRGAN to downscale the precipitation data from 0.25° to 0.125° and 0.0625° resolutions and found that the downscaling results closely matched station observations. We also introduce a custom trained VGG based feature extractor that can act as a backbone for other DL models using meteorological data.

Our study establishes that SRGAN can be used as a reliable statistical downscaling model. SRGAN yields result faster than the dynamical RCMs, allowing for more practical real-time applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Climate studies have facilitated insights into short- and long-term climate change impacts. To understand the complex climate dynamics of the Earth, researchers study the continual intricate interactions within the land, ocean and atmosphere. Such understanding helps to formulate efforts for weather events and climate change. The observation data recorded at various stations are converted to gridded format and used for simulation and study of climate dynamics.

It is often challenging to obtain high-resolution grids from station data. Thus the gridded observational data used in meteorological studies have a spatial resolution of a few hundred kilometres. Similarly, the outputs of Global Climate Models (GCMs) are also limited to the spatial resolutions of the order of hundreds of kilometers.

The resolution of such models and station observation data is too coarse for studies at the regional and local levels of a few kilometers. Therefore, a need arises to generate the high-resolution projections.

High-resolution projections are obtained by a technique called **Downscaling** that uses data known at large scales to make predictions for local scales. The downscaling approaches comprise of the Dynamical Downscaling and Statistical Downscaling methods. The dynamical downscaling methods employ regional, numerical models on the output of ESMs, GCMs and give high-resolution projections for local regions. These are highly computationally intensive. Another approach is statistical downscaling **(SD)** which develop a statistical relationship between the large scale variables and the local scale

variables. This relationship is used on the observation data or the outputs from GCMs to get the high-resolution projections for local regions.

Recent advances in Single Image Super-Resolution **(SISR)** motivate the use of deep learning models for statistical downscaling. A significant advantage of these methods is that they capture the non-linearity present in the data. Previous works that employed deep learning methods for the statistical downscaling have shown promising results. We used **SRGAN**, a deep learning architecture that can possibly achieve state of the art results for the SD problem.

1.1 Contributions and thesis structure

The annual monsoon rains are of utmost importance to the Indian economy. As the monsoon brings more than 50% of India's yearly rainfall, it impacts the livelihoods of a large Indian population directly or indirectly. Thus, its study becomes inherently essential. This work performs deep learning based downscaling of precipitation data over the Indian land region. Having regional rainfall insights is very valuable, especially for the agriculture sector. Also, such reliable insights can reasonably equip us for extreme events like floods or famine.

The thesis is organised as follows: Section 3 provides details of previous works on downscaling of precipitation data using DL algorithms, followed by the theory of downscaling and deep leaning in Section 2. Sections 4.1 and 4.2 discuss the details of the data and methodology used in this work. Section 5 covers the ablation study conducted for the model. Finally, the results and conclusion of our work are provided in Sections 6 and 7.

Chapter 2

Theoretical Background

The chapter serves as a brief outline of the theoretical concepts pertaining to our work. These essential topics present a sufficient background to study and understand the thesis. We discuss the downscaling problem and look into various techniques for interpolating geospatial data. Next, we stride into details of deep learning and GANs.

2.1 What is Downscaling?

Downscaling is a general term that refers to a basket of techniques used to derive projections from low-resolution to high-resolution meteorological data. These methods introduce new gridpoints within the same area, and the objective is to estimate their values. Based on the approach used, such strategies are broadly categorised into dynamical and statistical downscaling. The following subsection features these methods and their limitations.

2.1.1 Dynamical Downscaling

This approach includes the use of high-resolution local dynamical models. The outputs from GCMs or observational data is input to the local numerical models to generate climate variables at the scales of interest. The regional climate models (RCMs) yield the local projections by solving non-linear partial differential equations governing the physics of regional climate variables. Such predictions are very accurate as they explicitly model the underlying physical principles such as the laws of thermodynamics, fluid mechanics etc.

However, it is highly computationally expensive to model and compute the local dynamics. Such simulations require high computing power, thus limiting the resolution enhancement. As RCMs take a lot of time to generate results, large supercomputers are vital to run models and speed up the process. The compute requirements and the time complexity of dynamical downscaling techniques cap its usability in many places.

2.1.2 Statistical Downscaling

This approach includes developing a statistical relationship between the large and local scale variables. The observational data or the large scale outputs from the GCMs are input to the statistical models to estimate the regional climate variables. Unlike the RCMs, these statistical downscaling methods do not require heavy computing resources enabling site-specific projections. These methods are less time-consuming, which facilitates practicality and actionability.

However, the SD methods may not be very accurate like the dynamical approaches. The statistical relationship developed determines the precision of these models. Generally, bias correction strategies are adopted to improve the fitting of statistical models. As the climate changes over time, the relationship may not invariably be valid and must be revised.

Statistical downscaling can be formulated as a functional mapping between low resolution **(LR)** data and corresponding high resolution **(HR)** data. An SD model (Equation: 2.1) will take $\mathbf{X}$: LR data as input and generate $\hat{\mathbf{Y}}$: super resolution **(SR)** data as output.

$$\hat{\mathbf{Y}} = \mathbf{F}(\boldsymbol{\Theta}, \mathbf{X}) \tag{2.1}$$

The objective is to find a set of parameters ($\boldsymbol{\Theta}$) such that the loss ($\mathbf{L}(\mathbf{Y}, \hat{\mathbf{Y}})$) between the generated data ($\hat{\mathbf{Y}}$) and its corresponding ground truth ($\mathbf{Y}$) is minimized.

2.2 Interpolation

The climate data is acquired by numerous stations located across the country. Due to the varying geographic conditions, there is irregular spacing between

these stations. Any dynamical or statistical studies require converting the station data onto an equispaced regular grid. Such conversion is achieved using interpolation techniques. Similarly, interpolation allows estimating the station information using the gridded data.

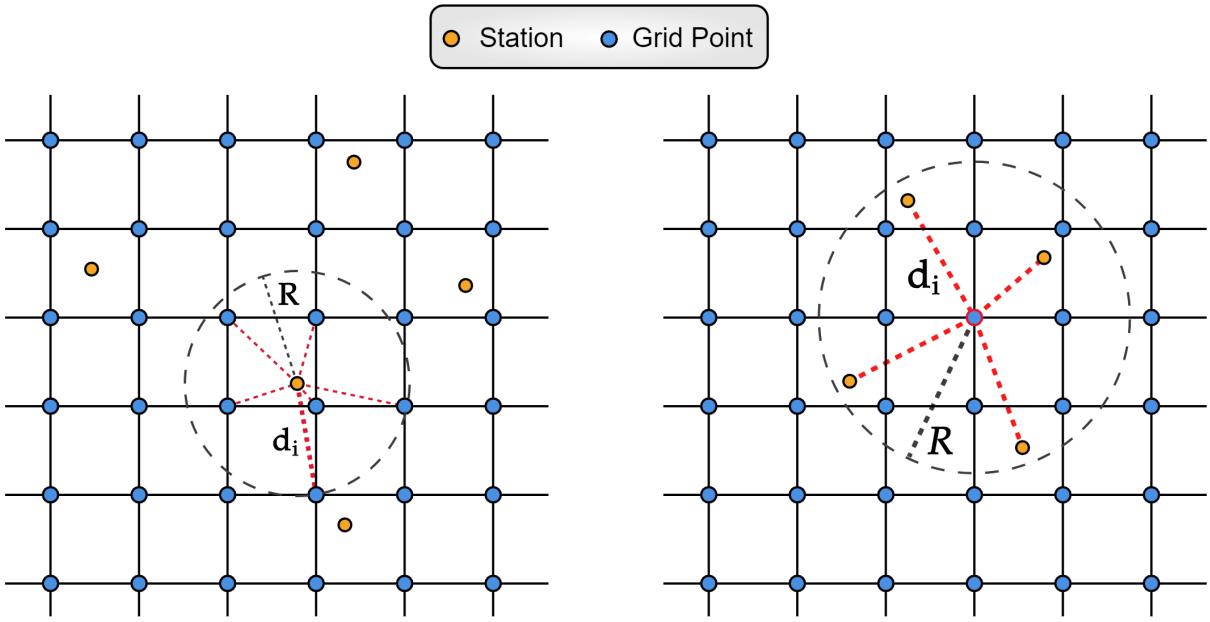


Figure 2.1: **Left:** Grid-to-Station interpolation ; **Right:** Station-to-Grid interpolation

2.2.1 Mathematical Formulation

Consider a set of finite scattered points $X_i = (x_i, y_i, z_i, p_i)$, where: x_i, y_i, z_i: spatial coordinates and p_i: precipitation data value.

We need to esimate the precipitation value p_o for some point $X_o := (x_o, y_o, z_o)$

Let $d(X_o, X_i)$: Distance between known point X_i and unknown point X_o

Example 1:

Consider a set of 5 points $X_1, ..., X_5$. Find the precipitation value for X_o = (60.51, 40.35, 76.21, ? / 110.99) using the given information.

X_1 = (83.96, 32.75, 28.40, 89.30), X_2 = (19.24, 16.73, 24.90, 12.70)
X_3 = (45.88, 69.71, 29.61, 78.43), X_4 = (98.30, 28.97, 20.85, 109.37),
X_5 = (91.04, 59.82, 33.12, 129.66)

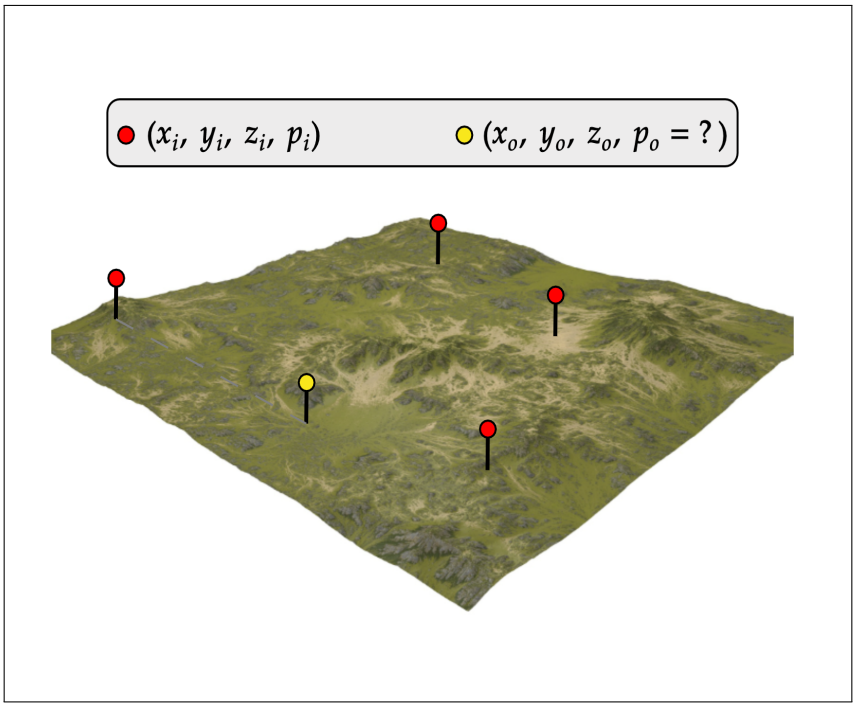


Figure 2.2: Interpolation Scheme

The following sections discuss interpolation methods in detail used for grid-to-station or station-to-grid interpolation. We will solve **Example: 1** using these interpolation methods.

2.2.2 Inverse distance weighting (IDW)

IDW is a multivariate interpolation method for estimating unknown location's values using the known location's values. It assumes each point has a local influence over other points in the considered space. This influence is

proportional to the inverse of the distance between points.

The precipitation value for some point X_o is estimated using:

$$p(X) = \begin{cases} \frac{\sum_N^{i=1} w_i(X) u_i}{\sum_N^{i=1} w_i(X)} & \text{if } d(X, X_i) \neq 0 \ \forall \text{ i.} \\ p_i & \text{if } d(X, X_i) = 0 \text{ for some i.} \end{cases} \tag{2.2}$$

The weights in Equation: 2.2 are calculated by:

$$w_i(X) = \frac{1}{d(X_o, X_i)^p} \tag{2.3}$$

In Equation: 2.3, $p \in \mathbb{R}^+$ is called power parameter. Higher values of p give higher weightage to the close points.

Solution 1: Using IDW method

First we calculate $d(X_o, X_i)$ for the given points.

For $X_1 \rightarrow$

$$\begin{aligned} d(X_o, X_1) &= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \\ &= \sqrt{(83.96 - 60.51)^2 + (32.75 - 40.35)^2 + (28.40 - 76.21)^2} \\ d(X_o, X_1) &= 53.79 \end{aligned}$$

Similarly, $d(X_o, X_2) = 69.95$, $d(X_o, X_3) = 56.98$, $d(X_o, X_4) = 67.98$, $d(X_o, X_5) = 56.28$

Using $p = 2$, in Equation: 2.3 we get:
$w_1 = 0.0003$, $w_2 = 0.0002$, $w_3 = 0.0003$, $w_4 = 0.0002$, and $w_5 = 0.0003$

Now, using these obtained weights in Equation: 2.2:

$$\begin{aligned} p_o &= \frac{(0.02679 + 0.0025409 + 0.02352915 + 0.02187527 + 0.03890017)}{(0.0003 + 0.0002 + 0.0003 + 0.0002 + 0.0003)} \\ p_o &= 87.41 \end{aligned}$$

2.2.3 Shepard's method

Shepard's method is a modification of the IDW method. It uses the same inverse distance weighting relation. This method uses a modified weighting scheme as given in Equation: **2.4**

$$W_i = (S_i)^2 * (1 + t_i) \tag{2.4}$$

The weights are modified using the weighting factors $S_i = S(d_i)$ based on search radius. Here D_x is the search radius

$$S_i = \begin{cases} \text{Station data used directly,} & d_i = 0 \\ \text{Weight based on } \frac{1}{d_i}, & 0 < d_i < \frac{D_x}{3} \\ \text{Weight based on } 27(\frac{d_i}{D_x} - 1)^2/4D_x, & \frac{D_x}{3} < d_i < D_x \\ \text{Station data not used,} & d_i > D_x \end{cases} \tag{2.5}$$

Shepard's method includes a direction factor to improve the interpolation. The direction factor calculates the influence of a data point from the point of interest with other points in the same direction. The directional weighting term is given by:

$$t_i = \frac{\sum S_j[1 - cos(D_i P D_j)]}{\sum S_j} \tag{2.6}$$

$$D_i P D_j = \frac{(x - x_i)(x - x_j) + (y - y_i)(y - y_j)}{d_i d_j} \tag{2.7}$$

Here: P is the point of interest and D_i are nearby points.

- If D_j is in same direction from P as D_i, $t_i \to 0$ ($\because 1 - cos\theta \approx 0$)
- If D_j is opposite P from D_i, $t_i \to 2$ ($\because 1 - cos\theta \approx 2$)

Solution 2: Using Shepard's method

The $d(X_o, X_i)$ values are already calculated in Solution 1.

Let $D_x = 100$ (Search Radius). For point X_1 we have $d_1 = d(X_o, X_1) = 53.79$
Using Equation: 2.5 we get:

$$S_1 = 27(\frac{d_i}{D_x} - 1)^2/4 \cdot D_x = 27(\frac{53.79}{100} - 1)^2/4 \cdot 100 = 0.014$$

Similarly, $S_2 = 0.006$, $S_3 = 0.012$, $S_4 = 0.006$, and $S_5 = 0.012$

Next, we calculate the direction factors using Equation: 2.6, 2.7

$$\begin{aligned} t_1 &= \frac{\sum S_j[1 - cos(D_1PD_j)]}{\sum S_j} \\ &= \frac{S_2[1 - cos(D_1PD_2)]) + ... + (S_5[1 - cos(D_1PD_5)]}{(S_2 + S_3 + S_4 + S_5)} \end{aligned}$$

$$\begin{aligned} D_1PD_2 &= \frac{(x_o - x_1)(x_o - x_2) + (y_o - y_1)(y_o - y_2) + (z_o - z_1)(z_o - z_2)}{d_1 d_2} \\ &= \frac{(60.5 - 83.9)(60.5 - 19.2) + ... + (76.2 - 28.4)(76.2 - 24.9)}{(53.7 * 69.9)} \end{aligned}$$

$D_1PD_2 = 0.442$

Similarly, calculate D_1PD_j for other values of j. We get $t_1 = 0.256$
Now, using Equation: 2.4 we get:

$$W_1 = (S_1)^2 * (1 + t_1) = (0.014)^2 * (1 + 0.256) = 0.0002$$

Similarly, we calculate values for: W_2, W_3, W_4, W_5

Now, using these obtained weights in Equation: 2.2 we get: $p_o = 86.32$

2.2.4 Kriging

Kriging is a statistical interpolation method that considers the distance and the degree of spatial correlation between points to estimate unknown values. Kriging uses the weighted average scheme (Equation: 2.8a) to estimate unknown values using nearby known points.

$$Z_o = w^T Z \tag{2.8a}$$

$$Aw = b \tag{2.8b}$$

The weights are defined by the localized pattern produced by the sample points. The weights are calculated by solving for w in Equation: 2.8b.

$$A = \gamma(x_i, x_j) \tag{2.9a}$$

$$b = \gamma(x_o, x_i) \tag{2.9b}$$

The semi variance between two points spaced by a distance h is given by:

$$\gamma(h) = \frac{1}{2}(Z(x+h) - Z(x))^2 \qquad \text{(Semi-variance)} \tag{2.10}$$

The values for $\gamma(X_o, X_i)$ is computed by fitting a semivariogram with semivariance values on the y-axis and the distance on the x-axis.

Note: For points closer in space, the semivariance is less compared to far away points.

Generates a measure of error or uncertainty for the estimates.

We have used the Kriging interpolation method for both station-to-grid and grid-to-station interpolation in this work.

Solution 3: Using Kriging method

From Equation: 2.8a we get:

$$\begin{aligned} Z_o &= \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix}^T \begin{bmatrix} Z_1 & Z_2 & Z_3 & Z_4 & Z_5 \end{bmatrix} \\ &= \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix}^T \begin{bmatrix} 89.30 & 12.70 & 78.43 & 109.37 & 129.66 & 110.99 \end{bmatrix} \end{aligned}$$

To calculate the weights w_i we solve the Equation: 2.8b

$$Aw = b$$

$$\begin{bmatrix} \gamma(x_1,x_1) & \gamma(x_1,x_2) & \gamma(x_1,x_3) & \gamma(x_1,x_4) & \gamma(x_1,x_5) \\ \gamma(x_2,x_1) & \gamma(x_2,x_2) & \gamma(x_2,x_3) & \gamma(x_2,x_4) & \gamma(x_2,x_5) \\ \gamma(x_3,x_1) & \gamma(x_3,x_2) & \gamma(x_3,x_3) & \gamma(x_3,x_4) & \gamma(x_3,x_5) \\ \gamma(x_4,x_1) & \gamma(x_4,x_2) & \gamma(x_4,x_3) & \gamma(x_4,x_4) & \gamma(x_4,x_5) \\ \gamma(x_5,x_1) & \gamma(x_5,x_2) & \gamma(x_5,x_3) & \gamma(x_5,x_4) & \gamma(x_5,x_5) \end{bmatrix} * \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = \begin{bmatrix} \gamma(x_o,x_1) \\ \gamma(x_o,x_2) \\ \gamma(x_o,x_3) \\ \gamma(x_o,x_4) \\ \gamma(x_o,x_5) \end{bmatrix}$$

Note that $\gamma(x_i, x_j) = 0$ if $i = j$ and $\gamma(x_i, x_j) = \gamma(x_j, x_i)$.
Next, we calculate the values of semi-variance $\gamma(x_i, x_j)$ and put it in the above equation.
Let's calculate semi-variance for points X_1 and X_2:

$$\begin{aligned} \gamma(X_1, X_2) &= \frac{1}{2}(Z_1 - Z_2)^2 \\ &= \frac{(89.30 - 12.70)^2}{2} \\ \gamma((X_1, X_2) &= \end{aligned}$$

Similarly, we can calculate $\gamma(X_i, X_j)$ for all points.

To calculate the values of $\gamma(X_o, X_i)$, we graph a semivariogram with semivariance values on the y-axis and the distance on the x-axis.

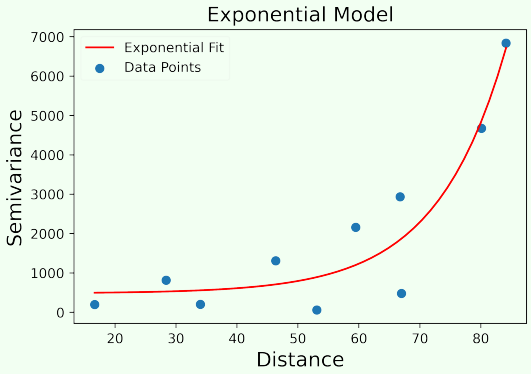


Figure 2.3: Semivariogram

From the above semivariogram, we can estimate the values of $\gamma(X_o, X_i)$ using the distance $d(X_o, X_i)$ between X_o and X_i.

Use the obtained values of $\gamma(X_i, X_j)$ and $\gamma(X_o, X_i)$ in Equation: 2.8b

$$\begin{bmatrix} \gamma(x_1,x_1) & \gamma(x_1,x_2) & \gamma(x_1,x_3) & \gamma(x_1,x_4) & \gamma(x_1,x_5) \\ \gamma(x_2,x_1) & \gamma(x_2,x_2) & \gamma(x_2,x_3) & \gamma(x_2,x_4) & \gamma(x_2,x_5) \\ \gamma(x_3,x_1) & \gamma(x_3,x_2) & \gamma(x_3,x_3) & \gamma(x_3,x_4) & \gamma(x_3,x_5) \\ \gamma(x_4,x_1) & \gamma(x_4,x_2) & \gamma(x_4,x_3) & \gamma(x_4,x_4) & \gamma(x_4,x_5) \\ \gamma(x_5,x_1) & \gamma(x_5,x_2) & \gamma(x_5,x_3) & \gamma(x_5,x_4) & \gamma(x_5,x_5) \end{bmatrix} * \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = \begin{bmatrix} \gamma(x_o,x_1) \\ \gamma(x_o,x_2) \\ \gamma(x_o,x_3) \\ \gamma(x_o,x_4) \\ \gamma(x_o,x_5) \end{bmatrix}$$

Solving the above equation, we get: $w_1 =$, $w_2 =$, $w_3 =$, $w_4 =$, $w_5 =$
Now, use the obtained weights in Equation: 2.8a

$$Z_o = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix}^T \begin{bmatrix} 89.30 & 12.70 & 78.43 & 109.37 & 129.66 & 110.99 \end{bmatrix}$$
$$=$$
$$Z_o = 87.16$$

Chapter 3

Literature Review

This chapter discusses DL models, particularly SRCNN, DeepSD, Augmented ConvLSTM and DCN UNet, used for downscaling meteorological data. These approaches have achieved significant results for the SD tasks.

3.1 SRCNN

Super-Resolution Convolutional Neural Network (SRCNN) [1] is one of the first works that attempted the SISR problem using deep learning techniques. It is a three layer deep CNN architecture, used to learn a functional mapping in spatial domain between the LR and HR data. The input LR data is interpolated to match the size of the HR data and the network is trained end-to-end with the objective to minimize the mean squared error (Equation: 3.1) as loss function.

$$\mathbf{MSE} = \frac{\mathbf{1}}{\mathbf{n}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \|\mathbf{F}(\mathbf{X_i}, \mathbf{\Theta}) - \mathbf{Y_i}\|^{\mathbf{2}} \tag{3.1}$$

Multiple SRCNN modules can be stacked to learn spatial representations at multiple scales where each SRCNN increases the resolution by a factor of s. This allows it to achieve resolution enhancement by a higher factor while requiring less complexity in the spatial representations. Such a network is called **Stacked SRCNN**.

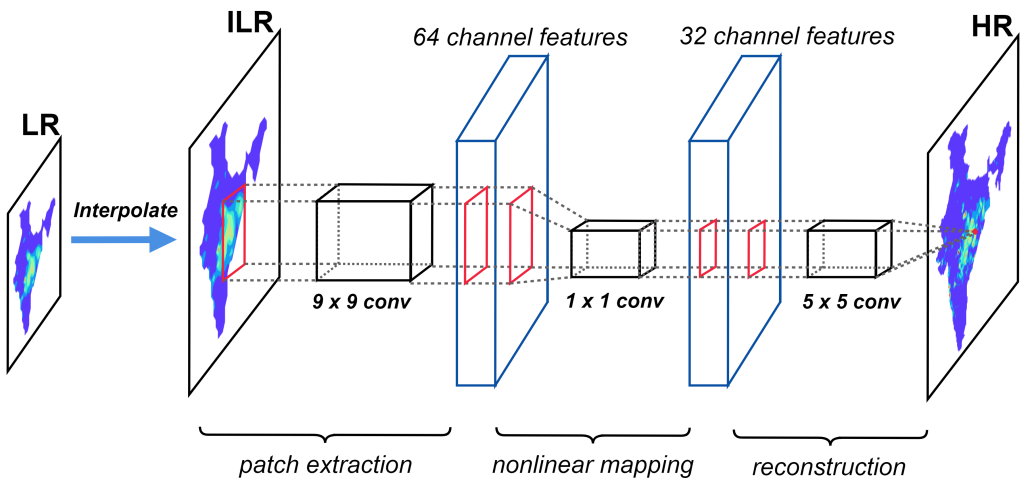


Figure 3.1: SRCNN Network
Source: Adopted from Dong et al.(2014) [1]

3.2 DeepSD

Deep Statistical Downscaling (Deep SD) [2] uses an augmented stacked SRCNN for SD of precipitation data. This novel SD technique leverages by using topography/ elevation in addition to the LR precipitation data as input. The topographical features are available in HR that aids the network to learn to estimate HR precipitation. Each SRCNN is trained independently using their respective input-output resolutions and stacked at test/inference time. Each resolution enhancement is estimated from the previous layer's estimate and its associated HR elevation. The stacked nature of the DeepSD architecture allows the model to capture both regional and local patterns.

Kumar et al.(2021)[3] used DeepSD for downscaling of precipitation data over the Indian region. This work showed that DeepSD successfully outperforms other methods like linear interpolation, SRCNN and stacked SRCNN. This architecture is also trained with the objective to minimize MSE as loss function. A complete diagram of DeepSD method is decipted in Figure: 3.2

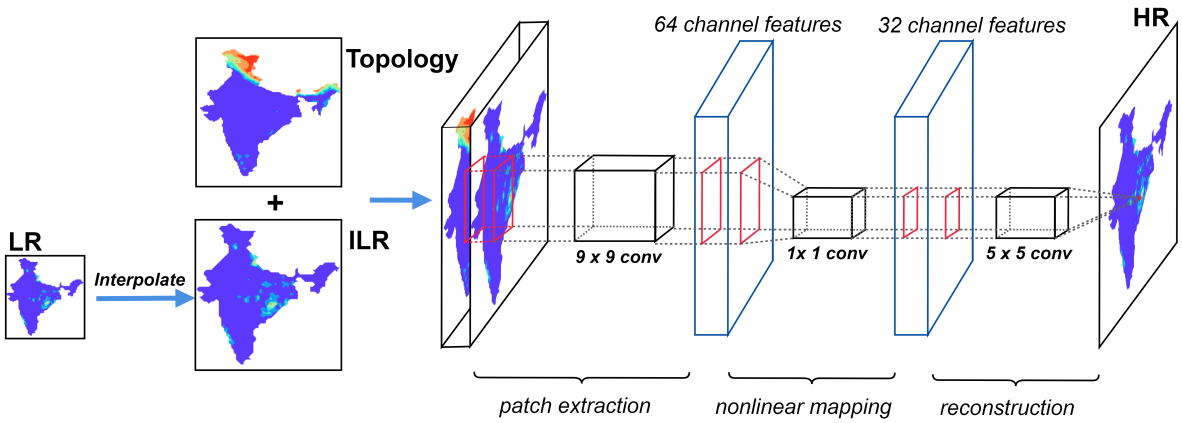


Figure 3.2: DeepSD Network
Source: Kumar et al.(2021)[3]

3.3 ConvLSTM-SR

Harilal et al.(2021)[4] proposed a recurrent convolutional LSTM based approach for SD of climate data. This work captures temporal dependencies in addition to the spatial dependencies by combining fully connected LSTMs with convolutions. On top of the ConvLSTM layers, they have used a novel SR block that increases the resolution of high dimensional feature data obtained from the preceding ConvLSTM layers. In addition to the precipitation and elevation, the input is augmented with other physics guided auxiliary variables like: relative humidity, pressure and 3 wind components.

$$\mathbf{RMSE} = \sqrt{\frac{\sum_{\mathbf{i=1}}^{\mathbf{n}} \left\|\mathbf{F}(\mathbf{X_i}, \mathbf{\Theta}) - \mathbf{Y_i}\right\|^{\mathbf{2}}}{\mathbf{n}}} \tag{3.2}$$

The model is trained end-to-end with an objective of minimizing the root-mean-square error (Equation: 3.2) as loss function. By capturing both spatio-temporal dependencies and including additional auxiliary variables in the input, the ConvLSTM-SR architecture is able to outperform DeepSD.

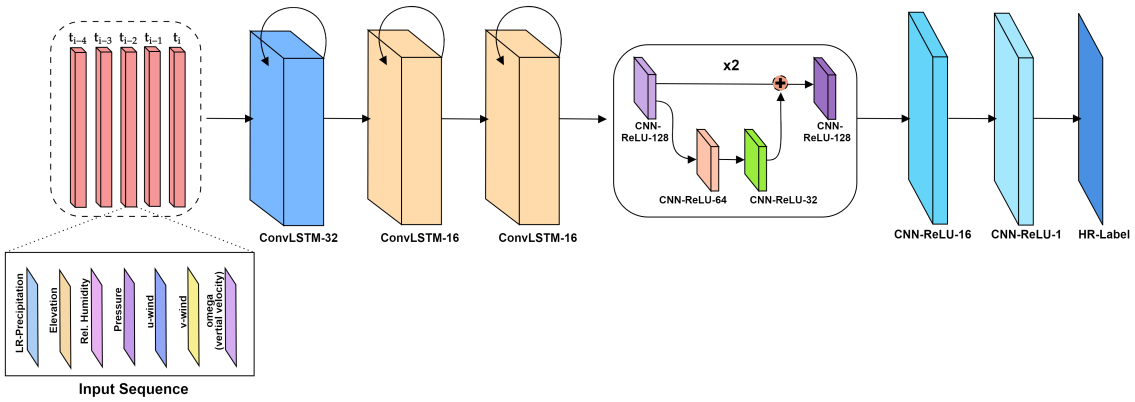


Figure 3.3: Augmented Conv LSTM Network
Source: Harilal et al.(2021) [4]

3.4 Deconvolutional Network (DCN): U-Net

DCN U-Net [5] presented an encoder-decoder based architecture for downscaling of high-frequent meteorological variables like precipitation. This method uses a U-Net architecture to learn spatio-temporal functional mapping. The input LR data is spatially downscaled using bilinear interpolation and augmented with the topography channel. This input tensor is passed to the encoder part of the U-Net and the time variable is appended in the latent space just before the decoder part of the model.

$$\mathbf{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = \underbrace{|\mathbf{Y} - \hat{\mathbf{Y}}|_1}_{data\ loss} + \underbrace{\lambda|\nabla\mathbf{Y} - \nabla\hat{\mathbf{Y}}|_1}_{gradient\ loss} \tag{3.3}$$

This method combines L1 norm with the gradient loss to penalize the differences in the values and in the derivatives. This loss function (Equation: 3.3) aids in the reconstruction of higher frequency details.

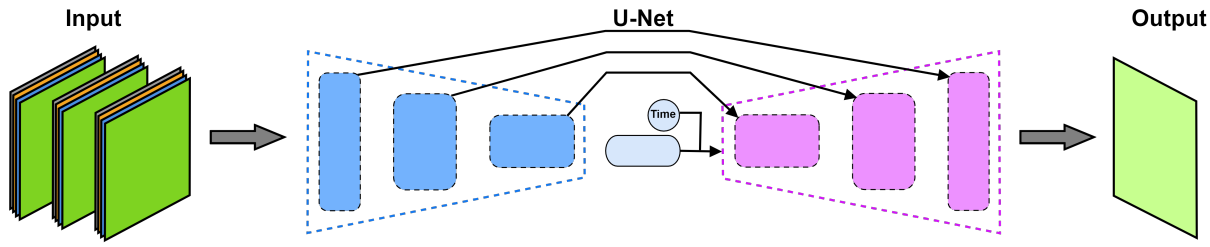


Figure 3.4: DCN U-Net Network
Source: Serifi et al.(2021) [5]

3.5 What is Deep Learning?

Deep learning or DL is a subset of machine learning that has witnessed monumental development in the past decade. The success of DL attributes to its remarkable results for perceptual tasks such as image classification and speech recognition. Apart from the perceptual tasks, DL has shown promising results for weather forecasting, protein folding, and healthcare problems.

A central idea in DL is to learn representations and recognise patterns within the given data and use these insights for problems like classification or regression. DL methods use a vast amount of data to learn a mapping from input to a target. The feedback from the evaluation of mapping output against the target revises the mapping. The process of making modifications to the mapping is called learning.

At its core, every DL algorithm follows a standard flow (Figure: 3.5). DL models are composed of multiple **layers** that transform the given input. The **weights** of the layer define the mode of transformation. A **loss function** measures the fidelity of a set of weights. It calculates the distance between the model output and the respective target to determine the score of model's performance for a particular example. The calculated loss serves as feedback to modify the weights to reduce the loss. An **optimizer** updates the weights using the **backpropagation** algorithm.

The entire cycle from passing the inputs to finally updating the model weights is called the training step. Training a DL model implies repeating the training step multiple times until convergence is achieved. The convergence state is when the model loss does not decrease on further training.

The following section discusses some commonly used neural networks, activation functions, and loss functions.

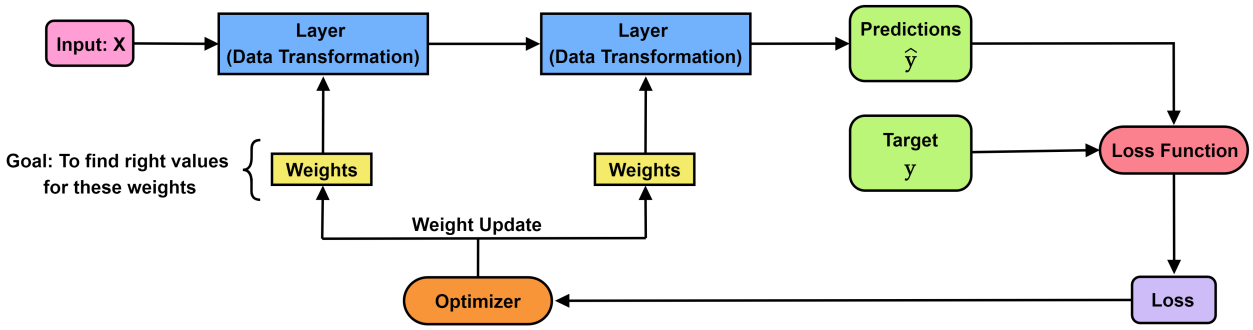


Figure 3.5: Flowchart of a DL algorithm

3.5.1 Deep Neural Networks

Neural Networks (NNs or Neural Nets) are composed of multiple differentiable functions. The term deep implies the usage of multiple successive layers in neural networks. These NNs learn an end to end mapping between the input and the targets for the given tasks.

Feed Forward Neural Networks

Feed Forward Neural Networks are deemed as the fundamental building blocks of deep learning. The term feedforward refers to the unidirectional nature of these networks. It consists of an input layer followed by multiple hidden layers and an output layer.

Each hidden layer has some weights and bias terms associated with it, which it learns during training. The general terms for a feedforward network is given by:

- The pre-activation at layer i is given by:

  $$a_i(x) = W_i h_{i-1}(x) + h_i$$

  where weights W_i : $m \times n$ matrix for n inputs and and m neurons in layer i.
  Here: $a_i \in \mathbb{R}^m$, $W_i \in \mathbb{R}^{m \times n}$, $x_i \in \mathbb{R}^{n \times 1}$ and $b_i \in \mathbb{R}^m$

- The activation at layer i is given by:

  $$h_i(x) = g(a_i(x))$$

  where g is an activation function (details in activation section)

- The activation at output layer L is given by:

$$f(x) = h_L = O(a_L)$$

  where O is output activation function.

The two hidden layer feedforward network illustrated in Figure: 3.6 can be written as

$$\hat{y} = f(x) = O(W_3 g(W_2 g(W_1 x + b) + b_2) + b_3)$$

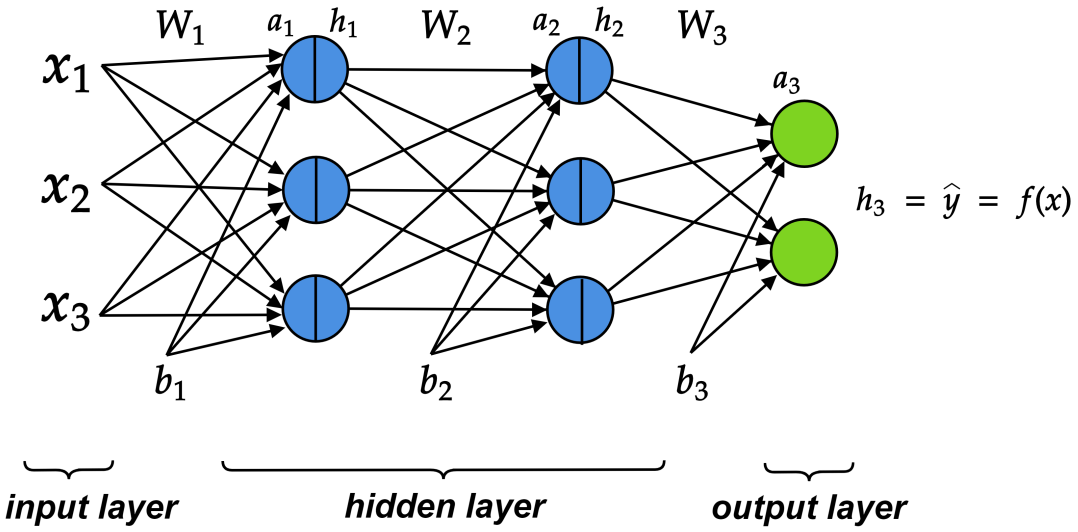


Figure 3.6: Feed Forward Neural Networks

Despite its great success on various tasks, feed-forward neural networks have some drawbacks. The number of parameters increases with the depth of the network, which makes them prone to overfitting. Also, due to long chains of functions, the gradients can vanish.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs or ConvNets) have earned massive success on tasks related to spatial data. Modern computer vision and image processing lean heavily on these neural nets. CNNs have addressed various drawbacks of the feed-forward networks.

The general principle behind CNN is the 2D convolution operation, which is defined as:

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a,j+b} K_{a,b}$$

where I : input matrix and K : weight matrix.

For an input of width: W_I, height: H_I and depth: D_I, the shape of output on applying 2D convolution is given by:

$$W_O = \frac{W_I - F + 2P}{S} + 1$$
$$H_O = \frac{H_I - F + 2P}{S} + 1$$
$$D_O = K$$

where F : spatial extent of filter, K : number of filters, P : padding and S : stride

CNNs apply a series of convolution operations to the input to learn its feature representations. It uses sparse connectivity (less number of connections between neurons) and weight sharing (neurons within the same channel share weights), giving them an advantage over feed-forward networks. Sparse topology implies fewer computations, while weight sharing implies faster learning. The translation invariance nature of Conv-Nets is also a result of sparse connections and weight sharing.

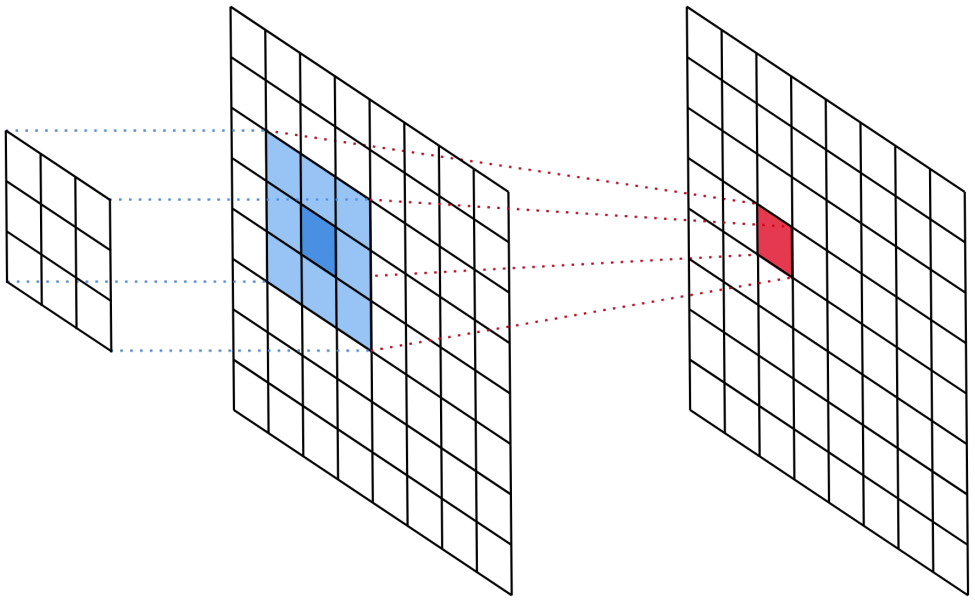


Figure 3.7: 2D convolution operation

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of neural networks used for sequential learning tasks. Unlike spatial data, in sequential data, the outputs also depend on previous inputs. Here the length of inputs is not fixed. The sequence learning problems are defined as follows:

$$y_t = \hat{f}(x_1, x_2, ..., x_n)$$

When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks (Figure: 3.8). These forward pass of an RNN is defined as follows:

$$a_t = b + Wh_{t-1} + Ux_t$$
$$h_t = tanh(a_t)$$
$$o_t = c + Vh_t$$
$$\hat{y}_t = softmax(o_t)$$

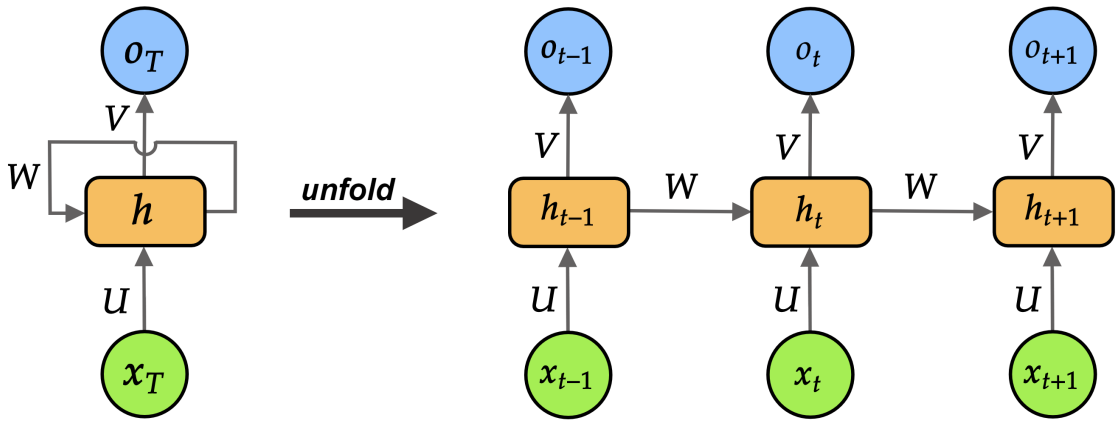


Figure 3.8: Recurrent Neural Network

However, RNNs have some limitations. For long sequences, RNNs fail to capture the sequence's dependencies. This phenomenon is attributed to the problem of vanishing gradients in these networks.

Long Short Term Memory Networks

Long Short Term Memory Networks (LSTMs) are an advanced version of the vanilla RNNs. LSTMs have the ability to capture the long-term dependency, making them more robust than RNNs.

An LSTM network consists of multiple memory blocks called cells. The cell state and the hidden state of the current cell get transferred to the next cell. Each cell comprises gates: the input, output, and forget gate.

Gates:

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \text{ (output gate)}$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \text{ (input gate)}$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \text{ (forget gate)}$$

States:

$$\tilde{s_t} = \sigma(W h_{t-1} + U x_t + b)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s_t}$$
$$h_t = o_t \odot \sigma(s_t)$$

The **input gate** adds new information to the cell state. It controls the information that gets added to the cell state. The **forget gate** keeps check of the information, which is no more required. It is responsible for removing such information from the cell state. Finally, the **output gate** looks for helpful information from the current cell state and outputs the same.

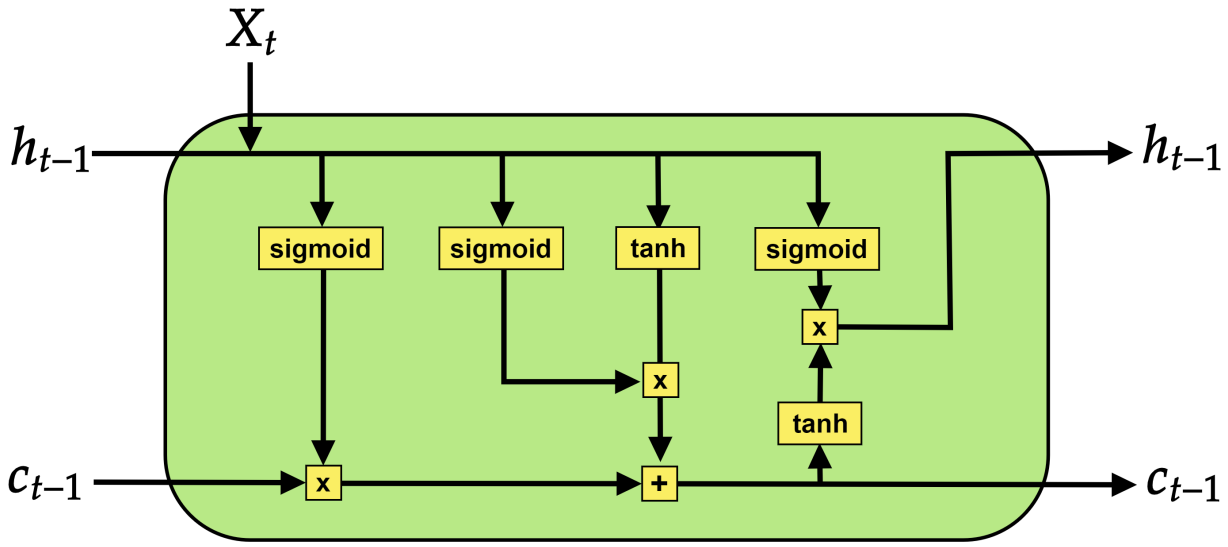


Figure 3.9: LSTM Block

3.5.2 Activation Functions

The cause for the success of neural networks is their competency to capture the non-linearity present in the data. Activation functions enable neural networks to capture non-linearity. Before passing input to the next layer, the activation function applies a non-linear transformation to it. This transformation determines the importance of the information. The following excerpt discusses some commonly used activation functions:

Sigmoid

The sigmoid activation function is widely used for tasks that require probability as an output. Its range is between 0 and 1, so it is best suited for such tasks. Sigmoid has a smooth gradient, preventing any sudden jumps in the output. However, it is almost flat for the large positive and negative values of x, pushing the gradients to converge to zero. This can lead to the vanishing

gradients problem, which forbids the network from learning further.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Hyperbolic Tangent

The Hyperbolic Tangent or the Tanh activation function is centred around 0 and outputs values between -1 and 1. For large positive values, it tends to be 1, and for large negative values, it tends to be -1. Tanh has very steep gradients, and it also fronts the problem of vanishing gradients. However, tanh is preferred over sigmoid in practice due to its zero centred nature, which does not limit its gradients to moving in a particular direction.

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Rectified Linear Unit

The Rectified Linear Unit or ReLU appears as a non-linear extension of a linear function. It outputs zero for negative inputs, making it more computationally efficient as it does not activate all the units simultaneously. However, ReLU encounters the dying ReLU problem. The gradient is zero for the negative inputs, leading to dead neurons during backpropagation. Here dead neurons refer to the state when the weights do not update due to zero gradient value. The dying ReLU problem can inhibit the network's learning capability.

$$f(x) = max(0, x)$$

Leaky ReLU

The Leaky ReLU activation function solves the dying ReLU problem by introducing a slight positive slope for negative values. The non zero gradients allow the weights to update for negative values. However, as such gradients are small, learning can take more time. Also, the behaviour of networks can be invariant for negative input, limiting the predictions.

$$f(x) = max(0.1x, x)$$

Parametric ReLU

Parametric ReLU or PReLU also aims to solve the dying ReLU problem by introducing a slope factor: s for negative values. The network learns the value of a by backpropagation. This allows the network to learn the value of a that best suits the task. However, including a learnable parameter in the activation function can increase the computations during training.

$$f(x) = max(sx, x)$$

where a: slope parameter for negative values

Exponential Linear Units

Exponential Linear Unit or ELU also addresses the dying ReLU problem, using a log curve for negative values. ELU benefits from a smooth curve in the negative region compared to the sharp curve in ReLU and PReLU. However, using an exponential operation comes with the cost of computing time. Also, the value of α is manually tuned, requiring some intuition followed by hit and trial.

$$f(x) = max(\alpha x, x)$$

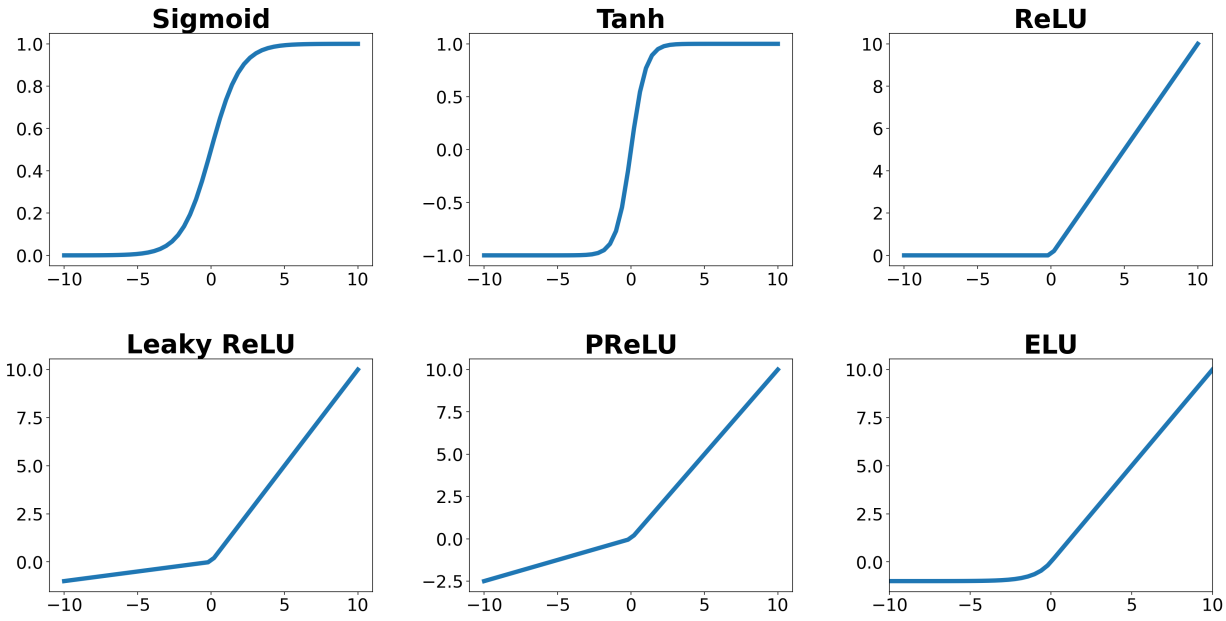


Figure 3.10: Activation Functions

3.5.3 Loss functions

The loss or objective function is essential for optimization in statistical learning. The loss function computes the distance between the network output and the expected ground truth to evaluate the learning performance of a neural network. The calculated loss determines the weight updates on backpropagation. The learning of a neural network directly depends upon selecting the loss function. Therefore, choosing the correct loss function that suits our problem statement is substantial

Generally, loss functions are broadly classified into two classes:

- Classification Loss: Used for classification problems that require predicting a discrete class output.
- Regression Loss: Used for regression problems that require predicting a specific continuous output.

Some commonly used classification loss functions are:

Binary Cross Entropy Loss

Binary Cross-Entropy or BCE loss is used in two-class classification problems. BCE penalizes exponentially for a large difference in the predictions and the targets. While for small differences, it penalizes approximately linearly.

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

where N : number of data points

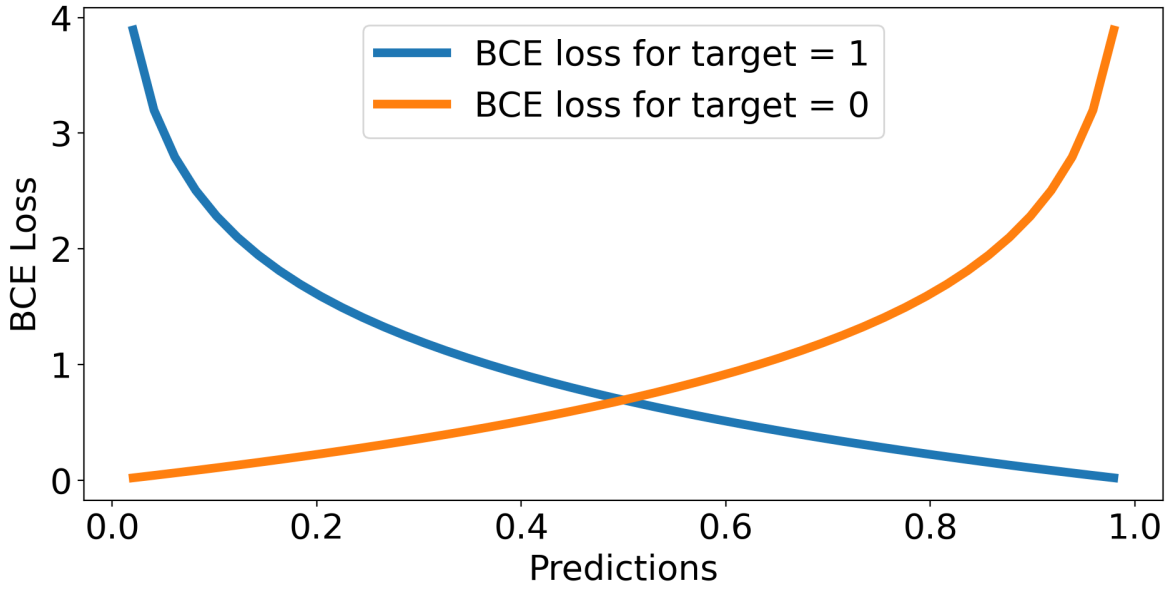


Figure 3.11: BCE Loss

Categorical Cross Entropy Loss

Categorical Cross-Entropy loss extends the idea of BCE for multi-class classification. It requires one-hot encoded labels to calculate the loss. The negative sign ensures that the loss becomes smaller when y_i and $\hat{y}_i$ are closer.

$$Loss = \sum_{i=0}^{n} y_i log(\hat{y}_i)$$

where n : number of classes

Some commonly used regression loss functions are:

Mean Absolute Error

Mean Absolute Error (MAE or L1 loss) computes the average of the absolute difference between the target (y_i) and the predicted values ($\hat{y}_i$). Calculating the absolute difference makes it robust to outliers.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Mean Squared Error

Mean Squared Error (MSE or L2 loss) is the most commonly used loss function for regression problems. It computes the average of the squared differences between the target and the predicted values. MSE is very sensitive to outliers as calculating the square gives them more weightage.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Huber Loss

Huber Loss integrates the L1 and L2 losses. It guarantees the robustness of MAE and the stability of MSE. It is linear for minor errors, while for significant errors, it becomes quadratic. It uses a parameter δ which regulates its behaviour between L1 and L2.

$$Loss = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

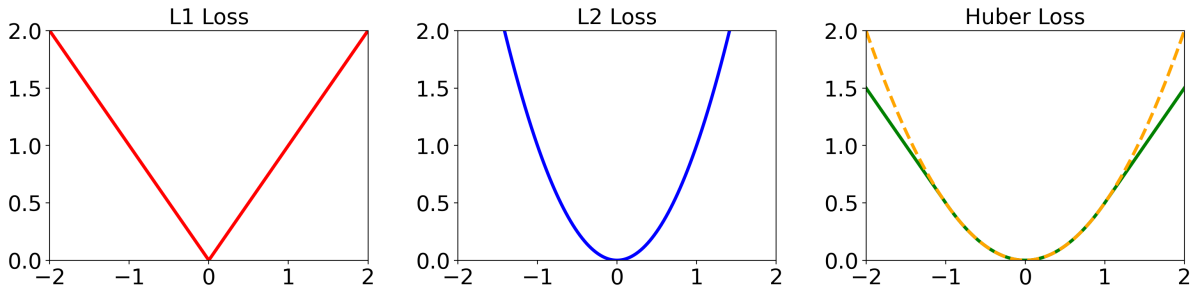


Figure 3.12: Regression Loss

3.6 Generative Adversarial Networks (GANs)

Neural networks have attained massive success on classification and regression problems that required finding patterns in the given data. A fundamental research question persisted:

Can neural networks generate new data?

GANs provided a potential solution to this open research problem. Invented in 2014 by Ian Goodfellow during his PhD, GANs was a breakthrough in ML research. Since its inception, there has been a remarkable growth in research and applications of GANs.

GANs are a class of ML algorithms that employs two models trained concurrently:

- **Generator:** A model trained to generate new data.
- **Discriminator:** A model trained to distinguish generated data from the real examples

The objective of the generator model is to generate data that is indistinguishable from the real training data. In contrast, the objective of the discriminator is to distinguish the data generated by the generator from the real training data. These two networks run parallel and try to outplay one another as they learn during training.

Here, we can quote the famous example from the original GANs paper [6] explaining the concept of GANs:

"The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles."

3.6.1 Working of GANs

The generator and discriminator models are neural networks with respective distinct cost functions. These networks train from the backpropagation

feedback of the discriminator loss. Both networks have a clear objective:

- **Generator** (G) **:** It seeks to maximize the discriminator loss i.e. to generate examples x^* such that $D(x^*) \to 1$
- **Discriminator** (D) **:** It seeks to minimize loss between the real and generated data i.e. for real sample x, $D(x) \to 1$ and for generated sample x^*, $D(x) \to 0$

In typical neural networks, the intent is to find a set of parameter θ such that it minimizes the cost function $J(\theta)$. While in GANs, there are two separate cost functions $J^{(G)}(\theta^{(G)}, \theta^{(D)})$ (for generator) and $J^{(D)}(\theta^{(G)}, \theta^{(D)})$ (for discriminator) each dependent on both network's parameters. However, each network can tune only its parameters, i.e. when training, the generator can update $\theta^{(G)}$ only, and the discriminator can update only $\theta^{(D)}$. Thus training GANs can be regarded as a min-max game rather than a straightaway optimization problem.

During training, the generator learns to synthesize data that follows the data distribution of the training set. An optimal generator can approximately capture the statistics of the real data.

3.6.2 Training GANs

Training GANs is a min-max game where both players: generator and discriminator, constantly try to outwit one another. The training algorithm for GANs is given in Algorithm: 1 (Adopted from: [7])

Training converges when both networks cannot improve further, i.e. Nash equilibrium is reached. Such a situation will arise when $J^{(G)}(\theta^{(G)}, \theta^{(D)})$ is minimized wrt $\theta^{(G)}$ and $J^{(D)}(\theta^{(G)}, \theta^{(D)})$ is minimized wrt $\theta^{(D)}$.

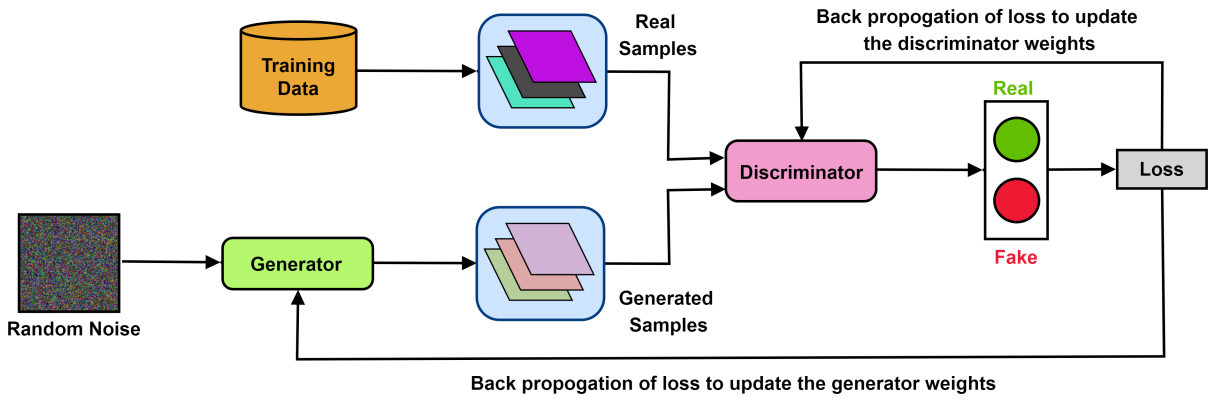


Figure 3.13: Working of GANs

Algorithm 1 GAN training algorithm

for each training iteration **do**

1. Train Discriminator:
   Get real examples from training data: x
   Generate fake examples: $G(z) = x^*$ using random noise vector: z
   Compute classification loss: $D(x)$ and $D(x^*)$
   Backpropogate total loss to update $\theta^{(D)}$

2. Train Generator:
   Generate fake examples: $G(z) = x^*$ using random noise vector: z
   Compute classification loss: $D(x^*)$
   Backpropogate total loss to update $\theta^{(G)}$

end for

Chapter 4

Data and Methodology

4.1 Data

We have used the daily rainfall data [8, 9, 10] from the India Meteorological Department(IMD). The data ranges from the years 1975 to 2009. The data is split into training and testing sets as follows:

- **Training:** 1975-2004
- **Testing:** 2005-2009

The high-resolution data has a spatial resolution of 0.25° x 0.25°. We have used the low resolution data in two resolutions: 1° x 1° and 0.5° x 0.5°. We use the LR data as input and HR data as output for building downscaling models.

4.1.1 Data Preparation

To prepare the LR data, we downsample the HR by factors of 2 and 4. We downsample the data using the interpolation (nearest) module from PyTorch [11].

$$0.25° \text{ (HR)} \xrightarrow{scalefactor=0.5} 0.5° \text{ (LR)}$$

$$0.25° \text{ (HR)} \xrightarrow{scalefactor=0.25} 1.0° \text{ (LR)}$$

4.1.2 Data Preprocessing

The preprocessing steps include min-max normalization (Equation: 4.1) of data. We calculate the minimum and maximum precipitation values by grouping the same days for all the years. We use these values for per day min-max scaling.

$$Z = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.1}$$

After the data preparation and preprocessing steps, our data looks as illustrated in Figure: 4.1

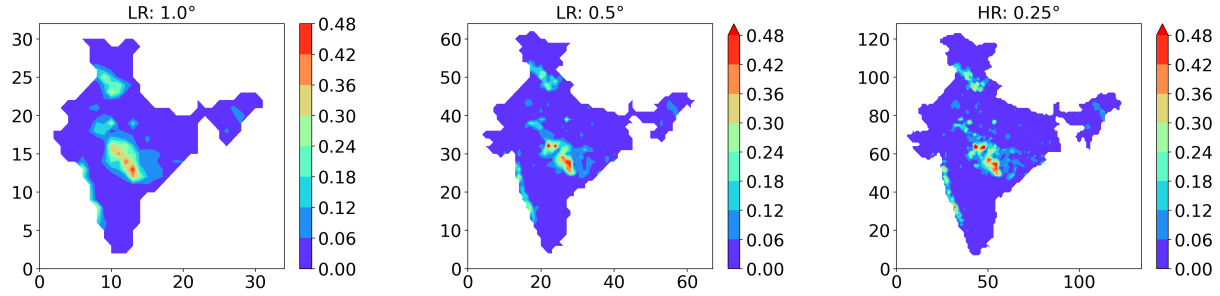


Figure 4.1: Data in different resolutions.

4.2 Methodology

We propose using **SRGAN** (Super-Resolution using a Generative Adversarial Network) [12] for the statistical downscaling problem. It is a **GAN** based deep learning architecture for SISR tasks. The SRGAN model is able to recover finer details when resolving at large scaling factors. This method addresses the problem of smooth reconstruction (section:) by using a novel loss function called perceptual loss. The state-of-the-art performance of SRGAN on SISR tasks is an evident motivation for its usage for the downscaling problem.

4.2.1 Architecture

The SRGAN architecture consists of two sub-modules: a Generator network and a Discriminator network discussed in detail in the subsequent sections.

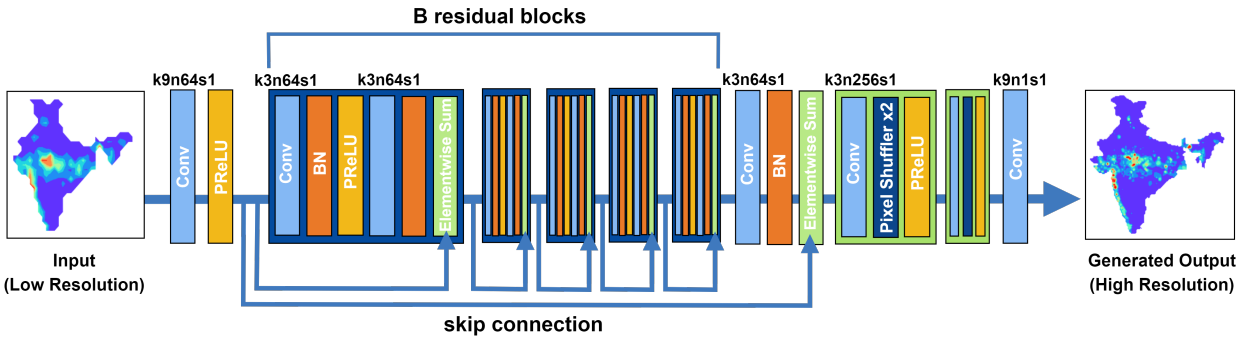


Figure 4.2: Generator Network (k: kernel size, n: number of feature maps, s: stride)
Source: Adopted from [12]

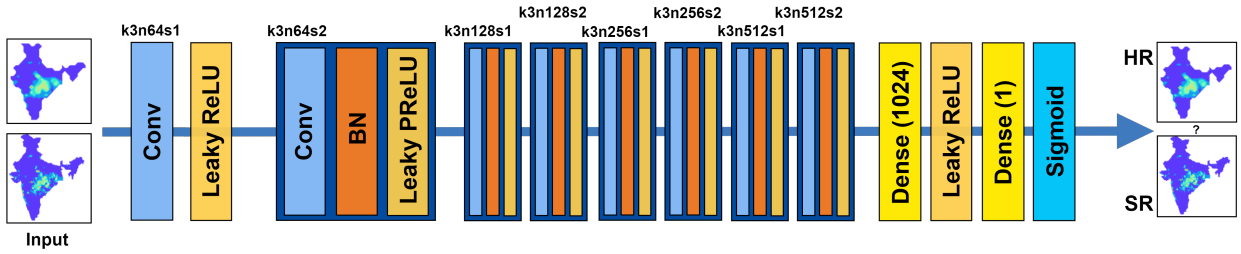


Figure 4.3: Discriminator Network (k: kernel size, n: number of feature maps, s: stride)
Source: Adopted from [12]

Generator

The generator network (Figure: 4.2) generates the SR output, given the LR data as input. It consists of multiple identical residual blocks with skip connections that enable the network to recover textures from heavily down-sampled data.

The **residual blocks** consist of two convolutional layers, each with 64 kernels of size 3 x 3, followed by batch-normalization layers. A Parametric ReLU activation function follows the first BN layer. PReLU generalizes the non-linearity required in different layers by learning the value for the negative slope. Finally, a skip connection performs an element-wise sum after the second BN layer with the input of RB.

Towards the end of the generator, two up-sampling blocks are present. Each up-sampling block has a convolutional layer followed by a pixel shuffle with scale factor = 2 and a PReLU activation. The pixel shuffle layer accounts for a 2x resolution increment for each up-sampling block.

All the convolutional layers in the generator uses a stride = 1, allowing the model to maintain a higher resolution. The idea here is to minimize heavy downsampling of the information in each layer. The final convolutional layer uses a tanh activation function. The advantage of using tanh is that it tends to produce crisper images.

Discriminator

The discriminator network (Figure: 4.3) is a binary classifier with an objective to distinguish the generated data (SR) from the ground truth data (HR). It takes the output generated by the generator network as input and determines if it is close to HR data.

It consists of eight convolutional layers, each followed by a leaky ReLU activation function. Except for the first Conv layer, the remaining layers house a BN layer before activation. After every two Conv layers, the number of kernels increases by a factor of two, ranging from 64 to 512 with size: 3 x 3

The discriminator head has two dense layers and a final sigmoid activation function that computes the probability for classification. The feedback from discriminator is passed to the generator (adversarial training) that allows the generator network to generate finer texture details.

4.2.2 Perceptual Loss

A major drawback of using MSE based loss function is that the generated solution appears overly smooth due to the point-wise average of the solutions. SRGAN addresses this problem by using a novel loss function called perceptual loss (Equation: 4.2) that aids in the adversarial training. The perceptual loss is computed as a weighted sum of content loss (Equation: 4.3) and adversarial loss (Equation: 4.4).

$$l^{SR} = \underbrace{l_X^{SR}}_{content\ loss} + \underbrace{10^{-3} l_{Gen}^{SR}}_{adversarial\ loss} \tag{4.2}$$

The content loss also called **VGG Loss** (Equation: 4.3) is defined as the euclidean distance between the feature representations of the generated data ($\hat{Y}$) and the respective ground truth data (Y). The feature representations are obtained by passing the data as input to a pre-trained VGG network

(Figure: 4.6). VGG [13] is an image classification architecture developed by the Visual Geometry Group (VGG) at the University of Oxford.

Here: $\hat{Y} = G_{\theta_G}(X)$ is the generated output from the generator. $W_{i,j}$ and $H_{i,j}$ describe the dimensions of the respective feature maps within the VGG network. $\phi_{i,j}$ denoted the feature map obtained by the j-th convolution (after activation) before the i-th max-pooling layer within the VGG-19 network.

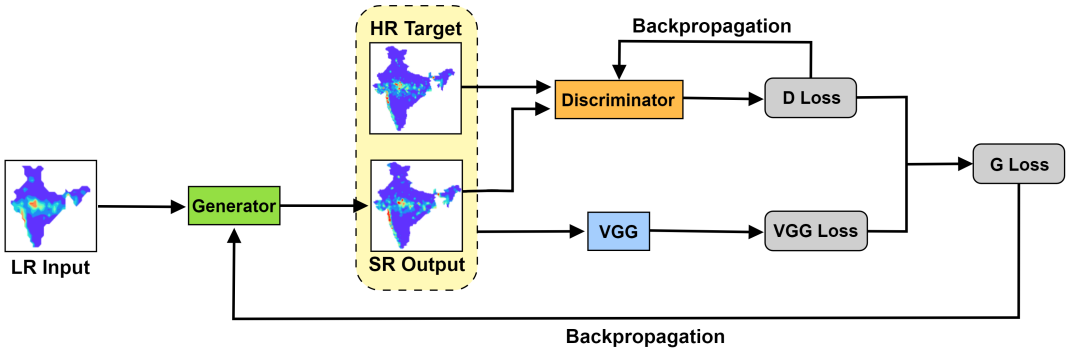
$$l^{SR}_{VGG/i,j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(Y)_{x,y} - \phi_{i,j}(\hat{Y})_{x,y})^2 \tag{4.3}$$

Adversarial loss (Equation: 4.4) computes the probability that the generated SR data is the ground truth HR data. Both the perceptual loss and adversarial training drives the SRGAN model to recover the finer texture details on super-resolution at large factors.

$$l^{SR}_{Gen} = \sum_{n=1}^{N} -logD_{\theta_D}(\hat{Y}) \tag{4.4}$$

4.3 Experiments

Generally, training neural networks is a straightforward optimization problem of finding a set of parameters that seek to minimize the loss function. However, training GANs is like a two player zero-sum game where both the generator and discriminator can tune only their own parameters and not each other's. The following sections explain in detail the procedure for training SRGAN.



fbox

Figure 4.4: SRGAN Training Algorithm

4.3.1 VGG Training

To compute the content loss we need to use a pretrained VGG network . However, we cannot use the publicly available pretrained VGG networks as they are trained on the ImageNet dataset consisting of natural RGB images. Instead, we have to train a VGG network on meteorological data such that it learns rich feature representations for it.

We trained a VGG model for the task of binary classification to distinguish between the LR and HR data. This trained model is used to compute the content loss component of the perceptual loss during the SRGAN adversarial training.

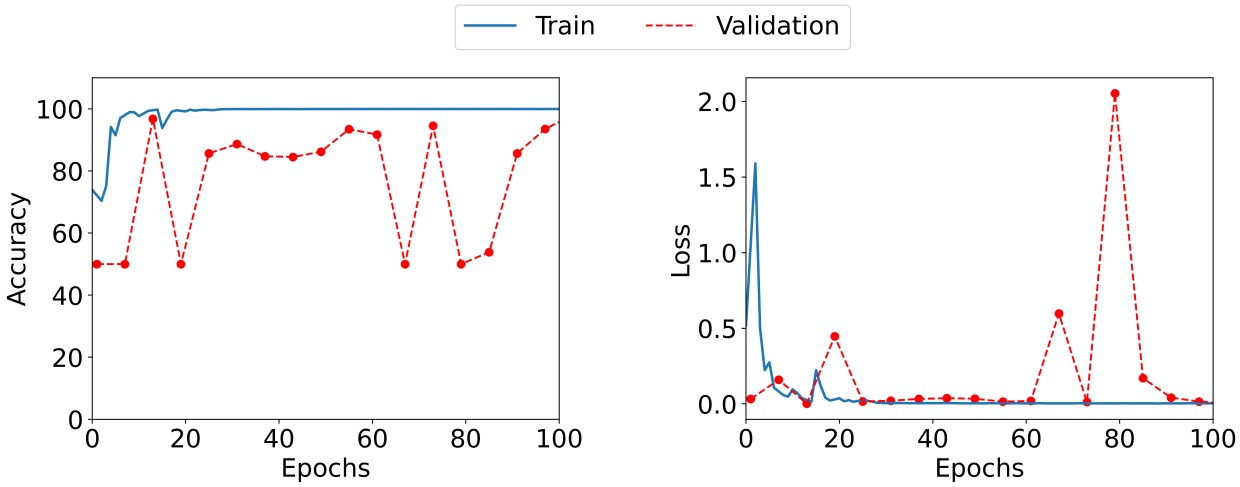


Figure 4.5: VGG learning curves

Table 4.1: **VGG validation results**

Accuracy	TP	FP	TN	FN
95.50	7042	181	5672	417

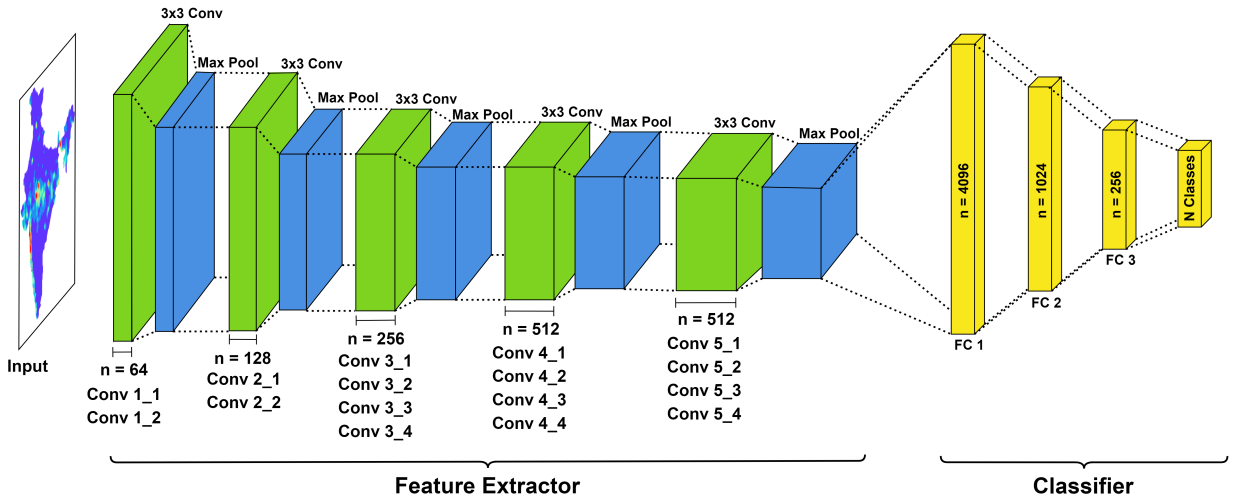


Figure 4.6: VGG Network

4.3.2 Generator: Pre-training

The generator network is initialized by pre training it with a MSE loss function. This allows the generator to avoid undesired local optima during adversarial training.

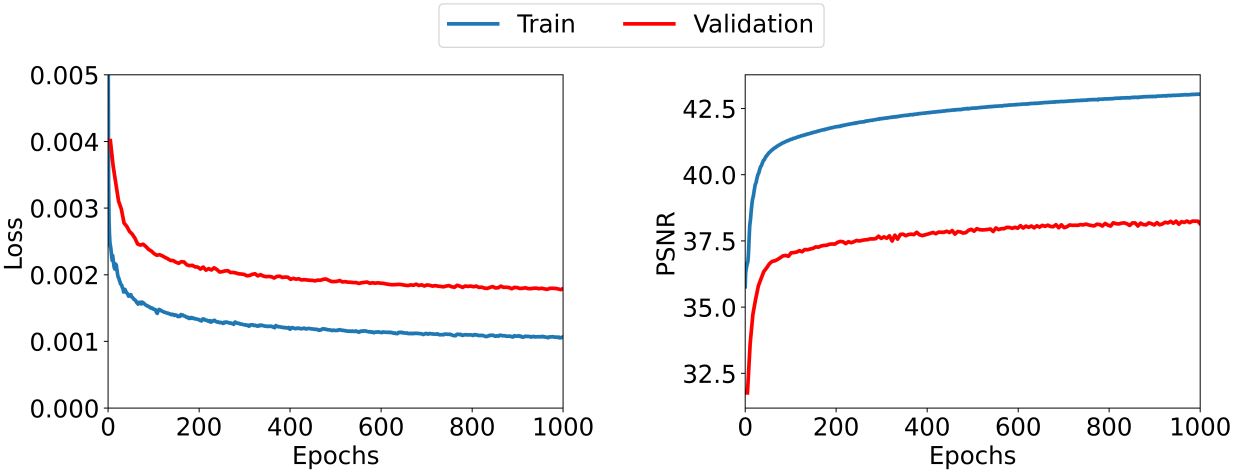


Figure 4.7: Generator pretraining learning curves

4.3.3 SRGAN: Adversarial Training

Finally, the pretrained generator (Section: 4.3.2) is trained alongside the discriminator network in adversarial setting. The discriminator learns to distinguish the generated output (SR) from the ground truth (HR). Perceptual

loss is computed from the VGG feedback (Section: 4.3.1) and the discriminator feedback. The perceptual loss is backpropagated to the generator allowing it to generate finer texture details.

As usually observed when training GANs, the learning curves (Figure: 4.8) show many fluctuations while training. When training in an adversarial setting, both the networks try to outperform one another, which leads to such behaviour.

Note: Here adversarial training does not refer to using adversarial examples for training. It simply means that we are training with an adversarial loss function (Equation: 4.4).

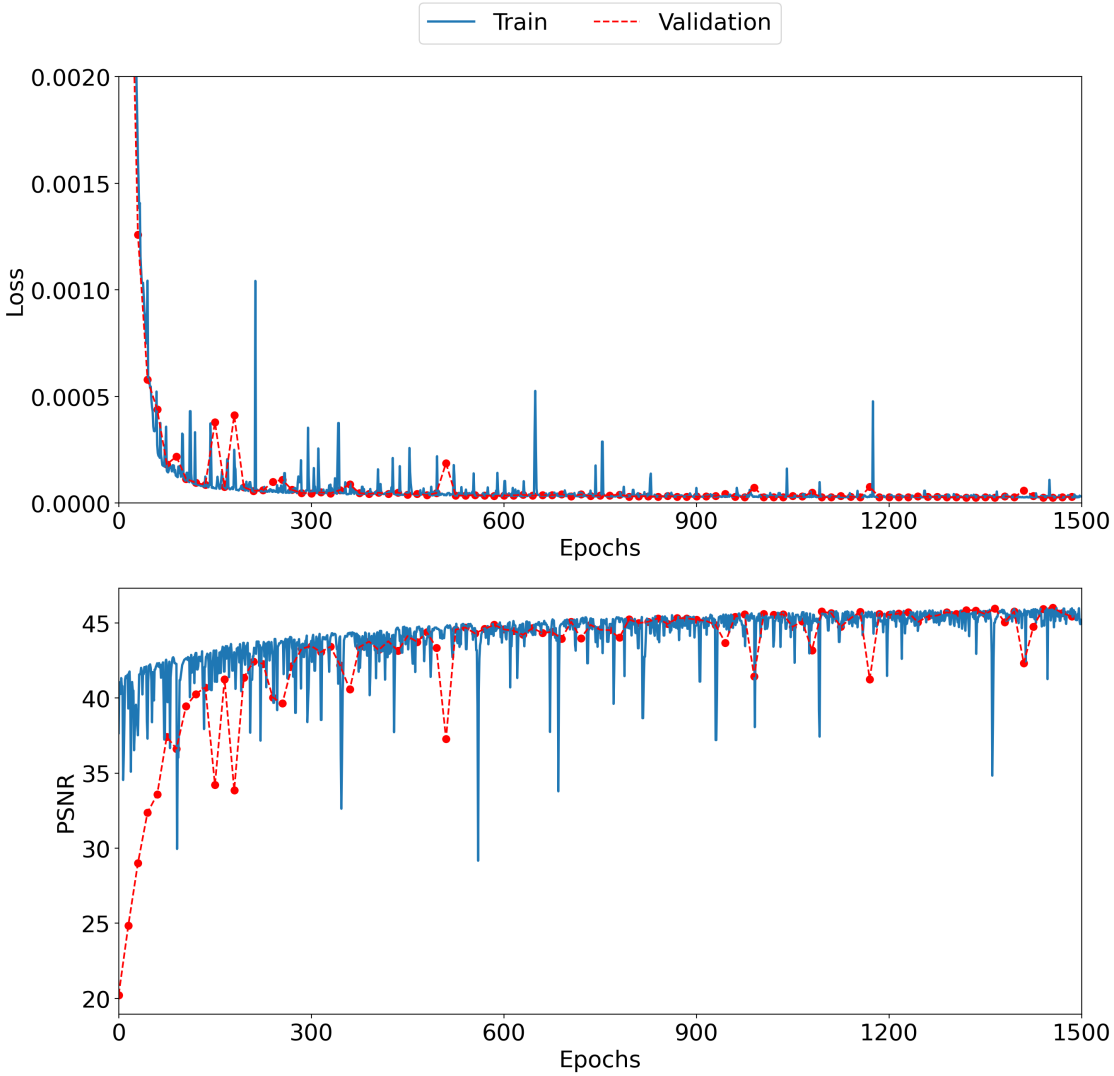


Figure 4.8: Adversarial training learning curves

The training details for Section: 4.3.1, 4.3.2 and 4.3.3 are summarized in **Table: 4.2**

Table 4.2: **Training Details**

Model	**Optimizer**	**Loss function**	**Batch Size**	**Epochs**
VGG	Adam (lr=10^{-2})	Cross Entropy Loss	512	100
Generator only	Adam (lr=10^{-4})	Mean squared error	512	1000
Both generator and discriminator	Adam (lr=10^{-5})	Perceptual Loss	256	1500

Computational Resources

All the models were trained on NVIDIA A100 40GB GPUs available at the PRATYUSH HPC facility at IITM Pune. A100 GPUs use tensor cores that enable mixed-precision computing offering massive speedups. We did all the code development and computing on the Apollo server of the PRATYUSH supercomputing facility.

Chapter 5

Ablation Study

The ablation study aims to study the significance of various components of a DL model. We experiment by adding, removing or modifying the parts of the model to check how it affects the model's performance. It is a handy way to comprehend the causality of any architecture. It also helps to identify the redundant parts of the model architecture.

We have conducted mainly three sets of experiments in the ablation study. These are discussed in detail in the following sections:

5.1 Residual Blocks

The residual blocks are an essential component of the generator model. The original SRGAN work uses 16 residual blocks in the generator. We experiment with the number of residual blocks to study the effect on the model's performance.

We conducted three experiments where the number of residual blocks is:

1. Reduced by a factor of two (i.e. 8),
2. Increased by a factor of two (i.e. 32),
3. Kept constant (i.e. 16)

We monitor which configuration works best for our use case.

We infer from the Figure 5.1 that using eight residual blocks yields the best results for our use case. A probable reason for this can be that even after

using skip connections, the features get diminished in deeper layers. Also, as our data is not very big, using a very deep network does not suit our task. Another benefit of using eight residual blocks is faster training and faster convergence.

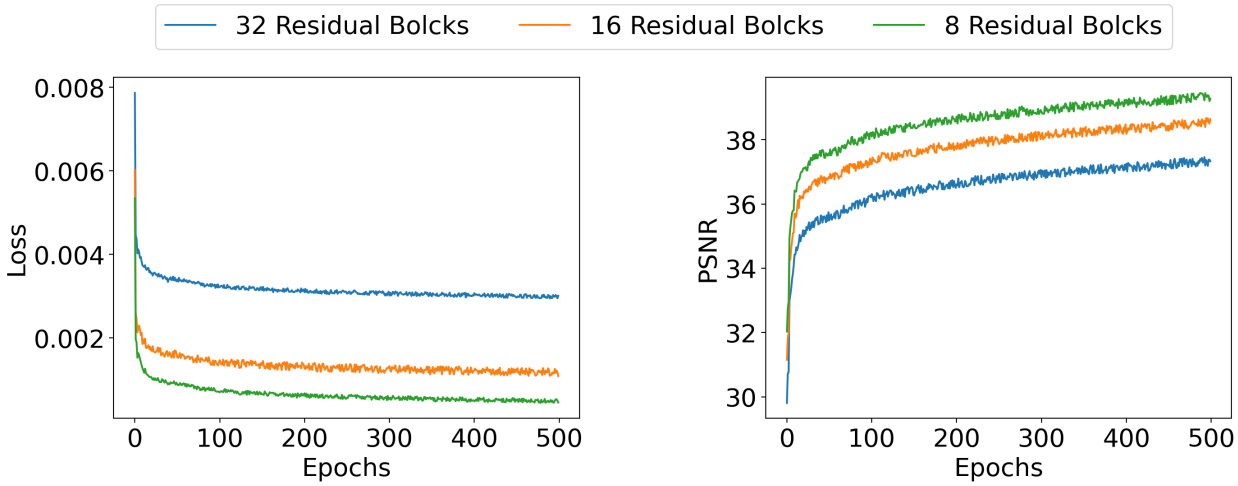


Figure 5.1: Learning curves for different number of residual blocks

5.2 Topology

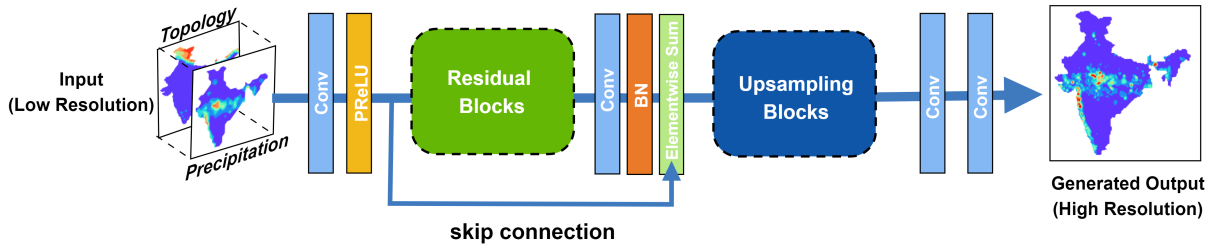
Related works like DeepSD, ConvLSTM and UNet have shown that augmenting topology with precipitation yields better results. Topology is a fundamental physical variable that determines the precipitation value at any location. All these methods interpolate the LR precipitation maps to match the size of the HR map and augment the HR elevation map in the input.

However, in SRGAN, LR input and the HR output size are different. Thus the topology map we augment to the input is in low resolution (Figure: 5.2a). We explored various ways to incorporate topology in high resolution that can potentially facilitate the model performance. We devised using a skip connection to concatenate the HR topology map just after the upsampling block (Figure: 5.2b). Since the feature resolution after upsampling blocks matches the HR output size, we can use HR topology here.

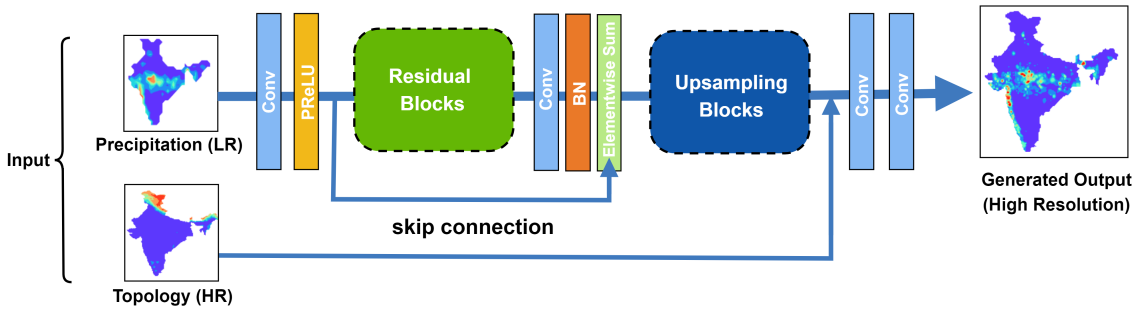
We trained the model with both these configurations and monitored the performance.

The topology ablation study suggests that concatenating the HR elevation

map after the upsampling blocks is beneficial (Figure: 5.3). The model performance improves significantly on using the HR elevation configuration. Our suggested scheme to augment the HR elevation maps works, and it is apt to use for this particular use case.



(a) Configuration 1: LR topology map is augmented with LR input.



(b) Configuration 2: HR topology map concatenated using a skip connection.

Figure 5.2: SRGAN architecture variants for augmenting elevation map.

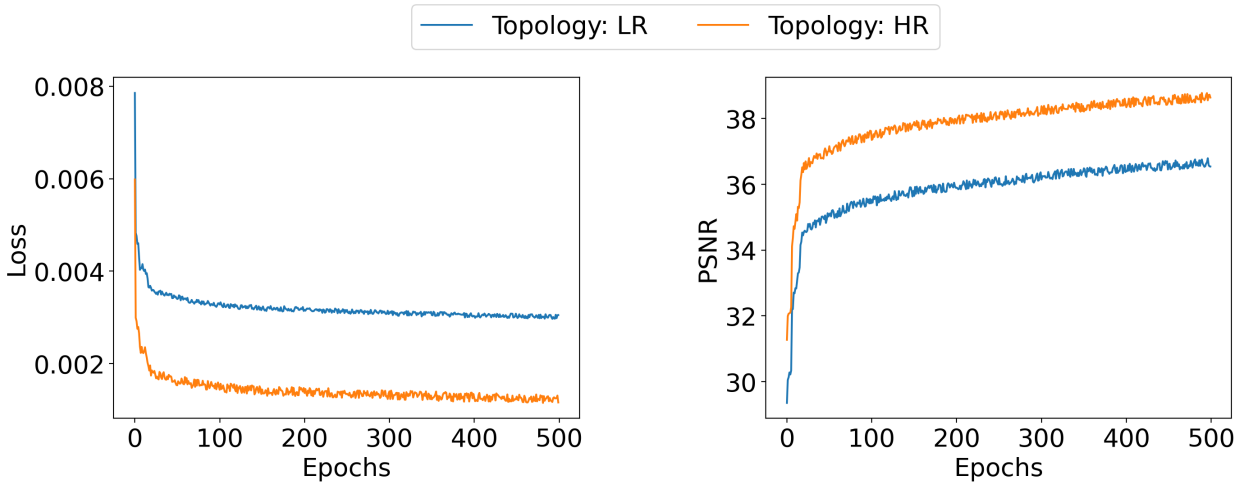


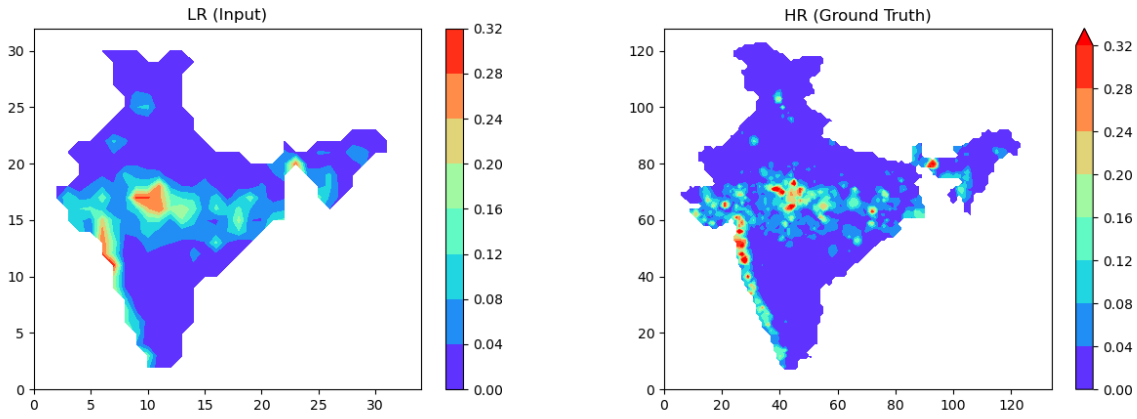
Figure 5.3: Learning curves for different elevation augmenting configurations

Chapter 6

Results and Discussion

6.1 Validation with HR Data

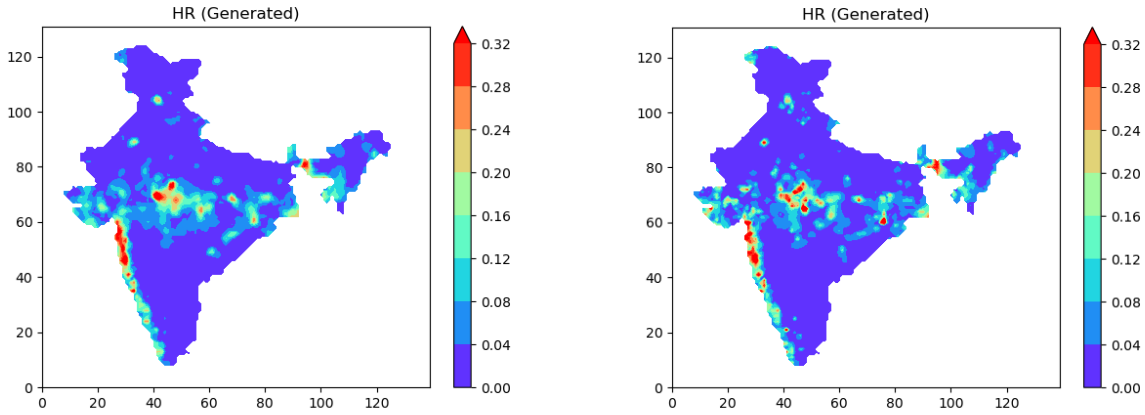
We trained the SRGAN model on the IMD precipitation data to achieve 4x resolution enhancement, i.e. $1.0° \rightarrow 0.25°$. On visual inspection of 2D images of rainfall amount on various days, we conclude that adversarial training enables SRGAN to recover finer details. The illustration plots show that the results from MSE training appear smooth compared to results from adversarial training. Our custom pre-trained VGG feature extractor has led the SRGAN model to generate and recover higher frequency details.



(a) LR: 1.0° input

(b) HR: 0.25° Ground Truth

Figure 6.1: Validation data



(a) Generated HR: 0.25° (Generator pretraining) (b) Generated HR: 0.25° (Adversarial training)

Figure 6.2: Downscaling results from SRGAN

Next, we compare SRGAN with other DL methods used for downscaling. We reproduced DeepSD, Conv LSTM, and DCN UNet by rigorously following the details given in the respective papers and shared code. We trained these models on IMD data for 4x resolution enhancement and compared the results using correlation as a metric. Figure 6.3 illustrates that SRGAN performs significantly better than other methods in terms of correlation.

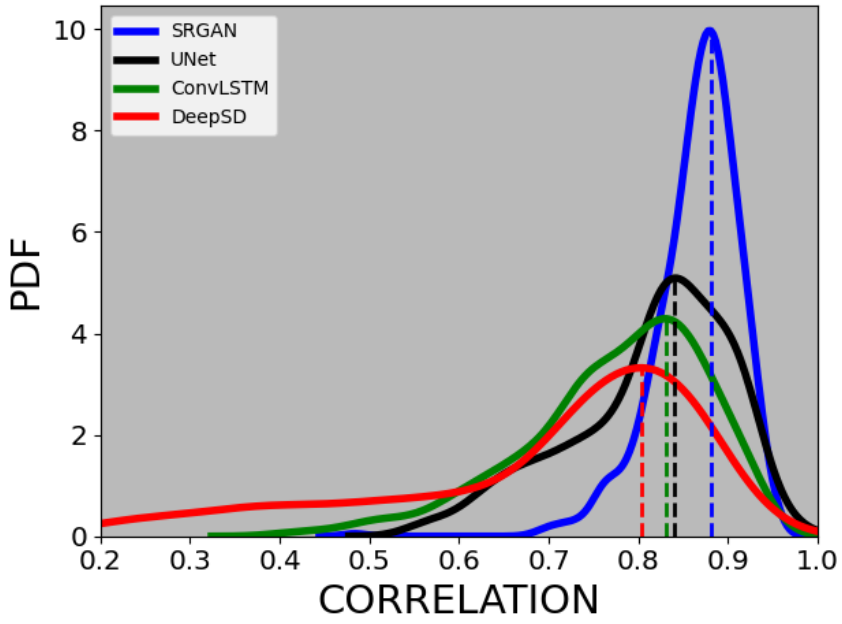
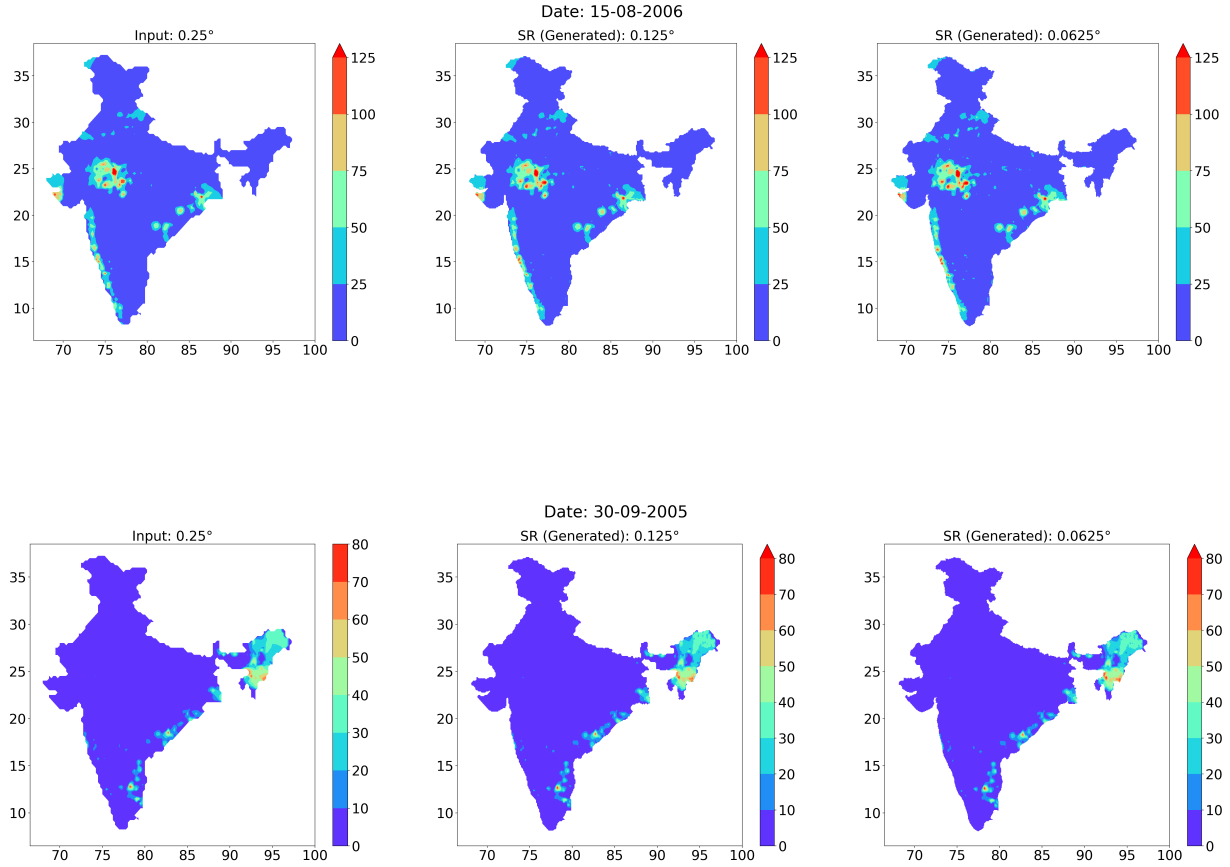


Figure 6.3: Comparisons of Correlation

6.2 Validation with Station Data

Next, we validate the SRGAN results with the station data from IMD. We use SRGAN to downscale 0.25° data to 0.125° (2x enhancement) and 0.0625° (4x enhancement). We perform validation for both resolutions by interpolating the downscaled data to the station locations and comparing the MSE and correlation.



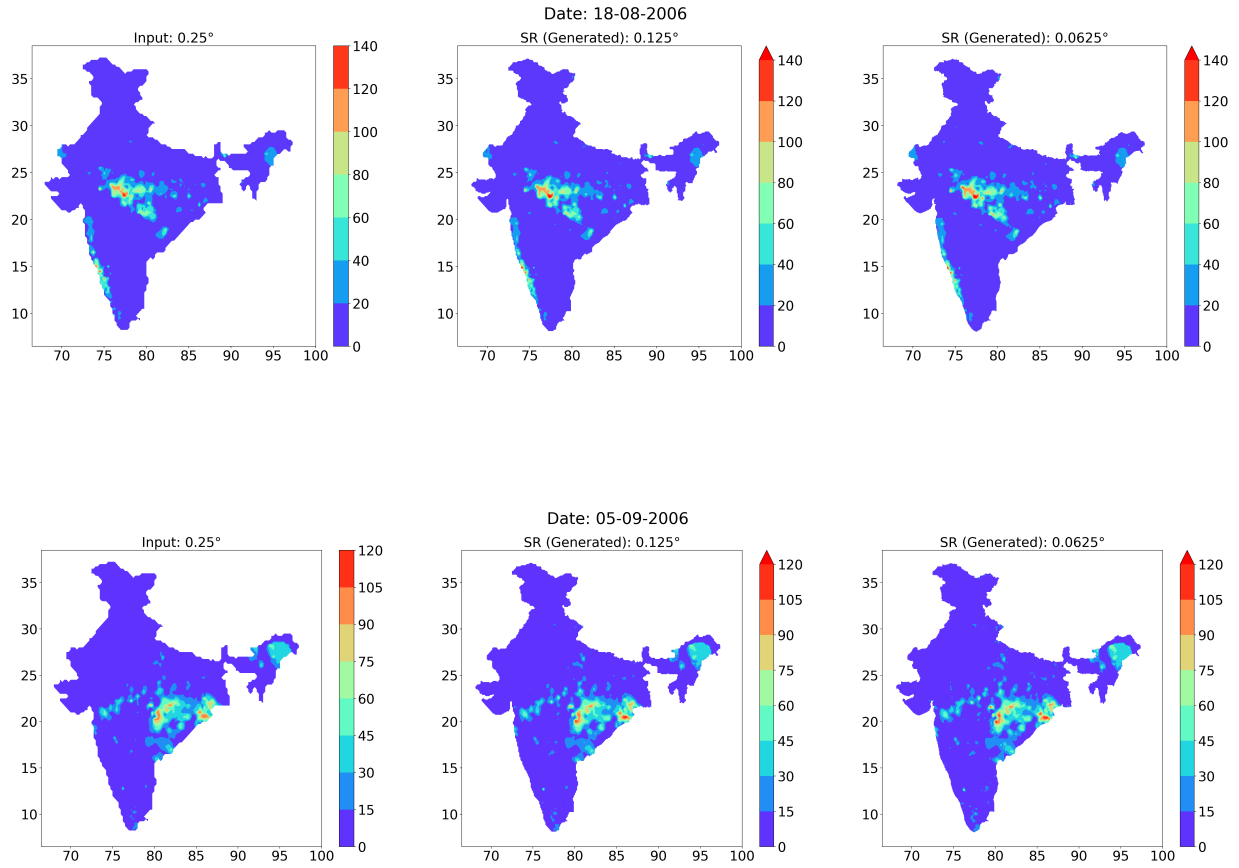
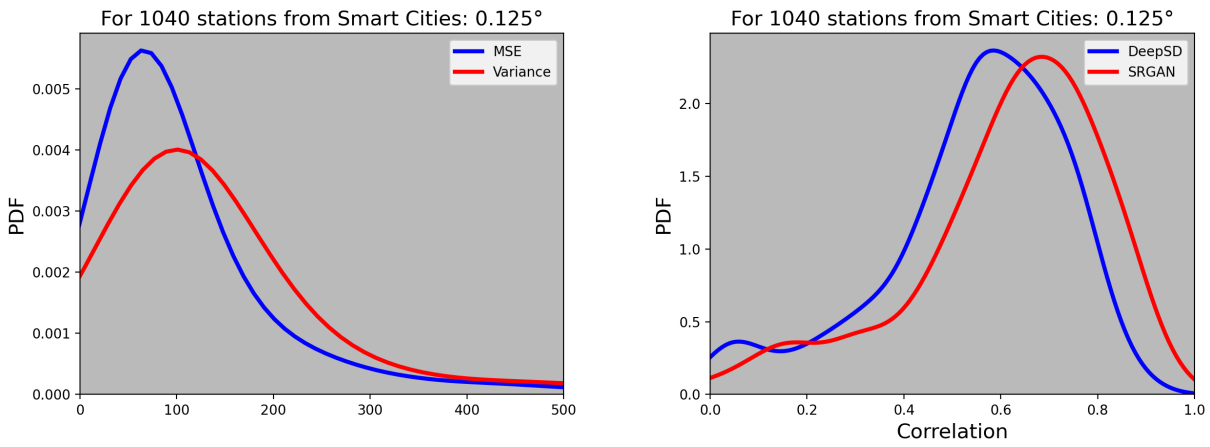


Figure 6.4: SRGAN results: 0.125° and 0.0625° downscaling

We assessed the results with IMD stations located in various smart cities in India. Out of the 100 smart cities in India, around 78 cities had an IMD station nearby. We were able to locate around 1040 stations nearby these cities. We use the Kriging interpolation method to interpolate the downscaled gridded data to the station location.

We compare the obtained MSE (between station data and interpolated downscaled data) with the precipitation variance at these stations. Similarly, we also compare the MSE and variance for individual smart cities from all geographical regions in India. The results are illustrated in sections 6.2.1 and 6.2.2 for both 0.125° and 0.0625° resolutions, respectively.

6.2.1 0.125 Degree Data



(a) MSE and Variance

(b) Correlation

Figure 6.5: Station validation results for 0.125°

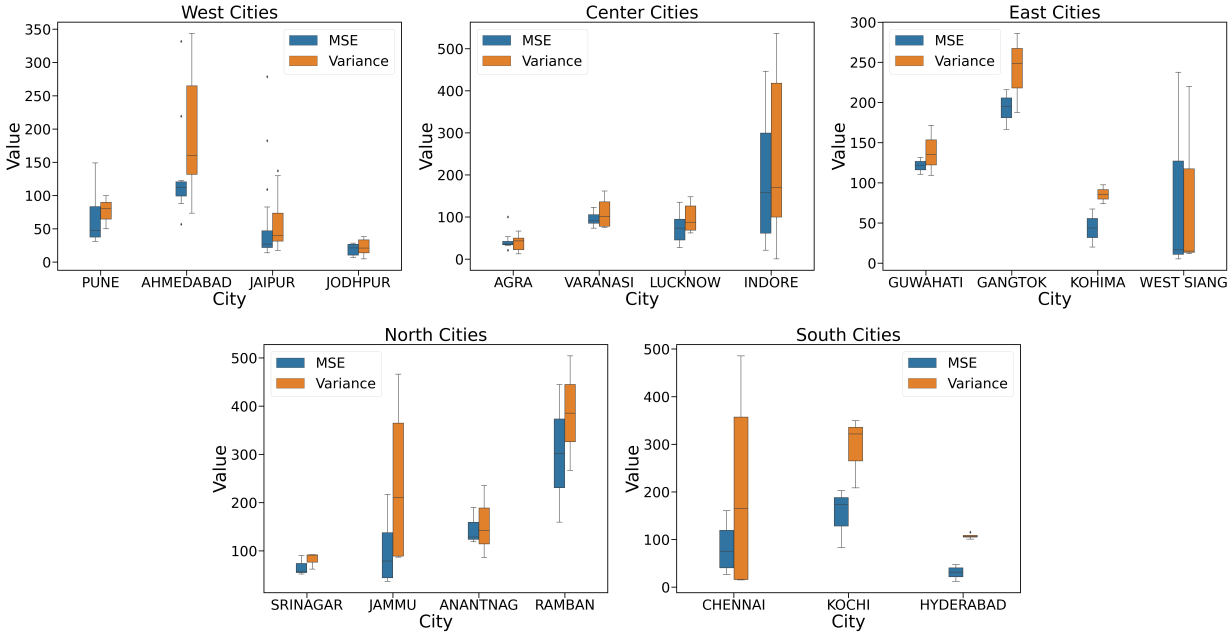
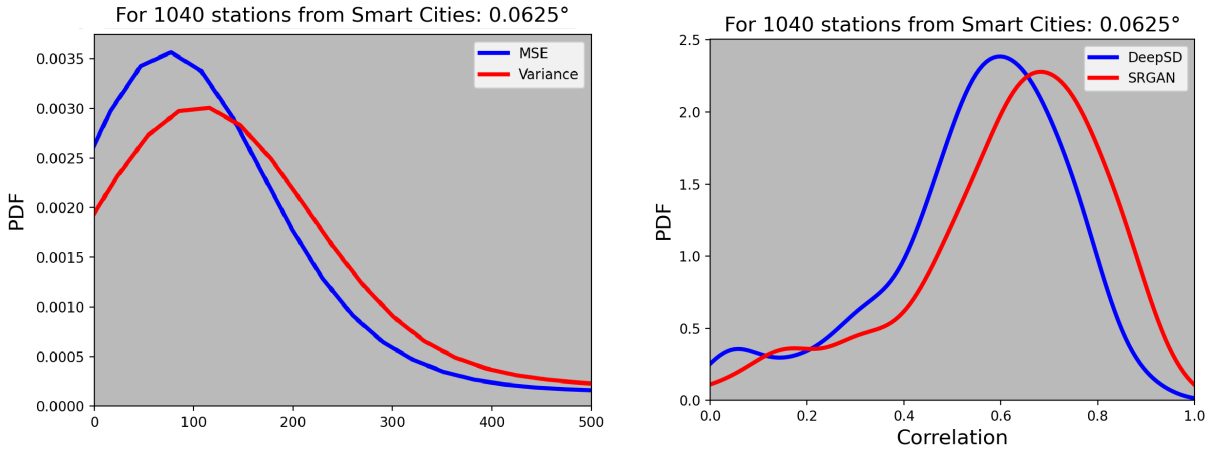


Figure 6.6: Validation for individual smart cities (0.125°)

6.2.2 0.0625 Degree Data



(a) MSE and Variance

(b) Correlation

Figure 6.7: Station validation results for 0.0625°

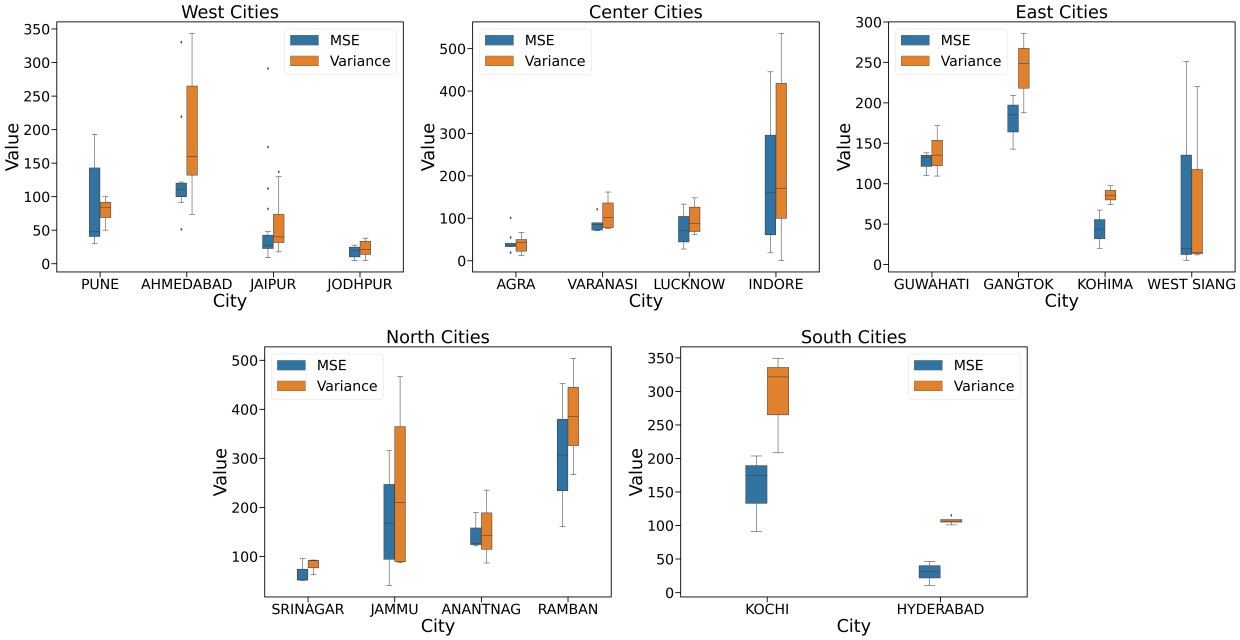


Figure 6.8: Validation for individual smart cities (0.0625°)

6.3 Validation of artefacts

We observed instances where some values have emerged in the downscaled data that were initially not present in the 0.25° ground truth data at that

location. It becomes necessary to validate such values to analyse whether our model generates undesired false positives.

For the 0.125° downscaled data, we select locations where such values appear. Next, we locate the IMD stations nearby these sites in the 12km search radius. We check if these stations recorded any rainfall on the days when such values appear. We also compare these locations with the closest 0.25° grid point.

The artefacts points found for a particular day are illustrated in Figure: 6.9a. The exact location of these points considered for validation are shown in Figure: 6.9c, 6.9d, 6.9e. We found that the nearby IMD stations indeed recorded rainfall for that day (Figure: 6.9b). We can conclude that the values appearing in the 0.125° downscaled data from SRGAN are not false positives. Due to the grid size, these values get omitted in the 0.25° data. SRGAN is able to estimate and recover such values.

We can do a similar analysis for the 0.0625 downscaled data also.

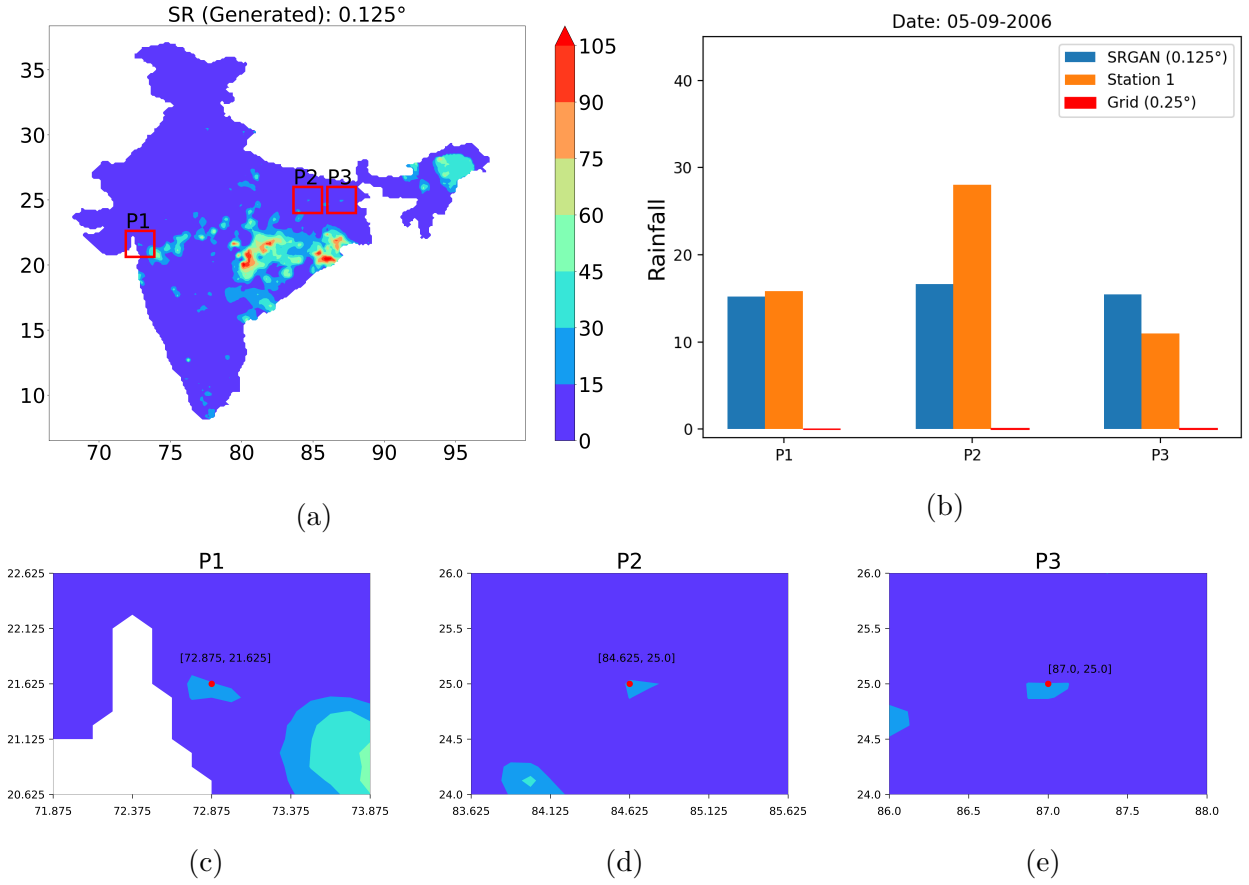


Figure 6.9: Validation of artefacts for 0.125° downscaled data

Chapter 7

Conclusion

This work applied a deep learning model called SRGAN for downscaling precipitation data over the Indian region. SRGAN is a generative adversarial network that uses a novel perceptual loss to recover finer details for large scaling factors. SRGAN addresses the smooth textures problem faced by other MSE based models. We have compared the SRGAN model with other DL models for the 4x downscaling task and found that SRGAN yields better results. We also explore the VGG network as a feature extractor for meteorological rainfall data. This custom trained feature extractor can serve as a backbone for other DL models that uses meteorological data.

We also explored the Kriging method for both station-to-grid and grid-to-station interpolation. Kriging has statistical nature, allowing it to capture the spatial correlation reasonably better than IDW and Shephard's method.

We also performed an ablation study to understand the behaviour of our model. This study revealed the model configurations that best suit our task.

Next, we have validated the 0.125 and 0.0625 generated results of SRGAN with the station data. The evaluation indicates that SRGAN generates significant results close to the actual station data. We also compare the prediction MSE and station variance for various smart cities in India. Finally, we investigated the artefacts emerging in downscaled data to check for false positives in our results. On evaluating such points, we found rainfall recorded by the nearby stations, rejecting the premise of false positives.

We conclude that SRGAN is a robust deep learning model for the downscaling task. Different checks suggest that it gives better and more reliable results.

7.1 Future Work

- Our study uses old IMD data from around $\sim$ 6000 ground stations. IMD has now released new data with about $\sim$ 8000 stations. We can use this data to improve the SRGAN downscaling performance. Additionally, the new data will allow for more rigorous validation.
- We noticed that the downscaling results improved using topology as an additional input parameter with precipitation data. We can augment other physical variables like temperature, pressure, wind, etc., to improve the skill of this work.
- This study focuses on using gridded station data for data downscaling. Moving forward, we can use outputs from Regional Climate Models (RCMs) to refine and evaluate the performance of the SRGAN further.
- We can use Kriging to interpolate other physical variables recorded at ground stations. This allows us to prepare gridded datasets for different variables required to augment during downscaling.
- We can also use deep learning based variants of kriging for the station-to-grid and grid-to-station interpolation.

Bibliography

[1] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[2] Thomas Vandal, Evan Kodra, Sangram Ganguly, Andrew Michaelis, Ramakrishna Nemani, and Auroop R Ganguly. Deepsd: Generating high resolution climate change projections through single image super-resolution. In *Proceedings of the 23rd acm sigkdd international conference on knowledge discovery and data mining*, pages 1663–1672, 2017.

[3] Bipin Kumar, Rajib Chattopadhyay, Manmeet Singh, Niraj Chaudhari, Karthik Kodari, and Amit Barve. Deep learning–based downscaling of summer monsoon rainfall data over indian region. *Theoretical and Applied Climatology*, 143(3):1145–1156, 2021.

[4] Nidhin Harilal, Mayank Singh, and Udit Bhatia. Augmented convolutional lstms for generation of high-resolution climate change projections. *IEEE Access*, 9:25208–25218, 2021.

[5] Agon Serifi, Tobias Günther, and Nikolina Ban. Spatio-temporal downscaling of climate data using convolutional and error-predicting neural networks. *Frontiers in Climate*, 3:26, 2021.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[7] V. Bok and J. Langr. *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning, 2019.

[8] M Rajeevan, Jyoti Bhate, JD Kale, and B Lal. High resolution daily gridded rainfall data for the indian region: Analysis of break and active monsoon spells. *Current Science*, pages 296–306, 2006.

[9] Madhaven Rajeevan, Jyoti Bhate, and Ashok K Jaswal. Analysis of variability and trends of extreme rainfall events over india using 104 years of gridded daily rainfall data. *Geophysical research letters*, 35(18), 2008.

[10] DS Pai, M Rajeevan, OP Sreejith, B Mukhopadhyay, and NS Satbha. Development of a new high spatial resolution (0.25× 0.25) long period (1901-2010) daily gridded rainfall data set over india and its comparison with existing data sets over the region. *Mausam*, 65(1):1–18, 2014.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[12] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan 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*, pages 4681–4690, 2017.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.