Image Dehazing

A major project report submitted in partial fulfilment of the requirement for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering / Information Technology

Submitted by

Ayush Vaish (201253)

Vibhav Ahuja (201215)

Under the guidance & supervision of

Dr. Nancy Singla

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

CERTIFICATE

I hereby declare that the work presented in this report entitled **"Image Dehazing"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** submitted in the department of **Computer Science & Engineering and Information Technology**, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Nancy Singla**, Assistant Professor (SG), Department of Computer Science and Engineering and Information Technology. The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Ayush Vaish (201253) **Vibhav Ahuja (201215)**

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Nancy Singla
Assistant Professor (SG)
Department of CSE & IT
Dated:

CANDIDATE'S DECLARATION

We hereby declare that the work presented in this report entitled **'Image Dehazing** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Nancy Singla** (Assistant Professor, Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Ayush Vaish
Roll No.: 201253

Vibhav Ahuja
Roll No.: 201215

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Supervisor Name: Dr. Nancy Singla
Designation: Assistant Professor (SG)
Department: Computer Science and Engineering/Information Technology
Dated:

ACKNOWLEDGEMENT

First, I express my heartiest thanks and gratefulness to the Lord for His divine blessing to make it possible to complete the project work successfully.

We owe our profound gratitude and indebtedness to our project supervisor **Dr. Nancy Singla**, who took keen interest and guided us all along in our project work titled **—Image Dehazing**, till the completion of our project by providing all the necessary information for developing the project. The project development helped us in research, and we got to know a lot of new things in our domain. We are really thankful to her.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, educating and non-instructing, who have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Ayush Vaish
(201253)

Vibhav Ahuja
(201215)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
CNN	Convolutional Neural Network
NYUv2	NYU-Depth V2
RESIDE	Realistic Single Image DEhazing
SVM	Support Vector Machine
JPEG	Joint Photographic Experts Group
IDE	Integrated Development Environment
RAM	Random Access Memory

ABSTRACT

Haze causes a loss of contrast, colour saturation, and overall sharpness by giving images a milky or fog-like appearance. It is possible for distant objects to appear faded and less distinct, and the image's overall visual quality may be severely diminished. Haze is a frequent problem in photography and computer vision, and it can be brought on by a variety of things, including atmospheric conditions (such fog or mist), pollution, and even the scattering of natural light.

Haze and air scattering have a major negative impact on image quality, visibility, and a number of computer vision applications, including scene analysis, object recognition, and object detection in outdoor photographs. Due to the complicated and non-linear haze creation and distribution, traditional image dehazing techniques frequently fail to remove haze and restore crisp details. A powerful and effective image dehazing approach that can precisely retrieve the real scene content from hazy photographs utilizing cutting-edge deep learning methodologies is urgently needed.

The main goal of this project is to create and put into use a residual-based deep CNN architecture that can recognize and take advantage of intricate correlations between hazy and clear images.

CHAPTER – 1 INTRODUCTION

1.1 INTRODUCTION

Haze is created when a lot of atmospheric particles or water droplets scatter or absorb light that passes through them. In addition to having severe colour attenuation low contrast and saturation and poor visual effect, images taken in haze and other weather conditions also have an impact on other weather condition also have an impact on other system that depends on optical imaging equipment, including target recognition and outdoor monitoring systems, satellite remote sensing systems and aerial photography systems. It presents several challenges for the goal of the research. Therefore, there is a pressing and practical need for efficient dehazing and sharpness recovery in order to enhance the quality of haze-degraded images and lessen the influence of haze and other meteorological circumstances on outdoor imaging systems.

Currently, image processing-based enhancement and physical model-based restoration are the two main categories of approaches for processing foggy images. The image processing-based enhancement method does not take into account the precise reason for the image degradation; instead, it starts with the image itself. To accomplish the goal of clarity the visual impact of the image is enhanced by raising its contrast and brightness. These procedures are typically well-established and effective and the processed data can satisfy the clarity criteria as well. Such approaches however are not scene or image-adaptive. The Image that has more variations in scene depth in particular is not successful. Moreover, the approach relies on picture enhancement rather than taking into account the fog quality reduction procedure which makes it unable to significantly boost image definition. There is more substantial distortion of the outcome and it is unable to remove the fog to restore the original appearance. Furthermore, unsuitable for further processing the produced images has a low visual impact.

A damaged model of fog-degraded photos is established by the physical model-based restoration method which examines the particular reasons behind image degradation. Techniques for processing images based on physical model have seen tremendous advancements in recent years. Fog image modelling is a popular use of this method which describes the fog image as the superposition of scene radiation and scattering effects. The physical model can be described as follows:

$$I(x) = J(x).t(x) + A[1 - t(x)] \quad \text{...eq(1)}$$

$$t(x) = e^{-\beta d(x)} \quad \text{...eq(2)}$$

Where x is the input image pixel point. The initial hazy picture input in denoted by I(x). J(x) represents the restored and dehazed image. A is the value of ambient atmosphere light. The distance between the object and the camera is represented by d(x) and t(x) is the optical route propagation map. T(x) shows exponential attenuation with the images depth d(x). The `scattering capacity of light per unit volume of the atmosphere is represented by the atmospheric scattering coefficient, β which typically taken a smaller constant.

1.2 PROBLEM STATEMENT

With its slight yet profound presence haze casts a noticeable shadow over the visual appeal of images changing their nature in a subtle yet significant way. This atmospheric phenomenon steals scenes of their natural brightness and clarity acting as a silent attacker. Three negative impacts of haze are apparent: a noticeable loss of contrast a subdued palette lacking in saturation and an overall blurring that softens the edges of once-sharp visual object. The resulting vision has an odd aspect to it, like looking through a misted veil where clarity gives way to a milky opaqueness.

Haze has consequence even in the world of faraway views, as object become blurry blends into one another making them appear fading and undefined. This tendency is made worse by meteorological circumstances like fog or mist, which throw off the observes ability to precisely distinguish feature and produce a visually chaotic scene. Airborne particles add to the haze and detract from the picturesque splendour that the lens captures so pollution also play a part in this visual compromise.

Natural light dispersion typically adds haze to the mix of the light and atmosphere making it difficult to sharp picture. In photography and computer vision, haze is a persistent challenge that demands careful approaches to maintain visual integrity. The hidden beauty beneath the ethereal veil of haze must be revealed by photographers and technologists navigating atmospheric challenges with a delicate mix of techniques and technologies.

1.3 OBJECTIVES

The primary objective of our project is to eliminate or minimize the haze which is a result of particles in the atmosphere in order to enhance the contrast and clarity and also to develop and validate an advanced image and video dehazing algorithm using deep convolutional neural network (CNN) with a residual based architecture. The key goals are as follows: -

- Development of Image and video dehazing model
- Implement a residual network in the second phase to leverage the ratio of foggy images or video frames and the previously estimated transmission map for efficient removal of haze
- **Transmission Map estimation: -** We will be developing a mechanism within our network which will be accurately estimating the transmission map from hazy images or video frames in the initial phase of our project's dehazing process.
- **Real world applicability: -** We will also try to assess the algorithm's potential for real-world applications where the atmospheric conditions pose challenges to visual perception.
- **Residual based dehazing: -** We will also be implementing a residual network in the second phase of our project to leverage the ratio of foggy images or video frames and the previously estimated transmission map for efficient removal of haze
- **Haze Removal: -** Our general objective is to improve the visibility of objects and scenes by reducing or removing haze to enhance the overall robustness.
- **Adaptability: -** Our dehazing algorithm should work well and be able to handle different lighting scenarios as well as to adjust the varying air conditions and haze levels.
- **Testing and validating the model: -** The testing and validation of the model is an essential stage of our project. So, by contrasting the model predictions with a set of test data we can easily evaluate our model's performance. We will be utilizing the NYU2 depth dataset for training the deep neural network and evaluate mode's performance via various metrics.

1.4 SIGNIFICANCE AND MOTIVATION OF THE PROJECT WORK

In our visual world, this project aims to redefine perception by addressing the challenge of haze affecting image and video clarity. It seeks to innovate and enhance technological capabilities to overcome atmospheric obstacles that hinders our ability to interpret visual data. The project is motivated by the human desire for clearer site, whether in a literal or metaphorical terms, and also aims to remove visual hindrances caused by weather conditions. The proposed to stage network model, featuring a deep, Convolutional neural network (CNN) architecture, represents a significant advancement in deep learning techniques. It challenges, conventional methods and aims to revolutionize image and video dehazing, pushing the boundaries of what can be achieved through human ingenuity and computational power. The strategic utilization of the NYU2 depth dataset is deliberate decision to ground the project in real-world data, ensuring that the solutions developed are applicable beyond controlled environments.

This dataset serves as a connection between theoretical brilliance and practical implementation. Beyond algorithms and data sets. This project also holds the potential for societal impact. It envisions a world where clearer satellite imagery aids, disaster response efforts and where autonomous vehicles navigate an obstructed landscape.

1.5 ORGANIZATION OF PROJECT REPORT

In the beginning, the project report unfolds as an organized story including the complexities of our exploration into image and video dehazing. In our first chapter we are focusing on introduction, problem statement, objectives and the motivation of our project work. We unveil the architecture of our deep convolutional neural network (CNN), with a residual based design, guiding us to explore the computational dehazing. We will be highlighting two crucial phases- transmission map estimation and residual based dehazing. With the methodology we have clarity on where our project will go during the whole process and how our project will work during the process. The NYU2 and reside dataset helps our algorithm to authenticate visual data. The experimental results evaluated through various metrics will mark our progress. As our project will go further, the comparison between chapters will provide a moment to showcase not only the brilliance of our approach, but also its superiority in handling atmospheric conditions. In the appendix we will be adding technical details such as code snippets, parameter configurations and any information which may aid fellow seekers in replicating and building upon our expedition. While crafting this report, we will be navigating not just through lines of code and results but also through aspirations and challenges that define our human pursuit for a better and a clearer vision which our naked eyes unable to see. Each chapter will be contributing towards the successful completion of our project on image and video dehazing using advance deep learning algorithms.

CHAPTER – 2 LITERATURE SURVEY

2.1 OVERVIEW OF RELEVANT LITERATURE

Several significant studies have highlighted recent developments in single-image dehazing. One such contribution is the Lightweight LD-Net design, which was described in [1]. This design not only shows adaptability beyond dehazing but also investigates its limitations in consistently retrieving cloudy images and color information. On a similar note, Ullah et al [2] presented groundbreaking approaches such as the Multi-level Fusion Module (MFM) and Residual Mixed-Convolution Attention Module, which outperform existing algorithms using datasets such as RESIDE.

Furthermore, Chaitanya and Snehasis [3] proposed a solution that is effective for both indoor and outdoor hazy photographs. However, it faces difficulties in reliably detecting the true hue of ground truth photographs in locations with high cloud thickness. In contrast, Li et al [4] provided with a semi-supervised method for effectively learning the domain gap between synthetic data and real-world images. However, its efficacy decreases under high-haze settings.

Furthermore, X. Li [5] and Guo, Fu et al [6] demonstrated the effectiveness of Deep Residual Learning (DRL) Networks. While these methods outperform others on the NYU-Depth V2 dataset, they are significantly slower than the AOD approach. These collaborative initiatives highlight the changing environment of single picture dehazing approaches, with each bringing new insights and breakthroughs to the field. Below is the literature survey for all the research papers studied to implement this project.

Table 1. Literature Review Table

S. No	Paper Title [Cite]	Journal (Year)	Tools/Techniques/Dataset	Results	Limitation
1.	Light-DehazeNet : A Novel Lightweigh	Journal (IEEE Transactions on Image	Lightweight LD-Net architecture.	The proposed model showing its	The performance is not up to the mark

	t CNN Architecture for Single Image Dehazing [1]	Processing (Volume: 30)) (2021)	NYU2, HSTS, OTS, SOTS, and HazeRD dataset	applicability not only for single image dehazing but also for other relevant applications.	since it is doing single reconstruction (hazy image and colour information reconstruction).
2.	Multi-Level Fusion and Attention-Guided CNN for Image Dehazing [2]	IEEE Transactions on Circuits and Systems for Video Technology (Volume: 31, Issue: 11, November 2021)	Multi-level Fusion Module (MFM), Residual Mixed-Convolution Attention Module RESIDE, and the real-world dataset.	The proposed two models have more advantage over the state-of-the-art methods for image dehazing.	
3.	Single image dehazing using improved cycleGAN [3]	Journal of Visual Communication and Image Representation, Volume 74, 2021	AOD Net Architecture and CycleGAN architecture NYU-Depth and reside beta datasets	The proposed method is robust for both indoor and outdoor hazy images and is able to preserve the minute texture information present in the images	Model cannot estimate the actual color of the ground truth Image if the thickness of the haze is very high.

				while dehazing them.	
4.	Semi-Supervised Image Dehazing [4]	IEEE Transactions on Image Processing (Volume: 29) (2019)	NYU Depth dataset RESIDE-C, HazeRD, and SOTS datasets	This shows that the proposed semi-supervised method is effective in learning the domain gap between synthetic data and real-world images, thus alleviating the over-fitting problems	The model performs less effective when the image suffers from severe haze
5.	Recursive Deep Residual Learning for Single Image Dehazing [5]	2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops	Deep Residual Learning (DRL) Network for Image Dehazing NYU-Depth V2 dataset	DRL outperforms all others methods by a large margin.	DRL method was only marginally slower than AOD(All-in-One Dehazing)
6.	A Cascaded Convolutional Neural	IEEE Access (Volume: 6) (2018)	Atmospheric Scattering Model and Cascaded CNN model NYU-V2 Depth dataset	The proposed method outperforms	The model is existing image artifacts and

	Network for Single Image Dehazing [6]			the state-of-the-art methods both on the synthetic and real-world hazy images.	noise because the training dataset is generated based on the atmospheric scattering model which does not take artifacts and noise into account
7.	Single Image Dehazing Using Ranking Convolutional Neural Network [7]	IEEE Transactions on Multimedia (Volume: 20, Issue: 6, June 2018)	Ranking-CNN Dataset 1) Dataset-Syn - Dense Haze 2) Dataset-Cap - Light Haze	Ranking CNN Model comes out to be more effective than classical CNN as it can capture the structural and statistical features simultaneously.	Efficiency of the Ranking CNN is less than other models.
8.	AOD-Net: All-in-One Dehazing	2017 IEEE International Conference	All-in-One Dehazing Network (AOD-Net). NYU-Depth V2 dataset	In all haze conditions (light,	

	Network [8]	on Computer Vision		medium or heavy), our AOD net model constantly improves detection, surpassing both naive Faster R-CNN and non-joint approaches	
9.	A Research on Single Image Dehazing Algorithms Based on Dark Channel Prior [9]	Journal of Computer and Communications > Vol.4 No.2, February 2016	Dark Channel Prior Single Image Dehazing Dataset was prepared by the authors	The proposed model achieves the highest SSIM value and hence attains a dehazing accuracy that brings the output image closest to the haze-free image.	Lack of a big dataset

2.2 KEY GAPS IN THE LITERATURE

As we further studied the research papers, we found that there were some key gaps in those papers which are as follows: -

- **Diversity in datasets: -** We found that many papers relied on a narrow or limited range of datasets selection such as NYU depth and Synthetic Objective Testing Set, which may not completely represent the diversity of real-world hazy scenarios. Many researchers expressed the need or requirement for more varied and realistic datasets to ensure the robustness and efficiency of dehazing algorithms across a wide range of conditions.
- **Discrepancy in Algorithm and Dataset: -** Several articles highlighted gaps in the size of datasets and the complexity of algorithms. In some algorithms, the model was not that much effective because of need of larger datasets and the dataset was smaller.
- **Inadequate quantitative evaluation: -** Some research papers failed to provide quantitative or qualitative results for their proposed algorithm. This gap raised the question about the reliability and effectiveness of the proposed techniques which they used in their model.
- **Bias in Training Data: -** In one or two papers, there was a higher chance of bias in the training data. Researchers were confused and concerned on biased training datasets, which may impact the generalization of models to real-world scenarios
- **Difficulty in handling hazy images: -** The paper on optimized contrast, enhancement mentioned some difficulties in restoring the densely easy images. They also expressed some limitations of certain algorithms in handling extreme hazy conditions. This raises some questions on robustness of existing techniques in some scenarios where the haze was particularly dense as compared to others.
- **Parameter Sensitivity: -** The real-time image and video design paper based on multiscale guided filter. Discussed the algorithms better results but also mentions parameter sensitivity. This raises questions on the reliability and ease of implementation of such algorithms.
- **Larger Datasets: -** Some algorithms work better on smaller datasets but some algorithms need larger datasets for better training of the data. Therefore, the need of a large and diverse dataset is crucial for better training and testing of the algorithm.
- **Training time concerns: -** Many research papers raised concerns about the higher training time. This is because of less GPU power and the model has to be trained perfectly on large datasets. Thus, takes a higher training time.

CHAPTER 3: SYSTEM DEVELOPMENT

3.1 REQUIREMENTS AND ANALYSIS

The requirements and analysis needed are mentioned below: -

FUNCTIONAL REQUIREMENTS

- Image upload and its processing
    i. The software we will be needing should be able to handle the uploaded images and also be able to save it for future processing.
    ii. We must be able to process the uploaded image into our model for extracting of necessary features and generate an output.
- Image Surrounding identification
    i. The model should be able to dehaze the image irrespective of the place it has been taken i.e., Outdoor or Indoor.
    ii. The model should be able to adjust according to the type of image and after analysing should be able to process and dehaze it and further generate output.
- Real-time processing - The system should be able to process the image and produce the output in minimum time, thus increasing its overall useability.
- Training mode - The system should be able to retrain the model with new data when available.

1. NON-FUNCTIONAL REQUIREMENTS

- Performance - The System should be able to dehaze the input image under 2 sec of time.
- Reliability - The system should attain a minimum possible model loss.
- Scalability - The model should be refined enough so that it can be scaled up to use in a website or android apps.
- Compatibility - The model should be compatible with the front-end frameworks in python like Flask, for the development of a website, where the user can upload the hazed image and will get the dehazed image from there.
- Maintainability - The codebase of the model and front-end should follow the industry best practices to increase readability and understandability.
- Training Dataset
    i. We have used two separate datasets for training and testing of the model.

ii. The dataset includes both indoor and outdoor images.

- Image resolution - Model supports for all the common types of the image formats like JPEG, PNG, JPG.
- Validation and testing - Testing on the testing dataset and producing the output.

2. HARDWARE REQUIREMENTS

- GPU(s): - High Performance GPUs are needed for efficient training like NVIDIA Maxwell GPU.
- CPU - A multicore-CPU is required so that it can efficiently train the model and handle its oppression like Quad-Core ARM Cortex-A57 Processor.
- RAM - A good amount of memory RAM is required such as 16GB LPDDR4.
- Storage - Adequate amount of storage is needed so that it can save all the necessary checkpoints while training of the model, and also to save some of the necessary files.

3. SOFTWARE REQUIREMENTS

- Language - Python is used throughout the project for the implementation.
- Deep Learning Frameworks - We have to choose suitable deep learning frameworks like TensorFlow and Keras for the training and testing of the models.
- Version Control - For the version control we will be using Git And we’ll be pushing the code into the Github.
- Development Environment - The development environment used for training and testing of the model is Jupyter notebook and vs code.

3.2 PROJECT DESIGN AND ARCHITECTURE

In this project we will we following the below project design where we will be first processing the dataset and after that training the model with network architecture given below in Fig 2

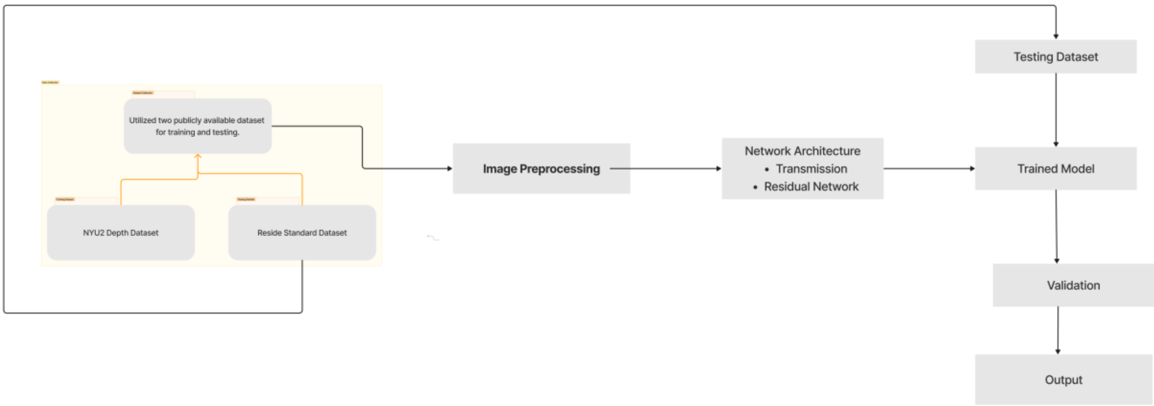

Fig:- 1 Project Design

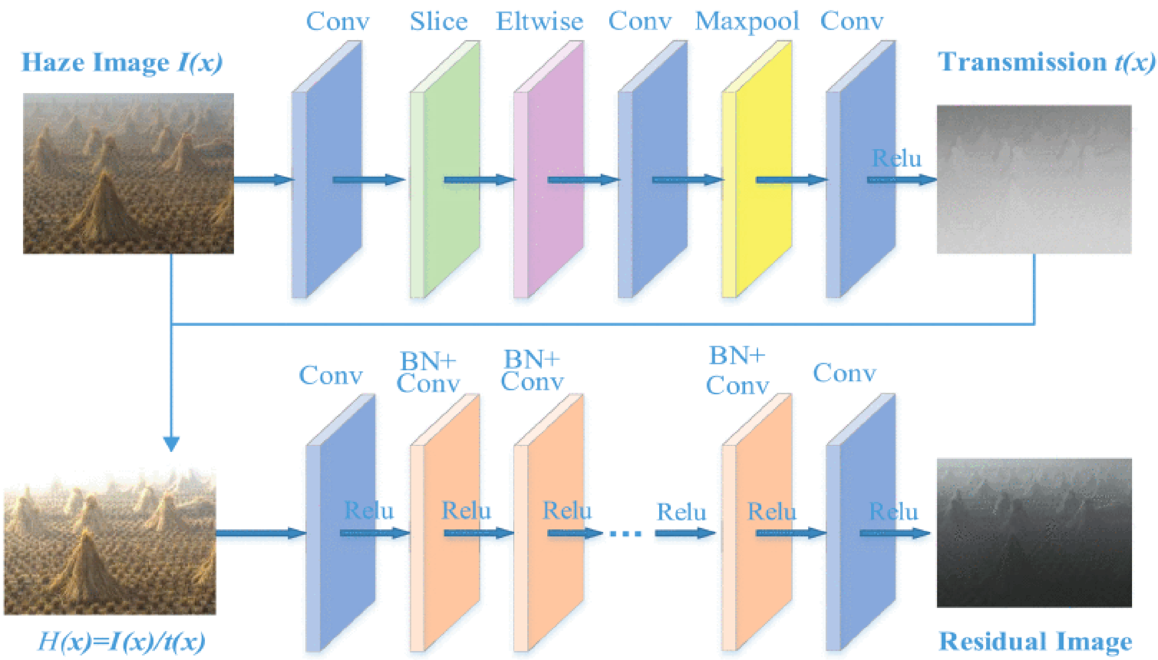

Fig:- 2 Network Architecture [5]

3.3 DATA PREPARATION

3.3.1 DATA COLLECTION

After surveying many existing similar models and research papers we have observed that many of them are using publicly available dataset – NYU2 Depth Dataset and Reside Standard Dataset.

We will be using separate datasets for training and testing of the model like: -
1) **NYU2 DEPTH DATASET - (TRAINING DATASET)**

- It comprised of video sequences from a variety of indoor scenes as recorded by the RGB and depth cameras
- It includes: -

Table 2. Training Dataset Description

	Number of Images
Densely Labelled Pairs of RGB and Depth Images	1449
New Scenes taken from 3 cities	464
New Unlabelled Frames	407024

2) **RESIDE (REALISTIC SINGLE IMAGE DEHAZING) – (TESTING DATASET)**

- It includes images with realistic scenes under the effect of haze.
- It includes both indoor and outdoor images.
- Each image in the dataset is accompanied by a corresponding truth image.
- This dataset is used by the researchers and developers to evaluate and contrast the performance of various dehazing techniques.
- It includes: -

Table 3. Testing Dataset Description

Subset	**Number of Images**
Synthetic Objective Training Set (SOTS)	500
Hybrid Subjective Testing Set (HSTS)	20

3.3.2 DATA PREPROCESSING

In order to train a better model and expect a good result and accuracy, we will be doing preprocessing on the dataset.

TRAINING DATASET

```python
import os, time, h5py, random, cv2
import numpy as np

import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow

from skimage.transform import resize
from sklearn.feature_extraction import image as skimg

from keras.models import Model
from keras.layers import Input, Activation, BatchNormalization, Conv2D, Conv3D
from keras.layers import Lambda, Concatenate, MaxPooling2D, Maximum, Add


import keras.backend as K
K.set_image_data_format('channels_last')
```

Fig. 3 Importing necessary libraries

In this snippet, we will be importing all the necessary libraries needed to load and process the data.

```python
def load_train_dataset(count = 20, patch_count = 10):
    dataset = 'D:/Major Projects/nyu_depth_v2_labeled.mat/nyu_depth_v2_labeled.mat'
    train_dataset = h5py.File(dataset, "r")

    trans_vals = [0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8]

    nyu_image_patches = None
    nyu_haze_patches = None
    nyu_random_transmission = []

    for i in range(count):
        image = train_dataset['images'][i]
        image = (image.transpose(2,1,0))/255.0
        patches = skimg.extract_patches_2d(image, (16, 16), max_patches=patch_count)
        if nyu_image_patches is not None:
            nyu_image_patches = np.concatenate((nyu_image_patches,patches))
        else:
            nyu_image_patches = patches

    for image in nyu_image_patches:
        transmission = random.choice(trans_vals)
        image = image*transmission+(1-transmission)
        nyu_random_transmission.append(transmission)
        if nyu_haze_patches is not None:
            nyu_haze_patches = np.concatenate((nyu_haze_patches, [image]))
        else:
            nyu_haze_patches = np.array([image])

    train_dataset.close()

    return {"clear_image_patch":nyu_image_patches, "transmission_values":nyu_random_transmission, "haze_image_patch":nyu_haze_patches}

d = load_train_dataset()
print(len(d))
```

Fig. 4 Function to load train dataset from NYU2 Dataset

In the previous snippet, a function is being implemented which is basically used to load the training dataset from the NYU2 Dataset. Here, we are also doing normalizing the data where the image is divided by 255.0, and depth matrix is divided by 4.0 in order to bring the pixel values of images and depth maps in the range of 0 to 1. After that it also extract patches of size 16x16 pixels, and for each extracted patch a random transmission value is chosen and then the image is modified to simulate haze with the randomly chosen value. It also creates a (mj.hdf5) file for the NYU2 dataset in the directory.

```
def create_train_dataset(count = 20, patch_count = 10, comp = 9, shuff = True):
  start_time = time.time()
  d = load_train_dataset(count, patch_count)
  print("--- %s seconds in creating dictionary ---" % (time.time() - start_time))
  print("Dictionary created")

  start_time = time.time()
  train_dataset = h5py.File("train_data.hdf5", "w")
  dset = train_dataset.create_dataset("clear_image",data = d["clear_image_patch"],compression=comp,shuffle=shuff)
  dset = train_dataset.create_dataset("transmission_value",data = d["transmission_values"],compression=comp,shuffle=shuff)
  dset = train_dataset.create_dataset("haze_image",data = d["haze_image_patch"],compression=comp,shuffle=shuff)
  train_dataset.close()
  print("--- %s seconds in creating dataset ---" % (time.time() - start_time))
  print("compression:",dset.compression," compression_opt:",dset.compression_opts," shuffle:",dset.shuffle," size:",os.stat("train_data.hdf5").st_size)
  print("Dataset created")
```

Fig. 5 Create Train Dataset using NYU2 Depth Dataset

In this code snippet, we are creating train dataset using NYU2 Depth Dataset. Firstly we are loading the dataset using the previous function ‘load_train_dataset()’, after loading we are creating a hdf5 file for the training data inside which we are creating 3 datasets – Clear Images, Transmission Value, and Haze Images.

```
temp = h5py.File('train_data.hdf5', 'r')
print(temp.keys())
plt.imshow(temp['clear_image'][1000])
plt.show()
plt.imshow(temp['haze_image'][1000])
plt.show()
print(temp['transmission_value'][1000])
print(temp['clear_image'].shape,temp['haze_image'].shape,len(temp['transmission_value']))
temp.close()
```

Fig. 6 Checking Created Dataset

Here, we are displaying the 1000th clear image, haze image, its associated transmission value and the shapes of clear and haze image.

```python
file = './train_data.hdf5'
train_dataset = h5py.File(file, 'r')
clean_image = np.array(train_dataset['clear_image'][:])
haze_image = np.array(train_dataset['haze_image'][:])
transmission_value = np.array(train_dataset['transmission_value'])

print ("number of training examples:", clean_image.shape[0])
print ("Clean Image Patch shape:", clean_image.shape)
print ("Haze Image Patch shape:", haze_image.shape)
```

Fig. 7 Printing the information of the loaded datasets

```
...    number of training examples: 60000
       Clean Image Patch shape: (60000, 16, 16, 3)
       Haze Image Patch shape: (60000, 16, 16, 3)
```

Fig. 8 Output (Information of dataset)

```python
tm_model = TransmissionModel((31,31,3))
tm_model.load_weights('D:/Major Projects/Image-Dehazing-Using-Residual-Based-Deep-CNN/Model and Weights/Weights/transmodel_weights.h5')
c = np.pad(haze_image,((0,0), (7,8), (7,8), (0,0)), 'symmetric')
nyu_transmission_map = tm_model.predict(c)
b = nyu_transmission_map.reshape(60000,16,16)
d = (haze_image*255.0).astype('uint8')
for i,val in enumerate(b):
    b[i] = TransmissionRefine(d[i],val)
m = nyu_transmission_map.reshape(60000,16,16)
```

Fig. 9 Refinement of Transmission Maps

In this, an instance of transmission model is created and loaded with pre-trained weights, the hazy images are then padded symmetrically to handle edge effects during convolution.

TESTING DATASET

```
for i in haze:
    im = Image.open('D:/Major Projects/SOTS/SOTS/outdoor/hazy/'+i)
    ss = str(i)
    ss = ss[:4]
    val = int(ss)
    print(ss)
    im.save('test/'+str(val)+'.jpg')
k=20
for i in range(len(l)):
    for j in range(k-20,k):
        im_c = Image.open('D:/Major Projects/SOTS/SOTS/outdoor/gt/'+clear[j])
        print(im_c)
        im_c.save('test/'+str(j)+'_clean.jpg')
    k = k + 20
```

Fig. 10 Extraction and Processing of Outdoor image Dataset

In this snippet, we are creating the Outside Images from the SOTS subset of the RESIDE dataset, saving it with the name of hazy as well its corresponding clear image with a suffix of '_clean' in order for us to test the models at later stages.

```
rhaze = random.sample(haze, 20)
for i in range(len(rhaze)):
    im = Image.open('D:/Major Projects/SOTS/SOTS/indoor/hazy/'+rhaze[i])
    c = rhaze[i].partition('_')[0]+'.png'
    im_c = Image.open('D:/Major Projects/SOTS/SOTS/indoor/gt/'+c)
    im_c.save('test_its/'+str(i)+'_clean.jpg')
    im.save('test_its/'+str(i)+'.jpg')
```

Fig. 11 Extraction and Processing of Indoor image Dataset

In this snippet, we are doing the same work for the Indoor Images from the SOTS subset of the RESIDE dataset.

3.4 IMPLEMENTATION

For the implementation part we will be using the Transmission Network Model and Residual-Based Network.

3.4.1 TRANSMISSION NETWORK MODEL

The transmission network model is a type of neural network utilized to estimate the transmission maps in images.

Key Features of TNM-

- **Input-Output Mapping** – Primary function is to map input data and it corresponding output data.
- **Supervised or Unsupervised Learning** – It supports both Supervised as well as unsupervised learning.
- **Loss Functions –** While training TNMs, it can optimize their parameters by minimizing loss function
- **Robustness –** These are robust.

For this project, we are using the concept of Transmission Network Model, to create transmission maps between input haze image and output clear image.

```
# Load Dataset from Google Drive
def load_train_dataset():

    file = 'D:/Major Projects/Image-Dehazing-Using-Residual-Based-Deep-CNN/Jupyter Notebooks/train_data.hdf5'
    train_dataset = h5py.File(file, 'r')
    clean_image = np.array(train_dataset['clear_image'][:])
    haze_image = np.array(train_dataset['haze_image'][:])
    transmission_value = np.array(train_dataset['transmission_value'])

    return clean_image, haze_image, transmission_value


# Gaussian Weight Initializtion for layers
weight_init = RandomNormal(mean=0.0, stddev=0.001)


# LearningRate Decay function
def lr_schedule(epoch, lr, logs={}):

    print('learning_rate:',lr)
    logs.update({'lr': lr})
    if epoch in (49,99):
        return lr*0.5
    else:
        return lr
```

Fig. 12 Learning Rate Decay Schedule function

In this function, we are initially loading the train dataset from the mj.hdf5 file created earlier. After that the we define the weight initialization strategy using a gaussian distribution with a mean of 0 and a standard deviation of 0.001. We are also scheduling the learning rate decay schedule i.e., learning rate is halved at epochs 49 and 99.

```
clean_image, haze_image, transmission_value = load_train_dataset()
transmission_value = transmission_value.reshape(-1,1,1,1)

print ("number of training examples:", clean_image.shape[0])
print ("Clean Image Patch shape:", clean_image.shape)
print ("Haze Image Patch shape:", haze_image.shape)
print ("Transmission Value shape:", transmission_value.shape)
```

Fig. 13 Displaying the Information

```
...  number of training examples: 60000
     Clean Image Patch shape: (60000, 16, 16, 3)
     Haze Image Patch shape: (60000, 16, 16, 3)
     Transmission Value shape: (60000, 1, 1, 1)
```

Fig. 14 Output of the above snippet

```python
def TransmissionModel(input_shape):

    X_input = Input(input_shape, name = 'input1')

    X = Conv2D(16, (3, 3), strides = (1, 1), kernel_initializer = weight_init, name = 'conv1')(X_input)
    X = Activation('relu', name = 'activation1')(X)

    # SLICE Block applied to X
    X1 = Lambda(lambda X: X[:,:,:,:4], name = 'slice1')(X)
    X2 = Lambda(lambda X: X[:,:,:,4:8], name = 'slice2')(X)
    X3 = Lambda(lambda X: X[:,:,:,8:12], name = 'slice3')(X)
    X4 = Lambda(lambda X: X[:,:,:,12:], name = 'slice4')(X)

    # MAXIMUM Block applied to 4 slices
    X = Maximum(name = 'merge1_maximum')([X1,X2,X3,X4])

    # CONV BLock for multi-scale mapping with filters of size 3x3, 5x5, 7x7
    X_3x3 = Conv2D(16, (3, 3), strides = (1, 1), padding = 'same', kernel_initializer = weight_init, name = 'conv2_3x3')(X)
    X_5x5 = Conv2D(16, (5, 5), strides = (1, 1), padding = 'same', kernel_initializer = weight_init, name = 'conv2_5x5')(X)
    X_7x7 = Conv2D(16, (7, 7), strides = (1, 1), padding = 'same', kernel_initializer = weight_init, name = 'conv2_7x7')(X)

    # CONCATENATE Block to join 3 multi-scale layers
    X = Concatenate(name = 'merge2_concatenate')([X_3x3,X_5x5,X_7x7])

    # MAXPOOL layer of filter size 7x7
    X = MaxPooling2D((7, 7), strides = (1, 1), name = 'maxpool1')(X)

    # CONV -> RELU BLock
    X = Conv2D(1, (8, 8), strides = (1, 1), kernel_initializer = weight_init, name = 'conv3')(X)
    X = Activation('relu', name = 'activation2')(X)

    # Create Keras model instance
    model = Model(inputs = X_input, outputs = X, name='TransmissionModel')

    return model
```

Fig. 15 Creating Neural Networks to estimate transmission maps

In this we are constructing neural networks to estimate transmission maps between input and output. In this we are making a total of 18 layers which are:-

- **1** Input Layer
- **2** Convolution Block
- **4** Lambda Layers
- **1** Maximum Layer
- **6** Multi-Scale Convolutional Blocks
- **1** Concatenate layer
- **1** MaxPooling2D Layer
- **2** Convolutional Block

```
model1 = TransmissionModel(haze_image.shape[1:])
model1.summary()
model1.compile(optimizer=SGD(0.001), loss=MeanSquaredError())
```

Fig. 16 Compiling the Model

```
Model: "TransmissionModel"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input1 (InputLayer)            [(None, 16, 16, 3)]  0           []

 conv1 (Conv2D)                 (None, 14, 14, 16)   448         ['input1[0][0]']

 activation1 (Activation)       (None, 14, 14, 16)   0           ['conv1[0][0]']

 slice1 (Lambda)                (None, 14, 14, 4)    0           ['activation1[0][0]']

 slice2 (Lambda)                (None, 14, 14, 4)    0           ['activation1[0][0]']

 slice3 (Lambda)                (None, 14, 14, 4)    0           ['activation1[0][0]']

 slice4 (Lambda)                (None, 14, 14, 4)    0           ['activation1[0][0]']

 merge1_maximum (Maximum)       (None, 14, 14, 4)    0           ['slice1[0][0]',
                                                                  'slice2[0][0]',
                                                                  'slice3[0][0]',
                                                                  'slice4[0][0]']

 conv2_3x3 (Conv2D)             (None, 14, 14, 16)   592         ['merge1_maximum[0][0]']

 conv2_5x5 (Conv2D)             (None, 14, 14, 16)   1616        ['merge1_maximum[0][0]']

 conv2_7x7 (Conv2D)             (None, 14, 14, 16)   3152        ['merge1_maximum[0][0]']
```

[A]

```
 merge2_concatenate (Concatenat  (None, 14, 14, 48)  0           ['conv2_3x3[0][0]',
 e)                                                               'conv2_5x5[0][0]',
                                                                  'conv2_7x7[0][0]']

 maxpool1 (MaxPooling2D)        (None, 8, 8, 48)     0           ['merge2_concatenate[0][0]']

 conv3 (Conv2D)                 (None, 1, 1, 1)      3073        ['maxpool1[0][0]']

 activation2 (Activation)       (None, 1, 1, 1)      0           ['conv3[0][0]']

==================================================================================================
```

[B]

Fig. 17 Compiled Transmission Model

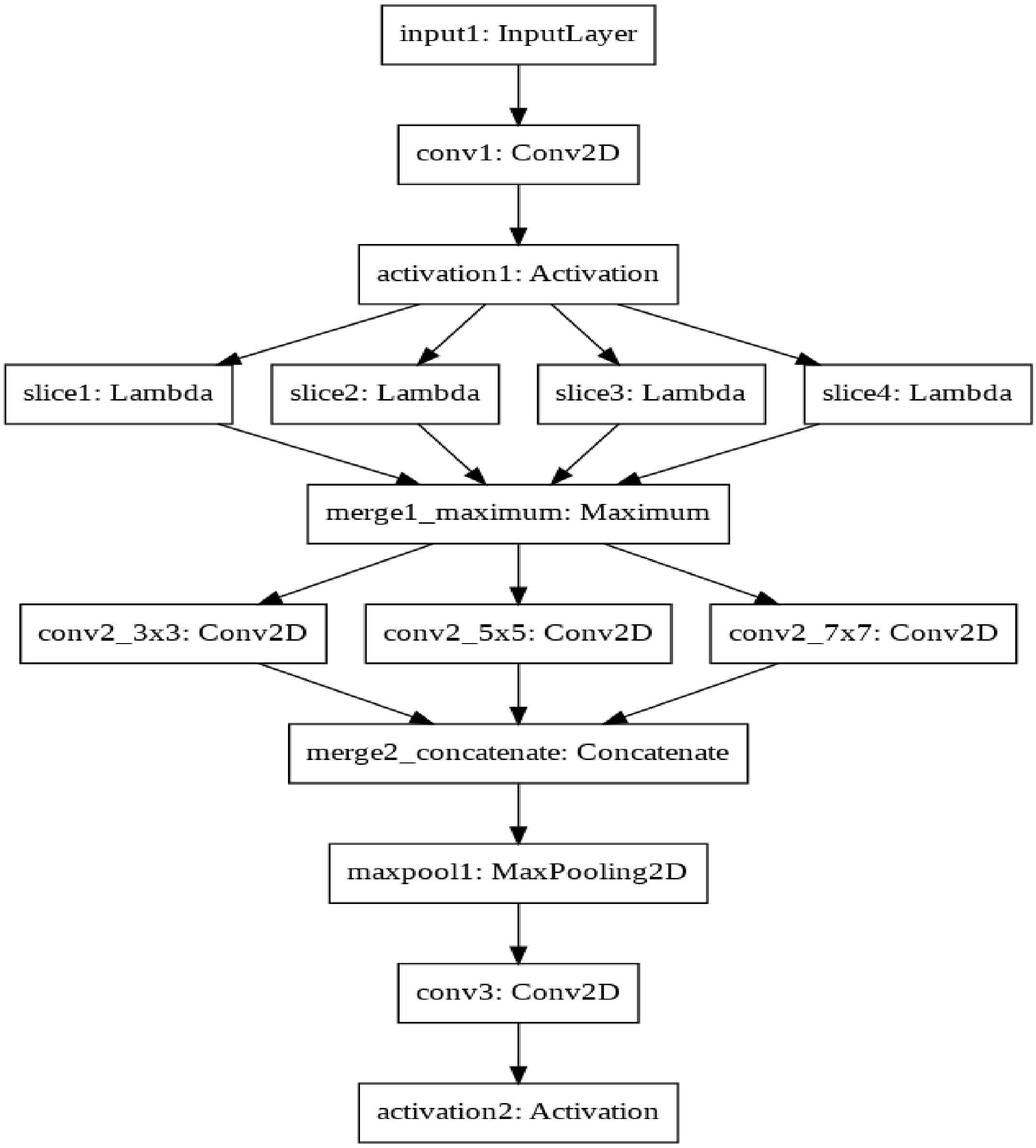

Fig. 18 Transmission Model

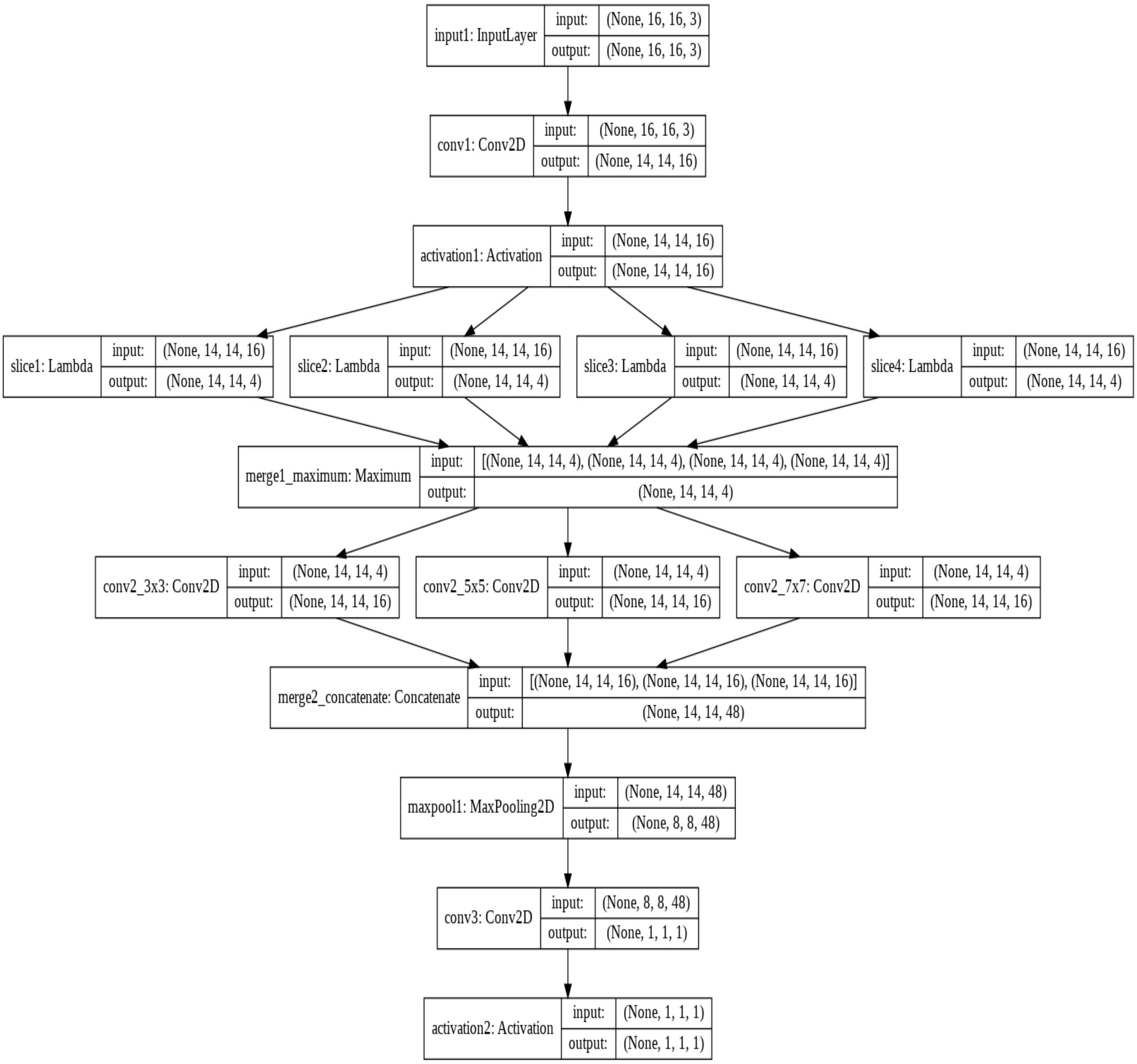

Fig. 19 Trans Model Shape

```
history1 = model1.fit(haze_image, transmission_value, batch_size = 30, epochs = 150, callbacks=[LearningRateScheduler(lr_schedule)])
```

Fig. 20 Training of Transmission Model

Now, we are training the transmission model with following parameters: -

- **150 -** Epochs
- **30** – Batch Size

- **LearingRateScheduler** – utilized to dynamically adjust the learning rate during training.

3.4.2 RESIDUAL-BASED NETWORK

It is a type CNN architecture, which was created to solve the problems the vanishing and exploding gradient issues that frequently arise with the deeper networks. Residual based learning has following key features: -

- **Residual Learning –** It uses residual blocks, also known as skip connection is the essential component of a ResNet. It also facilitates the optimization of deep networks by enabling activations to bypass one or more layers via a shortcut link.
- **Identify Shortcut Connections –** It introduced us with the concept of bypassing or skipping one or more layers. Primary idea behind this is that it is easier to optimize residual mapping which is the difference between the input and output rather than the desired output.
- **Deep Architectures –** Using these residual blocks, ResNet can be built with 50, 101 and 152 layers, but more deeper variants are available.

For this project, we are using the concept of learning residual information, representing the difference between the hazy and clear images.

```
import numpy as np
import h5py
import math

from keras.models import Model
from keras.layers import Input, Activation, BatchNormalization, Conv2D, Conv3D
from keras.layers import Lambda, Concatenate, MaxPooling2D, Maximum, Add
from keras.initializers import RandomNormal
from keras.optimizers import SGD
from keras.losses import MeanSquaredError
from keras.callbacks import Callback,LearningRateScheduler
from keras.utils import plot_model

import keras.backend as K
K.set_image_data_format('channels_last')

import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
```

Fig. 21 Importing Libraries

```python
def load_train_dataset():
    file = 'D:\Major Projects\Image-Dehazing-Using-Residual-Based-Deep-CNN\Jupyter Notebooks\mj.hdf5'
    train_dataset = h5py.File(file, 'r')
    clean_image = np.array(train_dataset['clean_image'][:])
    haze_image = np.array(train_dataset['haze_image'][:])
    transmission_map = np.array(train_dataset['transmission_map'])
    transmission_map_refine = np.array(train_dataset['transmission_map_refine'])

    return clean_image, haze_image, transmission_map, transmission_map_refine

# Gaussian Weight Initializtion for layers
weight_init = RandomNormal(mean=0.0, stddev=0.001)

# LearningRate Decay function
def lr_schedule(epoch,lr, logs={}):
    print('learning_rate:',lr)
    logs.update({'lr': lr})
    if epoch in (49,99):
        return lr*0.5
    else:
        return lr
```

Fig. 22 Learning Rate Decay Schedule function

In this function, we are initially loading the train dataset from the mj.hdf5 file created earlier. After that the we define the weight initialization strategy using a gaussian distribution with a mean of 0 and a standard deviation of 0.001. We are also scheduling the learning rate decay schedule i.e., learning rate is halved at epochs 49 and 99.

```python
clean_image, haze_image, transmission_map, transmission_map_refine = load_train_dataset()

print ("Number of training examples:", clean_image.shape[0])
print ("Clean Image Patch shape:", clean_image.shape)
print ("Haze Image Patch shape:", haze_image.shape)
print ("Transmission Map shape:", haze_image.shape)
print ("Transmission Map Refine shape:", haze_image.shape)
```

Fig. 23 Displaying the Information

In this snippet we are displaying the information of the dataset.

```
Number of training examples: 60000
Clean Image Patch shape: (60000, 16, 16, 3)
Haze Image Patch shape: (60000, 16, 16, 3)
Transmission Map shape: (60000, 16, 16, 3)
Transmission Map Refine shape: (60000, 16, 16, 3)
```

Fig. 24 Training Dataset Information

```
residual_input = np.clip(((haze_image/255.0)/np.expand_dims(transmission_map_refine,axis=3)),0,1)
residual_output = np.clip((residual_input-clean_image),0,1)
```

Fig. 25 Computation of residual_input and residual_output

In this first we compute the residual_input, which is computed by normalizing the haze-image. Similarly, the residual_output is computed by subtracting the clean-images from the residual input, after which it is clipped to ensure it stays within the [0, 1] range. Residual_Output represents the difference between residual_input and the clean image.

```
def ResidualBlock(X, iter):

    X_shortcut = X

    # BATCHNORMALIZATION → CONV Block
    X = BatchNormalization(axis = 3, name = 'res_batchnorm_' + str(iter))(X)
    X = Conv2D(1, (3, 3), strides = (1,1), padding = 'same', kernel_initializer = weight_init, name = 'res_conv_' + str(iter))(X)

    # Add shortcut value to main path, and pass it through a RELU activation
    X = Add(name = 'res_add_'+ str(iter))([X,X_shortcut])
    X = Activation('relu', name = 'res_activation_'+ str(iter))(X)

    return X

def ResidualModel(input_shape):

    X_input = Input(input_shape, name = 'input1')

    # CONV → RELU Block applied to X
    X = Conv2D(16, (3, 3), strides = (1, 1), padding = 'same', kernel_initializer = weight_init, name = 'conv1')(X_input)
    X = Activation('relu', name = 'activation1')(X)

    # X = Conv2D(8, (1, 1), kernel_initializer = weight_init, name='test_conv')(X)

    for i in range(17):
        X = ResidualBlock(X, i)

    # CONV BLock
    X = Conv2D(3, (3, 3), strides = (1, 1), padding = 'same', kernel_initializer = weight_init, name = 'conv2')(X)
    X = Activation('relu', name = 'activation2')(X)

    # Create Keras model instance
    model = Model(inputs = X_input, outputs = X, name='TransmissionModel')

    return model
```

Fig. 26 Residual-Based Network Model

In the residualBlock() function, we define a residual block which consists of a batch normalization, 3x3 convolution layer and the shortcut connection(which is added before passing through ReLU activation).

In the residualModel() function, we are defining the residual model using Keras and Tensorflow.

```
model2 = ResidualModel(residual_input.shape[1:])
model2.summary()
model2.compile(optimizer=SGD(0.001), loss=MeanSquaredError())
```

Fig. 27 Compiling the Residual Model

Here, we have created and compiled our Residual Based Network model consisting of :

- **1** Convolutional Block
- **17** Residual Block

```
==========================================
Total params: 4,436
Trainable params: 3,892
Non-trainable params: 544
__________________________________________
```

Fig. 28 Count of trainable and non-trainable params

```
history2 = model2.fit(residual_input, residual_output, batch_size = 30, epochs = 150, callbacks=[LearningRateScheduler(lr_schedule)])
```

Fig. 29 Training the Residual-Based Network Model

Now, we are training the residual model with following parameters:-

- **150 -** Epochs
- **30** – Batch Size
- **LearingRateScheduler** – utilized to dynamically adjust the learning rate during training.

3.4.3 FRONTEND INTEGRATION

In the project, a user-friendly web interface using Streamlit, which is a python framework for building interactive web applications. This interface allows the users to upload the hazed images and the web app will provide them with the dehazed image in real-time.

```python
def dehaze_image(img_name):
    input_image_orig = np.asarray(Image.open(img_name))/255.0
    input_image = np.pad(input_image_orig,((7,8), (7,8), (0,0)),'symmetric')

    model = TransmissionModel(input_image.shape)
    model.load_weights('./Model and Weights/Weights/transmodel_weights.h5')

    input_image = np.expand_dims(input_image, axis=0)
    trans_map_orig = model.predict(input_image)
    trans_map = trans_map_orig.reshape(input_image_orig.shape[:2])
    trans_map_refine = TransmissionRefine((input_image_orig*255.0).astype('uint8'),trans_map)

    res_map_input = input_image_orig/np.expand_dims(trans_map_refine, axis=(0,3))

    model = ResidualModel(res_map_input.shape[1:])
    model.load_weights('./Model and Weights/Weights/resmodel_weights.h5')
    res_map_output = model.predict(np.clip(res_map_input,0,1))

    haze_free_image = (res_map_input-res_map_output)
    haze_free_image = np.clip(haze_free_image,0,1)

    return haze_free_image[0]
```

Fig. 30 Function to Perform dehazing of Image

In this code snippet, the code implemented the function to dehaze the given image which has come for dehazing.

```python
st.title("Image Dehazing")


if  True:
    st.header("Upload Image")
    image_file = st.file_uploader("Upload Image", type=["png","jpg","jpeg"])
    if bool(image_file)==True :
        st.image(image_file)
        predictions = dehaze_image(image_file)
        st.image(predictions)
```

Fig. 31 Streamlit application to upload Image

In this above snippet, the code implemented the streamlit frontend web application with the feature of uploading the images form the device by the user.

3.5 KEY CHALLENGES

- The image dehazing problem is very complex and non-linear in nature because of light scattering by the atmospheric particles.

- One of the major challenges was that the quality of image is hugely varied since it has both the indoor and outdoor images, thus creating the difference in lighting and contrast conditions. Therefore, to create a model which is capable to dehaze both indoor and outdoor image with the ability to retain its more original colours.
- Since, it is a CNN it requires a heavy computational power to train the model and with the use of more and more big dataset consisting of more images we will be needing a heavy GPU enabled machine to train it efficiently and faster.

CHAPTER – 4 TESTING

Following the successful training of the model, the testing and validation phase will begin by evaluating the model's performance on randomly selected photos from the dataset.

4.1 TESTING STRATEGY

```
import numpy as np
import h5py
import math

from keras.models import Model
from keras.layers import Input, Activation, BatchNormalization, Conv2D, Conv3D
from keras.layers import Lambda, Concatenate, MaxPooling2D, Maximum, Add
from keras.initializers import RandomNormal
from tensorflow.keras.optimizers import schedules, SGD
from keras.callbacks import Callback
from keras.utils import plot_model

import keras.backend as K
K.set_image_data_format('channels_last')

import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow

from PIL import Image

import cv2

%matplotlib inline
```

Fig: - 32 Importing necessary python libraries for the testing of our model

This snippet imports various libraries for the testing our model. Here we imported numpy for numerical computing, h5py for interacting with HDF5 files and it is also used for storing large amounts of numerical data. Math library is for basic mathematical operations, keras is for high level neural networks API, tensorflow is for open-source machine learning framework, matplotlib for creating visualisations, PIL for opening, manipulating and saving different image file formats. This architecture includes convolutional layers, batch normalisation, activation functions. We set up an optimizer (Stochastic Gradient Descent) and a custom callback which we can use to monitor and control our model's training process. We configuring keras to use a specific image data format, likely 'channels_last'. We used

'%matplotlib inline' for displaying plots directly below the code cell whenever we will run it.

```
def Guidedfilter(im,p,r,eps):
  mean_I = cv2.boxFilter(im,cv2.CV_64F,(r,r))
  mean_p = cv2.boxFilter(p, cv2.CV_64F,(r,r))
  mean_Ip = cv2.boxFilter(im*p,cv2.CV_64F,(r,r))
  cov_Ip = mean_Ip - mean_I*mean_p
  mean_II = cv2.boxFilter(im*im,cv2.CV_64F,(r,r))
  var_I   = mean_II - mean_I*mean_I
  a = cov_Ip/(var_I + eps)
  b = mean_p - a*mean_I
  mean_a = cv2.boxFilter(a,cv2.CV_64F,(r,r))
  mean_b = cv2.boxFilter(b,cv2.CV_64F,(r,r))
  q = mean_a*im + mean_b
  return q

def TransmissionRefine(im,et):
  gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
  gray = np.float64(gray)/255
  r = 60
  eps = 0.0001
  t = Guidedfilter(gray,et,r,eps)
  return t
```

Fig. - 33 Util Functions

This snippet defines two functions, 'Guidedfilter' and 'TransmissionRefine' which are important components in context of an image processing and they are possibly associated with tasks like image/video dehazing. The Guidedfilter function implemented a technique commonly used for smoothing images while preserving those important edges by taking an imput image Im, a guidance image p, a window radius r and a regularisation parameter eps. This function calculates local mean, covariance and filters to produce a guided-filter output as q. Now our second function Transmission Refine refines a transmission map (et) by using the previously defined guided filter. The image input Im is converted to grayscale, normalised and then subjected to our guided filter process to enhance the transmission map. These two functions basically serve as essential key steps in a broader image processing pipelines which results in improvement of image quality by reducing or removing noise and refining transmission. Information for subsequent applications.

```
def TransmissionModel(input_shape):

    X_input = Input(input_shape, name = 'input1')

    # CONV → RELU Block applied to X
    X = Conv2D(16, (3, 3), strides = (1, 1), name = 'conv1')(X_input)
    X = Activation('relu', name = 'activation1')(X)

    # SLICE Block applied to X
    X1 = Lambda(lambda X: X[:,:,:,:4], name = 'slice1')(X)
    X2 = Lambda(lambda X: X[:,:,:,4:8], name = 'slice2')(X)
    X3 = Lambda(lambda X: X[:,:,:,8:12], name = 'slice3')(X)
    X4 = Lambda(lambda X: X[:,:,:,12:], name = 'slice4')(X)

    # MAXIMUM Block applied to 4 slices
    X = Maximum(name = 'merge1_maximum')([X1,X2,X3,X4])

    # CONV BLock for multi-scale mapping with filters of size 3x3, 5x5, 7x7
    X_3x3 = Conv2D(16, (3, 3), strides = (1, 1), padding = 'same', name = 'conv2_3x3')(X)
    X_5x5 = Conv2D(16, (5, 5), strides = (1, 1), padding = 'same', name = 'conv2_5x5')(X)
    X_7x7 = Conv2D(16, (7, 7), strides = (1, 1), padding = 'same', name = 'conv2_7x7')(X)

    # CONCATENATE Block to join 3 multi-scale layers
    X = Concatenate(name = 'merge2_concatenate')([X_3x3,X_5x5,X_7x7])

    # MAXPOOL layer of filter size 7x7
    X = MaxPooling2D((7, 7), strides = (1, 1), name = 'maxpool1')(X)

    # CONV → RELU BLock
    X = Conv2D(1, (8, 8), strides = (1, 1), name = 'conv3')(X)
    X = Activation('relu', name = 'activation2')(X)

    # Create Keras model instance
    model = Model(inputs = X_input, outputs = X, name='TransmissionModel')

    return model
```

Fig: - 34 Transmission model

This code has a neural network architecture known as residual neural network (ResNet) using a python library known as keras. Here, the ResNet is designed to learn important features within the input image very efficiently. The 'ResidualBlock function' defines as the fundamental building block of our network via batch normalisation, a 3x3 convolutional layer with a residual connection and this block will be applied iteratively in the Residual Model function where initially another convolutional block which is followed by 17 residual blocks to capture and retain crucial features. The model generates a 3- channel output and a Rectified Linear Unit (ReLU) activation. The model is tuned to enhance the ability to learn and represent patterns in the input data with a sole focus on feature extraction for our image processing task at hand.

```
def ResidualBlock(X, iter):

    # Save the input value
    X_shortcut = X

    # BATCHNORMALIZATION → CONV Block
    X = BatchNormalization(axis = 3, name = 'res_batchnorm' + str(iter))(X)
    X = Conv2D(1, (3, 3), strides = (1,1), padding = 'same', kernel_initializer=RandomNormal(mean=0.0, stddev=0.001), name = 'res_conv' + str(iter))(X)

    # Add shortcut value to main path, and pass it through a RELU activation
    X = Add(name = 'res_add'+ str(iter))([X,X_shortcut])
    X = Activation('relu', name = 'res_activation'+ str(iter))(X)

    return X

def ResidualModel(input_shape):

    X_input = Input(input_shape, name = 'input1')

    # CONV → RELU Block applied to X
    X = Conv2D(16, (3, 3), strides = (1, 1), padding = 'same', kernel_initializer=RandomNormal(mean=0.0, stddev=0.001), name = 'conv1')(X_input)
    X = Activation('relu', name = 'activation1')(X)

    for i in range(17):
        X = ResidualBlock(X, i)

    # CONV BLock
    X = Conv2D(3, (3, 3), strides = (1, 1), padding = 'same', kernel_initializer=RandomNormal(mean=0.0, stddev=0.001), name = 'conv2')(X)
    X = Activation('relu', name = 'activation2')(X)

    # Create Keras model instance
    model = Model(inputs = X_input, outputs = X, name='TransmissionModel')

    return model
```

Fig: - 35 Residual model

This code has a neural network architecture known as residual neural network (ResNet) using a python library known as keras. Here, the ResNet is designed to learn important features within the input image very efficiently. The ‘ResidualBlock function’ defines as the fundamental building block of our network via batch normalisation, a 3x3 convolutional layer with a residual connection and this block will be applied iteratively in the ResidualModel function where initially another convolutional block which is followed by 17 residual blocks to capture and retain crucial features. The model generates a 3- channel output and a Rectified Linear Unit (ReLU) activation.The model is tuned to enhance the ability to learn and represent patterns in the input data with a sole focus on feature extraction for our image processing task at hand.

```
def dehaze_image(img_name):
    input_image_orig = np.asarray(Image.open(img_name))/255.0
    input_image = np.pad(input_image_orig,((7,8), (7,8), (0,0)),'symmetric')

    model = TransmissionModel(input_image.shape)
    model.load_weights('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Model and Weights/Weights/transmodel_weights.h5')

    input_image = np.expand_dims(input_image, axis=0)
    trans_map_orig = model.predict(input_image)
    trans_map = trans_map_orig.reshape(input_image_orig.shape[:2])
    trans_map_refine = TransmissionRefine((input_image_orig*255.0).astype('uint8'),trans_map)

    res_map_input = input_image_orig/np.expand_dims(trans_map_refine, axis=(0,3))

    model = ResidualModel(res_map_input.shape[1:])
    model.load_weights('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Model and Weights/Weights/resmodel_weights.h5')
    res_map_output = model.predict(np.clip(res_map_input,0,1))

    haze_free_image = (res_map_input-res_map_output)
    haze_free_image = np.clip(haze_free_image,0,1)

    return haze_free_image[0]
```

Fig: - 36 Haze removal function

This code has a function 'dehaze_image' which is designed for image/video dehazing using a deep neural network. It begins with loading an input image, normalising its pixel values and padding them symmetrically. We will then be utilising two models which we have already pre trained. The 'TransmissionRefine' function will enhance the accuracy of the transmission estimation. The dehazing process will continue with the creation of a residual map which will further represent the difference between the original input and the refined transmission. The second model 'ResidualModel' will refine the residual map further and the resulting haze-free image will be obtained by subtracting the refined map from the original input. This will be done with the final output constrained to pixel values between 0 and 1.

```
out = dehaze_image('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/cones.jpg')
plt.imshow(out)
```

Fig: - 37 Output of trained model

This code segment includes an input image, and the system generates the dehazed output image using two pre-trained models.

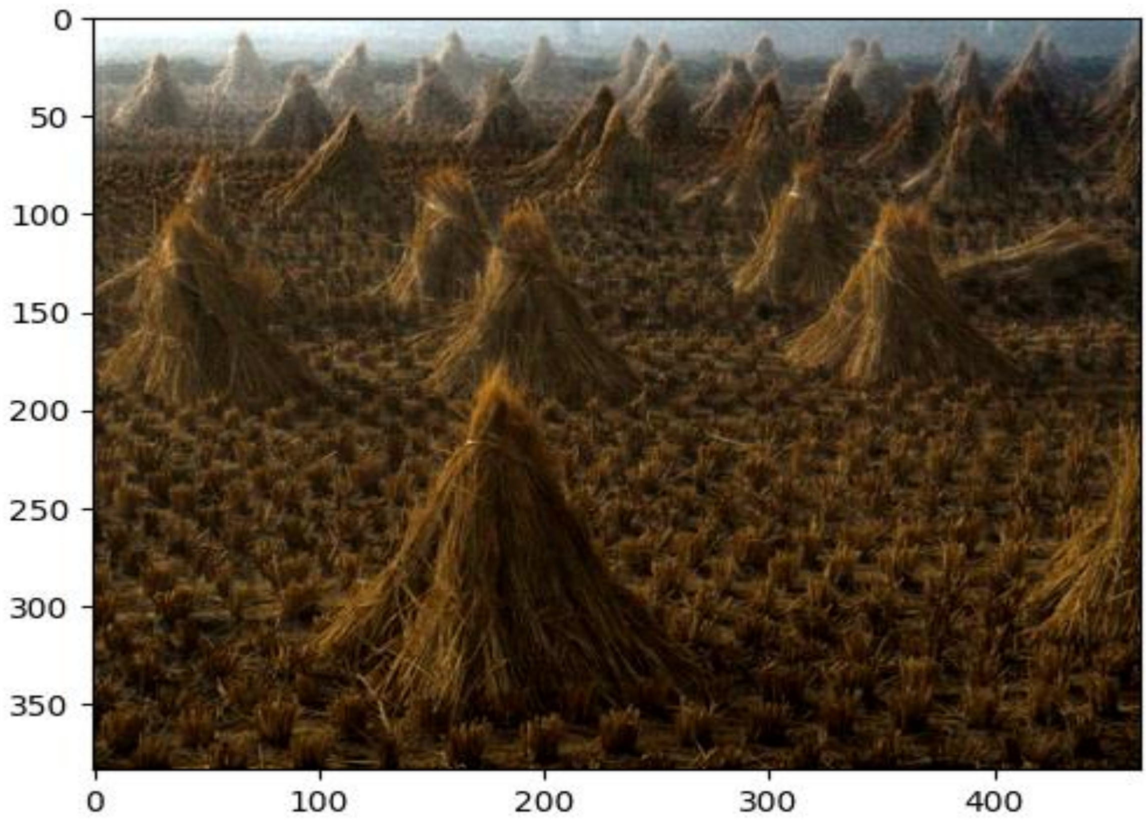

Fig:- 38 Output image based on pre-trained models

```
for i in range(23):
    if i == 0 or i==5 or i==8 or i==12 or i==13 or i==15 or i==20:
        continue
    out = dehaze_image('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Jupyter Notebooks/test/'+str(i)+'.jpg')
    plt.imsave(('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Jupyter Notebooks/updated_results/ots/'+str(i)+'_dehaze.png'),out)
```

Fig:- 39 Iterations for image dehazing

Here the code will iterate from 0 to 22 and for each iteration it will check whether the current index matches with some certain predetermined values (0,5,8,12,13,15 or 20). If it does then it will skip the loop otherwise it will proceed with the dehazing process. The 'dehaze_image' function is applied to images loaded from the directory and the results of dehazed images will be saved in a different directory with new and modified file name.

4.2 TEST CASES AND OUTCOMES

```
input_image_orig = np.asarray(Image.open('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Jupyter Notebooks/test/77.jpg'))/255.0
input_image = np.pad(input_image_orig,((7,8), (7,8), (0,0)),'symmetric')
```

Fig:- 40 Initial preprocessing of input image

```
model = TransmissionModel(input_image.shape)
# model.summary()
model.load_weights('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Model and Weights/Weights/transmodel_weights.h5')

input_image = np.expand_dims(input_image, axis=0)
trans_map_orig = model.predict(input_image)
trans_map = trans_map_orig.reshape(input_image_orig.shape[:2])
trans_map_refine = TransmissionRefine((input_image_orig*255.0).astype('uint8'),trans_map)
```

Fig: - 41 Predict Transmission Map

Here in this code the image in converted into a NumPy array, representing its pixel values and normalising them by dividing each pixel value by 255.0 for ensuring so that the values are in the range of 0 to 1. Then the image is symmetrically padded with a border of 7 pixels on the top, bottom, left and right sides. This is being done to accommodate the convolutional operations which may be applied during subsequent image processing task.

```
res_map_input = input_image_orig/np.expand_dims(trans_map_refine, axis=(0,3))
```

Fig. 42 Residual model input

Here in this code a model is employed for estimating transmission maps. The model initialises and loads, pre-trained weights for a deep learning model, which is known as TransmissionModel and we designed this to estimate transmission maps for our dehazing images. We applied this to an input image and resulting trans map is refined using our function which is TransmissionRefine function. These are some essential steps for effective dehazing processing in our overall image enhancement pipeline.

```
model = ResidualModel(res_map_input.shape[1:])
model.load_weights('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Model and Weights/Weights/resmodel_weights.h5')
res_map_output = model.predict(np.clip(res_map_input,0,1))
```

Fig: - 43 Predicting residual image

Here this code calculates a residual map by dividing our original input image via refined transmission map. This code is basically expanding the dimensions of refined trans map. Here this code initialises and loads our pre-trained weights for our Residual Model so that they can do the refining of residual maps in the dehazing process. Our model is then applied to input residual map so that we can obtain a refined output. This code captures the residual information of the image.

```
haze_free_image = (res_map_input-res_map_output)
haze_free_image = np.clip(haze_free_image,0,1)
```

Fig:- 44 Generate Dehazed Image

This code calculates a haze free image by subtracting our refined residual map from the original input map. After this step is done, the image is then clipped in pixel value between 0 to 1. Further doing these steps we get our dehazed image/video as our desired output which enhances the overall image quality via removing some haze artifacts.

```
print('Input Image')
plt.imshow(input_image_orig)
plt.show()

print('Transmission Map')
plt.imshow(trans_map)
plt.show()

print('Refined Transmission Map')
plt.imshow(trans_map_refine)
plt.show()

print('Residual Model Input Image')
plt.imshow(np.clip(res_map_input[0],0,1))
plt.show()

print('Residual Model Output Image')
plt.imshow(np.clip(res_map_output[0],0,1))
plt.show()

print('Generated Haze Free Image')
plt.imshow(haze_free_image[0])
plt.show()
```

Fig:- 45 Plotting of images at different steps

This previous code displays original input image, estimated transmission map, refined transmission map, input image for residual model, output image for residual model and final output of dehaze image. With the help of these various stages, we will be able to easily distinguish or differ different image dehazing processes.

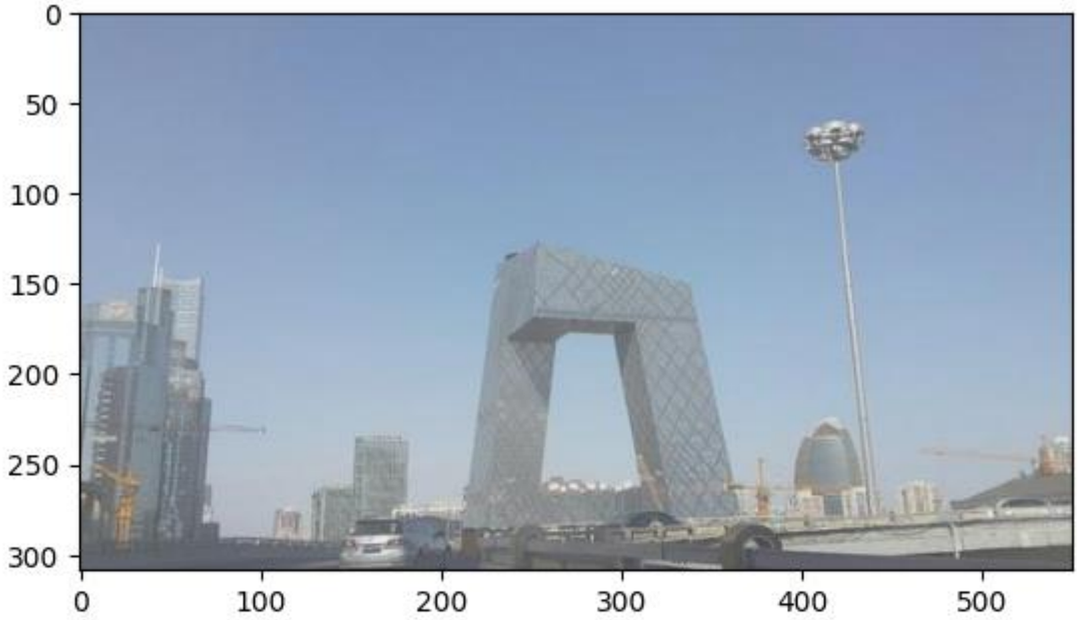

Fig:- 46 Input Image for dehazing

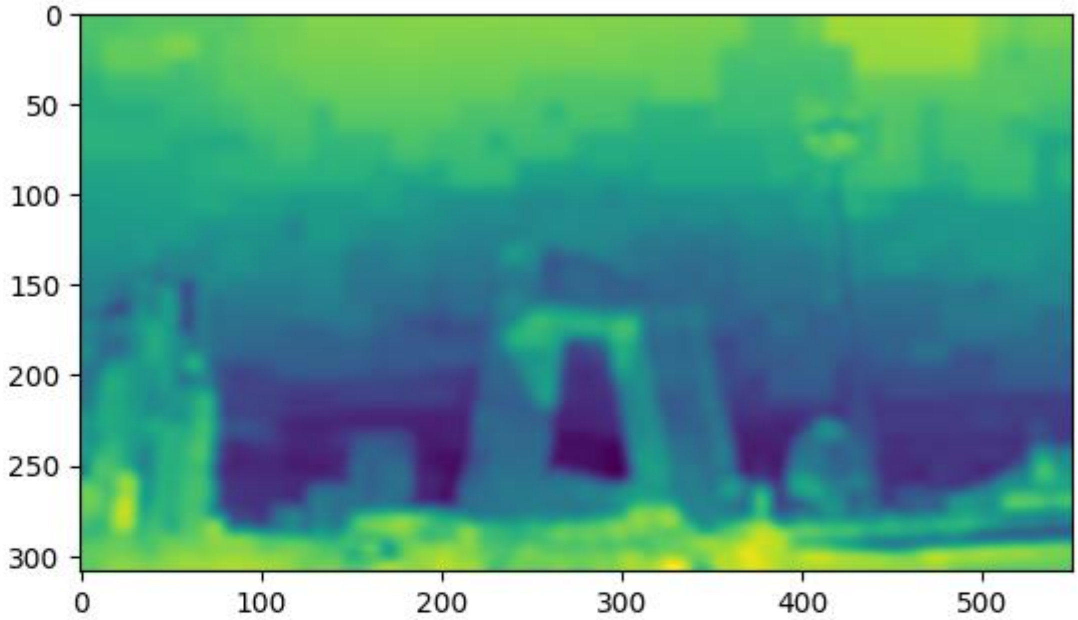

Fig:- 47 Transmission Map

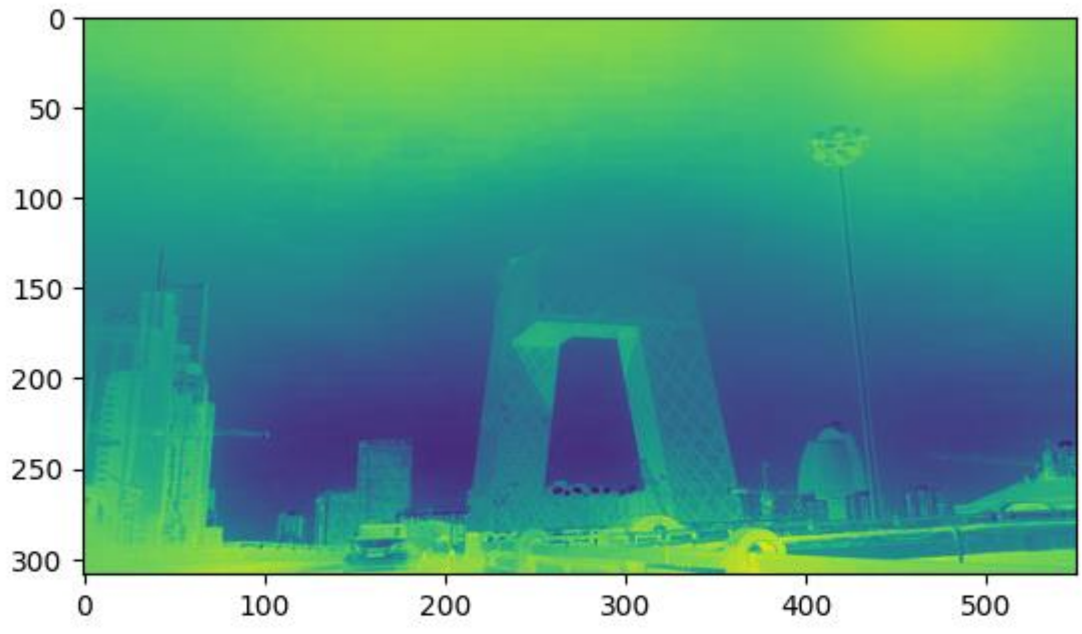

Fig:- 48 Refined Transmission Map

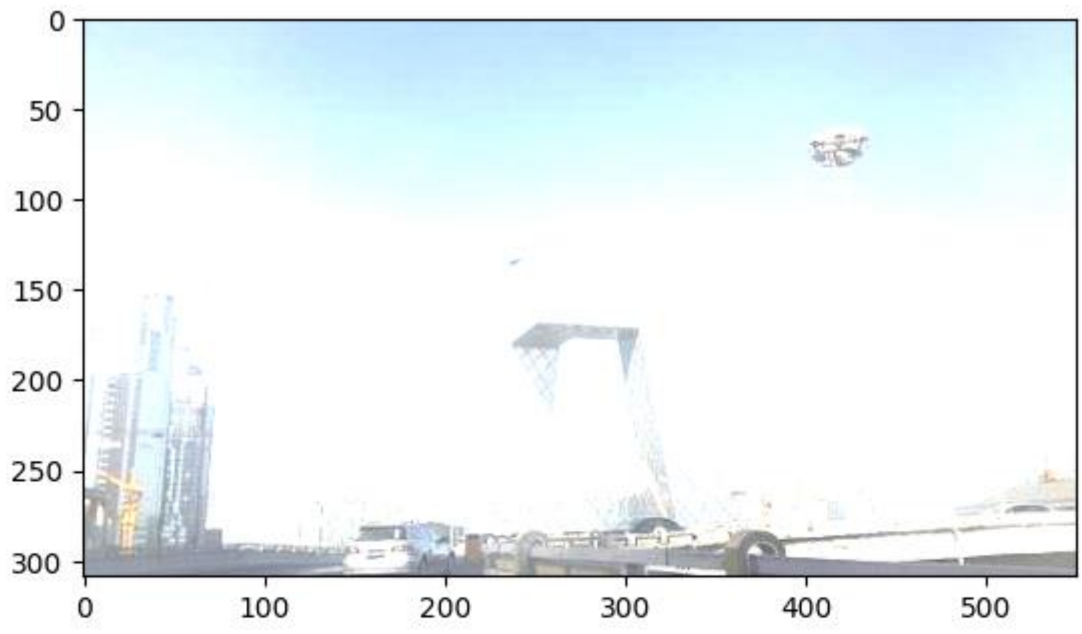

Fig:- 49 Residual Model Input Image

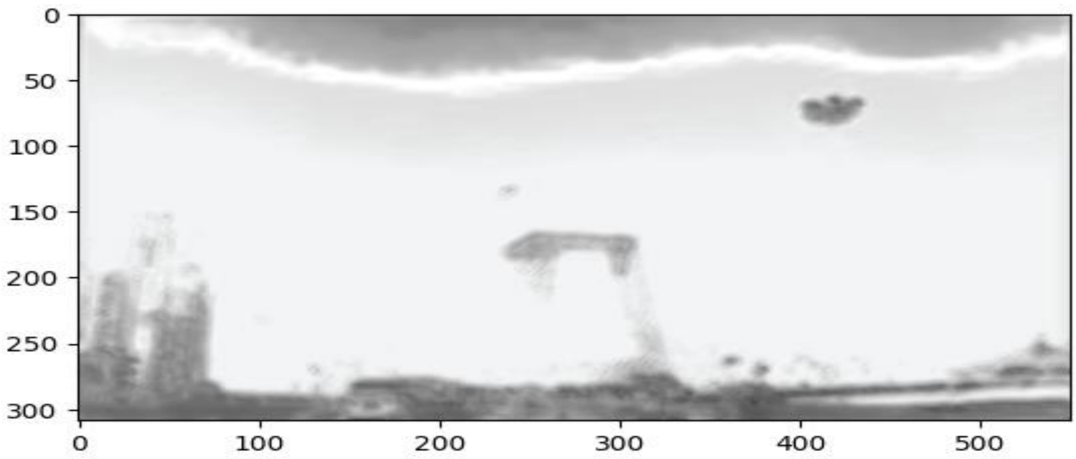

Fig:- 50 Residual Model Output Image

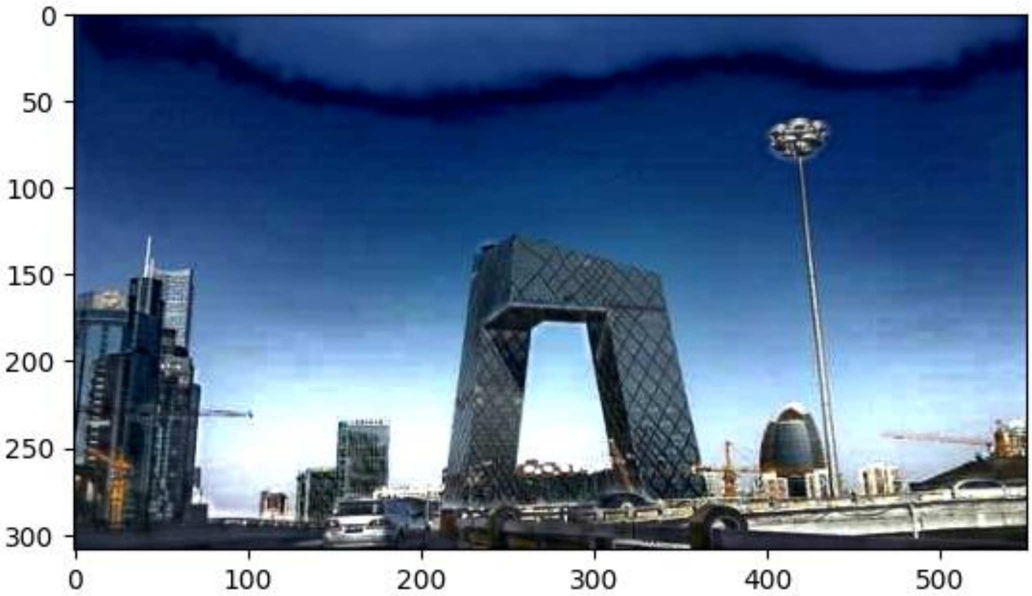

Fig- 51 Generated Haze Free Image

```
plt.imsave('trans.png',trans_map)
plt.imsave('trans_refine.png',trans_map_refine)
plt.imsave('res_in.png',np.clip(res_map_input[0],0,1))
plt.imsave('res_out.png',np.clip(res_map_output[0],0,1))
plt.imsave('dehazed.jpg',np.clip(haze_free_image[0],0,1))
```

Fig:- 52 Saving of Images

Here this code will save all images which were given by the model as an output. Code is using 'np.clip' function which will ensure the pixel values are within valid range of 0 to 1 before saving them.

CHAPTER 5: RESULTS AND EVALUATION

5.1 RESULTS

After successful training of the both the networks we will be validating the models and see how they are performing the testing dataset.

5.1.1 TRANSMISSION MODEL

```
plt.plot(history1.history['lr'])
plt.title('model learning rate')
plt.ylabel('learning rate')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper right')
plt.savefig('trans150-30-lr.png')
plt.show()
plt.plot(history1.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper right')
plt.savefig('trans150-30-loss.png')
plt.show()
```

Fig. 53 Plotting the Model Loss and Model Learning Rate

In this snippet, graphs are being plotted –

- **Model Learning Rate –** It shows the learning rate of the model over the epochs.

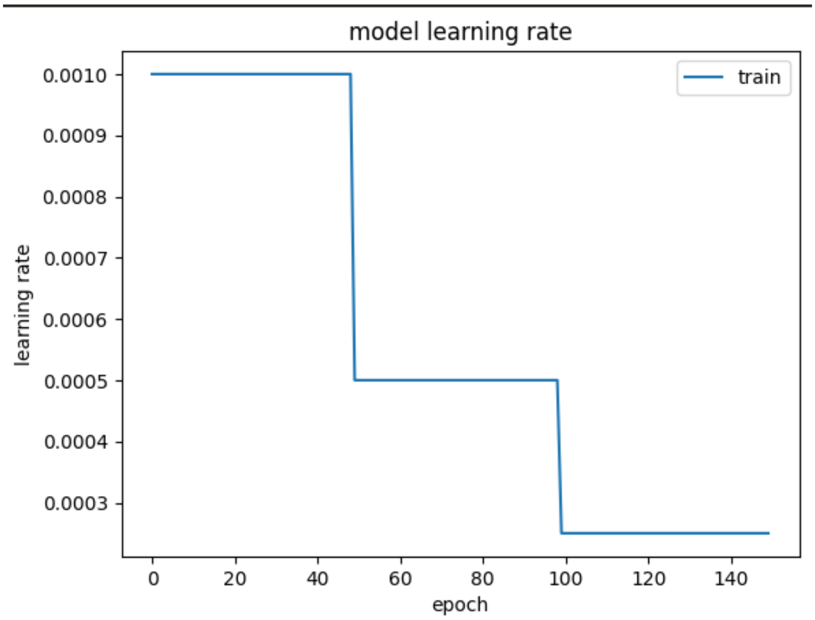

Fig. 54 Model Learning Rate

- **Model Loss** – It shows the Model Loss over the epochs.

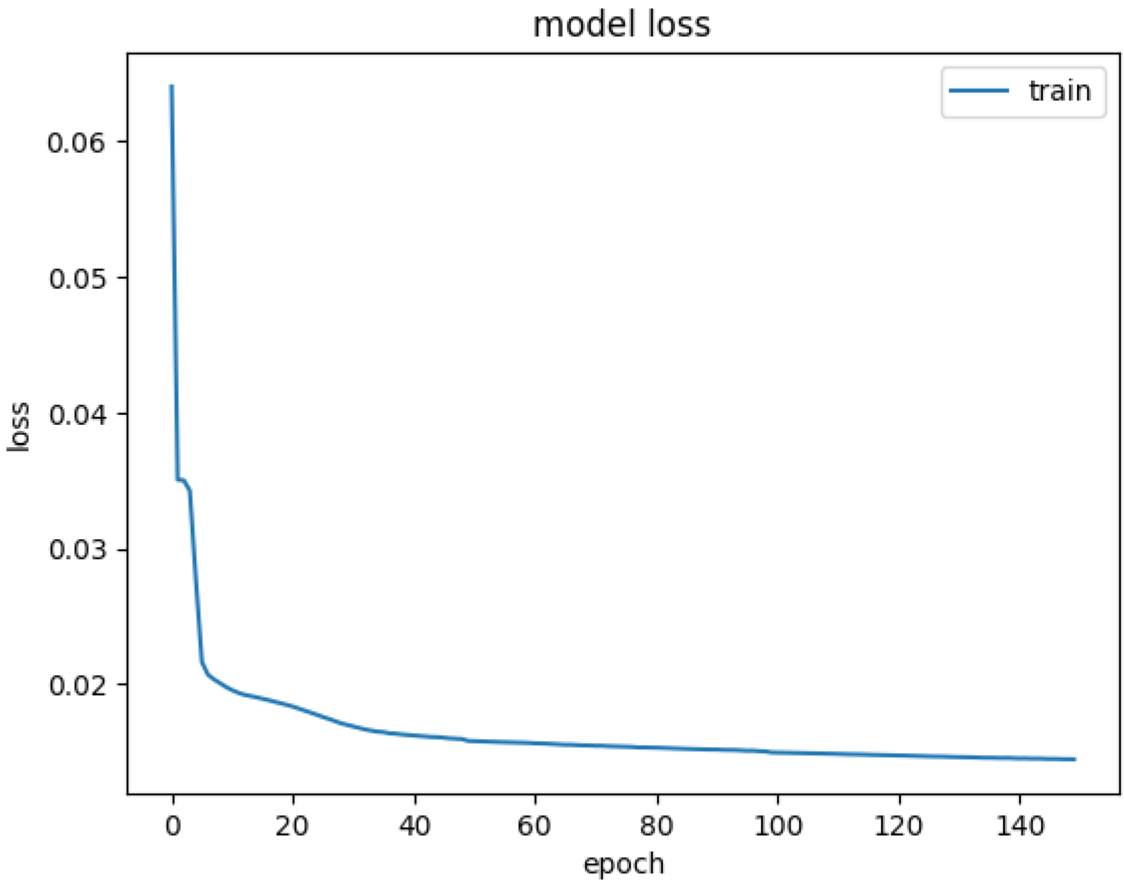

Fig. 55 Model Loss

```
model1.save("transmodel_150_30.h5")
model1.save_weights('transmodel_150_30_weights.h5')
```

Fig. 56 Saving the model and weights

Here, the transmission model along with the weights are being saved.

5.1.2 RESIDUAL BASED NETWORK

```python
plt.plot(history2.history['lr'])
plt.title('model learning rate')
plt.ylabel('learning rate')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper right')
plt.savefig('res150-30-lr.png')
plt.show()
plt.plot(history2.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper right')
plt.savefig('res150-30-loss.png')
plt.show()
```

Fig. 57 Plotting the Model Loss and Model Learning Rate

In this snippet, graphs are being plotted –

- **Model Learning Rate –** It shows the learning rate of the model over the epochs.

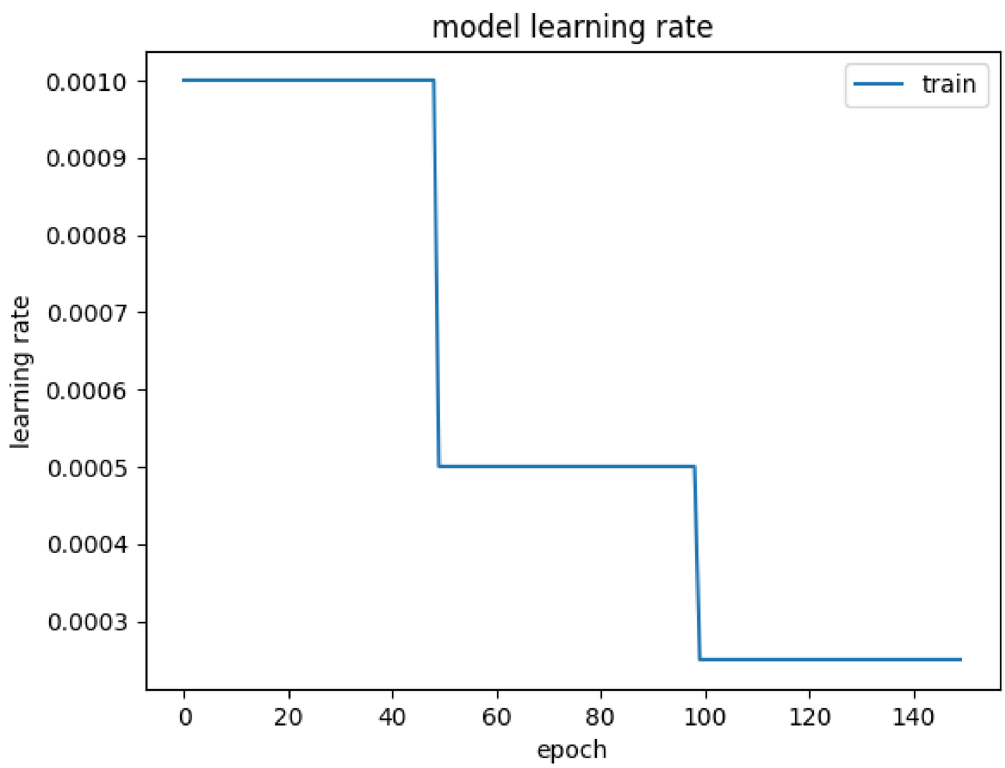

Fig. 58 Model Learning Rate

- **Model Loss –** It shows the Model Loss over the epochs.

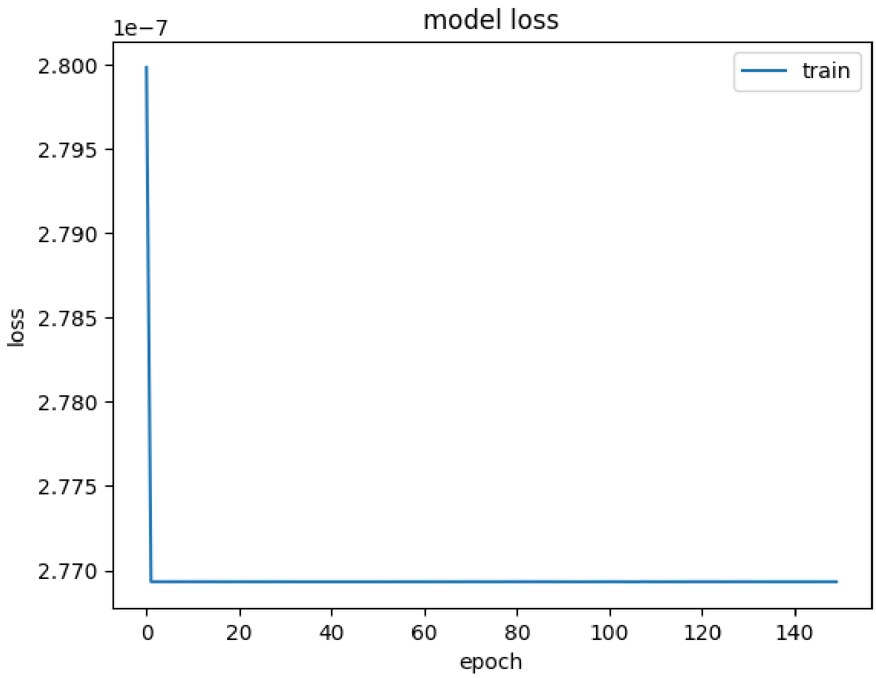

Fig. 59 Model Loss

```
model2.save('resmodel_150_30.h5')
model2.save_weights('resmodel_150_30_weights.h5')
```

Fig. 60 Saving the model and weights

Here, the Residual model along with the weights are being saved.

5.1.3 WEB APPLICATION

After successfully training and testing both models, the implementation of the web application is done and easily integrated the models within its framework.

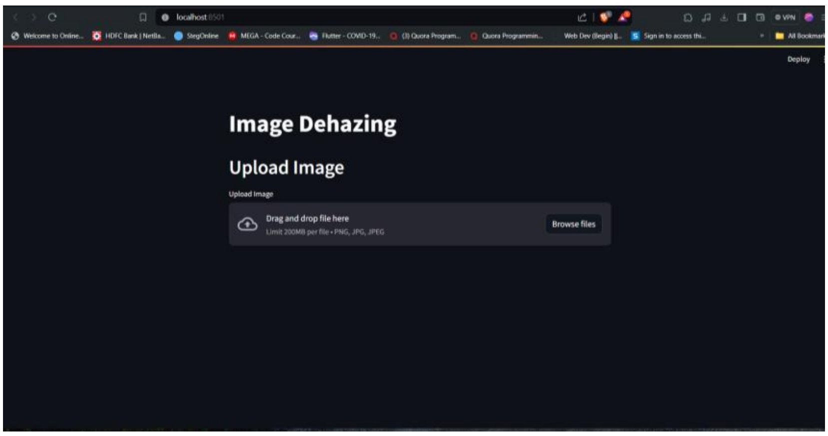

Fig 61 Home Page of our Web Application

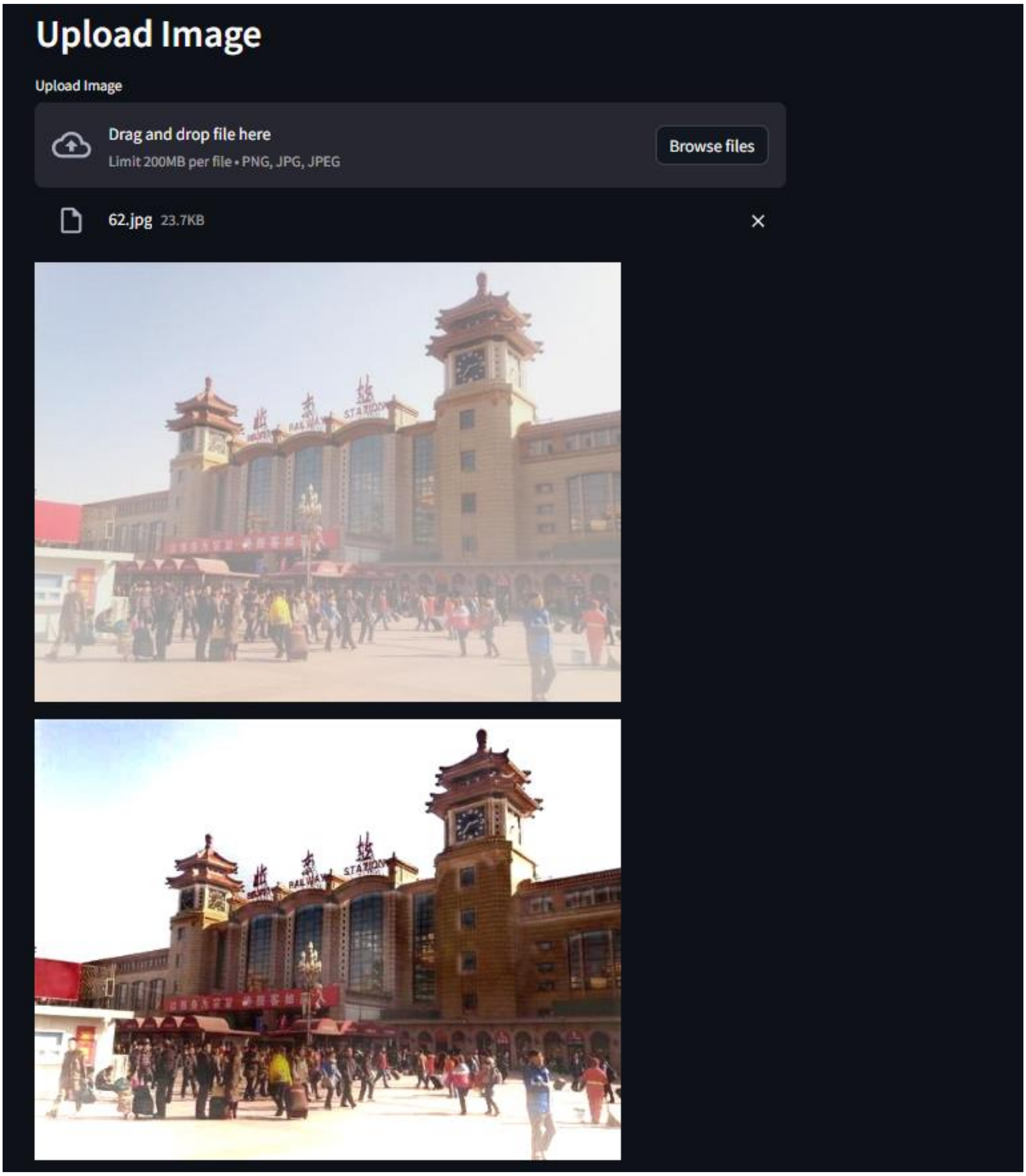

Fig 62 Dehazed Image generated by the web application

Here, it is clear that the web application is working properly, efficiently transforming the foggy image into a dehazed one.

Since it was demonstrated in Chapter 4 (4.2) that the trained model is capable of efficiently dehazing the image.

- The functional requirement of having a codebase capable of handling uploaded images and utilizing them for future use has been successfully fulfilled.

- Requirement for the model to be able to dehaze both indoor and outdoor images is also being satisfied by the model.
- The model is fast enough to be able to dehaze the image within 2 sec of time after being uploaded to it.

CHAPTER - 6

CONCLUSIONS AND FUTURE SCOPE

6.1 CONCLUSION

According to the project report, the team successfully created a single image dehazing solution based on a residual-based deep convolutional neural network (CNN). This method significantly eliminates the necessity for atmospheric light estimate, increasing the effectiveness of image dehazing. The network model is divided into two phases, with the residual network in charge of training the ambient light values. Extensive testing was carried out on both the NYU2 depth dataset and the RESIDE dataset, and the suggested model outperformed previous methods in terms of qualitative evaluation criteria. Notably, the model demonstrated significant effectiveness in dehazing varied settings, with little color distortion or image blurring seen. The results closely matched accepted criteria, demonstrating the efficacy of the proposed approach.

6.2 FUTURE SCOPE

The suggested approach offers several opportunities for future refinement and extension, notably in terms of video dehazing capabilities. There is much room for improvement in the model's efficiency. Furthermore, it is clear that a larger and more realistic dataset for training the network model will significantly improve its performance. Despite the painstaking efforts made in the current model, the team agrees that further enhancements are achievable, with the goal of achieving greater outcomes and eventually implementing the model in video dehazing applications.

REFERENCES

[1] H. Ullah et al., "Light-DehazeNet: A Novel Lightweight CNN Architecture for Single Image Dehazing," in IEEE Transactions on Image Processing, vol. 30, pp. 8968-8982, 2021, doi: 10.1109/TIP.2021.3116790.

[2] X. Zhang, T. Wang, W. Luo and P. Huang, "Multi-Level Fusion and Attention-Guided CNN for Image Dehazing," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 31, no. 11, pp. 4162-4173, Nov. 2021, doi: 10.1109/TCSVT.2020.3046625.

[3] B.S.N.V. Chaitanya, Snehasis Mukherjee, Single image dehazing using improved cycleGAN, Journal of Visual Communication and Image Representation, Volume 74, 2021, 103014, ISSN 1047-3203, https://doi.org/10.1016/j.jvcir.2020.103014.

[4] L. Li et al., "Semi-Supervised Image Dehazing," in IEEE Transactions on Image Processing, vol. 29, pp. 2766-2779, 2020, doi: 10.1109/TIP.2019.2952690.

[5] Y. Du and X. Li, "Recursive Deep Residual Learning for Single Image Dehazing," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Salt Lake City, UT, USA, 2018, pp. 843-8437, doi: 10.1109/CVPRW.2018.00116.

[6] C. Li, J. Guo, F. Porikli, H. Fu and Y. Pang, "A Cascaded Convolutional Neural Network for Single Image Dehazing," in IEEE Access, vol. 6, pp. 24877-24887, 2018, doi: 10.1109/ACCESS.2018.2818882.

[7] Y. Song, J. Li, X. Wang and X. Chen, "Single Image Dehazing Using Ranking Convolutional Neural Network," in IEEE Transactions on Multimedia, vol. 20, no. 6, pp. 1548-1560, June 2018, doi: 10.1109/TMM.2017.2771472.

[8] B. Li, X. Peng, Z. Wang, J. Xu and D. Feng, "AOD-Net: All-in-One Dehazing Network," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 4780-4788, doi: 10.1109/ICCV.2017.511.

[9] Alharbi, E. , Ge, P. and Wang, H. (2016) A Research on Single Image Dehazing Algorithms Based on Dark Channel Prior. Journal of Computer and Communications, 4, 47-55. doi: 10.4236/jcc.2016.42006.

[10] Yeh, C.-H. et al. (2018) ‘Single image dehazing via Deep Learning-based image restoration’, 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC) [Preprint]. doi:10.23919/apsipa.2018.8659733.

[11] Ren, W. *et al.* (2016) ‘Single image dehazing via multi-scale convolutional Neural Networks’,*Computer Vision–ECCV 2016*, pp. 154–169. doi:10.1007/978-3-319-46475-6_10.

[12] Kim, J.-H. *et al.* (2013) ‘Optimized contrast enhancement for real-time image and video dehazing’, *Journal of Visual Communication and Image Representation*, 24(3), pp. 410–425. doi:10.1016/j.jvcir.2013.02.004.

[13] Hodges, C., Bennamoun, M. and Rahmani, H. (2019) ‘Single image dehazing using Deep Neural Networks’, *Pattern Recognition Letters*, 128, pp. 70–77. doi:10.1016/j.patrec.2019.08.013.

[14] Sahu, G. *et al.* (2021) ‘Single image dehazing using a new color channel’, *Journal of Visual Communication and Image Representation*, 74, p. 103008. doi:10.1016/j.jvcir.2020.103008.

[15] Jiang, Y. *et al.* (2017) ‘Image dehazing using adaptive bi-channel priors on superpixels’, *Computer Vision and Image Understanding*, 165, pp. 17–32. doi:10.1016/j.cviu.2017.10.014.

[16] Rong, Z. and Jun, W.L. (2014) ‘Improved wavelet transform algorithm for single image dehazing’, *Optik*, 125(13), pp. 3064–3066. doi:10.1016/j.ijleo.2013.12.077.

[17] Gao, Y. *et al.* (2014) ‘A fast image dehazing algorithm based on negative correction’, *Signal Processing*, 103, pp. 380–398. doi:10.1016/j.sigpro.2014.02.016.

[18] Yin, S., Wang, Y. and Yang, Y.-H. (2021) ‘Attentive U-recurrent encoder-decoder network for image dehazing’, *Neurocomputing*, 437, pp. 143–156. doi:10.1016/j.neucom.2020.12.081.

[19] Feng, T. *et al.* (2021) 'URNet: A U-net based residual network for image dehazing', *Applied Soft Computing*, 102, p. 106884. doi:10.1016/j.asoc.2020.106884.

[20] Ashwini, K., Nenavath, H. and Jatoth, R.K. (2022) 'Image and video dehazing based on transmission estimation and refinement using Jaya algorithm', *Optik*, 265, p. 169565. doi:10.1016/j.ijleo.2022.169565.

[21] Ren, W. and Cao, X. (2018) 'Deep Video Dehazing', Advances in Multimedia Information Processing – PCM 2017, pp. 14–24. doi:10.1007/978-3-319-77380-3_2.

[22] Lv, X., Chen, W. and Shen, I. (2010) 'Real-time Dehazing for image and video', *2010 18th Pacific Conference on Computer Graphics and Applications* [Preprint]. doi:10.1109/pacificgraphics.2010.16.

[23] Goncalves, L.T. et al. (2017) 'Deepdive: An end-to-end Dehazing method using Deep Learning', 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI) [Preprint]. doi:10.1109/sibgrapi.2017.64.

[24] Zhang, X. *et al.* (2021) 'Learning to restore hazy video: A new real-world dataset and a new method', *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [Preprint]. doi:10.1109/cvpr46437.2021.00912.

[25] Zhang, S., He, F. and Yao, J. (2018) 'Single image dehazing using Deep Convolution Neural Networks', *Advances in Multimedia Information Processing – PCM 2017*, pp. 128–137. doi:10.1007/978-3-319-77380-3_13.

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: ________________________________ **Department:** __________________ **Enrolment No** __________

Contact No. ________________________________ **E-mail.** ____

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): ___________________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at................... (%). Therefore, we

are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

Image Dehazing

ORIGINALITY REPORT

10%	6%	8%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Jinjiang Li, Guihui Li, Hui Fan. "Image Dehazing Using Residual-Based Deep CNN", IEEE Access, 2018 Publication	2%
2	export.arxiv.org Internet Source	1%
3	medium.com Internet Source	1%
4	pdfs.semanticscholar.org Internet Source	1%
5	faculty.ucmerced.edu Internet Source	1%
6	technodocbox.com Internet Source	1%
7	research.snu.edu.in Internet Source	1%
8	Xiaoqin Zhang, Tao Wang, Wenhan Luo, Pengcheng Huang. "Multi-Level Fusion and Attention-Guided CNN for Image Dehazing",	1%

IEEE Transactions on Circuits and Systems for Video Technology, 2021
Publication

9	Rajat Tiwari, Bhawna Goyal, Ayush Dogra. "Image enhancement using convolution neural network due to aerosols suspended in environment", Materials Today: Proceedings, 2023 Publication	<1 %
10	Jinjiang Li, Guihui Li, Hui Fan.. "Image Dehazing using Residual-based Deep CNN", IEEE Access, 2018 Publication	<1 %
11	deepai.org Internet Source	<1 %
12	www.arxiv-vanity.com Internet Source	<1 %
13	B.S.N.V. Chaitanya, Snehasis Mukherjee. "Single image dehazing using improved cycleGAN", Journal of Visual Communication and Image Representation, 2021 Publication	<1 %
14	Chuansheng Wang, Zuoyong Li, Jiawei Wu, Haoyi Fan, Guobao Xiao, Hong Zhang. "Deep Residual Haze Network for Image Dehazing and Deraining", IEEE Access, 2020 Publication	<1 %

15	Submitted to Liverpool John Moores University Student Paper	<1 %
16	iopscience.iop.org Internet Source	<1 %
17	Submitted to Chandigarh University Student Paper	<1 %
18	Zhe Yu, Jinye Peng. "Adaptive multi-information distillation network for image dehazing", Multimedia Tools and Applications, 2023 Publication	<1 %
19	Submitted to Uttar Pradesh Technical University Student Paper	<1 %
20	Weichao Yi, Liquan Dong, Ming Liu, Mei Hui, Lingqin Kong, Yuejin Zhao. "Priors-assisted dehazing network with attention supervision and detail preservation", Neural Networks, 2024 Publication	<1 %

Exclude quotes	Off	Exclude matches	< 14 words
Exclude bibliography	On		