Image and Video Dehazing Using Advance Deep Learning Algorithms

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

Harsh Thakran (201534)

Rahul Sharma (201508)

Under the guidance & supervision of

Dr. Vipul Kumar Sharma (Assistant Professor (SG))

Department of Computer Science & Engineering and Information Technology

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

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled **"Image and Video Dehazing using Advance Deep Learning Algorithms"** 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 our own work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Vipul Kumar Sharma, Assistant Professor(SG), 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.

Harsh Thakran	Rahul Sharma
201534	201508

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

Dr. Vipul Kumar Sharma

Assistant Professor (SG)

Department of Computer Science & Engineering and Information Technology

Dated:

CERTIFICATE

This is to certify that the work presented in this report entitled **"Image and Video Dehazing using Advance Deep Learning Algorithms"** 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 work carried by **Harsh Thakran (201534)** and **Rahul Sharma (201508)** out over a period from August 2023 to May 2024 under the supervision of **Dr. Vipul Kumar Sharma, Assistant Professor(SG), Department of Computer Science & Engineering and Information Technology**.

Harsh Thakran	Rahul Sharma
201534	201508

The above statement made by the candidate is true to the best of my knowledge.

Dr. Vipul Kumar Sharma
Assistant Professor (SG)
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat

ACKNOWLEDGEMENT

We are highly indebted to all the members of Department of Computer Science and Engineering and Information Technology, Jaypee University of Information Technology for their constant supervision and support, yet providing all the necessary data related to this project and additionally for their support in finishing the project. We would also like to express our heartfelt gratitude towards Dr. Vipul Kumar Sharma, Assistant Professor(SG), as our mentor and project supervisor, for his very kind cooperation and invaluable advices which helped us overall in completion of this project and for giving us such attention and time when we needed it the most.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
NYU2	New York depth dataset
GPU	Graphics Processing Unit
ResNet	Residual Neural Network
SOTS	Synthetic Objective Testing Set
RESIDE	REalistic Single Image DEhazing
RGB	Red,Green and Blue
VS Code	Visual Studio Code
SSD	Solid State Drive
RAM	Random Access Memory
CPU	Central Processing Unit

ABSTRACT

In today's world of computer vision, restoration of images and videos sometimes gets degraded by some conditions such as atmospheric conditions, especially haze which becomes a significant challenge. Some conventional methods also often struggles with the complexities of manually designed haze-related features. This study will present a new and unique approach on image and video dehazing using a deep convolutional neural network (CNN) with a residual-based architecture which will provide a major solution for hazy images and videos. The proposed algorithm on dehazing tackles the inherent difficulties of single-image dehazing by dividing the network model into two distinct important phases. In the initial stage, the network will take a hazy image or video as input and estimates the transmission map. Subsequently, in the second stage, the ratio of the foggy images/video frames and the transmission map serves as input for a residual network, effectively removing the haze. An important advantage of this approach is the ability to bypass the estimation of atmospheric lights, thus enhancing the overall efficiency of the dehazing process. We are using NYU2 depth dataset to train the network, and the experimental results evaluated through various metrics, which will result in the efficiency and robustness of the proposed method. In conclusion, this research harnesses the power of deep neural networks to address the challenges associated with manually designed features for image and video dehazing. The proposed two-stage network model demonstrates effectiveness and robustness, showing promising results in image and video dehazing. By eliminating the need for explicit estimation of atmospheric light, the algorithm contributes to the efficiency of the dehazing process and holds potential for real-world applications where atmospheric conditions hinder visual perception.

CHAPTER - 1
INTRODUCTION

1.1 Introduction

Let’s imagine this scenario in our mind where light interacts with small water droplets or countless tiny particles which are suspended in the air and they creates a phenomenon known as haze. This haze, is commonly seen in different weather conditions which does more than just reproduce, an attractive images and videos. It leads to some significant problems such as colour distortion, low contrast and poor saturation. Thus the impact extends beyond aesthetics and presents some challenges for optical imaging instruments used in systems such as satellite remote sensing, aerial photography and outdoor monitoring. Overcoming these challenges is very important and essential for these systems to operate effectively. So, to address this issue, there is an urgent need for techniques that can improve the quality of images and videos affected by haze and reduce the impact on outdoor and indoor imaging systems.

There are some existing methods which can be categorised into two major groups which are image processing based enhancement and another one is physical model based restoration. While the first one is very well established and very well efficient, it enhances images and videos without fully understanding the underlying cause of degradation by enhancing contrast and brightness to improve visual appeal but it falls short in scenarios with wearing depths or complex changes. Also, it does not tackle the reduction in the fog quality, thus resulting in images with poor definition and a distorted appearance which is not favourable. This is where our Residual based deep CNN algorithm comes in which is according to us will be an Advanced solution that goes beyond mere enhancement, aiming to eliminate fog and restore images and videos to their original clarity as far as possible.

The physical model-based restoration method is a type of method which analyses the specific causes of image and video frames degradation and establishes a degraded model of fog-degraded images. In past few years, they were significant advances which have been made in image processing methods based on physical models. The physical model can be described as :-

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

$$\boldsymbol{t(x) = e^{-\beta d(x)}} \tag{1}$$

where,

x is input of the image pixel point,

I(x) is original input haze image,

J(x) is dehazed and restored image ,

A is ambient atmospheric light value.

t(x) is optical path propagation map

d(x) is distance between the object and camera

where the *t(x)* undergoes exponential attenuation with the depth *d(x)* of the image.

β is atmospheric scattering coefficient, which represents the scattering capacity of light per unit volume of the atmosphere which will generally take a small constant.

1.2 Problem Statement

The Quality of images and videos is crucial in today's digital age, impacting fields such as photography. However, haze and fog sometimes can significantly degrades the clarity of visual media. Our project focuses on leveraging some advanced deep learning algorithms to enhance the images and videos.Our goal is to design CNN networks such that they can easily remove haze from all media.We also know the importance of real-time video dehazing. Therefore, our project will also ensure seamless frame-to-frame transitions while efficiently removing haze. Our project also aims to bridge the gap where is there between theoretical advancements and practical applications, offering a promising solution for real-world scenarios where atmospheric conditions gives trouble in visual perceptions. Our main project aims to tackle the issue of haze and fog by using advanced deep learning algorithms. We plan to enhance the image and video dehazing by employing cutting-edge neural network designs. Upon the completion of this project, it will have the potential to transform how we experience and engage with visual media in foggy and hazy settings.

1.3 Objectives

The primary objective of our project is to eliminate or minimise 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 poses 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 utilising 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 revolutionise image and video dehazing, pushing the boundaries of what can be achieved through human ingenuity and computational power. The strategic utilisation of the NYU2 depth dataset is delibrate 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 landscapes. Ultimately, the project aspires clarity into our visual communication.

1.5 Organization of Project Report

In the beginning, the project report unfolds as a organised 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

S. No	Paper Title [Cite]	Journal (Year)	Tools/ Techniques/ Dataset	Results	Limitation
[1]	Real-time image and video dehazing based on multiscale guided filtering	Journal of Multimedia Tools Applications (2022)	Algorithm:- Fast guided filter. Dataset:- Synthetic and real world hazy images.	Paper offers better results to state-of-the-art methods with higher computational efficiency.	Parameter sensitivity, accuracy in physical model estimation.
[2]	Single img dehazing using improved cycleGAN	Journal of Visual Communication (2021)	Algorithm:- CycleGAN Dataset: The NYU2 depth dataset, RESIDE	Qualitative evaluations on various datasets.	Training time is more.
[3]	A survey on video Dehazing using deep learning	Journal of Physics: Conference Series (2020)	Algorithms:- GAN,Rain Density Classifier Dataset:-NYU Depth V2	Paper discusses dehazing algos for enhancing visibility.	Dataset and Algorithms are limited
[4]	Single image haze removal using GAN	International Conference of WiSPNET (2020)	Algorithms:- cGAN Dataset:- NYU Depth dataset, SOTS	The model performs well on the custom dataset and competes on the SOTS dataset	Small dataset may affect the complexity of the haze in the input image

Table:- 1 Literature Survey

S. No	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitation
[5]	Single image dehazing:An analysis on generative adversarial network	Journal of Computer and Communication (2020)	Algorithms:- cGAN Dataset:- Synthetic Objective Testing Set	It discusses Image Dehazing using GANs comparing AODNet, cGAN, outperforms others.	Performance may vary depending on the complexity of the hazy images
[6]	Benchmarking, single-image dehazing and beyond	IEEE Transactions on Image Processing (2019)	Algorithm:- RESIDE Benchmark Dataset:- Synthetic Outdoor	Paper discusses creation of the RESIDE dataset	Need for more diverse and realistic datasets
[7]	Deep Video Dehazing with semantic segmentation	IEEE Transactions on Image Processing (2019)	Algorithm:- Video Dehazing Network Dataset:- NYU Depth dataset	The algorithm successfully removes haze and enhances image visibility in videos.	Bias training data
[8]	Single image dehazing via nin-dehazenet	IEEE Access (2019)	Algorithm:- NiN-DehazeNet Dataset:- Synthetic dehazing image	It concluded that NIN-DehazeNet could get the best defogging effect.	Require large dataset.

Table:- 2 Literature Survey

S. No	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitation
[9]	Image Dehazing Using Residual-Based Deep CNN	IEEE Access (2018)	Algorithm:- Deep CNN Dataset: The NYU2 depth dataset	It compares dehazing methods using subjective and objective measures	Indoor dataset bias and high complexity.
[10]	Single image dehazing via Deep Learning-based image restoration	APSIPA ASC (2018)	Algorithm:- DehazeNet Dataset:-RESIDE Dataset	Algo generalizes well on real hazy images, preserves details and color, and improves iteratively.	Dehazing from a single image is challenging.
[11]	Single image dehazing via multi-scale convolutional Neural Networks	Computer Vision-ECCV (2016)	Algorithm:- Stochastic gradient descent Dataset:- NYU Depth dataset	Paper results of a multi-scale network for single-image dehazing	-
[12]	Optimized contrast enhancement for real-time image and video dehazing.	Journal of Visual Communication (2013)	Algorithm:- Unsharp masking Dataset:- Newyork1, Newyork2,Mountain, Riverside, Roadview	It doesn't provide any quantitative or qualitative results of the proposed algorithm	Difficulty in restoring densely hazy images. Sensitive to object brightness

Table:- 3 Literature Survey

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 generalisation of models to real-world scenarios

- **Difficulty in handling hazy images:-** The paper on optimised 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 works better on smaller datasets but some algorithms needs 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 which are needed are mentioned below :-

3.1.1 Functional Requirements

- Haze removal accuracy
  - The system in which we are running our model must effectively and accurately remove haze from both images and videos to enhance visibility.
  - The degree of haze reduction should be adjustable to varying levels of haziness in input data.

- Image upload and processing
  - The software which we will be needing should be able to handle the given uploaded images and also it should be able to save those images for future processing.
  - We must be able to process the uploaded image into our model and further use it for extracting the necessary features and generate a desirable output.

- Image Surrounding identification
  - The model which we will be using should be able to D is the image irrespective of the environment in which it has been taken i.e., Outdoor or Indoor.
  - Our preferred model should be able to adjust according to the type of image and after analysing that image, the model should be able to process and dehaze it and further generates a desirable output.

- Real-time processing
  - The system in which we are running our model should be able to process the image and produce the output in minimum time span as much as possible, thus resulting in increasing its overall usability.
  - The system in which we are running our model should have a high GPU and RAM so that the model gets trained in minimum time as much as possible for overall faster runtime.

- Training mode
  - The system in which we are running, our model should be able to retrain the model with new data when available in minimum time as much as possible.

3.1.2 Non Functional Requirements

- Performance
  - The system should be able to dehaze the input image under two seconds of time.

- Reliability
  - The system in which our model is running 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 front-end frameworks in Python, like flask, for the development of a website, where the users can upload the hazed images or videos and they will get the clear image or video from there.

- Maintainability
  - The CodeBase of the model and the front-end should follow the industries best practices to increase the readability and understandability.

- Usability
  - The user interface should be user friendly, minimising the learning curve for users interacting with our dehazing system.

- Training Dataset
  - We have used to separate data sets for training and testing of our model for better results in both image and video dehazing.
  - The dataset which we have used includes both indoor and outdoor images for our model.

- Image Resolution
  - Our model supports for all common types of the image formats like JPEG, PNG, JPG.

- Validation and testing
  - Testing on the testing data said and producing the desirable output.

- Resource Utilisation
  - The system should utilise system resources very efficiently so that it can optimise CPU and memory usage to prevent them from excessive consumption and maintaining overall system stability.

- Accuracy
  - Algorithm must produce accurate results across different outputs, ensuring reliability in the outputs.

- Adaptability
  - The system should be adaptable to changes in the hardware configurations, software updates without compromising performance or functionality.

3.1.3 Hardware Requirements

- GPU(Graphics Processing Unit)
  - High performance GPUs are needed for efficient training and computations for deep learning processes. Eg-NVIDIA Maxwell GPU.

- CPU(Central Processing Unit)
  - 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(Random Access Memory)
  - A good amount of memory RAM is required such as 32GB LPDDR4 is required for larger datasets and more complex models.
  - Generally RAM requirement depends upon the size of the input data and the complexity of the neural network model.

- Storage
  - A sufficient amount of storage space is necessary for storing the applications, model parameters and large data sets.
  - SSDs(Solid State Drives) are preferred nowadays for faster. Read and write speeds.

- Cooling Systems
    - We easily know that the deep learning tasks can generate significant heat especially during longer usage. To ensure that the system has an effective cooling system is necessary to prevent overheating so that the system maintains a consistent performance as needed.

- Power Supply
    - Power supply also plays a very vital role to meet the power demands of the GPU and other system components.
    - We must need a stable power supply for consistent performance.

3.1.4 Software Requirements

- Language
    - We must ensure that python is installed on the system as most deep learning, frameworks and libraries are a Python-based.

- Deep learning Frameworks
    - We must choose suitable deep learning frameworks like TensorFlow and Keras for the training and testing of the models.

- Version control
    - We will be using Git for the budget control and we will be pushing our called into Github.

- Development environment
    - Jupyter notebook and VS code for the development environment which we will be using for training and testing of our deep learning model.

3.2 Project Design and Architecture

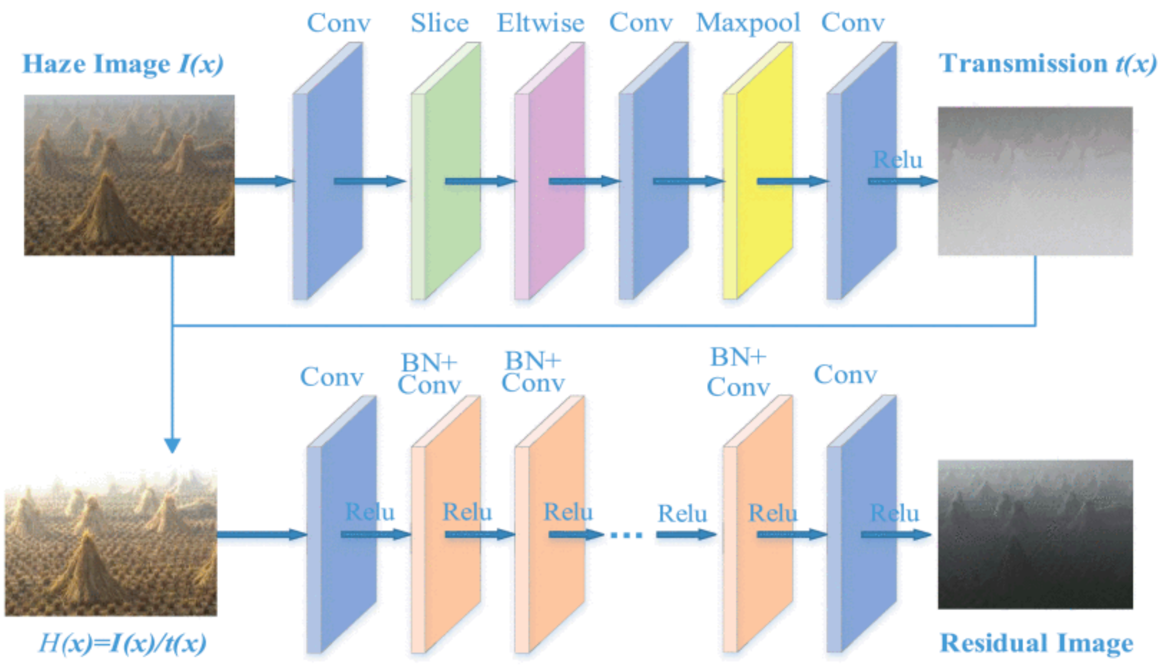


Figure:- 1 Network Architecture [7]

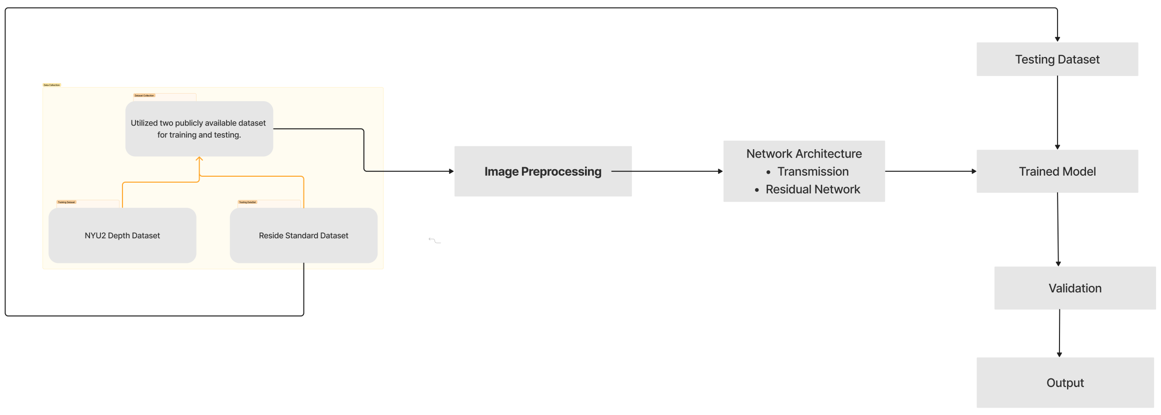


Figure:- 2 Project Design

3.3 Data Preparation

3.3.1 Data Collection

After observing many pre existing similar types of models and many few research papers we have concluded that many of them are using publicly available datasets such as NYU2 Depth Dataset and Reside Dataset

We have decided that we will be using separate datasets for training and testing of the model such as the following below:-

1. **NYU2 Depth Dataset - (Training Dataset)**

- This dataset comprises of video sequences from a different varieties of indoor scenes which were recorded and captured by the RGB and depth cameras.
- This dataset includes:-

Number of Images	
1449	Densely labeled pairs of RGB and depth images
464	New scenes being taken from 3 cities
407024	New unlabelled frames

Table:- 4 NYU2 depth dataset details

2. **RESIDE Dataset - (Testing Dataset)**

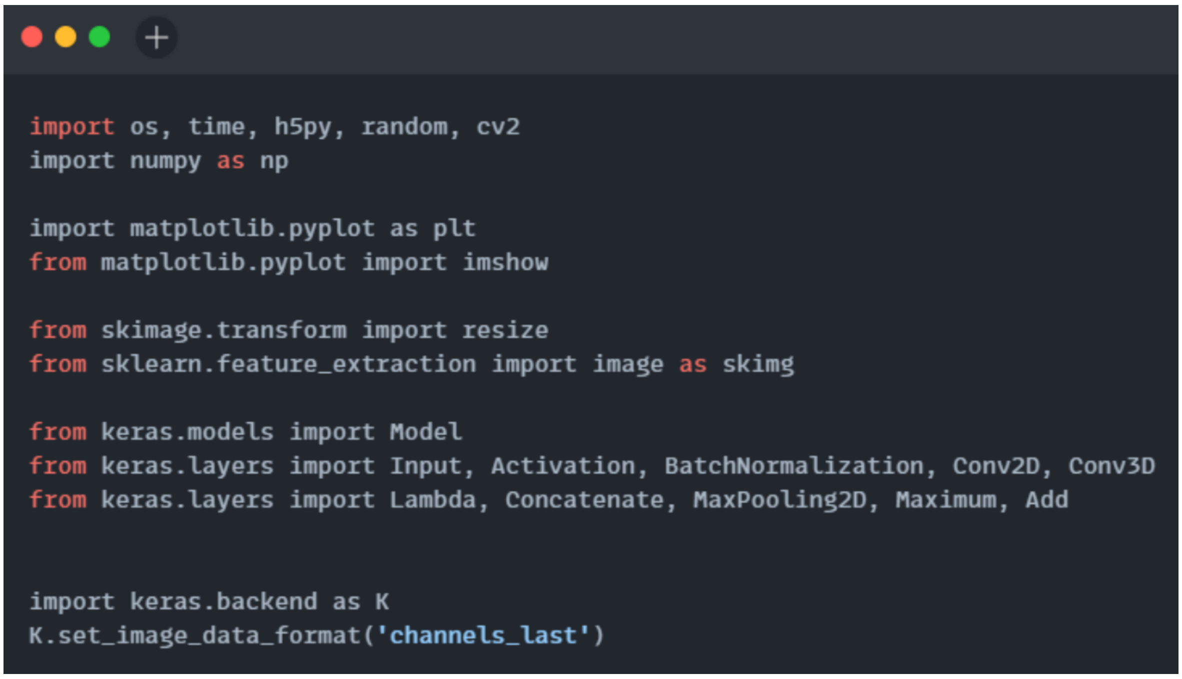
- This dataset includes images with realistic looking scenes under the effect of haze.
- This dataset includes both indoor and outdoor images.
- Each image in this dataset is accompanied by a corresponding truth image
- This dataset is being used by the researchers and developers to evaluate and contrast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

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

Table:- 5 RESIDE dataset details

3.3.2 Data Preprocessing

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

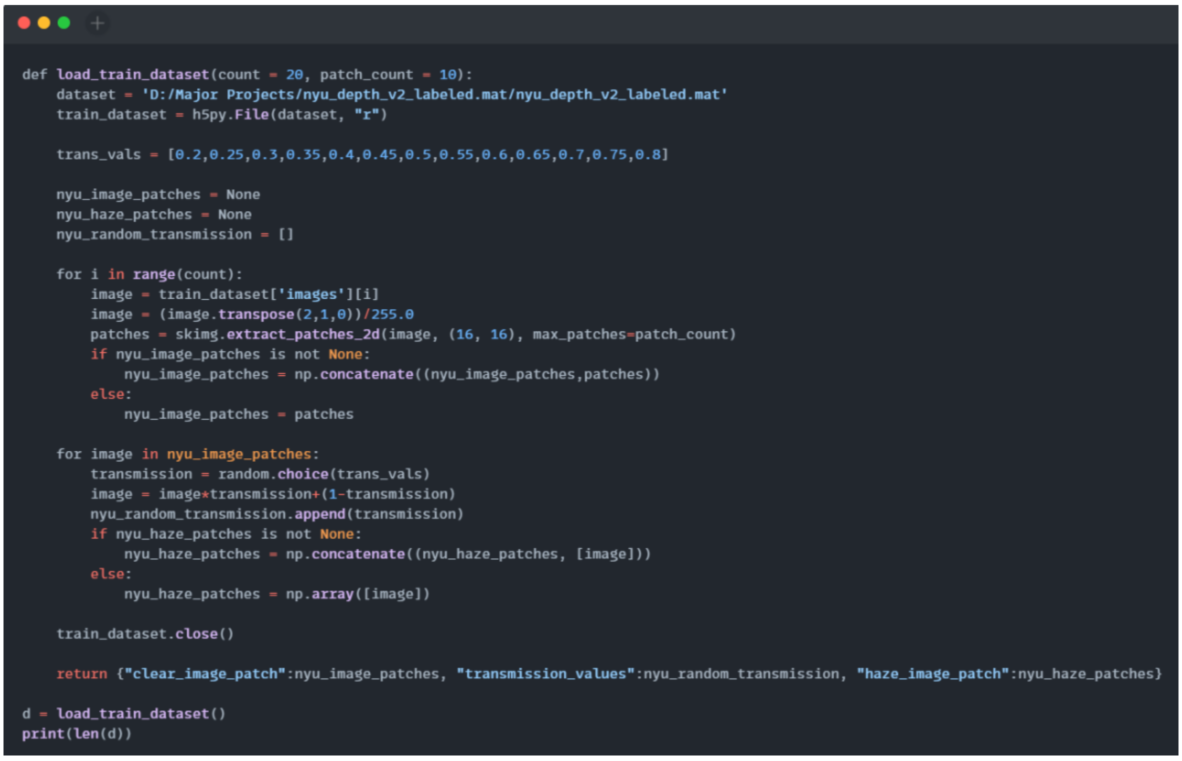


```
import os, time, h5py, random, cv2
import numpy as np

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')
```

Figure:- 3 Importing necessary python libraries

Training Dataset

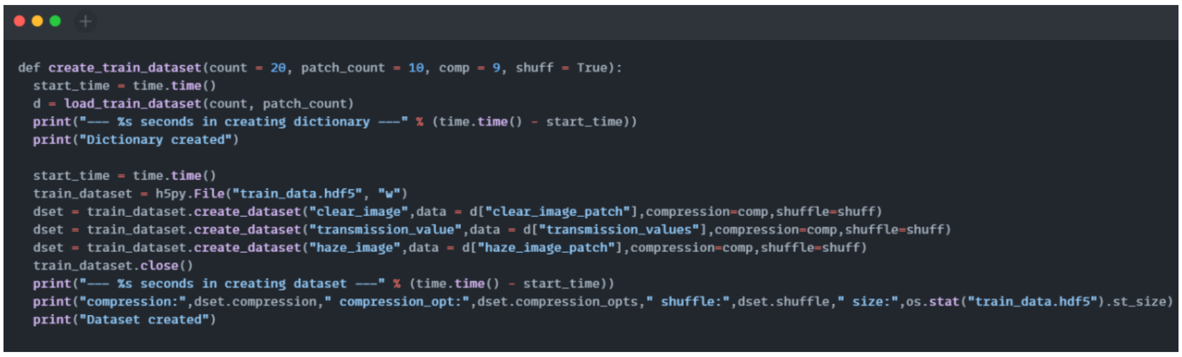
In this code we are importing all the python libraries which are necessary to load and process the data.



```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))
```

Figure:- 4 Function to load train dataset from NYU2 dataset

Here in this code we Are writing a function which is basically used to load the training data set from the NYU2 depth dataset.. Here, we are also doing normalising the data where the image is divided by 255.0, and the 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 will also extract patches of size 16 x 16 pixels, and for each extracted path a random transmission value is chosen and then the image is modified to simulate haze with the randomly chosen values.It

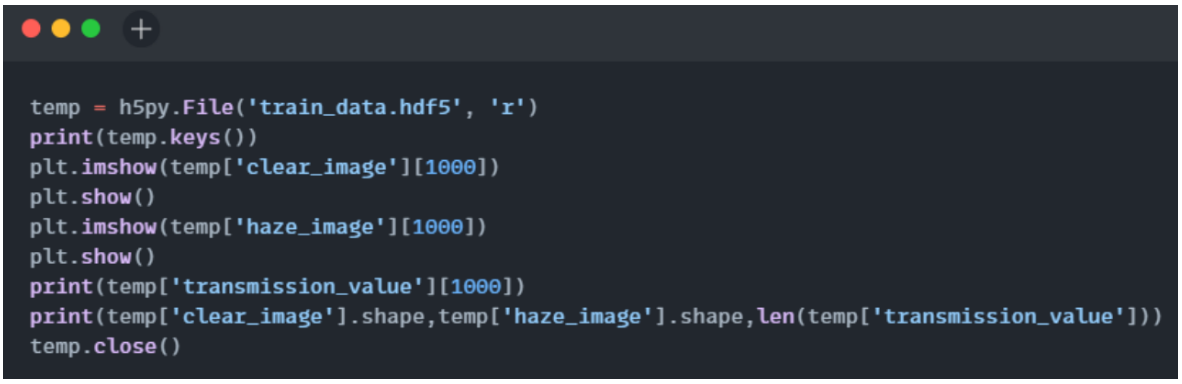


```python
def create_train_dataset(count = 20, patch_count = 10, comp = 9, shuff = True):
  start_time = time.time()
  d = load_train_dataset(count, patch_count)
  print("--- %s seconds in creating dictionary ---" % (time.time() - start_time))
  print("Dictionary created")

  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")
```

Figure:- 5 Creating Train dataset using NYU2 Depth dataset

also created a (mj.hdf5) file for the NYU2 dataset in the directory.

Here in this code we are creating train dataset using NYU2 depth dataset. Firstly, we are loading the dataset using the previous function ‘load_train_datset()’, and after loading we are creating a hdf5 file for the training data inside which we are creating 3 datasets- clear

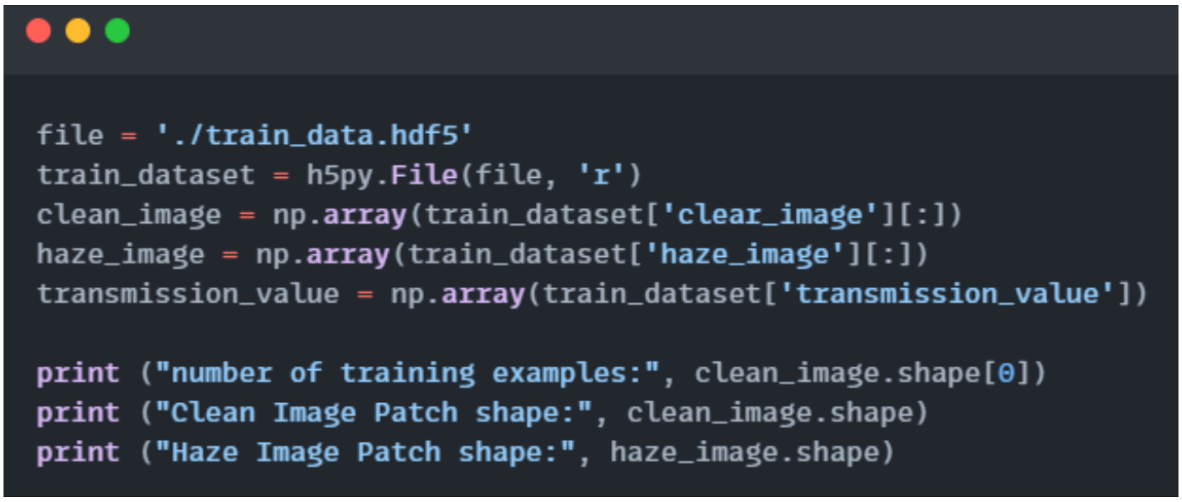


```
temp = h5py.File('train_data.hdf5', 'r')
print(temp.keys())
plt.imshow(temp['clear_image'][1000])
plt.show()
plt.imshow(temp['haze_image'][1000])
plt.show()
print(temp['transmission_value'][1000])
print(temp['clear_image'].shape,temp['haze_image'].shape,len(temp['transmission_value']))
temp.close()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

Figure:- 6 Checking and validating created dataset

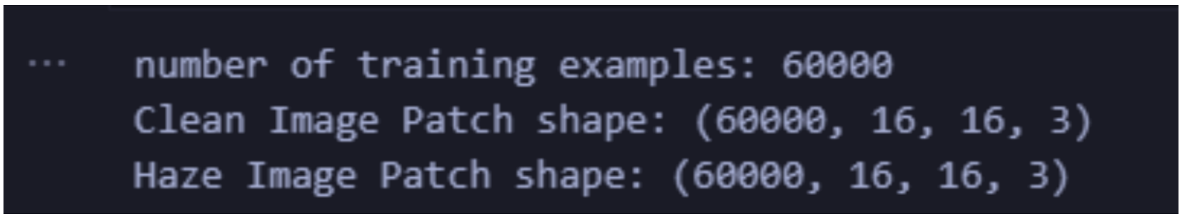
images, trans value and hazy images.

Here in this cold, we are displaying the 1000th clear image, haze image and their associated transmission value and the shapes of clear



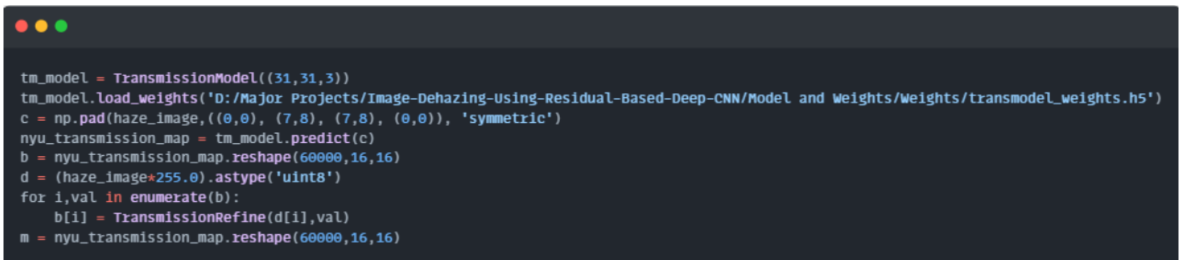
```
file = './train_data.hdf5'
train_dataset = h5py.File(file, 'r')
clean_image = np.array(train_dataset['clear_image'][:])
haze_image = np.array(train_dataset['haze_image'][:])
transmission_value = np.array(train_dataset['transmission_value'])

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)
```

Figure:- 7 Printing the information of loaded datasets



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

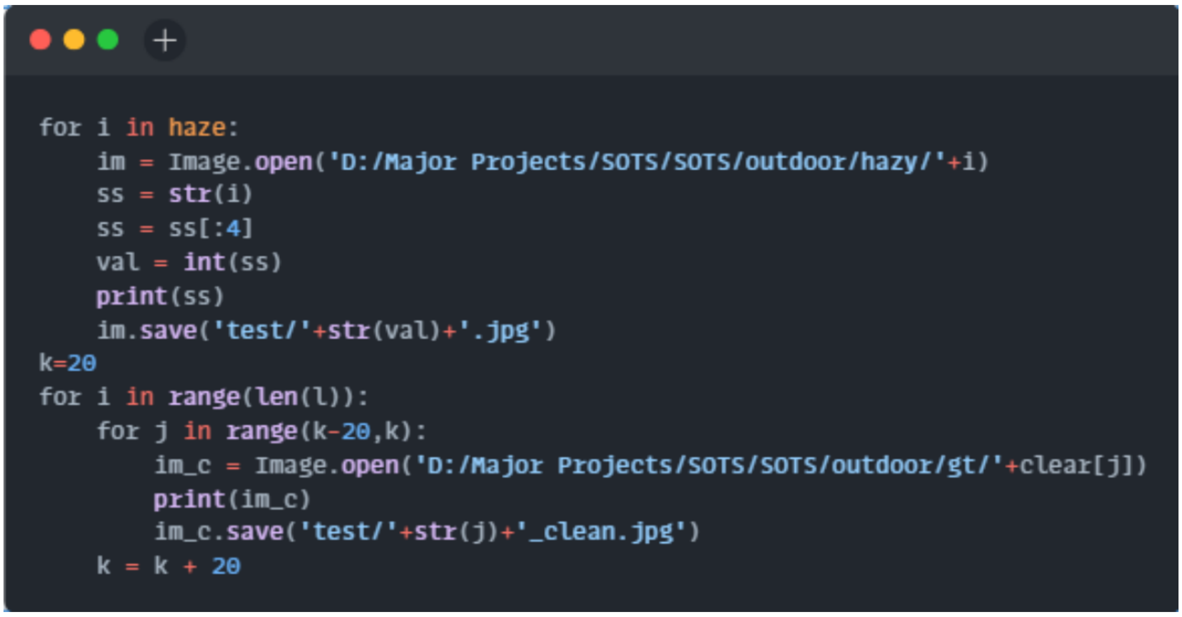
Figure:- 8 Output information of loaded datasets



```
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)
```

Figure:- 9 Refinement of Transmission Maps

In this piece of code 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.

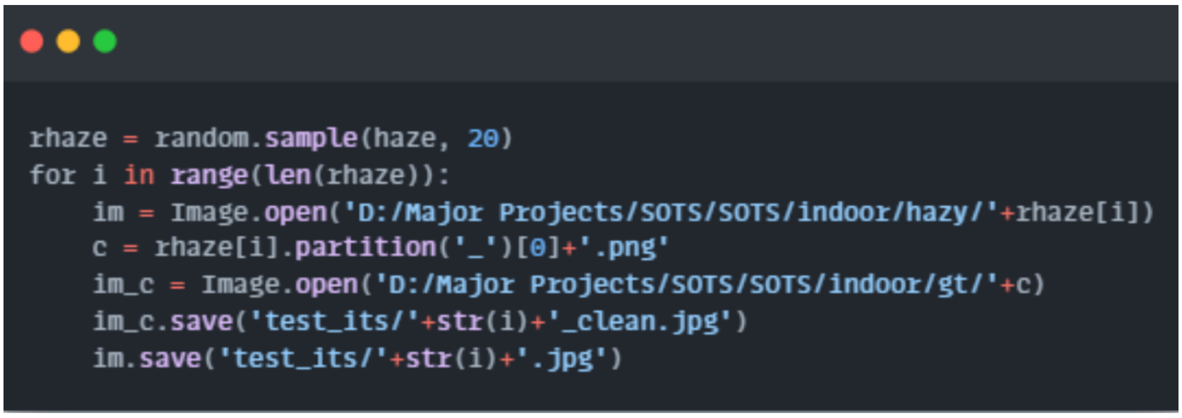


```
for i in haze:
    im = Image.open('D:/Major Projects/SOTS/SOTS/outdoor/hazy/'+i)
    ss = str(i)
    ss = ss[:4]
    val = int(ss)
    print(ss)
    im.save('test/'+str(val)+'.jpg')
k=20
for i in range(len(l)):
    for j in range(k-20,k):
        im_c = Image.open('D:/Major Projects/SOTS/SOTS/outdoor/gt/'+clear[j])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

Figure:- 10 Extraction and processing of outdoor image dataset

Testing Dataset

In this code, we are creating the outside images from the SOTS subset of the RESIDE dataset, saving it with the name of hazy as well with the 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')
```

Figure:- 11 Extraction and processing of indoor image dataset

Here in this code we are doing the same work for the indoor images from SOTS subset of the RESIDE dataset.

3.4 Implementation

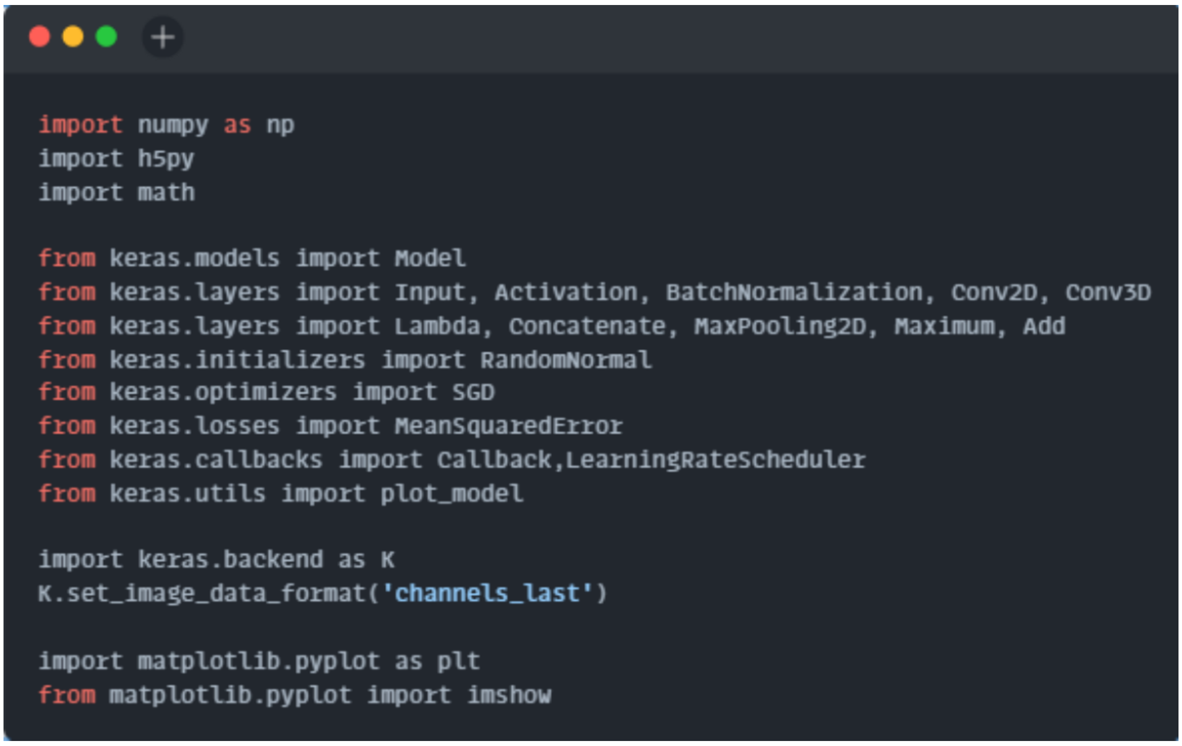
For the implementation part we will be using the Residual-Based Network.

Residual-Based Network for Image Dehazing

It is a type of CNN architecture, which was created to solve the problems of the vanishing and exploding gradient issues that frequently arises with the deeper networks. Raise, it will be still learning as the following key features which are as follows:-

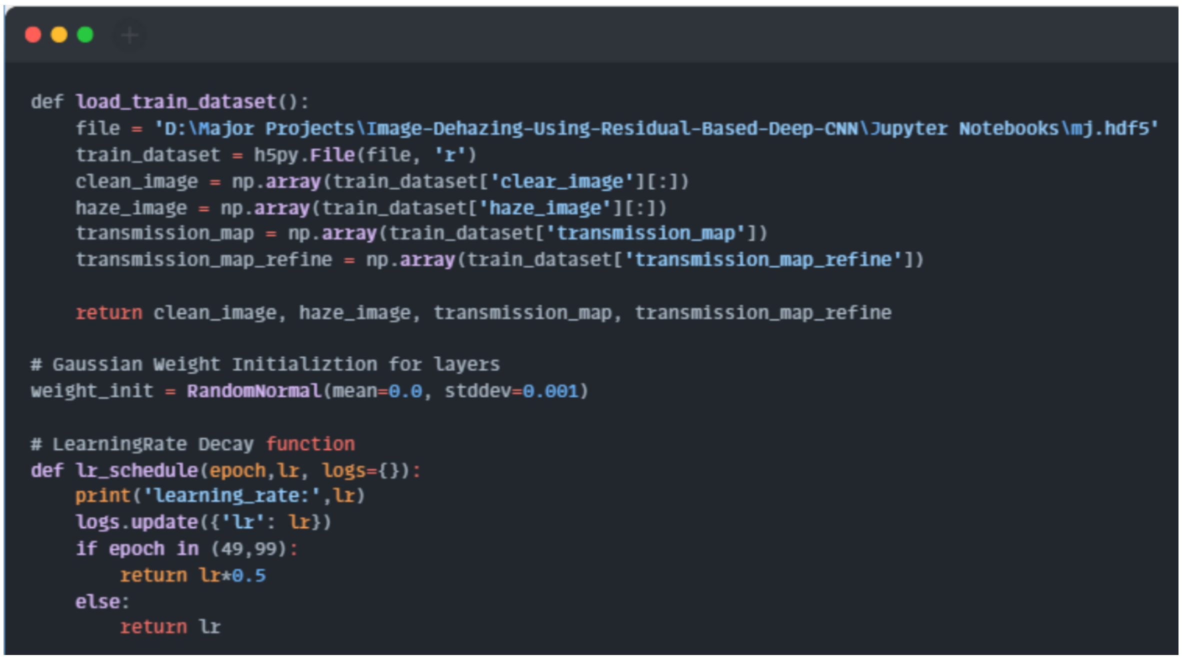
- **Residual Learning:-** It uses residual blocks which are also known as skip connection is an essential component of a ResNet. It also facilitates the optimisation of deep networks by enabling activations to bypass one or more layers via a shortcut link.
- **Identify Shortcut Connections:-** It introduced us with the concept of bypassing or skipping one or more layers. Primary idea behind this is that it is easier to optimise residual mapping which is the difference between the input and output rather than the desired output.

- **Deep architecture :-** Using these residual blocks, ResNet can be built with 50, 101 and 152 layers, but more deeper variants are available.



```python
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
```

Figure:- 12 Importing Libraries

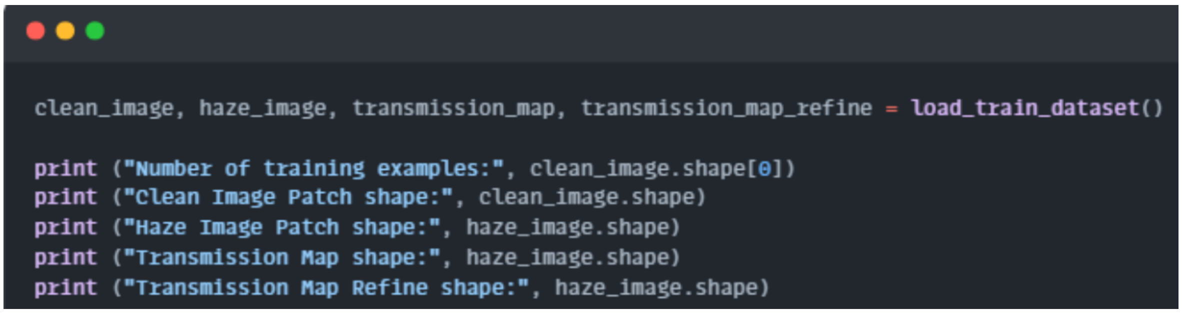


```python
def load_train_dataset():
    file = 'D:\Major Projects\Image-Dehazing-Using-Residual-Based-Deep-CNN\Jupyter Notebooks\mj.hdf5'
    train_dataset = h5py.File(file, 'r')
    clean_image = np.array(train_dataset['clear_image'][:])
    haze_image = np.array(train_dataset['haze_image'][:])
   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

Figure:- 13 Learning Rate decay schedule function

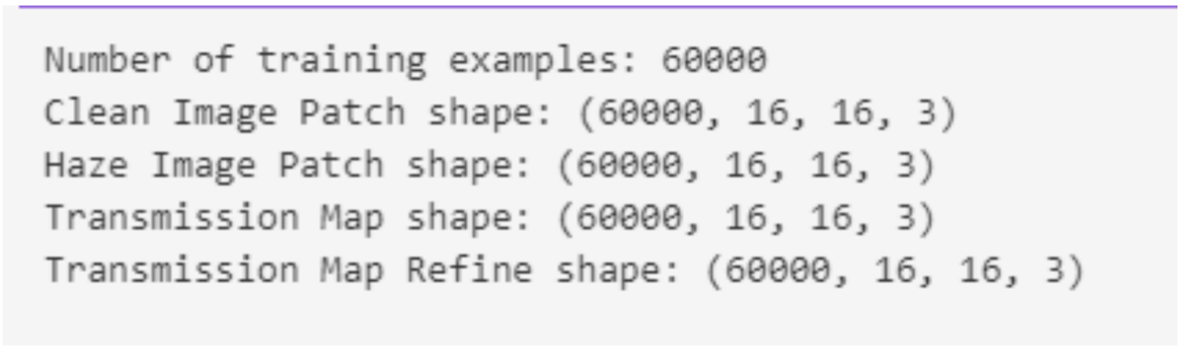
For this project, we are using the concept of learning residual information, representing the difference between the hazy and the clear images

Here in this code we are initially loading the trained dataset from the mj.hdf5 file created earlier. After that, we defined the weight initialisation 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.



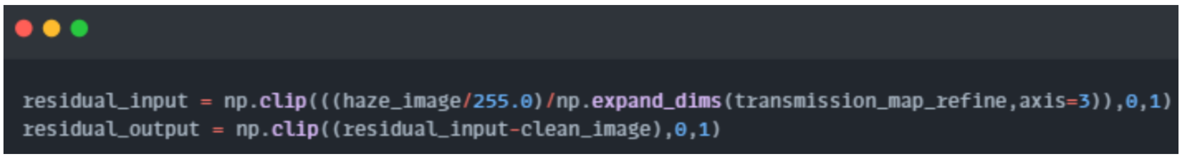
```
clean_image, haze_image, transmission_map, transmission_map_refine = load_train_dataset()

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)
```

Figure:- 14 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 Map shape: (60000, 16, 16, 3)
Transmission Map Refine shape: (60000, 16, 16, 3)
```

Figure:- 15 Training dataset information

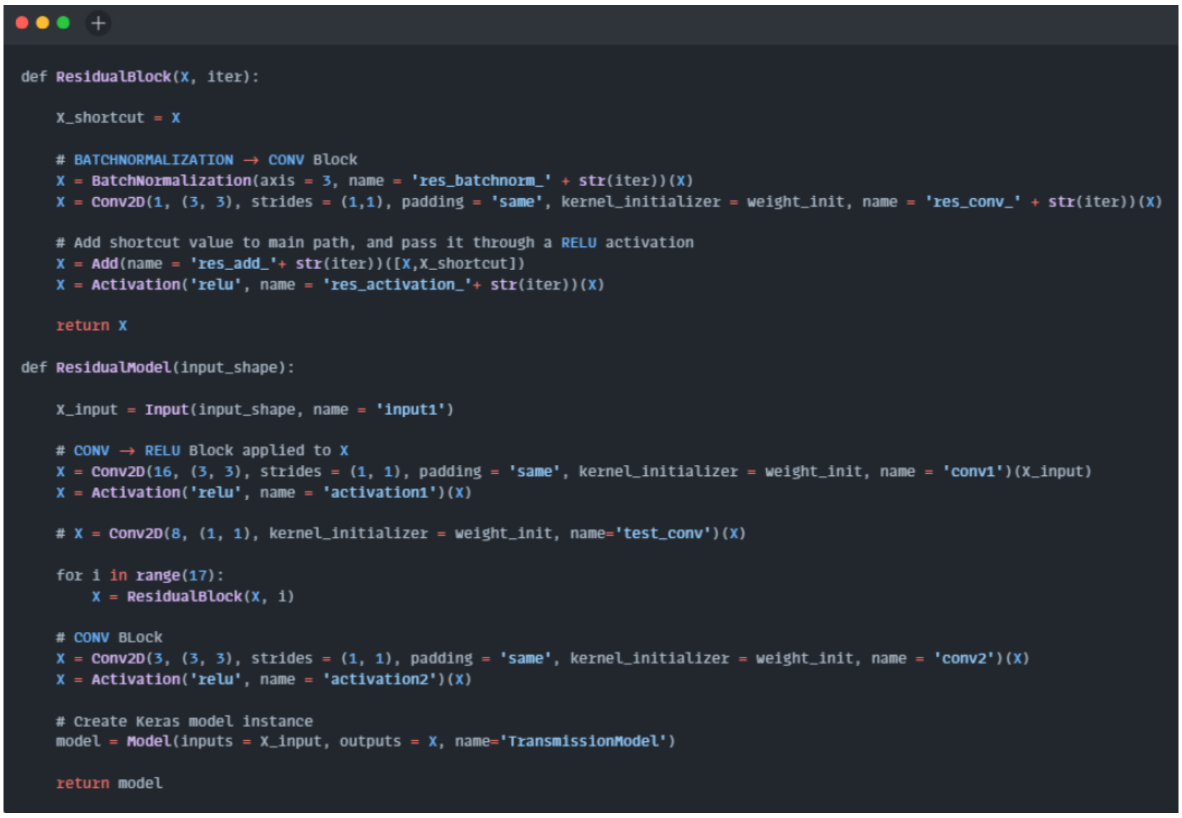


```
residual_input = np.clip(((haze_image/255.0)/np.expand_dims(transmission_map_refine,axis=3)),0,1)
residual_output = np.clip((residual_input-clean_image),0,1)
```

Figure:- 16 Computation of residual_input and residual_output

Here in this code we are displaying the information of the data set.

Here in this code firstly we computed the residual_input, which was computed by normalising the haze image. Similarly the residual_output is computed by subtracting the clean images from the residual input, after which it was clipped to ensure it stays within the

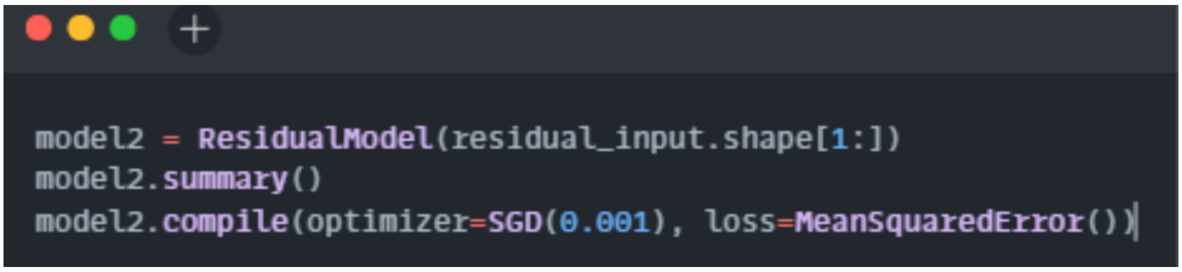


```
def ResidualBlock(X, iter):

    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
```

Figure:- 17 Residual- based Network model

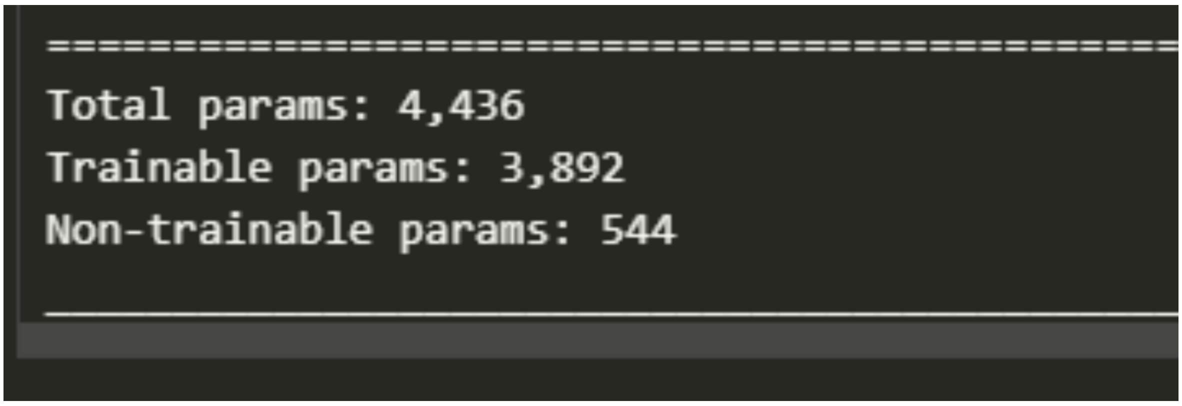
0 to 1 range. Residual_output represents the difference between residual_input and the clean image.

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



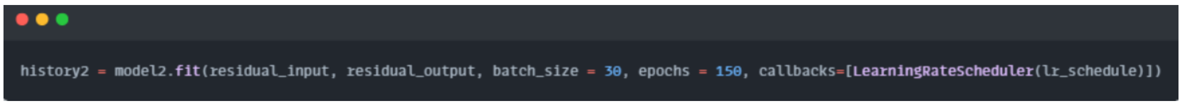
```
model2 = ResidualModel(residual_input.shape[1:])
model2.summary()
model2.compile(optimizer=SGD(0.001), loss=MeanSquaredError())
```

Figure:- 18 Compiling the residual model



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

Figure:- 19 Count of trainable and non-trainable parameters



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

Figure:- 20 Training the residual based network model

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

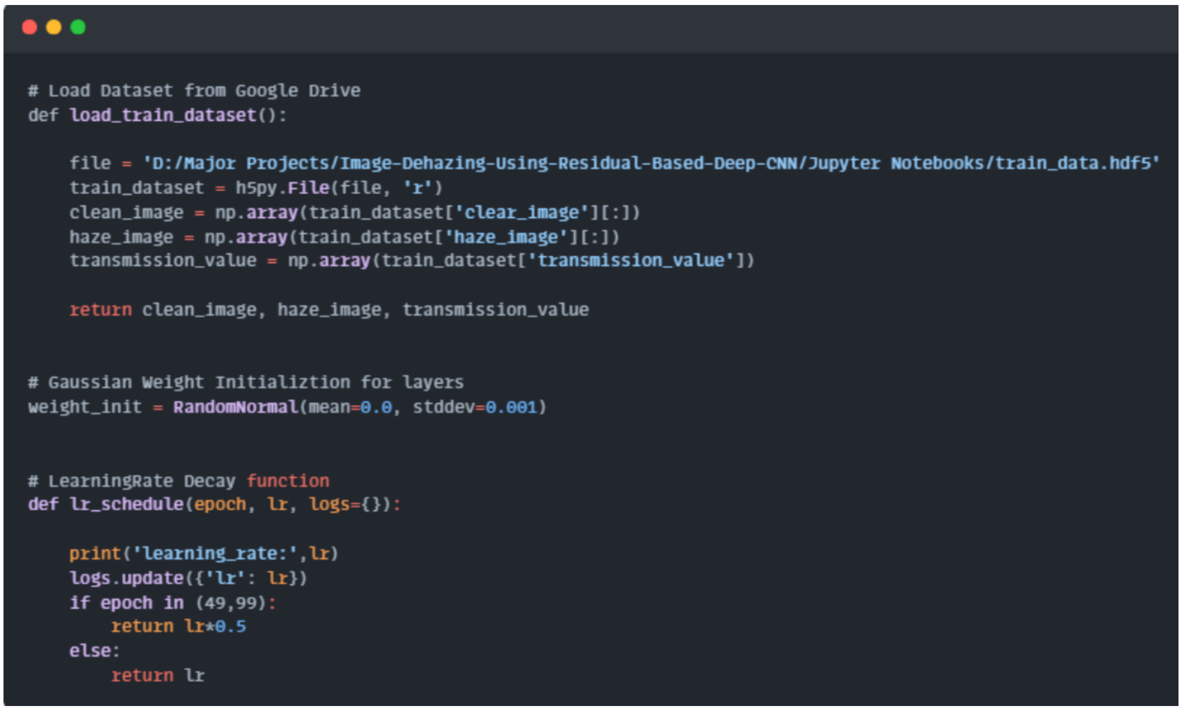
Here, we have created and compiled our Residual Based Network model consists of 1 convolutional block, 17 residual blocks.

Here we will be training the residual model with 150 Epochs, 30 batch size and a learning rate scheduler which will be utilised to dynamically adjust the Learning rate during training.

Transmission Network Model

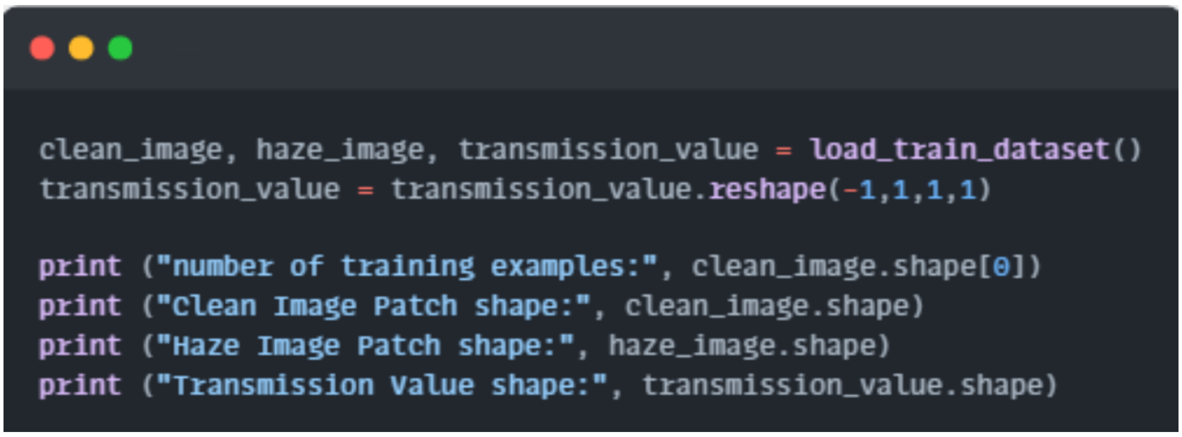
The transmission network model is a type of neural network which is utilised to estimate the transmission maps in images. It's key features are as follows:-

- **Input-output mapping :-** The primary function is to map input, data and its corresponding output data.
- **Supervised or unsupervised learning :-** It supports both supervised as well as unsupervised learning.
- **Loss functions:-** While training transmission network models, it can optimise there parameters by minimising loss function
- **Robustness:-** These are robust.



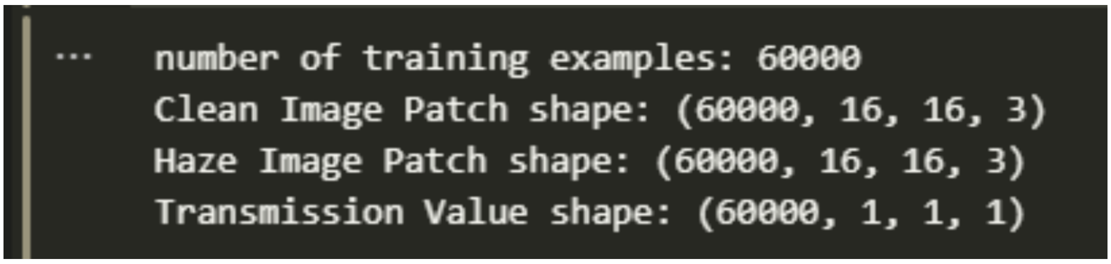
```
# Load Dataset from Google Drive
def load_train_dataset():

    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
```

Figure:- 21 Learning rate decay schedule function



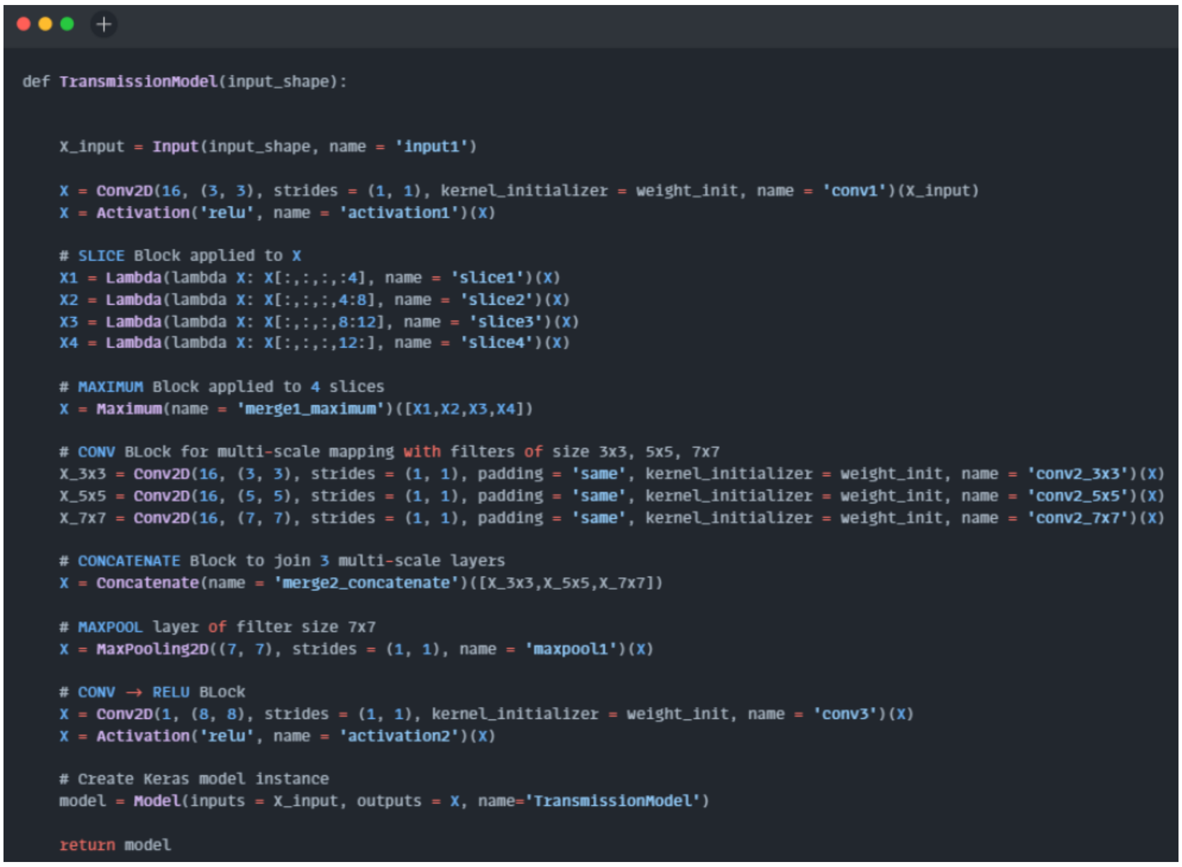
```
clean_image, haze_image, transmission_value = load_train_dataset()
transmission_value = transmission_value.reshape(-1,1,1,1)

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)
```

Figure:- 22 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)
```

Figure:- 23 Output of the above snippet



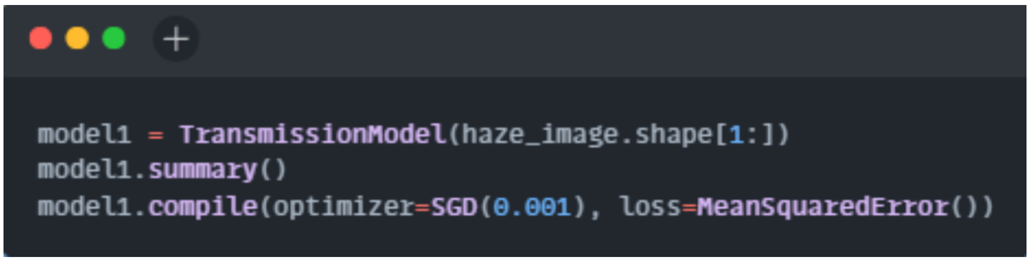
```
def TransmissionModel(input_shape):

    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
```

Figure:- 24 Creating neural networks to estimate transmission maps

Here in this code we are initially loading the trained dataset from the mj.hdf5 file created earlier. After that, we defined the weight initialisation 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.

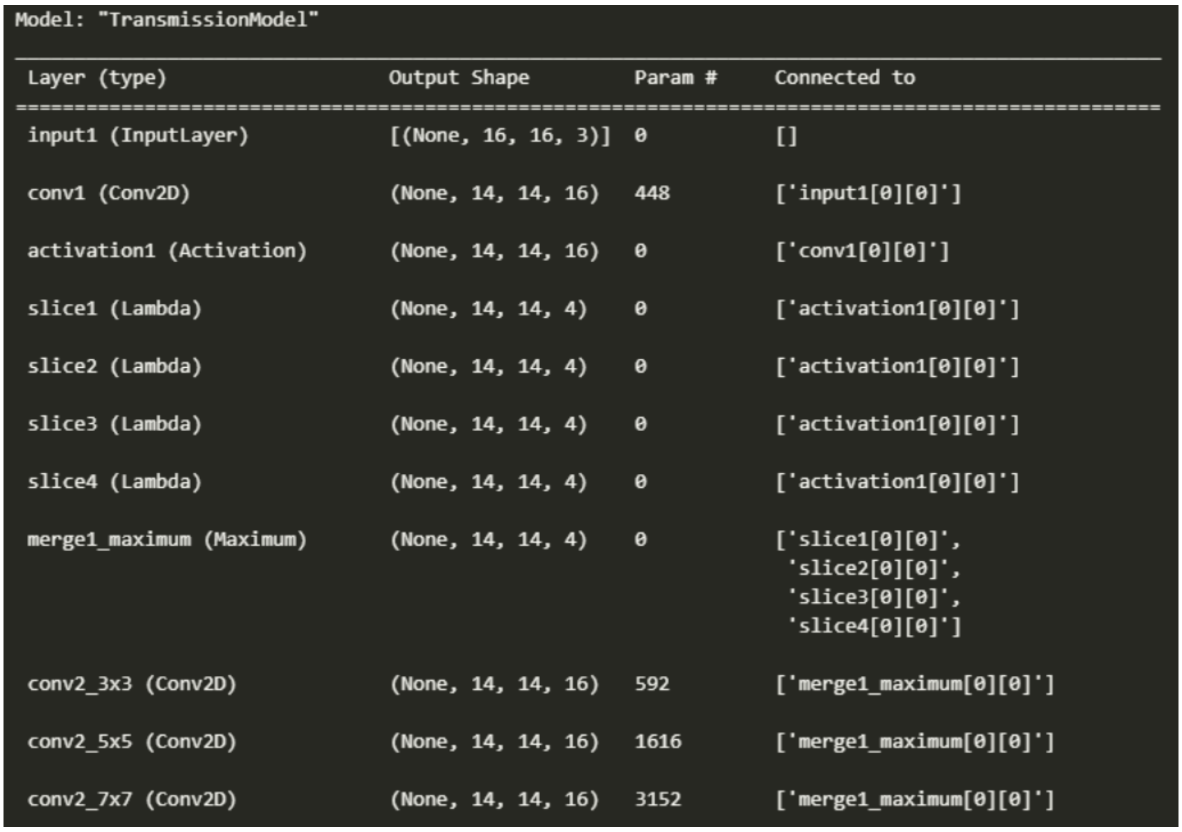
In this we Are constructing neural networks to estimate transmission maps between input and output. In this we are making a total of 18 layers which are :-

- 1 input layer
- 2 convolution layer
- 4 lambda layer
- 1 maximum layer
- 6 multiscale convolutional blocks
- 1 concatenate layer
- 1 maxPooling2D layer
- 2 Convolutional block



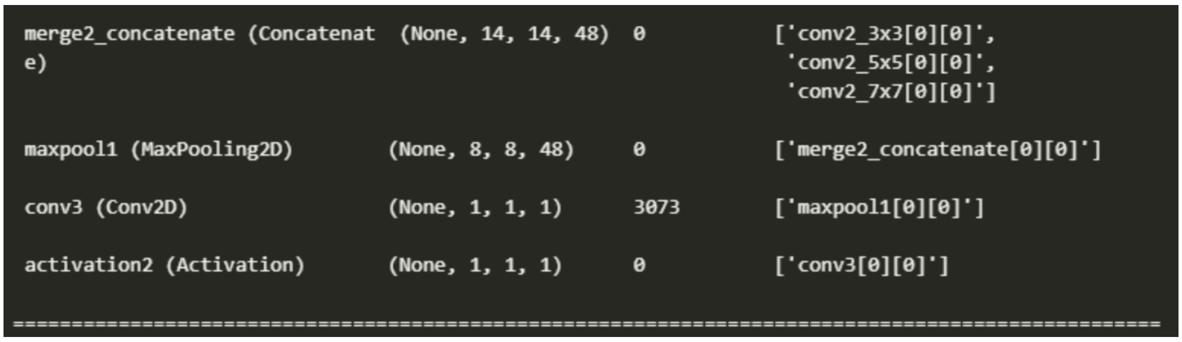
```
model1 = TransmissionModel(haze_image.shape[1:])
model1.summary()
model1.compile(optimizer=SGD(0.001), loss=MeanSquaredError())
```

Figure:- 25 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]']
```

Figure:- 26 [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]

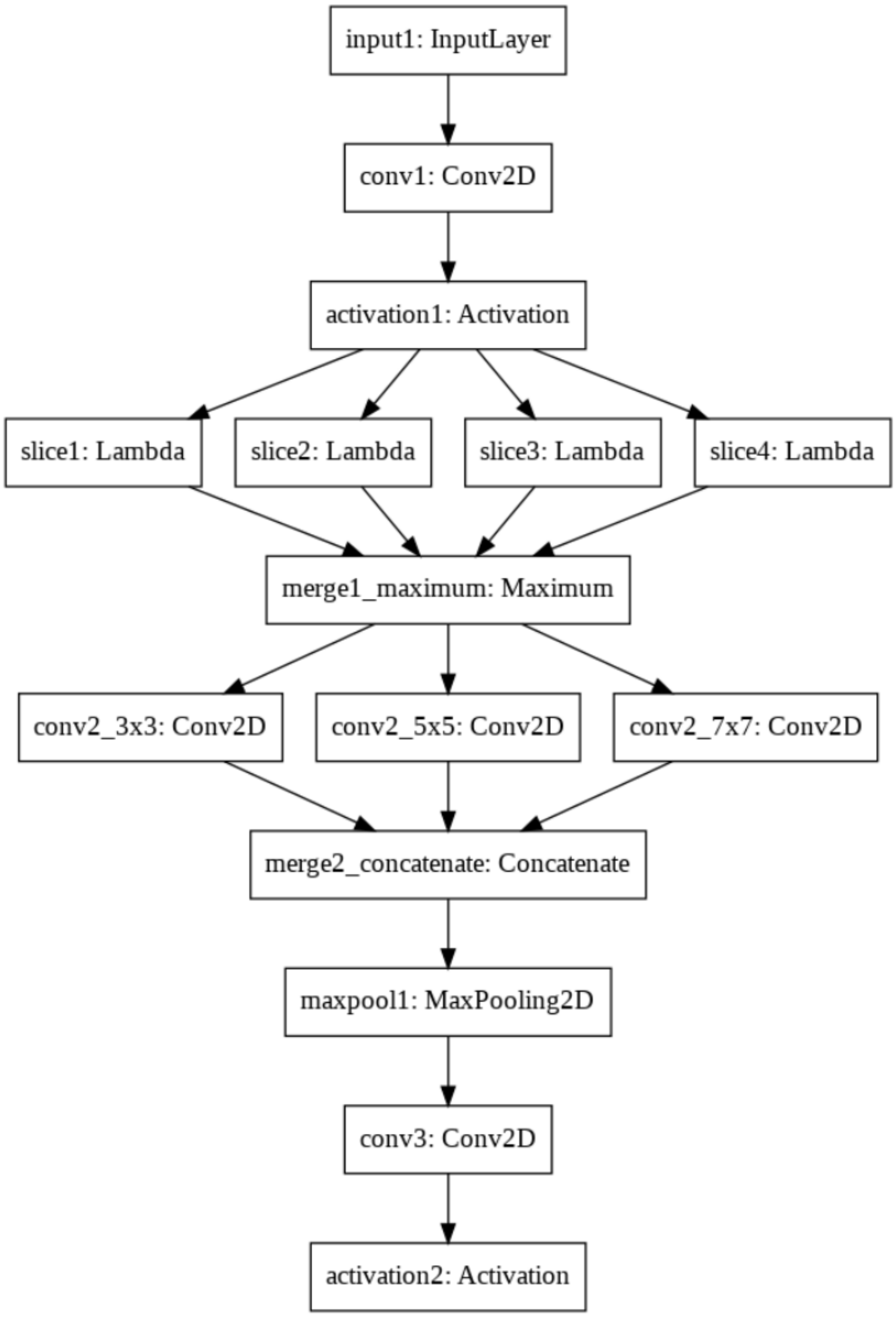


Figure :- 27 Transmission Model

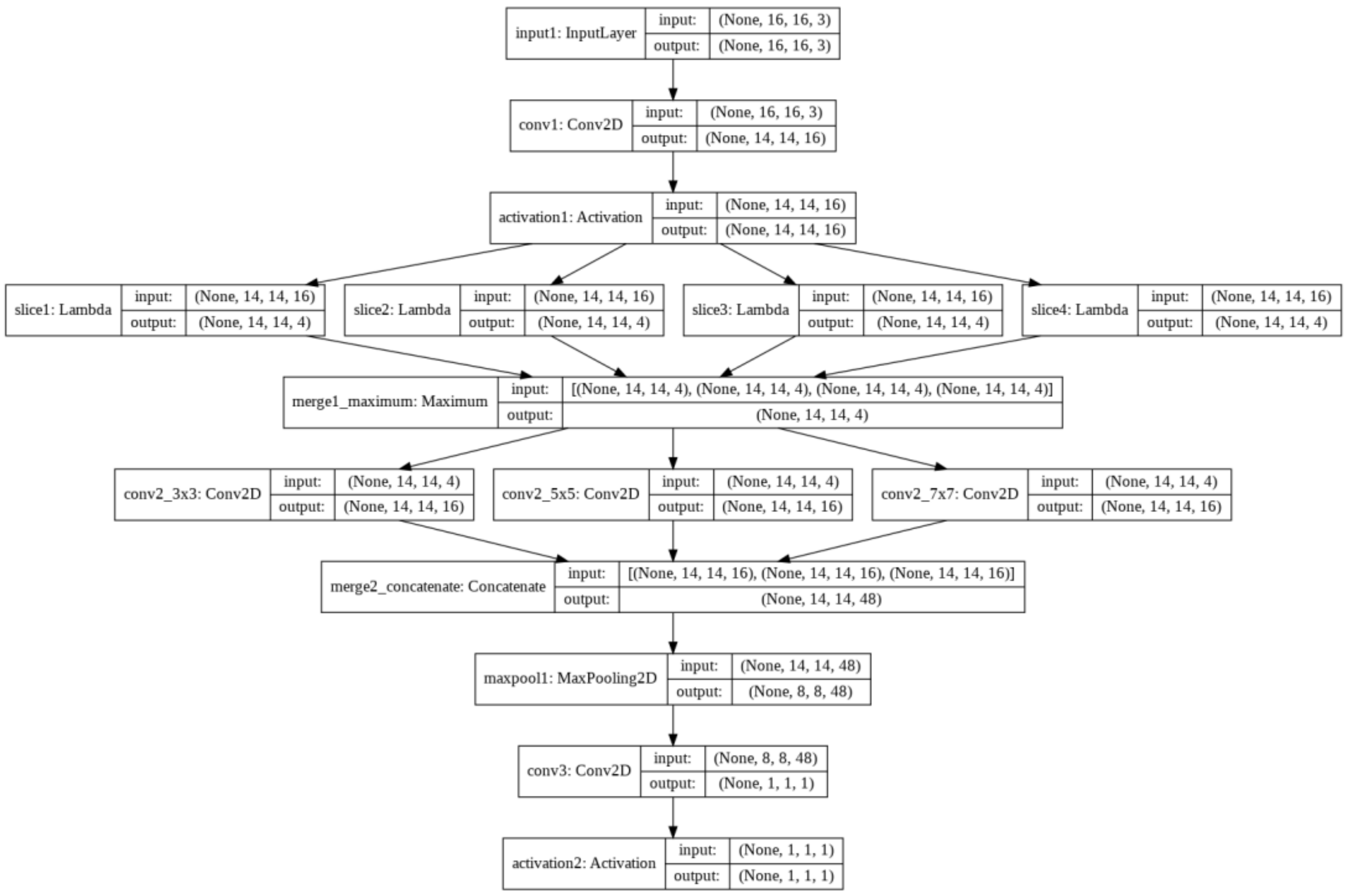
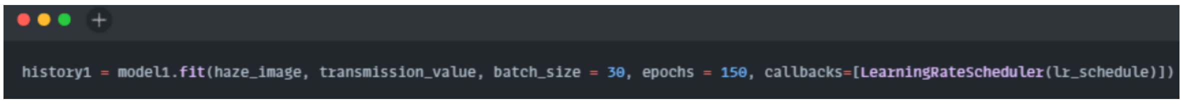


Figure:- 28 Trans Model Shape



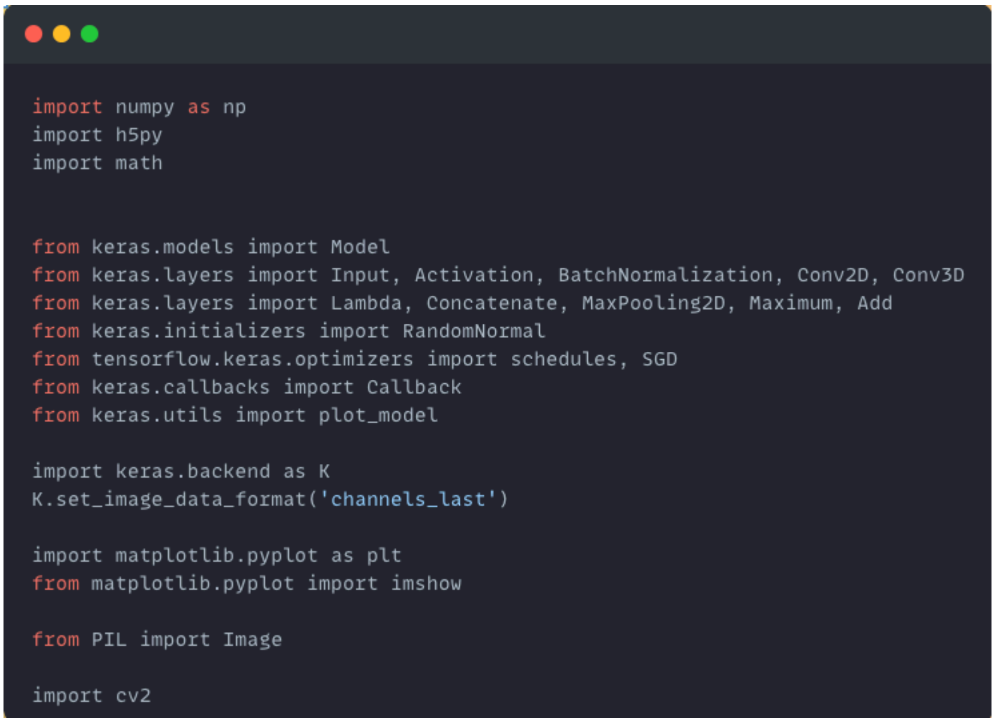
```
history1 = model1.fit(haze_image, transmission_value, batch_size = 30, epochs = 150, callbacks=[LearningRateScheduler(lr_schedule)])
```

Figure:- 29 Training of transmission model

Here we will be training the residual model with 150 Epochs, 30 batch size and a learning rate scheduler which will be utilised to dynamically adjust the Learning rate during training.

Residual-Based Network for video Dehazing

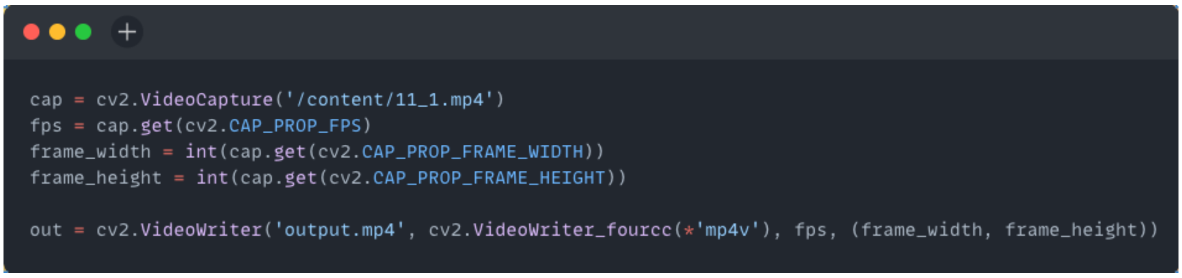
Here for video dehazing we are using the same residual based network which we have used earlier for image dehazing. We will be dividing our video into different number of frames and then we will be dehazing those frames and adding them back to make the video again.



```
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
```

Figure:- 30 Importing the libraries

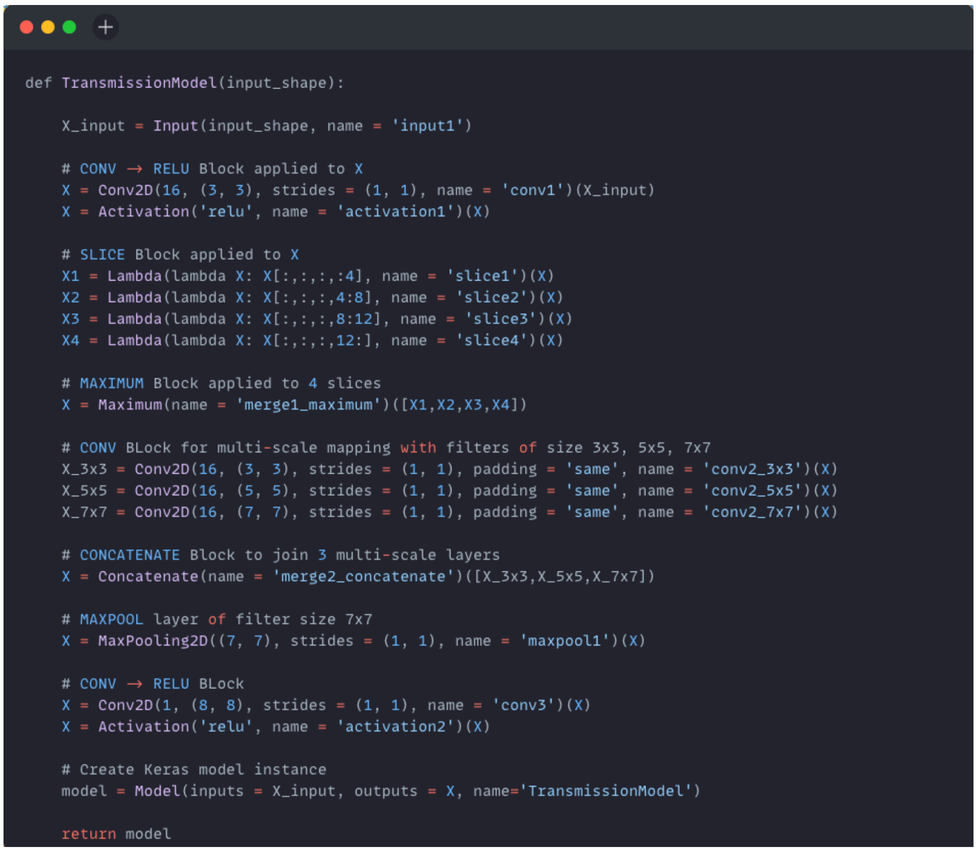
Here we are importing some necessary libraries for deep learning tasks using the Keras library with TensorFlow backend. Here we are importing NumPy for numerical computations, h5py for handling HDF5 files, and math for mathematical function. The Keras modules are imported for defining the neural network model, various layers like Conv2D and batch normalisation. This snippet also configures the image data format to ‘channels_last’ which is very common for CNNs. It also imports matplotlib for plotting, PIL for image processing and OpenCV for computer vision tasks.



```
cap = cv2.VideoCapture('/content/11_1.mp4')
fps = cap.get(cv2.CAP_PROP_FPS)
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

out = cv2.VideoWriter('output.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width, frame_height))
```

Figure:- 31 Code for video processing

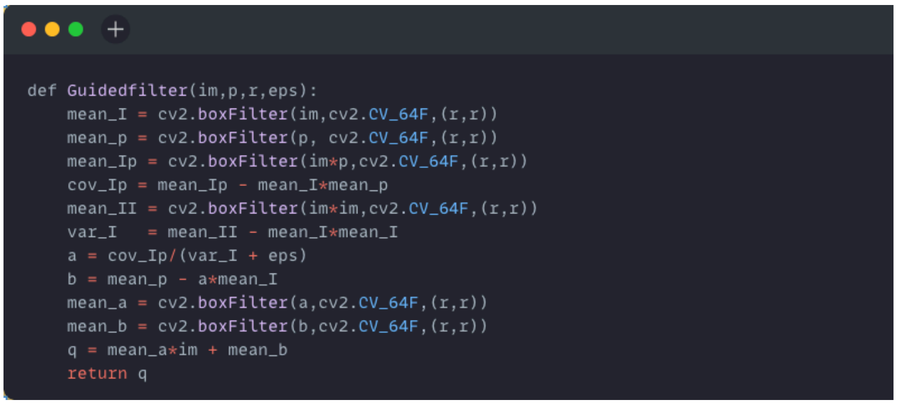
This code snippet is for video processing using OpenCV. It starts by capturing a video file and then retrieves frames per second(fps), frame width, and frame height from the captured video.



```
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
```

Figure:- 32 Code for transmission model

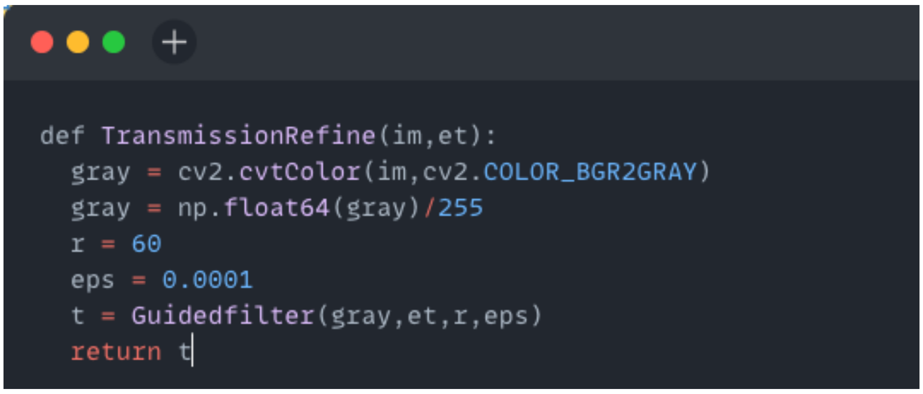
In the above code snippet the transmission model defines a CNN architecture for image transmission modelling. It is taking an input image shape as an argument and creates a Keras model with several layers. Here, the model starts with a convolutional layer followed by ReLU activation function. Next, it slices the output into four parts and then it merges them using a maximum operation. It is also applying convolutional blocks with different filter sizes and concatenates the outputs. The model is basically designed in such a way that it can do multi-scale feature mapping and image transmission tasks.



```
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
```

Figure:- 33 Code for Guided filter

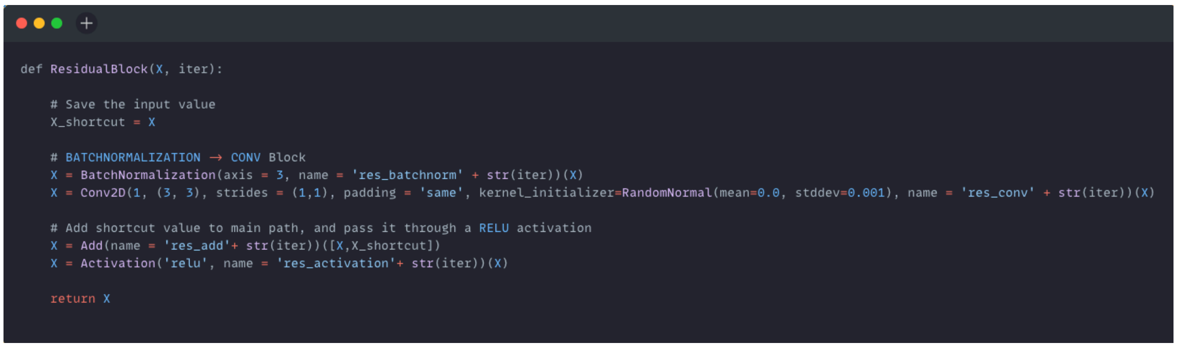
In the above code snippet the Guided filter function is performing guided filtering on an input image using a guidance image. It is calculating statistical measures such as means, covariance and variances and then it computes filter coefficients and a bias term. These are used for an enhanced output image that preserves edges and details while reducing the noise and smoothing the image.



```python
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
```

Figure:- 34 code for transmission refine

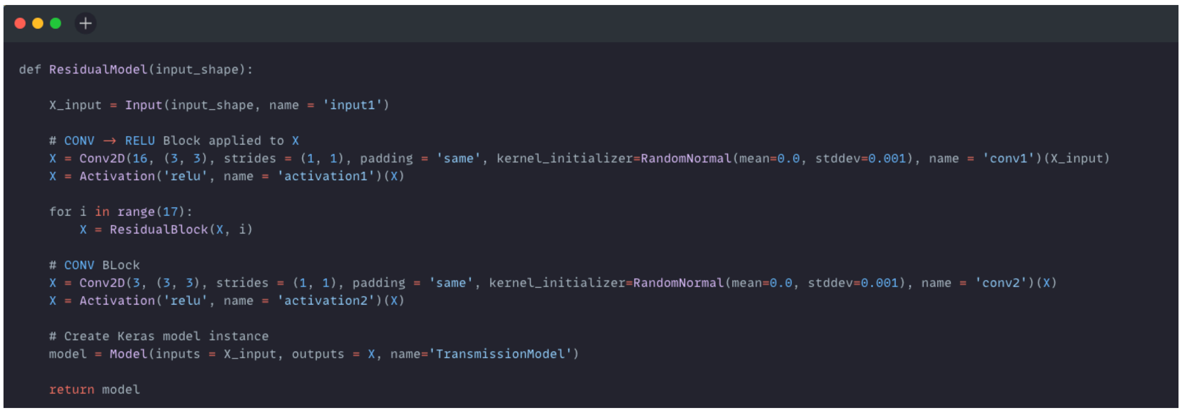
In the above snippet the function refines the transmission map of an image by converting it to grayscale, normalising it, and applying guided filters with some parameters.



```python
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
```

Figure:- 35 Code for Residual Block

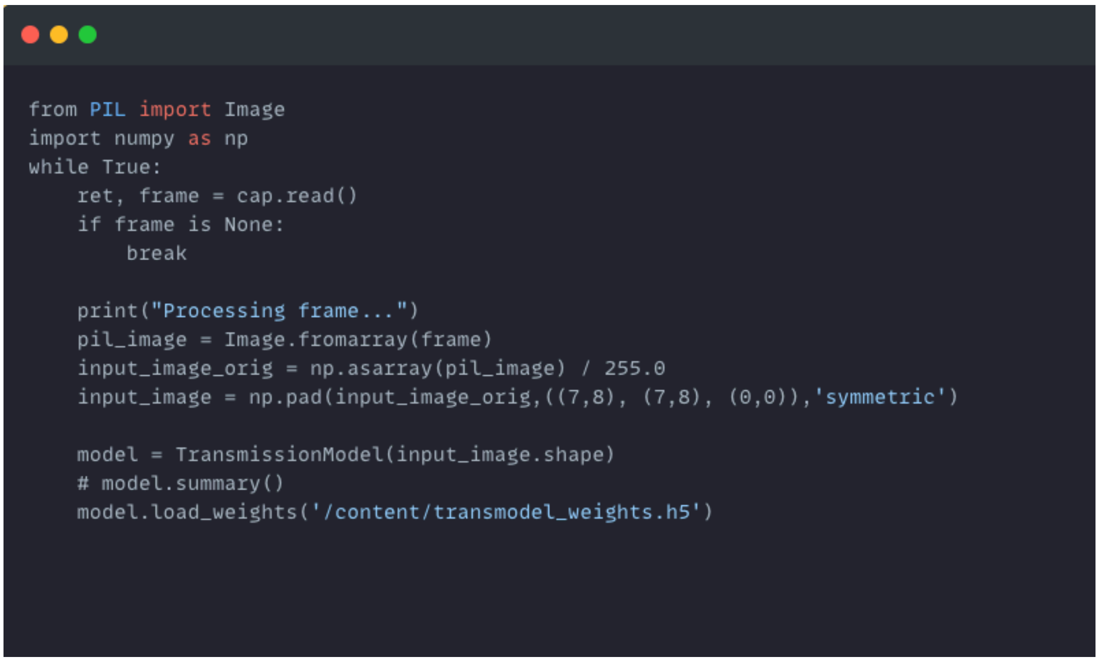
Here in the above code snippet the function is creating a residual block which which is a key component in deep learning architectures. This function includes a batch normalisation followed by a convolution layer of 3x3 kernel and one output channel. Then, the input value is added back to the main path and then passed it through a ReLU activation.



```python
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
```

Figure:- 36 Code for residual model

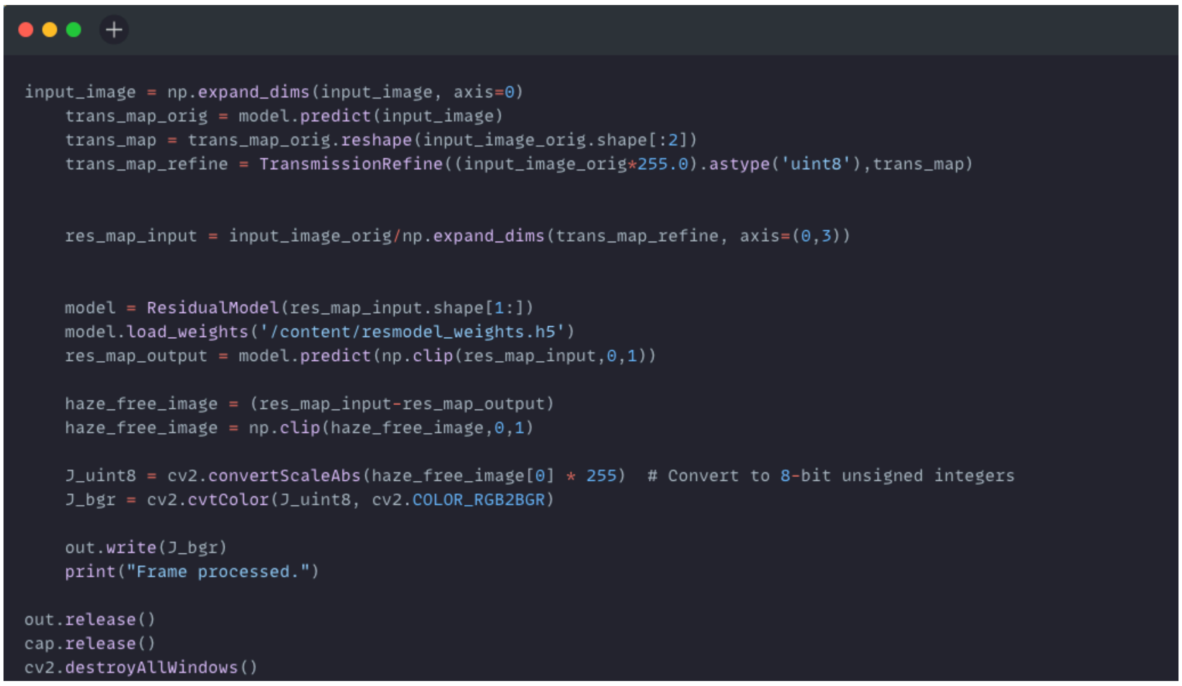
Here in the above code snippet the residual model function is constructing a deep residual model. It starts with an input layer and applies a convolutional block with a 3x3 kernel followed by ReLU activation. After that, it iteratively applies 17 instances of the Residual Block functions to create a deep residual structure. This process repeats again to improve the performance and give desired results from the model.



```python
from PIL import Image
import numpy as np
while True:
    ret, frame = cap.read()
    if frame is None:
        break

    print("Processing frame...")
    pil_image = Image.fromarray(frame)
    input_image_orig = np.asarray(pil_image) / 255.0
    input_image = np.pad(input_image_orig,((7,8), (7,8), (0,0)),'symmetric')

    model = TransmissionModel(input_image.shape)
    # model.summary()
    model.load_weights('/content/transmodel_weights.h5')
```

Figure:- 37 Code for frame processing from a video

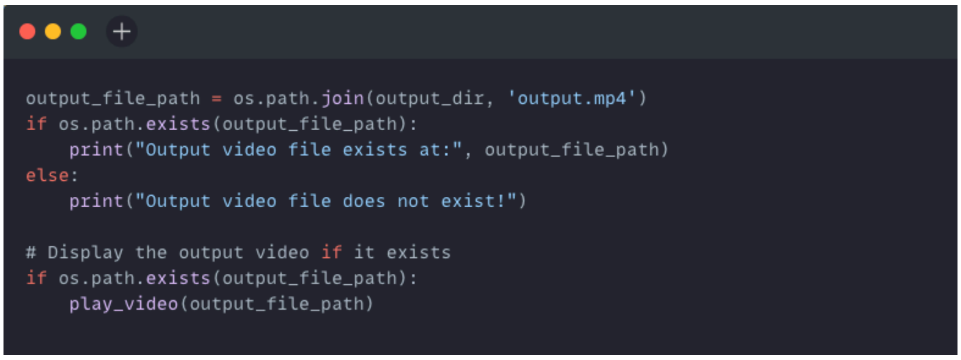
Here in the snippet the code is processing frames from a video using OpenCV by converting each frame into a PIL image and then into a NumPy array for normalization and padding. It loads pre-trained models with weights for further processing.



```python
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('/content/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)

    J_uint8 = cv2.convertScaleAbs(haze_free_image[0] * 255)  # Convert to 8-bit unsigned integers
    J_bgr = cv2.cvtColor(J_uint8, cv2.COLOR_RGB2BGR)

    out.write(J_bgr)
    print("Frame processed.")

out.release()
cap.release()
cv2.destroyAllWindows()
```

Figure:- 38 Code for frame processing from a video - 2

Here in the above snippet, the code segment processes frames from a video by first expanding the input image dimensions and generating the transmission map. It then refines this map and calculates a residual map based on the original image. This further undergoes and give us a haze-free frame of the video. Each processed frame is then converted and combined to make a output video file. This continues until all the frames are processed and the final video is saved.



```python
output_file_path = os.path.join(output_dir, 'output.mp4')
if os.path.exists(output_file_path):
    print("Output video file exists at:", output_file_path)
else:
    print("Output video file does not exist!")

# Display the output video if it exists
if os.path.exists(output_file_path):
    play_video(output_file_path)
```

Figure:- 39 Code for Output file

Here in the snippet the code check if an output video file exists in the directory or not. If exists, then it will print a message confirming its existence along with the file path. If it doesn't exist then it will print a message indicating that the output video file isn't found.

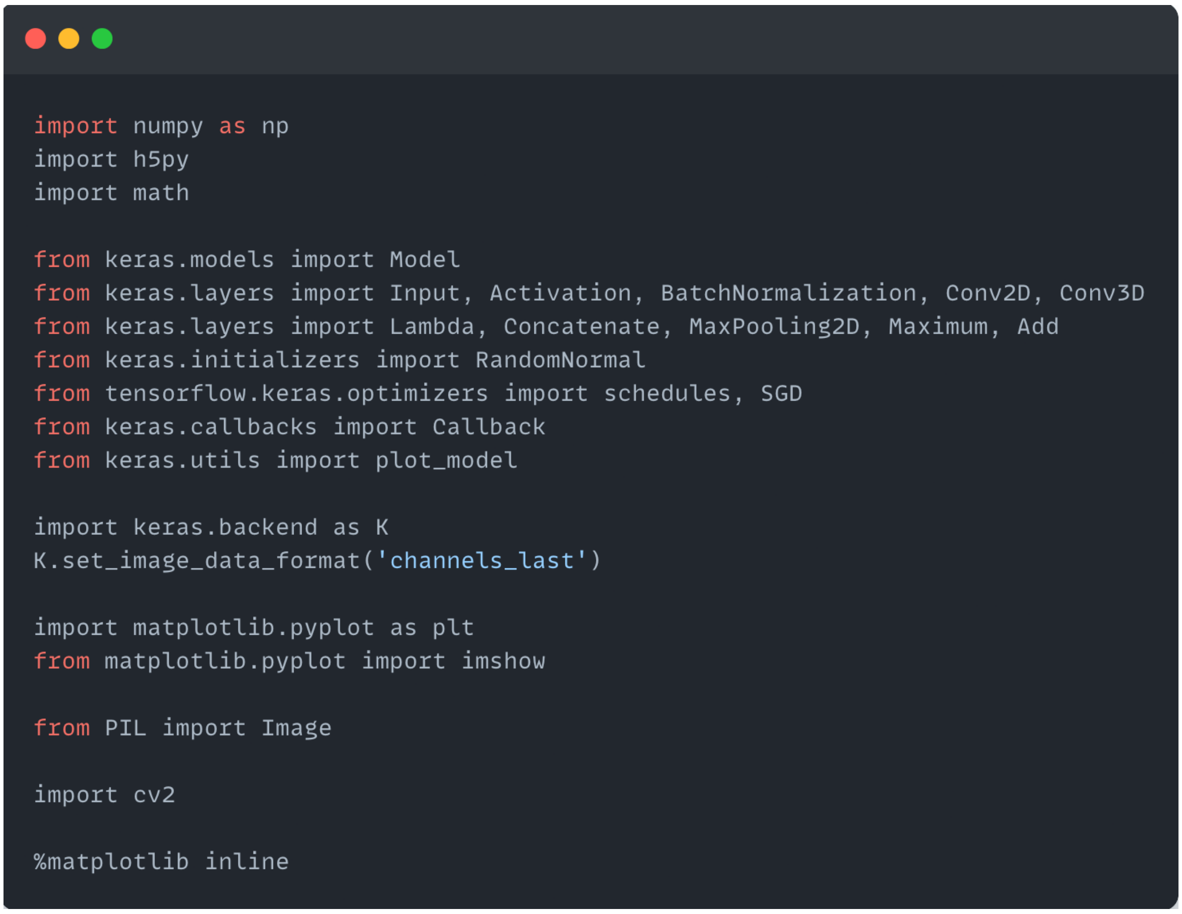
3.5 Key Challenges

- Our 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 lightning 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 heavily computational power to train the model and with the use of more and more big data set consisting of more images, we will be needing a heavy GPU enabled machine to train it efficiently and faster.

CHAPTER - 4

TESTING

4.1 Testing Strategy

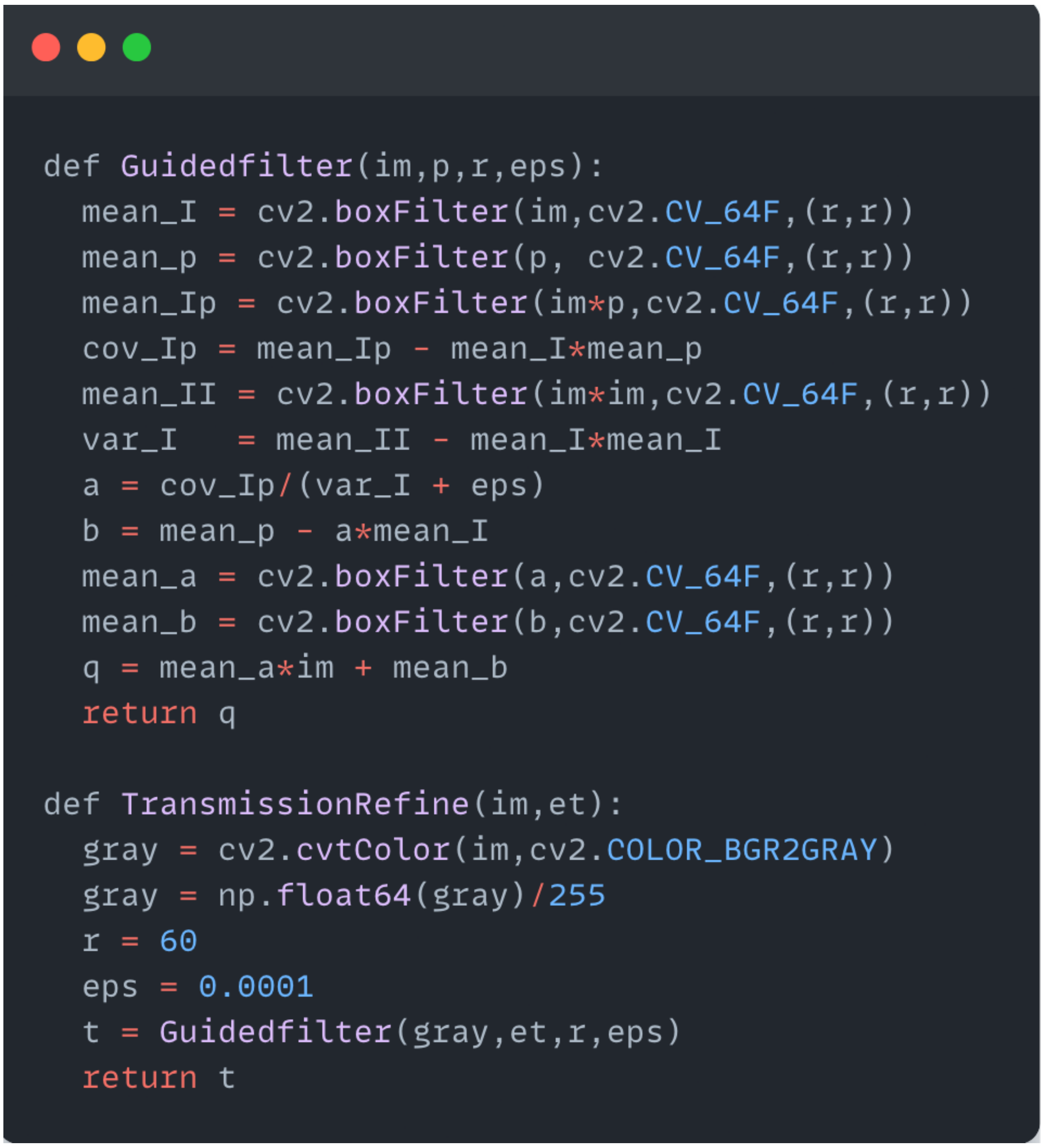


```
import numpy as np
import h5py
import math

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
```

Figure :- 40 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'.

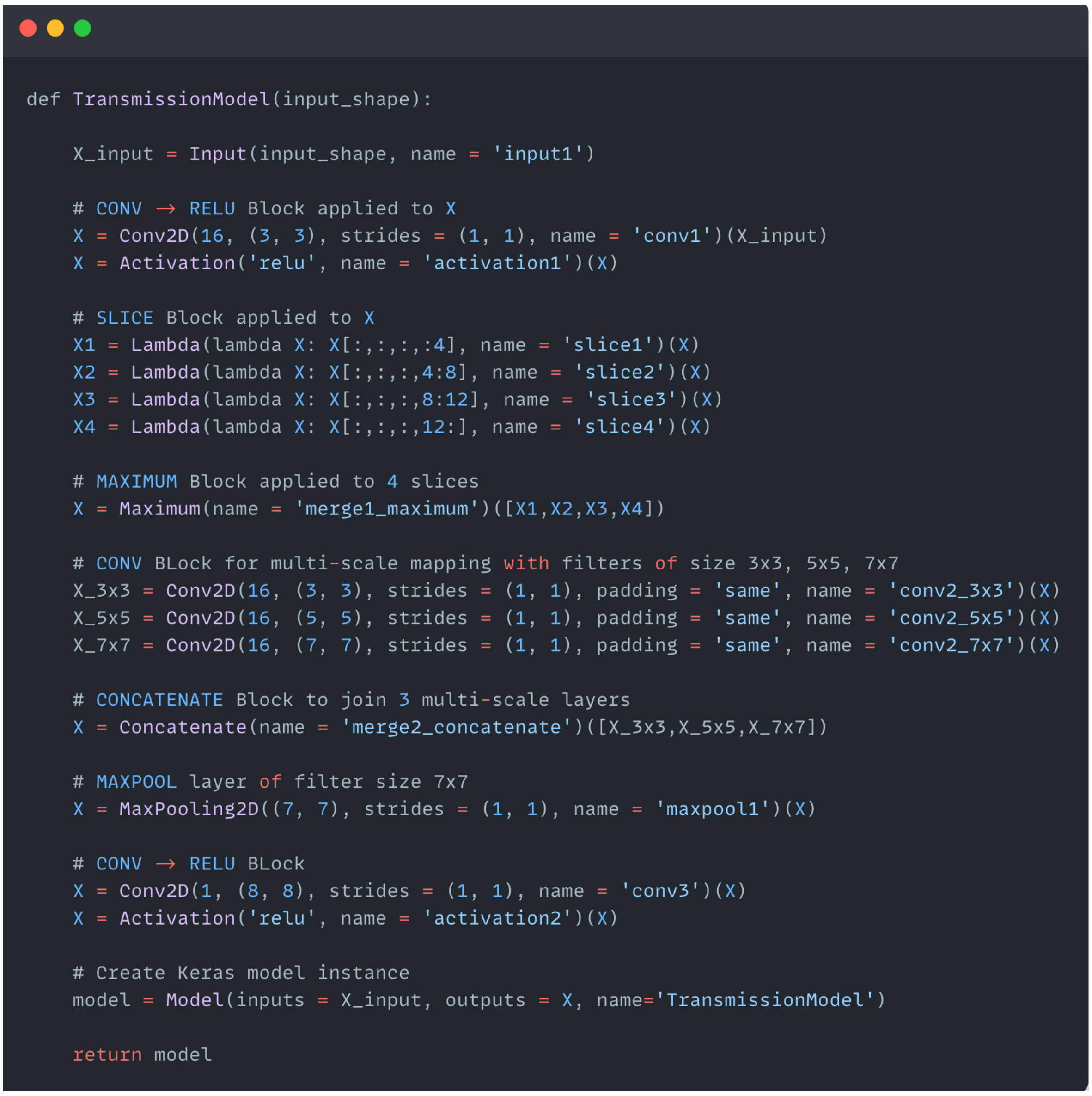
We used '%matplotlib inline' for displaying plots directly below the code cell whenever we will run it.



```python
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
```

Figure:- 41 Util Functions

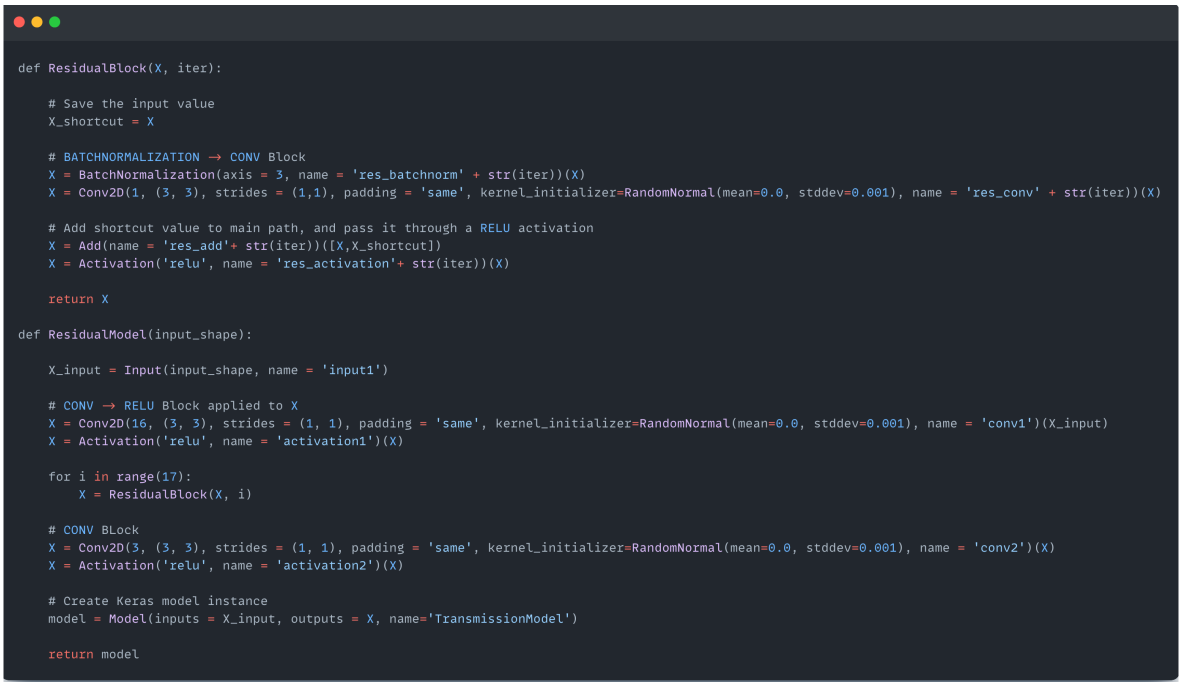
This snippet defines two functions, 'Guidedfilter' and 'TransmissionRefine' which are important components in context of an image processing and they are possibly associated with tasks like image/video dehazing. The Guidedfilter function implemented a technique commonly used for smoothing images while 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 TransmissionRefine 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 function basically serves 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
```

Figure:- 42 Transmission model

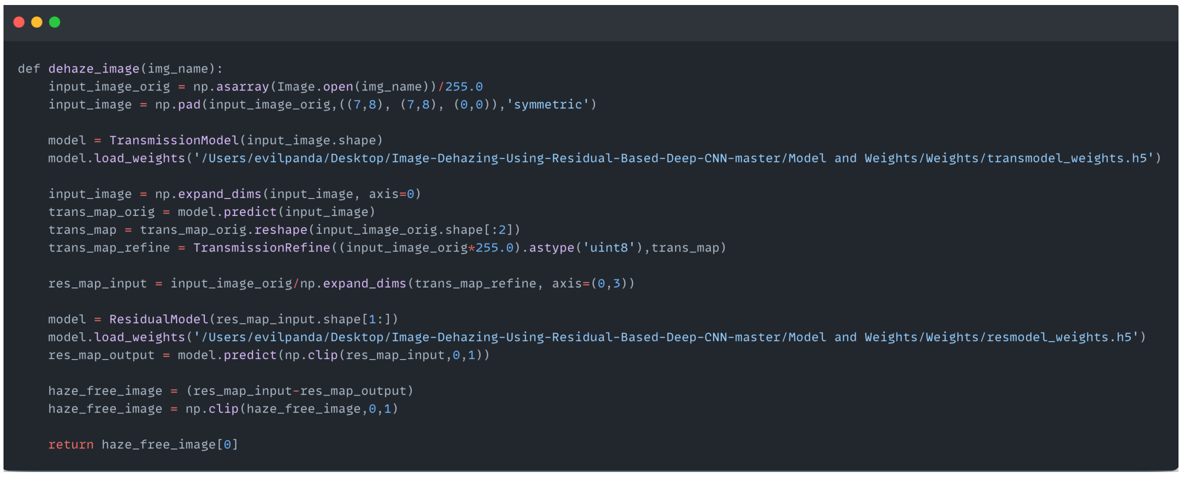
This code has a convolutional neural network (CNN) model called TransmissionModel which is being used via Keras library in python. This model is designed for a specific image processing task which is related to transmission map estimation. Here, the initial block involves around a convolutional layer with rectified linear unit (ReLU) activation which will enhance non-linearity in our neural network. The input tensor is sliced into four parts, and here the maximum operation is applied across all these slices which will result in creating a fused representation. Then, we have three convolutional blocks with filter sizes of 3x3, 5x5, 7x7 respectively. The output of these blocks will capture multiscale features. For an instance a max-pooling layer with a filter size 7x7 will downsample the features. Another convolutional layer with ReLU activation will produce a single-channel output for our model. Overall, model aims to capture and fuse multiscale information for transmission map estimation.



```
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
```

Figure:- 43 Residual model

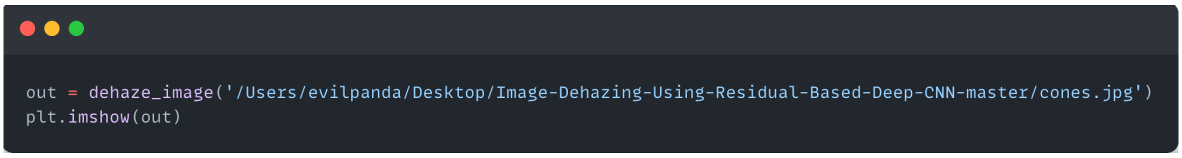
This code has a neural network architecture known as residual neural network(ResNet) using a python library known as keras. Here, the ResNet is designed to learn important features within the input image very efficiently. The ‘ResidualBlock function’ defines as the fundamental building block of our network 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.



```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('/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]
```

Figure:- 44 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.



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

Figure:- 45 Output of trained model

Here in this code we are inserting an input image and based on those two pre-trained model, it will give us the dehazed output image.

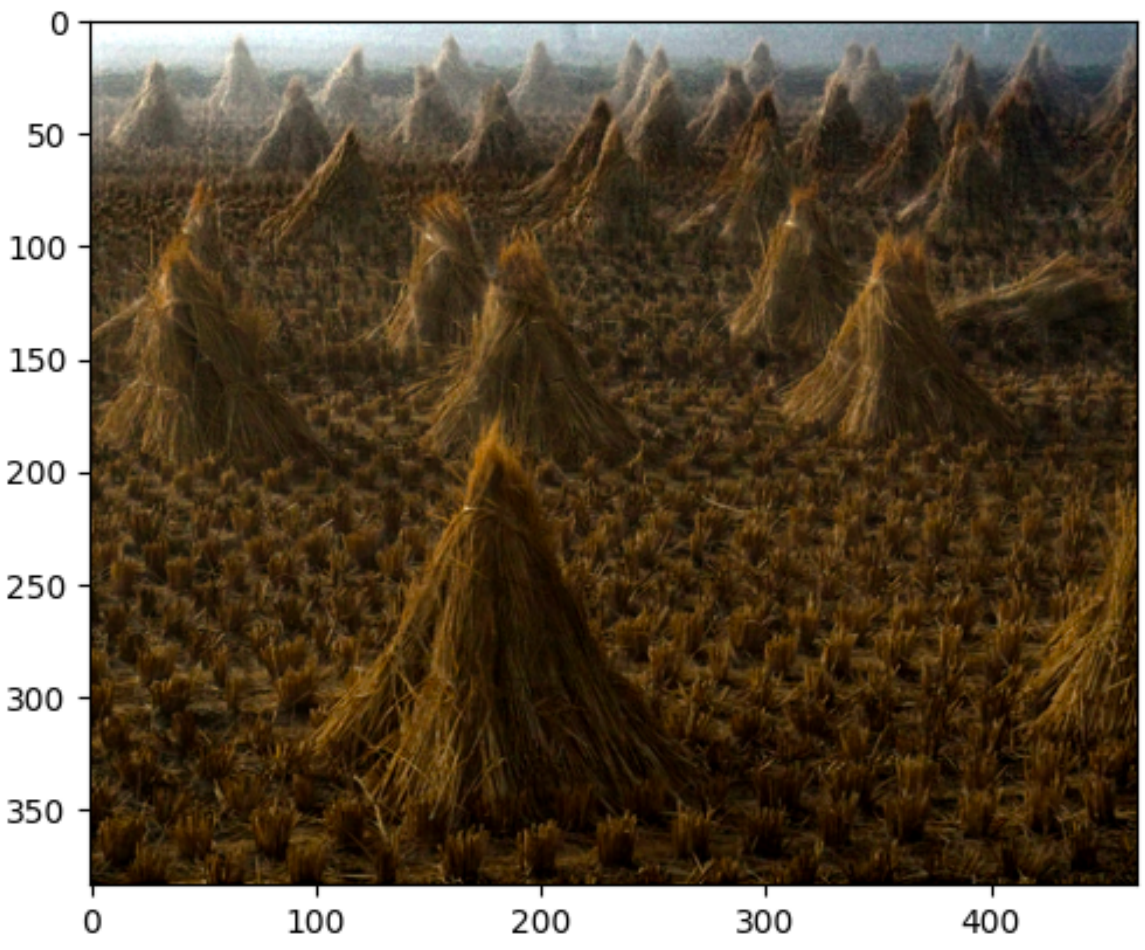
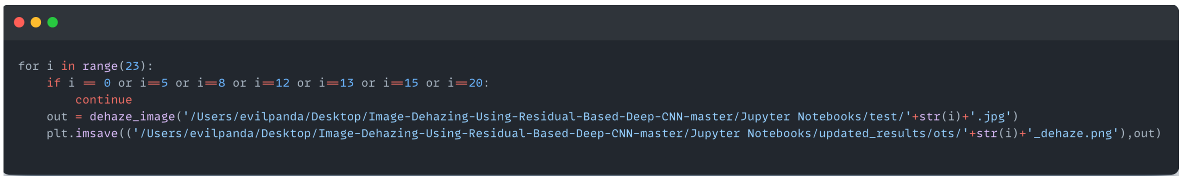


Figure:- 46 Output image based on pre-trained models

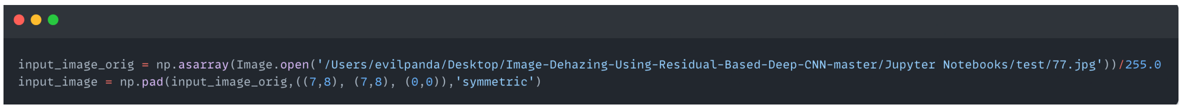


```
for i in range(23):
    if i == 0 or i==5 or i==8 or i==12 or i==13 or i==15 or i==20:
        continue
    out = dehaze_image('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Jupyter Notebooks/test/'+str(i)+'.jpg')
    plt.imsave(('/Users/evilpanda/Desktop/Image-Dehazing-Using-Residual-Based-Deep-CNN-master/Jupyter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

Figure:- 47 Iterations for image dehazing

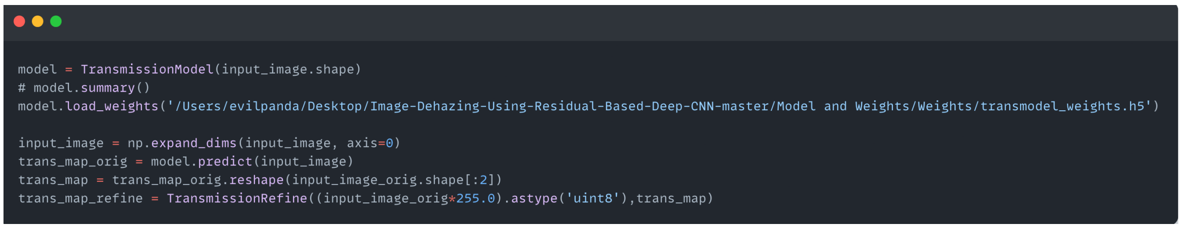
Here the code will iterate from 0 to 22 and for each iteration it will check whether the current index matches with some certain predetermined values (0,5,8,12,13,15 or 20). If it does then it will skip the loop otherwise it will proceed with the dehazing process. The ‘dehaze_image’ function is applied 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')
```

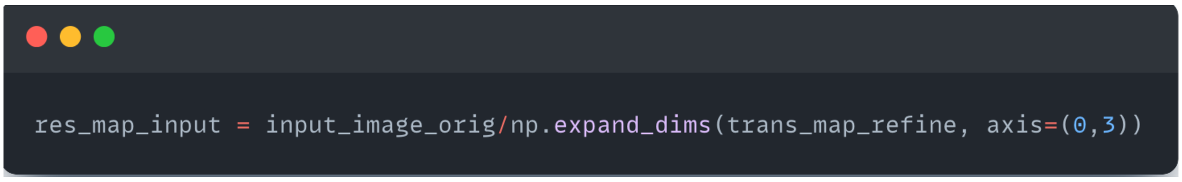
Figure:- 48 Initial preprocessing of input image



```
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)
```

Figure:- 49 Predict Transmission Map

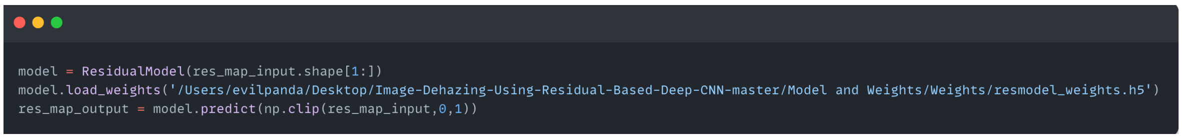
Here in this code the image in converted into a NumPy array, representing its pixel values and normalising them by dividing each pixel value by 255.0 for ensuring so that the values are in the range of 0 to 1. Then the image is symmetrically padded with a border of 7 pixels on the top, bottom, left and 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))
```

Figure:- 50 Residual model input

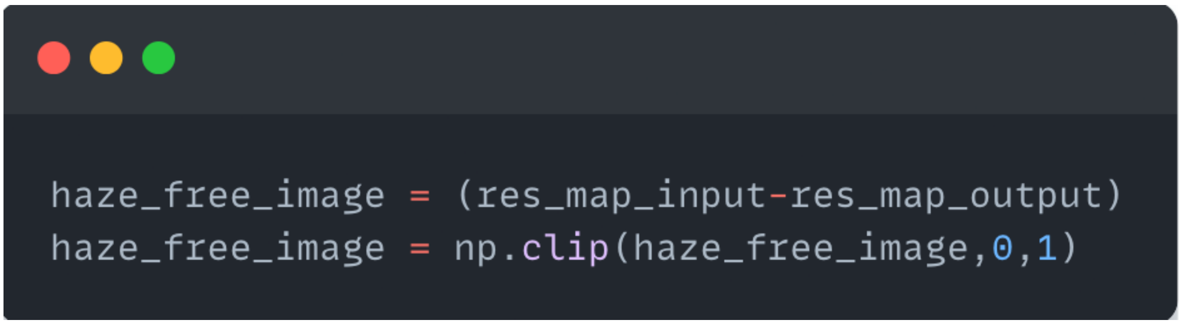
Here in this code a model is employed for estimating transmission maps. The model initialises and loads, pre-trained weights for a deep learning model, which is known as TransmissionModel and we designed this to estimate transmission maps for our dehazing images. We applied this to an input image and 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))
```

Figure:- 51 Predicting residual image

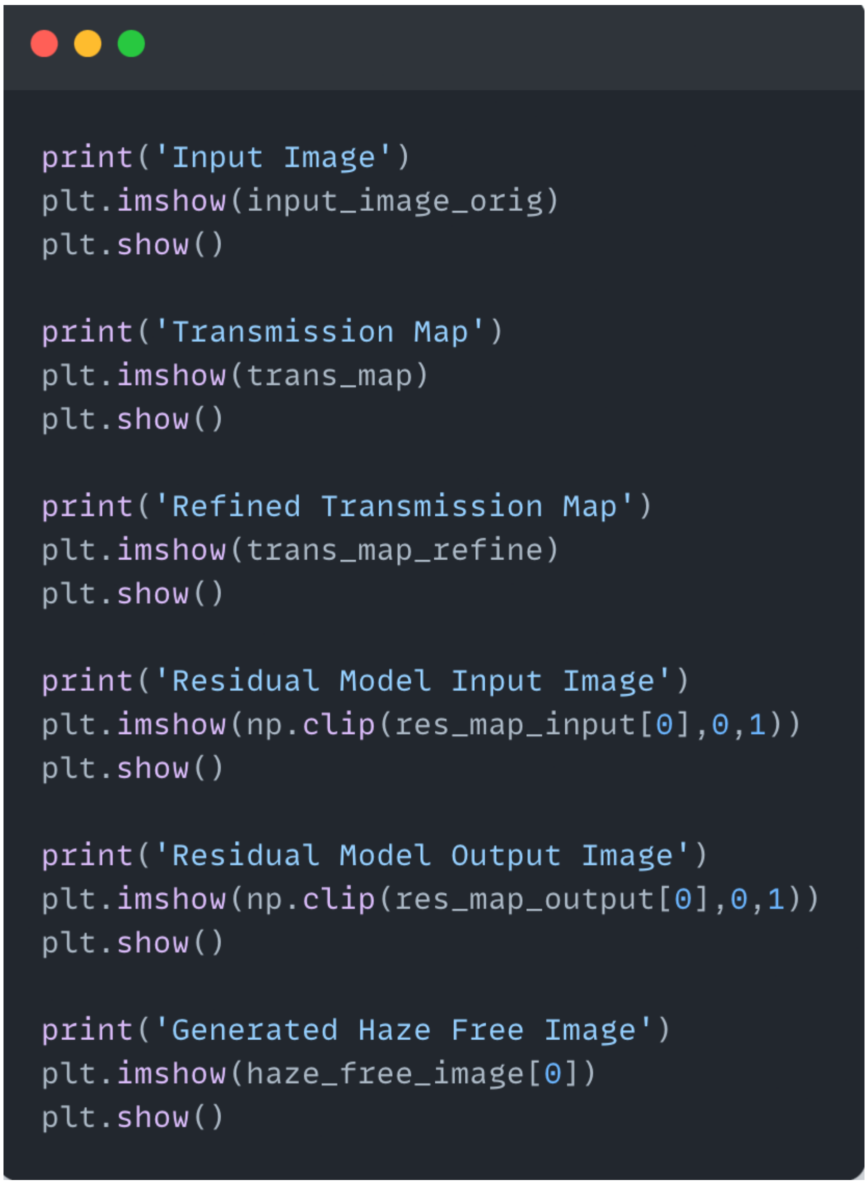
Here this code calculates a residual map by dividing our original input image via refined transmission map. This code is basically expanding the dimensions of refined trans map. Here this code initialises and loads our pre-trained weights for our Residual Model so that they can do the refining of residual 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)
```

Figure:- 52 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.



```python
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()
```

Figure:- 53 Plotting of images at different steps

This 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.

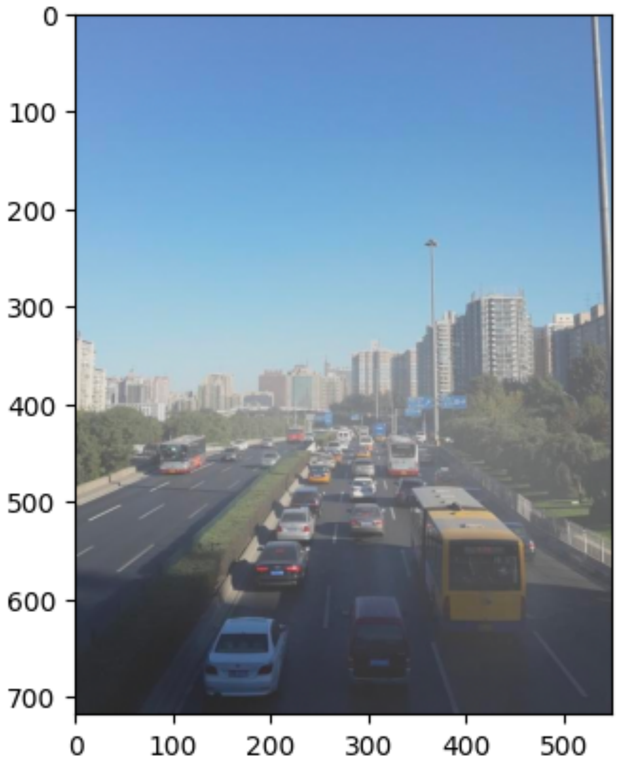


Figure:- 54 Input Image for dehazing

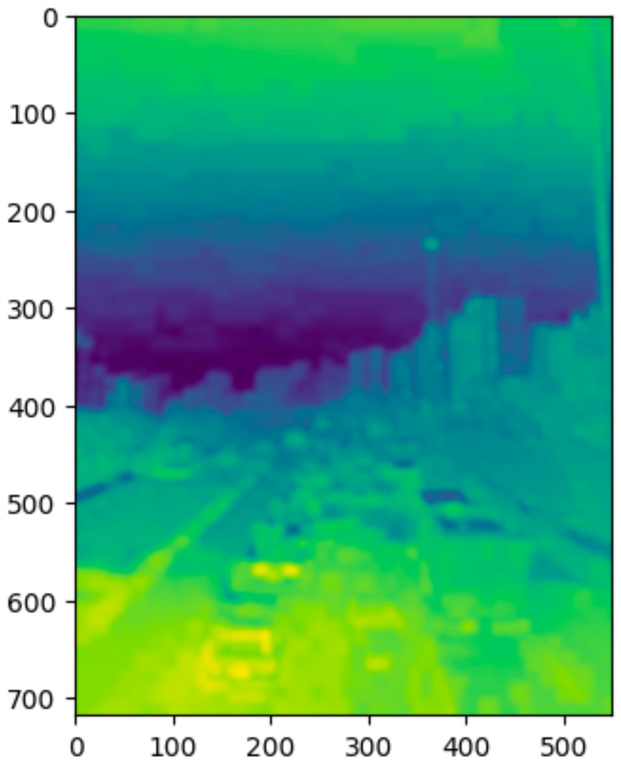


Figure:- 55 Transmission Map

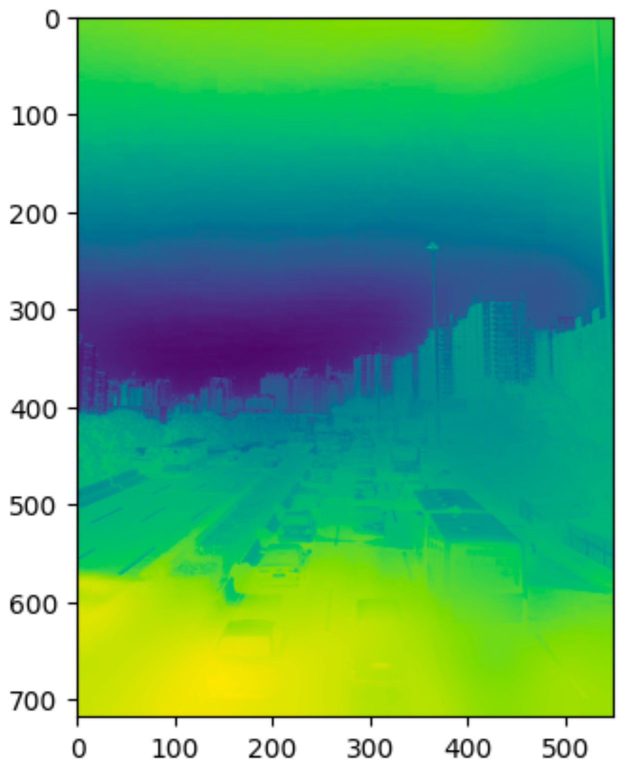


Figure:- 56 Refined Transmission Map

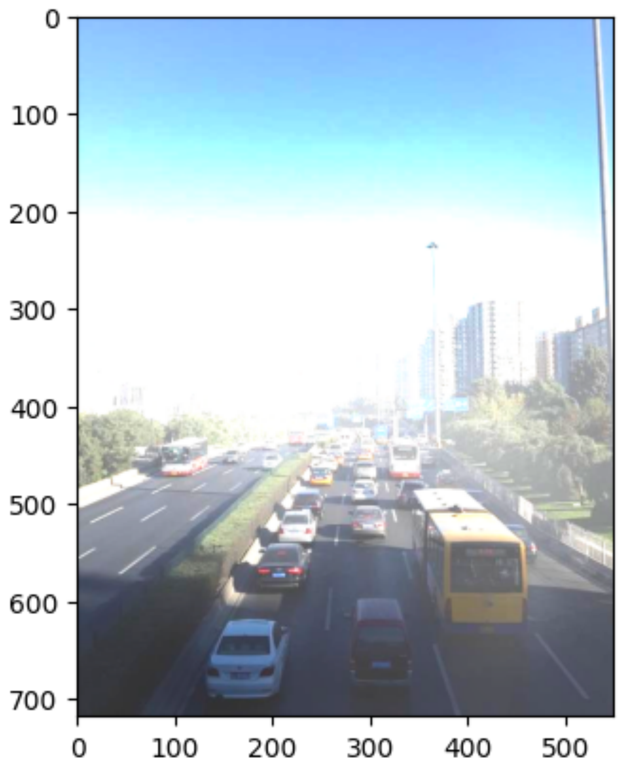


Figure:- 57 Residual Model Input Image

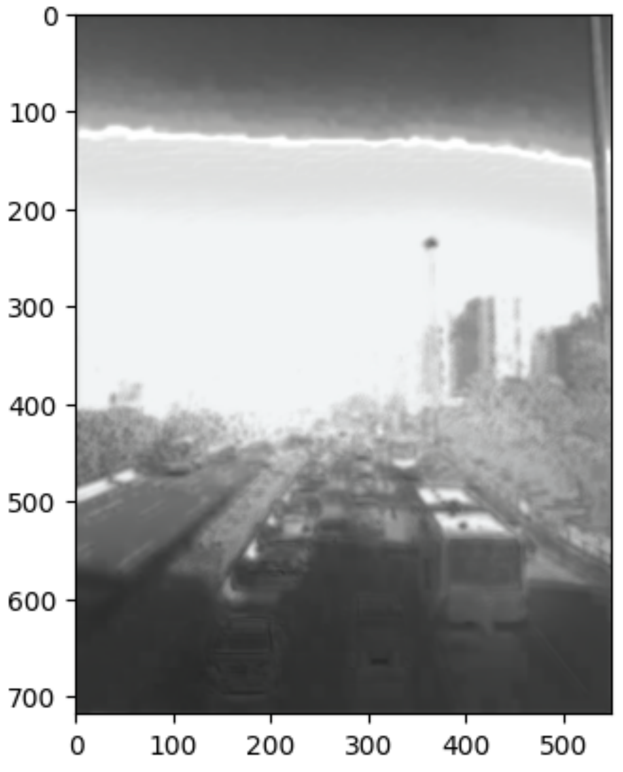


Figure:- 58 Residual Model Output Image

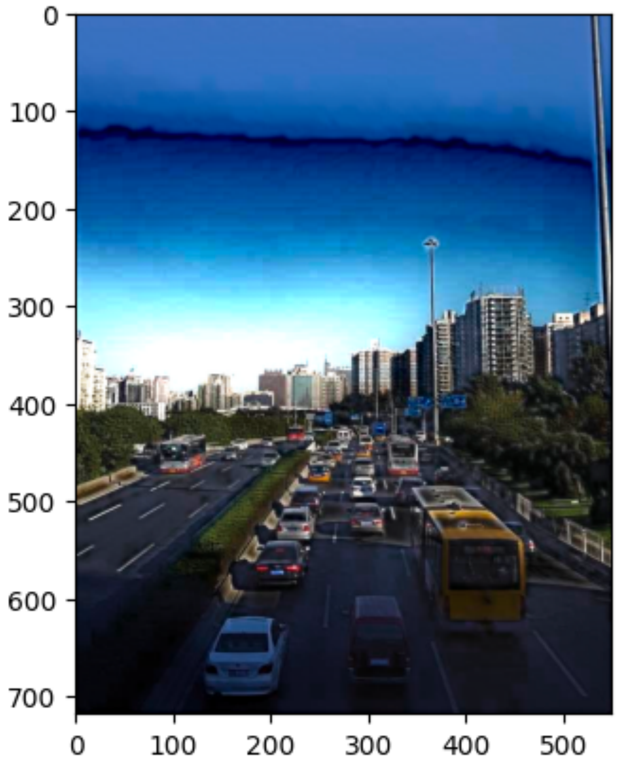
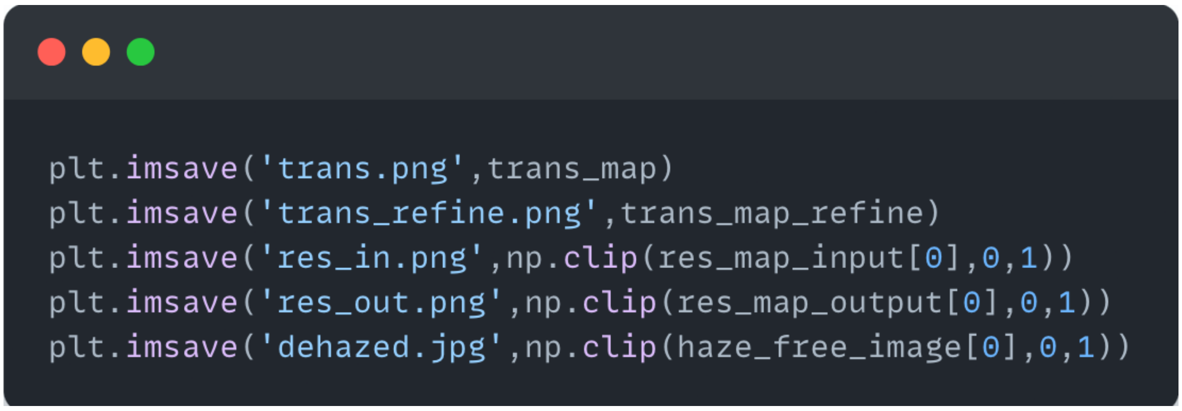


Figure:- 59 Generated Haze Free Image



```
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))
```

Figure:- 60 Saving of Images

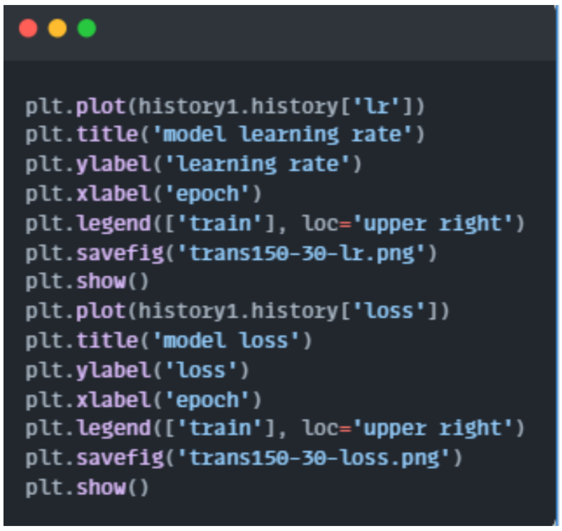
Here this code will save our all images which were given by the model as an output. We have also specified the names of output images in our code itself. We are using 'np.clip' function which will ensure the pixel values are within valid range of 0 to 1 before saving them.

CHAPTER - 5
RESULTS AND EVALUATION

5.1 Results

After successful training of both the networks. We will be validating the models and can see how they're performing on the testing data set.

Transmission Model Network



```
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()
```

Figure:- 61 Plotting the model loss and model learning rate

Model Learning Rate:- It shows as the learning rate of the model over the epochs

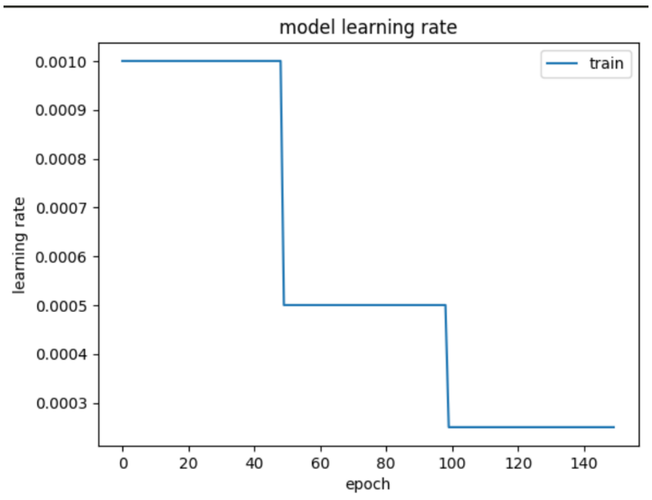


Figure:- 62 Model Learning Rate

Model Loss:- It shows the model loss over the epochs.

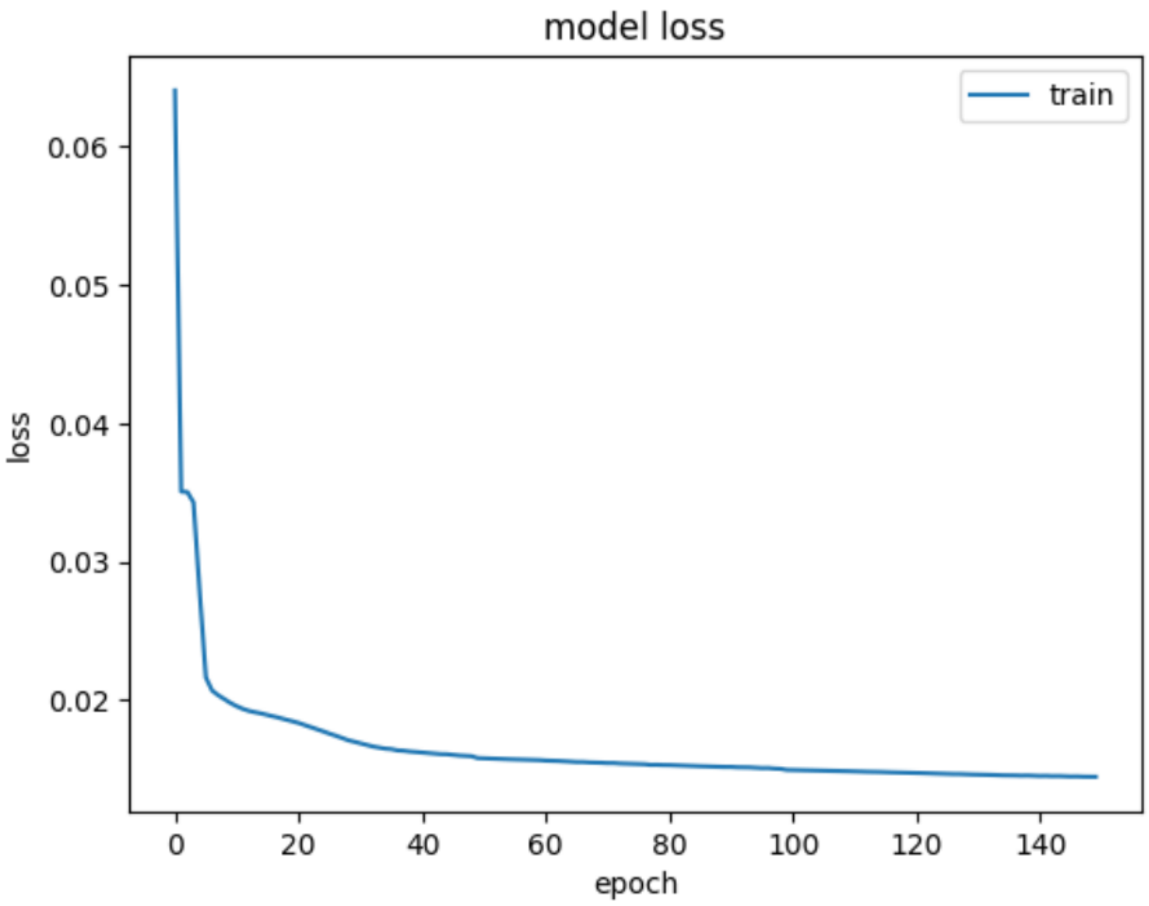
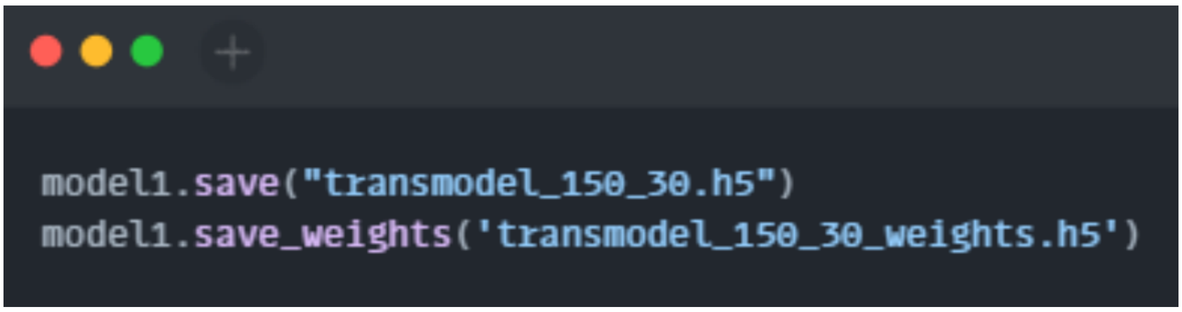


Figure:- 63 Model Loss

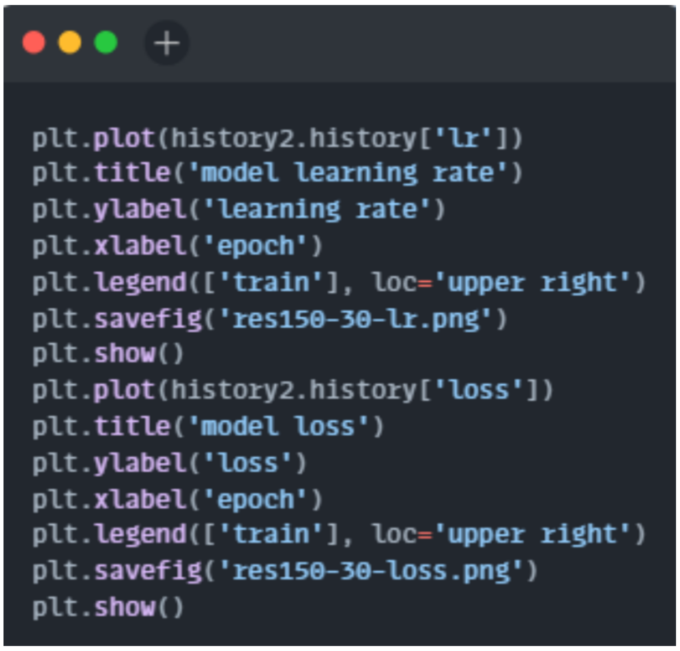


```
model1.save("transmodel_150_30.h5")
model1.save_weights('transmodel_150_30_weights.h5')
```

Figure:- 64 Saving the model and weights

Here we Are saving the transmission model as well as the weights.

Residual Model Network



```
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()
```

Figure:- 65 Plotting Model loss and learning rate

Model Learning Rate:- It shows as the learning rate of the model over the epochs

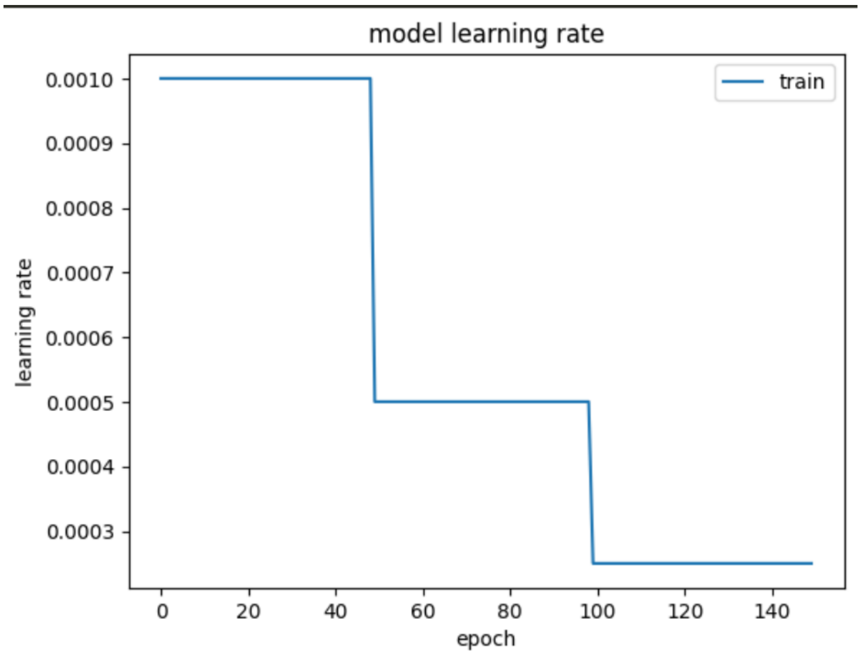


Figure:- 66 Model Learning Rate

Model Loss:- It shows the model loss over the epochs.

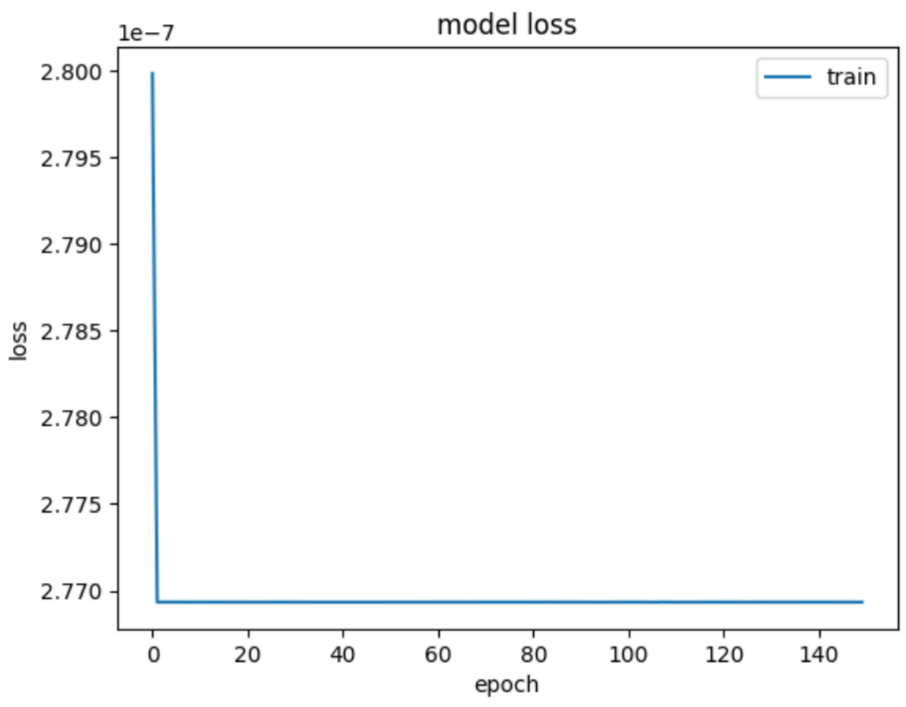
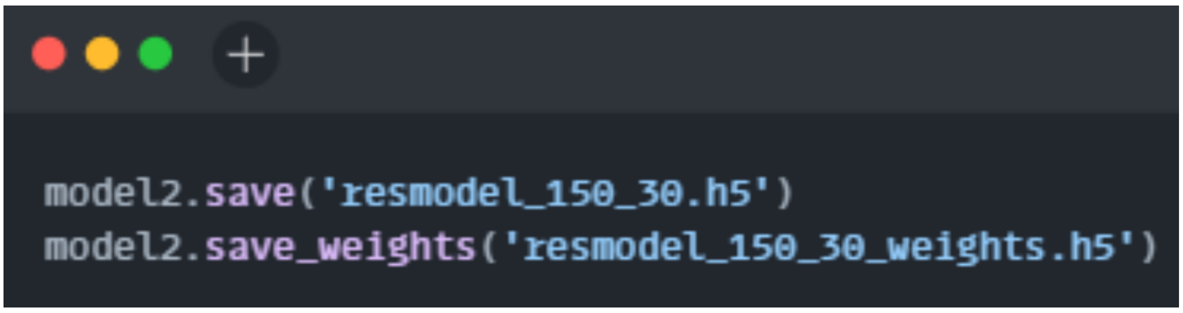


Figure:- 67 Model Loss



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

Figure:- 68 Saving the model and weights

Here we are saving Residual model as well as the weights.

Since we have seen in the Chapter 4.2 that our trained model is able to dehaze the image efficiently.

- Our functional requirement of having a code base which is capable of handling the uploaded images and utilise it for future. Use has been fulfilled.
- Requirement for the model to be able to dehaze both indoor and outdoor images is also being satisfied by the model, as seen in the Chapter 4.2.
- The model is fast enough to be able to dehaze the images within two seconds of time after being uploaded.

Video dehazing

We have seen an improvement in our model when we did it on a video. Though the results were not upto the mark, our model still tried it's best in giving the results.

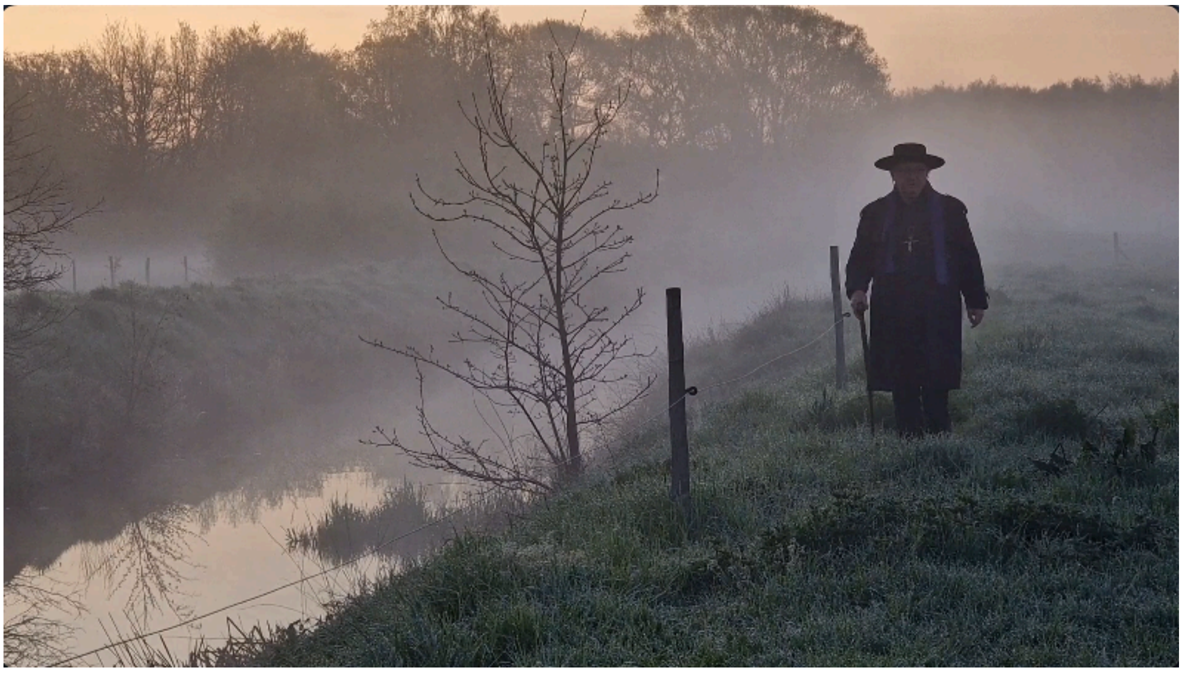


Figure:- 69 Frame of an original input video (1)

Figure:- 70 Frame of an output video (1)

Figure:- 71 Frame of an original input video (2)

Figure:- 72 Frame of an output video (2)

Further, we have seen in the Chapter 4.2 that our trained model is able to dehaze the video efficiently too.

- Our functional requirement of having a code base which is capable of handling the uploaded videos and utilise it for future use has been fulfilled.
- Requirement for the model to be able to dehaze videos by converting into frames is also being satisfied by the model, as seen in the Chapter 4.2.
- The model is fast enough to be able to dehaze the video of 480p resolution and timing depends upon how long the video timing is. If the duration of the video uploaded is 15 seconds, then it is taking around 60-70 seconds to give us the desired video output.

CHAPTER - 6
CONCLUSIONS AND FUTURE SCOPE

6.1 Conclusion

As we are concluding our project report we have implemented an approach on single image dehazing and single video-dehazing using residual-based deep convolutional neural network (CNN). Our approach has avoided the estimation of atmospheric lights and improved the efficiency of image dehazing. We divided our network model into two phases and our residual network trained the values of atmospheric lights. We have tested our model's efficiency on the NYU2 depth dataset and also on RESIDE dataset. Also, we have tested a few videos too for video dehazing. When we looked at the results we found out that our model outperformed in terms of quantitative and qualitative evaluation metrics. Our proposed model performed very efficiently on dehazing processes for different scenes with no obvious color distortion or any image blurring etc. The results were very closer to the standard results. We are also aiming that our residual model also has the potential to be applied to real-time image and video dehazing too and we can also optimised them further with our model to improve the performance. We have tried our best in giving a valuable model and insights via this project report with the help of deep neural network methods for image dehazing

6.2 Future Scope

Our proposed model has many future scopes where we can improve our model and extend them for both image and video dehazing.Right now we have only tested for 480p videos because of low GPU power. We can further improve efficiency of our model. We will try to further extend our model for video dehazing also. We will be needing a larger and more realistic dataset for the training of our network model which will further help in performance of our model. Although we think that we have tried our best on this model but we still think that we can improve our model further for better result and to implement our model efficiently on both image and video dehazing.

REFERENCES

[1] Van Nguyen, T., Vien, A.G. and Lee, C. (2022) ‘Real-time image and video dehazing based on multiscale guided filtering’, *Multimedia Tools and Applications*, 81(25), pp. 36567–36584. doi:10.1007/s11042-022-13533-4.

[2] Chaitanya, B.S.N.V. and Mukherjee, S. (2021) ‘Single image dehazing using improved cyclegan’, *Journal of Visual Communication and Image Representation*, 74, p. 103014. doi:10.1016/j.jvcir.2020.103014.

[3] Feng, Y. (2020) ‘A survey on video Dehazing using deep learning’, *Journal of Physics: Conference Series*, 1487(1), p. 012018. doi:10.1088/1742-6596/1487/1/012018.

[4] Bharath Raj, N. and Venketeswaran, N. (2020) ‘Single image haze removal using a generative adversarial network’, *2020 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET)* [Preprint]. doi:10.1109/wispnet48689.2020.9198400.

[5] Khatun, A. *et al*. (2020) ‘Single image dehazing: An analysis on generative adversarial network’, *Journal of Computer and Communications*, 08(04), pp. 127–137. doi:10.4236/jcc.2020.84010.

[6] Li, B. *et al*. (2019) ‘Benchmarking single-image Dehazing and beyond’, *IEEE Transactions on Image Processing*, 28(1), pp. 492–505. doi:10.1109/tip.2018.2867951.

[7] Ren, W. *et al*. (2019) 'Deep Video Dehazing with semantic segmentation', *IEEE Transactions on Image Processing*, 28(4), pp. 1895–1908. doi:10.1109/tip.2018.2876178.

[8] Yuan, K. *et al*. (2019) 'Single image dehazing via nin-dehazenet', *IEEE Access*, 7, pp. 181348–181356. doi:10.1109/access.2019.2958607.

[9] Li, J., Li, G. and Fan, H. (2018) 'Image dehazing using residual-based deep CNN', *IEEE Access*, 6, pp. 26831–26842. doi:10.1109/access.2018.2833888.

[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.

Harsh_Report

ORIGINALITY REPORT

8%	5%	6%	%
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	www.researchgate.net Internet Source	1%
3	pdfs.semanticscholar.org Internet Source	<1%
4	link.springer.com Internet Source	<1%
5	assets.researchsquare.com Internet Source	<1%
6	neptune.ai Internet Source	<1%
7	www.jstage.jst.go.jp Internet Source	<1%
8	www.arxiv-vanity.com Internet Source	<1%
	iopscience.iop.org	

9	Internet Source	<1 %
10	m.scirp.org Internet Source	<1 %
11	www.comp.hkbu.edu.hk Internet Source	<1 %
12	ia-petabox.archive.org Internet Source	<1 %
13	Abeer Ayoub, Walid El-Shafai, Fathi E. Abd El-Samie, Ehab K. I. Hamad, El-Sayed M. EL-Rabaie. "Review of dehazing techniques: challenges and future trends", Multimedia Tools and Applications, 2024 Publication	<1 %
14	www.degruyter.com Internet Source	<1 %
15	etheses.whiterose.ac.uk Internet Source	<1 %
16	ar5iv.labs.arxiv.org Internet Source	<1 %
17	ouci.dntb.gov.ua Internet Source	<1 %
18	www.coursehero.com Internet Source	<1 %
	www.mdpi.com	

19	Internet Source	<1%
20	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%
21	Wen Su, Haifeng Zhang, Jia Li, Wenzhen Yang, Zengfu Wang. "Monocular Depth Estimation as Regression of Classification using Piled Residual Networks", Proceedings of the 27th ACM International Conference on Multimedia, 2019 Publication	<1%
22	Wenfei Zhang, Jian Liang, Liyong Ren, Haijuan Ju, Enshi Qu, Zhaofeng Bai, Yao Tang, Zhaoxin Wu. "Real-time image haze removal using an aperture-division polarimetric camera", Applied Optics, 2017 Publication	<1%
23	researchrepository.wvu.edu Internet Source	<1%
24	"Advances in Multimedia Information Processing – PCM 2017", Springer Science and Business Media LLC, 2018 Publication	<1%

25	Isha Kansal, Singara Singh Kasana. "Minimum preserving subsampling-based fast image de-fogging", Journal of Modern Optics, 2018 Publication	<1 %
26	Kangle Yuan, Jianguo Wei, Wenhuan Lu, Naixue Xiong. "Single Image Dehazing via NIN-DehazeNet", IEEE Access, 2019 Publication	<1 %
27	Sahadeb Shit, Dip Narayan Ray. "Review and evaluation of recent advancements in image dehazing techniques for vision improvement and visualization", Journal of Electronic Imaging, 2023 Publication	<1 %
28	Xiaojie Guo, Yang Yang, Chaoyue Wang, Jiayi Ma. "Image dehazing via enhancement, restoration, and fusion: A survey", Information Fusion, 2022 Publication	<1 %
29	Zhuo Su, Ruizhi Liu, Yuxin Feng, Fan Zhou. "Attention-adaptive multi-scale feature aggregation dehazing network", Journal of Visual Communication and Image Representation, 2023 Publication	<1 %
30	ebin.pub Internet Source	<1 %

31	es.scribd.com Internet Source	<1 %
32	export.arxiv.org Internet Source	<1 %
33	studentsrepo.um.edu.my Internet Source	<1 %
34	Amina Khatun, Mohammad Reduanul Haque, Rabeya Basri, Mohammad Shorif Uddin. "Single Image Dehazing: An Analysis on Generative Adversarial Network", Journal of Computer and Communications, 2020 Publication	<1 %
35	Ceyhun Yildiz, Hakan Acikgoz, Deniz Korkmaz, Umit Budak. "An improved residual-based convolutional neural network for very short-term wind power forecasting", Energy Conversion and Management, 2021 Publication	<1 %
36	Fan Guo, Jin Tang, Hui Peng. "A Markov Random Field Model for the Restoration of Foggy Images", International Journal of Advanced Robotic Systems, 2014 Publication	<1 %
37	"Artificial Intelligence", Springer Science and Business Media LLC, 2022 Publication	<1 %

38	Yue Wang, Yongkai Yin, Xiulun Yang, Qian Sun, Xiangfeng Meng, Guoliang Shentu, Baoqing Sun. "Polarimetric dehazing based on fusing intensity and degree of polarization", Optics & Laser Technology, 2022 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

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