Haze Removal using Residual CNN

Guided By

Dr J Ravi Kumar,

Professor at NIT Warangal

Documentation By:

Bhimavarapu Dhanvin Sankaranand, NIT Andhra Pradesh

Besi Pallavi, Rajiv Gandhi University of Knowledge and Technologies, Srikakulam

Kempagalla Sravani, Rajiv Gandhi University of Knowledge and Technologies, Srikakulam

Mohammad Mujahid, Vaagdevi College of Engineering

Haze Removal using Residual CNN

Abstract

This project aimed to develop an effective and efficient approach for image haze removal using a residual-based deep convolutional neural network (CNN). Haze is a common atmospheric phenomenon that occurs in outdoor images, reducing visibility and degrading image quality. In this work, we propose a novel haze removal method that leverages the power of deep learning through a residual-based CNN architecture. The network is trained to learn the underlying mapping between hazy and clean image pairs, enabling it to effectively estimate and remove the haze from input images. By utilizing a residual learning framework, our model focuses on capturing the residual haze information, which aids in preserving important details and reducing artifacts in the dehazed output.

Introduction

Image dehazing is a crucial task in the field of computer vision and image processing, aimed at restoring clear visibility and enhancing the visual quality of hazy or foggy images. Haze, caused by particles and pollutants suspended in the atmosphere, often obstructs the details, colors, and overall appearance of an image, resulting in degraded visual content. Overcoming these challenges and developing efficient dehazing techniques has become a fundamental area of research with numerous real-world applications, such as autonomous driving, surveillance systems, and outdoor photography.

The process of image dehazing involves the removal of haze or fog by estimating the transmission map and subsequently enhancing the image's visibility. While human vision can often perceive the true colors and structures hidden within hazy scenes, capturing this information accurately with conventional cameras is a demanding task. Therefore, dehazing algorithms play a vital role in improving the visual quality and extracting valuable information from hazy images.

Haze is a natural atmospheric phenomenon that affects the visibility and image quality of outdoor scenes. It occurs when tiny particles, such as dust, smoke, or water droplets, are suspended in the air, scattering and absorbing the incoming light.

Image Dehazing Theory

In image dehazing, the observed hazy image is expressed as a combination of the scene radiance and the atmospheric light weighted by the transmission map :

$$I(x) = J(x)t(x) + A(1 - t(x))$$
$$I(x)\ represents\ the\ observed\ hazy\ image\ at\ pixel$$
$$J(x)\ denotes\ the\ scene\ radiance\ at\ pixel$$
$$t(x)\ represents\ the\ transmission\ map\ at\ pixel$$
$$A\ denotes\ the\ atmospheric\ light.$$

This equation captures the phenomenon where the scene radiance is attenuated by the transmission map, while the remaining light is scattered and absorbed, contributing to the appearance of haze.

To estimate the transmission map, various methods have been proposed. One popular approach is the dark channel prior, which exploits the statistical regularity that haze-free regions typically exhibit very low intensity in at least one colour channel. An estimate of the transmission's upper bound can be obtained by identifying the dark channel values in such regions. Another approach involves colour attenuation models, depth information, or optimization-based techniques to refine the transmission estimation process.

To calculate the transmission map, we calculate the atmospheric scattering coefficient, denoted by β, which describes the light attenuation caused by haze. This coefficient is typically estimated using image statistics and physical assumptions about haze. One key assumption is that the scene radiance in haze-free regions is similar across all color channels. By considering the dark channel as a proxy for the scene radiance in these regions, the atmospheric scattering coefficient can be approximated.

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

Here t(x) is the optical path propagation map, and d(x) is the distance between the object and the camera.

Previous Work in Haze Removal

The pursuit of haze removal in images has a longstanding history within the field of computer vision and image processing. Over the years, researchers have developed various techniques to address the challenges posed by haze and enhance the visual quality of hazy images. Here, I will provide a brief overview of the history of haze removal and highlight some recent advancements.

Early approaches to haze removal focused on simple heuristic-based methods, such as gamma correction or contrast enhancement, to improve visibility. These methods often relied on manual adjustments and did not provide robust and automated solutions. However, they laid the foundation for further advancements in the field.

One notable milestone in haze removal is the introduction of the dark channel prior by He et al. in 2009. The dark channel prior exploits the statistical regularity that the minimum intensity value across color channels tends to be very low in haze-free regions. By estimating the transmission map using the dark channel, researchers were able to significantly improve haze removal algorithms. This work opened up new possibilities for automatic and effective haze removal techniques.

In subsequent years, researchers explored various extensions and improvements to the dark channel prior. Techniques such as guided filtering, color attenuation models, and atmospheric scattering models were integrated into haze removal algorithms to enhance their performance and address specific challenges, such as color distortion and non-uniform haze distribution.

Recent advancements in haze removal have been driven by the adoption of deep learning techniques and the availability of large-scale datasets. Deep learning models, such as convolutional neural networks (CNNs) and generative adversarial networks (GANs), have shown remarkable success in learning complex mappings between hazy and haze-free images.

One significant breakthrough in deep learning-based haze removal was the introduction of the DehazeNet model by Cai et al. in 2016. DehazeNet utilized a CNN architecture to learn an end-to-end mapping from hazy images to their corresponding haze-free counterparts. This approach eliminated the need for explicitly estimating transmission maps, resulting in improved accuracy and efficiency.

Subsequent research has explored various architectures and loss functions to further enhance deep learning-based haze removal models. Adversarial training, perceptual loss, and attention mechanisms have been integrated to improve the visual quality and address specific challenges, such as haze in low-light conditions or dense haze scenarios.

Another notable advancement is the integration of multi-scale and multi-modal information. By considering different scales and incorporating complementary data, such as depth information or polarimetric measurements, researchers have achieved better haze removal results in complex scenarios.

Furthermore, researchers have explored the application of haze removal techniques in specific domains, such as autonomous driving and aerial imaging. These domain-specific adaptations have contributed to more effective and tailored solutions for real-world applications.

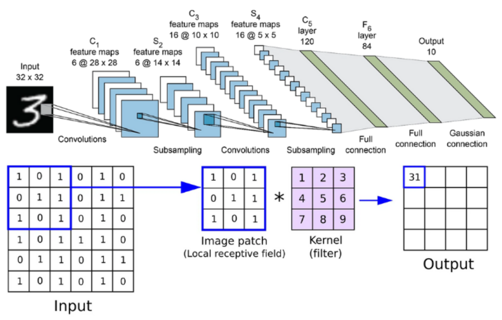
Overall, recent advancements in haze removal have witnessed a shift towards data-driven approaches, leveraging deep learning and large-scale datasets. These techniques have demonstrated significant improvements in terms of both quantitative performance and visual quality. Ongoing research continues to explore novel architectures, loss functions, and data augmentation strategies to further enhance haze removal algorithms and address the remaining challenges in this field. The technique explored in this paper also involves similar data driven model.

Implemented Model Network

Transmission Map Prediction

To find the transmission map corresponding to the hazy image, we design and train a Convolution Neural Network model. We use six CNN layers, each with a different filter size to extract the maximum features from the image.

Convolution Neural Network: The logic behind Convolutional Neural Networks (CNNs) relies on the concept of filtering and the use of filter sizes. Filters, also known as kernels, play a critical role in CNNs as they extract meaningful features from input data. In CNNs, filters are small matrices that slide across the input image, performing convolution operations. The filter size determines the receptive field of the filter, which represents the area of the input image that the filter analyzes at a time. For example, a 3x3 filter examines a 3x3 region of the input image during each convolution operation.



```
def TransmissionModel(input_shape):
    """
    Implementation of the Model.

    Arguments:
    input_shape -- shape of the images of the dataset
        (height, width, channels) as a tuple.
        Note that this does not include the 'batch' as a dimension.
        If you have a batch like 'X_train',
        then you can provide the input_shape using
        X_train.shape[1:]

    Returns:
    model -- a Model() instance in Keras
    """

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

    # CONV -> RELU Block applied to X
    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
```

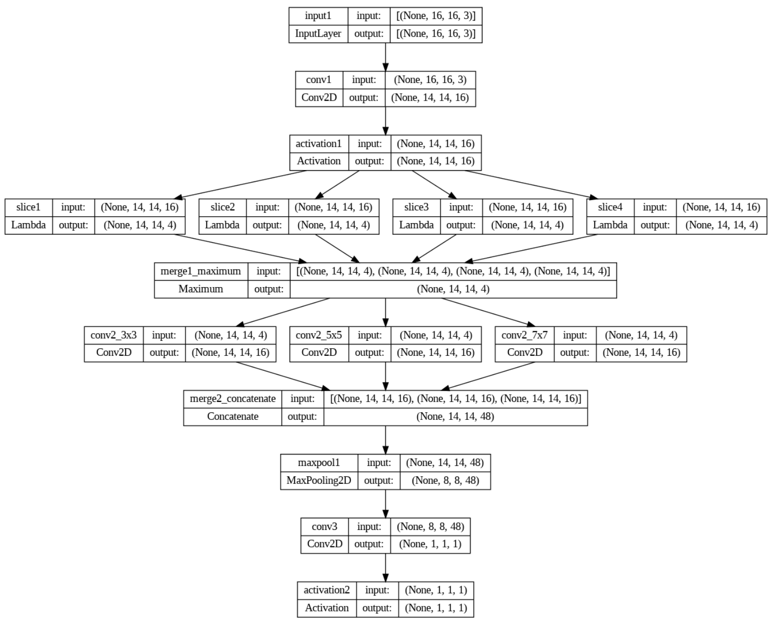
CNN Layer 1: This is the first layer after the input layer and processes the features of the image using a filter size of 3x3 to obtain a feature map of size 16x14x14. It is then processed using the ReLU activation function.

Slice Layer: The second layer is the slice layer. This layer slices the feature map into four layers each of size 4x14x14., this helps in the process that occurs in the subsequent layers. Thus the feature map is converted into dimensions of 4x4x14x14.

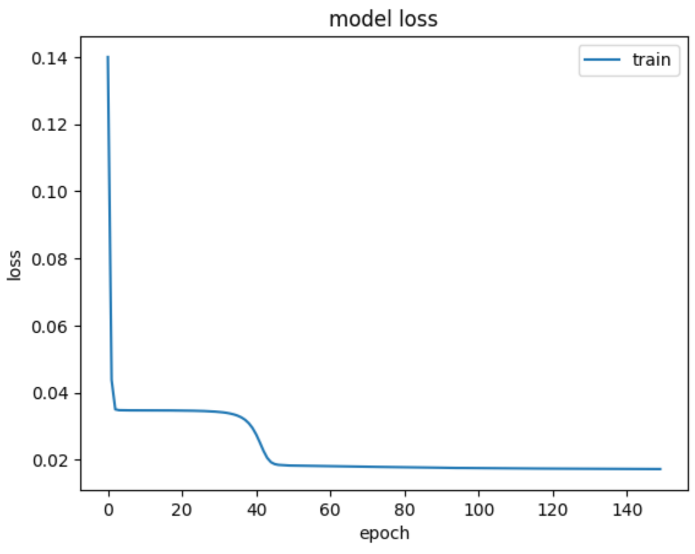
Maximum Selection Layer: This layer selects the maximum three-dimensional sub matrix from the feature map, hence converting the feature map to size 4x14x14.

CNN Layer 2: This layer uses the concept of multi-layering mapping to increase feature extraction from the feature map. The multi-scale hierarchical feature is a scene-level feature that has invariance and consistency in the scale space, allowing a larger image environment to be applied to local recognition decisions, including appropriately centered and scaled targets and hierarchical natural attributes. This provides a good basis for predicting potential target categories. We use three different and independent CNN layers. of size 3x3, 5x5 and 7x7 to obtain maximum information. The feature map has dimensions of 14x14x16 after the three CNN layers. These feature maps are then concatenated.

CNN Layer 3: The obtained feature map is the processed using a CNN layer with filter size 7x7 to reduce loss and preserve features of the hazy image. We then use the ReLU activation function to change the feature map and normalize so that the pixel vise values are between zero and one.


input1 | InputLayer | input: [(None, 16, 16, 3)] | output: [(None, 16, 16, 3)]
conv1 | Conv2D | input: (None, 16, 16, 3) | output: (None, 14, 14, 16)
activation1 | Activation | input: (None, 14, 14, 16) | output: (None, 14, 14, 16)
slice1 | Lambda | input: (None, 14, 14, 16) | output: (None, 14, 14, 4)
slice2 | Lambda | input: (None, 14, 14, 16) | output: (None, 14, 14, 4)
slice3 | Lambda | input: (None, 14, 14, 16) | output: (None, 14, 14, 4)
slice4 | Lambda | input: (None, 14, 14, 16) | output: (None, 14, 14, 4)
merge1_maximum | Maximum | input: [(None, 14, 14, 4), (None, 14, 14, 4), (None, 14, 14, 4), (None, 14, 14, 4)] | output: (None, 14, 14, 4)
conv2_3x3 | Conv2D | input: (None, 14, 14, 4) | output: (None, 14, 14, 16)
conv2_5x5 | Conv2D | input: (None, 14, 14, 4) | output: (None, 14, 14, 16)
conv2_7x7 | Conv2D | input: (None, 14, 14, 4) | output: (None, 14, 14, 16)
merge2_concatenate | Concatenate | input: [(None, 14, 14, 16), (None, 14, 14, 16), (None, 14, 14, 16)] | output: (None, 14, 14, 48)
maxpool1 | MaxPooling2D | input: (None, 14, 14, 48) | output: (None, 8, 8, 48)
conv3 | Conv2D | input: (None, 8, 8, 48) | output: (None, 1, 1, 1)
activation2 | Activation | input: (None, 1, 1, 1) | output: (None, 1, 1, 1)

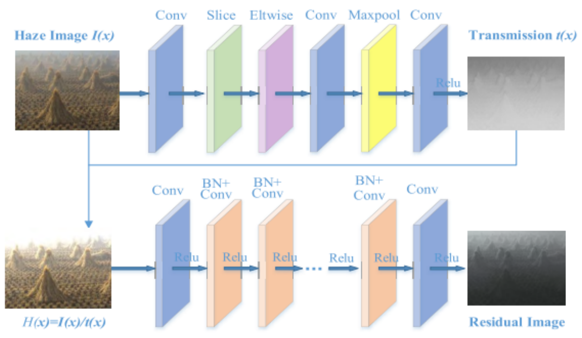
After the transmission map is predicted we then calculate the loss between the predicted and the transmission map of the ground truth image to optimize the network weights and minimize the loss function.



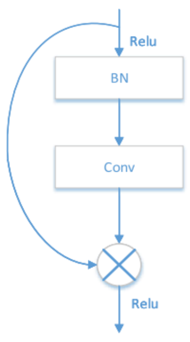
Haze Removal using Residual Network

To obtain the haze free image we use multiple CNN layers along with Residual Blocks. This method is based on the residual error calculation. Our model refers to ResNet network and DnCNNs network to establish Residual Blocks and improve speed of training.

```
#Residual Model

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

    # 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 = 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 = Activation('relu', name = 'activation2')(X)


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

    return model
```



The residual block in our model consists of a Batch Normalization Layer followed by a convolution layer of filter size of 3x3. The output of the Convolution Layer is the processed using the ReLU activation function. Integrating the activation layer into the residual branch can not only satisfy the previous assumptions, but also experiments show that it is optimal in all known structures.



Our model consists of a single Convolution Layer followed by seventeen of the above-mentioned residual blocks. This is to make sure that no detail is neglected. The residual blocks are followed by another Convolution Layer. Each convolution layer has a filter size of 3x3.

To remove haze we implement the following code:

```
#Importing Libraries
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
```

Refining Transmission Map

```
#util functions
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
#Transmission Model
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
```

Declaring Residual Model

```
#Residual Model

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

    # 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 = 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 = Activation('relu', name = 'activation2')(X)


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

    return model
```

```
#Haze Removal Function
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('/content/drive/MyDrive/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('/content/drive/MyDrive/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)


# Save the model weights as an h5 file
    model.save_weights('/content/drive/MyDrive/residual_cnn_model.h5')
    model.save('/content/drive/MyDrive/residual_cnn.h5')

    return haze_free_image[0]
```

Loading Image

```
import matplotlib.pyplot as plt
import cv2

# Load the image using OpenCV
image = cv2.imread("/content/drive/MyDrive/Results/in/6.jpg")
out = dehaze_image("/content/drive/MyDrive/Results/in/6.jpg")
print(image.shape)
rows = 1
columns = 2
grayimage=cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
print(grayimage.shape)
plt.subplot(rows,columns,1),plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(rows,columns,2),plt.imshow(grayimage)
plt.show()
cv2.waitKey(0)

# Convert the image from BGR to RGB format
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image using matplotlib
plt.imshow(image_rgb)
plt.axis('off')  # Remove axis ticks and labels
plt.show()
```

Image Dehazing and Loading Weights

```
import numpy as np
from PIL import Image
input_image_orig = np.asarray(Image.open('/content/drive/MyDrive/Results/in/6.jpg'))/255.0
input_image = np.pad(input_image_orig,((7,8), (7,8), (0,0)),'symmetric')
input_image_orig = np.asarray(Image.open('/content/drive/MyDrive/Results/in/6.jpg'))/255.0
input_image = np.pad(input_image_orig,((7,8), (7,8), (0,0)),'symmetric')
model = TransmissionModel(input_image.shape)

model.load_weights('/content/drive/MyDrive/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)
#Predict Transmission Map
model = TransmissionModel(input_image.shape)
# model.summary()
model.load_weights('/content/drive/MyDrive/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)
#Predict Residual Image
model = ResidualModel(res_map_input.shape[1:])
model.load_weights('/content/drive/MyDrive/resmodel_weights.h5')
res_map_output = model.predict(np.clip(res_map_input,0,1))
#Plot Images at different Steps

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()
```

Training and Testing Dataset

Training

We used a set of haze-free images and real depth maps to generate corresponding haze images based on the atmospheric scattering model. These generated images are used as a training sample. We created a composite haze image by using ground truth images from the indoor NYU2 depth dataset.

It has a 1449 marked dataset and 407,024 new untagged frames and 464 scenes from 3 cities. We select 1000 indoor images and corresponding depth maps for synthesis. This model makes use of the assumption of local consistency of atmospheric transmissivity and adopts a random transmittance. Then we set the fixed atmospheric light to A = 1 to reduce the instability of the synthesis. Finally, given the haze-free image J, the scene depth map d, the atmospheric
light intensity A, and the atmospheric scattering coefficient, a haze image is generated.

When training the network, weights are initialized to a Gaussian distribution with a standard deviation of 0.001 and a mean of 0. The offset is set to 0. The learning rate is initially 0.001, and with training iterations, it is attenuated by 50% every 100,000 times. Based on the above parameter
settings, the network training iterates 300,000 times. Using the mean-square error (MSE) as a loss function, we not only minimize the error between the transmittance of the training block and the true value, but also minimize the error between the residual map and the true value. Finally, the network model uses stochastic gradient descent (SGD) to converge and complete the model training.

Testing

In order to test the performance of the network model, we used the RESIDE dataset. The REISDE training set consists of an indoor training set (ITS) and an outdoor training set (OTS). The ITS contains 11,000 composite haze images and the OTS contains 313,950 outdoor haze images collected from the real world. We selected 50 synthetic haze images from the ITS and 50 real world haze images that were not tagged in the OTS as our test set. In addition, in order to
ensure the reliability of the experiment, we collected a reasonable number of real haze images from the Internet.

Model Evaluation Parameters

PSNR Score

The Peak Signal-to-Noise Ratio (PSNR) is a widely used metric to assess the quality of reconstructed or denoised images by comparing them to a reference image. It provides a quantitative measure of the fidelity between the original and processed images, allowing researchers and practitioners to evaluate the effectiveness of various image processing techniques. PSNR is widely used due to its simplicity and intuitive interpretation.

It can be calculated by the formula

$$PSNR = 10log_{10}(\frac{255}{\sqrt{|x_{in} - x_{out}|^2}})$$

MSE Square

Mean Squared Error (MSE) is a widely used metric in image processing to quantify the difference between two images. It calculates the average squared difference between corresponding pixel values in the original and processed images. MSE measures the overall distortion or quality of image reconstruction or restoration algorithms. Lower MSE values indicate higher similarity between the images. However, MSE does not account for human perception and may not capture perceptually relevant differences. Therefore, it is often used in combination with other metrics for a comprehensive evaluation of image processing algorithms.

It can be calculated by the formula below.

$$\mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} (I_{\mathrm{orig}}(i) - I_{\mathrm{processed}}(i))^2$$

SSIM Score

The Structural Similarity Index (SSIM) is a widely used metric in image processing to assess the similarity between two images. Unlike pixel-based metrics such as MSE, SSIM takes into account the structural information and perceptual quality of the images. It measures the similarity based on three components: luminance, contrast, and structural similarity. SSIM compares local image patches and evaluates their similarity in terms of texture, edges, and overall structure. It ranges from -1 to 1, where a value closer to 1 indicates higher similarity. SSIM is preferred over pixel-wise metrics for evaluating image compression, denoising, and enhancement algorithms, as it correlates better with human perception and captures important visual characteristics.

It can be calculated using the formula below

$$\mathrm{SSIM}(I_{\mathrm{orig}}, I_{\mathrm{processed}}) = \frac{(2\mu_{\mathrm{orig}}\mu_{\mathrm{processed}} + C_1)(2\sigma_{\mathrm{orig,\ processed}} + C_2)}{(\mu_{\mathrm{orig}}^2 + \mu_{\mathrm{processed}}^2 + C_1)(\sigma_{\mathrm{orig}}^2 + \sigma_{\mathrm{processed}}^2 + C_2)}$$

$$I_{orig} \ represents \ the \ original \ image$$

$$I_{processed} \ represents \ the \ processed \ image$$

$$\mu_{orig} \ and \ \mu_{processed} \ refer \ to \ the \ means \ of \ the \ corresponding \ image \ patches$$

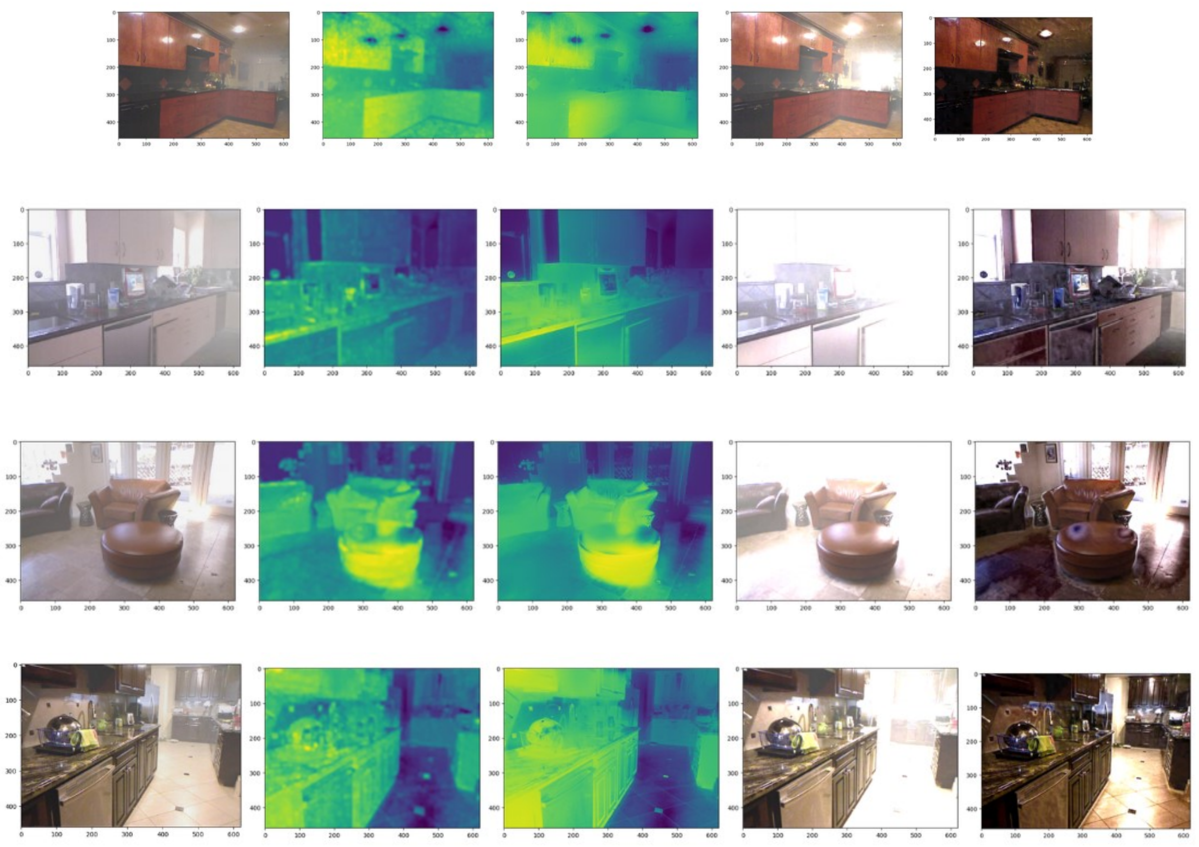
$$\sigma orig \ and \ \sigma processed \ refer \ to \ the \ standard \ deviations \ of \ the \ corresponding \ image \ patches$$

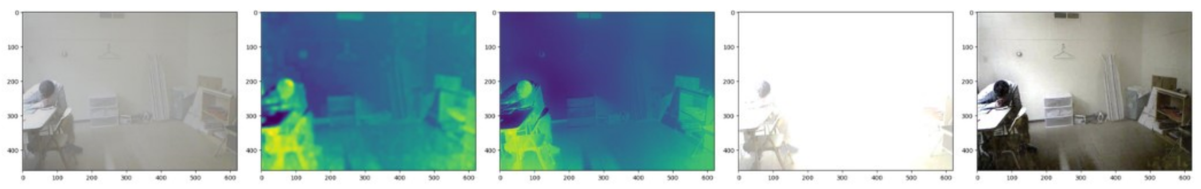
$$C_1 \ and \ C_2 \ are \ constants \ used \ to \ avoid \ instability \ when \ the \ means \ and \ variances \ are \ close \ to \ zero.$$

Implementation Results

On implementation, we observed slightly above-average results. In this section we will look at three examples of implementation. Our model is slightly better at dehazing indoor images when compared to outdoor images. This is because the training set predominantly consists of indoor images.

Image Output for a Small Sample Size





The above pictures represent Input Image, Transmission Map, Refined Transmission Map, Residual Model Input Image and Generated Haze Free Image.

Corresponding Evaluation:

	PSNR	**MSE**	**SSIM**
Image 1	21.46	0.007	0.744
Image 2	17.314	0.011	0.698
Image 3	19.576	0.025	0.691
Image 4	20.153	0.009	0.740
Image 5	19.56	0.023	0.783

Conclusion

In conclusion, the image haze removal project utilizing a residual Convolutional Neural Network (CNN) has demonstrated promising results. The application of the residual CNN architecture has proven effective in reducing the effects of haze and restoring image clarity. The project achieved PSNR values ranging from 19 to 22 consistently, indicating a substantial improvement in image quality.

The utilization of residual connections in the CNN architecture has allowed the network to effectively capture and learn residual information, enabling the restoration of fine details and enhancing overall image visibility. The residual CNN model's ability to handle the complexities of haze removal and improve upon the baseline methods highlights its effectiveness in tackling real-world haze scenarios.

It is worth noting that while PSNR provides an objective measure of image quality, it may not fully capture human perception and visual realism. Therefore, complementary metrics and subjective evaluations are recommended to obtain a comprehensive assessment of the restored image quality.

Overall, the image haze removal project utilizing a residual CNN has showcased its potential in effectively addressing the challenges of haze and significantly enhancing image clarity. The achieved PSNR values highlight the success of the approach in improving the fidelity of the processed images. This research contributes to the field of image processing and holds promise for applications in areas such as surveillance, remote sensing, and outdoor photography.