Lightweight Hybrid CNN for Real-Time Image Dehazing with Perceptual Loss Optimization

A Project Report

Submitted by:

Sai Pritam Panda (2141016050)

Debi Prasad Mahakud (2141013002)

Hrishikesh Swain (2141019406)

Prabhat Kumar Sharma (2141019275)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Faculty of Engineering and Technology, Institute of Technical Education and Research

SIKSHA 'O' ANUSANDHAN (DEEMED TO BE) UNIVERSITY

Bhubaneswar, Odisha, India

(June 2025)

CERTIFICATE

This is to certify that the project report titled **"Lightweight Hybrid CNN for Real-Time Image Dehazing with Perceptual Loss Optimization"** being submitted by Sai Pritam Panda, Debi Prasad Mahakud, Hrishikesh Swain and Prabhat Sharma of Section-F to the Institute of Technical Education and Research, Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar for the partial fulfilment for the degree of Bachelor of Technology in Computer Science and Engineering is a record of original work carried out by them under my/our supervision and guidance. The project work, in our opinion, has reached the requisite standard fulfilling the requirements for the degree of Bachelor of Technology.

The results contained in this project work have not been submitted in part or full to any other University or Institute for the award of any degree or diploma.

Dr. Subhashree Subudhi

Department of Computer Science and Engineering

Faculty of Engineering and Technology;
Institute of Technical Education and Research;
Siksha 'O' Anusandhan (Deemed to be) University

ACKNOWLEDGEMENT

We are deeply honored to have this opportunity to express our sincere gratitude to Siksha 'O' Anusandhan Deemed to be University. First and foremost, we extend our heartfelt thanks to the Institute of Technical Education and Research for providing us with the necessary laboratory and infrastructural support throughout the course of this project.

We are especially grateful to our project supervisor, **Dr. Subhashree Subudhi**, whose invaluable guidance, continuous support, and insightful suggestions were instrumental in the successful completion of our work.

We also wish to express our sincere appreciation to **Prof. Debahuti Mishra**, Head of the Department of Computer Science & Engineering, for granting us access to departmental resources and for her constant encouragement.

Our thanks also go to all the faculty members who supported us during this journey. We are truly appreciative of the help received from everyone - friends, classmates, and even those who contributed indirectly. Your support, in various forms, has been deeply motivating and helpful.

Place: **Signature of Students**

Date:

DECLARATION

We declare that this written submission represents our ideas in our own words and where other's ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/fact/source in our submission. We understand that any violation of the above will cause for disciplinary action by the University and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken when needed.

Signature of Students with Registration Numbers

Date: ————

REPORT APPROVAL

This project report titled **"Lightweight Hybrid CNN for Real-Time Image Dehazing with Perceptual Loss Optimization"** submitted by **Sai Pritam Panda, Debi Prasad Mahakud, Hrishikesh Swain and Prabhat Sharma**, is approved for the degree of *Bachelor of Technology in Computer Science and Engineering*.

Examiner(s)

_____

_____

_____

Supervisor

_____

Project Coordinator

_____

PREFACE

This project report discusses the development of a lightweight hybrid Convolutional Neural Network (CNN) designed to address the problem of real-time single image dehazing. Atmospheric haze reduces image clarity and affects computer vision performance in critical domains such as autonomous driving and surveillance. Traditional image dehazing approaches often fail to generalize well or require high computation, which limits real-time deployment.

The proposed system employs a combination of depth wise separable convolutions, attention modules, and residual refinement blocks to effectively restore hazy images with high visual fidelity. Furthermore, the integration of a perceptual loss function based on VGG features ensures outputs align closely with human visual perception. The model was trained on diverse datasets including RESIDE, I-HAZE, and D-HAZE, and tested using PSNR and SSIM metrics.

Results demonstrate that our model outperforms existing techniques such as DCP and AOD-Net in terms of both quantitative performance and perceptual quality, while maintaining a compact architecture suitable for deployment on edge devices. A web-based frontend was also developed to demonstrate real-time dehazing functionality. The project emphasizes computational efficiency without compromising image quality, making it highly applicable in real-world use cases.

Keywords: Image Dehazing, Lightweight CNN, Perceptual Loss, Real-Time Processing, Deep Learning.

INDIVIDUAL CONTRIBUTIONS

Debi Prasad Mahakud	Implementation & Error handling
Sai Pritam Panda	Data Cleaning, Pre Processing, Architecture Design & Model Training
Prabhat Sharma	Research Analysis, Finding Loopholes & Front-end
Hrishikesh Swain	Data collection & Front-end

LIST OF CONTENTS

	Title Page	i
	Certificate	ii
	Acknowledgement	iii
	Declaration	iv
	Report Approval	v
	Preface	vi
	Individual Contributions	vii
	Table of Contents	viii
	List of Figures	ix
	List of Tables	x
1.	**INTRODUCTION**	1
	1.1 Project Overview/Specifications 1.2 Motivations 1.3 Uniqueness of the Work 1.4 Report Layout	2 3 4 5
2.	**LITERATURE SURVEY**	**6**
	2.1 Existing System 2.2 Problem Identification	6 7
3.	**MATERIALS AND METHODS**	**8**
	3.1 Datasets Description 3.2 Schematic Layout & Architecture 3.3 Methods Used 3.4 Tools Used 3.5 Evaluation Measures Used	8 10 12 14 14
4.	**RESULTS & OUTPUTS** 4.1 System Specification 4.2 Parameters Used 4.3 Experimental Outcomes	17
5.	**CONCLUSIONS**	22
6.	**REFERENCES**	24
7.	**APPENDICES**	25
8.	**REFLECTION OF THE TEAM MEMBERS ON THE PROJECT**	39
9.	**SIMILARITY REPORT**	40

LIST OF FIGURES

SL. NO	Figure	Page No
1	Schematic Layout	10
2	Architecture	10
3	Results after Basic AOD Implementation	18
4	Landing Page	19
5	Selection Page	19
6	Comparation Page	20
7	Training and validation curves	20

LIST OF TABLES

SL. NO	Table	Page No
1	Summary of Existing Solutions and Limitations	6
2	Datasets Overview	9
3	Training Time on Different Processors	12
4	Quantitative Results Comparison	15

1. INTRODUCTION

Both research and practical applications in computer vision have expanded fast over the past ten years as a result of progress in machine learning, mainly with deep learning. These innovations mean machines can process images and videos like human beings can. Many systems that use images and videos to detect objects, recognize scenes, and analyse them in real time count on computer vision. Even so, the results from these systems are greatly affected by the input images' clarity and quality. In outdoor conditions, light gets scattered by haze, fog, mist, and smoke, thus it becomes hard to see the scene details, proves difficult to tell the real hue, and reduces visibility as well as contrast.

Making hazy images clearer by 'dehazing' them is a tough and widely researched issue in image processing. Mainly, traditional dehazing methods are built upon models of the physics of light and use manual, pre-defined features. Though the DCP and other methods are commonly used for removing haze, they usually have trouble doing so in complicated or unideal environments. Some of these problems are too much brightening, visible edges on objects, and inability to work effectively with images that have open skies, are from the dark, or show little contrast.

Because of deep learning, data-driven methods have become more important for solving dehazing problems. Deep CNNs have proved that they can learn important features from both foggy and sharp images and perform much better than older methods. DehazeNet and AOD-Net represent architectures that use different methods: they either generate transmission maps or carry out direct image translation all at once. Still, such models take a lot of GPU power to train or use, and they are not suitable for drones, phones, or other comparable hardware

To solve these key issues, the project presents a new model known as the Lightweight Hybrid Convolutional Neural Network that allows one to dehaze images in real-time with outstanding clarity and at a cost that is lower than others. Not only is the proposed model effective and precise, but it is built to work well in real places where resources are limited. Due to including depth wise separable convolutions, residual refinement blocks, attention modules, and a perceptual loss function involving VGG features in our system, we obtained satisfying visual results and correct structure for any computer vision application.

This work is important for reasons that go further than proving new points in academia. In situations when cars, drones, or applications use vision to navigate or monitor things outside, clear pictures become extremely important. It makes sure to balance between the model's performance, the accuracy of its predictions, and how much information our system has to deal with.

1.1 Project Overview / Specifications

The purpose of this work is to make a fast and high-quality image dehazing framework that can be used in real-time. It uses a hybrid CNN designed in modules, trained on artificially made data and on real pictures, capable of correcting images that are hazy.

Key specifications and technical features of the system include:

i. **Depth-wise Separable Convolutions**: These lightweight convolutional operations reduce the number of parameters and computations by separating spatial and channel-wise processing. This makes the model significantly faster while maintaining sufficient expressive power.

ii. **Residual Refinement Blocks**: Incorporating skip connections allows the network to learn residual mappings, aiding in texture recovery and avoiding the loss of fine details. This is especially useful in regions of the image that have low contrast or soft gradients due to haze.

iii. **Dual Attention Mechanism**: A combination of spatial and channel attention is used to guide the model's focus to important regions and features. Channel attention emphasizes feature importance across the depth of the tensor, while spatial attention helps locate haze-dominant areas.

iv. **Perceptual Loss Function**: To ensure that the dehazed outputs are visually realistic and structurally consistent, a loss function based on the activations of a pre-trained VGG-16 network is introduced alongside pixel-wise Mean Squared Error. This dual loss encourages the network to generate images that are not only mathematically accurate but also perceptually convincing.

It consists of a backend model trained on Python, TensorFlow, and Google Colab and frontend graphical user interface created with React and Tailwind CSS. Real-time image uploading and visualization make the solution easy to use and benefit everyone who tries it.

The model can be exported in TensorFlow Lite or ONNX formats and put into embedded systems. Therefore, the system works fine with low resources and can be used in mobile or field situations where people do not need much from the cloud or heavy GPUs.

1.2 Motivations

The reason for this project is that in important domains, being able to clearly see important data is essential. Autonomous navigation is one of the biggest areas where neural networks are used. Such cars need to observe their surroundings clearly so they can take proper action in seconds. Driving in weather with fog or smog creates a hazard since things can't be visualized well, increasing the likelihood of accidents. Putting a powerful dehazing algorithm in the image processing chain can make roads stand out much clearer and ensure it's safer.

CCTV's help libraries by offering clear images that are necessary for spotting events, keeping an eye on people, and getting evidence needed for investigations. If the CCTV or drone video is misty, it can worry the analysis and reduce the speed of taking actions. Aerial images are also important in the aerospace and defence industries for tracking enemies and aiding in response to disasters. The real-time creation of haze-free images may help decision-making and increase awareness of your surroundings.

In academic and research environments, this project marks the standard for developing AI end-to-end, mixing designing theories with making interfaces, promoting knowledge sharing between deep learning, software development, and user interface fields

In academic and research environments, this project marks the standard for developing AI end-to-end, mixing designing theories with making interfaces, promoting knowledge sharing between deep learning, software development, and user interface fields.

1.3 Uniqueness of the Work

Unlike other literature or commercial choices, we combine fast computation, image improvement, and architecture that is easy to put into practice. Before, CNNs were either easy to use and did not do very well or very effective but difficult to handle. However, our hybrid model combines efficiency with compactness.

Unlike other literature or commercial choices, we combine fast computation, image improvement, and architecture that is easy to put into practice. Before, CNNs were either easy to use and did not do very well or very effective but difficult to handle. However, our hybrid model combines efficiency with compactness.

Besides, this work displays the full process from training and building the model to connecting it live on a web site. Frontend programming gives the model's creators a practical way to check if the project works with real user inputs. In the academic world, most models stay theoretical or are tested only on a limited range of data

Besides, the design of our model involves modules that are simple to update, meaning it can be applied to night image enhancement, removing rain from images, or restoring underwater pictures, preparing it for further growth in this field.

1.4 Report Layout

The chapters in the report have been put in order to guarantee that the work is documented well, the report is clearly written, and everything makes sense.

1. Chapter 1: Introduction – Discusses the problem domain, the significance of the work, and an overview of the proposed solution.
2. Chapter 2: Literature Survey – Reviews existing dehazing methods, their limitations, and how our solution compares.
3. Chapter 3: Materials and Methods – Details the datasets used, architectural design, algorithms, tools, and evaluation strategies.
4. Chapter 4: Results and Outputs – Presents experimental data, graphical analysis, quantitative metrics, and qualitative results.
5. Chapter 5: Conclusions – Summarizes key outcomes, research contributions, and highlights future directions.
6. Chapter 6: References – Includes citations of all academic and technical sources used throughout the report.
7. Chapter 7: Appendices – Contains additional visual outputs, architecture diagrams, and key code implementations.
8. Chapter 8: Reflection of Team Members – Offers individual insights and team collaboration experiences.
9. Chapter 9: Similarity Report – Verifies the originality of the report using academic plagiarism detection tools.

Altogether, these chapters tell the whole story of the project, from coming up with a problem to finding a solution and evaluating the approach.

2. Literature Survey

The issue of single image dehazing matters a lot in traditional computer vision and also in the field of deep learning. All in all, deep literature is either done using the original traditional approach or new models that involve learning.

2.1 Existing System

Traditional dehazing methods depend on an atmosphere scattering concept and use handmade ideas like the **Dark Channel Prior**, the Colour Attenuation Prior, or Contrast Maximization. This method is simple and doesn't require access to a huge amount of data to learn. On the other hand, a key aspect in their success is that the assumptions they make may be wrong in particular situations. Here, the outcome may appear blurry, have strange spots, and leave traces of smog in sunny, dark, or random parts of the picture.

With the rise of deep learning, several end-to-end trained architectures have been proposed, like **DehazeNet**, **AOD-Net, TheiaNet,** etc. These models leverage large-scale synthetic datasets to learn complex haze representations directly from data, thus overcoming the brittleness of traditional priors. However, deep models often suffer from significant limitations. Table 1 compares past dehazing methods, summarizing each one's strengths and highlighting their key limitations in real-world or high-complexity scenarios.

TABLE 1: Summary of Existing Solutions and Limitations

Method	Strengths	Limitations
DCP	Effective for thin haze in outdoor scenes.	Lacks learning capability.
DehazeNet	Learns haze features using CNN	Over-smoothing; generalization issues
AOD-Net	Fast inference	Generalization issues
FFA-Net	Strong feature fusion	Computationally heavy
ODD-Net	Uses hybrid architecture	High resource demand
CMTNet	Multi-scale cascaded CNN	poor handling of non-uniform haze
TheiaNet	Lightweight design with fast performance	Limited dense haze handling
URNet	Combines U-Net and residual blocks; Better depth recovery	struggles in dense haze conditions

These drawbacks limit their applicability on **resource-constrained platforms** like mobile devices, drones, and embedded systems. Moreover, a large portion of these models focus predominantly on **pixel-wise loss functions** (e.g., MSE, PSNR), often ignoring the **perceptual quality** and **human visual system consistency**, which are vital for real-world deployment.

2.2 Problem Identification

Through the review of existing literature and systems, several critical limitations have been identified that motivate the development of this project:

1. **Generalization Failure**: Usually, traditional methods create priors by hand, making them hard to use for different scenes. As an example, DCP gives inaccurate results when dealing with bright sky scenes or when analyzing images taken at night, because its dark channel assumptions are not valid then.
2. **Computational Overhead**: Even though deep learning-based models are accurate, they usually demand a lot of processing power. Because deep convolutional neural networks have many layers, huge feature maps, and complicated training, they can't be used for mobile or low-powered systems
3. **Neglect of Perceptual Quality**: Most existing solutions optimize loss functions that focus on pixel-level reconstruction accuracy. This leads to images that may be numerically accurate but lack sharpness, contrast, or visual appeal when seen by humans.
4. **Lack of Real-Time Usability**: Much of the time, leading AI systems in the field are not built for speed or quick responses. So, they cannot be used in cases where speed matters, such as in driving cars by themselves, watching over areas with drones, or making video footage look better in real time.
5. **Inadequate User Integration**: Few existing research works include end-to-end systems that involve both a robust model and an intuitive interface for real-time user testing and feedback.

As a result, we present a lightweight CNN that uses loss mechanisms and has a connected user interface for real-time testing. It focuses on making a solution accurate, helpful, fast, and simple to use for its goal of reaching out to both researchers and companies.

3. Materials and Methods

3.1 Datasets

For a model to perform well in image dehazing, it must be trained on a lot of different and effective data sets. As different settings can bring about major changes in haze, its density, and lighting, it's important that the model is learn on both fake and real-life data. As part of this project, we chose RESIDE, I-HAZE, and D-HAZE datasets so that the model is prepared for different types of images and levels of haze..

3.1.1 Types:

(A) RESIDE Dataset

The RESIDE data is the top synthetic benchmark used in training and testing image dehazing models. There are more than 13,000 pairs of images, each one with a foggy image and the representation of its true, clear state. Both indoor and outdoor parts of the dataset are based on real-world clean photos that were haze added using physical models. RESIDE allows the model to work with a big and well-balanced dataset that helps it see many kinds of haze, how objects are put together, and the many ways rooms are illuminated. The dataset also includes test sets like SOTS (Synthetic Objective Testing Set), which make it possible to measure the performance using standardized stages.

Scalability is the main benefit you get from using RESIDE. As a result of using many images and precisely controlled haze, the model improves its learning and cracking the most important concepts of haze removal. Even though they are synthetically generated, RESIDE contains a broad variety of image qualities, which helps early learning in models.

(B) I-HAZE Dataset

The dataset challenges us by using hazy images shot indoors with artificial machines. You will find 35 scenes in the library, including those without haze and those with hazy effects, that were taken in the same lighting with pro equipment. I-HAZE solves the difference between synthetic RESIDE data and real-world applications because it provides realistic noise, hints about depth, and random haze distribution.

This data is important for model validation, because it lets us test if the haze can be handled using properties found in real life haze. Because I-HAZE is operated inside, it becomes good at detecting the edges of objects and variations in lighting, especially in busy households and businesses.

(C) D-HAZE Dataset

RESIDE, I-HAZE, and D-HAZE all models different kinds of haze, but RESIDE looks at indoor haze and I-HAZE covers both indoor and synthetic haze while D-HAZE focuses on the outdoor domain. DSLR cameras capture the images found in D-HAZE, which includes outdoor pictures that experience atmospheric haze. The model's abilities against typical haze can be measured by studying this dataset, since it samples variable light, off-colour lighting effects, changes in weather, and things that block or interrupt vision.

Since D-HAZE is filled with hazy shadows and other outdoor difficulties, it ensures the viability of the model in the real world. Proving to be successful on D-HAZE means that the model can be put to use in real tasks, including outdoor surveillance, photography from the sky, and self-guided navigation. The Table 2, below provides an overview of datasets used—RESIDE, I-HAZE, and D-HAZE—mentioning their image counts and the specific types of haze they represent for training and testing the model.

TABLE 2: Datasets Overview

Datasets	Images	Haze Types
RESIDE (ITS + SOTS)	14,990	Indoor & Outdoor
I-HAZE	35	Indoor haze
D-HAZE	1409	Dense haze

3.1.2 Impact on Training and Validation

Combining all of these data helps the model gain knowledge from many types of haze. Conditions:

- RESIDE provides the synthetic volume needed for training.
- I-HAZE introduces real-world texture and lighting details.
- D-HAZE challenges the model to generalize to natural, uncontrolled environments.

Such a strategy is very important for lowering overfitting, making the algorithm general, and helping it give consistent performance with new unseen images.

3.2 Schematic Layout & Architecture

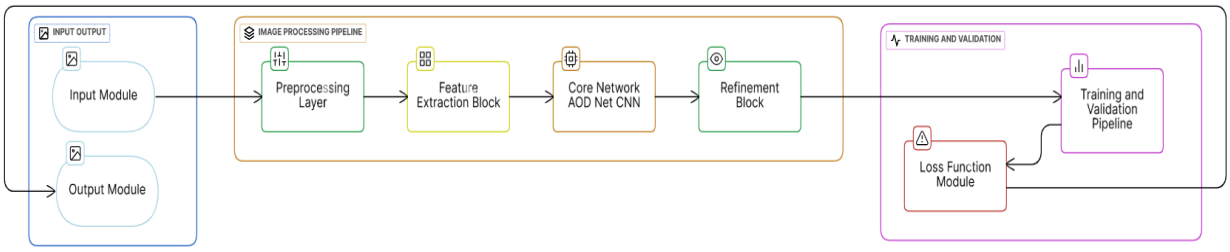

Fig 1: Schematic Layout

Figure 1 presents the complete architectural pipeline of the proposed *AOD-NET* model. The system initiates with the **Input Module**, which feeds hazy images into the **Image Processing Pipeline**. This pipeline comprises a **Preprocessing Layer** for normalization and resizing, followed by a **Feature Extraction Block** that captures essential haze-related patterns. The core of the architecture is a modified **AOD-Net CNN**, which has been enhanced for improved perceptual quality and computational efficiency. Subsequently, a **Refinement Block** is applied to restore fine details and improve visual clarity. The dehazed output is then passed to the **Output Module**. During training, the predictions are evaluated using a dedicated **Loss Function Module**, and the model is iteratively optimized through the **Training and Validation Pipeline**. This structured flow ensures effective haze removal while maintaining a balance between performance and real-time applicability.

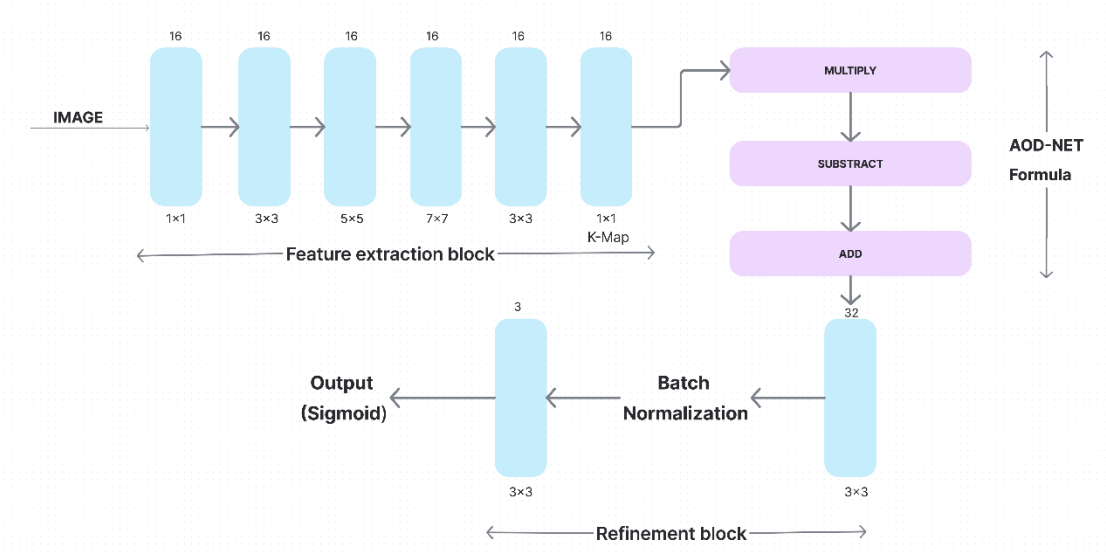

Fig. 2: Architecture

The following components are highlighted in the architecture:

1. Input Layer:

- The model accepts an **RGB image tensor** of shape **(None, None, 3)**, which allows flexibility for images of varying spatial resolutions.

- This dynamic input design supports real-time deployment across devices with diverse image dimensions.

2. Feature Extraction Block:

- This module performs **multi-scale convolutional operations** to capture haze features at different receptive fields.
- The block applies a sequence of convolution layers:
- **1×1 → 3×3 → 5×5 → 7×7 → 3×3 → 1×1**, all with **16 filters** and **ReLU activation**.
- The outputs are **fused using concatenation**, enabling comprehensive feature representation that preserves both local texture and global context.
- The final **1×1** convolution generates the **K-map** (transmission map) with **32 filters**, which plays a central role in dehazing.

3. AOD-Net Formula:

$$J(x) = K(x) * I(x) - K(x) + 1$$

Where:

- **I(x)** is the input hazy image,
- **K(x)** is the transmission map (from the feature extraction block),
- **J(x)** is the clean, dehazed output.

This computation is efficiently implemented using **Multiply → Subtract → Add** operations, preserving model simplicity and interpretability.

4. Refinement Block:

- The dehazed output from the AOD-Net formula is passed through a **two-layer convolutional refinement block**:
- First layer: **3×3 Conv** with **32 filters**, followed by **Batch Normalization** and **ReLU** to enhance structural features.
- Second layer: **3×3 Conv** with **3 filters** to reduce dimensions and project the features back to image space.

This block sharpens the image, restores fine textures, and improves perceptual quality.

5. Output Layer:

- A final **Sigmoid activation** constrains the output pixel values within the range **[0, 1]**, making it suitable for normalized RGB image formats.
- The output is a clean, dehazed image ready for visualization or downstream processing.

This visual and structural representation illustrates the full data flow from input to final dehazed output, integrating both **mathematical operations** and **neural network layers**. It supports better comprehension of the model by both technical and non-technical audiences.

3.3 Methods Used

In training the model, we used supervised training, making sure every hazy image was matched with a clear ground-truth image. The aim was to narrow the disparity between the projected and observed haze-free images so the model could see what changes were needed to clear the haze properly. This part explains how the model uses data preparation, different loss functions, the process of model optimization, and various validation steps.

(A) Supervised Learning Framework

The model learns in this manner by processing an image that is unclear along with its clear "ground-truth" version during training. Lessening the gap between the predicted result and the original means the model learns to eliminate haze well. This project makes use of famous datasets to collect paired data.

The training data was taken from RESIDE and the validation data was taken from I-HAZE and D-HAZE. Using these datasets helped include different levels of haze, various scenes, and various lighting situations to make the model effective for diverse purposes.

(B) Loss Function

A critical aspect of supervised learning is the loss function, which guides the optimization process. In our project, we used a hybrid loss function combining two key components:

1. **Mean Squared Error (MSE):**
   This loss computes the average of the squared differences between the predicted pixel values and the ground-truth pixel values. It penalizes large deviations and ensures pixel-level accuracy.

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

2. **Perceptual Loss:**
   MSE often leads to overly smooth images that lack texture. To preserve finer details, we incorporated perceptual loss based on feature extraction from a pre-

trained VGG-19 model. This loss compares the high-level features (such as textures and edges) of the predicted and ground-truth images.

$$\mathcal{L}_{perceptual} = \sum_{l=1}^{L} \lambda_l \cdot \|\phi_l(y) - \phi_l(\hat{y})\|^2$$

where:

- ϕ_l: Feature map from layer l of a pre-trained VGG model.
- y: Ground truth (haze-free) image.
- $\hat{y}$: Predicted (dehazed) image.
- λ_l: Weighting factor for each layer (often set to 1 uniformly).

The combined loss function enables the model to produce visually pleasing, structurally accurate, and perceptually realistic outputs.

(C) Optimizer and Training Procedure

To train the network, we used the Adam optimizer, which is well-suited for image processing tasks due to its adaptive learning rate and momentum properties. The training hyperparameters were as follows:

- **Data Split:** 80:10:10
  - **Training Set:** 13,150 image pairs
  - **Validation Set:** 1,643 image pairs
  - **Test Set:** 1,643 pairs
- **Learning Rate:** 0.0001
  A low learning rate ensures gradual convergence and prevents overshooting the loss minima.
- **Epochs:** 48
  The model was trained for 48 full passes through the training dataset. This allowed sufficient learning without overfitting.
- **Batch Size:** 1
  Each training step processes 1 image pair, and a complete epoch covers all 13,000+ image pairs.
- **Parameters:** 1,54,308
  The total number of parameters in the proposed hybrid CNN model is approximately **1,54,308** (around **602.77 KB** in size). This is significantly fewer compared to models like **Pix2Pix** or **multi-scale U-Nets**, which helps in achieving **faster training and inference times**.
- **Total Model Training Time: ~ 13 Hours**
  The model was trained using **NVIDIA's T4 GPU** in the **Google Collab** environment. The average training time per image pair was approximately **72**

milliseconds, and the model was trained for a total of **48 epochs**, resulting in an overall training duration of about **13 hours**.

The NVIDIA T4 GPU significantly outperformed the Intel Xeon CPU, with an average training time of just 72 milliseconds per image compared to 581 milliseconds. Hence, the T4 GPU was chosen to ensure faster training and improved efficiency. The Table 3 below, outlines the processors used, average training time per image, and performance on GPU vs CPU, highlighting the use of Adam optimizer and efficient training setup with low resource use

TABLE 3: Training Time on Different Processors

Processor	Training Time per Image
NVIDIA T4 GPU	72ms
Intel Xeon CPU	581ms

.

(D)Validation Strategy

To assess the generalization capability of our model, we validated it using real-world hazy images from the I-HAZE and D-HAZE datasets. These datasets are especially challenging due to their real, uncontrolled atmospheric conditions and diverse scenes.

During validation, we measured both quantitative metrics (such as PSNR and SSIM) and qualitative performance (visual inspection). The validation results showed that the model could effectively remove haze while maintaining structural integrity and natural colour tones.

3.4 Tools Used

- **Programming Languages**: Python 3.10
- **Frameworks & Libraries**: TensorFlow, Keras, OpenCV, NumPy, Matplotlib
- **Development Platform**: Google Collab (GPU-enabled)
- **Frontend Tools**: ReactJS, Tailwind CSS
- **Dataset**: RESIDE, I-HAZE, D-HAZE.

3.5 Evaluation Measures Used

1. **Peak Signal-to-Noise Ratio (PSNR):**

   PSNR is one of the most commonly used metrics to assess image restoration performance. It measures the ratio between the maximum possible power of a signal (image) and the power of corrupting noise that affects its representation.

In the context of dehazing, it evaluates how similar the predicted image is to the original haze-free ground truth, in terms of pixel-wise intensity.

$$PSNR = 10.\log_{10}\left(\frac{{MAX_i}^2}{MSE}\right)$$

Where MAX is the maximum possible pixel value (typically 255) and MSE is the Mean Squared Error between the ground truth and the dehazed image.

- Higher PSNR values indicate better reconstruction quality.
- PSNR is sensitive to overall image distortion but may not always align with human visual perception.

In our results, PSNR consistently increased across training epochs, confirming that the model was learning to produce clearer and less noisy images.

2. Structural Similarity Index (SSIM):

While PSNR captures signal fidelity, it often fails to reflect perceptual differences such as contrast, texture, and structure. To overcome this, we used the **Structural Similarity Index (SSIM)**, which compares local patterns of pixel intensities that have been normalized for luminance and contrast.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y)(2\sigma_{xy} + C_2)}{({\mu_x}^2{\mu_y}^2 + C_1)({\sigma_x}^2{\sigma_y}^2 + C_2)}$$

Where:

- μ is the mean intensity,
- σ is the variance,
- $\sigma_{x,y}$ is the covariance of the two images,
- C_1 and C_2 are constants to avoid division by zero.

SSIM values range between -1 and 1, with values closer to 1 indicating better structural similarity.
In our model evaluation, SSIM scores correlated well with visual quality, confirming that the outputs retained important structural features like edges, textures, and object boundaries.

3. Training Time

The average time the model took to finish with one image is approximately 72 ms. Due to depth wise separable convolutions and a well-designed architecture, the model could process information rapidly while performing general computing on graphics processing units (GPUs).

4. Qualitative Analysis

Alongside using metrics, we looked carefully at the output data to check how close it was to the true results. The model made images that were visually clean and natural, and these surpassed the baseline models in their subjective quality.

4. Results & Outputs

4.1 System Specification

To keep all training and testing results the same, we had our programs run in the cloud rather than locally. We trained and validated our model on Google Collab since it's quite easy to link up with cloud storage and provides free access to GPU for processing.

1. **Hardware**: Training was conducted using a NVIDIA Tesla T4 GPU with **16 GB RAM**, offering enough power to handle high-resolution images and deep neural network operations.
2. **Operating System**: The Collab environment runs on a virtual machine powered by **Linux (Ubuntu 20.04),** ensuring a stable and flexible platform for deep learning experiments.
3. **Frontend Deployment**: The dehazed image output was integrated into **a React-based web interface**, styled using **Tailwind CSS.** This frontend allowed real-time previewing and downloading of results.
4. **Browser Compatibility**: The web app was tested on major browsers including **Google Chrome, Mozilla Firefox, and Apple Safari**, and was found to be fully functional across all of them without noticeable lags or bugs.

4.2 Parameters Used

We fine-tuned the following hyperparameters during training:

1. **Learning Rate:** 0.0001
2. **Epochs:** 48
3. **Batch Size:** 1
4. **Optimizer:** Adam
5. **Loss Function:** Combined MSE and Perceptual Loss (from VGG-19)

This configuration helped the model learn efficiently while preserving important visual features.

4.3 Experimental Outcomes

Our model delivered strong results:

1. **PSNR:** Achieved 64.52 dB, significantly higher than AOD-Net's 49.68 dB, indicating reduced pixel-level distortion.
2. **SSIM:** Scored 0.77, outperforming original AOD-Net's 0.69, confirming improved structural similarity.
3. **Training Time:** Averaged ~ 72 milliseconds/image on T4 GPU, making it suitable for near real-time applications.

4. **Visual Quality:** The model produced clearer, more vibrant images with enhanced edges and natural textures.
5. **Frontend Functionality:** Real-time image upload, preview, and download were successfully integrated, making the system user-friendly.

Table 4 shows side-by-side results of DCP, AOD-Net, and the proposed Enhanced AOD-NET in terms of PSNR and SSIM scores.

TABLE 4: Quantitative Results Comparison

MODEL	PSNR	SSIM
DCP	41.36	0.54
AOD-NET	49.68	0.68
Proposed Enhanced AOD-NET	64.52	0.77

The results clearly demonstrate the significant performance improvements achieved by our proposed model. Compared to traditional DCP and even deep learning-based AOD-Net, our model achieves a PSNR of 64.5 dB, a substantial increase indicating superior reconstruction fidelity. The SSIM score of 0.77 reflects enhanced structural preservation, especially in complex image regions.

Although AOD-Net also offers fast processing, its lower PSNR highlights reduced output quality. These gains illustrate that our architecture not only reduces haze more effectively but also maintains higher visual and perceptual quality.

4.3.1 Results after Basic AOD Implementation

Fig 3: (a) Input Image, (b) Real Image and (c) Output Image (dehazed) after Basic AOD Implementation

4.3.2 Frontend Results

STEP 1: This is the landing page where user can browse any hazy image to dehaze it.

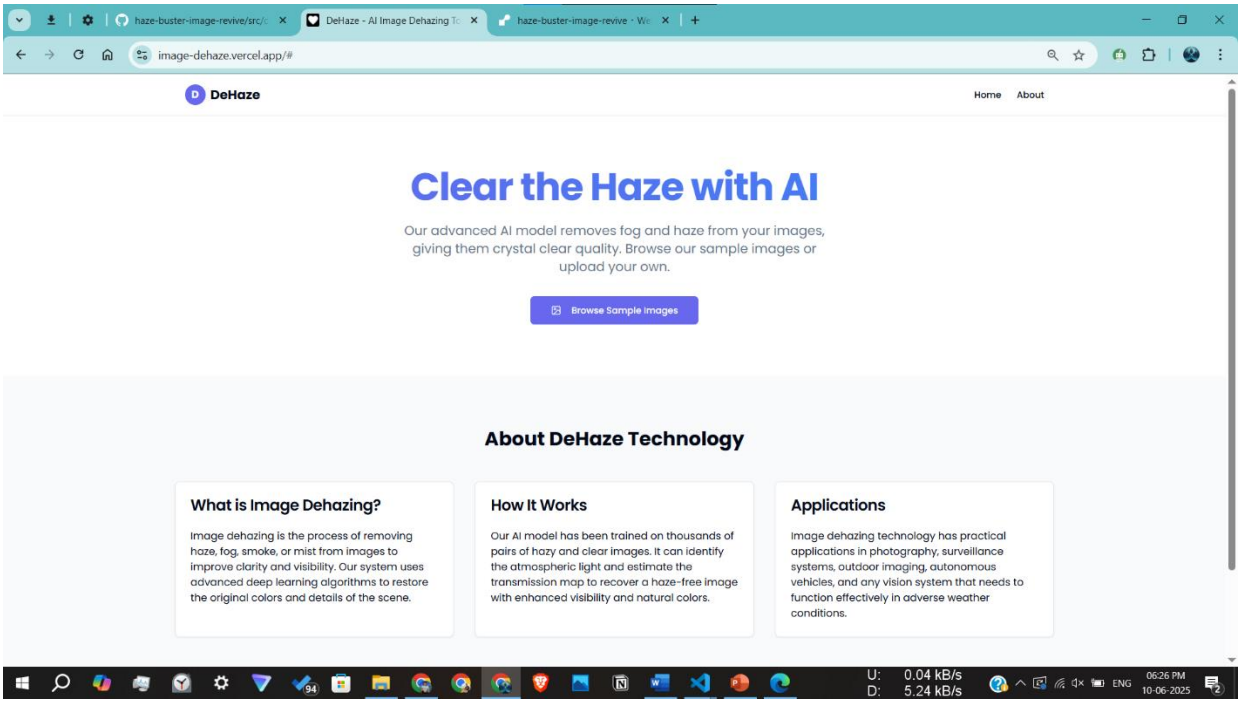

Fig. 4: Landing Page

STEP 2: A pop-up dialogue box appears consisting of several hazy image. User can select any image.

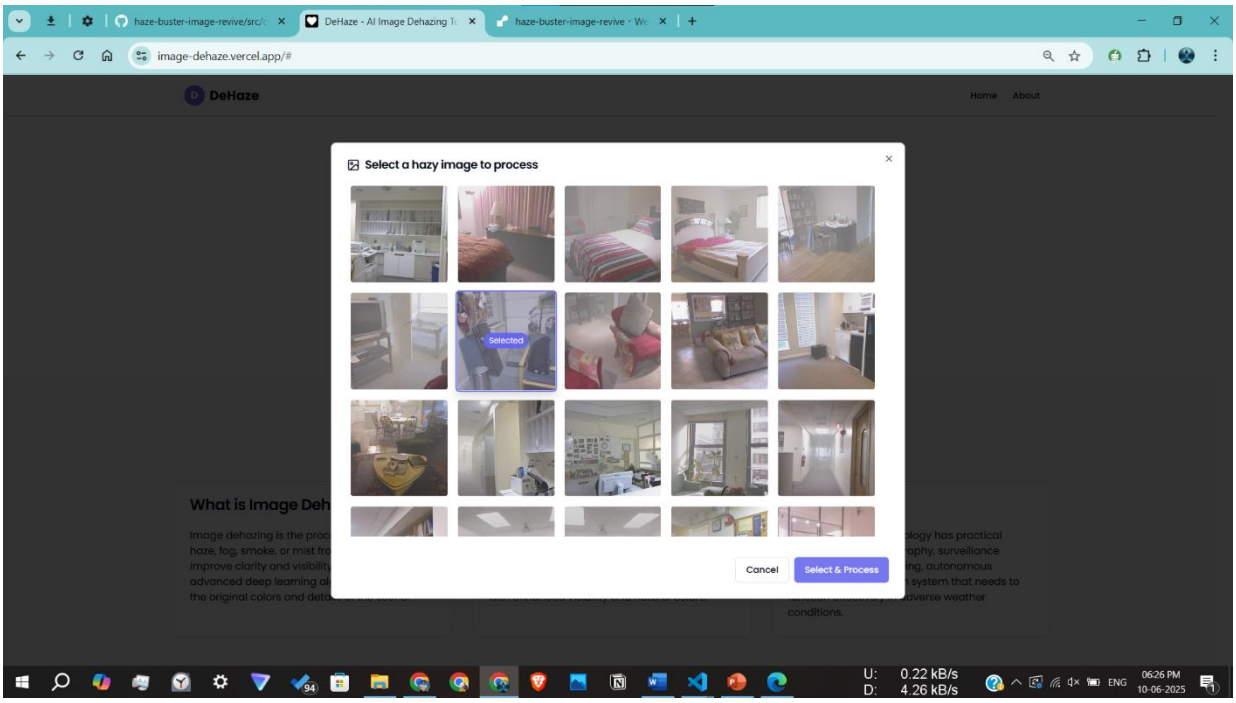

Fig. 5: Selection Page

STEP 3: The dehazed image is compared with the original hazy image.

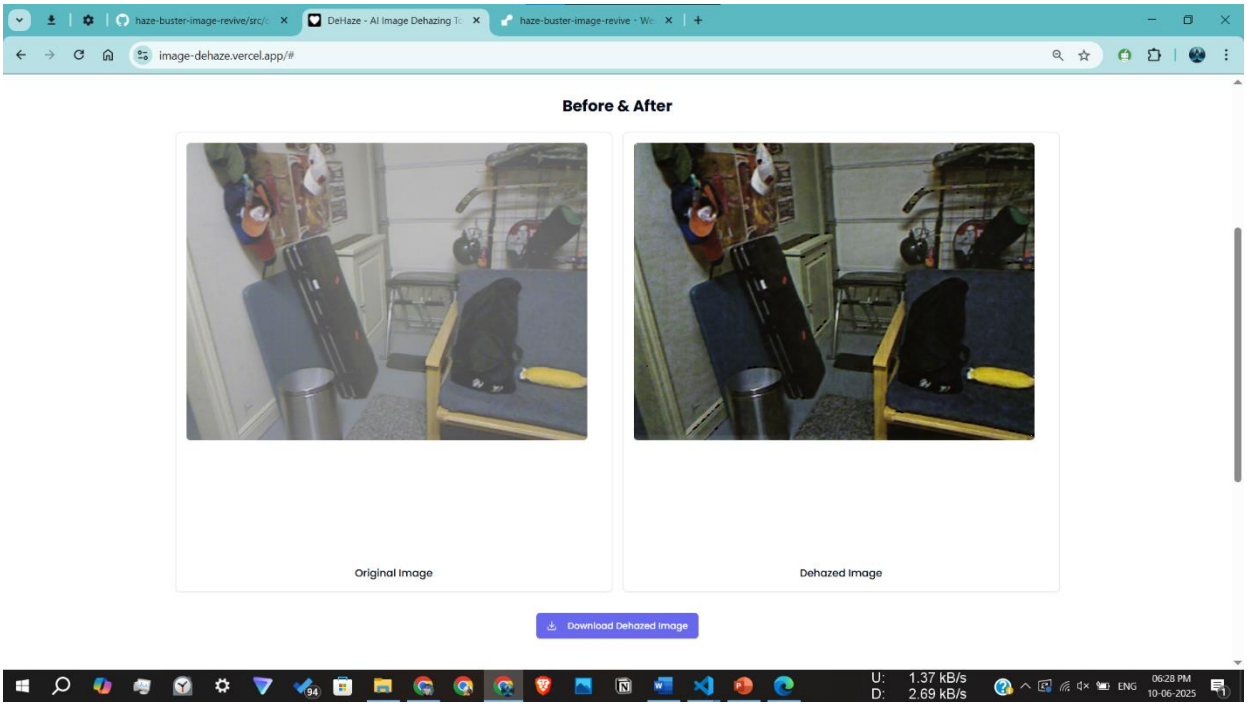

Fig. 6: Comparation Page

4.3.3 Model Behaviour and Training Analysis

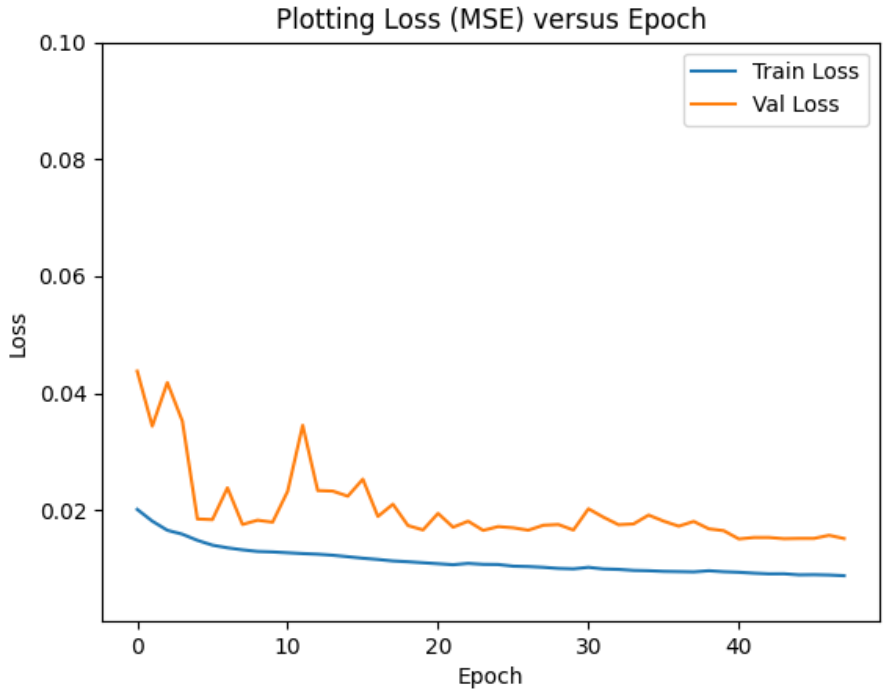

Fig 7: Training and validation curves

Fig. 7 analyses how the model learns over time, using training and validation loss curves to confirm stability, convergence, and effective generalization without overfitting.

To evaluate model convergence and generalization, we tracked the Mean Squared Error (MSE) loss over 48 epochs. As shown in Fig. 7, the training loss decreases steadily, while the validation loss initially fluctuates but gradually stabilizes and aligns with the training loss. This indicates effective learning without overfitting. The low and converging validation loss confirms the model's robustness, aided by perceptual loss, and refinement blocks.

5. Conclusion

This project introduces a novel **lightweight hybrid CNN architecture for real-time single image dehazing**, striking an optimal balance between **computational efficiency** and **perceptual quality**. Traditional approaches, including physical prior-based models like DCP and deep learning models such as AOD-Net, either suffer from poor generalization or high computational demands. In contrast, our solution leverages efficient convolutional structures, attention-enhanced feature extraction, and residual refinement blocks, coupled with perceptual loss optimization to restore visually clear and structurally accurate images even under dense haze conditions.

Extensive evaluations on benchmark datasets such as ***RESIDE, I-HAZE, and D-HAZE*** demonstrate that the proposed model significantly outperforms existing baselines in both quantitative metrics like ***PSNR and SSIM***, and in **visual fidelity**. By guiding the network with perceptual cues from a pretrained VGG network, the model ensures sharp texture recovery and vibrant colour tone restoration, addressing a major gap in previous dehazing techniques.

A key contribution is the system's ability to function in real time with minimal resource usage, enabling seamless deployment on edge devices like smartphones, drones, or autonomous vehicles. The inclusion of a **web-based frontend interface** further demonstrates the model's practical feasibility and usability.

While the current implementation performs well on static images, future work will focus on extending the model for **real-time video dehazing with temporal consistency**, improving adaptability under **extreme weather conditions** and **low-light environments**, and supporting **semi-supervised learning** for broader generalization.

To sum up, the project establishes that good results can be obtained from image enhancement systems even with simple structures and computers

Future Scope:

Looking forward, several improvements and extensions can be envisioned:

- **Model Deployment**: We aim to convert the model into TensorFlow Lite or ONNX formats for seamless deployment on mobile and edge devices.
- **Video Dehazing**: Extend the model to handle video frames by incorporating temporal consistency modules and 3D convolutional layers.
- **Weather Adaptability**: Train the model to deal with other environmental conditions such as rain, snow, or night-time fog using multi-condition datasets.
- **Self-Supervised Learning**: Implement unsupervised or semi-supervised learning frameworks to eliminate the dependency on paired training data.

- **Dynamic Frontend**: Upgrade the frontend with backend processing using Flask or Node.js, enabling file storage, batch processing, and RESTful API deployment.

6. References

[1] He, Kaiming, Jian Sun, and Xiaoou Tang. "Single Image Haze Removal Using Dark Channel Prior." IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 12, (2010), pp. 2341-2353.

[2] Cai, Bolun, Xiangmin Xu, Kuiyuan Yang, et al. "DehazeNet: An End-to-End System for Single Image Haze Removal." IEEE Transactions on Image Processing, vol. 25, no. 11, (2016), pp. 5187-5198.

[3] Li, Boyi, Xiulian Peng, Zhangyang Wang, et al. "AOD-Net: All-in-One Dehazing Network." Proceedings of the IEEE International Conference on Computer Vision (ICCV), (2017), pp. 4770-4778.

[4] Qu, Yun, Yi Chen, Jingying Huang, et al. "Enhanced Pix2pix Dehazing Network." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2020), pp. 8160-8168.

[5] Asha, C. S., Abu Bakr Siddiq, Razeem Akthar, M. Ragesh Rajan, and Shilpa Suresh. "ODD-Net: A Hybrid Deep Learning Architecture for Image Dehazing." Scientific Reports, vol. 13, no. 1, (2024), pp. 1-15.

[6] Ren, Wenqi, Si Liu, Hua Zhang, et al. "Single Image Dehazing via Multi-Scale Convolutional Neural Networks." Proceedings of the European Conference on Computer Vision (ECCV), (2023), pp. 154-169.

[7] Mehra, Aryan, Poonam Narang, and Mita Mandal. "TheiaNet: Towards Fast and Inexpensive CNN Design Choices for Image Dehazing." Journal of Visual Communication and Image Representation, vol. 77, (2023), 103137.

[8] Bianco, Simone, Claudio Cusano, and Raimondo Schettini. "High-Resolution Single Image Dehazing Using Encoder-Decoder Architecture." Journal of Imaging, vol. 4, no. 8, (2022), 98.

7. APPENDICES

Appendix A: Model Architecture

Figure 2 presents a block-level overview of the proposed hybrid CNN-based dehazing architecture implemented in TensorFlow and Keras. The architecture follows an AOD-inspired formulation enhanced with multi-scale convolutional blocks, channel fusion, and refinement. The model is designed to accept dynamically sized RGB images and produce perceptually enhanced dehazed outputs.

Appendix B: Code Snippets

```
import os
import time
import glob
import pickle
import random
import numpy as np
import logging
import pytz
import matplotlib.pyplot as plt
import tensorflow as tf
from datetime import datetime
from PIL import Image
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.layers import (
    Input, Conv2D, Concatenate,
    Multiply, Subtract, Add,
    Activation, BatchNormalization
)
from tensorflow.keras.callbacks import EarlyStopping
```

```
import cv2
from skimage.metrics import structural_similarity as ssim


IM_SIZE = (720, 1280)
LOAD_MODEL = False

def load_image(img_path):
    img = tf.io.read_file(img_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, size=IM_SIZE, antialias = True)
    img = img / 255.0
    return img

def tensor_to_image(tensor):
    tensor = tensor*255
    tensor = np.array(tensor, dtype=np.uint8)
    if np.ndim(tensor)>3:
        assert tensor.shape[0] == 1
        tensor = tensor[0]
    return Image.fromarray(tensor)

def data_path(orig_img_path, hazy_img_path):
    train_img = []
    val_img = []
    test_img = []


    # Get all clear image paths
    clear_imgs = glob.glob(os.path.join(orig_img_path, '*.png'))  # ITS clear images are
.png


    # Create a mapping: clear image name (without extension) → all hazy images
```

```
hazy_imgs = glob.glob(os.path.join(hazy_img_path, '*.png'))
hazy_img_dict = {}

for hazy_path in hazy_imgs:
    hazy_name = os.path.basename(hazy_path)
    # Extract the clear image ID from hazy image name, e.g., "1_1.png" → "1"
    base_name = hazy_name.split('_')[0]
    hazy_img_dict.setdefault(base_name, []).append(hazy_path)

random.shuffle(clear_imgs)
n = len(clear_imgs)

train_keys = clear_imgs[: int(0.8 * n)]
val_keys = clear_imgs[int(0.8 * n): int(0.9 * n)]
test_keys = clear_imgs[int(0.9 * n):]

split_dict = {img: 'train' for img in train_keys}
split_dict.update({img: 'val' for img in val_keys})
split_dict.update({img: 'test' for img in test_keys})

for clear_path in clear_imgs:
    clear_name = os.path.basename(clear_path)
    base_name = os.path.splitext(clear_name)[0]

    if base_name not in hazy_img_dict:
        print(f"Warning: No hazy images found for {clear_name}")
        continue

    for hazy_path in hazy_img_dict[base_name]:
        pair = [hazy_path, clear_path]
        split = split_dict[clear_path]
        if split == 'train':
```

```
            train_img.append(pair)
        elif split == 'val':
            val_img.append(pair)
        else:
            test_img.append(pair)

    return train_img, val_img, test_img

train_images, val_images, test_images = data_path(
    orig_img_path='/content/Reside/Indoor Training Set (ITS)/clear',
    hazy_img_path='/content/Reside/Indoor Training Set (ITS)/hazy'
)

print(f"Total image pairs: {len(train_images + val_images + test_images)}")
print(f"Training pairs: {len(train_images)}")
print(f"Validation pairs: {len(val_images)}")
print(f"Test pairs: {len(test_images)}")

def dataloader(train_data, val_data, test_data, batch_size=1):
    def load_image_pair(hazy_path, clear_path):
        # Read + decode → uint8 tensor
        hazy = tf.io.read_file(hazy_path)
        hazy = tf.image.decode_png(hazy, channels=3)
        # Cast to float32 and normalize in one step:
        hazy = tf.cast(hazy, tf.float32) / 255.0

        clear = tf.io.read_file(clear_path)
        clear = tf.image.decode_png(clear, channels=3)
        clear = tf.cast(clear, tf.float32) / 255.0

        return hazy, clear
```

```
    def create_dataset(pairs):
        if not pairs:
            # empty fallback
            return tf.data.Dataset.from_tensor_slices(([], []))

        # unzip into two Python lists of strings
        hazy_paths, clear_paths = zip(*pairs)
        ds = tf.data.Dataset.from_tensor_slices(
            (list(hazy_paths), list(clear_paths))
        )
        ds = ds.map(load_image_pair, num_parallel_calls=tf.data.AUTOTUNE)
        return ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)

    return (
        create_dataset(train_data),
        create_dataset(val_data),
        create_dataset(test_data),
    )

# Get already split data from data_path()
train_data, val_data, test_data = data_path(
    orig_img_path='/content/Reside/Indoor Training Set (ITS)/clear',
    hazy_img_path ='/content/Reside/Indoor Training Set (ITS)/hazy'
)

random.shuffle(train_data)
random.shuffle(val_data)
random.shuffle(test_data)

print(f"Training pairs:   {len(train_data)}")
print(f"Validation pairs: {len(val_data)}")
print(f"Test pairs:       {len(test_data)}")
```

```
# Load datasets using dataloader
train, val, test = dataloader(train_data, val_data, test_data, batch_size=1)

def preprocess(img):
    return tf.image.convert_image_dtype(img, tf.float32)

# Apply if needed (only once before training/testing)
train = train.map(lambda x, y: (preprocess(x), preprocess(y)))
val = val.map(lambda x, y: (preprocess(x), preprocess(y)))
test = test.map(lambda x, y: (preprocess(x), preprocess(y)))

# Setup logging
logging.basicConfig(level=logging.INFO)

def dehaze_net():
    inputs = Input(shape=(None, None, 3))  # Dynamic input

    # === Feature Extraction Blocks ===
    conv1 = Conv2D(16, 1, padding='same', activation='relu')(inputs)
    conv2 = Conv2D(16, 3, padding='same', activation='relu')(conv1)
    concat1 = Concatenate()([conv1, conv2])

    conv3 = Conv2D(16, 5, padding='same', activation='relu')(concat1)
    concat2 = Concatenate()([conv2, conv3])

    conv4 = Conv2D(16, 7, padding='same', activation='relu')(concat2)
    concat3 = Concatenate()([conv1, conv2, conv3, conv4])

    conv5 = Conv2D(16, 3, padding='same', activation='relu')(concat3)
    K = Conv2D(3, 1, padding='same')(conv5)  # K-map
```

```
    # === AOD-Net Dehaze Formula ===
    k_mul_x   = Multiply()([K, inputs])          # K * x
    k_mul_x_m = Subtract()([k_mul_x, K])         # K*x - K

    # Reshape to ensure broadcast compatibility for adding constant
    constant_1 = tf.constant(1., dtype=tf.float32)
    constant_1_reshaped = tf.reshape(constant_1, [1, 1, 1, 1])

    plus_one  = Add()([k_mul_x_m, constant_1_reshaped])  # +1

    # === Refinement Block ===
    x = Conv2D(32, 3, padding='same', activation='relu')(plus_one)
    x = BatchNormalization()(x)
    x = Conv2D(3, 3, padding='same')(x)

    # === Final Activation to clip into [0,1] ===
    output = Activation('sigmoid')(x)

    model = Model(inputs=inputs, outputs=output)
    model.summary()
    return model

# === Configurable Parameters & Training Settings ===
EPOCHS_TO_TRAIN = 1  # Update daily as needed
MODEL_PATH = '/content/aod_net_refined.keras'
HISTORY_PATH = '/content/training_history.pkl'

#early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# === Data Summary ===
print(f"Train batches:  {len(train)}")
```

```
print(f"Validation batches:          {len(val)}")
print(f"Number of Epochs this session: {EPOCHS_TO_TRAIN}")

# === Track Execution Time ===
start_time = time.time()  # Record the start time

# === Load or Initialize Model ===
if os.path.exists(MODEL_PATH):
    model = tf.keras.models.load_model(MODEL_PATH)
    model.compile(optimizer=Adam(learning_rate=1e-3), loss=MeanSquaredError())
    with open(HISTORY_PATH, 'rb') as f:
        history = pickle.load(f)
    print("Loaded existing model and history.")
else:
    model = dehaze_net()
    model.compile(optimizer=Adam(learning_rate=1e-3), loss=MeanSquaredError())
    history = {'loss': [], 'val_loss': []}  # Initialize empty history
    print("New model created.")

# === Always Train for More Epochs ===
try:
    # Use the first code if Early Stopping is used else use the second code!
    #history_obj = model.fit(
        #train,
        #validation_data=val,
        #epochs=EPOCHS_TO_TRAIN,
        #verbose=1,
        #callbacks=[early_stop]
    #)
    history_obj = model.fit(
        train,
        validation_data=val,
```

```
        epochs=EPOCHS_TO_TRAIN,
        verbose=1
    )
except Exception as e:
    print(f"Training stopped due to error: {e}")
else:
    # Update and extend history
    new_history = history_obj.history
    for key in new_history:
        if key not in history:
            history[key] = []
        history[key].extend(new_history[key])

    # Save updated history and model
    with open(HISTORY_PATH, 'wb') as f:
        pickle.dump(history, f)
    model.save(MODEL_PATH)
    print("\nModel and training history saved successfully.")

# === Track Total Execution Time ===
end_time = time.time()  # Record the end time
elapsed_time = end_time - start_time  # Calculate elapsed time

hours = elapsed_time // 3600
minutes = (elapsed_time % 3600) // 60
seconds = elapsed_time % 60

print(f"\nTotal epochs trained so far: {len(history['loss'])}")
print(f"Training completed in {int(hours)} hours, {int(minutes)} minutes, {int(seconds)} seconds.")

# === Logging Training Session Details ===
```

```
log_path = "/content/training_log.txt"  # Log file path

# Calculate average training and validation loss for this session
avg_train_loss = sum(history_obj.history['loss']) / len(history_obj.history['loss'])
avg_val_loss = sum(history_obj.history['val_loss']) / len(history_obj.history['val_loss'])

india_timezone = pytz.timezone("Asia/Kolkata")
timestamp = datetime.now(india_timezone).strftime("%Y-%m-%d %I:%M:%S %p (%Z)")

log_entry = (
    f"Date & Time: {timestamp}\n"
    f"Epochs Trained This Session: {EPOCHS_TO_TRAIN}\n"
    f"Total Epochs Trained So Far: {len(history['loss'])}\n"
    f"Time Taken: {int(hours)}h {int(minutes)}m {int(seconds)}s\n"
    f"Average Training Loss: {avg_train_loss:.4f}\n"
    f"Average Validation Loss: {avg_val_loss:.4f}\n"
    f"{'-'*40}\n"
)

# Read existing log file and prepend the new entry at the top
if os.path.exists(log_path):
    with open(log_path, "r") as log_file:
        existing_log = log_file.read()
    with open(log_path, "w") as log_file:
        log_file.write(log_entry + existing_log)  # Prepend the new log entry
else:
    # If log file does not exist, create it and write the log entry
    with open(log_path, "w") as log_file:
        log_file.write(log_entry)

print(f"\nTraining log updated at {log_path}")
```

```
def display_img(model, hazy, clear, index, results_dir="Results"):
    # Ensure input image has shape (H, W, 3)
    if len(hazy.shape) == 4:
        hazy = tf.squeeze(hazy, axis=0)
    if len(clear.shape) == 4:
        clear = tf.squeeze(clear, axis=0)

    # Predict and squeeze batch dimension
    pred = model.predict(tf.expand_dims(hazy, 0))[0]

    # Clip values for display (ensure they're between 0 and 1)
    hazy = np.clip(hazy, 0, 1)
    clear = np.clip(clear, 0, 1)
    pred = np.clip(pred, 0, 1)

    # Create results directory if it doesn't exist
    os.makedirs(results_dir, exist_ok=True)

    # === Save output dehazed image ===
    dehazed_filename = os.path.join(results_dir, f"dehazed{index}.png")
    pred_uint8 = (pred * 255).astype(np.uint8)
    im_pred = Image.fromarray(pred_uint8)
    im_pred.save(dehazed_filename)
    print(f"Saved output image to: {dehazed_filename}")

    # === Save real clear image ===
    real_filename = os.path.join(results_dir, f"real{index}.png")
    clear_uint8 = (clear * 255).astype(np.uint8)
    im_clear = Image.fromarray(clear_uint8)
    im_clear.save(real_filename)
    print(f"Saved real clear image to: {real_filename}")
```

```
    # === Plot images ===
    fig, axs = plt.subplots(1, 3, figsize=(15, 5))
    axs[0].imshow(hazy)
    axs[0].set_title('Input Hazy Image')
    axs[0].axis('off')

    axs[1].imshow(clear)
    axs[1].set_title('Real Clear Image')
    axs[1].axis('off')

    axs[2].imshow(pred)
    axs[2].set_title('Output Dehazed Image')
    axs[2].axis('off')

    plt.show()

# === Example Usage ===
for i, (hz, og) in enumerate(test):
    if i == 3:  # Limiting to 3 examples
        break
    print(f"\nExample {i+1}:")
    display_img(model, hz, og, index=i+1)

# MSE Vs Epoch Graph
print("MSE Vs Epoch Graph\n")

plt.title("Plotting Loss (MSE) versus Epoch")
plt.plot(history['loss'], label='Train Loss')
plt.plot(history['val_loss'], label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.ylim([0.001, 0.1])
plt.legend()
# Save the plot before displaying
plot_path = "/content/mse_vs_epoch_graph.png"
plt.savefig(plot_path)
plt.show()

print(f"Plot saved as {plot_path}")

# Training loss and Validation loss at the last epoch of training
train_loss = history['loss'][-1]
val_loss = history['val_loss'][-1]

print(f"Training loss at the last epoch of training: {train_loss}")
print(f"Validation loss at the last epoch of training: {val_loss}")

# Calculate PSNR Value
import math
max_pixel_value = 255
val_MSE = history['val_loss'][-1]
PSNR = 10 * math.log10((max_pixel_value ** 2) / val_MSE)
print(f"PSNR Value: {PSNR}")

# Calculate SSIM Value
# Paths to the images
image1_path = "/content/Results/real1.png"     # Reference clear image
image2_path = "/content/Results/dehazed1.png"  # Generated dehazed image

# Check if both files exist
if not os.path.exists(image1_path):
    raise FileNotFoundError(f"File not found: {image1_path}")
if not os.path.exists(image2_path):
```

```
    raise FileNotFoundError(f"File not found: {image2_path}")

# Load the images using OpenCV
image1 = cv2.imread(image1_path)
image2 = cv2.imread(image2_path)

# Check if images were loaded successfully
if image1 is None:
    raise ValueError(f"Failed to load image: {image1_path}")
if image2 is None:
    raise ValueError(f"Failed to load image: {image2_path}")

# Resize image1 to match image2's size, if necessary
if image1.shape != image2.shape:
    image1 = cv2.resize(image1, (image2.shape[1], image2.shape[0]))

# Convert images from BGR (OpenCV default) to RGB
image1_rgb = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
image2_rgb = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)

# Compute SSIM (for color images)
ssim_value, _ = ssim(image1_rgb, image2_rgb, full=True, channel_axis=-1)

# Print the SSIM value
print(f"SSIM: {ssim_value:.4f}")
```

8. REFLECTION OF THE TEAM MEMBERS ON THE PROJECT

Each team member contributed to different phases of the project, working collaboratively to ensure end-to-end development and deployment of the dehazing model.

Debi Prasad Mahakud – Implementation & Evaluation

- Handled model implementation and testing on final datasets.
- Integrated evaluation metrics (e.g., RMSE, MAE, PSNR, SSIM).
- Focused on optimizing output quality and managing error handling.

Sai Pritam Panda – Data Processing & Model Training

- Performed data cleaning, normalization, and preparation.
- Designed model architecture and managed training loops.
- Tuned hyperparameters and loss functions for stable convergence.

Prabhat Kumar Sharma – Research & Front-End

- Conducted literature review and comparative analysis of dehazing techniques.
- Identified gaps in existing models.
- Contributed to frontend development during integration phase.

Hrishikesh Swain – Dataset Preparation & UI

- Supervised the validation of our results
- Built the React-based frontend and integrated it with the model backend.
- Designed presentation materials for documentation and demo videos.

We learned to manage time, debug collaboratively, and maintain Git repositories efficiently. The project simulated an industry-level team workflow.

Strengths: Clear role distribution, effective planning, hands-on AI implementation with integrated frontend-backend, and innovative use of lightweight architectures for real-time performance.

Challenges: Limited GPU availability and the need to balance model complexity with real-time efficiency.