DEEP LEARNING BASED IMAGE DEBLURRING

A PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

By
Batch – A13

K. Lavanya (21JG1A0550) **M.V.S. Harini** (21JG1A0562)
D. Vineetha (21JG1A0520) **K. Akshaya** (21JG1A0548)

Under the esteemed guidance of
Mrs. V. Gowtami Annapurna
Assistant Professor
CSE Department

Department of Computer Science and Engineering

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN
[Approved by AICTE NEW DELHI, Affiliated to JNTUK Kakinada]
[Accredited by National Board of Accreditation (NBA) for B. Tech. CSE, ECE & IT – valid from 2019-22 and 2022-25]
[Accredited by National Assessment and Accreditation Council (NAAC) with A Grade-Valid from 2022-2027]
Kommadi, Madhurawada, Visakhapatnam – 530048

2021 – 2025

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project report titled **“DEEP LEARNING BASED IMAGE DEBLURRING”** is a bonafide work of following IV/IV B.Tech students in the Department of Computer Science and Engineering, Gayatri Vidya Parishad College of Engineering for Women affiliated to JNT University, Kakinada during the academic year 2024-25, in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology of this university.

K. Lavanya (21JG1A0550)	**M. V. S. Harini** (21JG1A0562)
D. Vineetha (21JG1A0520)	**K. Akshaya** (21JG1A0548)

Signature of the Guide	Signature of the HOD
Mrs. V. Gowtami Annapurna	**Dr. P. V. S. L. Jagadamba**
Asst. Professor	Professor
Dept. of CSE	Dept. of CSE

External Examiner

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people who made it possible and whose constant guidance and encouragement crown all the efforts with success.

We feel elated to extend our sincere gratitude to our guide, **Mrs. V. Gowtami Annapurna**, Assistant Professor for encouragement all the way during analysis of the project. Her annotations, insinuations and criticisms are the key behind the successful completion of the thesis and for providing us all the required facilities.

We would like to take this opportunity to extend our gratitude to Project Coordinator, **Dr. V. Lakshmana Rao,** Associate Professor of Computer Science and Engineering for making us to follow a defined path and for advising us to get better improvements in the project.

We express our deep sense of gratitude and thanks to **Dr. P. V. S. Lakshmi Jagadamba**, Professor and Head of the Department of Computer Science and Engineering for her guidance and for expressing her valuable and grateful opinions in the project for its development and for providing lab sessions and extra hours to complete the project.

We would like to take this opportunity to express our profound sense of gratitude to **Dr. R. K. Goswami**, Principal and **Dr. G. Sudheer**, Vice Principal for allowing us to utilize the college resources thereby facilitating the successful completion of our thesis.

We are also thankful to both teaching and non-teaching faculty of the Department of Computer Science and Engineering for giving valuable suggestions for our project.

K. Lavanya (21JG1A0550) **M. V. S. Harini** (21JG1A0562)

D. Vineetha (21JG1A0520) **K. Akshaya** (21JG1A0548)

TABLE OF CONTENTS

ABSTRACT

Image deblurring is an important task in computer vision, where the goal is to restore sharp images from blurry ones. The blurriness can be caused by motion, defocus, or environmental conditions. Traditional approaches, such as Convolutional Neural Networks (CNNs), are good at feature extraction but struggle with generalization and handling diverse blur levels in blind scenarios. On the other hand, Generative Adversarial Networks (GANs) are good at generating realistic textures and preserving details but may introduce artifacts like ringing and blocking. These limitations interfere with the performance of each model when applied independently. To address these challenges, this project proposes a dual-stage deep learning approach that combines both CNNs and GANs to improve image deblurring by utilizing the strengths of both. This proposed architecture uses the CNN in the first stage to extract important features and performs an initial restoration. Then, in the second stage the GAN refines the image, enhances the details and textures while ensuring that the result looks natural and realistic.

LIST OF FIGURES

LIST OF TABLES

LIST OF SCREENS

LIST OF ACRONYMS

GPU	General Processing Unit
CPU	Control Processing Unit
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
API	Application Programming Interface
UML	Unified Modelling Language
OS	Operating System
RAM	Random Access Memory
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index
ADAM	Adaptive Moment Estimation
ReLU	Rectified Linear Unit
RAM	Random Access Memory

1. INTRODUCTION

Image deblurring is an important task in computer vision, where the goal is to restore sharp images from blurry ones. Blurry images, often caused by shaky hands, motion, or focus issues, make it difficult to see important details. This happens due to shaky hands, movement, or focus issues. Traditional methods like Wiener filtering and motion blur estimation often fail to handle all types of blurs, often resulting in overly smooth or unclear images. To overcome these challenges, we use a deep learning approach combining CNNs and GANs.

In this approach, CNNs (Convolutional Neural Networks) are good at extracting important details from images, helping to identify patterns and edges lost due to blur, while GANs (Generative Adversarial Networks) focus on refining textures, ensuring that restored images look as natural and realistic as possible. By combining both CNNs and GANs, our model aims to create a more effective deblurring model that can handle different types of blurs, prevent artifacts, and generate clearer images that closely resemble the original images.

We trained our model on the GoPro dataset, which contains 3,214 image pairs - 2,103 for training and 1,111 for testing. Each image originally has a resolution of 1280×720 pixels, which is then resized to 256×256 pixels for model input. The training set is further split into 80% for training and 20% for validation, resulting in 1,682 pairs for training and 421 pairs for validation.

This project aims to develop an image deblurring model that restores images while preserving natural textures and details.

1.1 MOTIVATION

In our project, we are not just fixing blurry images but bringing back lost details and making pictures look clear and natural. Every image we improve helps capture precious moments the way they were meant to be seen, whether it is a beautiful landscape, a special memory, or just an everyday photo. Every step we take brings us closer to a world where no important detail is lost to blur. By using CNNs and GANs together, we go beyond old methods that do not work well for all types of blurs. Our model makes images look sharper and realistic. Our goal is to help technology bring out the best in every photo, making them clearer and more visually appealing. Every clear image we create is a step towards that future.

1.2 PROBLEM DEFINITION

Blurry images in photography happen due to shaky hands, movement, or focus issues, causing photos to lose important details. Traditional methods often fail to handle all types of blurs and can make images look unnatural. Deep learning techniques like CNNs and GANs offer a better way to restore sharpness while making images look realistic. However, balancing clarity and detail preservation remains a challenge. This project focuses on using CNNs for feature extraction and GANs for texture refinement to improve image quality. By testing our model on datasets like GoPro, we aim to make image restoration more effective and visually enhanced.

1.3 OBJECTIVE OF THE PROJECT

Our objective is to develop a deep learning-based approach for image deblurring to restore clarity in blurry images. By using Convolutional Neural Networks (CNNs) for feature extraction and Generative Adversarial Networks (GANs) for texture refinement, we aim to enhance image sharpness while preserving natural details. Using datasets like GoPro, which contain paired blurry and sharp images, we aim to train and optimize our model for improved image restoration. Through this project, we aim to overcome the limitations of traditional deblurring methods, reduce artifacts, and produce high-quality, visually appealing images. Ultimately, our objective is to provide a more efficient and reliable solution for enhancing blurred photographs, making them clearer and more realistic.

1.4 LIMITATIONS OF THE PROJECT

Variations in blur levels and image quality can impact how well our project clears up blurry images. Deep learning models need a lot of computing power, making real-time processing difficult. The model may not always fully fix very blurry images and could leave small errors. Its performance depends on the dataset it was trained on, so results may vary based on the camera and lighting conditions.

1.5 ORGANIZATION OF DOCUMENTATION

Course overview of rest of the documentation work is explained below:

Chapter 1: Introduction describes the motivation and objective of the project.

Chapter 2: Literature survey describes the primary terms involved in the development of the project.

Chapter 3: Analysis deals with the detailed analysis of the project. Software Requirement Specification further contains software requirements, hardware requirements. Also, this chapter explains the algorithms used in the project.

Chapter 4: Contains design part and UML diagrams.

Chapter 5: Contains implementation of the project with relevant codes, screenshots of outputs.

Chapter 6: Contains diverse test case scenarios and validation testing implemented in the project.

Chapter 7: Contains conclusion of the project.

Chapter 8: Contains references.

2. LITERATURE SURVEY

2.1 INTRODUCTION

Yan Q et al. [1] introduced SharpFormer, a Transformer-based model designed to improve image deblurring by preserving both local and global details. Traditional CNN-based methods often fail to capture complex blur variations due to their fixed filter structure. To address this, the authors proposed a Locality Preserving Transformer (LTransformer) that enhances fine details while maintaining long-range dependencies. Additionally, they introduced a Hybrid Block, combining CNN and Transformer elements to balance global context and local feature extraction. They also incorporated a Dynamic Convolution (DyConv) Block to adaptively handle different blur patterns. The model was tested on the GoPro and REDS datasets, showing better results than existing deblurring techniques. The study highlights the importance of integrating Transformers with CNNs to achieve better deblurring results while maintaining fine details and realistic textures.

Yae and Ikehara [2] proposed IRFTNet, a lightweight model for single-image deblurring based on the UNet architecture. It features the IRFTblock for efficient feature extraction, the LFS module for better information transfer, and a multi-output structure. On the GoPro dataset, IRFTNet achieved a PSNR of 32.98 dB while using about half the FLOPs of DeepRFT. The study emphasizes reducing computational complexity while maintaining high image quality.

Lee and Cho [3] proposed a Locally Adaptive Channel Attention-Based Spatial-Spectral Neural Network (LACA-SSN) for single-image deblurring. This model uses two specialized components: a spatial restorer to correct image intensity and a spectral restorer to refine fine details by analyzing both magnitude and phase of frequency components. Unlike previous methods that only use magnitude information, LACA-SSN leverages both, enhancing the model’s ability to capture intricate textures and object boundaries. The model was evaluated on datasets like GoPro and HIDE, where it outperformed other state-of-the-art methods in terms of PSNR and SSIM. The study emphasizes the benefits of combining spatial and spectral information and suggests future improvements through integrating semantic analysis and motion estimation for better results.

Y. Quan et al. [4] proposed a deep-learning-based approach for nonblind image deblurring using a Complex-Valued Convolutional Neural Network (CV-CNN) in the Gabor domain. This method enhances deblurring by leveraging the Gabor transform’s ability to capture spatial and frequency information and the richer representation of complex numbers. The model uses an unrolled optimization framework with two key processes: inversion and denoising, where the CV-CNN handles noise from unknown distributions effectively. The approach was tested on datasets with different noise levels and outperformed state-of-the-art methods in PSNR and SSIM. The study highlights the benefits of combining Gabor representation and CV-CNNs, suggesting future work on extending these techniques to other image restoration tasks.

Q. Zhao et al. [5] introduced a CNN-Transformer Multiscale Hybrid Architecture (CTMS) for image deblurring. This model addresses the limitations of traditional CNN-based methods by combining convolutional neural networks (CNNs) with transformers. CTMS uses a coarse-to-fine strategy where CNN subnetworks handle coarser-scale blurs, while a transformer processes finer-scale images to capture long-range dependencies. The model also includes a feature modulation network to adapt to varying blur patterns. On the GoPro dataset, CTMS achieved a PSNR of 32.73 dB and an SSIM of 0.959, outperforming other deblurring models. The study emphasizes the effectiveness of combining CNN and transformer approaches for handling diverse blur patterns and suggests further research into improving computational efficiency and tackling mixed-blur scenarios.

A. Esmaeilzehi et al. [6] compared various approaches to image deblurring, focusing on deep-learning-based methods. They discussed model-based techniques like sparsity prior, nonlocal self-similarity, and learned natural image priors. The study highlights how convolutional neural networks (CNNs) are used for nonblind image deblurring by treating the deconvolution process as a denoising task. Methods like regularized inverse filtering combined with deep learning models have improved deblurring performance. The paper suggests further research on enhancing blur kernel estimation and leveraging advanced CNN architectures to improve image clarity

Y. Wu et al. [7] proposed a Two-Level Wavelet-Based Convolutional Neural Network (CNN) for image deblurring, combining Discrete Wavelet Transform (DWT) with deep learning. Traditional deblurring methods often struggle to balance texture preservation and model efficiency. To address this, the authors introduced a two-level wavelet transform that separates image context from texture information, reducing computational complexity while enhancing image details. They also modified the Inception module by adding a pixel-wise attention (PA) mechanism and a channel scaling factor to improve feature extraction while keeping the model lightweight. The proposed method was tested on real-world and synthetic datasets, including GoPro demonstrating results that match or surpass those of existing deblurring models. The study highlights the advantages of combining wavelet transforms with CNNs to achieve better deblurring results while keeping the model efficient and lightweight.

Wu et al. [8] proposed a deep-learning method using a Multi-Level Wavelet Convolutional Neural Network (MWCNN) to restore images degraded by Cauchy noise and blur. The model uses a forward-backward splitting (FBS) optimization technique, allowing efficient image restoration without additional variables. By applying a multi-noise-level training strategy, the model handles various noise intensities. On the Set14 dataset, their method outperformed conventional techniques like total variation and low-rank priors in both PSNR and SSIM. The study emphasizes the model's ability to preserve texture details and suggests further improvements through advanced denoising strategies and optimization methods.

Albluwi et al. [9] proposed a deep-learning model called DBSRCNN for image deblurring and super-resolution. This model extends the SRCNN architecture by adding additional convolutional layers and a feature concatenation layer to enhance image reconstruction. The model was tested on the Set5 and Set14 datasets under both blind and non-blind scenarios with different levels of Gaussian blur. Experimental results showed that DBSRCNN outperformed SRCNN in terms of PSNR and SSIM across all blur levels. The study highlights the effectiveness of combining low-level and enhanced features for improved image restoration and suggests further exploration of deeper architectures for better performance.

C. Min et al. [10] proposed a blind image deblurring method using a Recursive Deep Convolutional Neural Network (R-DbCNN) combined with wavelet transform. The wavelet transform extracts both low and high-frequency components from blurred images, while R-DbCNN removes artifacts and reduces redundancy. The model uses parameter pre-selection and sharing to improve training efficiency. Experiments on different blur types, including Gaussian and motion blur, showed that the proposed method achieved higher PSNR, lower MSE, and better SSIM compared to other techniques like BM3D and DnCNN. The study suggests further optimization of R-DbCNN and exploring advanced wavelet techniques for better deblurring performance

Y. Song et al. [11] proposed a novel blind image deblurring method using a recursive deep CNN improved by wavelet transform. They first decompose the blurred image into low- and high-frequency components using wavelet transform. Then, a deep recursive convolutional neural network (R-DbCNN) is used to reduce data redundancy and remove blur. Their approach outperforms traditional CNN-based methods in both deblurring quality and training speed. The authors demonstrate the effectiveness of their method on various types of blurs, including motion, Gaussian, and out-of-focus blur.

Zhang et al. [12] proposed a spatially variant recurrent neural network (RNN) for dynamic scene deblurring. Their model consists of three main parts: a feature extraction network, an RNN for deblurring, and an image reconstruction network. The RNN is designed to handle spatially varying blur by learning pixel-wise weights using a deep convolutional neural network (CNN). This approach allows the model to capture a large receptive field while maintaining a small model size. The method was evaluated on benchmark datasets and outperformed existing algorithms in terms of PSNR, SSIM, speed, and model size. The study suggests further exploration of efficient spatially variant models for complex motion blur scenarios.

Kupyn et al. [13] introduced a GAN-based approach for blind motion deblurring of a single image. The model uses a conditional generative adversarial network (cGAN) with a ResNet-based generator and a PatchGAN-style critic trained using Wasserstein loss with gradient penalty. It combines adversarial loss with perceptual loss computed on VGG-19 feature maps to produce visually realistic and structurally accurate outputs. Unlike traditional methods relying on pixel-wise losses, DeblurGAN focuses on high-level feature reconstruction for sharper results. The generator learns to estimate the residual between blurred and sharp images, which improves training efficiency and generalization. DeblurGAN is significantly faster than previous deep learning models while maintaining competitive performance on PSNR and SSIM metrics. It was evaluated on standard datasets like GoPro and Kohler, showing superior visual quality and strong generalization. The authors also proposed a novel data synthesis technique for training, using random motion trajectories to generate blurred images.

Nah et al. [14] proposed a deep multi-scale convolutional neural network (CNN) for dynamic scene deblurring. Their model uses a coarse-to-fine approach, where blurry images are progressively restored across multiple scales. Unlike conventional methods that estimate blur kernels, their network directly predicts sharp images, reducing artifacts caused by inaccurate kernel estimation. They trained the model on a newly created large-scale dataset using high-speed video frames to generate realistic blur. The method outperformed existing techniques in both PSNR and SSIM on datasets like GoPro, highlighting the effectiveness of their kernel-free approach for complex motion blur scenarios.

Zhang et al. [15] proposed a deep CNN denoiser prior to enhance image restoration tasks such as deblurring and super-resolution. They integrated fast convolutional neural network (CNN) denoisers into a model-based optimization framework using the half-quadratic splitting (HQS) method. The CNN denoisers, trained on large datasets, efficiently handle Gaussian noise while preserving fine image details. Experimental results showed that their approach achieved higher PSNR and SSIM scores compared to traditional methods like BM3D and WNNM across various blur and noise levels. The study emphasizes the advantages of combining fast CNN denoisers with model-based optimization and suggests further exploration of multi-task restoration frameworks for better performance.

The Table 1 shows the comparative Study of various related works discussed above at a single glance.

Table 1. A Comparative Study of Various Related Works

S. No	**Title & Authors**	**Published Journal and Year**	**Dataset Used**	**Approach**	**Outcome**	**Limitations**
1.	SharpFormer: Learning Local Feature Preserving Global Representations for Image Deblurring. Yan Q, Gong D, Zhang Y, Wang P, Zhen Zhang, and Javen Q. Shi	IEEE Transactions on Image Processing, 2023	GoPro	SharpFormer is a Transformer-based model, integrates global and local features using an LTransformer Block. A Hybrid Block combines LTransformer and DyConv.	SharpFormer achieves 33.10 dB PSNR and 0.96 SSIM on GoPro, delivering sharper images with fewer artifacts.	SharpFormer's main limitation is its high computational cost, as Transformer-based models require more resources than CNNs.
2.	Inverted Residual Fourier Transformation for Lightweight Single Image Deblurring. Shunsuke Yae, Masaaki Ikehara	IEEE Access, 2023	GoPro, HIDE, RealBlur	Proposed a lightweight network (IRFTNet) based on UNet with three components: Inverted Residual Fourier Transformation block (IRFTblock), Lower Feature Synthesis (LFS), and multiple outputs structure.	Achieved 32.98 dB PSNR on GoPro dataset with half the FLOPs and 60% of the parameters compared to DeepRFT.	Limited to motion blur images. Further research needed for handling complex real-world blurs and exploring more lightweight designs with improved accuracy.

S. No	**Title & Authors**	**Published Journal and Year**	**Dataset Used**	**Approach**	**Outcome**	**Limitations**
3.	Locally Adaptive Channel Attention-Based Spatial–Spectral Neural Network for Image Deblurring. Ho Sub Lee, Sung In Cho	IEEE Transactions on Circuits and Systems for Video Technology, 2023	GoPro, HIDE	Proposed a Locally Adaptive Channel Attention-Based Spatial–Spectral Network (LACA-SSN) using two restorers spatial and spectral to address image deblurring.	Achieved superior by improving PSNR by 1.0209 dB on GoPro and by 1.7371 dB on HIDE datasets.	Struggles with handling large motion and severely distorted information, like human faces.
4.	Nonblind Image Deblurring via Deep Learning in Complex Field**.** Yuhui Quan, Peikang Lin, Yong Xu, Yuesong Nan, Hui Ji	IEEE Transactions on Neural Networks and Learning Systems, 2022	BSDS500 dataset**,** Sun et al.’s dataset, Levin et al.’s dataset	Introduced a Gabor-domain complex-valued CNN (CV-CNN) for image deblurring using unrolled optimization. It uses Gabor-domain priors for inversion and a CV-CNN for denoising.	Achieved superior performance in terms of PSNR and SSIM compared to state-of-the-art methods. Better generalization to unseen noise data.	Struggles with certain fine image details and residual blurring in complex textures.
5.	Rethinking Image Deblurring via CNN-Transformer Multiscale Hybrid Architecture**.** Qian Zhao, Hao Yang, Dongming Zhou, Jinde Cao	IEEE Transactions on Instrumentation and Measurement, 2022	GoPro, HIDE, RealBlur	Proposed a hybrid architecture (CTMS) combining CNN and transformer. It uses CNN subnetworks with varying receptive fields and an efficient transformer block for multiscale deblurring.	Achieved 32.73 dB PSNR and 0.959 SSIM on GoPro dataset. across multiple datasets.	Limited to motion blur. Struggles with blurs caused by low resolution, noise, or out-of-focus effects due to training data limitations.

S. No	Title & Authors	Published Journal and Year	Dataset Used	Approach	Outcome	Limitations
6.	UPDResNN: A Deep Light-Weight Image Upsampling and Deblurring Residual Neural Network**.** Alireza Esmaeilzehi, M. Omair Ahmad, M. N. S. Swamy	IEEE Transactions on Broadcasting, 2021	BSD 200 (training), BSD 100 (testing)	Proposed a three-stage light-weight CNN called UPDResNN for joint image upsampling and deblurring using multi-scale residual blocks and dual loss functions.	Achieved superior PSNR and SSIM on multiple benchmark datasets while maintaining low computational complexity and parameter count.	Limited to handling blurring and downsampling together; struggles with more complex degradation scenarios.
7.	Two-Level Wavelet-Based Convolutional Neural Network for Image Deblurring**.** Yeyun Wu, Pan Qian, Xiaofeng Zhang	IEEE Access, 2021	GoPro, Köhler	Proposed a two-level wavelet-based CNN that combines discrete wavelet transform (DWT) with a modified Inception module using pixel-wise attention (PA).	Achieved superior PSNR and SSIM on both real-world (GoPro) and synthetic (Köhler) datasets while reducing model complexity.	Struggles with very complex motion blur and might produce artifacts. Limited generalization to other types of image degradation.

S. No	Title & Authors	Published Journal and Year	Dataset Used	Approach	Outcome	Limitations
8.	Deep Multi-Level Wavelet-CNN Denoiser Prior for Restoring Blurred Image with Cauchy Noise**.** Tingting Wu, Wei Li, Shilong Jia, Yiqiu Dong, Tieyong Zeng	IEEE Signal Processing Letters, 2020	Set14	Proposed a deep-learning-based image denoiser using a Multi-Level Wavelet CNN (MWCNN) trained to remove Cauchy noise and blur. Utilized Forward-Backward Splitting (FBS) for efficient optimization.	Improved image restoration performance under Cauchy noise with better PSNR and SSIM values compared to traditional methods.	Limited to Cauchy noise; struggles with other complex noise types and may require different models for other noise distributions.
9.	Image Deblurring and Super-Resolution Using Deep Convolutional Neural Networks**.** Fatma Albluwi, Vladimir A. Krylov, Rozenn Dahyot	IEEE International Workshop on Machine Learning for Signal Processing (MLSP), 2018	Set5,91-image dataset	Proposed a De-blurring Super-Resolution Convolutional Neural Network (DBSRCNN) that extends SRCNN to handle both deblurring and super-resolution.	DBSRCNN outperforms SRCNN in both blind and non-blind scenarios across various blur levels, achieving higher PSNR and SSIM.	Limited to Gaussian blur; performance decreases with increasing blur levels and may not generalize to other types of blurs.

S. No	Title & Authors	Published Journal and Year	Dataset Used	Approach	Outcome	Limitations
10.	Blind Deblurring via a Novel Recursive Deep CNN Improved by Wavelet Transform. Chao Min, G Wen, B Li, F Fan	IEEE Access, 2018	Custom dataset	Proposed a two-step method combining wavelet transform and a Recursive Deep Convolutional Neural Network (R-DbCNN) to remove various blurs. Utilizes wavelet transform and a recursive CNN for artifact removal and image reconstruction.	Outperformed other methods (e.g., BM3D, DnCNN) in PSNR, SSIM, and MSE across multiple blur types. Achieved better deblurring.	Limited to specific types of blurs (Gaussian, motion, and similar artifacts).
11.	Single Image Dehazing Using Ranking-CNN Y. Song et al.	IEEE Transactions on Multimedia, 2018	Dataset-Syn and Dataset-Cap	Proposed a novel Ranking-CNN model to estimate scene transmission for image dehazing.	Achieved superior performance in both objective and subjective evaluations compared to other methods.	Struggles with images containing extremely dense haze and may require large training datasets for accuracy.

S. No	Title & Authors	Published Journal and Year	Dataset Used	Approach	Outcome	Limitations
12.	Dynamic Scene Deblurring Using Spatially Variant Recurrent Neural Networks. Jiawei Zhang, Jinshan Pan, Jimmy Ren, Yibing Song, Linchao Bao, Rynson W.H. Lau, Ming-Hsuan Yang	IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018	Dynamic Scene Dataset	Proposed a spatially variant recurrent neural network (RNN) with pixel-wise weights learned by a deep CNN to model dynamic scene blur.	Achieved higher PSNR (29.19 dB) and SSIM (0.9306) compared to other methods while reducing model size (37.1 MB) and computation time.	Struggles with extremely large blur variations and may not generalize well to unseen complex motion patterns.
13.	DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, Jiri Matas	IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018	GoPro, Kohler, and custom synthetic blur dataset	DeblurGAN uses a ResNet-based generator and PatchGAN-style critic. It combines Wasserstein GAN with gradient penalty and perceptual loss (VGG features). It also introduces a synthetic blur generation method using random camera trajectories.	Achieved 28.7dB PSNR and 0.958 SSIM on GoPro. Demonstrated competitive performance on Kohler dataset and improved real-world object detection performance. Runs 5x faster than DeepDeblur with fewer parameters.	Slightly lower PSNR than some CNN methods (e.g., Nah et al.). Performance heavily depends on dataset and may not align well with pixel-wise metrics.

S. No	Title & Authors	Published Journal and Year	Dataset Used	Approach	Outcome	Limitations
14.	Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring. Seungjun Nah, Tae Hyun Kim, Kyoung Mu Lee	IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017	GoPro, Köhler, Lai dataset	Proposed a deep multi-scale CNN model with residual blocks to restore sharp images. Introduced a new realistic GoPro dataset for training.	Achieved state-of-the-art performance with 29.08 dB PSNR on the GoPro dataset. Faster deblurring (3.09 seconds per image).	Limited to motion blur scenarios. May not generalize well to other types of blurs (e.g., defocus or atmospheric blur).
15.	Learning Deep CNN Denoiser Prior for Image Restoration. Kai Zhang, Wangmeng Zuo, Shuhang Gu, Lei Zhang	IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017	BSD68, Set5, Set14	Proposed a CNN-based denoiser integrated into the Half Quadratic Splitting (HQS) optimization framework for various image restoration tasks like denoising, deblurring, and super-resolution.	Achieved superior PSNR performance on image denoising (29.15 dB for noise level 25) and competitive results for image deblurring and super-resolution.	Requires training different denoisers for varying noise levels, and performance depends on the quality of the training dataset.

2.2 EXISTING SYSTEM

Based on our literature survey, advanced image restoration techniques like SharpFormer, Hybrid Blocks, and Dynamic Convolution (DyConv) significantly improve deblurring performance. SharpFormer utilizes transformer architecture to capture long-range dependencies, extracting global features while preserving local details. Its self-attention mechanism restores scene structure even under severe motion blur, maintaining fine textures and edges. Hybrid Blocks combine CNNs and transformers, where the transformer branch captures large-scale dependencies, and the CNN branch preserves fine details. This fusion enhances both structural coherence and texture clarity in deblurred images. DyConv dynamically adjusts filters to manage non-uniform blur, using multiple parallel kernels with an attention mechanism to emphasize key features. These techniques enable precise deblurring, resulting in sharp, detailed, and visually consistent images, which are crucial for accurate image analysis in complex scenarios. Refer Figure 1 for Existing system.

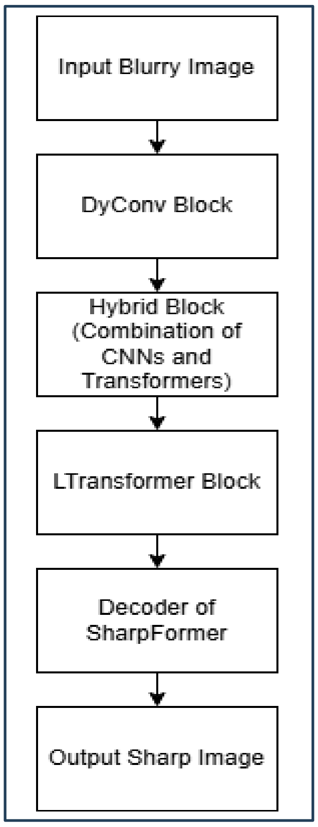


Figure 1. Existing System

2.3 DISADVANTAGES OF EXISTING SYSTEM

- The existing system focuses mainly on motion blur and struggles to handle other types of image distortions like noise, defocus blur, or low-light conditions.
- The model's performance heavily depends on large and diverse training datasets, making it less effective when dealing with unseen or unusual blur patterns.
- The system requires significant computational power and memory, making it challenging to use on devices with limited resources.
- It struggles to accurately deblur images with highly irregular or complex motion patterns, reducing its effectiveness in such scenarios.
- The process of converting images into patches for the LTransformer causes information loss, especially when handling medium-resolution images

2.4 PROPOSED SYSTEM

In the proposed system, we introduced a deep learning approach for advanced image deblurring by utilizing a Generative Adversarial Network (GAN) architecture. The detailed proposed system is shown in Figure 2.

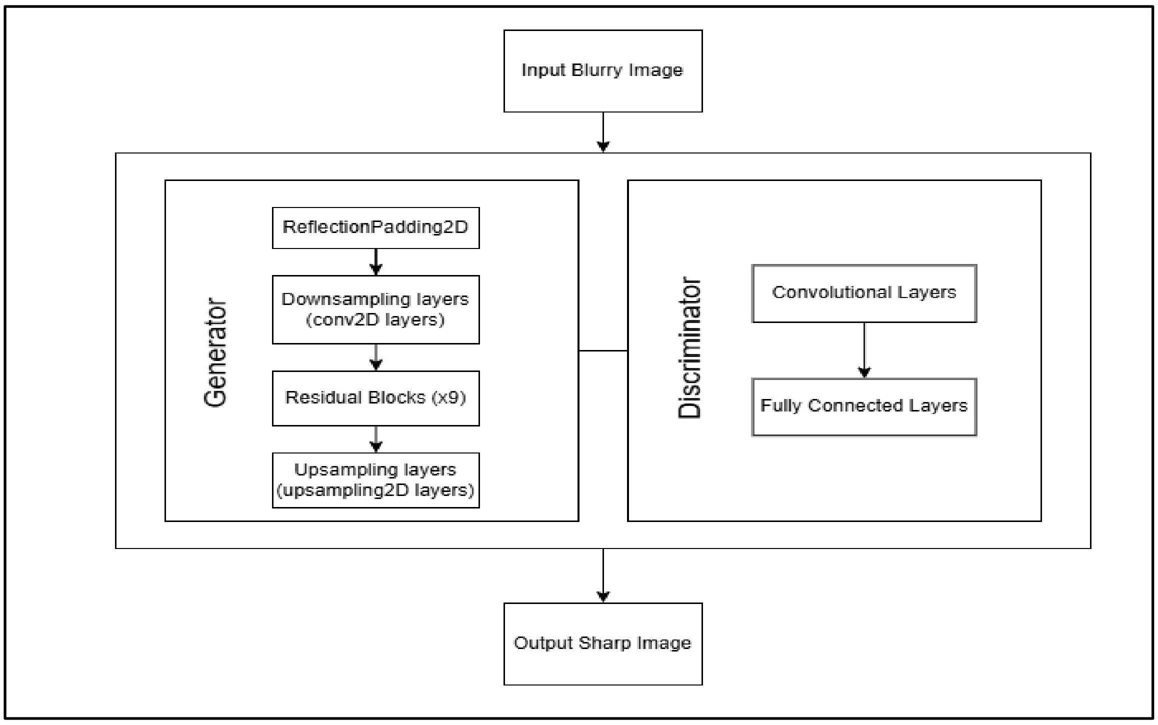


Figure 2. Proposed System

Our model is based on a Generative Adversarial Network (GAN) architecture, which consists of a generator and a discriminator. The generator utilizes a ResNet-based convolutional neural network (CNN) to extract features and restore blurry images. Deep residual connections are employed to preserve important image details while learning deblurring filters. To further enhance output quality, reflection padding is applied to preserve image boundaries and minimize artifacts. Downsampling layers are used to extract features at different scales improves the sharpness and texture clarity of the restored images. Upsampling layers increase the spatial resolution to generate a high-resolution and sharp output.

Discriminator uses a PatchGAN-based architecture to refine details and ensure realistic outputs. The PatchGAN-based discriminator focuses on local image patches rather than the entire image, enabling better texture preservation and enhanced fine details by evaluating small sections of the image to determine their sharpness. By providing adversarial feedback, the discriminator helps the generator produce sharper and more realistic images. This approach, , combining the generator's global feature extraction and upsampling with the discriminator's detailed local analysis, aims to enhance image clarity, resulting in visually consistent and high-quality deblurred images.

2.5 CONCLUSION

In this chapter, we explored existing image deblurring techniques to design a more effective solution using a GAN-based architecture. Our system combines a ResNet-based generator for feature extraction and image restoration with a PatchGAN-based discriminator to refine fine details and improve image quality. The generator uses ReflectionPadding2D, followed by Downsampling layers and Residual Blocks to effectively learn deblurring filters while preserving image details. Upsampling layers are used to produce the final sharp image. The discriminator focuses on local image patches and guides the generator towards more realistic and sharper outputs. This combined approach enhances deblurring performance, offering a promising solution for applications requiring clear and high-resolution images.

3. REQUIREMENT ANALYSIS

3.1 INTRODUCTION

The main focus of this project is to develop an image deblurring system that restores blurry images into clear and sharp ones. Blurry images, often caused by shaky hands, motion, or focus issues, make it difficult to see important details. Traditional methods like Wiener filtering struggle to handle all types of blurs, often resulting in overly smooth or unclear images. To overcome these challenges, we use a deep learning approach combining CNNs and GANs. CNNs extract lost details and edges, while GANs refine textures to produce more realistic images. We trained our model on the GoPro dataset, which contains 3,214 image pairs - 2,103 for training and 1,111 for testing. Each image originally had a resolution of 1280×720 pixels, which is then resized to 256×256 pixels for model input. The training set is further split into 80% for training and 20% for validation, resulting in 1,682 pairs for training and 421 pairs for validation. This project aims to create a robust deblurring system that delivers sharp images while preserving natural details and textures.

3.2 SOFTWARE REQUIREMENT SPECIFICATION

3.2.1 User Requirements

User requirements refer to what end-user's need from a system in terms of features and usability. These include both functional needs and non-functional expectations (such as being fast, easy to use, and ensuring data safety).

Functional Requirements

Table 2. Functional Requirements

S. No.	**Requirement**	**Description**
1.	Image Upload	Users should be able to upload a blurred image via a drag-and-drop area or file picker.

S. No.	**Requirement**	**Description**
2.	Deblurring Output	Users should receive a deblurred image after uploading a valid blurred image.
3.	File format validation	Only .png and .jpeg image formats should be accepted. Invalid formats must trigger an error message.
4.	File size limitation	Image files should be limited to a maximum of 50MB. Larger files should be rejected with a warning
5.	Image validation	System should detect blank images and display appropriate warnings.
6.	Image Comparision	Users should be able to view original and deblurred images side by side for comparison.

Non-Functional Requirements

Table 3. Non-Functional Requirements

S. No.	**Requirement**	**Description**
1.	Usability	Interface should be modern, intuitive, and easy to use for non-technical users.
2.	Browser Compatibility	The application should work on all modern browsers (Chrome, Firefox, Edge, Safari).
3.	Error Handling	User-friendly error messages should be displayed for invalid input, system errors, or failed processing.
4.	Performance	The deblurring process should complete within a few seconds (depends on image size and system speed).

3.2.2 Software Requirements

The software requirements include the languages, libraries, packages, operating system and different tools used for developing the project. The software requirements are shown in Table 4. We used python for coding and the required Python packages are mentioned in Table 5.

Table 4. Software Requirements

S. No.	Software	Version
1.	Google Colab	3.10
2.	Visual Studio Code Editor	1.99.0
3.	Operating System	64-bit operating system, x64-based processor
4.	Python	3.11.11
5.	Anaconda3	24.11.3
6.	Jupyter Notebook	7.3.2
7.	Node.js	22.14.0
8.	TypeScript	11.2.0

Python Packages

Python packages we used in our project are included in Table 5. We used the command **pip install** "**package_name==version**" to download the packages.

Table 5. Python Packages

S. No.	Packages Used	Description
1.	tensorflow (2.18.0)	It is a python-based package which provides implementing different machine learning and AI tools.
2.	scikit-learn (1.6.1)	Scikit-learn is an opensource python package functionality supporting supervised and unsupervised learning.

S. No.	Packages Used	Description
3.	matplotlib (3.10.0)	It is a cross-platform, data visualization and graphical plotting library for python and its numerical extension
4.	numpy (2.0.2)	Numpy package is fast and accurate, used for scientific computing in python
5.	tqdm (4.67.1)	Tqdm is a progress bar library that displays a visually appealing representation of the progress of long-running operations, making it easier to gauge how long tasks might take.
6.	os	os provides functions to interact with the operating system, such as file and directory manipulation.
7.	time	It offers time-related functions like delays, timestamps, and performance measurement.
8.	glob	Glob finds all file paths matching a specified pattern using wildcard matching.
9.	shutil	It enables high-level file operations like copying, moving, and deleting files or directories.
10.	hashlib	It provides hashing algorithms (like SHA-256) for creating secure hash digests of data.
11.	cv2 (OpenCV)	It is a powerful open-source computer vision library used for image and video processing, including tasks like filtering, edge detection, face recognition, and object tracking.

Node Packages (Frontend)

Node.js packages used in our project are listed in Table 6. We installed them using the command **"npm install package_name@version".** These packages support the frontend development of our application, including UI design, animation, file upload functionality, and optimized image rendering.

Table 6. Node Packages

S. No.	Packages Used	Description
1.	Npm (11.2.0)	Node Package Manager is the default package manager for Node.js. It is used to install, manage, and share JavaScript and TypeScript libraries and tools
2.	Next.js (14.1.3)	React-based web framework for SSR, routing, and full-stack development.

S. No.	Packages Used	Description
3.	Tailwind CSS (3.4.1)	Utility-first CSS framework for fast and responsive UI design.
4.	react-dropzone (14.3.8)	It is a File uploader library for drag-and-drop functionality in React apps.
5.	lucide-react (0.350.0)	It is a customizable icon library for React.
6.	next/image	It is a Next.js built-in component used for optimized image rendering and performance
7.	Framer Motion (12.5.0)	A React animation library for declarative motion and transitions.
8.	React (18.2.0)	JavaScript library for building user interfaces.

3.2.3 Hardware Requirements

The below Table 7 shows the minimum hardware requirements of our project.

Table 7. Minimum Hardware Requirements

S. No	Hardware Requirements	Configuration
1.	Processor	Intel Core - i5
2.	GPU	Tesla T4 / P100
3.	RAM	12GB
4.	Storage	15GB

3.3 CONTENT DIAGRAM OF PROJECT

We divided the image deblurring process into several modules, as shown in Figure 3. First, we use the GoPro dataset, which contains 3,214 blurry and sharp image pairs (2,103 for training and 1,111 for testing) with a 1,280×720 resolution to ensure real-world performance. We preprocess the dataset by resizing, normalizing and augmenting which is then passed to our model for training. The CNN extracts important features and patterns from blurry images, while the GAN refines textures for realistic outputs. This combined approach handles various types of blurs, reduces distortions, and produces clear images. The working of each module is explained in detail in the following sections.

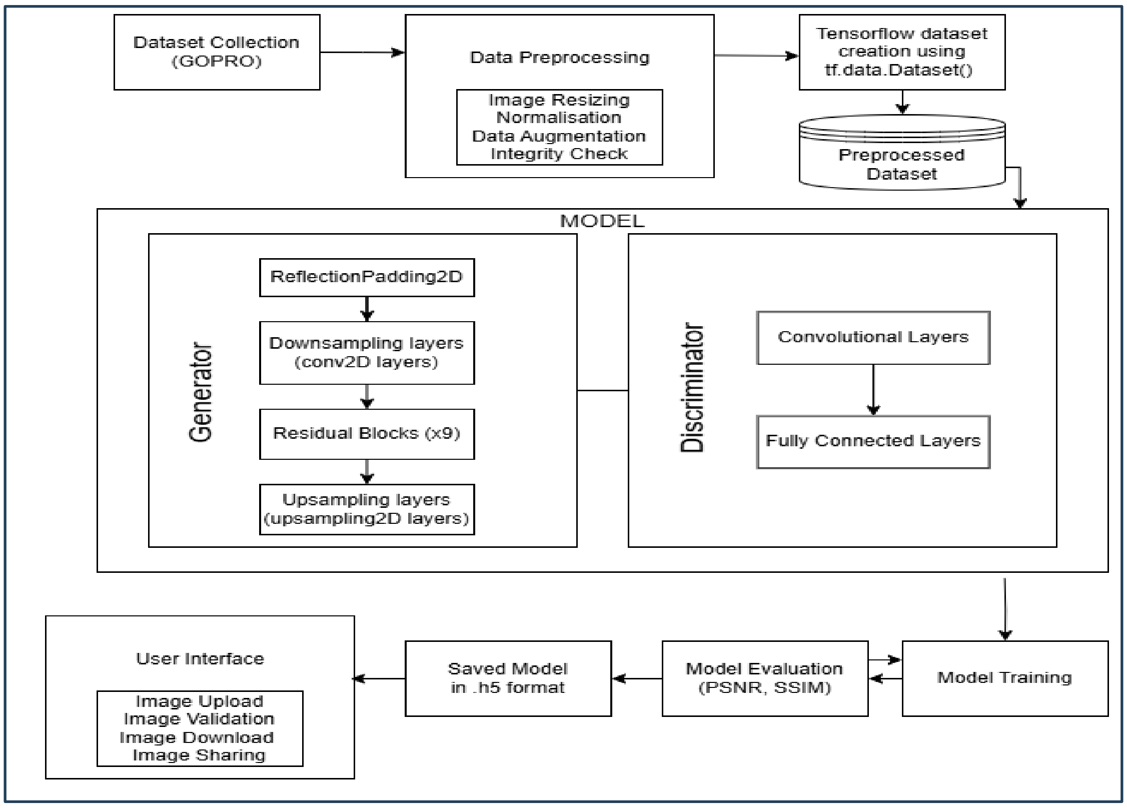


Figure 3. Content Diagram

The modules identified are discussed below:

Dataset Collection: The process begins with collecting the GOPRO dataset, which contains high-quality images used for training and testing the model.

Data Preprocessing: The images are resized, normalized, and augmented to improve model performance. An integrity check ensures all data is clean and usable.

TensorFlow Dataset Preparation: Includes convertion of the organized GoPro dataset into a tf.data.Dataset for efficient loading, shuffling, and batching. The dataset is stored in TensorFlow format, including .shard and .snapshot directories along with metadata files like dataset_spec.pb and snapshot.metadata.

Model Definition: Our model follows GAN architecture, consisting of a ResNet-based Generator that restores blurred images and a PatchGAN Discriminator that helps the generator refine its output by distinguishing between real and generated images by analysing local image patches and outputting a real/fake score.

Model Training: The Generator and Discriminator are trained together in an adversarial setup, constantly improving through competition.

Model Evaluation: The model's output is evaluated using PSNR and SSIM metrics to check how closely it resembles the original high-quality image.

User Interface: The final model is deployed through a simple interface where users can upload blurry images and download the enhanced versions.

3.4 FLOWCHARTS AND ALGORITHMS

The Figure 4 describes the flow chart of our project, starting from data collection, followed by preprocessing of dataset. Later the dataset is used to train the model.

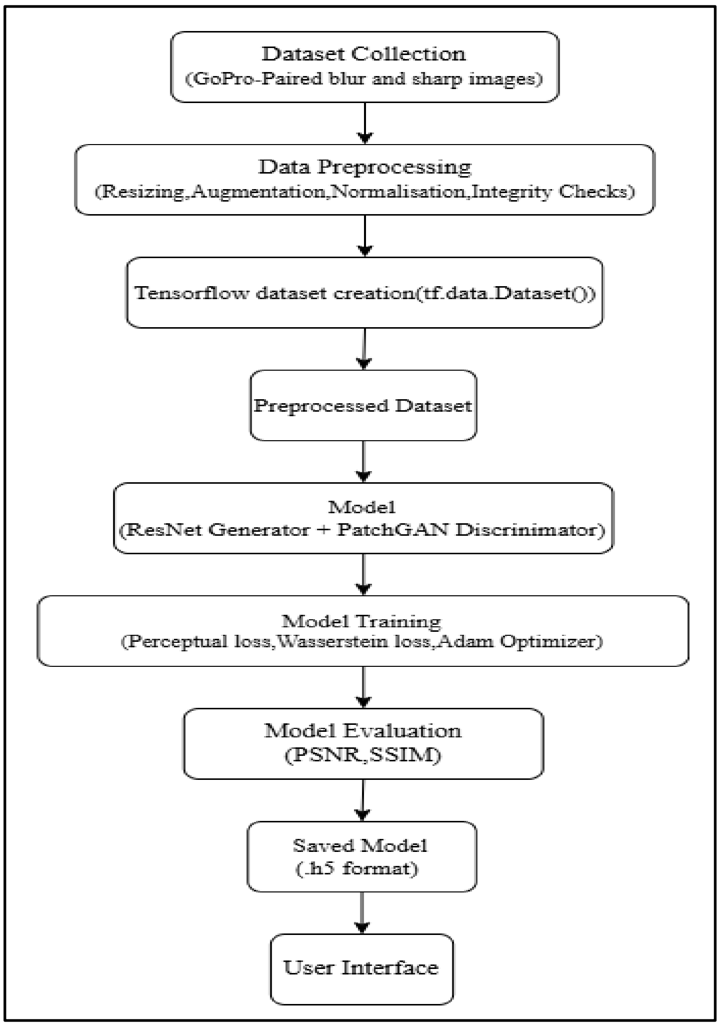

Dataset Collection
(GoPro-Paired blur and sharp images)
Data Preprocessing
(Resizing,Augmentation,Normalisation,Integrity Checks)
Tensorflow dataset creation(tf.data.Dataset())
Preprocessed Dataset
Model
(ResNet Generator + PatchGAN Discrinimator)
Model Training
(Perceptual loss,Wasserstein loss,Adam Optimizer)
Model Evaluation
(PSNR,SSIM)
Saved Model
(.h5 format)
User Interface

Figure 4. Flowchart of the Project

3.4.1 ResNet Generator and PatchGAN Discriminator

The architecture is based on a generative adversarial framework designed for high-quality image restoration. It consists of two key components: the Generator, responsible for producing enhanced images, and the Discriminator, which evaluates the authenticity of the generated output to guide the Generator during training. The figure 5 shows the architecture of the model.

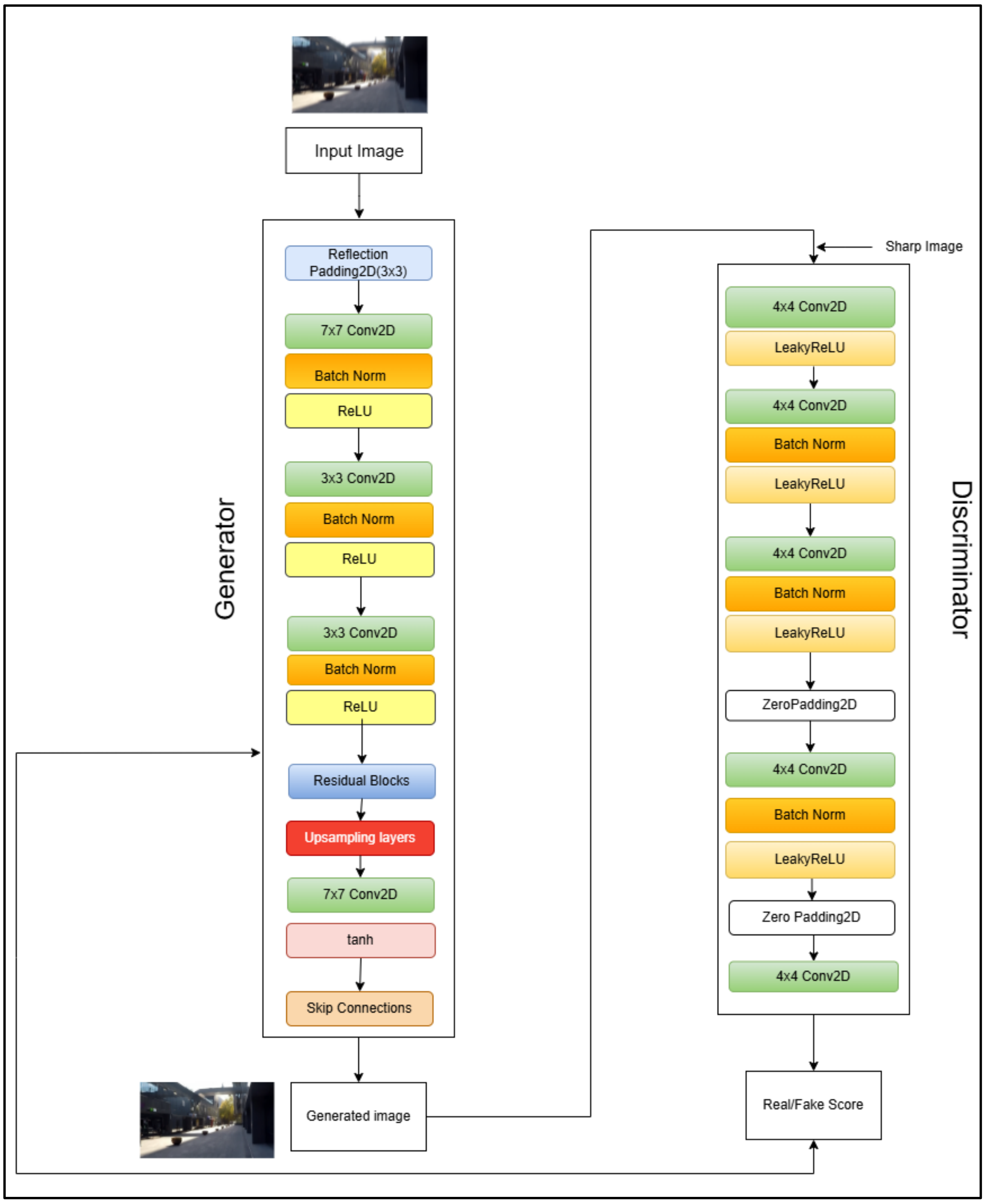


Figure 5. Architecture of ResNet Generator and PatchGAN Discriminator

ResNet Generator

The ResNet Generator is designed to transform low-quality or degraded input images into high-quality, photo-realistic outputs. The network begins with Reflection Padding, which helps maintain edge information during convolution operations. This is followed by a sequence of convolutional layers (with kernel sizes of 7×7 and 3×3), each coupled with Batch Normalization and ReLU activations to extract rich hierarchical features from the input image.

At the core of the Generator are several Residual Blocks, which introduce skip connections that allow the network to learn modifications or "residuals" relative to the input. These blocks help in preserving spatial information and improving gradient flow, making training more stable and effective for deep architectures.

The feature maps are then passed through Upsampling Layers, which increase the spatial resolution of the features to match the original image size. The final convolutional layer (7×7) followed by a tanh activation maps the features back into the image domain. A skip connection is also used at the end to merge input features with the generated output, enhancing detail and consistency in the restored image.

PatchGAN Discriminator

The PatchGAN Discriminator focuses on evaluating the realism of small image patches instead of the entire image, encouraging the Generator to produce images with high-frequency details and fine textures.

The Discriminator is composed of stacked 4×4 convolutional layers with LeakyReLU activations and Batch Normalization, which progressively reduce spatial dimensions while increasing feature complexity. Zero Padding is strategically applied to preserve feature alignment across layers.

Rather than outputting a single real/fake prediction, the Discriminator produces a grid of scores, each representing the authenticity of a corresponding image patch. This patch-level feedback acts as a strong supervisory signal, helping the Generator produce outputs with sharper edges, realistic textures, and fewer artifacts.

3.5 CONCLUSION

This chapter gives a glimpse of all the essential requirements for the successful implementation of our image deblurring project. We have established the necessary software environment, including the specific Python version and key libraries such as TensorFlow, PyTorch, and others required for model development, training, and evaluation. Furthermore, we have defined the minimum hardware capacity and operating system compatibility to ensure the project can be executed effectively. The non-functional requirements, focusing on compatibility, capacity, environment, performance and reliability, provide a framework for assessing the overall quality and usability of the final deblurring solution. This comprehensive requirement analysis lays a solid foundation for the subsequent development and deployment of the ResNet-based generator and PatchGAN discriminator model for image deblurring.

4. DESIGN

4.1 INTRODUCTION

Unified modelling language (UML) is used to represent the software in a graphical format, that is it provides a standard way to visualize how a system is designed. Good UML design is followed for a better and precise software output. The UML provides different constructs for specifying, visualizing, constructing and documenting the artifacts of software systems UML diagrams are used to better understand the system, to maintain the document information about the system and emphasizes on the roles, actors, actions, actions and process. The UML, diagrams considered are:

- Use-case Diagram
- Class Diagram
- Sequence Diagram
- Activity Diagram

Relationships: The relationships among different entities in following diagrams are given below:

- **Association** - This relationship tells how two entities interact with each other
- **Dependency** - An entity is dependent on another if the change of that is reflecting the other.
- **Aggregation** - It is a special kind of association which exhibits part-of relationship.
- **Generalization** - This relationship describes the parent-child relationship among different entities.
- **Realization** - It is a kind of relationship in which one thing specifies the behavior or a responsibility to be carried out and the other thing carries out that behavior.

4.2 UML DIAGRAMS

4.2.1 Use Case Diagram

Use case diagrams capture the behavior of a system and help to emphasize the requirements of the system. It describes the high-level functionalities of the system. It consists of the following artifacts namely Actor, Use case, Secondary Actor and Use case Boundary.

Actor: A role of a user that interacts with the system being modeled is represented by an actor. In this project, the primary actor is the User, who initiates the interactions such as uploading the image and accessing the deblurred output.

Use-Case: A use case is a function that the system performs to help the user achieve their goals. In this project, there are five main use cases - Upload Image, Validate Image, Deblur Image using Model, Download Deblurred Image, and Share Deblurred Image each representing important functionalities of the image deblurring process.

Secondary Actor: Secondary actors are entities that assist in achieving the goals but do not directly initiate the use cases. In this diagram, the System is considered a secondary actor as it handles all the internal processes such as validation, image processing, and result delivery.

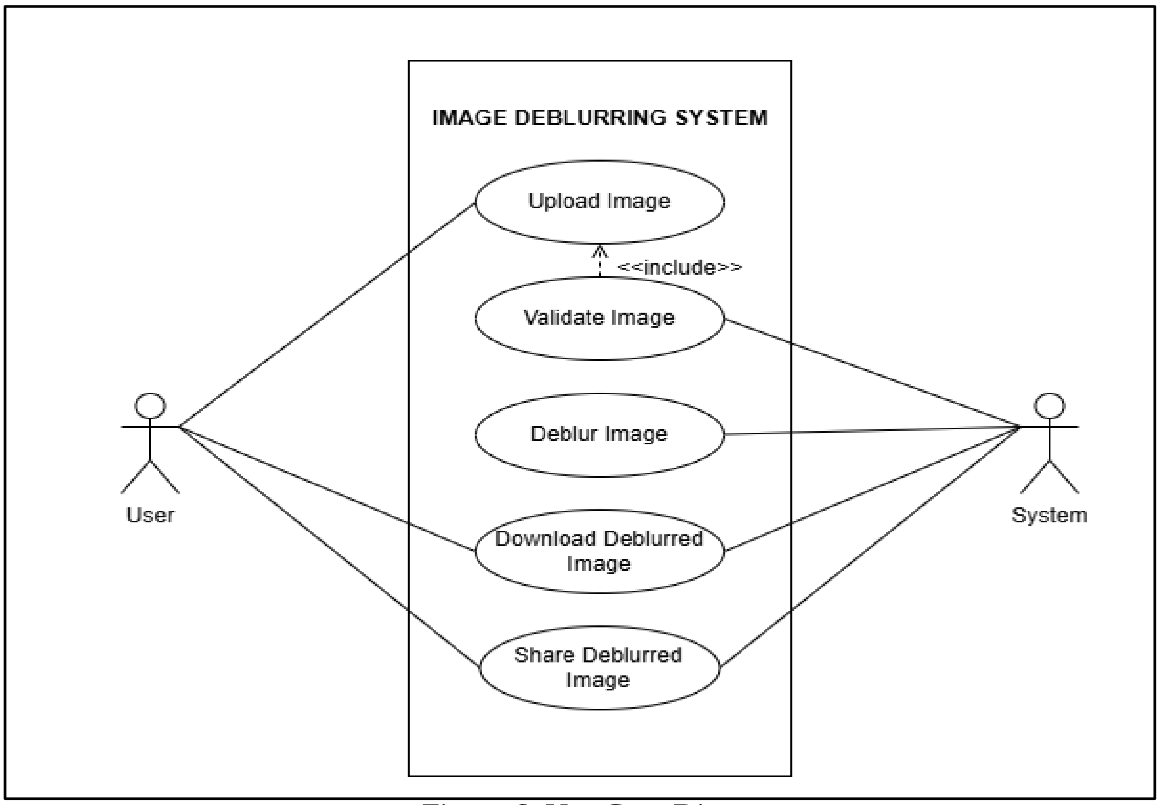


Figure 6. Use Case Diagram

4.2.2 Class Diagram

A Class Diagram is a static overview of the system used to highlight the important aspects as well as executables in the system. These are the only diagrams which can be directly mapped with object-oriented systems. Class diagrams show the responsibilities of a class and the relationships among different classes in the system.

Class: A class represents a collection of similar objects and consists of a name, responsibilities, and attributes. The classes considered in our Image Deblurring Project are: User, Model, and System.

Attributes: Attributes are the properties or characteristics of a class. The names of the classes, their attributes, responsibilities, and relationships are explained in detail in the following table 8.

Table 8. Class Diagram Artifacts

S. No.	**Class Name**	**Attributes**	**Responsibilities**	**Relationships**
1	User	name, email and phoneNo	uploadImage(), downloadImage(), shareImage()	Interacts with ValidateImage, System
2	Model	modelName, modelType	processImage(), generateDeblurredImage()	Provides processed image to System
3	System	systemId, version	provideDeblurredImage(), validateImageSize(), validateImageFormat()	Coordinates between User and DeblurringModel, receives image from User, interacts with System

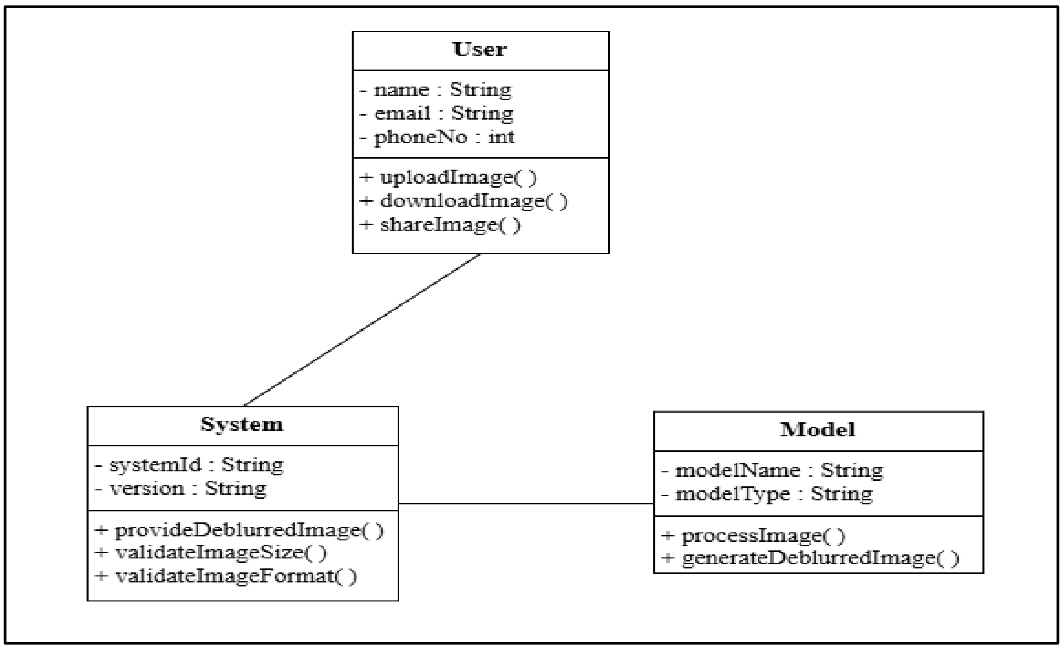


Figure 7. Class Diagram

4.2.3 Sequence Diagram

The sequence diagram, also known as an event diagram, depicts the flow of messages through the system. It aids in the visualization of a variety of dynamic scenarios. It represents communication between two lifelines as a time-ordered series of events, as if these lifelines were present at the same time. The message flow is represented by a vertical dotted line in UML, while the lifeline is represented by a vertical bar. Sequence diagrams also encompass iterations and branching. The sequence diagram in Figure 8 displays the order in which different elements of the system interact and the sequence in which these interactions occur. It illustrates how and in what order the components of the system work together to complete the image deblurring process.

Lifeline: A lifeline represents an individual participant in the sequence diagram. In this project, the lifelines are the User and the System, indicating their involvement and interaction over time.

Actor: An actor is a role that interacts with the system but is not part of the system itself. In our project, the User is the primary actor who initiates the upload and takes actions like downloading or sharing the output.

Message: Messages represent the communication between lifelines. In this diagram, messages are exchanged between the User and the System to perform tasks such as image upload, validation, processing, and delivery of the deblurred image. There are 10 messages in total, which include both call and return messages.

Call Message: These are messages that invoke an operation in the receiving lifeline. For example, the User sends a message to the System to Upload Image, and the System in turn processes several steps like Validate Image, Send Valid Image for Processing, and Process Image with Deblurring Model.

Return Message: These messages indicate the flow of data from the receiver of a call back to the sender. For example, if the image is invalid, the system returns a message to the user stating Reject if Invalid, or if successful, it Provides Deblurred Image for Downloading or Sharing.

The sequence diagram provides a detailed view of how data and control flow between the

User and the System, from the moment an image is uploaded to when it is either downloaded or shared. It visually communicates the interaction pattern and the order of operations in the image deblurring process.

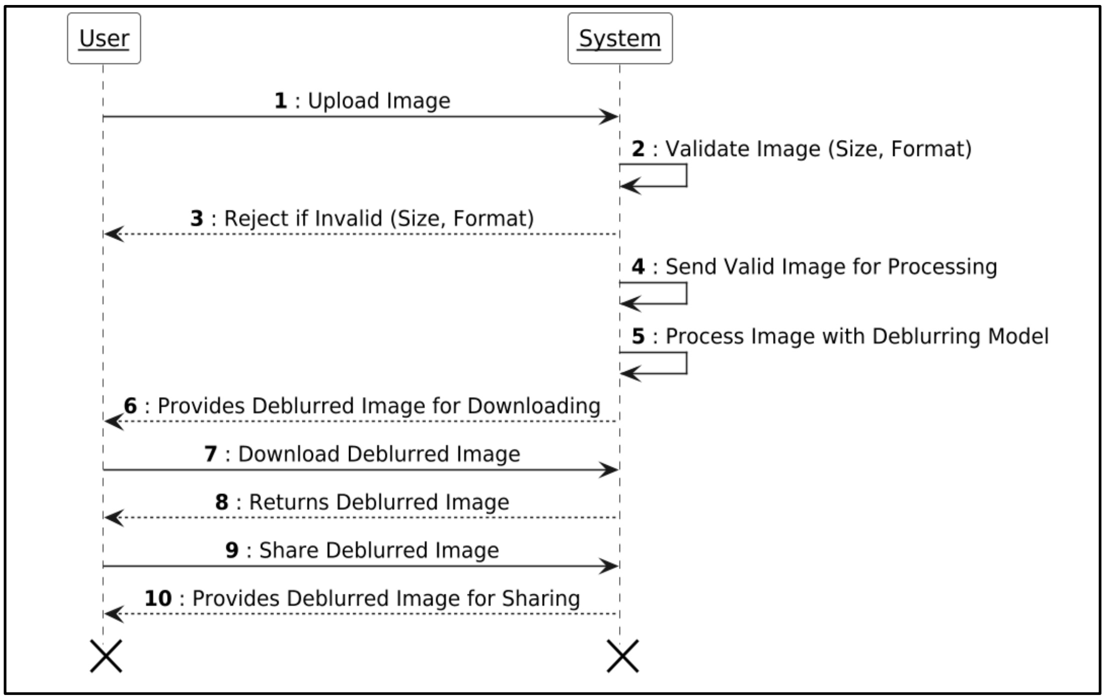

User
System
1 : Upload Image
2 : Validate Image (Size, Format)
3 : Reject if Invalid (Size, Format)
4 : Send Valid Image for Processing
5 : Process Image with Deblurring Model
6 : Provides Deblurred Image for Downloading
7 : Download Deblurred Image
8 : Returns Deblurred Image
9 : Share Deblurred Image
10 : Provides Deblurred Image for Sharing

Figure 8. Sequence Diagram

4.2.4 Activity Diagram

Activity diagrams shows the flow of activities within a system, illustrating how different actions lead to desired outcomes. They highlight the sequence of steps and decision points involved in achieving specific goals. Similar to use case diagrams, activity diagrams consist of various artifacts as explained below:

Activity: Each activity represents a specific action or task performed within the system to accomplish a goal. In our project, activities include “uploading the image,” “validating the image,” “processing the image with the deblurring model,” and “providing the deblurred image to the user.”

Actor: Actors represent roles or entities interacting with the system. The primary actor, User, initiates the workflow by uploading the image and later decides to either download or share the final deblurred image.

Activity Boundary: This boundary encloses all activities within the system and may include decision nodes, control flows, and objects that guide the sequence of operations.

The activity diagram in our project depicts the flow of activities involved in uploading an image and generating the deblurred output. It starts with the user uploading an image, followed by a validation step that checks the image’s size and format. Based on this validation, the system either rejects the image and notifies the user or processes it further. If valid, the image is sent through a deblurring model to generate a clearer version. Once processing is complete, the system provides the deblurred image, and the user can then choose to download or share it.

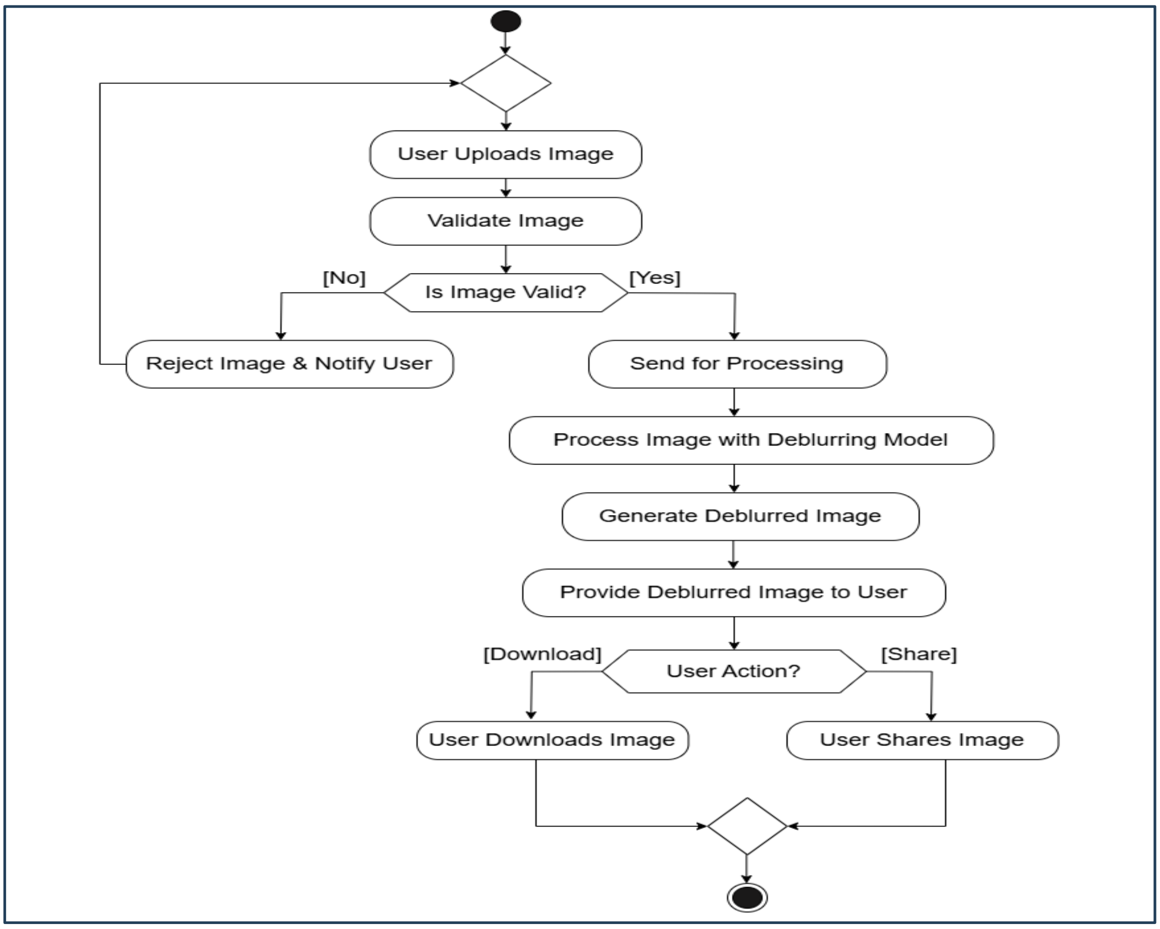


Figure 9. Activity Diagram

4.3 MODULE DESIGN AND ORGANIZATION

A module is defined as the unique and addressable components of the software which can be solved and modified independently without disturbing (or affecting in very small amount) other modules of the software. Thus, every software design should follow modularity. The process of breaking down a software into multiple independent modules where each module is developed separately is called Modularization. In this project we have the following modules, a pictorial illustration of the same is shown as architecture diagram in the Figure 10.

- Data Collection
- Data Preprocessing
- Dataset Conversion
- Models Training
- Model Evaluation
- User Interface

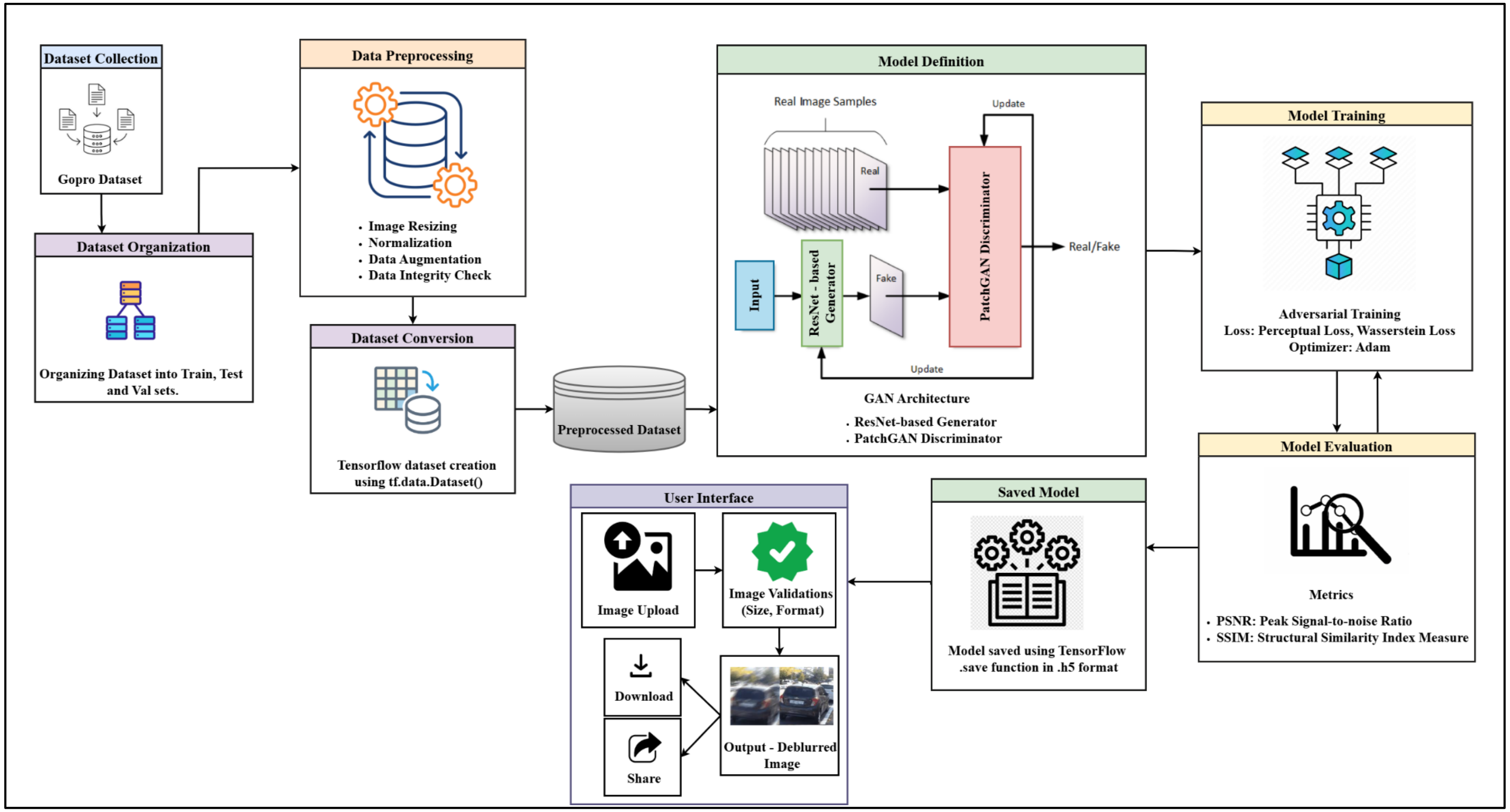


Figure 10. Module Design Diagram

Dataset collection

The dataset utilized in this project is the GOPRO dataset, which consists of high-quality paired images blurred inputs and their corresponding sharp images. Initially, the raw dataset was structured with multiple subfolders containing both blur and sharp image sequences.

To streamline the training process, the dataset was reorganized into three main sets: train, validation, and test. Within each set, the images were further categorized into blur and sharp folders. During this process, care was taken to preserve image pair integrity and to avoid filename conflicts by appending subfolder names to each file. The final structured dataset allows for efficient and error-free loading during model training and evaluation.

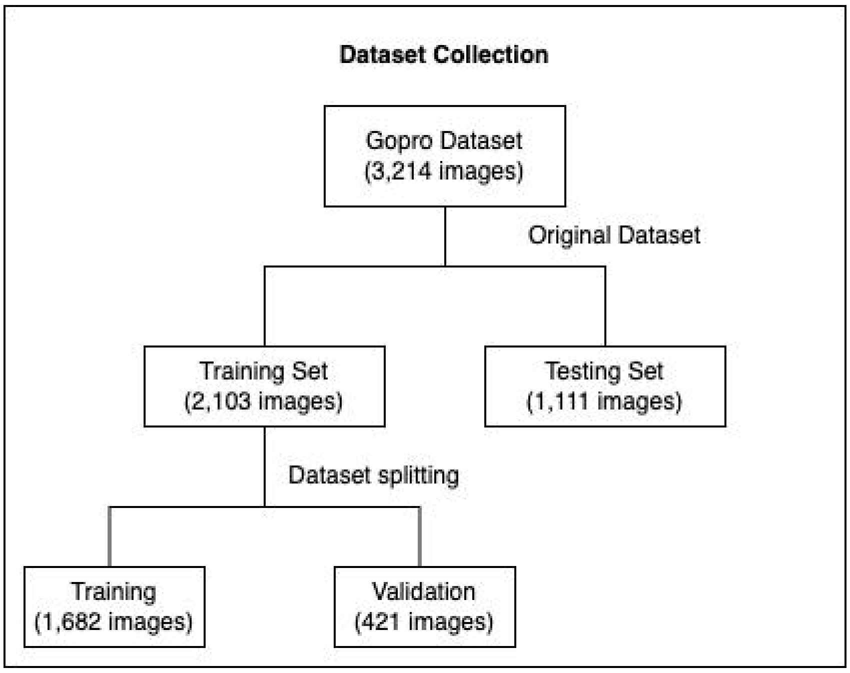


Figure 11. Architecture of Dataset Collection

Data Preprocessing

Preprocessing ensures the dataset is in a standard format (e.g. all images are the same size). This step is essential to ensure that the dataset is consistent before training a model. The Data Preprocessing process in our project includes resizing, normalization and augmentation of the images, as shown in Figure 12.

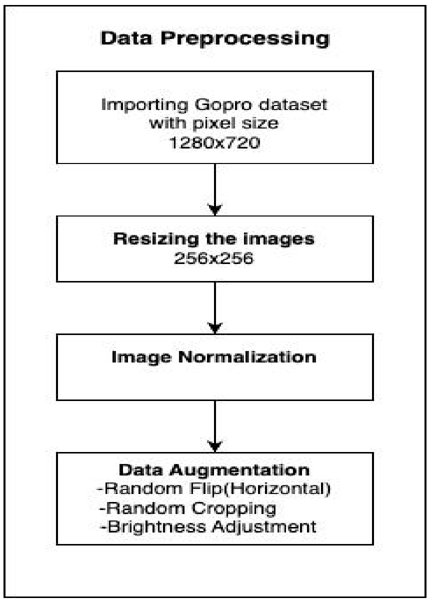


Figure 12. Architecture of Data Preprocessing

(A) Image Resizing: As part of preprocessing, all images were resized to a fixed dimension of 256 × 256 pixels from 1280x720 pixels. This uniformity ensures consistency across the dataset and compatibility with the neural network architecture used in the model.

(B) Normalization: After resizing, the pixel values of the images were normalized to fall within the range of [-1, 1]. This normalization is important for deep learning workflows, as it helps stabilize training, speeds up convergence, and allows the model to learn more effectively by keeping input values within a predictable range.

(C) Data Augmentation: Image augmentation is a crucial step in improving the model's generalization by introducing variability in the training data. In our project, augmentation techniques were applied to both blurred and sharp image pairs to simulate real-world conditions and prevent overfitting. We employed operations such as random horizontal flipping, random cropping, resizing, and brightness adjustment using TensorFlow's augmentation functions.

- **Random Flip:** We applied a stateless random horizontal flip to the input images. This mirrors the image from left to right and ensures that both the blurred and sharp images remain aligned post-transformation.

- **Random Cropping and Resizing:** Images were first stacked and randomly cropped to 224×224, then resized to 256×256. This helps the model focus on smaller regions and promotes spatial invariance.
- **Brightness Adjustment:** Brightness levels were randomly varied using a fixed seed to ensure consistent augmentation across both blurred and sharp images. This simulates variations in lighting and camera exposure. In our project we adjusted the brightness level of the dataset in the range [-0.2, +0.2].

Dataset Conversion

After preprocessing the dataset is converted into a TensorFlow dataset where images are loaded into the model in form of tensors for speeding up the training process. Includes convertion of the organized GoPro dataset into a tf.data.Dataset for efficient loading, shuffling, and batching. The dataset is stored in TensorFlow format, including .shard and .snapshot directories along with metadata files like dataset_spec.pb and snapshot.metadata.

Model Training

To achieve high-quality deblurring, we adopted a GAN-based architecture consisting of a ResNet-based Generator and a PatchGAN-style Discriminator. The training process was designed to balance perceptual quality and adversarial learning by incorporating multiple loss functions and evaluation metrics.

The Generator is a ResNet-based structure. It uses reflection padding, downsampling layers, nine ResNet blocks, and bilinear upsampling. The Discriminator is trained to distinguish between real sharp images and generated outputs, using convolutional layers and zero-padding to achieve patch-level classification.

- **Loss Functions:** We used a combination of Wasserstein loss, Perceptual loss (with VGG16’s intermediate layer), and Gradient Penalty to stabilize training and improve the perceptual quality of outputs.
- **Optimizers:** Adam optimizers with carefully tuned learning rates were used for both generator and discriminator, including clipnorm to avoid exploding gradients.

- **Training Details:** The model was trained with a batch size of 8, using TensorFlow's efficient @tf.function for performance. Checkpoints were saved at each epoch to preserve model states.

User Interface

The user interface provides a simple and interactive platform where users can upload a blurred image for deblurring. Once uploaded, the system performs validation checks on the image uploaded (size, format and if the image is empty i.e. it has valid objects or not) and then processes the image using a deep learning model and displays the deblurred result. Users are given the option to either download the deblurred image or share it directly via WhatsApp using an integrated share button. This makes the tool easy to use and accessible, even for non-technical users.

4.4 CONCLUSION

The UML diagrams are used to plot the functionalities of the system in advance. The architecture and the UML diagrams makes the system vivid and perfect for implementation. The division of the project into modules not only makes it easier for understanding the purpose but also in implementation. Implementation of these divided modules and debugging becomes easier. The visual representation of the modules and sub modules help in understanding of their working and also help in integration of the sub modules.

5. IMPLEMENTATION AND RESULTS

5.1 INTRODUCTION

The project's implementation stage is when the theoretical design is translated into a workable system. As a result, it can be seen as the most crucial stage in ensuring the success of a new system and giving the user confidence that the system will work and be effective. The implementation step entails meticulous planning, research of the existing system and its implementation limitations, designing of changeover methods and evaluation of changeover methods.

5.2 EXPLAINATION OF KEY FUNCTIONS

The key Functions in our system include the following:

5.2.1 Dataset Collection and Preprocessing

Before training a deep learning model, it’s important to prepare the data in an efficient way the model can understand and learn from effectively. In our project, we used the Gopro dataset, which contains the real-world examples of blurred and sharp image pairs perfect for training a deblurring model.

Each image is resized from 1280x720 to 256×256 pixels to maintain consistency and reduce computational load. We also normalize the pixel values to arrange of [-1,1], which helps the model train more smoothly and efficiently.

To ensure that our model not only learns effectively but also generalizes well to unseen data, we divided the dataset into three parts. The 2,103 training pairs provided with the dataset were further split into 80% for training (1,682 pairs) and 20% for validation (421 pairs) to fine-tune the model and prevent overfitting. The testing set, consisting of 1,111 image pairs, remained unchanged and was used for final evaluation.

We also use TensorFlow’s tf.data pipeline to speed up this entire process. It handles tasks like loading, shuffling, batching, and prefetching the data, making training much faster and more efficient behind the scenes.

5.2.2 ResNet-Based Generator Network

The generator is the core of our deblurring system. It is designed using a ResNet (Residual Network) architecture with Instance Normalization, which helps in maintaining sharp features even after several convolutional layers. The generator receives a blurred image and attempts to reconstruct a sharp image.

In the project, this generator is trained to convert low-quality, motion-blurred images into clean, sharp versions by extracting fine features. ResNet blocks with skip connections are used to ensure the flow of gradients and prevent vanishing during training.

5.2.3 PatchGAN-Based Discriminator Network

The discriminator is implemented using a PatchGAN architecture, which divides the image into small patches and evaluates the realism of each patch independently. This ensures that the generator produces high-frequency realistic textures instead of globally smooth images thus helping in refining the image further.

The discriminator learns to distinguish between real sharp images and fake (generated) ones. Its feedback forces the generator to produce more sharp outputs. Batch Normalization is used in the discriminator for training stability.

5.2.4 Model Training

The model training is done using a custom loop in TensorFlow, where blurred images are passed through a generator to produce sharp predictions. The training process combines perceptual loss and adversarial Wasserstein loss to guide the learning. Both the generator and discriminator are updated using their own optimizers. Throughout training, key performance metrics like PSNR, SSIM, and losses are tracked. The model is trained over multiple epochs with a batch size of 8, and the best version is saved for later use in image deblurring.

5.2.5 Model Evaluation Metrics – PSNR and SSIM

The performance of the deblurring model is evaluated using Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM). PSNR measures how close the generated image is to the original sharp image based on pixel intensities. SSIM evaluates the perceptual similarity, focusing on structure, luminance, and contrast.

Increasing PSNR and SSIM values during training indicate that the model is generating sharper and more accurate images.

5.2.6 Frontend Interface and User Interaction Handling

In this project we used Next.js to design the frontend that provides a user-friendly interface for uploading blurred images for deblurring. It performs essential validations such as checking the file size (max 50MB), ensuring only .png and .jpeg formats are accepted, and rejecting empty (no object) images. These validations enhance system performance, prevent unnecessary processing, and ensure that only meaningful inputs reach the backend for model inference.

Once an image passes all these validation checks, it’s sent to the backend where the trained TensorFlow model is loaded for inference. The backend receives the image, runs it through the model to perform deblurring, and sends back the sharpened version. All of this happens seamlessly behind the scenes, making the experience feel smooth and responsive for the user.

5.3 METHOD OF IMPLEMENTATION

This section outlines the technical implementation details of the image deblurring system. It covers both the frontend and backend code structure as well as the deep learning model architecture used for image deblurring. The implementation is divided into two major parts: the system's functional components and the core model training pipeline. Also this section outlines the results of the project and their analysis.

5.3.1 Forms

The entire application has been modularly implemented and categorized into different types of forms, each representing different layers of user interaction and system functionality:

Form 1: User Interface Forms

UI Forms (User Interface Forms) refer to the visual and interactive parts of an application that users directly engage with. In a web app like yours, UI forms are pages or components where users input data, trigger actions, and view results.

src/app/upload/page.tsx: Provides the main upload form where users can drag-and-drop or browse for an image.It also performs client-side validation for file type, size, and checks for blurriness or object presence and nitiates the image deblurring process upon successful validation.

```
"use client";
import { useState, useCallback, useEffect, useRef } from 'react';
import Image from 'next/image';
import { Download, RefreshCw, ZoomIn, ZoomOut, Share2, MoveLeft, ArrowLeft,
RotateCcw, Camera, Sparkles } from 'lucide-react';
import { useRouter } from 'next/navigation';
import { motion } from 'framer-motion';
import Link from 'next/link';
interface ImagePosition {
  x: number;
  y: number;
  scale: number;
}
export default function ProcessPage() {
  const router = useRouter();
  const [originalImage, setOriginalImage] = useState<string | null>(null);
  const [processedImage, setProcessedImage] = useState<string | null>(null);
  const [isProcessing, setIsProcessing] = useState(false);
  const [showExportOptions, setShowExportOptions] = useState(false);
  const [exportFormat, setExportFormat] = useState<'PNG' | 'JPEG'>('PNG');
  const [customFilename, setCustomFilename] = useState('');

  const [blurredPosition, setBlurredPosition] = useState<ImagePosition>({ x: 0, y: 0,
scale: 1 });
  const [deblurredPosition, setDeblurredPosition] = useState<ImagePosition>({ x: 0,
y: 0, scale: 1 });
  const blurredDragRef = useRef<{ isDragging: boolean; startX: number; startY:
number }>({ isDragging: false, startX: 0, startY: 0 });
  const deblurredDragRef = useRef<{ isDragging: boolean; startX: number; startY:
number }>({ isDragging: false, startX: 0, startY: 0 });
  useEffect(() => {
    const storedImage = localStorage.getItem('originalImage');
    if (!storedImage) {
      router.push('/upload');
      return;
    }
  setOriginalImage(storedImage);
  processImage(storedImage);
 }, [router]);
 const processImage = async (imageData: string) => {
  try {
   setIsProcessing(true);
   const response = await fetch(imageData);
   const blob = await response.blob();
   const formData = new FormData();
   formData.append('image', blob);
   const apiResponse = await fetch('/api/deblur', {
    method: 'POST',
    body: formData,
   });
   const data = await apiResponse.json();
   if (!apiResponse.ok) {
    throw new Error(data.error + (data.details ? `\n\nDetails: ${data.details}` : ''));
   }
   if (!data.processedImage) {
    throw new Error('No processed image received from the model');
   }
   setProcessedImage(`data:image/jpeg;base64,${data.processedImage}`);
  } catch (error) {
   console.error('Error processing image:', error);
   alert(error instanceof Error ? error.message : 'Failed to process image. Please try
again.');
   router.push('/upload');
  } finally {
   setIsProcessing(false);
  }
 };
 const handleZoom = (
  e: React.WheelEvent<HTMLDivElement>,
  setPosition: React.Dispatch<React.SetStateAction<ImagePosition>>,
  currentPosition: ImagePosition
 ) => {
  if (e.ctrlKey || e.metaKey) {
   e.preventDefault();
   const delta = e.deltaY;
   const zoomFactor = 0.1;
   const newScale = delta > 0 ?
    Math.max(0.1, currentPosition.scale - zoomFactor) :
    Math.min(3, currentPosition.scale + zoomFactor);

   setPosition(prev => ({
    ...prev,
    scale: newScale
   }));
  }
 };
 const handleImageWheel = (e: WheelEvent, isBlurred: boolean) => {
  e.preventDefault();
  const target = e.currentTarget as HTMLElement;
  if (!target) return;
  const rect = target.getBoundingClientRect();
  const x = e.clientX - rect.left;
  const y = e.clientY - rect.top;
  const deltaY = e.deltaY;
  const scaleChange = deltaY > 0 ? 0.9 : 1.1;
  if (isBlurred) {
   const newScale = Math.min(Math.max(blurredPosition.scale * scaleChange, 0.5),
3);
   setBlurredPosition({ ...blurredPosition, scale: newScale });
  } else {
   const newScale = Math.min(Math.max(deblurredPosition.scale * scaleChange,
0.5), 3);
   setDeblurredPosition({ ...deblurredPosition, scale: newScale });
  }
 };
 const handleMouseDown = (e: React.MouseEvent, isBlurred: boolean) => {
  const dragRef = isBlurred ? blurredDragRef : deblurredDragRef;
  dragRef.current = {
   isDragging: true,
   startX: e.clientX - (isBlurred ? blurredPosition.x : deblurredPosition.x),
   startY: e.clientY - (isBlurred ? blurredPosition.y : deblurredPosition.y),
  };
 };
 const handleMouseMove = (e: React.MouseEvent, isBlurred: boolean) => {
  const dragRef = isBlurred ? blurredDragRef : deblurredDragRef;
  if (!dragRef.current.isDragging) return;
  const newX = e.clientX - dragRef.current.startX;
  const newY = e.clientY - dragRef.current.startY;
  requestAnimationFrame(() => {
   if (isBlurred) {
    setBlurredPosition(prev => ({ ...prev, x: newX, y: newY }));
   } else {
    setDeblurredPosition(prev => ({ ...prev, x: newX, y: newY }));
   }
  });
 };
 const handleMouseUp = (isBlurred: boolean) => {
  const dragRef = isBlurred ? blurredDragRef : deblurredDragRef;
  dragRef.current.isDragging = false;
 };
 const resetPositions = () => {
  setBlurredPosition({ x: 0, y: 0, scale: 1 });
  setDeblurredPosition({ x: 0, y: 0, scale: 1 });
 };
 const handleDownload = async () => {
  if (!processedImage) return;
```

```
  if (showExportOptions) {
    const link = document.createElement('a');
    const canvas = document.createElement('canvas');
    const ctx = canvas.getContext('2d');
    const img = new window.Image();
    img.onload = () => {
      canvas.width = img.width;
      canvas.height = img.height;
      ctx?.drawImage(img, 0, 0);
      const filename = customFilename || `deblurred_${Date.now()}`;
      const mimeType = exportFormat === 'PNG' ? 'image/png' : 'image/jpeg';
      canvas.toBlob((blob) => {
        if (blob) {
          const url = URL.createObjectURL(blob);
          link.href = url;
          link.download = `${filename}.${exportFormat.toLowerCase()}`;
          link.click();
          URL.revokeObjectURL(url);
        }
      }, mimeType);
    };
    img.src = processedImage;
  } else {
    const link = document.createElement('a');
    link.href = processedImage;
    link.download = `deblurred_${Date.now()}.png`;
    link.click();
  }
};
const handleShare = async () => {
  if (!processedImage) return;
  try {
    if (navigator.share) {
      const blob = await fetch(processedImage).then(r => r.blob());
      const file = new File([blob], 'deblurred-image.jpg', { type: 'image/jpeg' });
      await navigator.share({
        title: 'Deblurred Image',
        files: [file]
      });
    } else {
      alert('Sharing is not supported on this device/browser');
    }
  } catch (error) {
    console.error('Error sharing:', error);
  }
};
useEffect(() => {
  const preventDefault = (e: WheelEvent) => {
    if (e.ctrlKey) {
      e.preventDefault();
    }
  };
  document.addEventListener('wheel', preventDefault, { passive: false });
  return () => document.removeEventListener('wheel', preventDefault);
}, []);
return (
  <div className="min-h-screen bg-gradient-to-br from-purple-50 via-blue-50 to-
white dark:from-gray-900 dark:via-gray-800 dark:to-gray-900 relative">
    <div
      className="fixed inset-0 opacity-30 dark:opacity-15 pointer-events-none"
      style={{
        backgroundImage: 'url(/demo/bg.gif)',
        backgroundSize: '100% 100%',
        backgroundPosition: 'center',
        backgroundRepeat: 'no-repeat',
        width: '100vw',
        height: '100vh'
      }}
    />
    <main className="container mx-auto px-4 py-8 relative z-10">
      <div className="max-w-6xl mx-auto mb-8">
        <div className="flex items-center justify-between">
          <motion.div
            initial={{ opacity: 0, x: -20 }}
            animate={{ opacity: 1, x: 0 }}
            transition={{ duration: 0.5 }}
          >
            <Link
              href="/upload"
              className="inline-flex items-center gap-2 text-gray-600 hover:text-gray-
900 dark:text-gray-400 dark:hover:text-white transition-colors"
            >
              <ArrowLeft className="h-5 w-5" />
              Back to Upload
            </Link>
          </motion.div>
          <motion.button
            onClick={resetPositions}
            className="inline-flex items-center gap-2 text-gray-600 hover:text-gray-
900 dark:text-gray-400 dark:hover:text-white transition-colors"
            initial={{ opacity: 0, x: 20 }}
            animate={{ opacity: 1, x: 0 }}
            transition={{ duration: 0.5 }}
            whileHover={{ scale: 1.05 }}
            whileTap={{ scale: 0.95 }}
          >
            <RotateCcw className="h-5 w-5" />
            Reset View
          </motion.button>
        </div>
      </div>
      <motion.div
        className="grid grid-cols-1 md:grid-cols-2 gap-8 max-w-6xl mx-auto
relative"
        initial={{ opacity: 0, y: 20 }}
        animate={{ opacity: 1, y: 0 }}
        transition={{ duration: 0.5, delay: 0.2 }}
      >
        <div className="space-y-2 relative">
          <div className="flex items-center justify-between mb-2">
            <h2 className="text-lg font-medium">Blurred Image</h2>
            <div className="flex items-center gap-2">
              <motion.button
                onClick={() => {
                  const newScale = Math.min(blurredPosition.scale * 1.2, 3);
                  setBlurredPosition({ ...blurredPosition, scale: newScale });
                }}
                className="p-2 rounded-full hover:bg-gray-100 dark:hover:bg-gray-800
transition-colors"
                whileHover={{ scale: 1.1 }}
                whileTap={{ scale: 0.9 }}
              >
                <ZoomIn className="h-5 w-5" />
              </motion.button>
              <motion.button
                onClick={() => {
                  const newScale = Math.max(blurredPosition.scale * 0.8, 0.5);
                  setBlurredPosition({ ...blurredPosition, scale: newScale });
                }}
                className="p-2 rounded-full hover:bg-gray-100 dark:hover:bg-gray-800
transition-colors"
                whileHover={{ scale: 1.1 }}
                whileTap={{ scale: 0.9 }}
              >
                <ZoomOut className="h-5 w-5" />
              </motion.button>
              <span className="text-sm text-gray-
500">{Math.round(blurredPosition.scale * 100)}%</span>
            </div>
          </div>
          <motion.div
            className="relative aspect-video w-full overflow-hidden rounded-2xl bg-
gray-100 dark:bg-gray-800 cursor-move shadow-lg"
            initial={{ opacity: 0, scale: 0.95 }}
            animate={{ opacity: 1, scale: 1 }}
            transition={{ duration: 0.5 }}
            onMouseDown={(e) => handleMouseDown(e, true)}
            onMouseMove={(e) => handleMouseMove(e, true)}
            onMouseUp={() => handleMouseUp(true)}
            onMouseLeave={() => handleMouseUp(true)}
            onWheel={(e) => handleZoom(e, setBlurredPosition, blurredPosition)}
            style={{ touchAction: 'none' }}
          >
            {originalImage && (
              <div
                className="absolute inset-0"
                style={{
                  transform: `translate(${blurredPosition.x}px, ${blurredPosition.y}px)
scale(${blurredPosition.scale})`,
                  transformOrigin: 'center',
                  transition: 'transform 0.1s ease-out'
                }}
              >
                <Image
                  src={originalImage}
                  alt="Original"
                  fill
                  sizes="(max-width: 768px) 100vw, 50vw"
                  className="object-contain"
                />
              </div>
            )}
          </motion.div>
        </div>
        <div className="space-y-2 relative">
          <div className="flex items-center justify-between mb-2">
            <h2 className="text-lg font-medium">Deblurred Image</h2>
            <div className="flex items-center gap-2">
              <motion.button
                onClick={() => {
                  const newScale = Math.min(deblurredPosition.scale * 1.2, 3);
                  setDeblurredPosition({ ...deblurredPosition, scale: newScale });
                }}
                className="p-2 rounded-full hover:bg-gray-100 dark:hover:bg-gray-800
transition-colors"
                whileHover={{ scale: 1.1 }}
                whileTap={{ scale: 0.9 }}
              >
                <ZoomIn className="h-5 w-5" />
              </motion.button>
              <motion.button
                onClick={() => {
                  const newScale = Math.max(deblurredPosition.scale * 0.8, 0.5);
                  setDeblurredPosition({ ...deblurredPosition, scale: newScale });
                }}
                className="p-2 rounded-full hover:bg-gray-100 dark:hover:bg-gray-800
transition-colors"
                whileHover={{ scale: 1.1 }}
                whileTap={{ scale: 0.9 }}
              >
                <ZoomOut className="h-5 w-5" />
              </motion.button>
              <span className="text-sm text-gray-
500">{Math.round(deblurredPosition.scale * 100)}%</span>
            </div>
          </div>
          <motion.div
            className="relative aspect-video w-full overflow-hidden rounded-2xl bg-
gray-100 dark:bg-gray-800 cursor-move shadow-lg"
            initial={{ opacity: 0, scale: 0.95 }}
            animate={{ opacity: 1, scale: 1 }}
            transition={{ duration: 0.5, delay: 0.2 }}
            onMouseDown={(e) => handleMouseDown(e, false)}
            onMouseMove={(e) => handleMouseMove(e, false)}
            onMouseUp={() => handleMouseUp(false)}
            onMouseLeave={() => handleMouseUp(false)}
            onWheel={(e) => handleZoom(e, setDeblurredPosition, deblurredPosition)}
            style={{ touchAction: 'none' }}
          >
            {processedImage && (
              <div
                className="absolute inset-0"
                style={{
                  transform: `translate(${deblurredPosition.x}px,
${deblurredPosition.y}px) scale(${deblurredPosition.scale})`,
                  transformOrigin: 'center',
                  transition: 'transform 0.1s ease-out'
                }}
              >
                <Image
                  src={processedImage}
                  alt="Enhanced"
                  fill
                  sizes="(max-width: 768px) 100vw, 50vw"
                  className="object-contain"
                />
              </div>
            )}
            {isProcessing && (
              <div className="absolute inset-0 bg-background/80 backdrop-blur-sm flex
items-center justify-center">
                <div className="text-center">
                  <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-
primary mx-auto mb-2"></div>
                  <p className="text-primary font-medium">Processing...</p>
                </div>
              </div>
            )}
          </motion.div>
        </div>
```

```
        </motion.div>
        <motion.div
          className="flex justify-end gap-4 mt-8 max-w-6xl mx-auto px-4"
          initial={{ opacity: 0, y: 20 }}
          animate={{ opacity: 1, y: 0 }}
          transition={{ duration: 0.5, delay: 0.4 }}
        >
          <motion.button
            onClick={() => setShowExportOptions(true)}
            className="inline-flex items-center gap-2 bg-gradient-to-r from-purple-600
to-blue-600 hover:from-purple-700 hover:to-blue-700 text-white px-6 py-3 rounded-
full text-lg font-medium transition-all"
            whileHover={{ scale: 1.05 }}
            whileTap={{ scale: 0.95 }}
          >
            <Download className="h-5 w-5" />
            Download
          </motion.button>
          <motion.button
            onClick={handleShare}
            className="inline-flex items-center gap-2 bg-white dark:bg-gray-800 text-
gray-900 dark:text-white px-6 py-3 rounded-full text-lg font-medium shadow-lg
hover:shadow-xl transition-all"
            whileHover={{ scale: 1.05 }}
            whileTap={{ scale: 0.95 }}
          >
            <Share2 className="h-5 w-5" />
            Share
          </motion.button>
        </motion.div>
        {showExportOptions && (
          <div className="fixed inset-0 bg-black/50 flex items-center justify-center z-
50">
            <div className="bg-white dark:bg-gray-800 p-6 rounded-lg shadow-xl max-
w-md w-full">
              <h3 className="text-lg font-semibold mb-4">Export Options</h3>
              <div className="space-y-4">
                <div>
                  <label className="block text-sm font-medium mb-1">Format</label>
                  <select
                    value={exportFormat}
                    onChange={(e) => setExportFormat(e.target.value as 'PNG' | 'JPEG')}
                    className="w-full p-2 border rounded dark:bg-gray-700"
                  >
                    <option value="PNG">PNG</option>
                    <option value="JPEG">JPEG</option>
                  </select>
                </div>
                <div>
                  <label className="block text-sm font-medium mb-1">Filename</label>
                  <input
                    type="text"
                    value={customFilename}
                    onChange={(e) => setCustomFilename(e.target.value)}
                    placeholder="Enter filename"
                    className="w-full p-2 border rounded dark:bg-gray-700"
                  />
                </div>
                <div className="flex justify-end gap-2">
                  <button
                    onClick={() => setShowExportOptions(false)}
                    className="px-4 py-2 text-sm font-medium text-gray-700 dark:text-
gray-300 hover:bg-gray-100 dark:hover:bg-gray-700 rounded"
                  >
                    Cancel
                  </button>
                  <button
                    onClick={() => {
                      handleDownload();
                      setShowExportOptions(false);
                    }}
                    className="px-4 py-2 text-sm font-medium text-white bg-blue-600
hover:bg-blue-700 rounded"
                  >
                    Export
                  </button>
                </div>
              </div>
            </div>
          </div>
        )}
      </main>
    </div>
  );
}
```

src/app/process/page.tsx: Displays the deblurred image alongside the original for visual comparison.It also includes zoom/pan controls, result details, and options to export or reset the image and handles loading states and processing transitions**.**

```
"use client";
import { useState, useCallback } from 'react';
import { useDropzone } from 'react-dropzone';
import { Upload, MoveLeft, Camera, Sparkles, ImageIcon } from 'lucide-react';
import { useRouter } from 'next/navigation';
import { motion } from 'framer-motion';
export default function UploadPage() {
  const router = useRouter();
  const [isDragging, setIsDragging] = useState(false);
  const [uploadedFile, setUploadedFile] = useState<{ name: string; data: string } |
null>(null);
  const checkImageContent = (imageElement: HTMLImageElement):
Promise<boolean> => {
    return new Promise((resolve) => {
      const canvas = document.createElement('canvas');
      const ctx = canvas.getContext('2d');
      canvas.width = imageElement.width;
      canvas.height = imageElement.height;
      if (ctx) {
        ctx.drawImage(imageElement, 0, 0);
        const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
        const data = imageData.data;
        const histR = new Array(256).fill(0);
        const histG = new Array(256).fill(0);
        const histB = new Array(256).fill(0);
        let totalPixels = 0;
        for (let i = 0; i < data.length; i += 4) {
          histR[data[i]]++;
          histG[data[i + 1]]++;
          histB[data[i + 2]]++;
          totalPixels++;
        }
        const calculateEntropy = (hist: number[]) => {
          return hist.reduce((entropy, count) => {
            if (count === 0) return entropy;
            const p = count / totalPixels;
            return entropy - (p * Math.log2(p));
          }, 0);
        };
        const entropyR = calculateEntropy(histR);
        const entropyG = calculateEntropy(histG);
        const entropyB = calculateEntropy(histB);
        const avgEntropy = (entropyR + entropyG + entropyB) / 3;
        let sum = 0;
        let sumSq = 0;
        for (let i = 0; i < data.length; i += 4) {
          const gray = (data[i] + data[i + 1] + data[i + 2]) / 3;
          sum += gray;
          sumSq += gray * gray;
        }
        const mean = sum / totalPixels;
        const variance = (sumSq / totalPixels) - (mean * mean);
        const stdDev = Math.sqrt(variance);
        let uniformRegionCount = 0;
        for (let i = 0; i < data.length; i += 4) {
          const gray = (data[i] + data[i + 1] + data[i + 2]) / 3;
          if (Math.abs(gray - mean) < 15) {
            uniformRegionCount++;
          }
        }
        const uniformRatio = uniformRegionCount / totalPixels;
         const isEmpty = avgEntropy < 3 || uniformRatio > 0.9 || stdDev < 20;
        resolve(!isEmpty);
      } else {
        resolve(false);
      }
    });
  };
  const validateImage = (file: File): Promise<void> => {
    return new Promise((resolve, reject) => {
      const img = new Image();
      img.onload = async () => {
        try {
          const hasContent = await checkImageContent(img);
               if (!hasContent) {
            reject(new Error('The image appears to be empty or lacks meaningful content.
Please upload a valid image.'));
            return;
          }
               resolve();
        } catch (error) {
          console.warn('Validation warning:', error);
          reject(new Error('Failed to validate image. Please try a different image.'));
        }
      };
      img.onerror = () => reject(new Error('Failed to load image for validation'));
      img.src = URL.createObjectURL(file);
    });
  };
  const onDrop = useCallback(async (acceptedFiles: File[]) => {
    if (acceptedFiles.length === 0) return;
     const file = acceptedFiles[0];
    const allowedFormats = ['image/png', 'image/jpeg', 'image/jpg'];
    if (!allowedFormats.includes(file.type)) {
      alert('Only PNG and JPEG/JPG images are allowed.');
      return;
    }
    const maxSize = 50 * 1024 * 1024;
    if (file.size > maxSize) {
      alert('Image size should not exceed 50MB.');
      return;
    }
    try {
      await validateImage(file);
          const reader = new FileReader();
      reader.onload = () => {
        const base64String = reader.result as string;
        setUploadedFile({ name: file.name, data: base64String });
        localStorage.setItem('originalImage', base64String);
      };
      reader.readAsDataURL(file);
    } catch (error) {
      alert(error instanceof Error ? error.message : 'Failed to validate image');
    }
  }, []);
  const handleProcess = () => {
    if (uploadedFile) {
      router.push('/process');
    }
  };
  const { getRootProps, getInputProps, isDragActive } = useDropzone({
    onDrop: async (acceptedFiles, rejectedFiles) => {
      if (rejectedFiles.length > 0) {
        const file = rejectedFiles[0];
        if (file.errors[0]?.code === 'file-invalid-type') {
          alert('Only PNG and JPEG/JPG images are allowed.');
        } else if (file.errors[0]?.code === 'file-too-large') {
          alert('Image size should not exceed 50MB.');
        }
        return;
      }
      const file = acceptedFiles[0];
      if (!file) return;
      try {
        await validateImage(file);
               const reader = new FileReader();
        reader.onload = () => {
          const imageData = reader.result as string;
          setUploadedFile({ name: file.name, data: imageData });
          localStorage.setItem('originalImage', imageData);
        };
        reader.readAsDataURL(file);
      } catch (error) {
        alert(error instanceof Error ? error.message : 'Failed to validate image');
      }
    },
    accept: {
      'image/png': ['.png'],
      'image/jpeg': ['.jpg', '.jpeg']
    },
    maxSize: 50 * 1024 * 1024,
    multiple: false,
    onDragEnter: () => setIsDragging(true),
    onDragLeave: () => setIsDragging(false),
    onDropAccepted: () => setIsDragging(false)
  });
  const handleFileChange = async (e: React.ChangeEvent<HTMLInputElement>) =>
{
    const file = e.target.files?.[0];
    if (!file) return;
    const allowedFormats = ['image/png', 'image/jpeg', 'image/jpg'];
    if (!allowedFormats.includes(file.type)) {
      alert('Only PNG and JPEG/JPG images are allowed.');
      e.target.value = '';
      return;
    }
    const maxSize = 50 * 1024 * 1024;
    if (file.size > maxSize) {
      alert('Image size should not exceed 50MB.');
      e.target.value = '';
      return;
    }
    try {
      await validateImage(file);
      const reader = new FileReader();
      reader.onload = (e) => {
        const imageData = e.target?.result as string;
        setUploadedFile({ name: file.name, data: imageData });
        localStorage.setItem('originalImage', imageData);
      };
      reader.readAsDataURL(file);
    } catch (error) {
      alert(error instanceof Error ? error.message : 'Failed to validate image');
      e.target.value = '';
    }
  };

  return (
    <div className="min-h-screen bg-gradient-to-br from-purple-50 via-blue-50 to-
white dark:from-gray-900 dark:via-gray-800 dark:to-gray-900 relative">
      {/* Background Image */}
      <div
        className="fixed inset-0 opacity-30 dark:opacity-15 pointer-events-none"
        style={{
          backgroundImage: 'url(/demo/bg.gif)',
          backgroundSize: '100% 100%',
          backgroundPosition: 'center',
          backgroundRepeat: 'no-repeat',
          width: '100vw',
          height: '100vh'
        }}
      />
      <main className="container mx-auto px-4 py-8 relative z-10">
        <div className="max-w-4xl mx-auto">
          <motion.div
            initial={{ opacity: 0, x: -20 }}
            animate={{ opacity: 1, x: 0 }}
            transition={{ duration: 0.5 }}
            className="mb-8"
          >
            <a href="/" className="inline-flex items-center gap-2 text-gray-600
  hover:text-gray-900 dark:text-gray-400 dark:hover:text-white transition-colors">
              <MoveLeft className="h-5 w-5" />
              Back to Home
            </a>
          </motion.div>
          <motion.div
            initial={{ opacity: 0, y: 20 }}
            animate={{ opacity: 1, y: 0 }}
            transition={{ duration: 0.5, delay: 0.1 }}
            className="text-center mb-12"
          >
            <h1 className="text-4xl font-bold mb-4">Upload Your Image</h1>
            <p className="text-gray-600 dark:text-gray-400">
              Choose a blurry image to enhance using our deep learning technology
            </p>
          </motion.div>
          <motion.div
            initial={{ opacity: 0, scale: 0.95 }}
            animate={{ opacity: 1, scale: 1 }}
            transition={{ duration: 0.5, delay: 0.2 }}
            className="relative"
          >
            <div
              {...getRootProps()}
              className={`border-2 border-dashed rounded-2xl p-12 transition-colors
  text-center relative overflow-hidden
                ${isDragActive ? 'border-purple-500 bg-purple-50 dark:bg-purple-900/10' :
  'border-gray-300 dark:border-gray-700'}
                hover:border-purple-500 dark:hover:border-purple-500`}
            >
              <input {...getInputProps()} onChange={handleFileChange} />
              <motion.div
                initial={false}
                animate={{ scale: isDragActive ? 1.1 : 1 }}
                transition={{ duration: 0.2 }}
                className="relative z-10"
              >
                <div className="mb-4">
                  <Upload className="h-12 w-12 mx-auto text-gray-400" />
                </div>
                <p className="text-lg mb-2">
                  {isDragActive ? 'Drop your image here' : 'Drag & drop your image here'}
                </p>
                <p className="text-sm text-gray-500 dark:text-gray-400 mb-4">
                  or click to browse
                </p>
                {uploadedFile && (
                  <div className="mt-4 p-4 bg-purple-50 dark:bg-purple-900/20 rounded-
  lg">
                    <p className="text-sm text-gray-600 dark:text-gray-300">
                      Uploaded: {uploadedFile.name}
                    </p>
                  </div>
                )}
              </motion.div>
            </div>
            {uploadedFile && (
              <motion.div
                initial={{ opacity: 0, y: 20 }}
                animate={{ opacity: 1, y: 0 }}
                transition={{ duration: 0.5 }}
                className="mt-6 text-center"
              >
                <button
                  onClick={handleProcess}
                  className="inline-flex items-center gap-2 bg-gradient-to-r from-purple-
  600 to-blue-600 hover:from-purple-700 hover:to-blue-700 text-white px-8 py-3
  rounded-full text-lg font-medium transition-all transform hover:scale-105"
                >
                  <Sparkles className="h-5 w-5" />
                  Process Image
                </button>
              </motion.div>
            )}
          </motion.div>
        </div>
      </main>
    </div>
   );
  }
```

Form 2: API Forms

API Forms refer to the parts of your application that handle communication between the frontend and backend. They are usually endpoints (e.g., in route.ts) that process requests, interact with logic or databases, and return responses.

api/deblur/route.ts: This form acts as the backend endpoint to handle image deblurring requests from the frontend. It also saves the uploaded image, invokes deblur.py, and returns the processed result as base64and ensures proper response formatting and error handling.

```
import { NextRequest, NextResponse } from 'next/server';
import { spawn } from 'child_process';
import { writeFile, readFile, unlink, mkdir } from 'fs/promises';
import { existsSync } from 'fs';
import path from 'path';
import { randomUUID } from 'crypto';
export async function POST(request: NextRequest) {
  let inputPath: string | null = null;
  let outputPath: string | null = null;
  try {
    const data = await request.formData();
    const image = data.get('image') as File;
    if (!image) {
      return NextResponse.json(
        { error: 'No image provided' },
        { status: 400 }
      );
    }
    const uploadsDir = path.join(process.cwd(), 'public', 'uploads');
    if (!existsSync(uploadsDir)) {
      await mkdir(uploadsDir, { recursive: true });
    }
    const inputFileName = `input-${randomUUID()}.jpg`;
    const outputFileName = `output-${randomUUID()}.jpg`;
    inputPath = path.join(uploadsDir, inputFileName);
    outputPath = path.join(uploadsDir, outputFileName);
    const buffer = Buffer.from(await image.arrayBuffer());
    await writeFile(inputPath, buffer);
    const pythonProcess = spawn('python', [
      path.join(process.cwd(), '..', 'deblur.py'),
      inputPath,
      outputPath
    ], { stdio: ["ignore", "pipe", "ignore"] });
    return new Promise((resolve, reject) => {
      let stdoutData = '';
      pythonProcess.stdout.on('data', (data) => {
        stdoutData += data.toString();
        console.log(`Python stdout: ${data}`);
      });
      pythonProcess.on('close', async (code) => {
        try {
          if (code !== 0) {
            console.error(`Python process exited with code ${code}`);
            if (inputPath) await unlink(inputPath).catch(console.error);
            if (outputPath) await unlink(outputPath).catch(console.error);
            return resolve(NextResponse.json({
              error: 'Failed to process image'
            }, { status: 500 }));
          }
          if (!existsSync(outputPath!)) {
            throw new Error('Output file was not created by the model');
          }
          const processedImage = await readFile(outputPath!);
          const base64Image = processedImage.toString('base64');
          await Promise.all([
            unlink(inputPath!),
            unlink(outputPath!)
          ]).catch(console.error);
          resolve(NextResponse.json({
            processedImage: base64Image,
            success: true
          }));
        } catch (error) {
          console.error('Error in Python process:', error);
          if (inputPath) await unlink(inputPath).catch(console.error);
          if (outputPath) await unlink(outputPath).catch(console.error);

          resolve(NextResponse.json({
            error: 'Failed to process image',
            details: error instanceof Error ? error.message : 'Unknown error'
          }, { status: 500 }));
        }
      });
      pythonProcess.on('error', (error) => {
        console.error('Python process error:', error);
        if (inputPath) unlink(inputPath).catch(console.error);
        if (outputPath) unlink(outputPath).catch(console.error);
        resolve(NextResponse.json({
          error: 'Failed to process image',
          details: error.message
        }, { status: 500 }));
      });
    });
  } catch (error) {
    console.error('Unexpected error:', error);
    if (inputPath) await unlink(inputPath).catch(console.error);
    if (outputPath) await unlink(outputPath).catch(console.error);
    return NextResponse.json({
      error: 'Failed to process image',
      details: error instanceof Error ? error.message : 'Unknown error'
    }, { status: 500 });
  }
}
```

api/process/route.ts: This route handles mock image processing logic or pre-validation responses. It also Returns sample output, image quality info, and simulated processing metrics and is primarily used for testing and frontend development stubs.

```
import { NextResponse } from 'next/server';
export async function POST(req: Request) {
  try {
    const formData = await req.formData();
    const image = formData.get('image') as File;
    if (!image) {
      return NextResponse.json(
        { error: 'No image provided' },
        { status: 400 }
      );
    }
    const bytes = await image.arrayBuffer();
    const base64 = Buffer.from(bytes).toString('base64');
    const dataUrl = `data:${image.type};base64,${base64}`;
    return NextResponse.json({
      success: true,
      processedImage: dataUrl,
      metrics: {
        inferenceTime: 100,
        quality: 'high'
      }
    });
  } catch (error) {
    console.error('Error processing image:', error);
    return NextResponse.json(
      { error: 'Failed to process image' },
      { status: 500 }
    );
  }
}
```

Form 3: Backend forms

In our system core python backend responsible for deblurring the input image using a saved model. It handles image preprocessing, model inference, and returns the enhanced result and is integrated into the system via the api/deblur route.

```
import sys
import os
import numpy as np
import cv2
import tensorflow as tf
load_model = tf.keras.models.load_model
Layer = tf.keras.layers.Layer
Lambda = tf.keras.layers.Lambda
K = tf.keras.backend
get_custom_objects = tf.keras.utils.get_custom_objects
import keras
keras.config.enable_unsafe_deserialization()
IMG_HEIGHT = 256
IMG_WIDTH = 256
AUTOTUNE = tf.data.AUTOTUNE
@tf.keras.utils.register_keras_serializable()
```

```
class ReflectionPadding2D(tf.keras.layers.Layer):
    def __init__(self, padding=(1, 1), **kwargs):
        self.padding = tuple(padding)
        self.input_spec = [tf.keras.layers.InputSpec(ndim=4)]
        super(ReflectionPadding2D, self).__init__(**kwargs)
    def compute_output_shape(self, s):
        return (s[0], s[1] + 2 * self.padding[0], s[2] + 2 * self.padding[1], s[3])
    def call(self, x, mask=None):
        w_pad, h_pad = self.padding
        return tf.pad(x, [[0, 0], [h_pad, h_pad], [w_pad, w_pad], [0, 0]], 'REFLECT')
    def get_config(self):
        config = {'padding': self.padding}
        base_config = super(ReflectionPadding2D, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
get_custom_objects().update({"ReflectionPadding2D": ReflectionPadding2D})
def perceptual_loss(y_true, y_pred):
    return K.mean(K.square(y_pred - y_true))
def wasserstein_loss(y_true, y_pred):
    return K.mean(y_true * y_pred)
def edge_loss(y_true, y_pred):
    return K.mean(K.abs(y_pred - y_true))
def generator_loss(y_true, y_pred):
    return K.mean(K.square(y_pred - y_true))
def discriminator_loss(y_true, y_pred):
    return K.mean(y_true * y_pred)
def gradient_penalty(y_true, y_pred):
    return K.mean(K.square(y_pred - y_true))
def custom_lambda(x):
    return x / 2
@tf.function
def load_and_preprocess_image(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=3)
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH],
method=tf.image.ResizeMethod.AREA)
    image = (image / 127.5) - 1
    return image
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError(f"Could not read image from {image_path}")
    original_size = img.shape[:2]
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    ds = tf.data.Dataset.from_tensor_slices([image_path])
    ds = ds.map(load_and_preprocess_image, num_parallel_calls=AUTOTUNE)
    ds = ds.batch(1).prefetch(AUTOTUNE)
    for img_tensor in ds:
        return img_tensor, original_size
def postprocess_image(img, original_size=None):
    if tf.is_tensor(img):
        img = img.numpy()
    img = np.squeeze(img)
    img = (img + 1) * 127.5
    img = np.clip(img, 0, 255).astype(np.uint8)
    if original_size is not None:
        img = tf.image.resize(
            tf.convert_to_tensor(img)[tf.newaxis, ...],
            [original_size[0], original_size[1]],
            method=tf.image.ResizeMethod.LANCZOS3
        )[0].numpy()
    img = cv2.cvtColor(img.astype(np.uint8), cv2.COLOR_RGB2BGR)
    return img
def main():
    if len(sys.argv) != 3:
        print("Usage: python deblur.py <input_path> <output_path>")
        sys.exit(1)
    input_path = sys.argv[1]
    output_path = sys.argv[2]
    try:
        model_path = os.path.join(os.path.dirname(__file__), 'generator_model.h5')

        print(f"Loading model from: {model_path}")
        print(f"Model file exists: {os.path.exists(model_path)}")
        custom_objects = {
            "ReflectionPadding2D": ReflectionPadding2D,
            "perceptual_loss": perceptual_loss,
            "wasserstein_loss": wasserstein_loss,
            "edge_loss": edge_loss,
            "generator_loss": generator_loss,
            "discriminator_loss": discriminator_loss,
            "gradient_penalty": gradient_penalty,
            "Lambda": Lambda,
            "custom_lambda": custom_lambda
        }
        model = load_model(model_path, custom_objects=custom_objects,
compile=False)
        print("Model loaded successfully")
        input_img, original_size = preprocess_image(input_path)
        output_img = model(input_img, training=False)
        final_img = postprocess_image(output_img[0], original_size)
        cv2.imwrite(output_path, final_img)
    except Exception as e:
        print(f"Error: {str(e)}", file=sys.stderr)
        sys.exit(1)
if __name__ == "__main__":
    main()
```

5.3.2 Implementation of Model

This section describes the design and implementation of the core deep learning model used for image deblurring. It includes the implementation code of both the generator and discriminator.

ResNet Generator: Defines a ResNet-based generator architecture with reflection padding, instance normalization, and residual blocks. It upsamples encoded features into a deblurred image using transposed convolutions and skip connections.

```
import tensorflow as tf
class ReflectionPadding2D(tf.keras.layers.Layer):
    def __init__(self, padding=(1, 1), **kwargs):
        self.padding = tuple(padding)
        self.input_spec = [tf.keras.layers.InputSpec(ndim=4)]
        super(ReflectionPadding2D, self).__init__(**kwargs)
    def compute_output_shape(self, s):
        return (s[0], s[1] + 2 * self.padding[0], s[2] + 2 * self.padding[1], s[3])
    def call(self, x, mask=None):
        w_pad, h_pad = self.padding
        return tf.pad(x, [[0, 0], [h_pad, h_pad], [w_pad, w_pad], [0, 0]], 'REFLECT')
def res_net(input, filters, kernel_size, strides=(1, 1), apply_dropout=False):
    initializer = tf.random_normal_initializer(0., 0.02)
    x = ReflectionPadding2D(padding=(1,1))(input)
    x = tf.keras.layers.Conv2D(filters, kernel_size, strides=strides,
kernel_initializer=initializer)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    if apply_dropout:
        x = tf.keras.layers.Dropout(0.5)(x)
    x = ReflectionPadding2D(padding=(1,1))(x)
    x = tf.keras.layers.Conv2D(filters, kernel_size, strides=strides,
kernel_initializer=initializer)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    return tf.keras.layers.Add()([input, x])
def Generator():
    initializer = tf.random_normal_initializer(0., 0.02)
    inputs = tf.keras.layers.Input(shape=[256, 256, 3])
    x = ReflectionPadding2D(padding=(3,3))(inputs)
    x = tf.keras.layers.Conv2D(64, (7,7), padding='valid',
kernel_initializer=initializer)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Conv2D(128, (3,3), padding='same', strides=2,
kernel_initializer=initializer)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Conv2D(256, (3,3), padding='same', strides=2,
kernel_initializer=initializer)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    for _ in range(9):
        x = res_net(x, 256, (3,3), apply_dropout=True)
    x = tf.keras.layers.UpSampling2D(interpolation='bilinear')(x)
    x = tf.keras.layers.Conv2D(128, (3,3), padding='same',
kernel_initializer=initializer)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.UpSampling2D(interpolation='bilinear')(x)
    x = tf.keras.layers.Conv2D(64, (3,3), padding='same',
kernel_initializer=initializer)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = ReflectionPadding2D(padding=(3,3))(x)
    x = tf.keras.layers.Conv2D(3, (7,7), padding='valid',
kernel_initializer=initializer)(x)
    x = tf.keras.layers.Activation('tanh')(x)
    outputs = tf.keras.layers.Add()([x, inputs])
    outputs = tf.keras.layers.Lambda(lambda z: z/2)(outputs)
    return tf.keras.Model(inputs=inputs, outputs=outputs, name='Generator')
generator = Generator()
generator.summary()
```

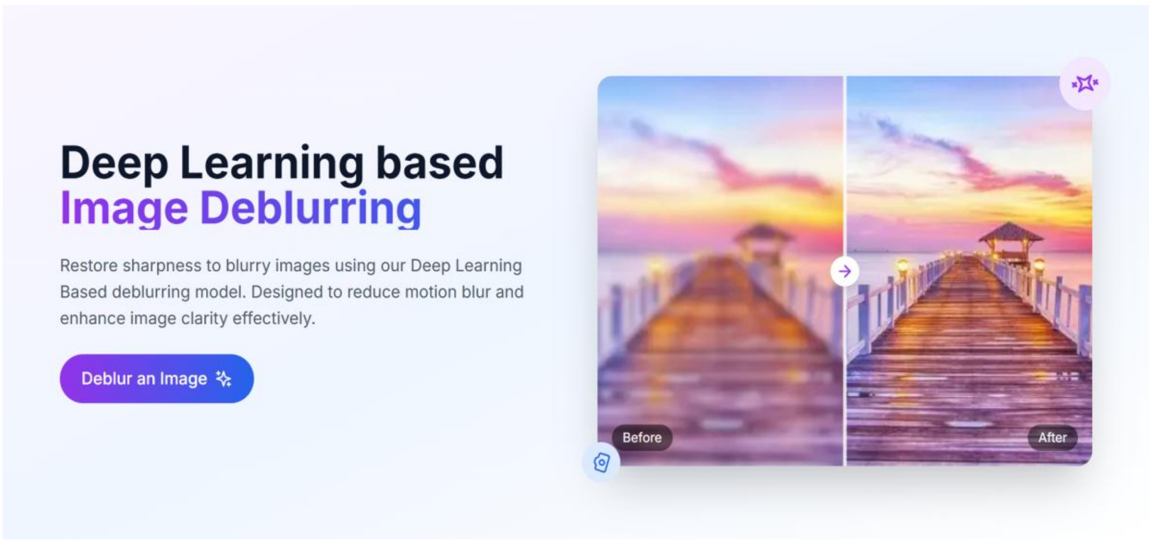
PatchGAN discriminator: Implements a PatchGAN-style discriminator that evaluates small patches of the image for realism. It distinguishes between real and generated (deblurred) images by comparing input-output image pairs.

```
def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)
    inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')
    tar = tf.keras.layers.Input(shape=[256, 256, 3], name='target_image')
    x = tf.keras.layers.concatenate([inp, tar])
    down1 = downsample(64, 4, False)(x)
    down2 = downsample(128, 4)(down1)
    down3 = downsample(256, 4)(down2)
    zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)
    conv = tf.keras.layers.Conv2D(512, 4, strides=1, kernel_initializer=initializer,
use_bias=False)(zero_pad1)
    batchnorm1 = tf.keras.layers.BatchNormalization()(conv)
    leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)
    zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)
    last = tf.keras.layers.Conv2D(1, 4, strides=1,
kernel_initializer=initializer)(zero_pad2)
    return tf.keras.Model(inputs=[inp, tar], outputs=last)
discriminator = Discriminator()
discriminator.summary()
```

5.3.3 Output Screens

The following screens illustrate the appearance of our website.

Screen 1 illustrates the Home page, which has a basic summary and redirects the user to the image upload page upon clicking "Deblur an Image" button.

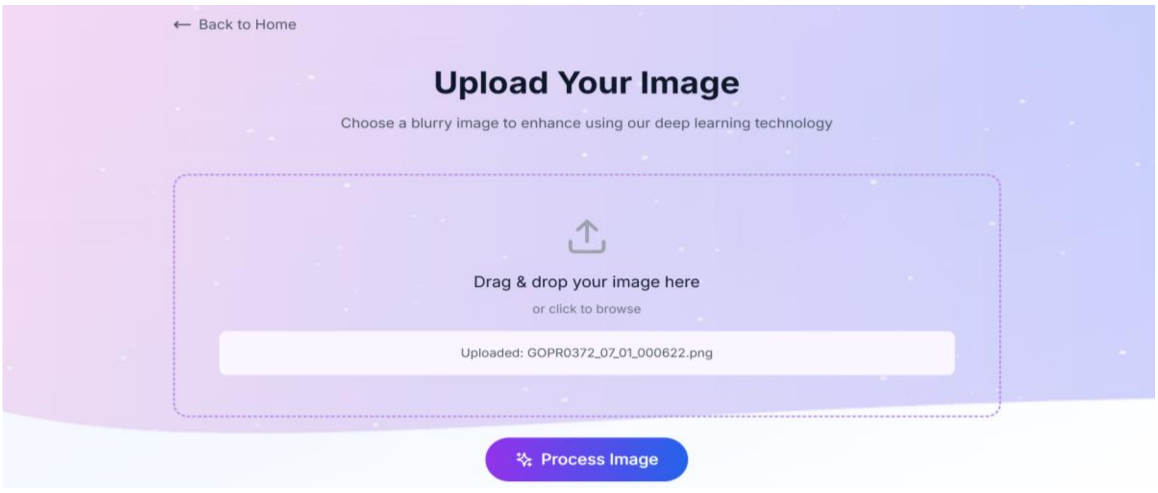


Deep Learning based Image Deblurring

Restore sharpness to blurry images using our Deep Learning Based deblurring model. Designed to reduce motion blur and enhance image clarity effectively.

Screen 1. Home Page

Screen 2 allows the user to upload a blurred image and redirects them to the processing page upon clicking the "Process Image" button, provided that a valid blurred image has been uploaded.



Upload Your Image

Choose a blurry image to enhance using our deep learning technology

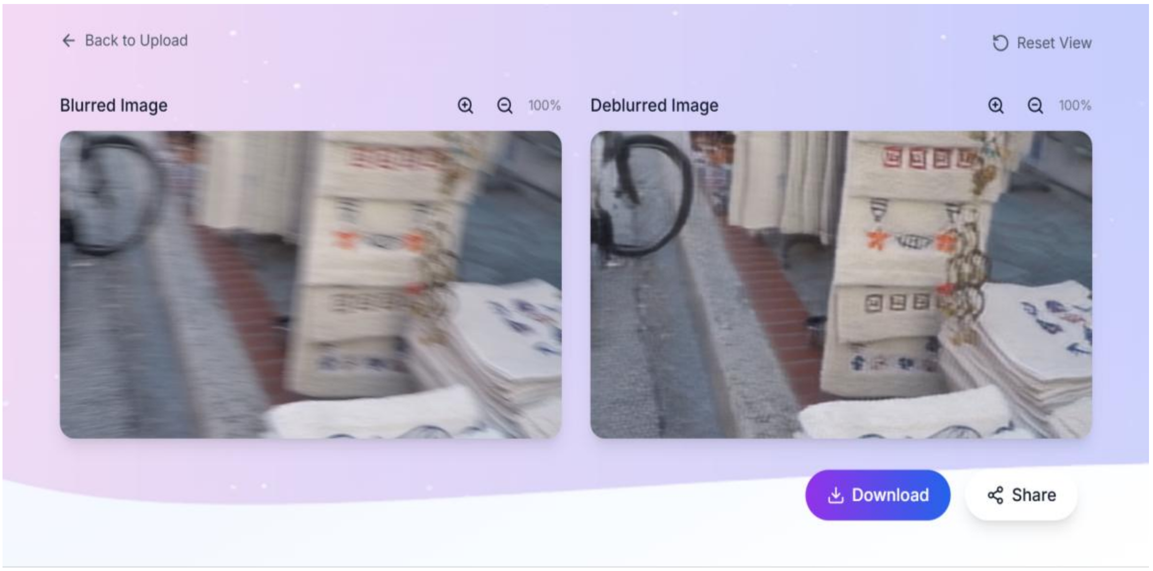
Drag & drop your image here

or click to browse

Uploaded: GOPR0372_07_01_000622.png

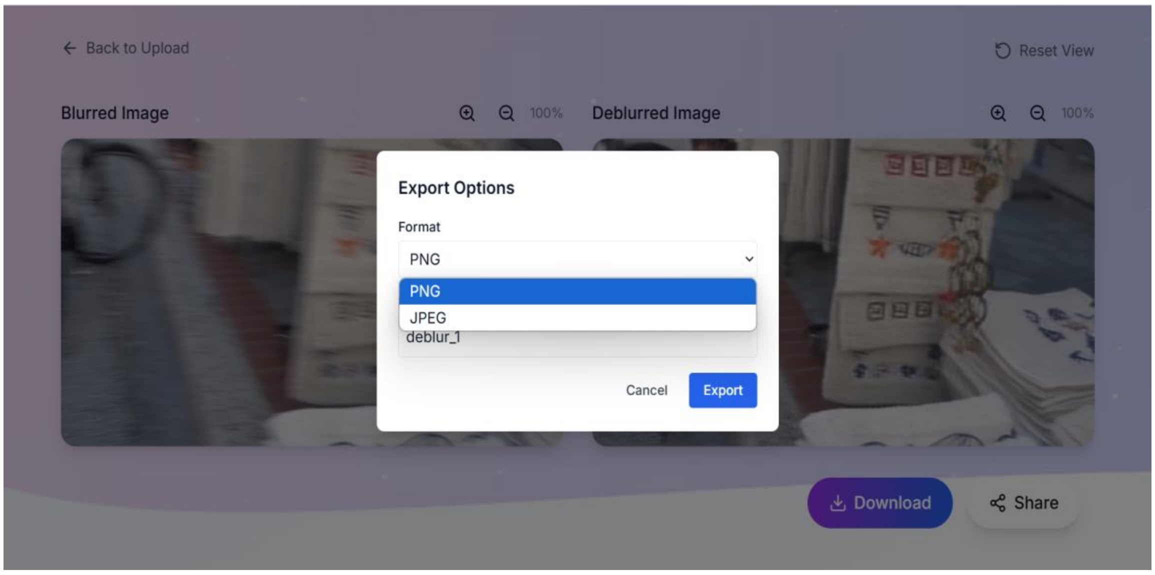
Screen 2. Image Upload Page

Screen 3 is the processing page where the actual deblurring process takes place and the result is displayed side by side for comparison. This also allows the user to download and share the image.



Screen 3. Processing Page – Displays Model Output

Screen 4 displays a pop-up after the user clicks the “Download” button. The user can name the image and choose to download it in either PNG or JPEG format based on their preference.



Export Options

Format

PNG

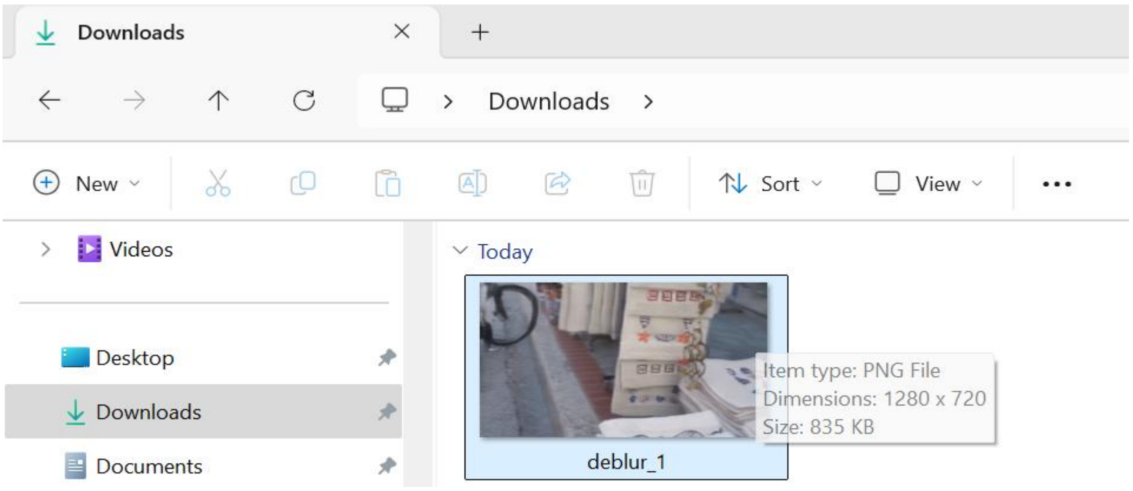
PNG
JPEG

deblur_1

Cancel Export

Screen 4. Image Download Using JPEG/PNG Format

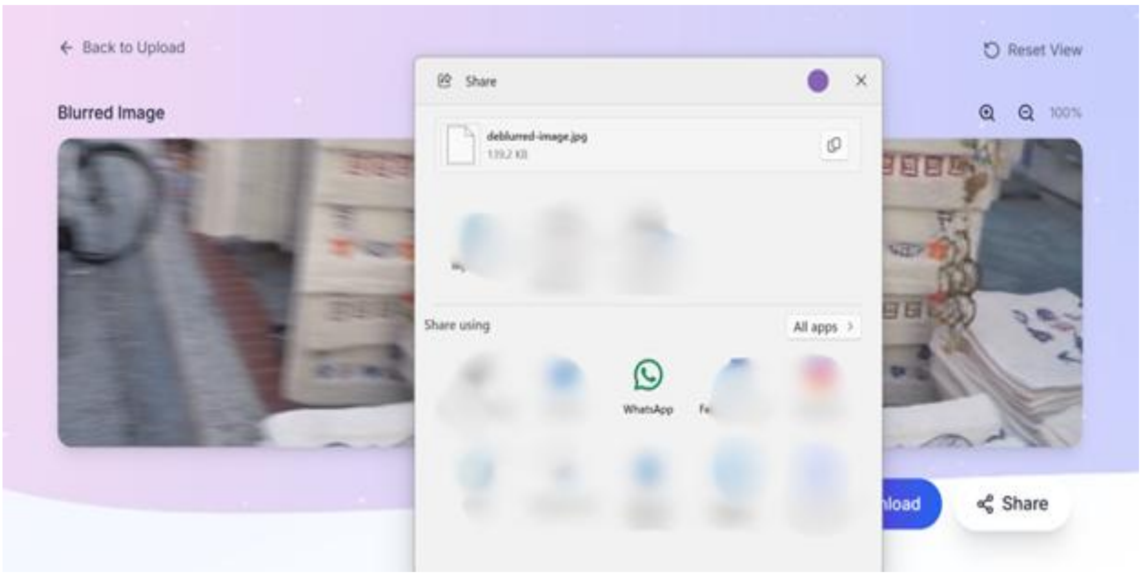
Screen 5 shows the successfully downloaded deblurred image in the device's downloads.



Item type: PNG File
Dimensions: 1280 x 720
Size: 835 KB

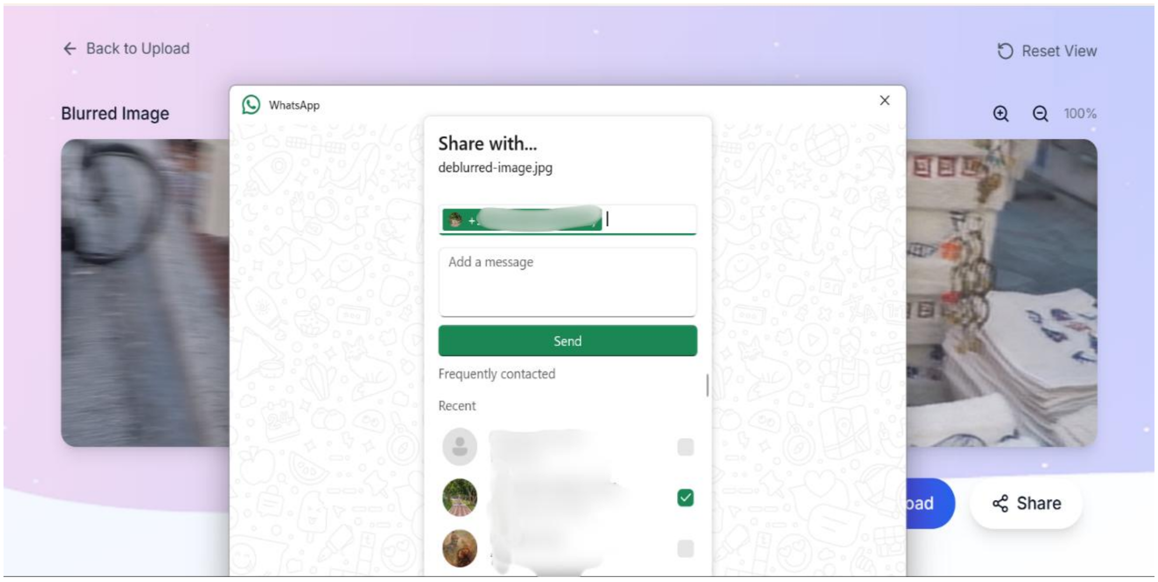
Screen 5. Download Preview

Screen 6 displays a pop-up after the user clicks the "Share" button. The user can share the image through WhatsApp.



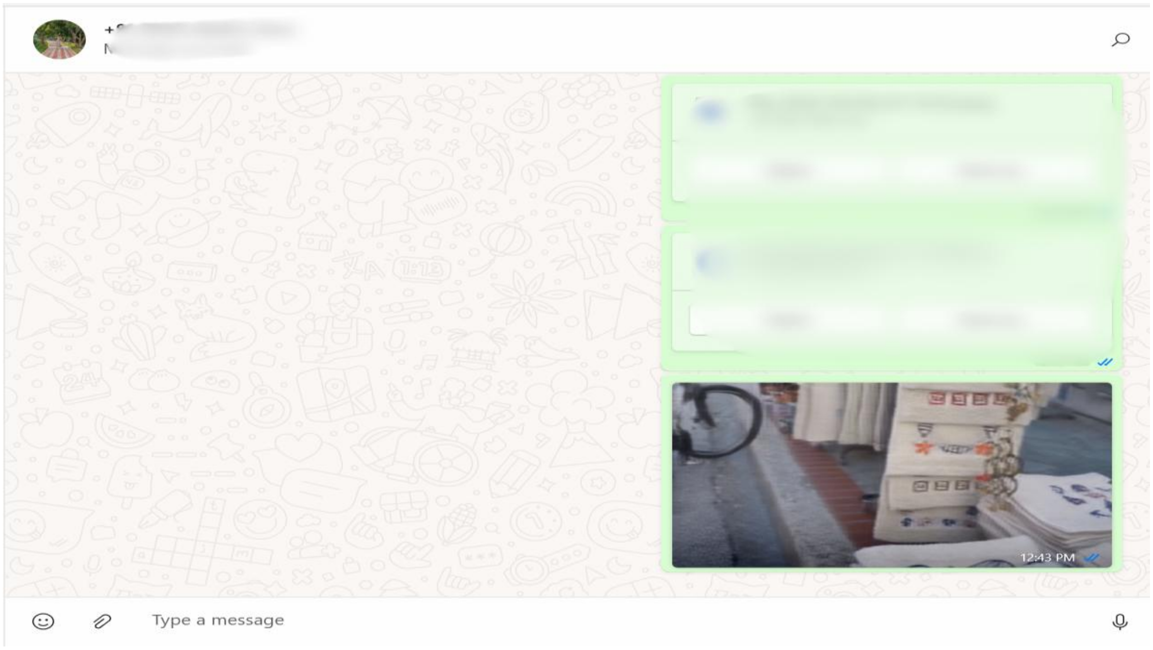
Screen 6. Image Sharing Using WhatsApp

Screen 7 displays the WhatsApp application interface, allowing the user to select a contact with whom they want to share the deblurred image.



Screen 7. Contact Selection for Image Sharing Through WhatsApp

Screen 8 confirms the successful sharing of the deblurred image via WhatsApp.



Screen 8. Preview of Shared Image Through WhatsApp

5.3.4 Result Analysis

To evaluate the performance of the image deblurring model, both quantitative and qualitative analyses were conducted. The analysis consists of two parts: results on the standard GoPro test set, and results on custom images not seen during training.

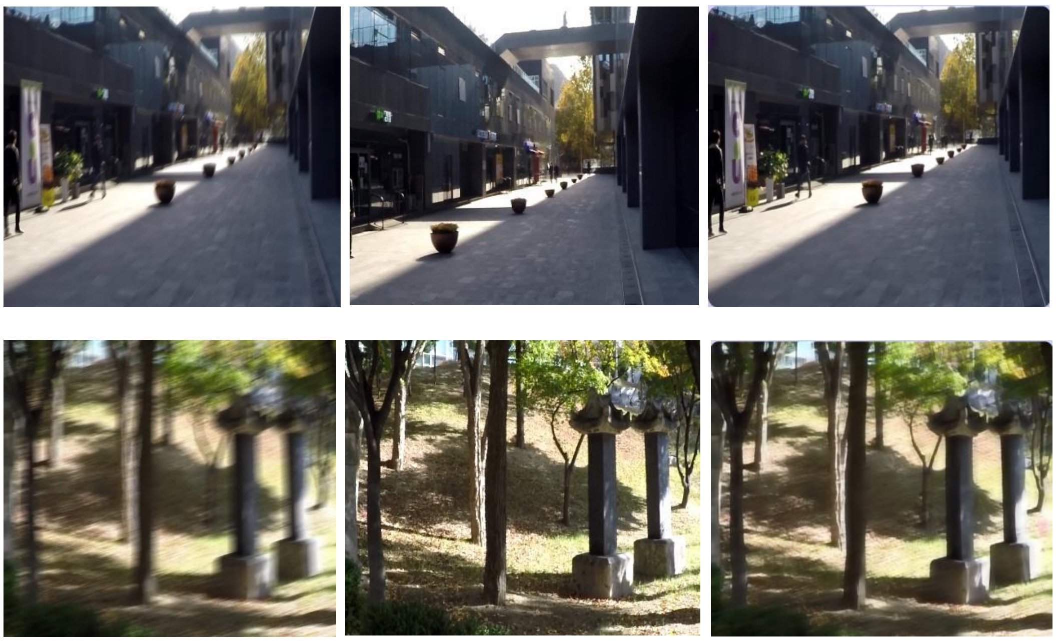
A. Visual Comparison on Test Set

To qualitatively assess the model's output, we present a side-by-side comparison of:

- The original blurred image from the GoPro test set
- Its corresponding ground truth sharp image
- The deblurred output predicted by the model

These comparisons demonstrate that the model is able to effectively recover sharpness and fine details. The predicted outputs are visually close to the ground truth in most cases, with clear improvements in edge sharpness and texture restoration.

Blur Images **Sharp Ground Truth** **Predicted Images**



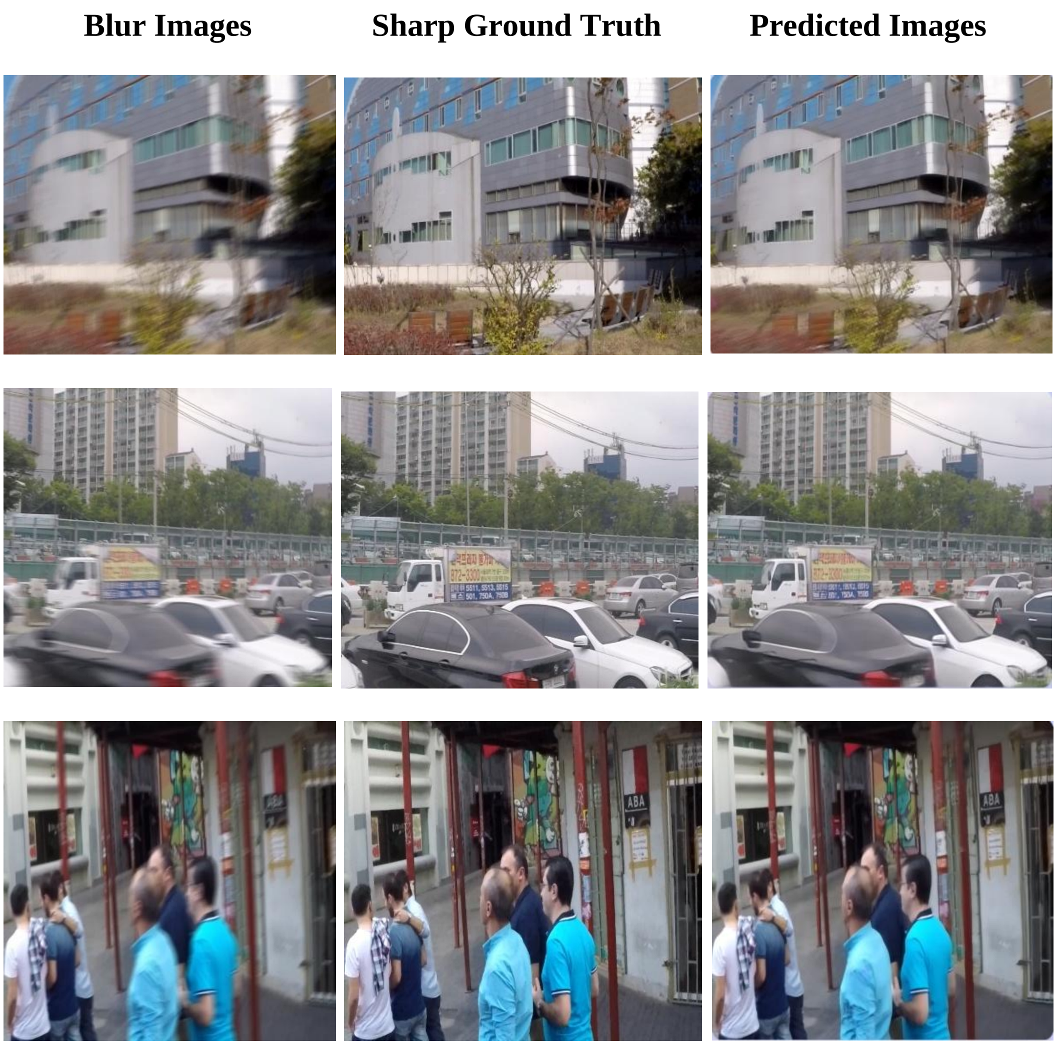


Figure 13. Visual Comparison on Test Set

B. Results on Custom Image

To test the model’s generalization ability, we also evaluated it on custom images not included in the dataset. These real-world blurred images were collected separately and fed into the trained model.

The results show that the model successfully generalizes to unseen data. It is able to reduce motion blur and enhance clarity, even in cases where lighting, texture, and blur patterns differ from the training data.

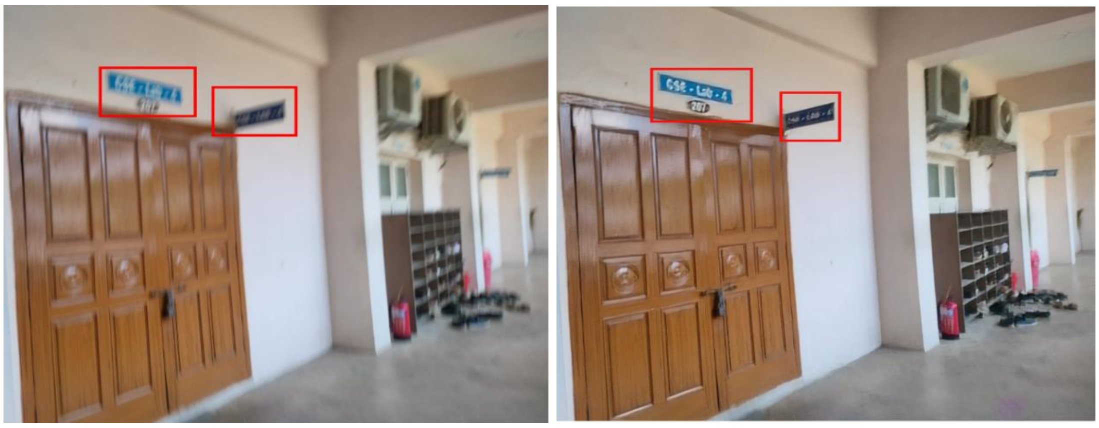


Figure 14. Result on New Image Which is Not in the Dataset

The above images show a comparison between a real-world blurred image and the corresponding deblurred output generated by the model. Noticeable improvements can be seen in the clarity of text and object boundaries. This demonstrates the model's effectiveness in handling custom, real-time blurred inputs.

C. Quantitative Evaluation

The PSNR and SSIM graphs below demonstrate the quantitative improvement in image quality across epochs during training. A steady rise in PSNR (Peak Signal-to-Noise Ratio) indicates that the model progressively enhances image clarity, while the increase in SSIM (Structural Similarity Index) confirms better preservation of structural information. These metrics validate the effectiveness of the ResNet-PatchGAN architecture in restoring sharpness and structural details.

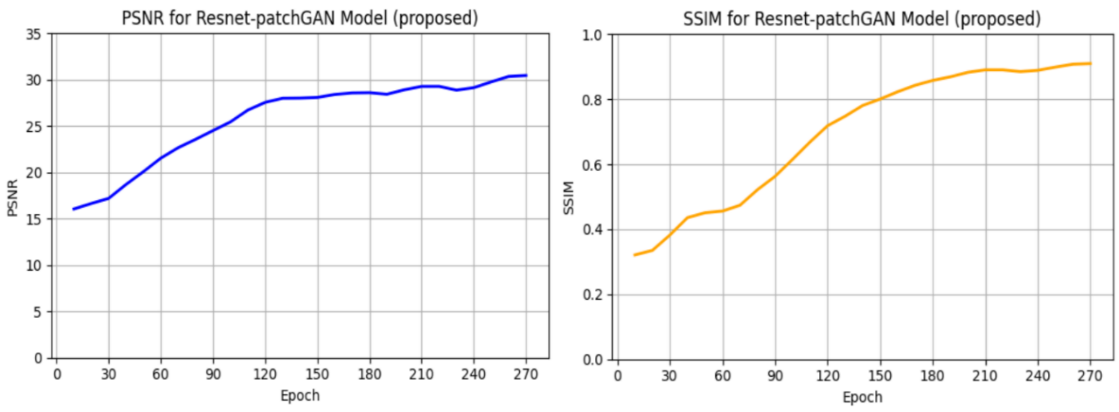


Figure 15. Plots on PSNR and SSIM Over Epochs for Proposed Model

D. Model Comparison with Existing Models

To thoroughly evaluate the performance of our proposed model (Resnet-PatchGAN), we compared it with two existing models: DeblurGAN and DeepDeblur-PyTorch. The comparison was done using two standard quantitative metrics: PSNR and SSIM.

Table 9. PSNR and SSIM Comparison with Existing Models

Deblurring Model	PSNR	SSIM
DeblurGAN	23.6	0.884
DeepDeblur-PyTorch	30.40	0.9018
Resnet-patchGAN (Proposed)	30.44	0.9094

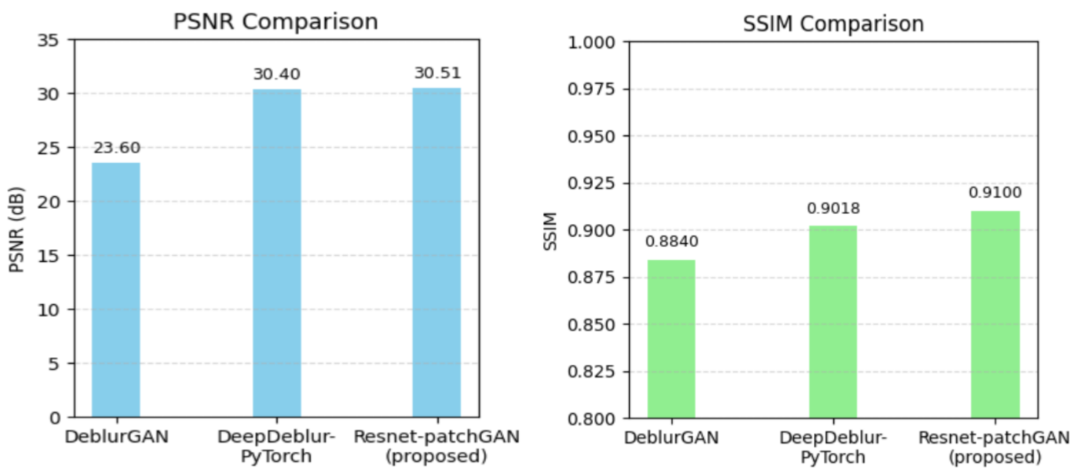


Figure 16. PSNR and SSIM Comparison Bar Plots

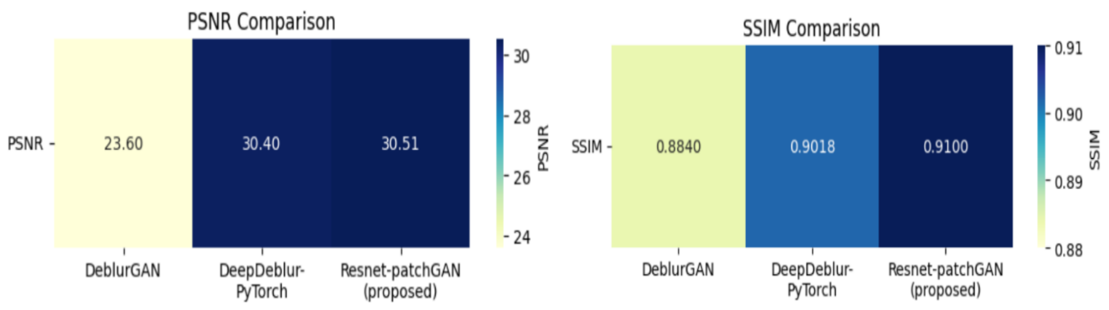


Figure 17. PSNR and SSIM Comparison Heatmaps

The above figure 17 shows the heatmaps which reinforce the quantitative data, where darker shades correspond to higher performance. The gradual color shift from DeblurGAN to our method clearly visualizes the incremental improvements.

5.4 CONCLUSION

In this chapter, the complete implementation pipeline of the proposed image deblurring model was presented, starting from architectural design to real-time performance analysis. The ResNet-PatchGAN model was successfully trained using a combination of perceptual and adversarial loss functions, showing consistent improvement across epochs in both PSNR and SSIM metrics. Our model has obtained the PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index Measure) of 30.51 db and 0.91 respectively. The visual results showcased clear enhancements in sharpness and readability for both test set and customized blurred images. Overall, the implementation effectively demonstrates the capability of the model to deblur images with high accuracy and structural consistency, validating its practical usability in real-world applications.

6. TESTING AND VALIDATION

6.1 INTRODUCTION

Testing is an important step to verify if the image deblurring model meets the expected requirements and produces clear, high-quality images. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.

Testing is important because ensures the model is free of errors, performs accurately, and functionally effective across different environments. Testing the model helps identify and fix issues early, ensuring the final product is reliable and performs optimally. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effective and customer satisfaction.

In this project, we are using the Gopro dataset which contains the pairs of blurry and sharp images. This dataset undergoes preprocessing before being passed to the model. The testing process involves measuring image quality, reducing blur, and preserving fine details. Preconditions are the environmental versions like it works good on Visual Studio Code, Python of version of 3.11.11. Most importantly, Google Colab supportive workspace in order to run the codes with GPU and avoid the compatibility issues, it needs to also support the latest version of Node.js to integrate the model with frontend.

6.2 DESIGN OF TEST CASES AND SCENARIO

The test cases are designed for our model as below in Table 10. The test cases are formatted so that the user can enter values in the proper format.

Table 10. Design of Test Cases and Scenarios

Test ID	**Test Scenario**	**Test Case**	**Pre-Condition**	**Test Steps**	**Expected Result**	**Post Condition**	**Actual Result**	**Status (Pass/Fail)**
TC_1	Verify if the system accepts a valid blurry image	Upload a valid blurry image	Image should be in JPG, JPEG, or PNG format	1. Click on "Deblur an image" button 2. Upload a blurry image. 3. Then click on “process image” to begin deblurring.	Deblurred image should be displayed on the UI	Image is successfully processed	Deblurred image is displayed with improved clarity	Pass
TC_2	Verify if the system rejects non-image files	Upload an invalid file (e.g., .txt, .pdf)	File should not be an image	1. Click on "Deblur an image" button 2. Upload a non-image file.	A pop-up message should appear saying "Only PNG and JPEG/JPG images are allowed."	User is unable to upload non-image files	A pop-up message appears: "Only PNG and JPEG/JPG images are allowed."	Pass

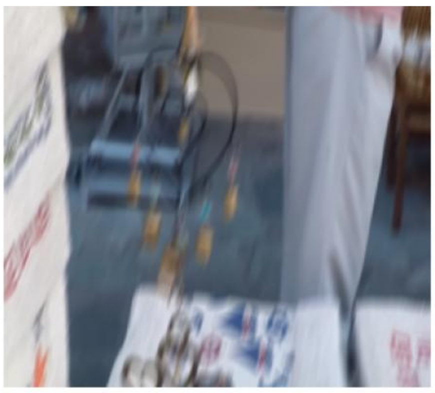
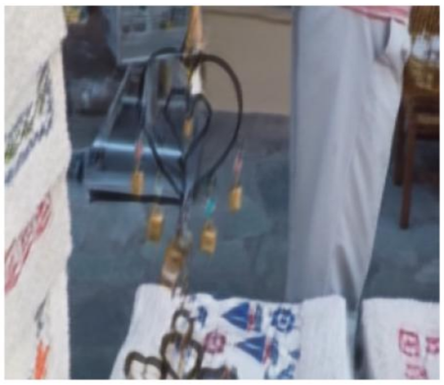
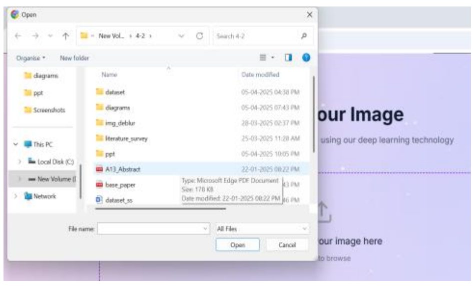
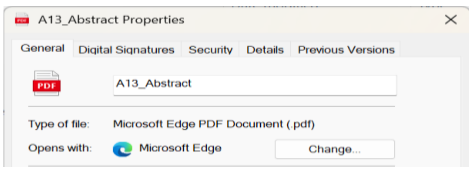
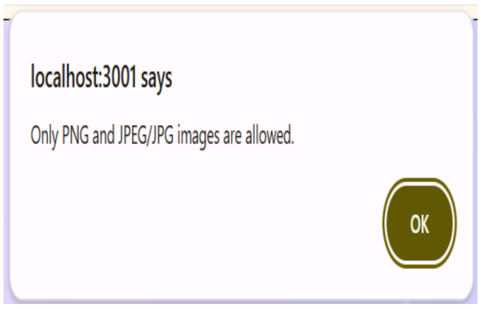
Test ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Expected Result	Post Condition	Actual Result	Status (Pass/Fail)
TC_3	Verify size of the image uploaded	Upload a 50 MB image or less	The system Should allow only images 50MB or less	1. Click on "Deblur an image" button 2. Upload a 50MB Image. 3. click on “process image” button.	Deblurred image should be displayed on the UI	Image is successfully processed	Deblurred image is displayed with improved clarity	Pass
TC_4	Verify size of the image uploaded	Upload a 50 MB image or more	Image should exceed the maximum file size allowed	1. Click on "Deblur an image" button. 2. Upload a 50MB Image.	A pop-up message should appear saying "Image size should not exceed 50MB."	The system prevents oversized images from being uploaded and does not process them.	A pop-up message appears: "Image size should not exceed 50MB."	Pass
TC_5	Verify Output Download Feature	Download the deblurred image	Successfully deblurred image available	1.Deblur the image 2. Click on "Download" button. 3. Select the image format and click export.	System allows users to download the processed image	Deblurred image is downloaded successfully	Image successfully downloaded in original format	Pass

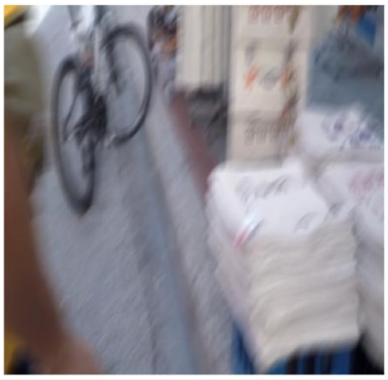
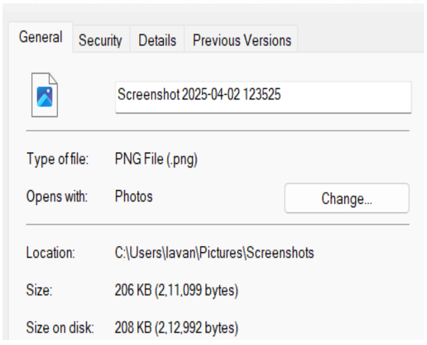
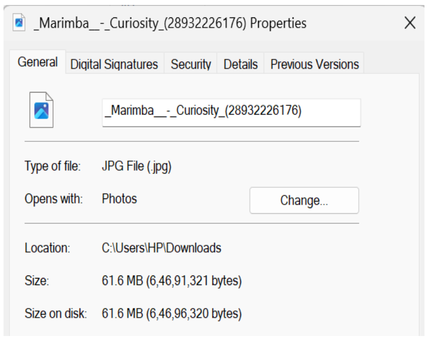
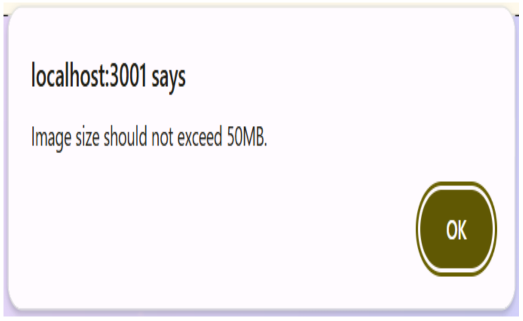
Test ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Expected Result	Post Condition	Actual Result	Status (Pass/Fail)
TC_6	Verify Image Sharing	Sharing the image	The device must have WhatsApp logged in and image file should be available.	1. Click on "Share" Button. 2. Choose WhatsApp 3. Confirm sharing.	The selected image should be successfully sent to the chosen contact or group on WhatsApp.	Image appears on the user's social media	Image successfully shared on social media	Pass
TC_7	Verify system behavior when an empty image is uploaded.	Upload an image with no objects to deblur (e.g., plain background, solid color, or noise)	Image should be in JPG, JPEG, or PNG format	1. Click on "Deblur an image" button 2. Select an empty image with no objects.	A pop-up message should appear: "The image appears to be empty or lacks meaningful content. Please upload a valid image."	The system does not process the image and remains on the upload screen	A pop-up message appears "The image appears to be empty or lacks meaningful content. Please upload a valid image."	Pass

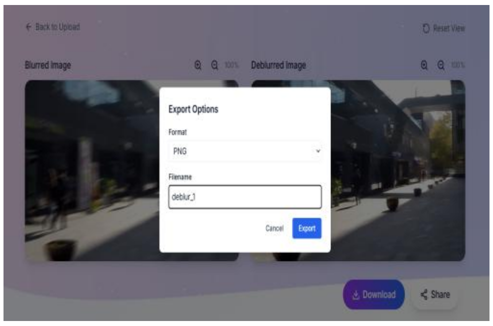
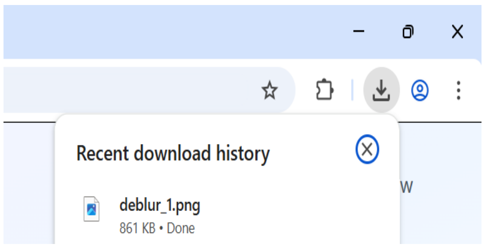
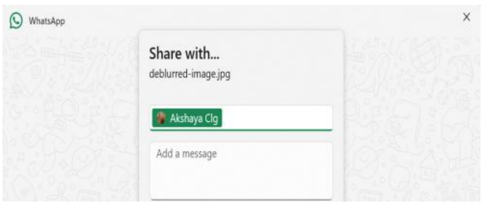
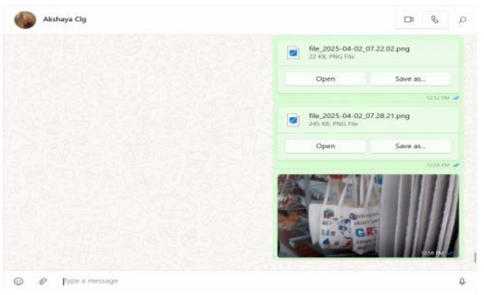
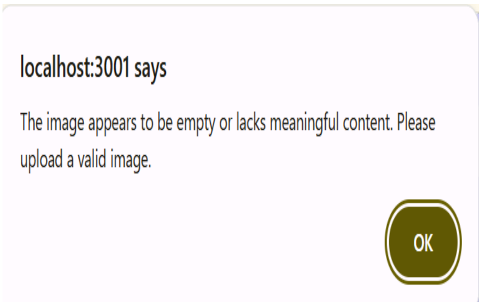
6.3 VALIDATION TESTING

It is testing where testers perform the functional and non-functional testing. Functional testing includes Unit Testing, Integration Testing and System Testing while the Non-functional testing includes User acceptance testing. The Table 11 demonstrates Validation testing performed in our project.

Table 11. Design of Test Cases for Validation Testing

Test Case	**Uploaded Image**	**Result**
Testing a valid blurry image		
Testing on invalid file (e.g. .txt, .pdf)	 Open New Vol... › 4-2 › Search 4-2 Organise ▾ New folder diagrams ppt Screenshots This PC Local Disk (C:) New Volume (Network Name Date modified dataset 05-04-2025 04:38 PM diagrams 05-04-2025 07:43 PM img_deblur 28-03-2025 02:37 PM literature_survey 25-03-2025 11:28 AM ppt 05-04-2025 10:05 PM A13_Abstract 22-01-2025 08:22 PM base_paper dataset_ss Type: Microsoft Edge PDF Document Size: 178 KB Date modified: 22-01-2025 08:22 PM File name: All Files Open Cancel our Image using our deep learning technology our image here to browse  A13_Abstract Properties General · Digital Signatures · Security · Details · Previous Versions A13_Abstract Type of file: Microsoft Edge PDF Document (.pdf) Opens with: Microsoft Edge · Change...	 localhost:3001 says Only PNG and JPEG/JPG images are allowed. OK

Test Case	**Uploaded Image**	**Result**
Testing on image size 50MB or less.	  General · Security · Details · Previous Versions Screenshot 2025-04-02 123525 Type of file: PNG File (.png) Opens with: Photos · Change... Location: C:\Users\lavan\Pictures\Screenshots Size: 206 KB (2,11,099 bytes) Size on disk: 208 KB (2,12,992 bytes)	
Testing on image size more than 50MB.	 Curiosity At "Marimba" Sols 1417 - 1427 NASA / JPL / MSSS / Justin Cowart  _Marimba_-_Curiosity_(28932226176) Properties General · Digital Signatures · Security · Details · Previous Versions _Marimba_-_Curiosity_(28932226176) Type of file: JPG File (.jpg) Opens with: Photos · Change... Location: C:\Users\HP\Downloads Size: 61.6 MB (6,46,91,321 bytes) Size on disk: 61.6 MB (6,46,96,320 bytes)	 localhost:3001 says Image size should not exceed 50MB. OK

Test Case	Uploaded Image	Result
Testing whether the image can be downloaded.		 Back to Upload Reset View Blurred Image 100% Deblurred Image 100% Export Options Format PNG Filename deblur_1 Cancel Export Download Share  Recent download history deblur_1.png 861 KB • Done
Testing image sharing option.		 WhatsApp Share with... deblurred-image.jpg Akshaya Clg Add a message  Akshaya Clg file_2025-04-02_07.22.02.png 22 KB, PNG File Open Save as... file_2025-04-02_07.28.21.png 245 KB, PNG File Open Save as... Type a message
Testing on Empty (no object) image		 localhost:3001 says The image appears to be empty or lacks meaningful content. Please upload a valid image. OK

6.4 CONCLUSION

This chapter validates our image deblurring model through testing, confirming its effectiveness in restoring clear and high-quality images and also emphasizes the validation of the system using different testing approaches like checking for the size, format and if the image is empty without any objects.

7. CONCLUSION

In this project, we focused on solving the problem of blurry images using deep learning. Blurry images can make it hard to understand visual content, especially in areas like security, photography, and healthcare. So, we built a model that can take a blurry image and turn it into a much clearer version. To do this, we used Deep Learning techniques called Convolutional Neural Network (CNNs) and Generative Adversarial Networks (GANs). Our model had two main parts: a generator (ResNet-based), which tried to make sharp images from blurry ones, and a discriminator (PatchGAN), which checked how realistic the generated images were. The generator uses ResNet blocks to help it learn fine image details, while the discriminator looks at small patches of the image to decide if they're real or fake. We trained our model on Gopro dataset that had pairs of blurry and sharp images. Over time, the model learned how to remove the blur and restore important image features like edges and textures. The results showed clear improvements - our deblurred images looked much closer to the original sharp images. Overall, this project showed that CNN-GAN based methods are very effective for image deblurring. The technique could be useful in many real-world situations where image quality matters.

In the future, this image deblurring system can be improved to offer more powerful and user-friendly features. One major advancement would be extending the system to support real-time video deblurring. This would allow the system to fix blurry frames as a video play, making it especially valuable for applications such as live streaming, virtual meetings, and security camera footage, where maintaining visual clarity in motion is essential. Another promising improvement is the introduction of an Auto-Enhance Mode, which would not only deblur images but also automatically adjust brightness, contrast, and sharpness in a single step. This all-in-one enhancement feature would simplify the process for users, offering quick and polished results with minimal effort. Additionally, the system could be upgraded to support batch image deblurring, allowing users to upload and process multiple images at once instead of one by one. This would save significant time and effort, particularly for photographers, designers, or anyone dealing with large sets of images.

8. REFERENCES

[1] Yan Q, Gong D, Zhang Y, Wang P, Zhang Z, and Shi J Q, 2023, "SharpFormer: Learning Local Feature Preserving Global Representations for Image Deblurring," IEEE Trans. Image Process., vol. 32, pp. 2857-2870. DOI: 10.1109/TIP.2023.3251029.

[2] Yae S and Ikehara M, 2023, "Inverted Residual Fourier Transformation for Lightweight Single Image Deblurring," IEEE Access, vol. 112, pp. 29175-29182. DOI: 10.1109/ACCESS.2023.3243173.

[3] H. S. Lee and S. I. Cho, 2023, "Locally Adaptive Channel Attention-Based Spatial–Spectral Neural Network for Image Deblurring," IEEE Trans. Circuits Syst. Video Technol., vol. 33, no. 10, pp. 5375–5390. DOI: 10.1109/TCSVT.2023.3250509.

[4] Y. Quan, P. Lin, Y. Xu, Y. Nan, and H. Ji, 2022, "Nonblind Image Deblurring via Deep Learning in Complex Field," IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 10, pp. 5387–5400. DOI: 10.1109/TNNLS.2021.3070596.

[5] Q. Zhao, H. Yang, D. Zhou, and J. Cao, 2022, "Rethinking Image Deblurring via CNN-Transformer Multiscale Hybrid Architecture," IEEE Trans. Instrum. Meas., vol. 72, Art. no. 5002415. DOI: 10.1109/TIM.2022.3230482.

[6] A. Esmaeilzehi, M. O. Ahmad, and M. N. S. Swamy, 2021, "UPDResNN: A Deep Light-Weight Image Upsampling and Deblurring Residual Neural Network," IEEE Trans. Broadcast., vol. 67, no. 2, pp. 538–548. DOI: 10.1109/TBC.2021.3068862.

[7] Y. Wu, P. Qian, and X. Zhang, 2021, "Two-Level Wavelet-Based Convolutional Neural Network for Image Deblurring," IEEE Access, vol. 9, pp. 45853–45863. DOI: 10.1109/ACCESS.2021.3067055.

[8] Wu, T., Li, W., Jia, S., Dong, Y., & Zeng, T., 2020, "Deep Multi-level Wavelet-CNN Denoiser Prior for Restoring Blurred Image with Cauchy Noise," IEEE Signal Process. Lett., vol. 27, pp. 1635–1639. DOI: 10.1109/LSP.2020.3023299.

[9] F. Albluwi, V. A. Krylov, and R. Dahyot, 2018, "Image Deblurring and Super-Resolution Using Deep Convolutional Neural Networks," Proc. IEEE 28th Int. Workshop Mach. Learn. Signal Process. (MLSP), Aalborg, Denmark. DOI: 10.1109/MLSP.2018.8516983.

[10] C. Min, G. Wen, B. Li, and F. Fan, 2018, "Blind Deblurring via a Novel Recursive Deep CNN Improved by Wavelet Transform," IEEE Access, vol. 6, pp. 69242–69252. DOI: 10.1109/ACCESS.2018.2880279.

[11] Y. Song, J. Li, X. Wang, and X. Chen, 2018, "Single Image Dehazing Using Ranking Convolutional Neural Network," IEEE Trans. Multimedia, vol. 20, no. 6, pp. 1548–1560. DOI: 10.1109/TMM.2017.2771472.

[12] J. Zhang, J. Pan, J. Ren, Y. Song, L. Bao, and R. W. H. Lau, 2018, "Dynamic Scene Deblurring Using Spatially Variant Recurrent Neural Networks," Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. DOI: 10.1109/CVPR.2018.00267.

[13] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas, 2018, “DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks,”Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. DOI: 10.1109/CVPR.2018.00854.

[14] S. Nah, T. H. Kim, and K. M. Lee, 2017, "Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring," Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), pp. 3883–3891. DOI: 10.1109/CVPR.2017.35.

[15] K. Zhang, W. Zuo, S. Gu, and L. Zhang, 2017, "Learning Deep CNN Denoiser Prior for Image Restoration," IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR). DOI: 10.1109/CVPR.2017.300.