Indian Institute of Technology Jodhpur

Course Name: Computer Vision (CSL7360)

Project Topic: Comparative Assessment of Resolution Enhancement Models for Satellite Images

Submitted by:

Akaash Chatterjee
Roll No: M24CSE002

Alok Dutta
Roll No: M24CSE032

Vishwanath Singh
Roll No: M24CSE030

Prathmesh Chintamani Gosavi
Roll No: M24CSA023

April 9, 2025

Google Colab Link

Contents

1 Introduction **2**
1.1 Introduction . . . 2
1.2 Objective . . . 2

2 Methodology **3**
2.1 Data Aquisition and Pre-processing . . . 3
2.1.1 Dataset Selection . . . 3
2.1.2 Data Preprocessing Pipeline . . . 3
2.2 Enhancement Approaches . . . 6
2.2.1 Traditional Interpolation Methods . . . 6
2.2.2 Deep Learning-Based Super-Resolution . . . 7
2.2.3 Traditional Enhancement Techniques . . . 8
2.2.4 Novel Hybrid Methods . . . 9
2.3 Evaluation Framework . . . 10
2.3.1 Image Quality Metrics . . . 10
2.3.2 Performance Analysis . . . 11
2.3.3 Visualization . . . 11
2.3.4 Implementation Details . . . 12

3 Results and Observations **13**
3.1 Tabular Comparision of Metrics . . . 13
3.2 Metrics Visualization . . . 14
3.3 Testing Results . . . 15

4 Important Links **16**

5 Website Screenshots **17**

6 Conclusion and Summary **19**

7 Team Member Contributions **20**

Chapter 1

Introduction

1.1 Introduction

High-resolution satellite imagery plays a critical role in numerous remote sensing applications, including urban planning, environmental monitoring, agriculture, and disaster management. However, the availability of high-resolution images is often limited due to technical constraints, cost, or atmospheric conditions. As a result, enhancing the spatial resolution of low-resolution satellite images has become a vital area of research in image processing and computer vision.

Resolution enhancement, also referred to as image super-resolution, aims to reconstruct high-quality images from their low-resolution counterparts. Over the years, a variety of methods have been developed to address this challenge, ranging from traditional interpolation techniques such as bicubic or Lanczos interpolation, to advanced deep learning models including convolutional neural networks (CNNs), generative adversarial networks (GANs), and transformer-based architectures.

1.2 Objective

The primary objective of this research is to develop and evaluate various image enhancement techniques for satellite imagery using the UC Merced Land Use dataset. The project aims to:

- Compare traditional interpolation methods, deep learning-based super-resolution, and various image enhancement techniques for improving visual quality.
- Develop novel hybrid approaches that combine super-resolution with traditional enhancement methods.
- Quantitatively evaluate each technique using standard image quality metrics such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), and Mean Squared Error (MSE).
- Analyze the computational efficiency and processing time of different enhancement methods.
- Create a user-friendly Streamlit web application to demonstrate and visualize the enhancement techniques, and deploy it for public or academic access.

Chapter 2

Methodology

This section describes the methodology followed for data preprocessing, vocabulary creation, and embedding initialization in the neural machine translation system.

2.1 Data Aquisition and Pre-processing

2.1.1 Dataset Selection

The UC Merced Land Use dataset is used for this study. It consists of 21 land-use classes with 100 images per class. For consistent testing and to manage memory usage, the following approach was adopted:

- Utilized the **UC Merced Land Use dataset** containing 21 land-use classes, each with 100 high-resolution images.
- Selected **10 specific categories** for consistent and balanced evaluation:
  - Agricultural
  - Baseball diamond
  - Beach
  - Buildings
  - Forest
  - Airplane
  - Freeway
  - Golf course
  - Harbor
  - Mobile home park
- **Limited the dataset to 1000 samples** (100 images per selected category) to manage computational and memory resources effectively.

2.1.2 Data Preprocessing Pipeline

To prepare the UC Merced dataset for training and evaluation, a systematic data preprocessing pipeline was implemented. The steps are as follows:

- **Download and extraction** of the UC Merced Land Use dataset from the official source.

```python
def download_uc_merced():
    """Download and extract the UC Merced Land Use dataset"""
    if not os.path.exists(DATA_DIR):
        os.makedirs(DATA_DIR)
        print("Downloading UC Merced dataset...")
        response = requests.get(DATASET_URL, stream=True)
        zip_path = os.path.join(DATA_DIR, "UCMerced_LandUse.zip")

        with open(zip_path, "wb") as file:
            for chunk in response.iter_content(chunk_size=1024):
                if chunk:
                    file.write(chunk)

        print("Extracting dataset...")
        with zipfile.ZipFile(zip_path, "r") as zip_ref:
            zip_ref.extractall(DATA_DIR)
        os.remove(zip_path)
        print("UC Merced dataset downloaded and extracted successfully!")
    else:
        print("UC Merced dataset already downloaded.")
```

- **Color space conversion:** All images were converted from BGR (default OpenCV format) to RGB color space for consistency in visualization and model compatibility.
- **Creation of paired low-resolution (LR) and high-resolution (HR) images:**
  - HR images were resized to **256×256 pixels**.
  - LR images were generated by downsampling to **128×128 pixels** to simulate a 2× resolution enhancement scenario.
  - A **Gaussian blur** with kernel size 3×3 and standard deviation $\sigma = 0.5$ was applied to the LR images to simulate realistic degradation.

```python
def create_paired_dataset(target_hr_size=256, scale_factor=2):
    """Creates paired low-resolution and high-resolution images for super-resolution training"""
    lr_images = []
    hr_images = []
    original_images = []
    image_paths = []

    lr_size = target_hr_size // scale_factor

    for category in SELECTED_CATEGORIES:
        category_path = os.path.join(DATASET_PATH, category)
        if not os.path.exists(category_path):
            print(f"Warning: Category {category} not found. Skipping...")
            continue

        files = os.listdir(category_path)[:SAMPLE_COUNT]

        for file in files:
            img_path = os.path.join(category_path, file)
            img = cv2.imread(img_path)

            if img is None:
                print(f"Warning: Could not read {file}. Skipping...")
                continue

            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            original_images.append(img.copy())
            image_paths.append(img_path)

            hr_img = cv2.resize(img, (target_hr_size, target_hr_size))
            hr_images.append(hr_img / 255.0)

            lr_img = cv2.resize(img, (lr_size, lr_size))
            lr_img = cv2.GaussianBlur(lr_img, (3, 3), 0.5)
            lr_images.append(lr_img / 255.0)

    return (np.array(lr_images, dtype=np.float32),
            np.array(hr_images, dtype=np.float32),
            original_images,
            image_paths)
```

- **Normalization:** Pixel values of all images were normalized to the range $[\mathbf{0}, \mathbf{1}]$ to facilitate stable and efficient model training.
- **Multi-resolution datasets:** Additional datasets were created with downscaling factors of $\mathbf{2\times}$, $\mathbf{4\times}$, **and** $\mathbf{8\times}$ to enable comprehensive testing across various super-resolution scales.

```python
def create_multi_resolution_dataset(original_size=256, target_size=256, downscale_factors=[2, 4, 8]):
    """Creates multiple datasets with different levels of downsampling for testing"""
    hr_images = []
    lr_datasets = {factor: [] for factor in downscale_factors}
    original_images = []
    image_paths = []

    for category in SELECTED_CATEGORIES:
        category_path = os.path.join(DATASET_PATH, category)
        if not os.path.exists(category_path):
            print(f"Warning: Category {category} not found. Skipping...")
            continue

        files = os.listdir(category_path)[:SAMPLE_COUNT]

        for file in files:
            img_path = os.path.join(category_path, file)
            img = cv2.imread(img_path)

            if img is None:
                print(f"Warning: Could not read {file}. Skipping...")
                continue

            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            original_images.append(img.copy())
            image_paths.append(img_path)

            hr_img = cv2.resize(img, (target_size, target_size))
            hr_images.append(hr_img / 255.0)

            for factor in downscale_factors:
                lr_size = target_size // factor
                downsampled = cv2.resize(img, (lr_size, lr_size))
                if factor >= 4:
                    downsampled = cv2.GaussianBlur(downsampled, (3, 3), 0.5)
                lr_img = cv2.resize(downsampled, (target_size, target_size))
                lr_datasets[factor].append(lr_img / 255.0)

    hr_images = np.array(hr_images, dtype=np.float32)
    for factor in downscale_factors:
        lr_datasets[factor] = np.array(lr_datasets[factor], dtype=np.float32)

    return hr_images, lr_datasets, original_images, image_paths
```

2.2 Enhancement Approaches

This section outlines the various resolution enhancement techniques explored in this study, categorized into traditional methods, deep learning-based approaches, and novel hybrid strategies.

2.2.1 Traditional Interpolation Methods

- **Bicubic Interpolation:** Uses cubic splines to estimate pixel values, providing smoother results than nearest-neighbor or bilinear interpolation.
- **Lanczos Interpolation:** Utilizes a sinc function windowed by another sinc function (Lanczos filter) for high-quality resampling and edge preservation.

```python
class InterpolationMethods:
    """Traditional interpolation methods for image upscaling"""

    @staticmethod
    def bicubic(image, scale=2):
        """Apply bicubic interpolation to upscale the image"""
        h, w = image.shape[:2]
        return cv2.resize(image, (w*scale, h*scale), interpolation=cv2.INTER_CUBIC)

    @staticmethod
    def lanczos(image, scale=2):
        """Apply Lanczos interpolation to upscale the image"""
        h, w = image.shape[:2]
        return cv2.resize(image, (w*scale, h*scale), interpolation=cv2.INTER_LANCZOS4)
```

2.2.2 Deep Learning-Based Super-Resolution

CNN Architecture

- **Input Layer:** Accepts variable-sized RGB images.
- **Convolutional Layers:** Stacked layers with LeakyReLU activation functions.
- **Residual Connections:** Included to facilitate better gradient flow and stable training.
- **Upsampling Layers:** Based on the required scale factor (e.g., 2×, 4×).
- **Output Layer:** Final convolutional layer with sigmoid activation to produce the enhanced image.

Training Parameters

- **Loss Function:** Mean Squared Error (MSE)
- **Optimizer:** Adam optimizer with a learning rate of 0.0002
- **Batch Size:** 8
- **Epochs:** 10
- **Validation Split:** 20%

```python
class SuperResolutionModel:
    """Deep learning-based super-resolution model"""
    @staticmethod
    def build_generator(scale_factor=1):
        """Build a super-resolution generator model based on CNN architecture"""
        input_layer = Input(shape=(None, None, 3))

        x = Conv2D(64, (3, 3), padding="same")(input_layer)
        x = LeakyReLU(alpha=0.2)(x)

        for _ in range(8):
            skip = x
            x = Conv2D(64, (3, 3), padding="same")(x)
            x = BatchNormalization()(x)
            x = LeakyReLU(alpha=0.2)(x)
            x = Conv2D(64, (3, 3), padding="same")(x)
            x = BatchNormalization()(x)
            x = tf.keras.layers.add([x, skip])

        if scale_factor > 1:
            for _ in range(int(np.log2(scale_factor))):
                x = UpSampling2D(size=2)(x)
                x = Conv2D(32, (3, 3), padding="same")(x)
                x = LeakyReLU(alpha=0.2)(x)

        x = Conv2D(3, (3, 3), padding="same", activation="sigmoid")(x)

        model = Model(input_layer, x)
        return model

    @staticmethod
    def train(model, lr_images, hr_images, epochs=10, batch_size=16):
        """Train the super-resolution model"""
        model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0002), loss="mse")

        history = model.fit(
            lr_images, hr_images,
            epochs=epochs,
            batch_size=batch_size,
            validation_split=0.2,
            verbose=1
        )
        return history
```

2.2.3 Traditional Enhancement Techniques

Wavelet Sharpening

- Discrete Wavelet Transform (DWT) using Haar wavelets
- Amplification of detail coefficients by a factor of 1.5
- Inverse DWT applied to reconstruct the enhanced image

Wiener Filtering

- Statistical approach used for image denoising and deblurring
- Kernel size: 5×5

- Special handling for numerical stability, including replacement of NaN values

```python
class EnhancementMethods:
    """Image enhancement techniques"""

    @staticmethod
    def wavelet_sharpening(image):
        """Apply wavelet-based sharpening to enhance image details"""
        img_float = image.astype(np.float32)

        if len(img_float.shape) == 3:
            r, g, b = cv2.split(img_float)
            r_coeffs = pywt.dwt2(r, 'haar')
            g_coeffs = pywt.dwt2(g, 'haar')
            b_coeffs = pywt.dwt2(b, 'haar')
            r_sharp = pywt.idwt2((r_coeffs[0], (r_coeffs[1][0]*1.5, r_coeffs[1][1]*1.5, r_coeffs[1][2]*1.5)), 'haar')
            g_sharp = pywt.idwt2((g_coeffs[0], (g_coeffs[1][0]*1.5, g_coeffs[1][1]*1.5, g_coeffs[1][2]*1.5)), 'haar')
            b_sharp = pywt.idwt2((b_coeffs[0], (b_coeffs[1][0]*1.5, b_coeffs[1][1]*1.5, b_coeffs[1][2]*1.5)), 'haar')
            sharpened = cv2.merge([r_sharp, g_sharp, b_sharp])
        else:
            coeffs = pywt.dwt2(img_float, 'haar')
            sharpened = pywt.idwt2((coeffs[0], (coeffs[1][0]*1.5, coeffs[1][1]*1.5, coeffs[1][2]*1.5)), 'haar')

        return np.clip(sharpened, 0, 255).astype(np.uint8)

    @staticmethod
    def wiener_filter(image, kernel_size=(5, 5)):

        def safe_wiener(channel):
            # Ensure float32 for processing
            channel = channel.astype(np.float32)
            # Apply Wiener filter
            filtered = wiener(channel, kernel_size)
            # Replace NaNs and Infs with zeros or safe values
            filtered = np.nan_to_num(filtered, nan=0.0, posinf=255.0, neginf=0.0)
            # Clip and convert back
            return np.clip(filtered, 0, 255).astype(np.uint8)

        if len(image.shape) == 3:
            r, g, b = cv2.split(image)
            r_deblurred = safe_wiener(r)
            g_deblurred = safe_wiener(g)
            b_deblurred = safe_wiener(b)
            deblurred = cv2.merge([r_deblurred, g_deblurred, b_deblurred])
        else:
            deblurred = safe_wiener(image)

        return deblurred
```

2.2.4 Novel Hybrid Methods

- **Wavelet-SR Hybrid:** Applies wavelet sharpening to the output of the super-resolution model.
- **Wiener-SR Hybrid:** Applies Wiener filtering to the super-resolved image for additional denoising and clarity.

```python
# Novel Hybrid Methods
@staticmethod
def wavelet_sr_hybrid(sr_image):
    """Combine Super-Resolution with Wavelet Sharpening"""
    img_uint8 = (sr_image * 255).astype(np.uint8)
    sharpened = EnhancementMethods.wavelet_sharpening(img_uint8)
    return sharpened.astype(np.float32) / 255.0

@staticmethod
def wiener_sr_hybrid(sr_image):
    """Combine Super-Resolution with Wiener Filter"""
    img_uint8 = (sr_image * 255).astype(np.uint8)
    deblurred = EnhancementMethods.wiener_filter(img_uint8)
    return deblurred.astype(np.float32) / 255.0
```

2.3 Evaluation Framework

This section outlines the evaluation strategies used to assess the performance of the various enhancement methods, including quantitative metrics, visualizations, and implementation-specific details.

2.3.1 Image Quality Metrics

The following standard image quality metrics were used to quantitatively evaluate the enhancement results:

- **Peak Signal-to-Noise Ratio (PSNR):** Measures the ratio between the maximum possible signal power and the power of corrupting noise. Higher PSNR values indicate better image quality.
- **Structural Similarity Index (SSIM):** Evaluates image similarity by comparing luminance, contrast, and structural information between the enhanced and reference images.
- **Mean Squared Error (MSE):** Calculates the average of the squared differences between corresponding pixels of the enhanced and ground truth images.

```python
"""#Evaluation Metrics:-"""

#========== EVALUATION METRICS ==========

class EvaluationMetrics:
    """Methods to evaluate image enhancement quality"""

    @staticmethod
    def calculate_psnr(img1, img2):
        """Calculate Peak Signal-to-Noise Ratio"""
        return psnr(img1, img2)

    @staticmethod
    def calculate_ssim(img1, img2):
        """Calculate Structural Similarity Index"""
        return ssim(img1, img2, data_range=1.0, channel_axis=-1)

    @staticmethod
    def calculate_mse(img1, img2):
        """Calculate Mean Squared Error"""
        return np.mean((img1 - img2) ** 2)

    @staticmethod
    def evaluate_method(enhanced_images, reference_images, method_name):
        """Evaluate a method using multiple metrics"""
        ssim_values = []
        psnr_values = []
        mse_values = []

        for i in range(len(enhanced_images)):
            img1 = enhanced_images[i].astype(np.float32) / 255.0 if enhanced_images[i].max() > 1.0 else enhanced_images[i].astype(np.float32)
            img2 = reference_images[i].astype(np.float32) / 255.0 if reference_images[i].max() > 1.0 else reference_images[i].astype(np.float32)

            ssim_values.append(EvaluationMetrics.calculate_ssim(img1, img2))
            psnr_values.append(EvaluationMetrics.calculate_psnr(img1, img2))
            mse_values.append(EvaluationMetrics.calculate_mse(img1, img2))

        return {
            "method": method_name,
            "ssim": np.mean(ssim_values),
            "psnr": np.mean(psnr_values),
            "mse": np.mean(mse_values)
        }
```

2.3.2 Performance Analysis

To assess computational performance and visual effectiveness, the following strategies were employed:

- Measurement of **processing time** for each enhancement method.
- **Batch processing** of 16 images at a time to optimize prediction speed.
- **Comprehensive visual comparison** of output images across all methods.
- **Statistical analysis** of the image quality metrics across the dataset.

2.3.3 Visualization

To aid in interpretation and presentation of results, several types of visualizations were generated:

- **Training history plots** showing loss curves for the super-resolution model.
- **Side-by-side visual comparisons** of original, degraded, and enhanced images.
- **Bar charts** to compare average PSNR, SSIM, and MSE values across methods.
- **Processing time comparison plots** to evaluate computational efficiency.
- **Difference maps** to visualize degradation levels and reconstruction accuracy.

2.3.4 Implementation Details

The following tools and practices were used in the development and execution of the evaluation pipeline:

- Implemented in **Python** using libraries such as **TensorFlow**, **OpenCV**, **scikit-image**, and **PyWavelets**.
- **Model saving and loading** functionality for efficient reuse and deployment.
- Structured and organized **results storage** in dedicated directories.
- Used **tqdm** progress bars to monitor and manage long-running processes.

Chapter 3

Results and Observations

This section presents the quantitative evaluation of various resolution enhancement methods based on three image quality metrics: Structural Similarity Index Measure (SSIM), Peak Signal-to-Noise Ratio (PSNR), and Mean Squared Error (MSE). The results were obtained by processing the test set with each method and calculating the average metric values.

3.1 Tabular Comparision of Metrics

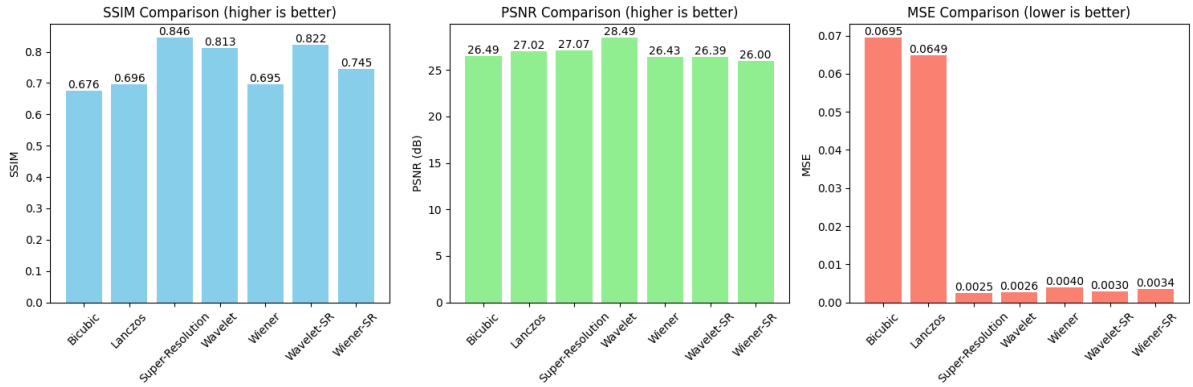
Table 3.1: Quantitative Comparison of Enhancement Methods

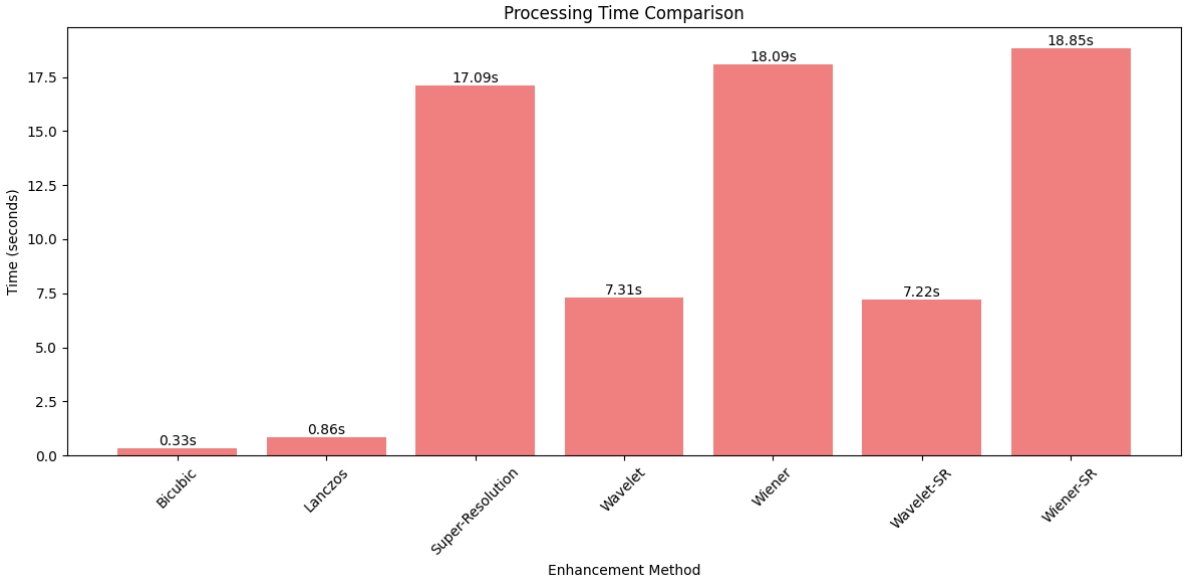
Method	**SSIM**	**PSNR (dB)**	**MSE**
Bicubic	0.6756	26.49	0.069530
Lanczos	0.6963	27.02	0.064853
Super-Resolution	**0.8456**	27.07	**0.002539**
Wavelet	0.8126	**28.49**	0.002618
Wiener	0.6949	26.43	0.004028
Wavelet-SR	0.8221	26.39	0.002962
Wiener-SR	0.7447	26.00	0.003429

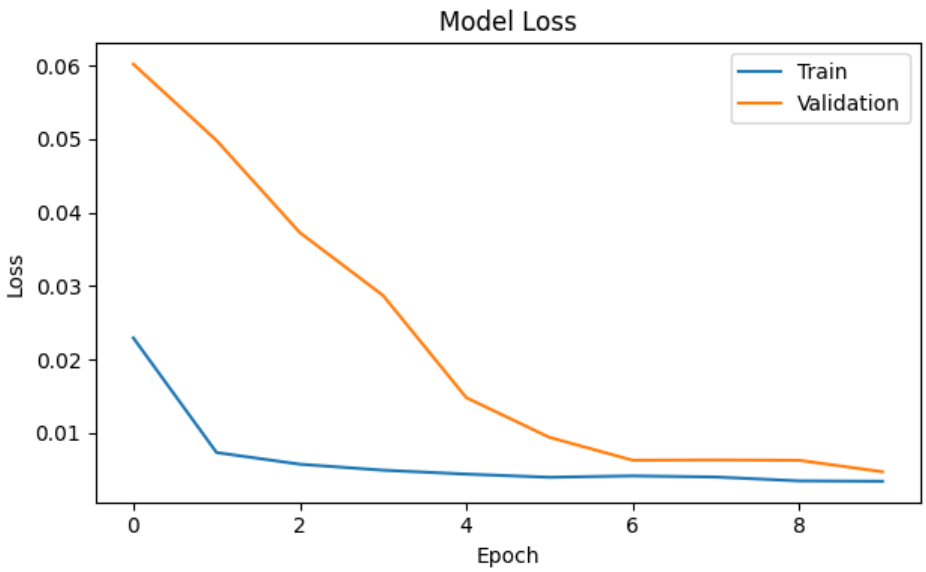
As shown in Table 3.1, the deep learning-based Super-Resolution model achieved the highest SSIM value, indicating superior structural similarity with the ground truth images. The Wavelet enhancement method recorded the highest PSNR value, reflecting better signal reconstruction. In terms of MSE, the Super-Resolution method demonstrated the lowest error, confirming its effectiveness in pixel-wise reconstruction accuracy.

Hybrid methods such as Wavelet-SR and Wiener-SR showed improved performance compared to their standalone traditional counterparts, demonstrating the potential of combining learning-based and classical techniques.

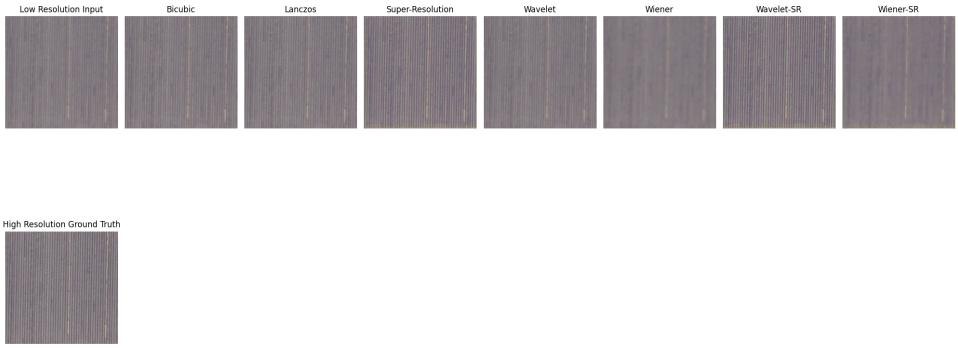
3.2 Metrics Visualization

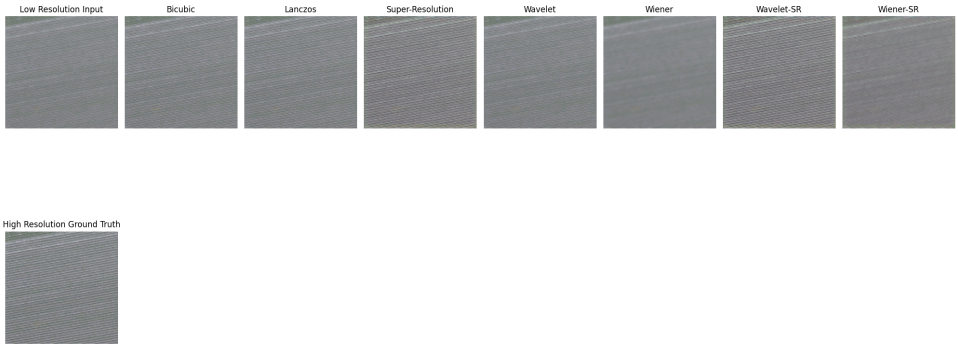


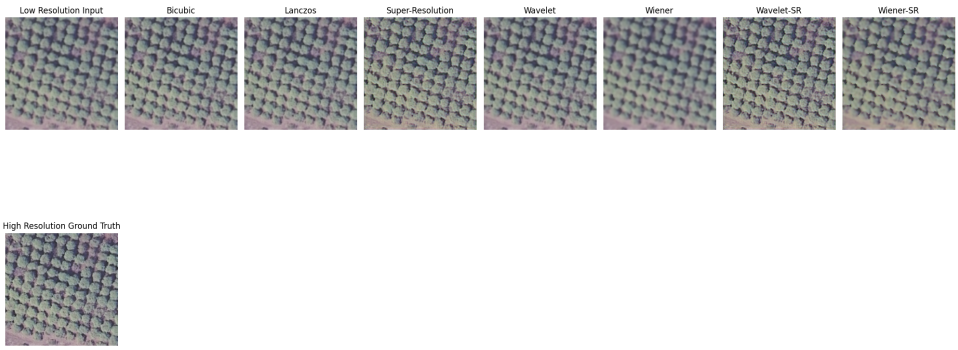




3.3 Testing Results







Chapter 4

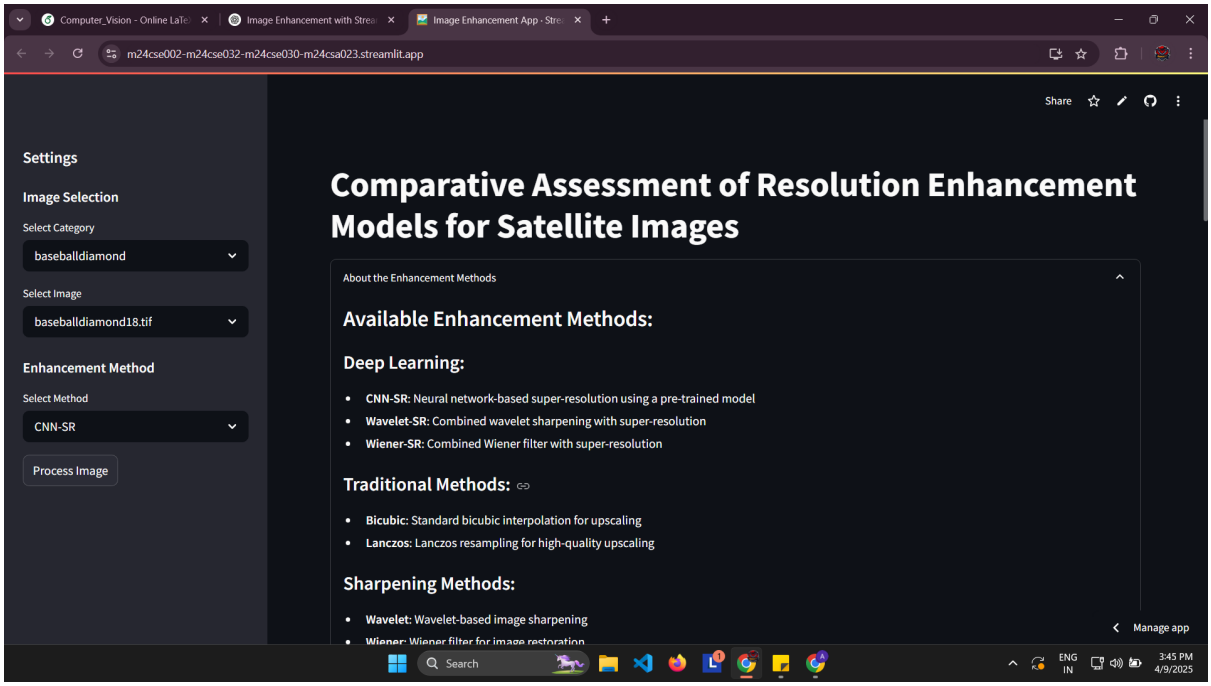
Important Links

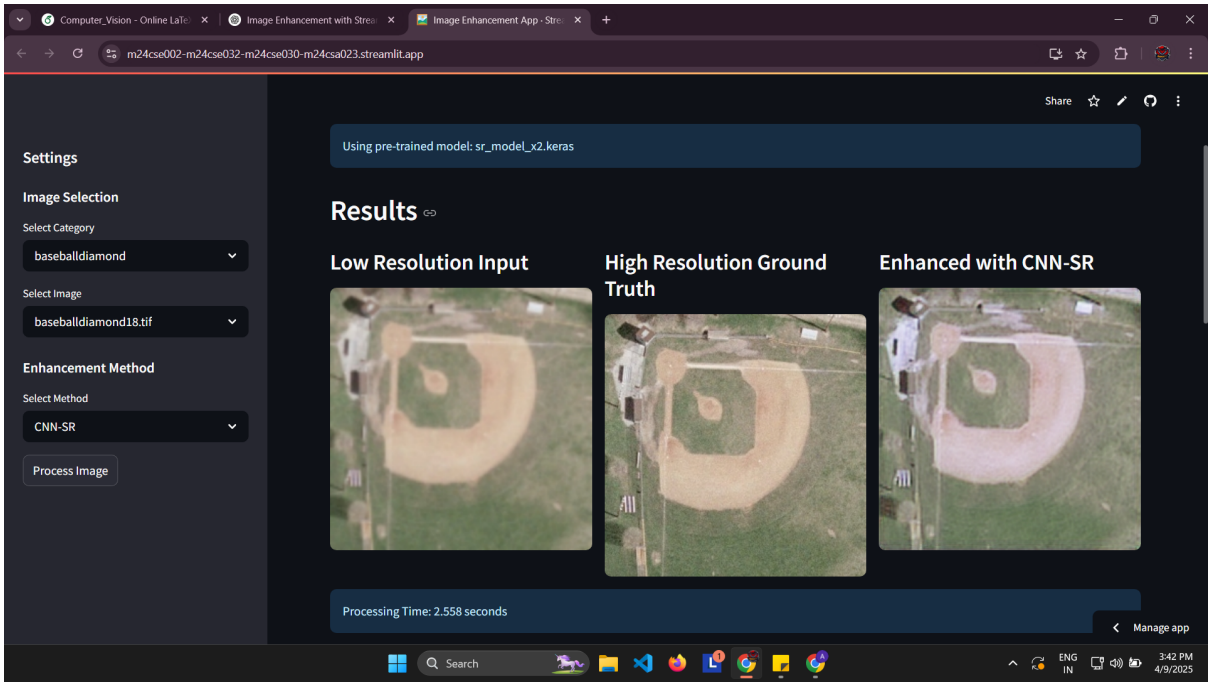
The following links provide access to the deployed application and the Colab notebook used for development and experimentation:

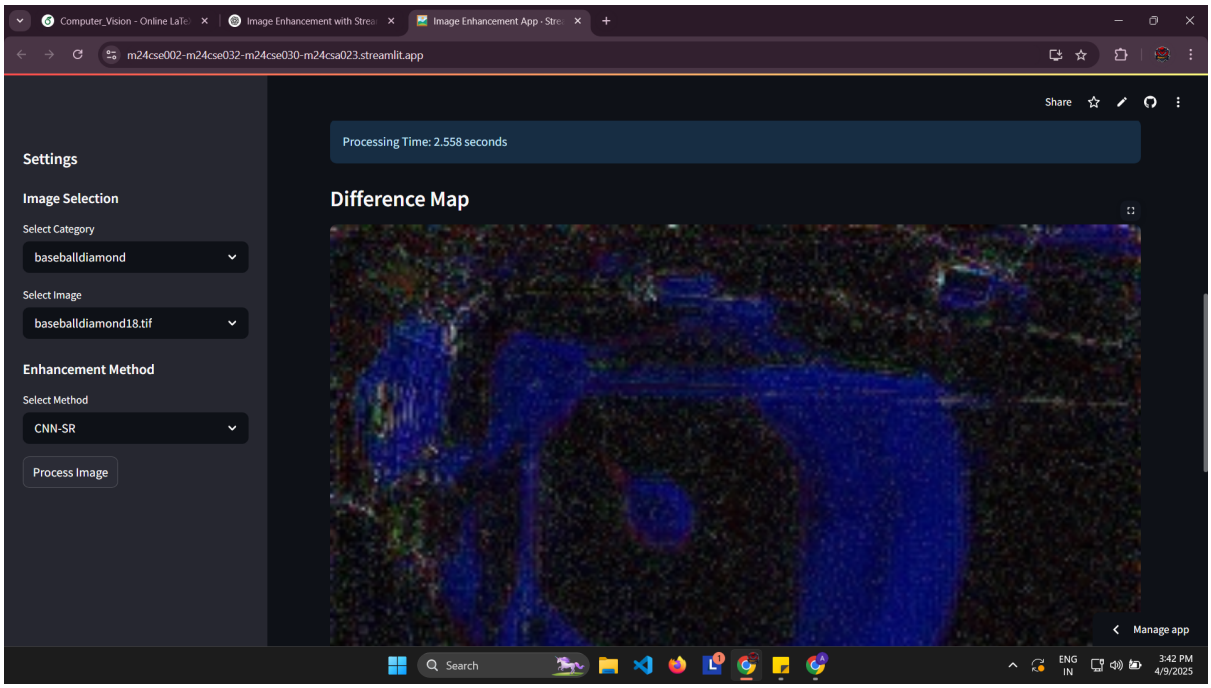
- **Deployment Link:** https://m24cse002-m24cse032-m24cse030-m24csa023.streamlit.app/
- **Colab Notebook:** https://colab.research.google.com/drive/1QwP7Iwlwbxg8fQDnciDQ$_w 19lr9tbaVI?us$ $sharing$
- **Github Link:** https://github.com/akaash897/Computer$_Vision$

Chapter 5

Website Screenshots




Computer_Vision - Online LaTe
Image Enhancement with Strea
Image Enhancement App · Stre
m24cse002-m24cse032-m24cse030-m24csa023.streamlit.app
Share
Settings
Image Selection
Select Category
baseballdiamond
Select Image
baseballdiamond18.tif
Enhancement Method
Select Method
CNN-SR
Process Image
Using pre-trained model: sr_model_x2.keras
Results
Low Resolution Input
High Resolution Ground Truth
Enhanced with CNN-SR
Processing Time: 2.558 seconds
Manage app
Search
ENG
IN
3:42 PM
4/9/2025


Computer_Vision - Online LaTe
Image Enhancement with Strea
Image Enhancement App · Stre
m24cse002-m24cse032-m24cse030-m24csa023.streamlit.app
Share
Settings
Image Selection
Select Category
baseballdiamond
Select Image
baseballdiamond18.tif
Enhancement Method
Select Method
CNN-SR
Process Image
Processing Time: 2.558 seconds
Difference Map
Manage app
Search
ENG
IN
3:42 PM
4/9/2025

Chapter 6

Conclusion and Summary

This study presents a comprehensive comparative analysis of various resolution enhancement techniques applied to satellite imagery using the UC Merced Land Use dataset. The evaluation encompassed traditional interpolation methods, deep learning-based super-resolution, classical image enhancement techniques, and hybrid approaches that integrate both paradigms.

Quantitative results, assessed through metrics such as SSIM, PSNR, and MSE, indicate that the deep learning-based Super-Resolution model delivers the best overall structural similarity and lowest reconstruction error. The Wavelet sharpening method achieved the highest PSNR, showcasing its effectiveness in preserving signal fidelity. Furthermore, hybrid models such as Wavelet-SR and Wiener-SR demonstrated notable improvements over their respective standalone methods, underscoring the potential of combining data-driven learning with traditional enhancement techniques.

In addition to metric-based evaluation, visual comparisons highlighted the ability of each method to recover fine details and suppress artifacts, providing practical insights into real-world applicability. The evaluation framework incorporated performance analysis, training visualization, and extensive testing across various degradation levels.

The key contributions of this project include:

- Implementation and comparison of multiple enhancement methods tailored to satellite imagery.
- A unified evaluation framework integrating quality metrics, runtime analysis, and visual inspection.
- Development of hybrid approaches that leverage the strengths of both deep learning and classical techniques.
- Creation of a scalable preprocessing and testing pipeline supporting multi-resolution experimentation.

Overall, this work contributes to the growing field of remote sensing image enhancement and paves the way for future research in deploying efficient and accurate resolution enhancement methods for large-scale satellite datasets.

Chapter 7

Team Member Contributions

Akaash Chatterjee (M24CSE002)

- Implemented the deep learning-based Super-Resolution model.
- Designed the CNN architecture and training pipeline for the SuperResolutionModel.
- Developed the Wavelet-SR hybrid enhancement approach by integrating wavelet sharpening with the SR model.
- Contributed to the main integration and evaluation workflow.
- Acted as Team Leader,helping and aiding in all steps including the main function

Alok Dutta (M24CSE032)

- Implemented traditional interpolation methods: Bicubic and Lanczos.
- Developed dataset preparation pipeline including UC Merced extraction and paired dataset generation.

Prathmesh Chintamani Gosavi (M24CSA023)

- Implemented Wavelet-based enhancement and sharpening methods.
- Developed the Wiener-SR hybrid model combining super-resolution with Wiener filtering.
- Handled Data Preprocesssing Steps

Vishwanath Singh (M24CSE030)

- Implemented the Wiener filter-based enhancement method.
- Developed visualization tools including metric comparison plots, time analysis charts, and degradation maps.