# Project: Multi-Stage Image Enhancement and Analysis Pipeline

## Group Number 22 - Members


- **M24DE2023** Raghavendra G M24DE2023@iitj.ac.in
- **M24DE2024** Rahul Bansal M24DE2024@iitj.ac.in
- **M24DE2025** Saurav Suman M24DE2025@iitj.ac.in
- **M24DE2032** Srishty Suman M24DE2032@iitj.ac.in
- **M24DE2035** Tarun Prajapati M24DE2035@iitj.ac.in


# Objective

The project is about designing a multi-stage image processing pipeline where each stage applies a different computer vision algorithm. Each stage is having one or more algorithm. We have covered following stages

- Preprocessing
- Edge Detection
- Super Resolution
- Object Detection & Segmentation
- Compression and Decompression

# Approaches used

For each stage following algorithm are used

- **Preprocessing**
- **Edge Detection**
- **Super Resolution**
- **SLIC, YOLO & RetinaNet**
- **Compression and Deceompression**

# Results

# Deployment link

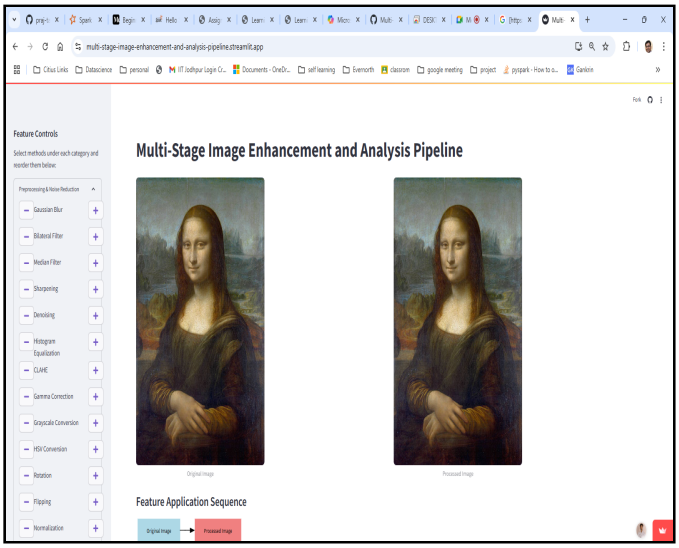
{https://multi-stage-image-enhancement-and-analysis-pipeline.streamlit.app/]

# Github link

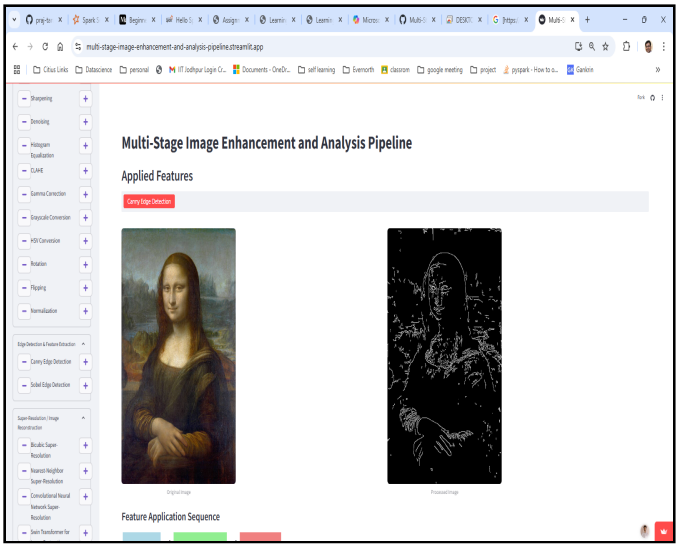
https://github.com/praj-tarun/Multi-Stage-Image-Enhancement-and-Analysis-Pipeline.git

# ScreenShot

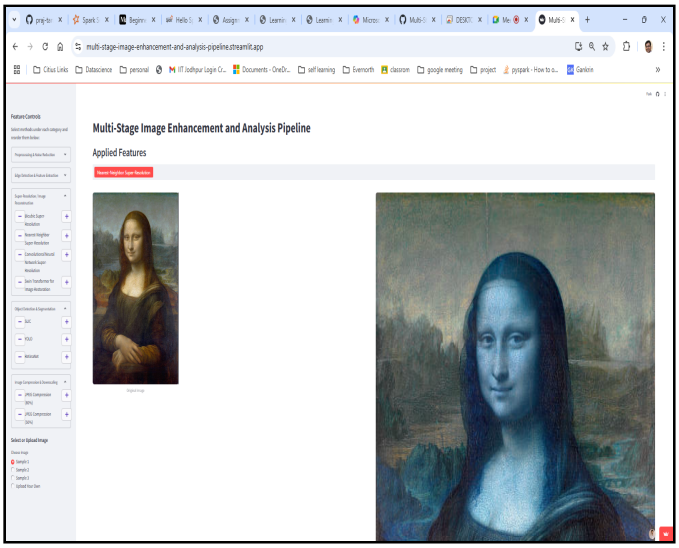
**Landing page Screenshot**



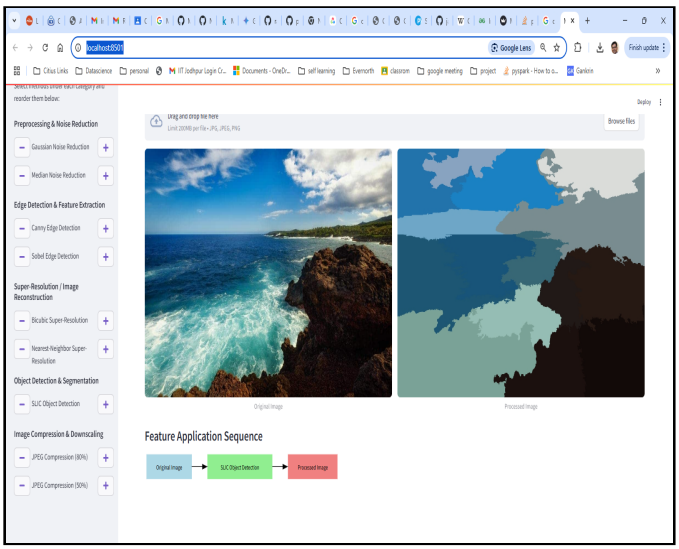

**Canny Edge Detection**



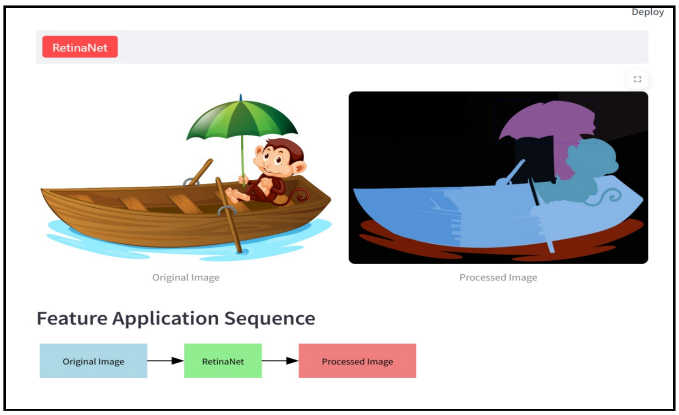

## Super resolution

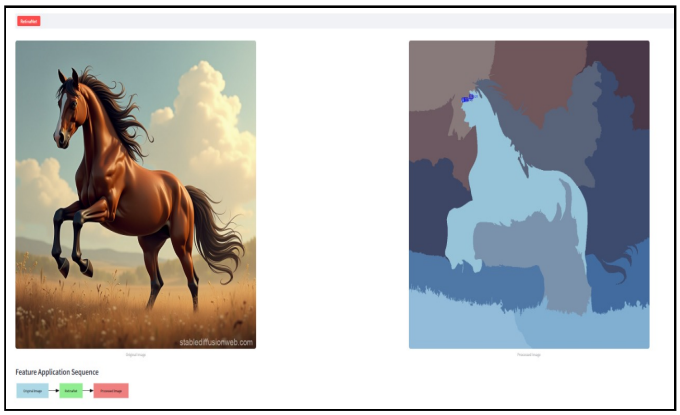



## SLIC

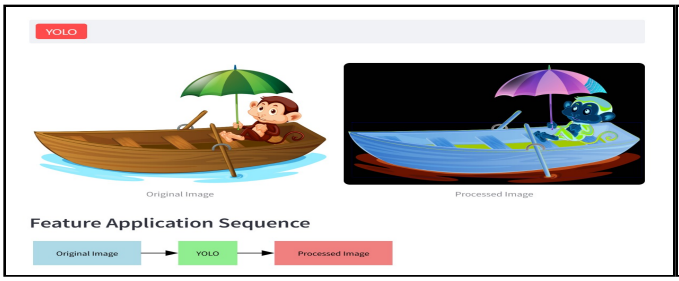


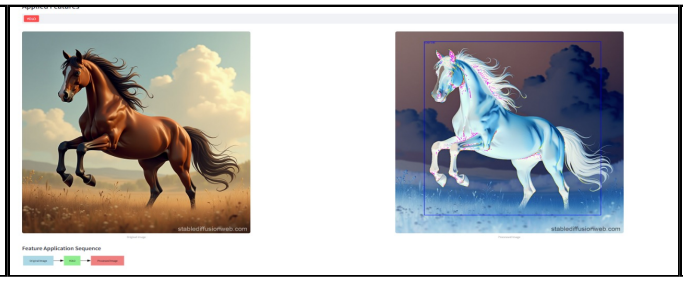
**RetinaNet**



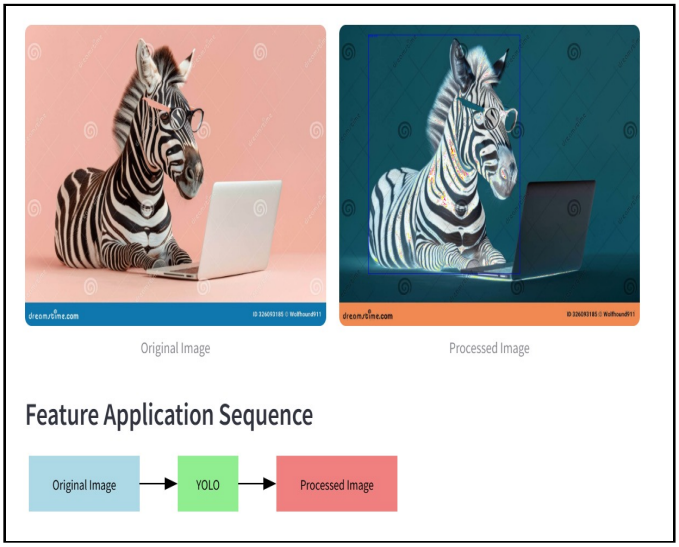



**Yolo**



**Comparison of RetinaNet and Yolo**




# Contribution of Each member

**M24DE2023** (Raghavendra G)

**Super-Resolution Methods**

This project implements various super-resolution methods to enhance low-resolution images using different techniques. The following methods are supported:

**1. Bicubic Super-Resolution** Method Name: Bicubic Interpolation

Description: The Bicubic interpolation method is a commonly used algorithm for resizing images. It uses a cubic convolution to calculate the new pixel values, offering better image quality than nearest-neighbor or bilinear interpolation methods, particularly for enlarging images.

Use Case: Suitable for general image resizing where the quality is relatively important but computational efficiency is key.

**2. Nearest-Neighbor Super-Resolution** Method Name: Nearest-Neighbor Interpolation

Description: Nearest-Neighbor interpolation is the simplest image resizing technique. It uses the value of the nearest neighboring pixel to assign to the new pixel. While fast, it tends to result in a blocky appearance in the resized image, especially for larger upscaling factors.

Use Case: Suitable for quick prototypes or applications where computational efficiency is crucial but visual quality is secondary.

**3. Convolutional Neural Network Super-Resolution (SRCNN)** Method Name: SRCNN (Super-Resolution Convolutional Neural Network)

Description: SRCNN is a deep learning-based approach to super-resolution. The model is trained to predict high-resolution images from low-resolution inputs using convolutional layers. It is more advanced than traditional interpolation methods, offering better results in terms of image quality and detail.

Use Case: Suitable for applications where higher image quality is required, and computational resources are available for training or using pre-trained models.

**4. Swin Transformer for Image Restoration (SwinIR)** Method Name: Swin Transformer for Image Restoration (SwinIR)

Description: SwinIR is a deep learning-based image restoration model that leverages the Swin Transformer, a powerful architecture for capturing long-range dependencies in images. It performs exceptionally well for tasks like super-resolution, denoising, and deblurring, outperforming traditional methods in terms of both quality and efficiency.

**Use Case:** Suitable for high-quality image enhancement tasks, especially when state-of-the-art results are required in areas like super-resolution and denoising.

**Function Overview** Function Name: super_resolve The core function that handles the super-resolution process based on the specified method. This function selects the appropriate technique to upscale a low-resolution image.

**Parameters:** img: The low-resolution input image that you want to upscale.

**method:** The super-resolution method to use. Possible values are:

**bicubic:** Bicubic interpolation for resizing.

**'nearest_neighbor'**: Nearest-Neighbor interpolation for resizing.

**'srcnn'**: Super-Resolution Convolutional Neural Network (SRCNN) for image enhancement.

**'swinir'**: Swin Transformer-based Image Restoration (SwinIR) for high-quality restoration.

**Returns:** A high-resolution image, which is the result of applying the selected super-resolution method to the input image.

**Example Usage:** You can use the super_resolve function to apply any of the super-resolution techniques to an image:

Bicubic Super-Resolution: Applies bicubic interpolation to the image.

Nearest-Neighbor Super-Resolution: Uses nearest-neighbor interpolation to upscale the image.

SRCNN Super-Resolution: Enhances the image using the SRCNN deep learning model.

SwinIR Super-Resolution: Restores the image quality using the Swin Transformer-based model.

**M24DE2024** (Rahul Bansal) :

Implementated slic segmentation

**M24DE2025** (Saurav Suman) : **Implementation RetinaNet:**

RetinaNet is a one-stage object detection model that combines speed and high accuracy, especially for detecting small or overlapping objects. It uses a backbone like ResNet with a Feature Pyramid Network (FPN) to extract rich, multi-scale features. Unlike other detectors, RetinaNet introduces Focal Loss, which focuses learning on hard, misclassified examples and reduces the impact of easy negatives. This makes it highly effective in handling class imbalance, a common issue in dense object detection. With its streamlined architecture and powerful loss function, RetinaNet bridges the gap between fast detectors like YOLO and accurate ones like Faster R-CNN.

**Steps**

1. Installing Tourch, Tourch Vision & RetinaNet (ResNet 50)
2. Loading a pretrained retinanet_resnet50_fpn Model using torchvision.models.detection
3. transform an image into a PyTorch tensor
4. Running inference on an Image
5. Extracting bounding boxes & class names
6. Applying segmentation using Image Draw

**M24DE2032** (Srishty Suman): **Implementation YOLO (V5):**

YOLOv5 is a fast, accurate one-stage object detection model developed by Ultralytics. It detects objects in a single forward pass using a CNN-based architecture, making it ideal for real-time applications. YOLOv5 uses CSPDarknet as a backbone, a PANet neck, and anchor-based predictions. It supports auto-learning bounding box anchors, data augmentation (like mosaic), and is implemented in PyTorch. YOLOv5 is lightweight, easy to train, and provides excellent speed-accuracy trade-off for applications like surveillance and autonomous driving.

**Steps**

1. Installing YOLOv5 & Dependencies
2. Loading a pretrained YOLOv5 Model using ultralytics

3. Running inference on an Image
4. Extracting bounding boxes & class names
5. Applying segmentation using Image Draw

**Comparision**

| Feature | RetinaNet | YOLO (v5) |
|---|---|---|
| Speed | Moderate | Very fast (real-time) |
| Accuracy | High, especially on small objs | Good, sometimes lower on small objs |
| Loss Function | Focal Loss | Binary cross-entropy |
| Detection Type | One-stage with advanced loss | Pure one-stage detector |
| Use Case | Dense scenes, balanced accuracy | Real-time applications |

**M24DE2035** (Tarun Prajapati) :

I was responsible for the design, development, and deployment of the core application framework, along with a complete preprocessing module. My contributions include:

### 🔧 Main Application

- Designed the overall layout and interaction flow of the Streamlit application.
- Developed a modular and scalable app structure, enabling easy integration of new features.
- Implemented dynamic drag-and-drop functionality for applying and reordering image processing steps.
- Deployed the fully working application to Streamlit Cloud with responsive UI and support for default images.

---

### 🧹 Preprocessing Module

I independently developed and integrated the following preprocessing techniques into the app.Each preprocessing method was built as a reusable function and integrated seamlessly into the pipeline, enabling users to apply and stack them in any order through the app interface.

- **Custom Denoising**
  Combined the original image with a Gaussian-blurred version to reduce noise while preserving image structure.
- **Gaussian Blur**
  Applied a standard Gaussian kernel to reduce minor image noise.
- **Bilateral Filter**
  Used edge-preserving smoothing to clean noise while retaining sharp transitions.
- **Median Filter**
  Replaced each pixel with the median of its neighbors to remove salt-and-pepper noise.
- **Sharpening**
  Enhanced fine details using an unsharp masking filter with tunable strength.
- **Histogram Equalization**
  Improved overall contrast by redistributing pixel intensities.
- **CLAHE**
  Applied adaptive histogram equalization to boost local contrast in limited regions.
- **Gamma Correction**
  Performed brightness correction using non-linear intensity transformation.
- **Grayscale Conversion**
  Converted color images to single-channel grayscale for simplified processing.
- **HSV Conversion**
  Transformed the image into HSV space to separate color and intensity components.
- **Rotation**
  Enabled image rotation using affine transformations for orientation adjustments.
- **Flipping**
  Added support for vertical and horizontal mirroring to augment image layout.
- **Normalization**
  Rescaled image intensity values to a consistent range for stable processing.