Project Report: Deep Learning Image Colorizer & Enhancer

By: Muhammad Ibrahim Alam

Date: November 22, 2025

1. Abstract

This project presents a hybrid software application designed to automatically colorize black-and-white images using Deep Learning. The system integrates a Python-based neural network backend with a MATLAB-based Graphical User Interface (GUI). Beyond basic colorization, the application includes a suite of post-processing tools (Contrast Enhancement, Edge Sharpening, and Scientific False-Color Mapping) to refine and analyze the output. The project demonstrates successful cross-language integration (MATLAB + Python) and the practical application of Convolutional Neural Networks (CNNs) in Digital Image Processing.

2. System Architecture

The project follows a **Hybrid Architecture**, leveraging the strengths of two different programming ecosystems:

- **Frontend (MATLAB App Designer):** Handles user interaction, image visualization, post-processing effects, and file management. It provides a robust environment for engineering-grade image analysis.
- **Backend (Python + OpenCV + PyTorch):** Handles the heavy-duty deep learning inference. Python is used for its extensive libraries in machine learning.
- **Communication Bridge:** MATLAB executes Python scripts via system commands, passing file paths as arguments to ensure seamless data transfer between the GUI and the AI model.

3. The Deep Learning Model

3.1 Model Architecture

The core colorization engine is based on the research paper **"Colorful Image Colorization"** by *Richard Zhang, Phillip Isola, and Alexei A. Efros (ECCV 2016)*.

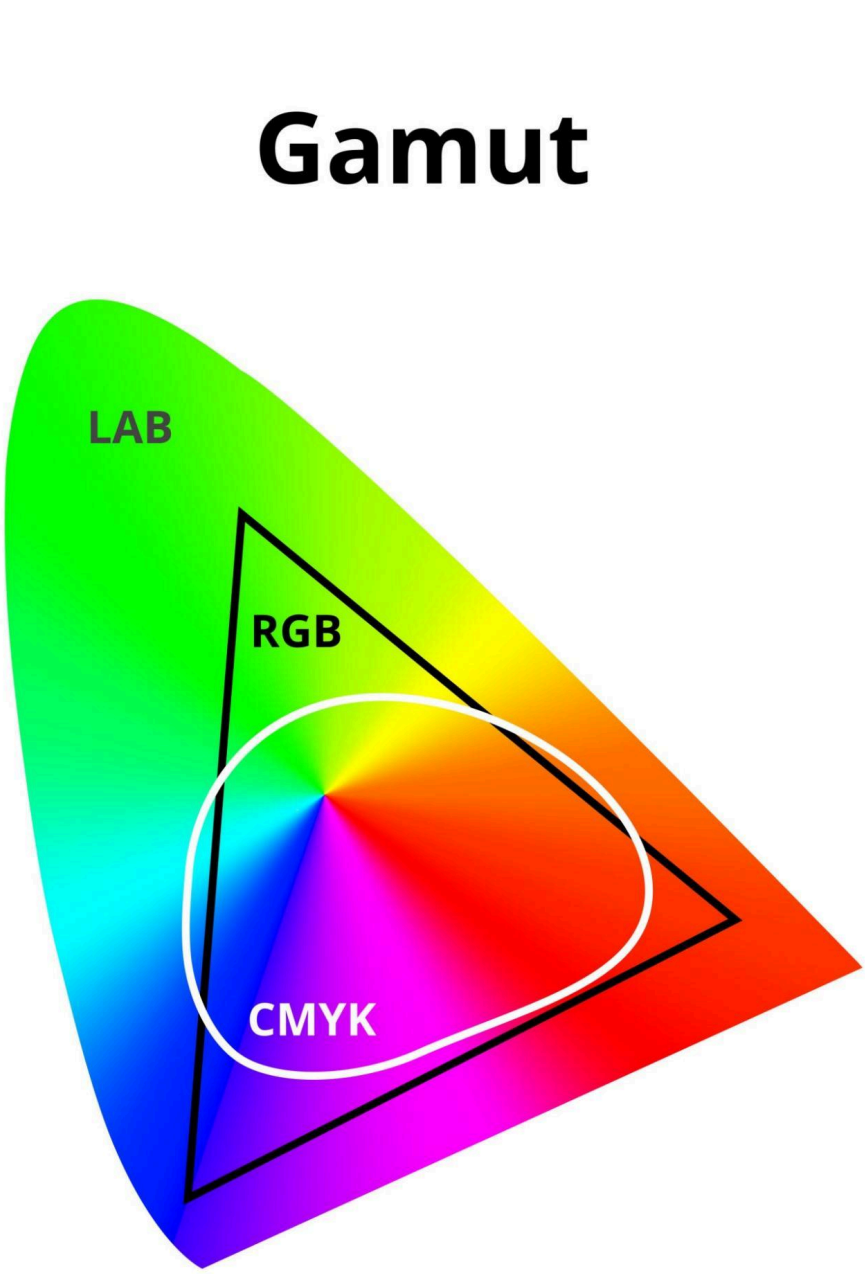
- **Type:** Convolutional Neural Network (CNN).
- **Structure:** The network resembles a **VGG-style architecture** but utilizes **Dilated Convolutions** (atrous convolutions). This allows the model to maintain a larger receptive field—seeing more of the image context at once—without downsampling the spatial resolution too heavily, which is crucial for pixel-level prediction tasks like colorization.

- **Layers:** The architecture consists of 22 convolutional layers organized into 8 blocks, followed by a Softmax distribution layer.

3.2 L*a*b Color Space Strategy

Unlike RGB images, which mix Red, Green, and Blue, this model operates in the **CIE L*a*b color space**:

- **Input (L):** The Lightness channel (Grayscale intensity). This is the only input provided to the network.
- **Output (ab):** The model predicts the 'a' (Green-Red) and 'b' (Blue-Yellow) channels.
- **Combination:** The input 'L' is concatenated with the predicted 'ab' to reconstruct the final color image.



Getty Images

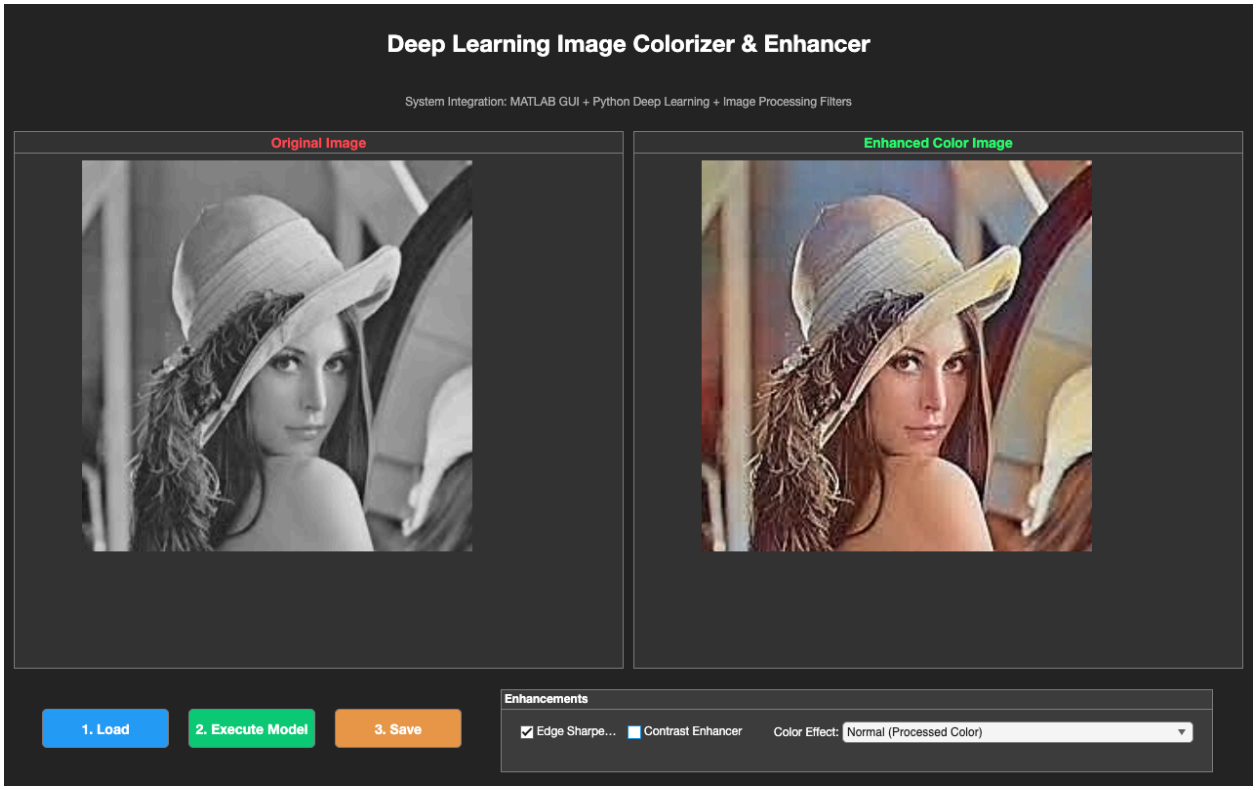
3.3 Training Methodology

The model was trained on the **ImageNet dataset**, consisting of approximately **1.3 million images**.

The Challenge:
Standard regression training (Mean Squared Error) tends to produce "safe," desaturated,

sepia-toned images because the average of all possible colors for a pixel is often gray.
The Solution (Class Rebalancing):

1. **Quantization:** The output space is quantized into **313 discrete color bins**. The problem is treated as a classification task rather than regression.
2. **Class Rebalancing:** A re-weighting loss function is used during training. It penalizes the model heavily for missing rare, vibrant colors and less for missing common background colors (like sky or ground). This forces the model to predict vibrant colors rather than defaulting to gray.



4. Key Application Features

The application introduces several advanced image processing features beyond simple colorization to ensure high-fidelity results.

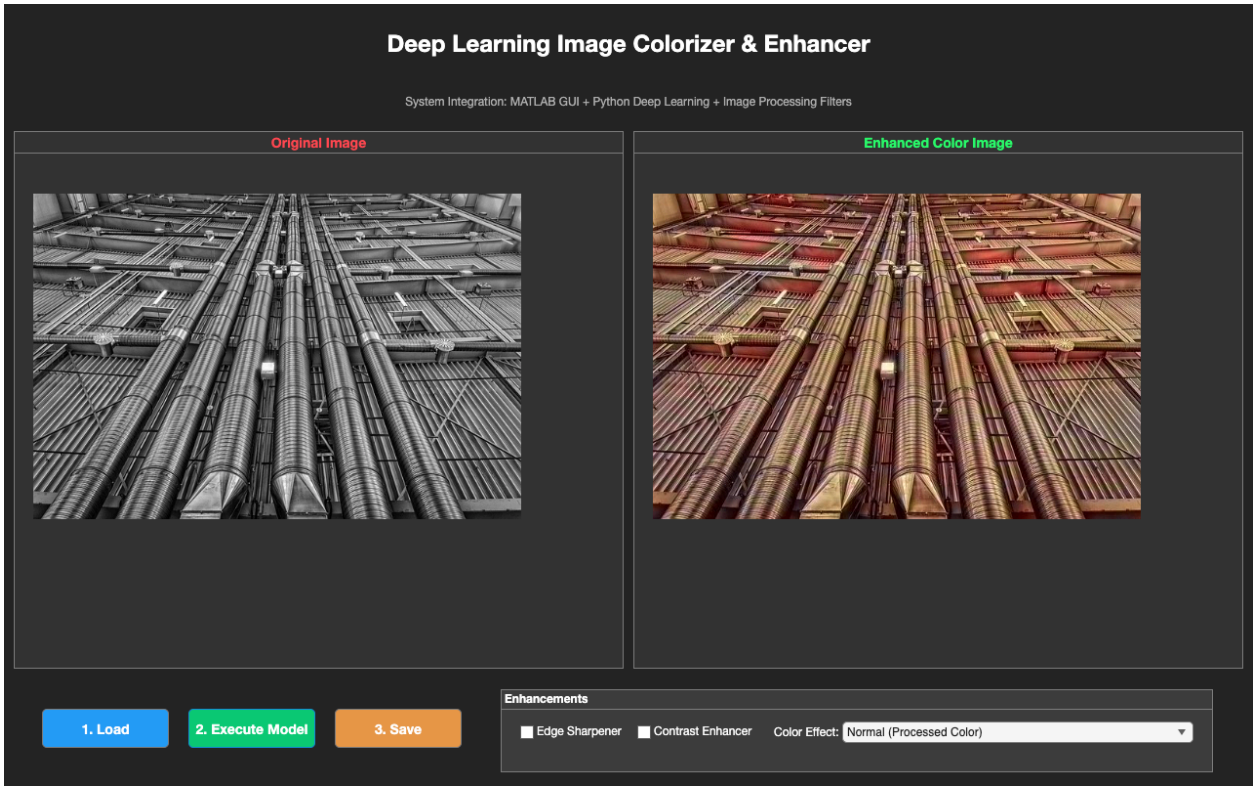
4.1 Feature: Edge Sharpener (Unsharp Masking)

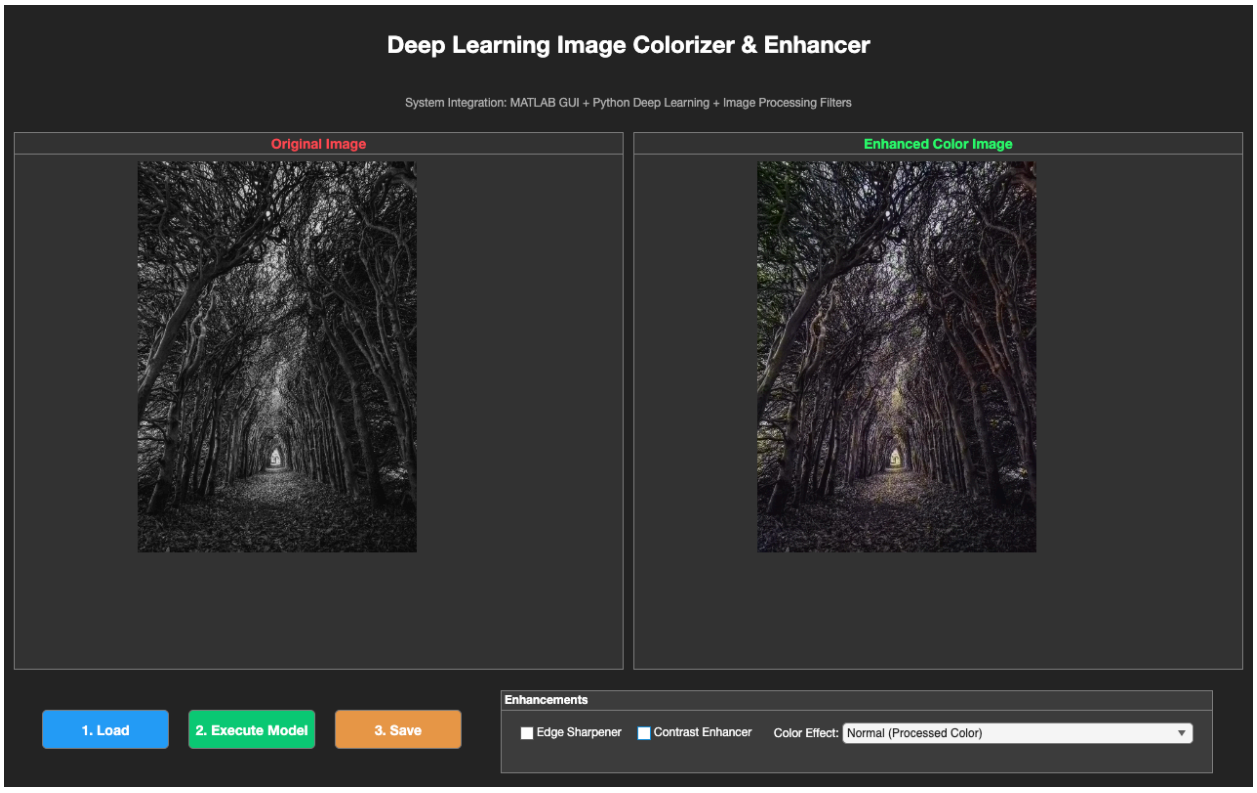
Deep learning models, particularly autoencoders, can sometimes produce "soft" images where high-frequency texture details (like hair, grass, or fabric) are smoothed out. To counter this, we implemented a dedicated **Edge Sharpener**.

- **Algorithm:** Unsharp Masking (imsharpen).
- **Methodology:** The system creates a blurred version of the original image (gaussian blur) and subtracts it from the original to isolate the "edges." These edges are then added back to the original image with a scaling factor.
- MATLAB Implementation:
  img = imsharpen(img, 'Radius', 1, 'Amount', 1.5);

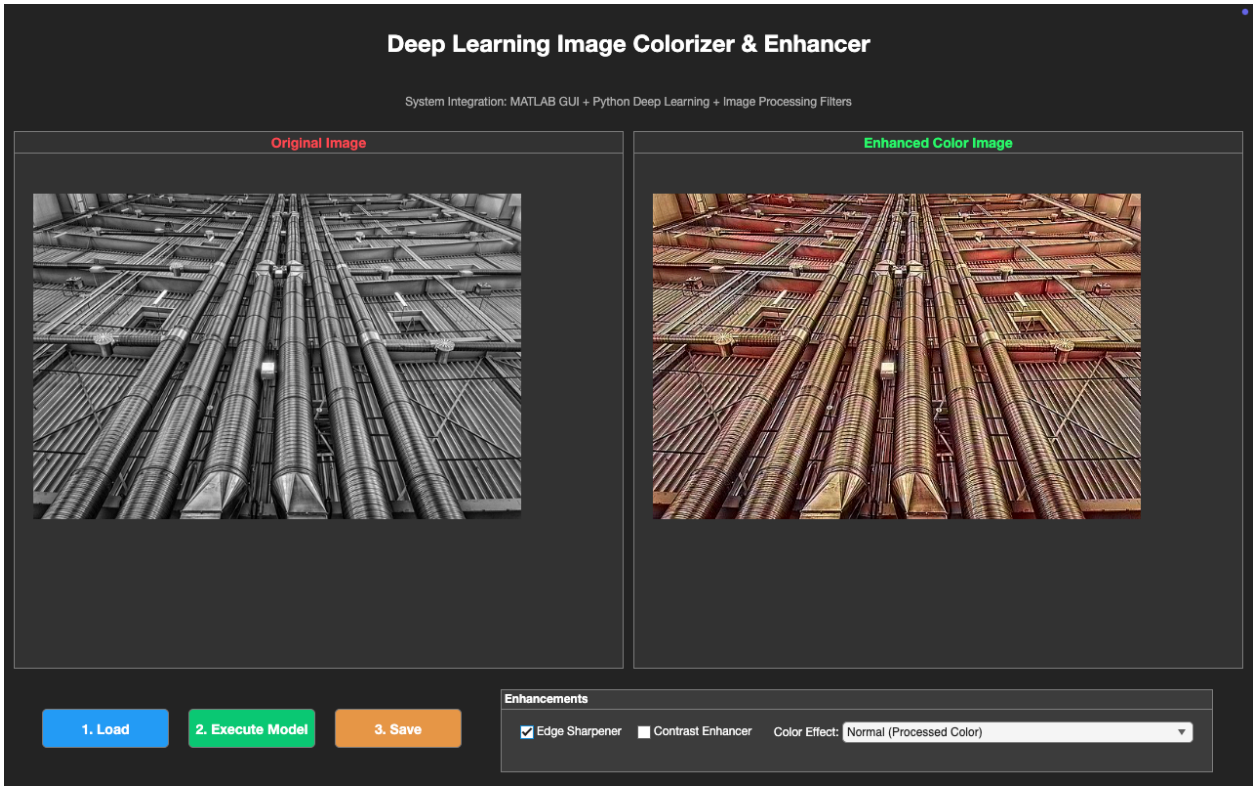
  We optimized the radius to 1.0 to enhance fine details without introducing "halo" artifacts around objects.

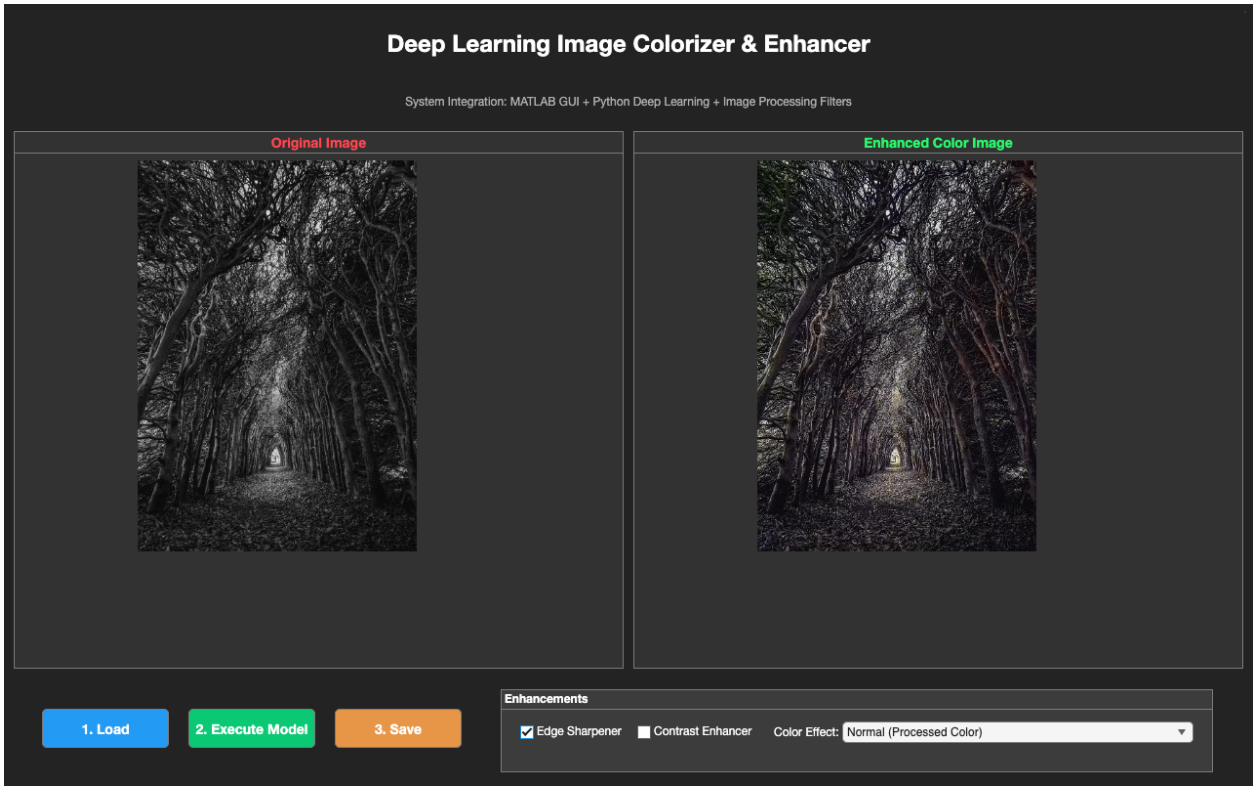
Before Edge Sharpener





After Edge Sharpener



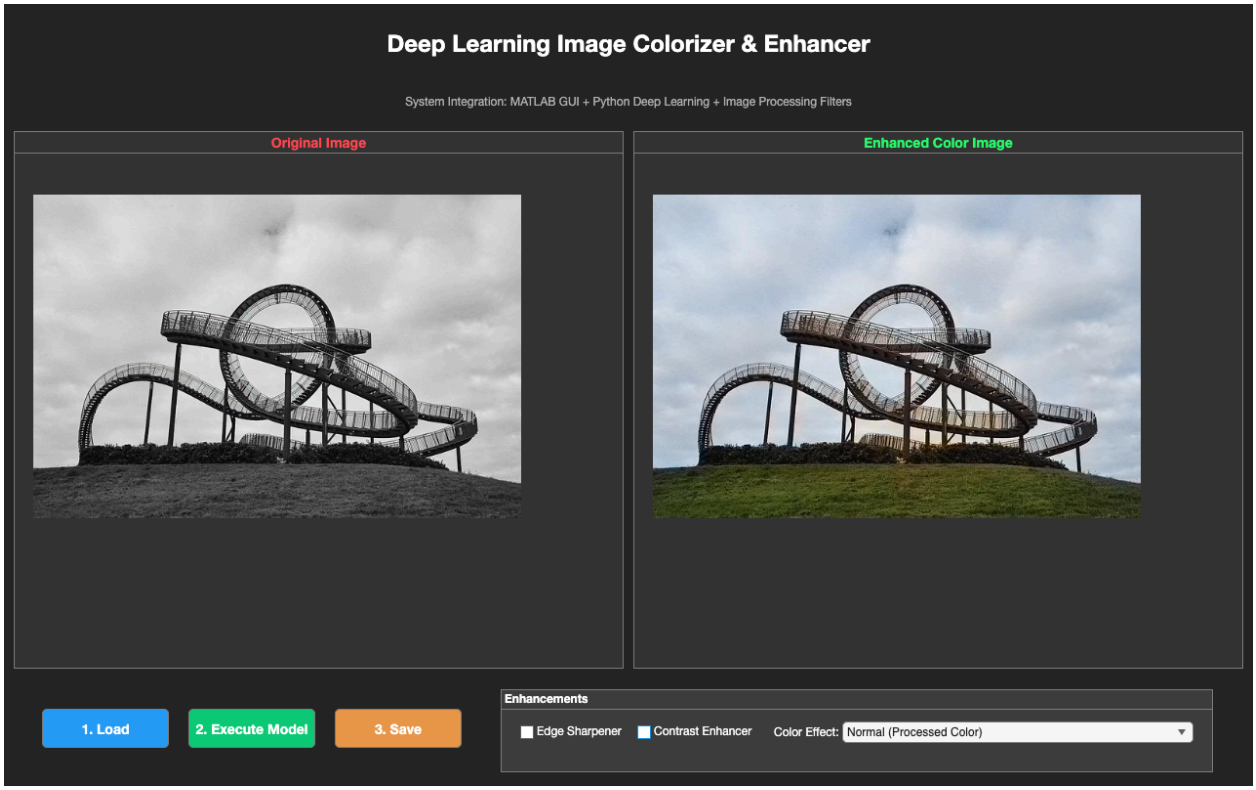


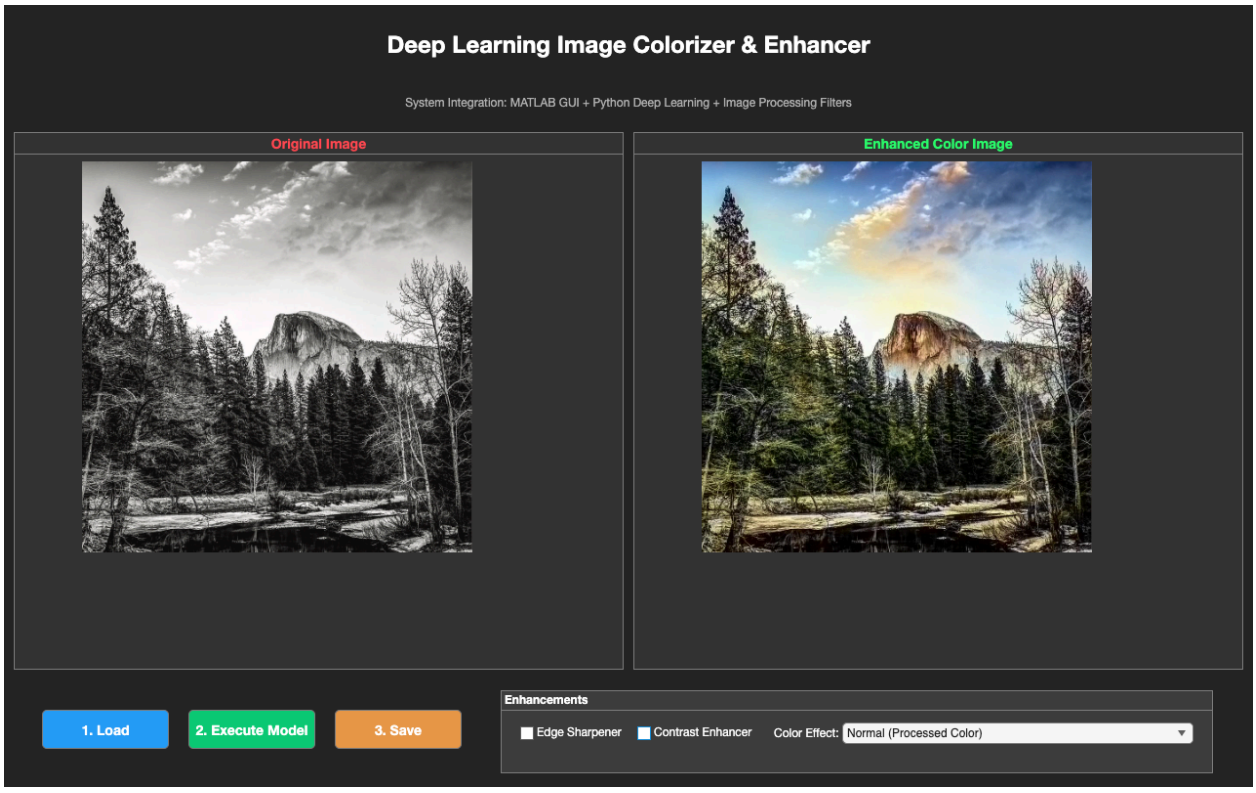
4.2 Feature: Contrast Enhancement (CLAHE)

Raw outputs from neural networks can sometimes appear "washed out" or lacking dynamic range. We implemented **Contrast Limited Adaptive Histogram Equalization (CLAHE)** to fix this.

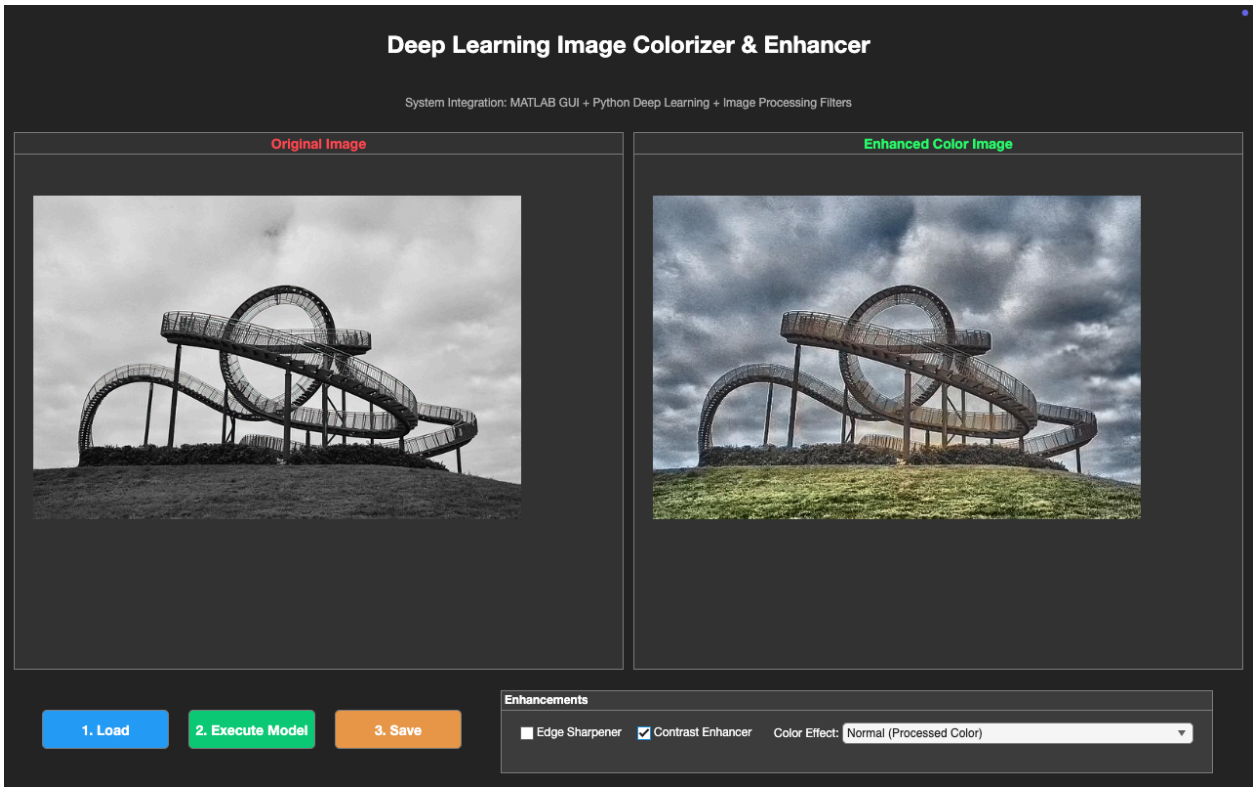
1. **Challenge:** Standard histogram equalization destroys color balance if applied to RGB channels directly.
2. **Solution (Luminance Processing):**
    - Convert the image from RGB to **Lab color space**.
    - Extract the **L (Luminance)** channel.
    - Apply adapthisteq only to the L channel.
    - Merge back with the original *a* and *b* color channels and convert to RGB.
3. **Benefit:** This enhances the lighting and shadow details while keeping the AI-generated colors natural and untouched.

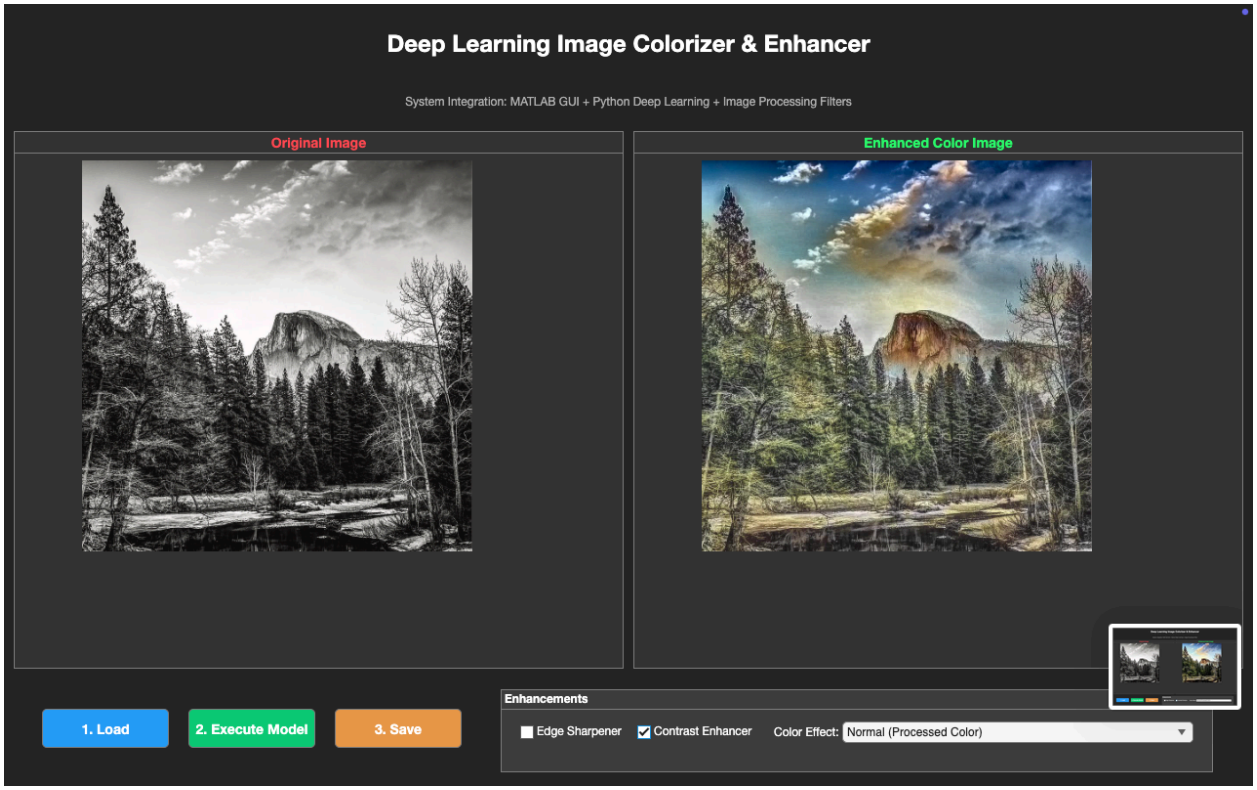
Before Contrast Enhancement





After Contrast Enhancement

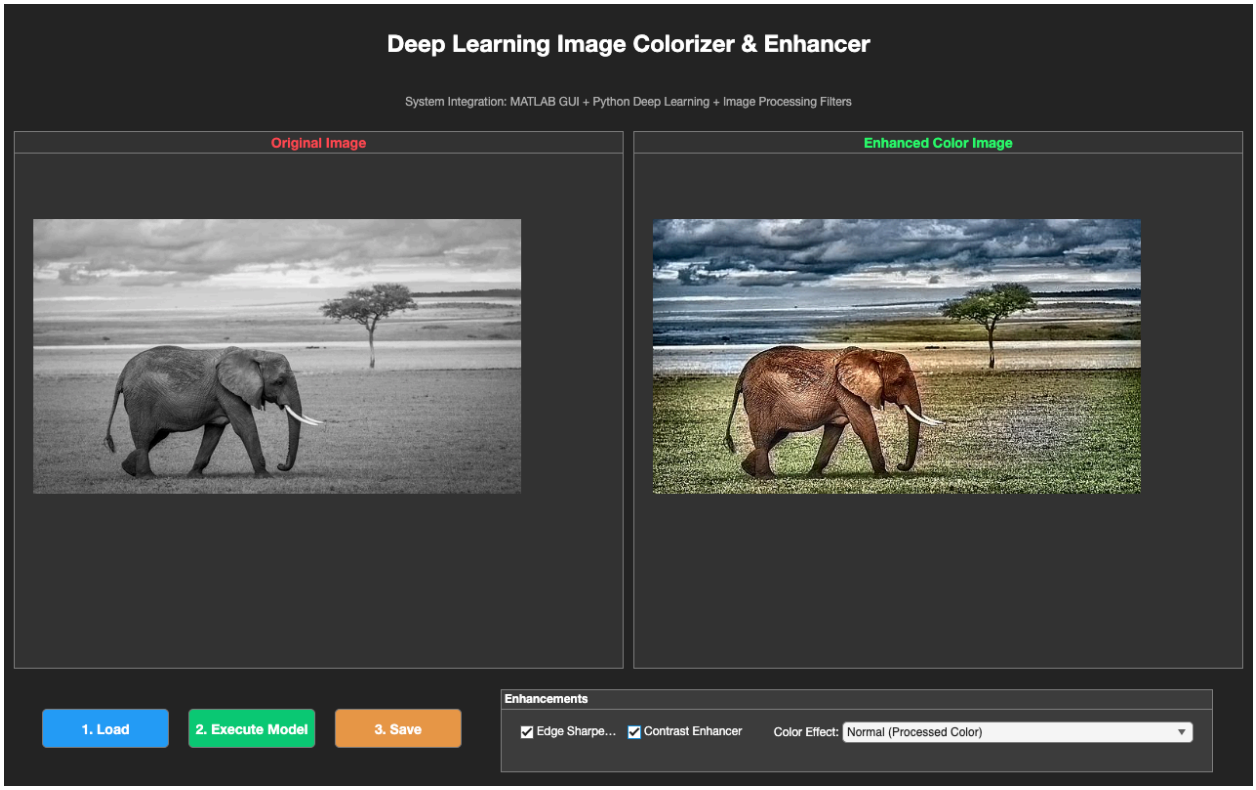


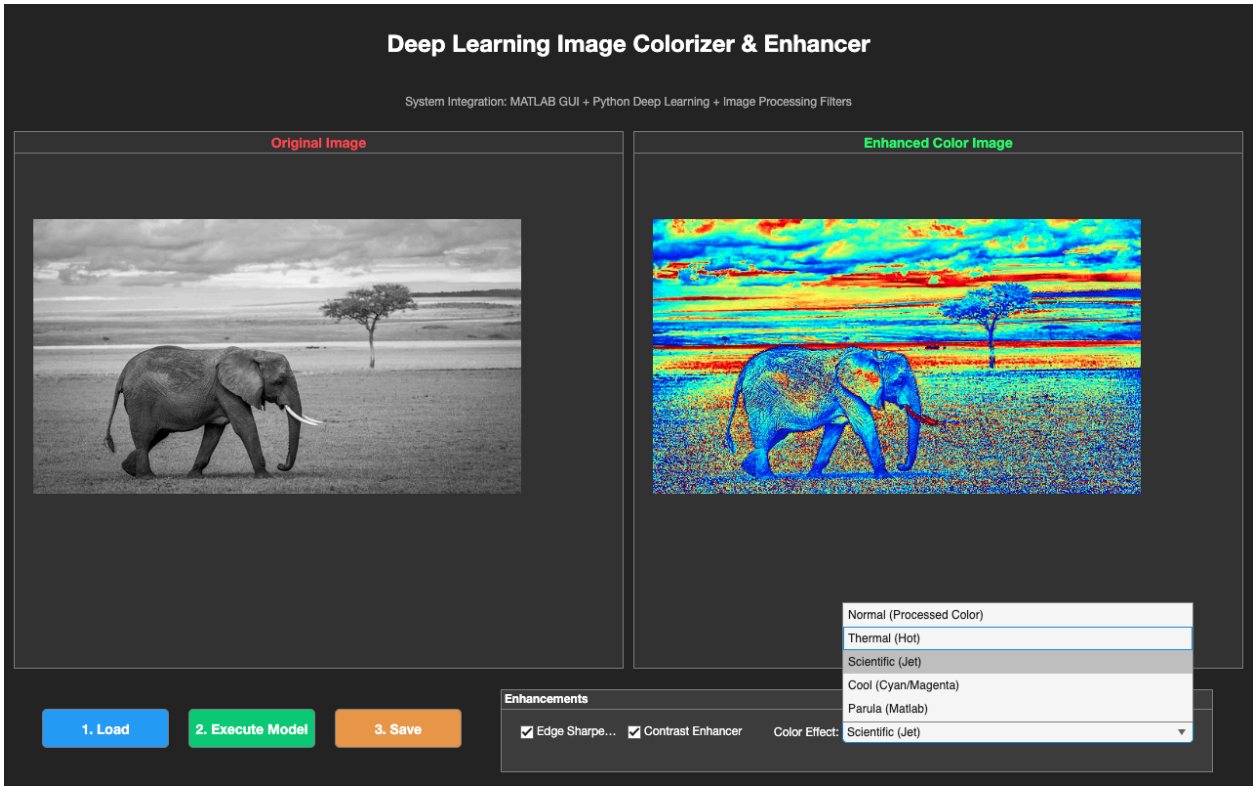


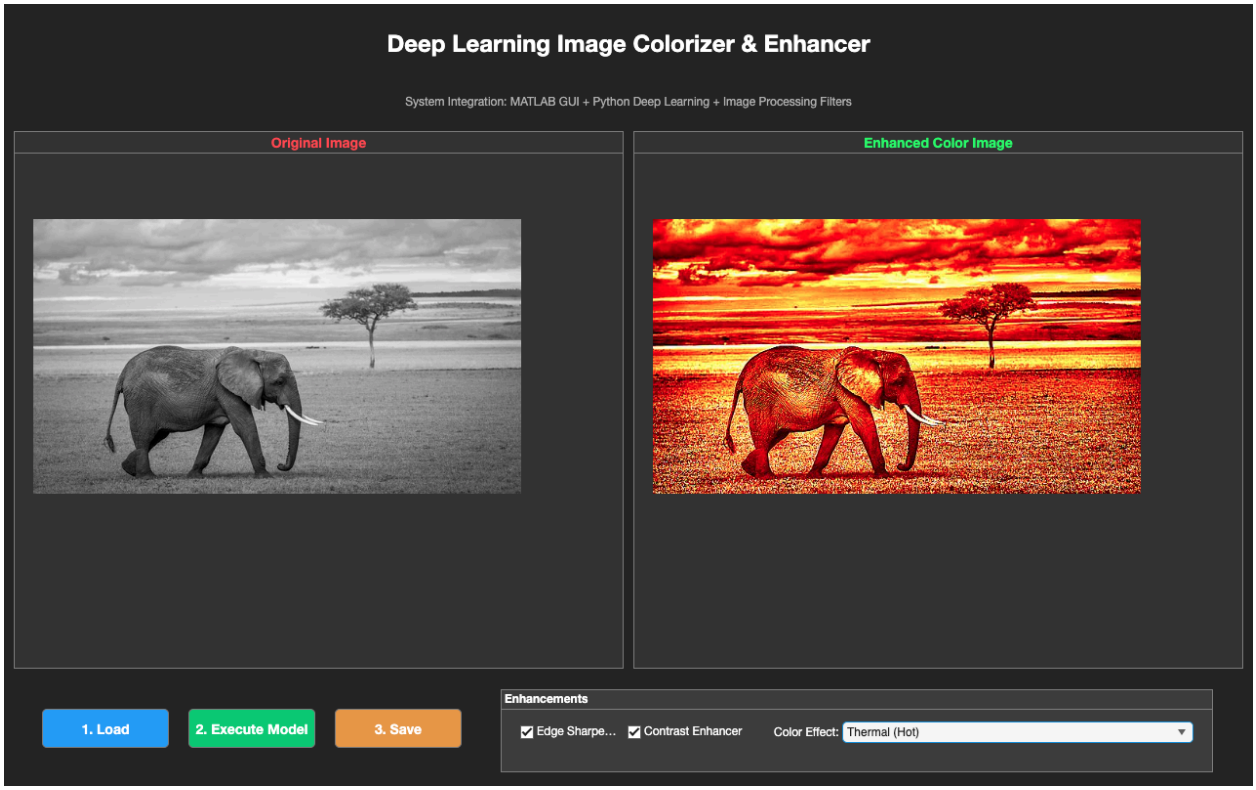
4.3 Feature: Scientific False-Color Mapping

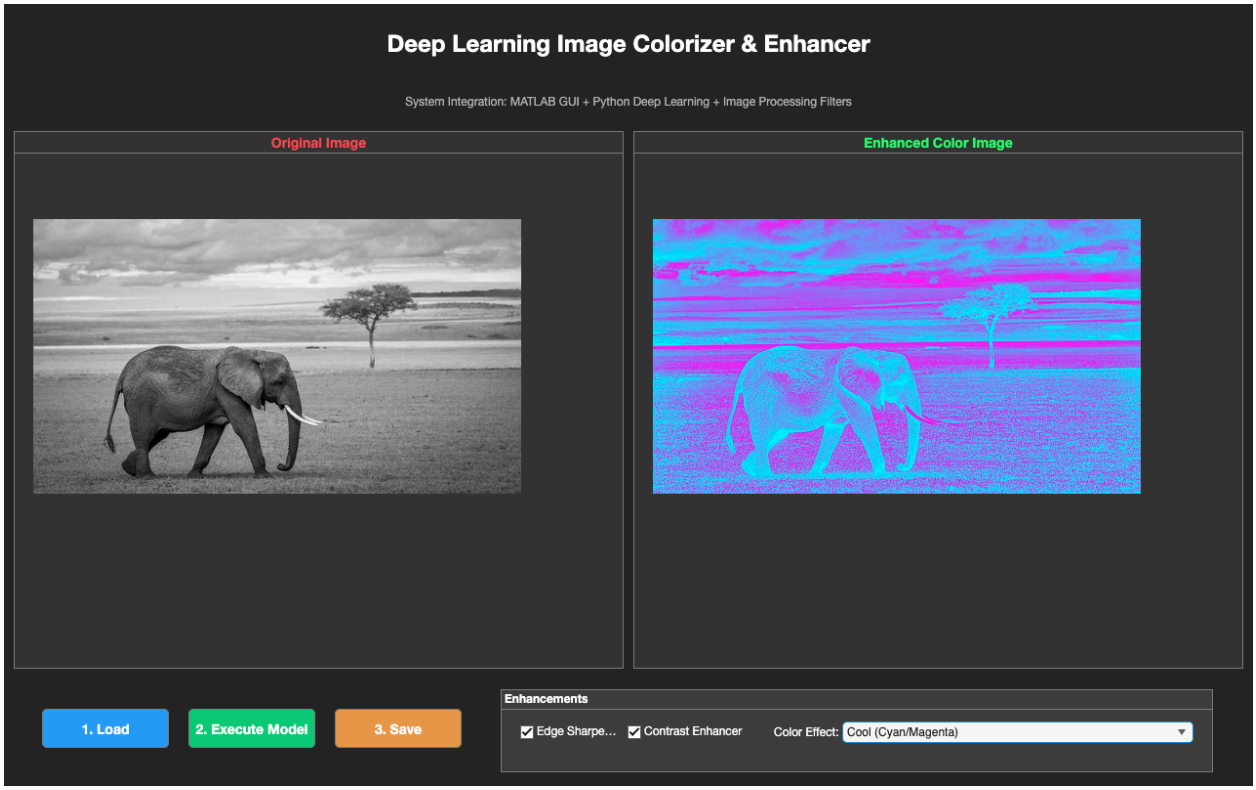
For technical analysis, the application includes a mapping feature that visualizes pixel intensity using scientific colormaps.

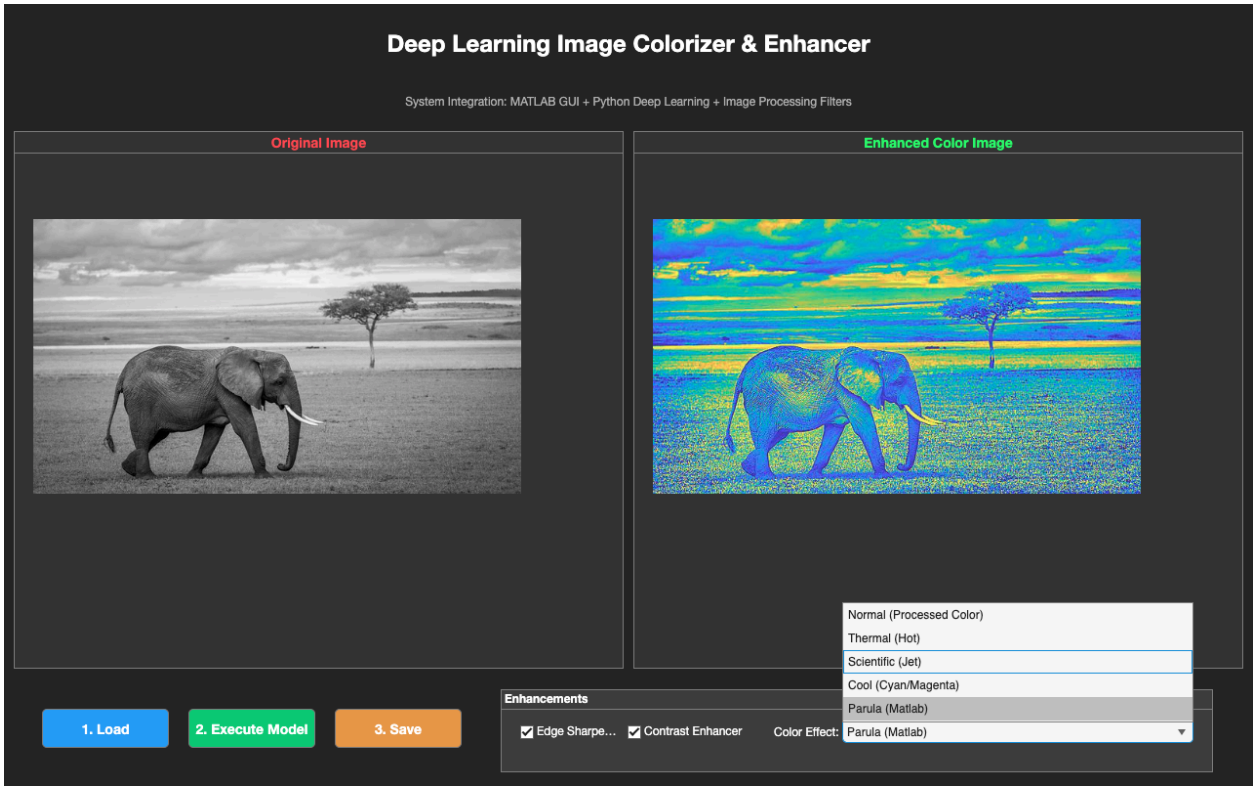
1. **Options:** Thermal (Hot), Scientific (Jet), and Parula.
2. **Utility:** This allows users to visualize the "heat" or intensity distribution of the image, useful for transforming standard photos into thermal-style visualizations for artistic or analytical purposes.




Deep Learning Image Colorizer & Enhancer
System Integration: MATLAB GUI + Python Deep Learning + Image Processing Filters
Original Image
Enhanced Color Image
Normal (Processed Color)
Thermal (Hot)
Scientific (Jet)
Cool (Cyan/Magenta)
Parula (Matlab)
1. Load
2. Execute Model
3. Save
Enhancements
Edge Sharpe...
Contrast Enhancer
Color Effect:
Scientific (Jet)


Deep Learning Image Colorizer & Enhancer
System Integration: MATLAB GUI + Python Deep Learning + Image Processing Filters
Original Image
Enhanced Color Image
1. Load
2. Execute Model
3. Save
Enhancements
Edge Sharpe...
Contrast Enhancer
Color Effect:
Thermal (Hot)



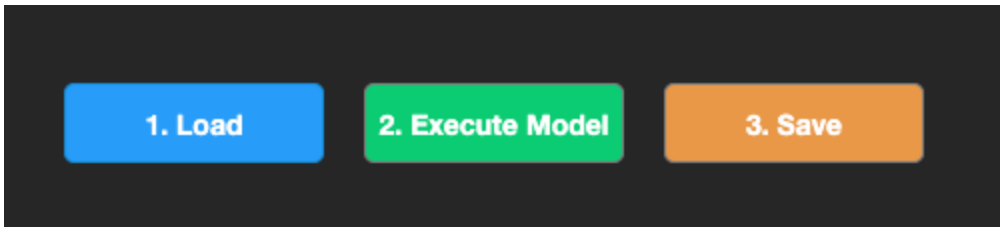


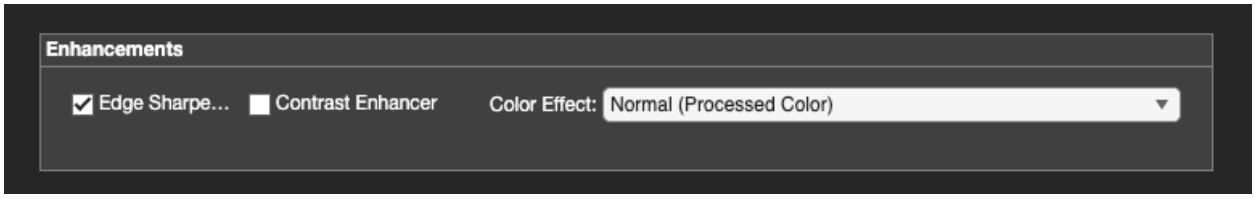
Observation: Application of the 'Scientific (Jet)' colormap on the AI output. This feature enables the analysis of pixel intensity distributions in a scientific context.

4.4 Feature: Robust Load & Save System

To make the software a complete user product, a robust File I/O (Input/Output) system was engineered.

1. **Smart Loading (uigetfile):** The system filters inputs to accept only compatible image formats (.jpg, .jpeg, .png, .bmp) to prevent backend crashes.
2. **Dynamic Timestamping:** To solve a known issue where MATLAB caches old images, the system automatically assigns a unique timestamp (e.g., result_123045.png) to every processed result, ensuring the display always updates instantly.
3. **Final Output Saving (uiputfile):** Users can export the **Final Processed Image** (containing all sharpening and contrast effects) to any location on their disk. The system creates a uint8 matrix of the displayed data and writes it directly to the storage.





5. Implementation Details

5.1 Files Structure

- ColorizerApp.m: The main MATLAB GUI source code handling UI logic and image processing filters.
- backend.py: The Python script that loads the .caffemodel weights and performs the forward pass.
- model/: Directory containing the pre-trained weights:

    - colorization_release_v2.caffemodel (The trained weights from ImageNet).
    - pts_in_hull.npy (The 313 color cluster centers used for quantization).

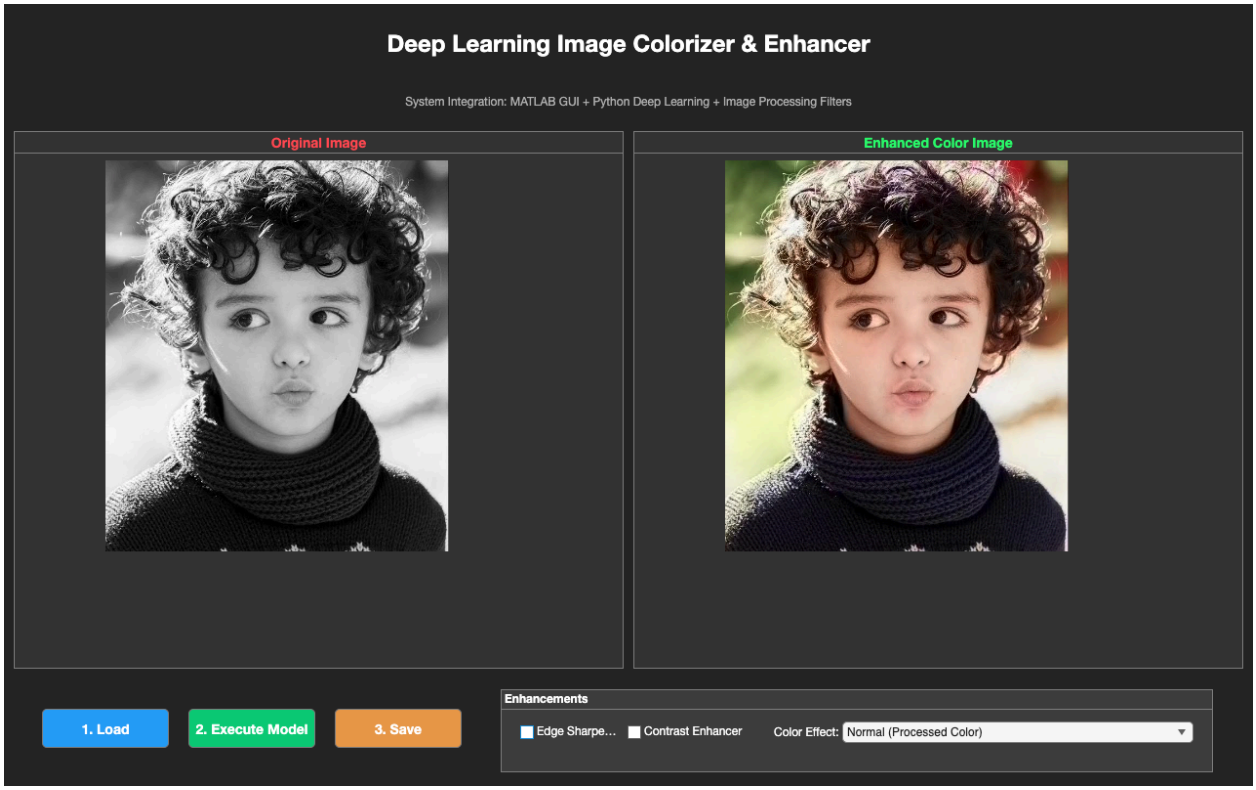
5.2 Tools Used

- **MATLAB R2023a:** App Designer, Image Processing Toolbox.
- **Python 3.9:** OpenCV (cv2), NumPy.

6. Experimental Results

6.1 GUI Interface & Human Subjects

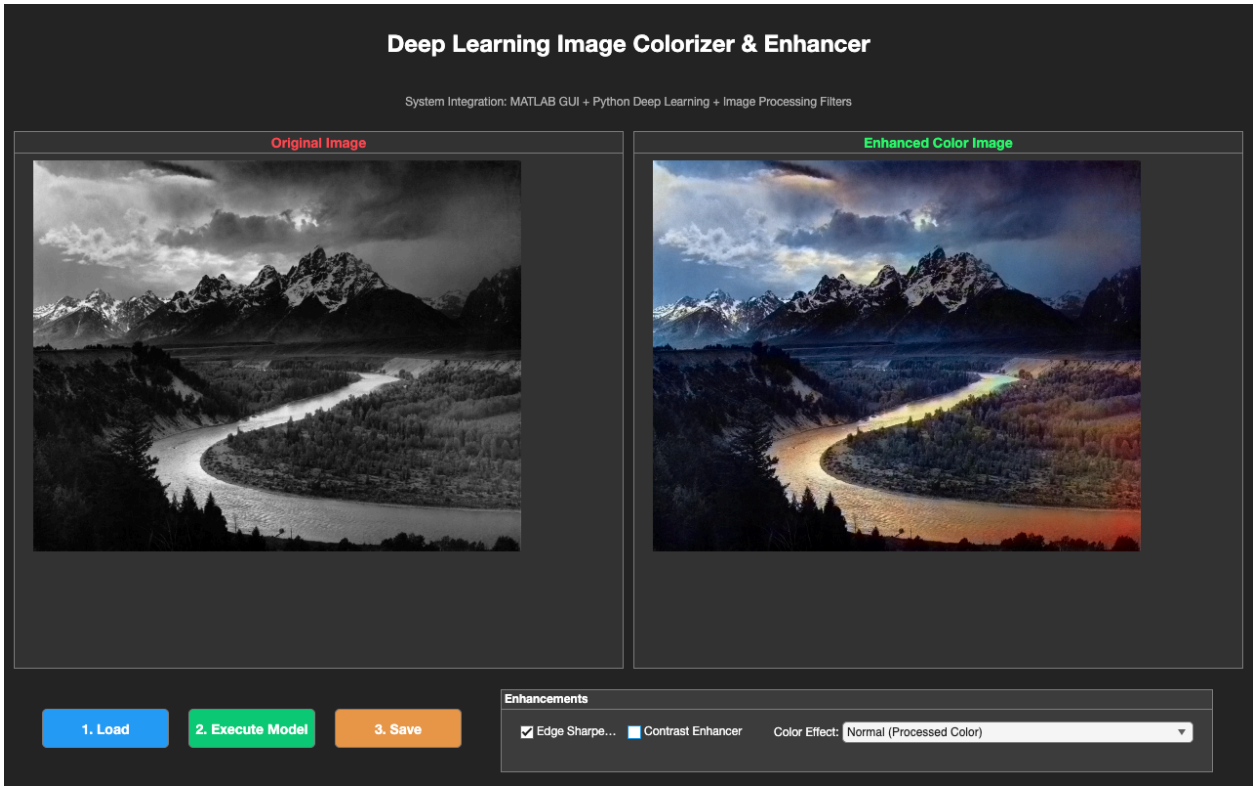
The application correctly identifies skin tones and clothing.

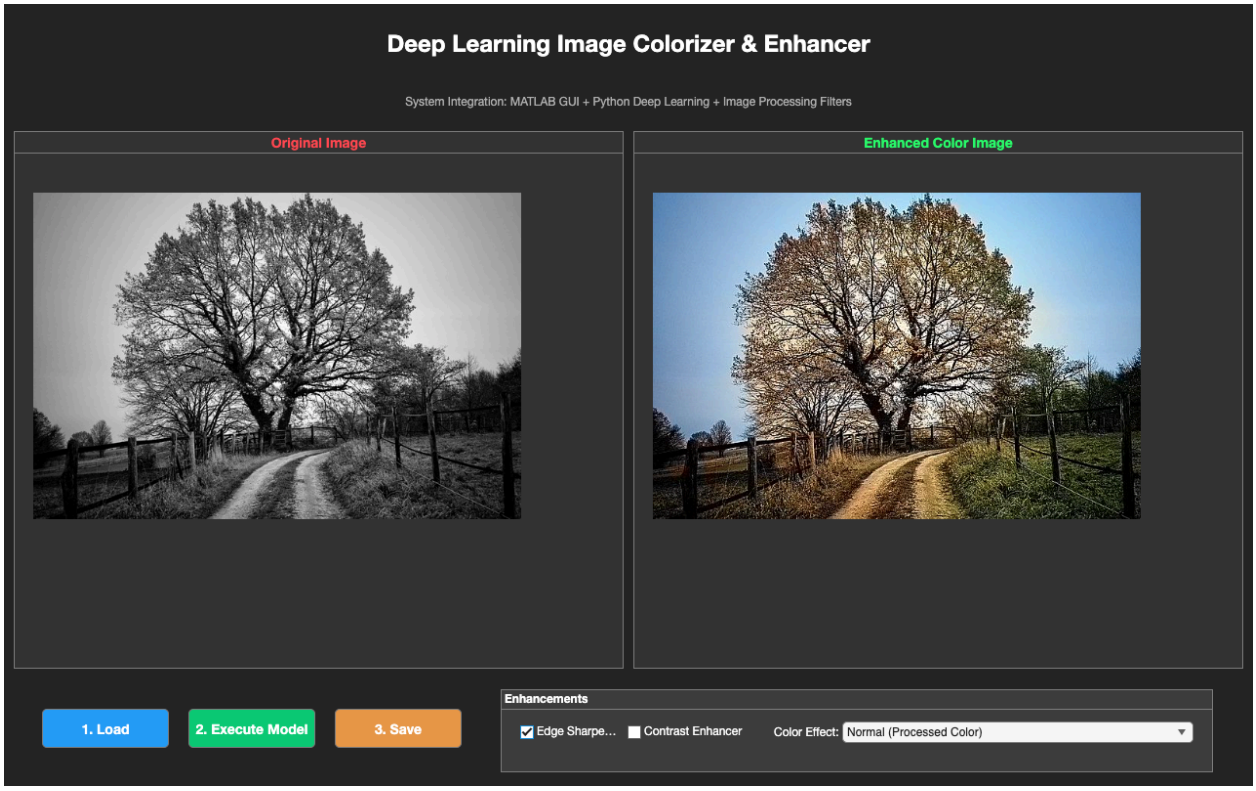


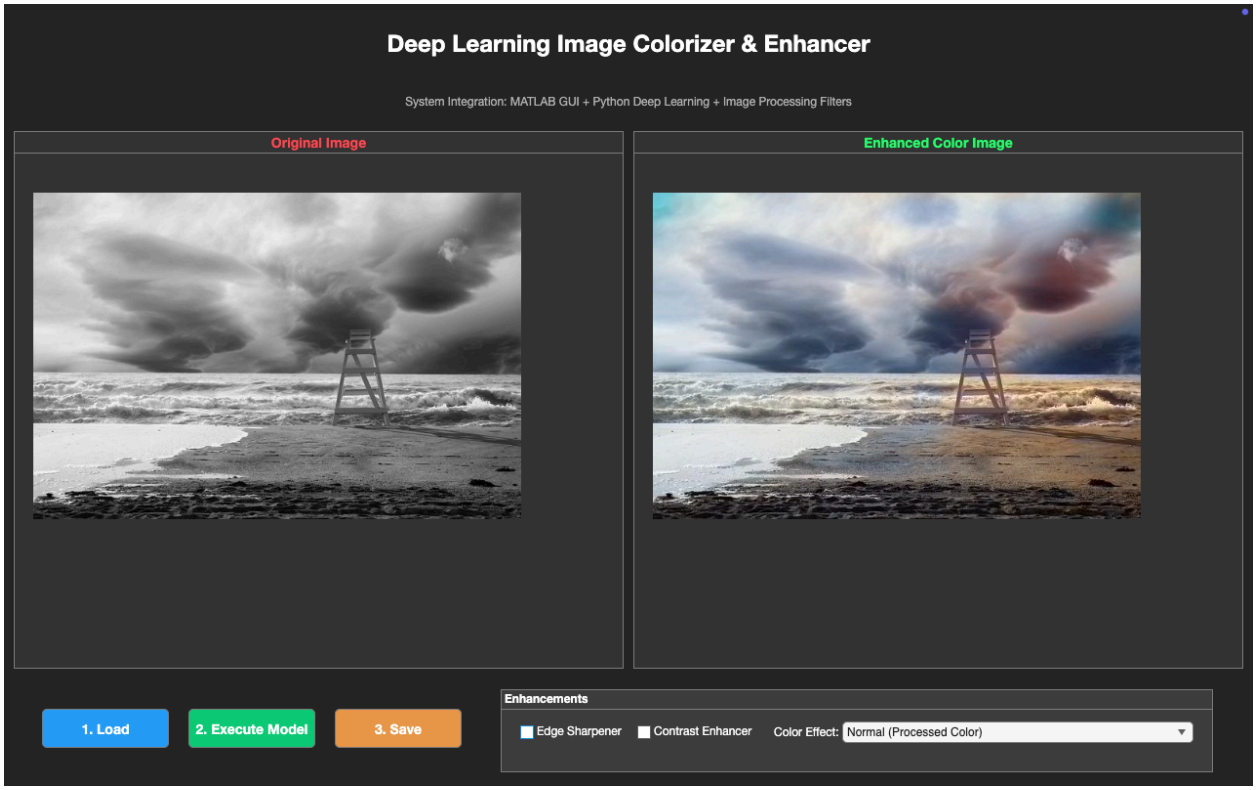
Observation: The model correctly identified the skin texture and hair color, reconstructing a realistic portrait from the grayscale input.

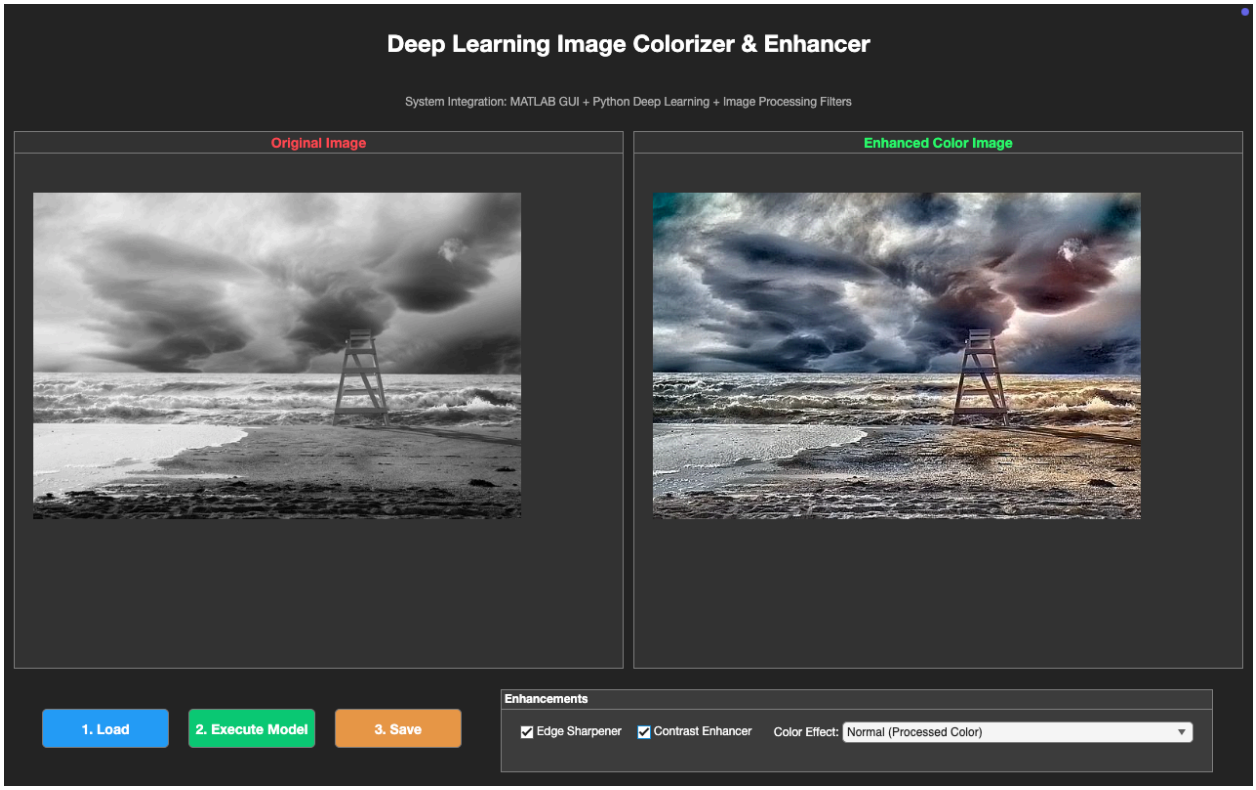
6.2 Landscape & Environment

The model uses semantic context to colorize natural elements.


Deep Learning Image Colorizer & Enhancer
System Integration: MATLAB GUI + Python Deep Learning + Image Processing Filters
Original Image
Enhanced Color Image
1. Load
2. Execute Model
3. Save
Enhancements
Edge Sharpe...
Contrast Enhancer
Color Effect:
Normal (Processed Color)


Deep Learning Image Colorizer & Enhancer
System Integration: MATLAB GUI + Python Deep Learning + Image Processing Filters
Original Image
Enhanced Color Image
1. Load
2. Execute Model
3. Save
Enhancements
Edge Sharpe...
Contrast Enhancer
Color Effect:
Normal (Processed Color)





Observation: The model distinguished between vegetation (green), sky (blue), and ground textures based on the training it received on ImageNet nature photos.

Installation & Setup

1. Set up the Python Environment

Open your terminal/command prompt in the project folder:

Create a virtual environment

python -m venv venv

Activate it (Windows)

venv\Scripts\activate

Activate it (Mac/Linux)

source venv/bin/activate

Install dependencies

pip install numpy opencv-python

2. Configure MATLAB

1. Open `ColorizerApp.m` in MATLAB.
2. Scroll to the `ProcessButtonPushed` function (approx. line 130).

Update the python path to match your computer's virtual environment path:
% Example for Mac

pythonExe = '/Users/yourname/Desktop/DIP/venv/bin/python';

% Example for Windows

% pythonExe = 'C:\Users\yourname\Desktop\DIP\venv\Scripts\python.exe';

How to Run

1. Open MATLAB.
2. Navigate the "Current Folder" to this project directory.
3. In the Command Window, type:
   app = ColorizerApp
4. The GUI will launch.
   - **Step 1:** Click **Load** to select a B&W image.

- **Step 2:** Click **Execute Model** to run the Deep Learning model.
- **Step 3:** Use the checkboxes to apply **Sharpening** or **Contrast**.
- **Step 4:** Click **Save** to export your final result.

📂 Project Structure

```
DIP_Project/
├── ColorizerApp.m          # Main MATLAB GUI source code
├── backend.py              # Python script for model inference
├── model/                  # Folder containing AI weights
│   ├── colorization_release_v2.caffemodel
│   ├── colorization_deploy_v2.prototxt
│   └── pts_in_hull.npy
└── README.md               # This documentation file
```

References

- **Model Architecture:** Zhang, R., Isola, P., & Efros, A. A. (2016). *Colorful Image Colorization*. ECCV.
- **Original Website :**https://richzhang.github.io/colorization/