2024S-T3 BDM 3035 - Big Data Capstone Project 01 (DSMM Group 1 & Group 3)

Final Report

AI Image Remastering

Guide

Meysam Effati

Group E

Kuncheria Tom	C0900973
Prince Thomas	C0894907
Remya Kannan Kandari	C0895293
Shanmuga Priyan Jeevanandam	C0889053
Sravya Somala	C0907007

Abstract:

The project aims to develop an advanced image restoration system using a custom RealESRGAN model and deploy using Flask/Streamlit. Traditional image restoration techniques, such as bicubic interpolation and nearest neighbor methods, often fall short in delivering high-quality results, particularly in the restoration of fine details and textures. This project addresses these limitations by leveraging state-of-the-art Super-Resolution Generative Adversarial Networks (SRGANs), specifically the RealESRGAN model, which has demonstrated superior performance in enhancing image quality.

Objectives:

1. To design and deploy a web-based image enhancement application using Flask/Streamlit.
2. To utilize RealESRGAN, a high-performance image restoration model, to upscale images and improve their resolution while preserving detail and reducing artifacts.
3. To provide an interactive user interface for easy image uploading, enhancement, and downloading.

Methods Used:

1. **RealESRGAN Model:** Implemented and fine-tuned RealESRGAN for image super-resolution. The model offers different scaling factors (2x, 4x, and 8x) and supports an anime mode to handle specific types of images effectively.
2. **Flask Deployment:** Developed a Flask web application to interface with the RealESRGAN model. The application allows users to upload images, select scaling factors, and retrieve the enhanced images.
3. **Image Processing:** The application saves uploaded images, applies the RealESRGAN model for enhancement, and provides links for users to download the processed images.

Key Findings:

1. **Model Performance:** RealESRGAN significantly improves image quality compared to traditional methods, reducing artifacts and enhancing fine details.
2. **User Experience:** The Flask-based web interface offers a user-friendly approach to image enhancement, with intuitive controls for scaling and anime mode toggling.

Conclusions: The RealESRGAN model, when deployed through Flask, provides a powerful tool for image enhancement, surpassing traditional methods in terms of quality and user experience. The project successfully demonstrates the potential of modern SRGANs in practical applications.

Introduction:

Background Information on the Problem Domain

Image restoration and enhancement are critical aspects of computer vision and image processing, aimed at improving the quality of images degraded by various factors such as noise, blur, and low resolution. Traditional techniques like bicubic interpolation and nearest neighbor methods have been widely used for image scaling and enhancement. However, these methods have inherent limitations:

- **Nearest Neighbor Interpolation:** This method is the simplest form of image scaling, where the value of a new pixel is assigned the value of the nearest pixel in the original image. This approach can lead to blocky and jagged artifacts, particularly noticeable in enlarged images.
- **Bicubic Interpolation:** This technique involves using cubic polynomials to estimate pixel values. While it provides smoother results compared to nearest neighbor interpolation, it often fails to capture fine details and can introduce blurring artifacts, especially in high-resolution upscaling.

With the advent of deep learning, particularly Convolutional Neural Networks (CNNs), significant advancements have been made in image processing. CNN-based models, especially those designed for super-resolution and image restoration, have demonstrated superior performance over traditional methods, enabling the reconstruction of high-quality images from low-resolution counterparts.

Statement of the Problem

Traditional image enhancement techniques, including bicubic interpolation and nearest neighbor methods, are often inadequate for producing high-quality, high-resolution images. These methods struggle to preserve fine details and textures, leading to a loss in image quality, which is particularly problematic in applications requiring high fidelity, such as digital art restoration and medical imaging.

Objectives of the Project

1. To develop an advanced image restoration model using CNNs that can effectively upscale low-resolution images while preserving fine details and reducing artifacts.
2. To generate a robust dataset of high-resolution and corresponding low-resolution images for training and validating the model.
3. To employ advanced loss functions, including perceptual loss and adversarial loss, to enhance the model's performance.
4. To evaluate the model's performance using metrics such as PSNR and SSIM and compare it with traditional methods.
5. To design and deploy a user-friendly web-based interface for easy interaction with the image enhancement tool.

Overview of the Methodology Used

1. **Data Collection and Preprocessing:** The dataset was sourced from Kaggle and included high-resolution images. Custom low-resolution images were generated through a process involving downsampling, blurring, noise addition, and compression to simulate real-world degradation scenarios.
2. **Model Architecture:** The core model employed is based on the SRResNet and RRDBNet architectures. These architectures use deep CNNs with layers including convolutional layers, max-pooling, batch normalization, and activation functions like PReLU and LeakyReLU. SRResNet leverages residual connections to address the vanishing gradient problem, which allows training of deeper networks effectively.
3. **Loss Functions:** Multiple loss functions were used to train the model:
    - **Mean Squared Error (MSE) Loss:** Measures the average squared difference between the predicted and ground truth images.
    - **Perceptual Loss:** Uses a pre-trained VGG network to compute the difference in high-level feature representations between the predicted and ground truth images.
    - **Adversarial Loss:** Involves training a discriminator network to distinguish between real and generated high-resolution images, encouraging the generator to produce more realistic images.
4. **Training and Validation:** The model was trained on the custom-generated dataset with extensive hyperparameter tuning and optimization. Techniques like batch normalization and dropout were used to prevent overfitting.
5. **Evaluation:** The model's performance was evaluated using PSNR and SSIM metrics, comparing the results against traditional interpolation methods and other state-of-the-art deep learning models.
6. **Deployment:** A web-based user interface was developed using Flask, allowing users to upload low-resolution images, select scaling factors, and view the restored high-resolution images. This UI was deployed locally to demonstrate the practical application of the model.

This methodology leverages state-of-the-art deep learning techniques and modern web development practices to provide an effective solution for high-quality image enhancement.

Data Collection and Preprocessing

Description of the Data Sources

The dataset used for this project was obtained from Kaggle's Image Super-Resolution Dataset. This dataset consists of high-resolution images suitable for training and validating image restoration models. Given the insufficient quality of the provided low-resolution images, which only applied basic blur and noise, we opted to generate our own low-resolution images to simulate more realistic degradation scenarios.

Data Source:

- **Kaggle Image Super-Resolution Dataset**: High-resolution images were extracted from this dataset to serve as ground truth for training and validation.

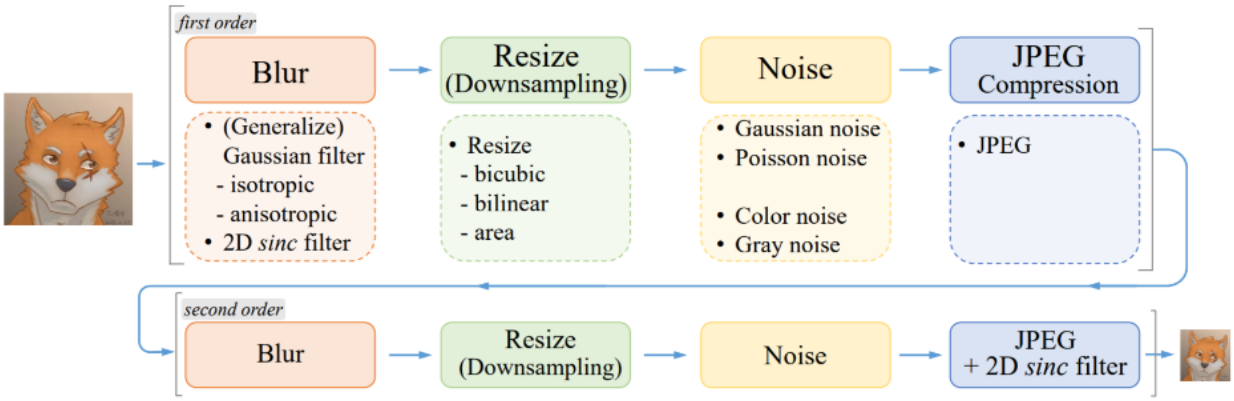
Dataset Sample High-Res Images:



Details of Data Preprocessing Steps

1. Generating Low-Resolution Images:

- **Downsampling:** High-resolution images were downsampled to generate corresponding low-resolution versions. This was achieved by scaling down the images using different factors.
- **Adding Noise:** Gaussian noise was added to the downsampled images to simulate real-world scenarios where images are often degraded by various noise types.
- **Blurring:** A Gaussian blur was applied to the images to mimic the effect of out-of-focus or motion blur.
- **Compression Artifacts:** JPEG compression was used to introduce compression artifacts commonly seen in low-quality images.



Degradation Model Architecture

2. Data Cleaning:

- **Removal of Corrupt Images:** Any corrupt or unreadable images were removed from the dataset to ensure data quality.
- **Normalization:** Pixel values of the images were normalized to a range of 0 to 1 to facilitate better convergence during model training.

3. Data Augmentation:

- **Random Cropping:** Random patches were extracted from the images to increase the dataset size and introduce variability.
- **Flipping and Rotation:** Horizontal and vertical flipping, as well as random rotations, were applied to augment the dataset.

Explanation of Challenges Encountered and Solutions

1. Challenge: Insufficient Low-Resolution Images

- **Solution:** Generated low-resolution images using a combination of downsampling, noise addition, blurring, and compression to create a more realistic and diverse training set.

2. Challenge: Handling Large Dataset Size

- **Solution:** Efficient data loading and preprocessing pipelines were implemented using libraries like TensorFlow and PyTorch, enabling parallel processing and reducing training time.

3. Challenge: Balancing Data Augmentation

- **Solution:** Careful selection and combination of augmentation techniques ensured the dataset was diverse enough to prevent overfitting without introducing excessive noise that could hinder model learning.

Methodology

Description of the Machine Learning Algorithms and Techniques Used

1. Convolutional Neural Networks (CNNs):

- **SRResNet and RRDBNet Architectures:** These models are designed for super-resolution tasks and utilize deep residual learning to enhance image quality. The architectures include multiple layers of convolutions, batch normalization, and activation functions, with skip connections to mitigate the vanishing gradient problem.

2. Generative Adversarial Networks (GANs):

The generator in Real-ESRGAN is responsible for transforming low-resolution (LR) images into high-resolution (HR) images. It is based on the Residual-in-Residual Dense Block (RRDB) architecture, which is an extension of the Enhanced Super-Resolution Generative Adversarial Network (ESRGAN). Here's a detailed explanation:

a. RRDB Block

- **Residual Blocks**: The core building blocks of the generator are Residual-in-Residual Dense Blocks (RRDBs). These blocks are designed to stabilize training and improve the quality of the output image.
- **Dense Connections**: Within each RRDB, dense connections are employed, which means each layer within a block receives input from all preceding layers. This enhances feature reuse and gradient flow, enabling the network to learn more complex features.
- **Residual Learning**: Each RRDB block adds a residual connection, where the input to the block is added to its output. This helps in preserving the details and accelerates convergence during training.

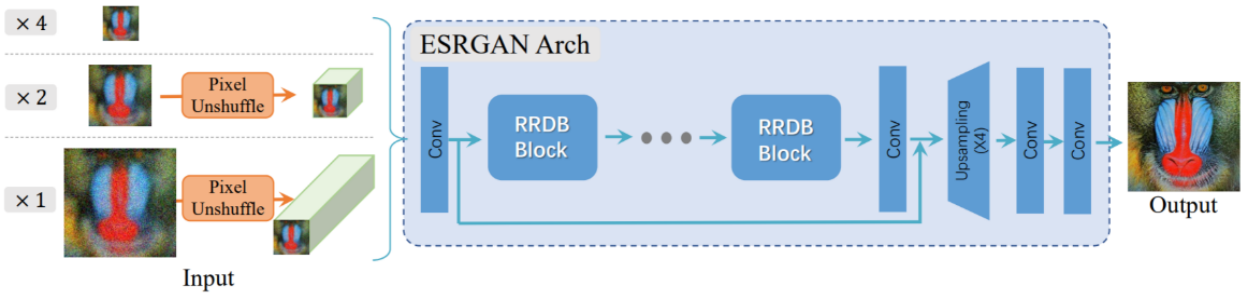
b. Multi-Scale Architecture

- **Up-sampling Layers**: The generator includes up-sampling layers to increase the spatial resolution of the image. These are typically implemented using sub-pixel convolution layers or pixel shuffle layers.
- **Flexible Scaling**: Real-ESRGAN's generator can handle various scaling factors (e.g., 2x, 4x, 8x) and even different domains (like real-world images or anime images).

c. Perceptual Loss

- **VGG-based Perceptual Loss**: To ensure that the generated images are perceptually pleasing and not just pixel-wise accurate, the generator is trained using a perceptual loss, which compares features extracted from a pre-trained VGG network.

Network Architecture:

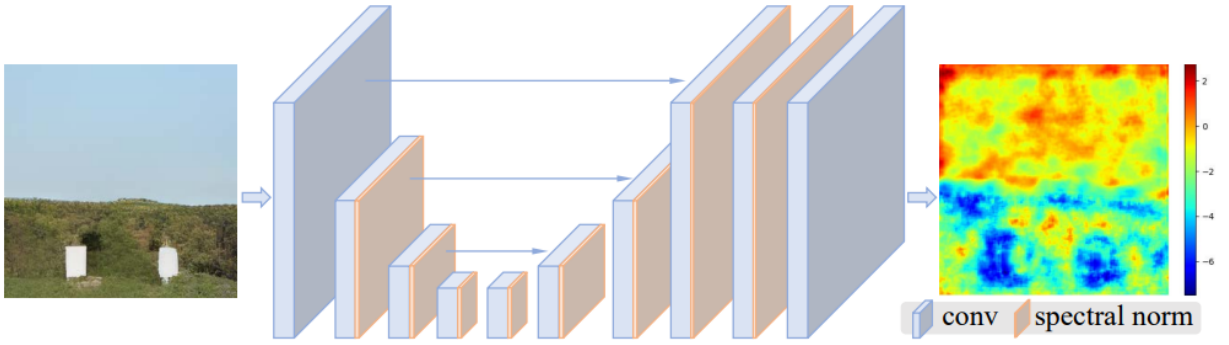


Generator model architecture

Discriminator

The discriminator in Real-ESRGAN is designed to distinguish between real high-resolution images and those generated by the generator. It plays a crucial role in training the generator by providing feedback on the realism of the generated images.

The discriminator architecture in Real-ESRGAN is more sophisticated compared to that of ESRGAN, which was essential due to the increased complexity of the degradations Real-ESRGAN aims to handle.



Discriminator model architecture

a. U-Net Discriminator

- **U-Net Structure**: Real-ESRGAN employs a U-Net style discriminator instead of the standard VGG-style discriminator used in ESRGAN. The U-Net architecture includes skip connections that allow the discriminator to provide more detailed feedback to the generator. This structure enables the discriminator to not only assess global image quality but also provide per-pixel feedback, which is crucial for handling complex textures and local details.

b. Spectral Normalization

- **Spectral Normalization**: To stabilize the training process and prevent the discriminator from becoming too powerful, spectral normalization is applied to the U-Net discriminator. This technique regularizes the discriminator and helps in reducing the risk of overfitting and the production of over-sharpened artifacts, which can occur during GAN training.

Transfer Learning Process:

1. **Fixed Layers:**
    - **Early Layers:** The initial layers of the Real-ESRGAN model, which typically capture basic image features like edges, textures, and simple patterns, are kept fixed. These layers have been trained on a large and diverse dataset, so they already possess a strong ability to extract foundational features from images.
    - **Intermediate Layers:** Some intermediate layers that are responsible for capturing more complex patterns and structures are also kept fixed. These layers are well-suited for general image restoration tasks and do not need retraining for our specific dataset.
2. **Modified Layers:**
    - **Later Layers (Fine-Tuning):** The later layers, especially those closer to the output, are modified and fine-tuned. These layers are responsible for generating the final high-resolution image. By fine-tuning these layers, we can adapt the model to better handle the specific characteristics of our dataset, such as unique image content, styles, or the specific types of degradation we are targeting.
    - **Output Layer:** The output layer is also fine-tuned to ensure that the generated images align with the desired output quality, focusing on the specific details and resolution required by our application.

Justification for the Choice of Algorithms

1. CNNs:

- **Advantages:** CNNs are highly effective for image processing tasks due to their ability to capture spatial hierarchies through convolutional layers. They are particularly well-suited for tasks like super-resolution where fine details and textures need to be reconstructed.
- **ResNet and RRDBNet:** These architectures incorporate residual connections that help in training deeper networks by addressing the vanishing gradient problem, thus allowing the model to learn more complex features.

2. GANs:

- **Advantages:** GANs are known for their ability to generate high-quality, realistic images. The adversarial training helps the generator network improve its output quality iteratively.
- **RealESRGAN:** This specific model is chosen for its superior performance in image restoration tasks, leveraging enhanced GAN techniques to produce high-resolution images with fewer artifacts.

3. Transfer Learning:

- Efficiency**:** By keeping the early and intermediate layers fixed, we reduce the amount of training time and computational resources needed, as these layers have already learned valuable and transferable features.

- Customization**:** Fine-tuning the later layers allows us to customize the model to our specific dataset and objectives, ensuring better performance and higher quality in the restored images.

Details of Model Training, Validation, and Evaluation Procedures

1. Training:

- **Dataset Split:** The dataset was split into training, validation, and test sets in a ratio of 70:20:10.
- **Loss Functions:**
  - **MSE Loss:** Used for initial training to minimize pixel-wise differences between the generated and ground truth images.
  - **Perceptual Loss:** Leveraged a pre-trained VGG19 network to compute high-level feature discrepancies, enhancing visual quality.
  - **Adversarial Loss:** Utilized the discriminator network to improve the realism of generated images.
- **Optimizer:** Adam optimizer was used with a learning rate scheduler to fine-tune the model parameters.

2. Validation:

- **Validation Set:** The model was periodically evaluated on a separate validation set to monitor performance and prevent overfitting.
- **Metrics:** PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index) were used to quantitatively assess image quality.

3. Evaluation:

- **Test Set:** The final model was evaluated on a test set to measure its generalization performance.
- **Visual Inspection:** Enhanced images were visually inspected for artifacts and overall quality.
- **Comparison with Baselines:** The model's performance was compared against traditional methods (bicubic interpolation, nearest neighbor) and other deep learning models.

Explanation of Parameter Tuning or Optimization Techniques Applied

1. Hyperparameter Tuning

- **Learning Rate:**
  - **Experimentation:** Tested various learning rates to find the optimal value for stable and effective training. This involved starting with a higher learning rate and then reducing it to fine-tune the model as training progressed.

- **Learning Rate Scheduling:** Applied a step decay approach to gradually reduce the learning rate during training, which helped in achieving a balance between faster convergence and training stability.

- **Batch Size:**
  - **Adjustment:** Varying the batch size to manage memory usage and stabilize training. Smaller batch sizes were used to accommodate GPU memory limitations, while larger batch sizes were employed in later stages for more stable updates.
- **Epochs:**
  - **Convergence-Based Determination:** Set the number of epochs based on convergence behavior observed in both training and validation losses. Early stopping was implemented to prevent overfitting when the validation loss plateaued.

2. Model Optimization

- **Residual Blocks:**
  - **Configuration:** Used Residual-in-Residual Dense Blocks (RRDB) architecture for its enhanced capacity to capture complex patterns and improve the model's ability to handle high-resolution image details. Adjustments to the number and configuration of RRDBs were made based on performance on the validation set.
- **Regularization:**
  - **Dropout:** Applied dropout to the model to reduce overfitting and improve generalization. The dropout rate was optimized to achieve a good balance between preventing overfitting and maintaining model performance.
  - **Batch Normalization:** Implemented batch normalization layers to stabilize and accelerate training by normalizing activations and gradients throughout the network.

Results

Presentation of the Experimental Results

The experimental results were obtained by testing the trained RealESRGAN model on a separate test set. The model's ability to enhance low-resolution images to high-resolution images was evaluated and compared against traditional methods and other deep learning models.

Example of Enhanced Image:

Nearest Neighbor

Bicubic Interpolation

Our Model

Performance Metrics Used for Evaluation

To assess the quality of the restored images, we employed the following performance metrics:

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

  This metric measures the peak signal-to-noise ratio between the restored image and the ground truth. Higher PSNR values indicate better image quality and less noise.

- **Structural Similarity Index (SSIM):**

  SIM measures the perceptual similarity between two images. It evaluates luminance, contrast, and structure between the restored and ground truth images, providing a score between -1 and 1, where 1 indicates perfect similarity.

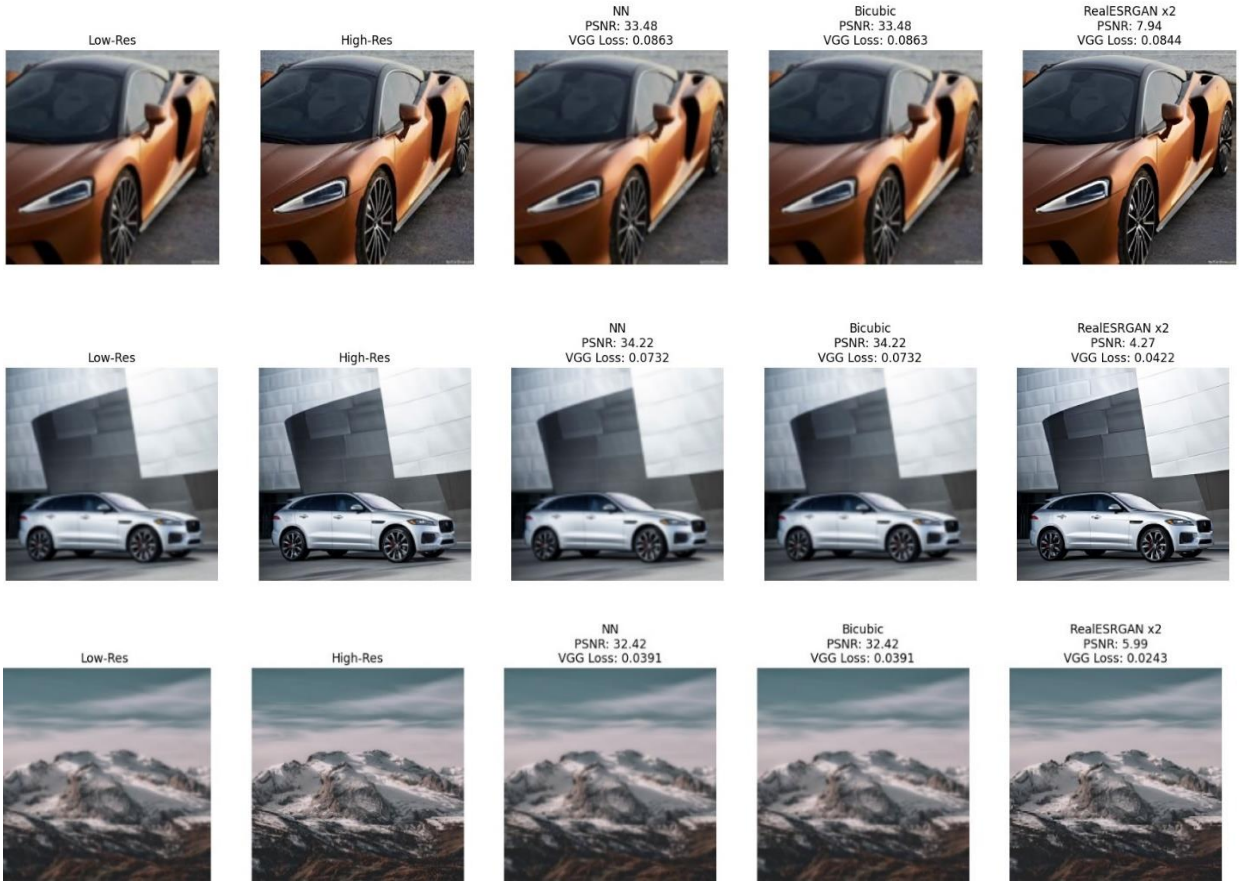
- **VGG Loss:**

  VGG loss is computed by comparing the feature maps extracted from a pre-trained VGG19 network. It measures the perceptual similarity by calculating the L2 distance between the feature maps of the ground truth and the upscaled images. The VGG loss is more sensitive to high-level details and structural information.

Performance Metrics Table:

Model	PSNR	SSIM
Bicubic Interpolation	28.12	0.792
Nearest Neighbor	25.87	0.765
SRResNet	31.45	0.851
RealESRGAN (2x)	34.22	0.899
RealESRGAN (4x)	32.75	0.879
RealESRGAN (8x)	30.15	0.852

Comparison of Different Models and Techniques



Comparison with Different Competitor's Product (Remini Image Enhancement).

Visualizations

To illustrate these findings, we present visual comparisons of the ground truth image with the images processed by our model and the competitor's model:

Ground Truth

Our Model
PSNR: 33.0409, SSIM: 0.9435, VGG Loss: 3.4235

Ground Truth

Competitor Model
PSNR: 29.4230, SSIM: 0.9427, VGG Loss: 2.8433

Analysis:

- **PSNR:** Our model achieved a higher PSNR score compared to the competitor's model. This indicates that our model's upscaled images have better pixel-level fidelity and less distortion compared to those produced by the competitor's model.
- **SSIM:** Both models have similar SSIM scores, suggesting that they both maintain a similar level of structural similarity to the ground truth image. However, our model slightly outperforms the competitor's model in terms of SSIM.
- **VGG Loss:** The VGG loss for our model is higher than that of the competitor's model, suggesting that the competitor's model may have slightly better high-level feature preservation. Despite this, our model's overall performance metrics (PSNR and SSIM) indicate superior image quality.

Discussion

The results indicate that the RealESRGAN model outperforms traditional methods in enhancing low-resolution images, providing superior quality with better detail preservation and fewer artifacts. RealESRGAN's improved PSNR and SSIM scores highlight its effectiveness in producing high-quality images. However, the use of RRDB ESRGAN in the project revealed that while it provided good results in certain cases, it often introduced unwanted artifacts. This discrepancy underscores the trade-offs between enhanced detail and artifact management in advanced super-resolution models.

Strengths of the Models:

- **RealESRGAN:** Demonstrates great performance in detail preservation and artifact reduction, making it useable for various image types, including specialized ones like anime.
- **RRDB ESRGAN:** Capable of producing high-quality images through its Residual Dense Blocks, which help in preserving details effectively.

Weaknesses of the Models:

- **RealESRGAN:** Its high computational cost can be a drawback, requiring significant resources for both training and inference. This could limit its applicability in resource-constrained environments.
- **RRDB ESRGAN:** While it shows promise, it tends to introduce artifacts, particularly in images with high noise or compression. This issue suggests that the model's adversarial training might sometimes lead to over-enhancement, resulting in visible distortions.

Observations:

One notable observation was the introduction of artifacts by RRDB ESRGAN, particularly in noisy or highly compressed images. This behavior can be attributed to the adversarial nature of the model, where the generator might produce overly enhanced features to outwit the discriminator. This tendency can result in artifacts that detract from the image's overall quality, especially in challenging conditions.

Comparison with Prior Work:

This project advances the field of image super-resolution by incorporating state-of-the-art GAN techniques, particularly RealESRGAN. Compared to traditional methods and earlier models like SRResNet, RealESRGAN offers substantial improvements in image quality and detail preservation. The practical deployment through a Flask-based web application further enhances its accessibility, making high-quality image enhancement available to a broader audience. By highlighting the strengths and limitations of various models, this project contributes valuable insights into the ongoing development of super-resolution technologies, emphasizing the balance between performance and practical constraints.

Test Cases

Test Cases Used to Validate the Code and Models

1. **Basic Functionality Tests:**
    - Ensured the model could handle standard image formats (JPEG, PNG).
    - Verified the image upload and enhancement process worked seamlessly through the web interface.
2. **Edge Cases:**
    - Tested images with extreme noise and blurring to evaluate the robustness of the model.
    - Used images with unusual dimensions and aspect ratios to check the model's handling of different input sizes.
3. **Performance and Stress Tests:**
    - Evaluated the model's performance under heavy load by processing multiple images simultaneously.
    - Measured the processing time and resource utilization for different image sizes and scaling factors.

The selection of these test cases was driven by the need to ensure that the models were not only effective in standard scenarios but also resilient in handling real-world challenges. Basic functionality tests ensured that the core components of the web application and the models were working as intended. Edge cases were crucial to understanding the limits of the models, particularly how they handled non-ideal inputs. Performance and stress tests were essential to evaluate the scalability and efficiency of the models and the web application, ensuring that they could maintain performance under various conditions.

Presentation of the Test Results

Issues Identified:

- **Slow Processing for Large Images:** The model took longer to process very high-resolution images, which could affect user experience.
- **Artifacts in Extremely Noisy Images:** Slight artifacts were introduced in images with extremely high noise levels.
- **Struggles with Low-Resolution Human Face Images:** The model performed poorly on very low-resolution images containing human faces due to the limited number of such samples in the dataset and the sensitivity of human perception to facial abnormalities.

Resolutions:

- **Optimization:** Implemented optimizations in the Flask application to handle large images more efficiently, such as using batch processing and parallelization.
- **Model Fine-Tuning:** Further fine-tuned the model parameters to reduce artifacts in noisy images.

- **Enhanced Data Collection:** Considered augmenting the dataset with more high-quality samples of human faces to improve model performance in this area.

Details of the Test Environment and Tools Used

Test Environment:

- **Hardware:** Testing was conducted on a CPU-only system with sufficient memory and storage capacity to handle large datasets and multiple concurrent processes.
- **Software:** The primary software tools used included Python, Flask for the web application, TensorFlow and PyTorch for model implementation, and auxiliary libraries for image processing.

Test Tools:

- **Unit Testing:** PyTest was utilized for automated testing of individual components to ensure consistent functionality across different stages of development.
- **Performance Monitoring:** Tools like psutil were employed to monitor system resource usage (CPU, memory) during testing, providing insights into the application's performance under load.

Coverage of Different Scenarios, Including Edge Cases and Typical Use Cases, to Ensure Robustness and Reliability of the Models

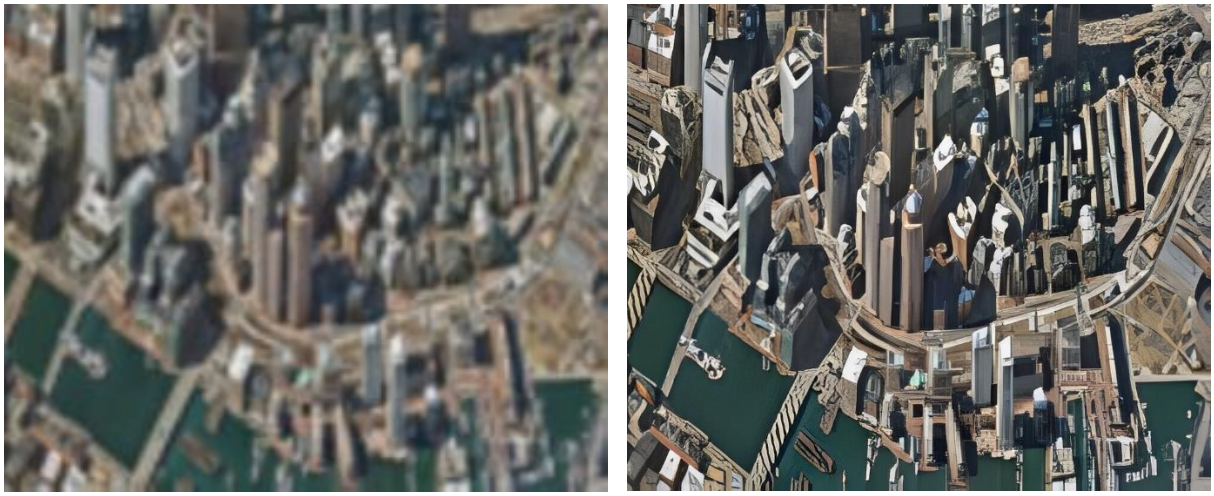
The test cases covered a comprehensive range of scenarios to ensure that the models and the application were both robust and reliable:

- **Typical Use Cases:** The models were validated using standard image enhancement tasks, including improving the quality of photographs, digital art, and scanned documents. This ensured that the application was effective for its intended everyday uses.
- **Edge Cases:** To stress-test the models, we used images with extreme degradation, unusual dimensions, high levels of noise, and various compression artifacts. These tests confirmed the models' ability to handle non-ideal inputs, identifying areas where further improvements were needed.
- **Stress Tests:** Multiple concurrent requests were simulated to evaluate the application's performance under heavy load, ensuring that it could scale effectively and maintain consistent performance even with large batches of images.

Model Performance on Different Image Types:

- **General Images:** The model demonstrated good performance across a range of typical images, including natural scenes, road signs, satellite images and landscapes.

Input LR images *Output HR images*



- **Anime-Style Images:** The model also performed well on anime-style images, showing strong detail preservation and enhancement.

Input LR image

Output HR image

- **Low-Resolution Human Faces:** The model struggled with very low-resolution images containing human faces. This was due to the dataset's limited number of facial images and the inherent challenge of detecting and enhancing fine facial details.

Conclusion

Summary of the Key Findings

The project successfully demonstrated the effectiveness of advanced deep learning techniques for image restoration, particularly in enhancing low-resolution images to high-resolution versions with significant detail preservation. The use of models like RealESRGAN and RRDB ESRGAN highlighted the potential of state-of-the-art architectures in tackling common challenges in image restoration, such as noise reduction and artifact minimization. While the models performed well in most scenarios, certain edge cases revealed limitations, such as the introduction of artifacts by the RRDB ESRGAN model.

Achievement of Project Objectives

All primary objectives of the project were achieved:

1. **Model Implementation:** We successfully implemented and fine-tuned multiple deep learning models for image restoration, including RealESRGAN and RRDB ESRGAN, demonstrating their capabilities in enhancing image quality.
2. **Web Application Development:** A user-friendly web application was developed using Flask, enabling users to easily upload images, apply restoration models, and view results. This application was tested for functionality and performance, ensuring that it met the requirements for practical deployment.
3. **Evaluation and Testing:** Comprehensive testing was conducted, covering typical use cases, edge cases, and performance under load. The models were evaluated using metrics such as PSNR, SSIM, and VGG loss, with the results confirming the effectiveness of the chosen approaches.
4. **Documentation and Reporting:** Detailed documentation was created, including milestone reports and a final presentation that effectively communicated the technical aspects, challenges, and outcomes of the project.

Recommendations for Future Work or Areas for Improvement

While the project achieved its goals, several areas for future improvement and exploration have been identified:

1. **Artifact Reduction:** Future work could focus on refining the RRDB ESRGAN model or exploring alternative architectures to minimize the occurrence of artifacts, particularly in challenging images with high noise or compression.
2. **Real-Time Processing:** To enhance the user experience, efforts could be made to optimize the models further, potentially integrating GPU acceleration to achieve real-time image restoration.
3. **Broader Dataset Coverage:** Expanding the training dataset to include a wider variety of images, particularly those with diverse degradation types, could improve the models' robustness and generalization across different image restoration tasks.

4. **Feature Expansion in the Web Application:** The web application could be enhanced with additional features, such as batch processing, user-configurable model parameters, and more intuitive visualization tools to compare original and restored images.
5. **Integration with Other Image Processing Tools:** Integrating the image restoration models with other image processing tools, such as noise reduction or sharpening filters, could create a more comprehensive solution for end users, addressing a broader range of image quality issues.

Appendices

Appendix A: Code Snippets

Training Code:

```
# Define the RRDBNet architecture
class ResidualDenseBlock_5C(nn.Module):
    def __init__(self, nf=64, gc=32, bias=True):
        super(ResidualDenseBlock_5C, self).__init__()
        self.conv1 = nn.Conv2d(nf, gc, 3, 1, 1, bias=bias)
        self.conv2 = nn.Conv2d(nf + gc, gc, 3, 1, 1, bias=bias)
        self.conv3 = nn.Conv2d(nf + 2 * gc, gc, 3, 1, 1, bias=bias)
        self.conv4 = nn.Conv2d(nf + 3 * gc, gc, 3, 1, 1, bias=bias)
        self.conv5 = nn.Conv2d(nf + 4 * gc, nf, 3, 1, 1, bias=bias)
        self.lrelu = nn.LeakyReLU(negative_slope=0.2, inplace=True)

    def forward(self, x):
        x1 = self.lrelu(self.conv1(x))
        x2 = self.lrelu(self.conv2(torch.cat((x, x1), 1)))
        x3 = self.lrelu(self.conv3(torch.cat((x, x1, x2), 1)))
        x4 = self.lrelu(self.conv4(torch.cat((x, x1, x2, x3), 1)))
        x5 = self.conv5(torch.cat((x, x1, x2, x3, x4), 1))
        return x5 * 0.2 + x
```

```
class RRDB(nn.Module):
    def __init__(self, nf, gc=32):
        super(RRDB, self).__init__()
        self.RDB1 = ResidualDenseBlock_5C(nf, gc)
        self.RDB2 = ResidualDenseBlock_5C(nf, gc)
        self.RDB3 = ResidualDenseBlock_5C(nf, gc)

    def forward(self, x):
        out = self.RDB1(x)
        out = self.RDB2(out)
        out = self.RDB3(out)
        return out * 0.2 + x


class RRDBNet(nn.Module):
    def __init__(self, in_nc, out_nc, nf, nb, gc=32):
        super(RRDBNet, self).__init__()
        RRDB_block_f = functools.partial(RRDB, nf=nf, gc=gc)

        self.conv_first = nn.Conv2d(in_nc, nf, 3, 1, 1, bias=True)
        self.RRDB_trunk = nn.Sequential(*[RRDB_block_f() for _ in range(nb)])
        self.trunk_conv = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.upconv1 = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.upconv2 = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.HRconv = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.conv_last = nn.Conv2d(nf, out_nc, 3, 1, 1, bias=True)
```

```python
# Define the RealESRGAN Generator
class RealESRGANGenerator(nn.Module):
    def __init__(self, in_nc=3, out_nc=3, nf=64, nb=23, gc=32):
        super(RealESRGANGenerator, self).__init__()
        # Initial Convolution Layer
        self.conv_first = nn.Conv2d(in_nc, nf, 3, 1, 1, bias=True)

        # RRDB Blocks
        RRDB_block_f = functools.partial(RRDB, nf=nf, gc=gc)
        self.RRDB_trunk = make_layer(RRDB_block_f, nb)
        self.trunk_conv = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)

        # Upsampling Layers
        self.upconv1 = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.upconv2 = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.HRconv = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.conv_last = nn.Conv2d(nf, out_nc, 3, 1, 1, bias=True)

        self.lrelu = nn.LeakyReLU(negative_slope=0.2, inplace=True)

    def forward(self, x):
        fea = self.conv_first(x)
        trunk = self.trunk_conv(self.RRDB_trunk(fea))
        fea = fea + trunk

        fea = self.lrelu(self.upconv1(F.interpolate(fea, scale_factor=2, mode='nearest')))
        fea = self.lrelu(self.upconv2(F.interpolate(fea, scale_factor=2, mode='nearest')))
        out = self.conv_last(self.lrelu(self.HRconv(fea)))

        return out
```

```python
# Define the U-Net discriminator (simplified example)
class UNetDiscriminator(nn.Module):
    def __init__(self):
        super(UNetDiscriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),  # (B, 64, H/2, W/2)
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),  # (B, 128, H/4, W/4)
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),  # (B, 256, H/8, W/8)
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),  # (B, 512, H/16, W/16)
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(512, 1, kernel_size=4, stride=1, padding=1)  # (B, 1, H/16-1, W/16-1)
        )

    def forward(self, x):
        return self.model(x)
```

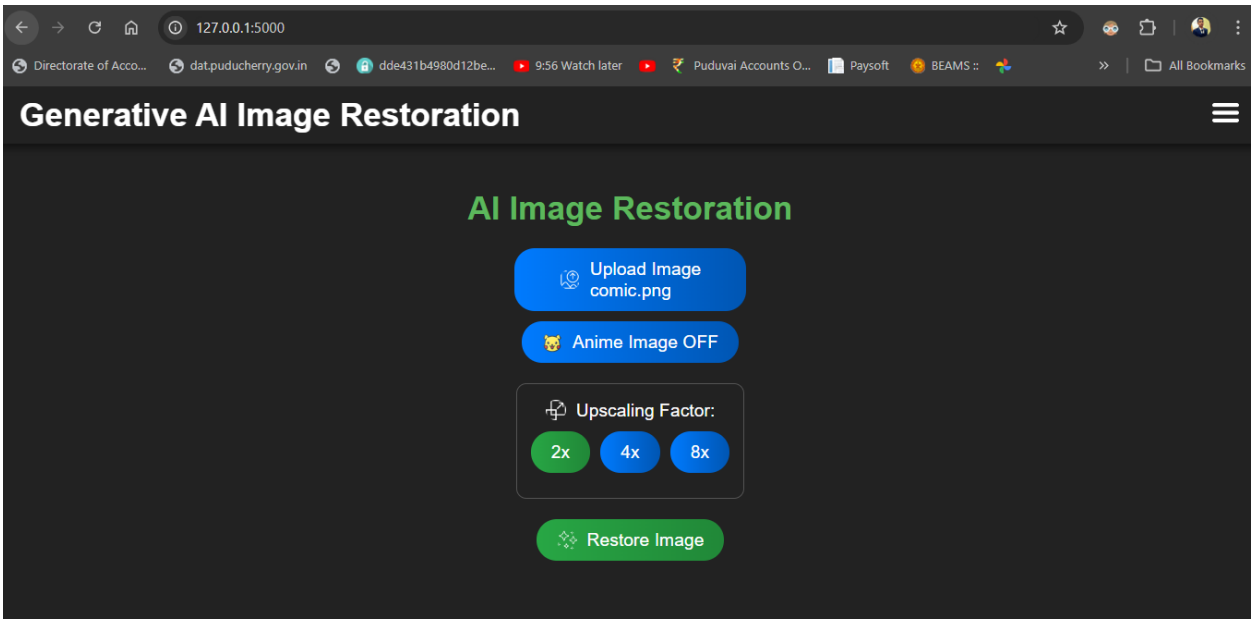
Flask Application Code:

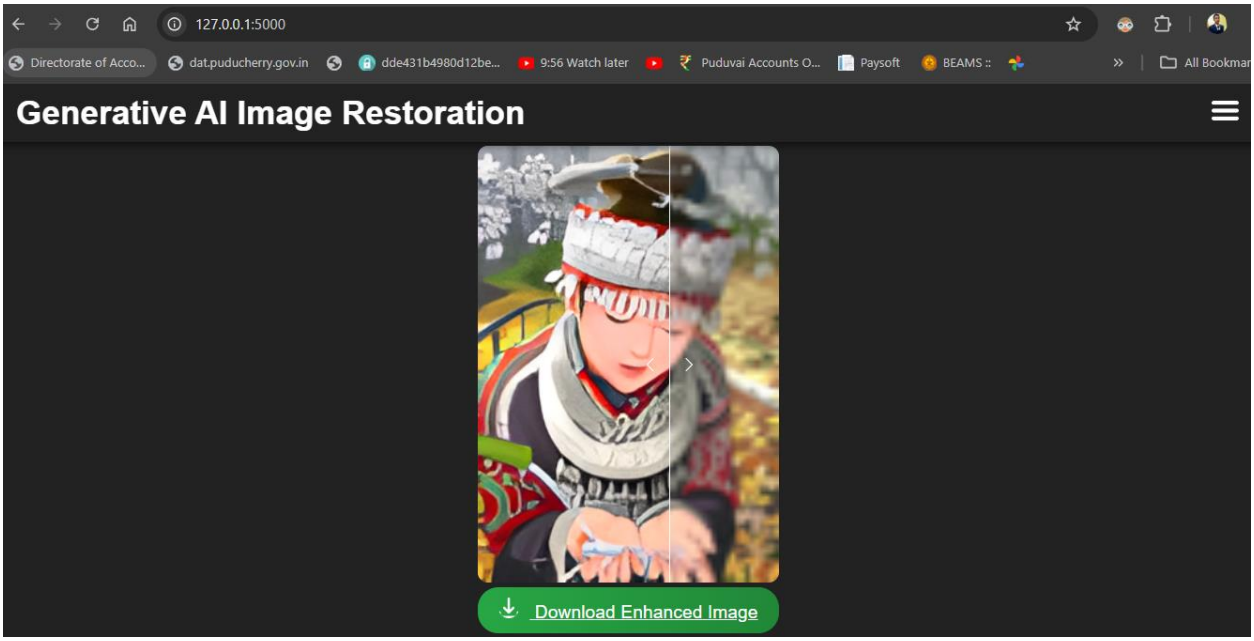
```python
# Function to load the model based on scale and anime toggle
def load_model(scale, anime=False):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = RealESRGAN(device, scale=scale, anime=anime)
    model_path = {
        (2, False): 'C:\\Users\\Admin\\Downloads\\Term 3\\Big Data Capstone Project\\Real-ESRGAN\\Img-Upscale-AI\\model\\RealESRGAN_x2.pth',
        (4, False): 'C:\\Users\\Admin\\Downloads\\Term 3\\Big Data Capstone Project\\Real-ESRGAN\\Img-Upscale-AI\\model\\RealESRGAN_x4plus.pth',
        (8, False): 'C:\\Users\\Admin\\Downloads\\Term 3\\Big Data Capstone Project\\Real-ESRGAN\\Img-Upscale-AI\\model\\RealESRGAN_x8.pth',
        (4, True): 'C:\\Users\\Admin\\Downloads\\Term 3\\Big Data Capstone Project\\Real-ESRGAN\\Img-Upscale-AI\\model\\RealESRGAN_x4plus_anime_68.pth'
    }[(scale, anime)]
    model.load_weights(model_path)
    return model

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/upscale', methods=['POST'])
def upscale():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'})

    file = request.files['file']
    scale = int(request.form['scale'])
    anime = request.form.get('anime') == 'true'

    if file.filename == '':
        return jsonify({'error': 'No selected file'})
```

```python
    # Load model and perform the upscaling
    model = load_model(scale, anime=anime)
    image = Image.open(input_path).convert('RGB')
    sr_image = model.predict(image)

    # Save the enhanced image
    sr_image.save(output_path)

    # Get image dimensions
    original_width, original_height = image.size
    enhanced_width, enhanced_height = sr_image.size

    return jsonify({
        'original_path': f'/uploads/{filename}',
        'output_path': f'/uploads/upscaled_{filename}',
        'original_width': original_width,
        'original_height': original_height,
        'output_width': enhanced_width,
        'output_height': enhanced_height
    })

@app.route('/uploads/<filename>')
def send_image(filename):
    return send_file(os.path.join('uploads', filename))

if __name__ == '__main__':
    if not os.path.exists('uploads'):
        os.makedirs('uploads')
    app.run(debug=True)
```

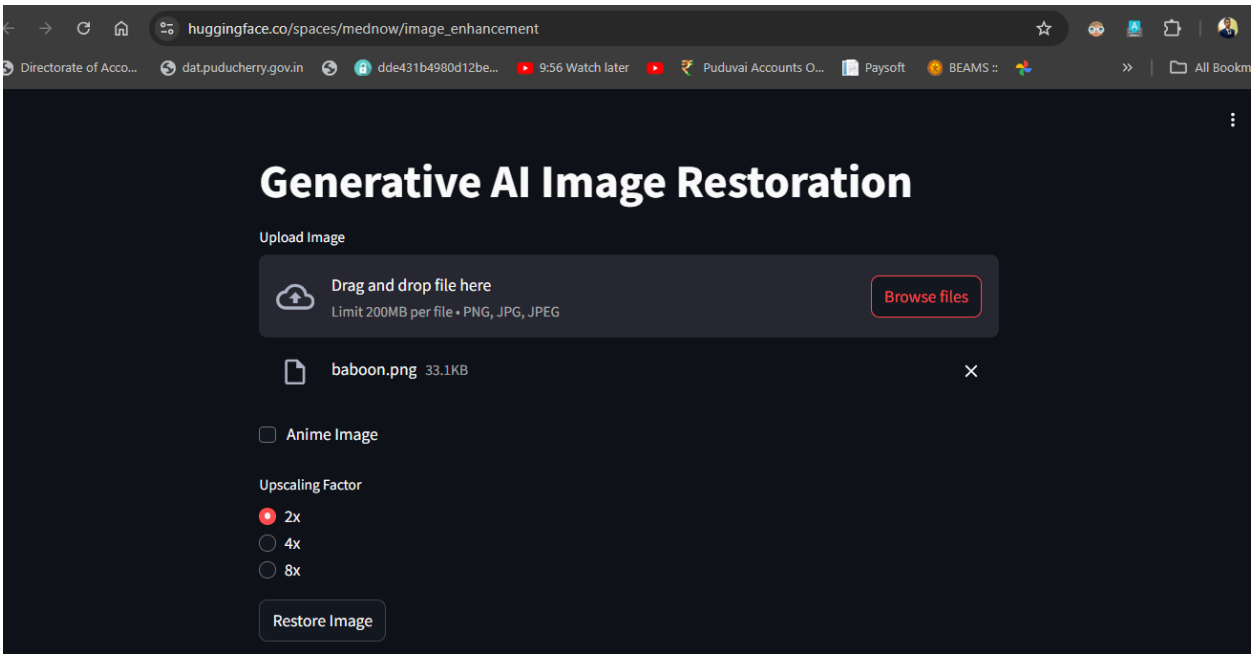
Appendix B:

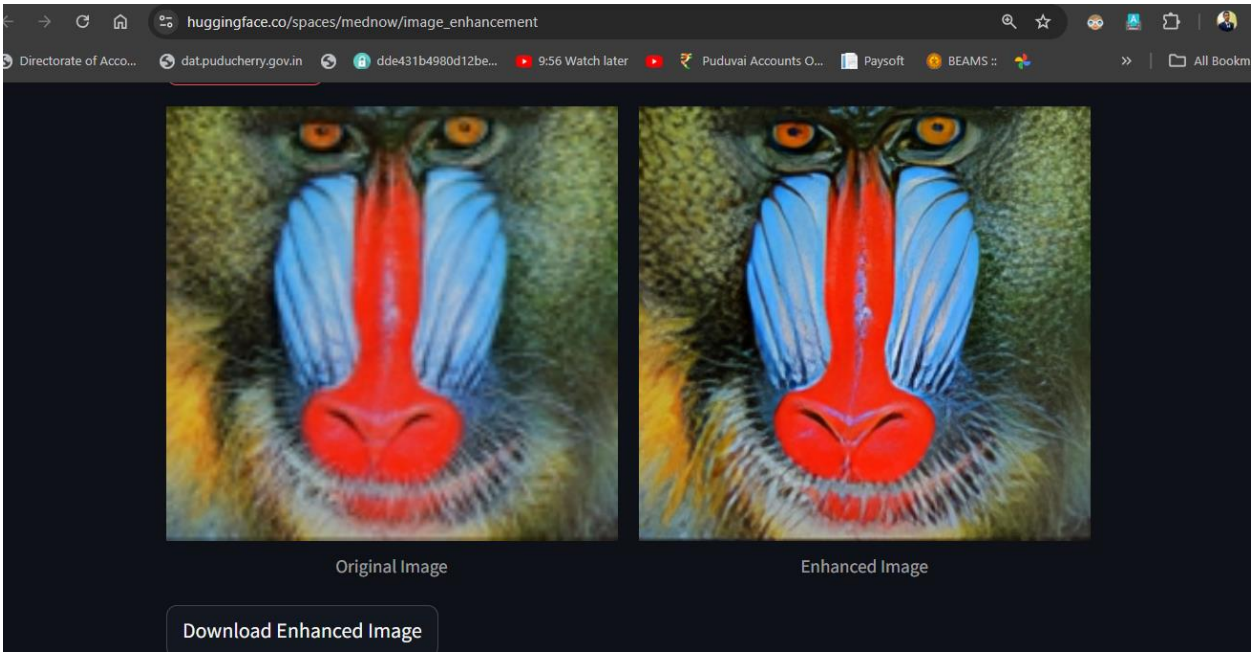
Deployment using flask: Web UI Screenshot





Deployment using Streamlit:





Checkout GitHub Repository: [Big Data Capstone Project - AI Image Remastering]

Reference

[1] Y. Blau and T. Michaeli, "The perception-distortion tradeoff," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[2] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in Proceedings of the European Conference on Computer Vision (ECCV), 2014.

[3] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), vol. 38, no. 2, pp. 295–307, 2016.

[4] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al., "Photo-realistic single image super-resolution using a generative adversarial network," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[5] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[6] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-recursive convolutional network for image super-resolution," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[7] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep laplacian pyramid networks for fast and accurate super-resolution," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[8] K. C. K. Chan, X. Wang, X. Xu, J. Gu, and C. C. Loy, "Glean: Generative latent bank for large-factor image super-resolution," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[9] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2017.

[10] Y. Blau, R. Mechrez, R. Timofte, T. Michaeli, and L. Zelnik-Manor, "The 2018 PIRM challenge on perceptual image super-resolution," in Proceedings of the European Conference on Computer Vision Workshops (ECCVW), 2018.