Group17

Manpurwar Ganesh
IIT Hyderabad
ai22btech11017@iith.ac.in

Tera Keshavardhan Reddy
IIT Hyderabad
ai22btech11029@iith.ac.in

K.D.V.S. Aditya
IIT Hyderabad
ai22btech11013@iith.ac.in

Ch. Kushwanth
IIT Hyderabad
ai22btech11006@iith.ac.in

Abstract

This report outlines the progress made in our project, which focuses on advanced image processing and deep learning techniques for image super-resolution and dataset management. Our work includes the development of a comprehensive pipeline for generating diverse image masks (irregular, center, structural, object removal, and text masks) and applying them to images, thereby enabling robust data augmentation for training purposes. We have implemented two versions of the IMDN (Information Multi-Distillation Network) model—one tailored for grayscale images using the MNIST dataset and another adapted for RGB images using the CIFAR-10 dataset—to achieve enhanced resolution restoration. Additionally, utility scripts have been developed to merge images from multiple folders and resize them to standardized dimensions, ensuring consistency across the dataset. For super-resolution tasks, we are currently testing the SwinIR model to explore its capabilities further, while our inpainting model remains under development. These contributions mark significant progress in our project, laying a solid foundation for further refinement and application of our methodologies in real-world image processing tasks.

1. Dataset Preparation

1.1. Objective

The goal of this pipeline is to process images from multiple folders by merging them into one, analyzing their dimensions, and resizing them to a standard size (256x256). The pipeline ensures proper handling of duplicate images, unreadable files, and non-image files.

1.2. Pipeline Breakdown

1.2.1. Step 1: Merging Images into One Folder

Function: `copy_images_to_one_folder(source_folders, destination_folder)`

Task:

- Merges images from multiple directories into a single folder.
- Handles duplicate filenames by renaming them.
- Ensures all images are copied with metadata.

Outcome:

- All images from `archive (1)`, `archive (2)`, and `archive` were successfully merged into `merged_dataset`.

1.2.2. Step 2: Analyzing Image Dimensions

Function: `analyze_images(image_folder)`

Task:

- Reads all images in the `merged_dataset` folder.
- Retrieves their dimensions (width $\times$ height).
- Counts occurrences of unique dimensions.
- Prints total images and unique dimensions.

Results:

- Total images processed: 3966
- Unique image dimensions found: 3253

Inference:

- The dataset contains a very high number of unique dimensions.
- Many images have different resolutions, requiring standardization for further processing.

1.2.3. Step 3: Resizing Images to 256$\times$256

Function: `resize_images(source_folder, destination_folder, target_size=(256, 256))`

Task:

- Reads images from `merged_dataset`.
- Resizes them to 256$\times$256 pixels.

- Saves them in `resized_dataset`.
- Handles unsupported file formats by converting them to .jpg.

Outcome:

- All valid images were successfully resized.
- Unreadable or corrupted images were skipped.

1.3. Key Findings

✓Successful Image Processing:

- 3966 images were processed.
- 3253 unique dimensions were found, indicating a highly diverse dataset.
- Resized images were stored in `resized_dataset`.

Challenges & Considerations:

- **High number of unique dimensions:** Some images might be very small or large. Consider filtering out extreme sizes.
- **Dividing based on dimensions:** After adding a margin of 20 for height and width , considering axb bxa as same dimensions also resulted in large no of unique dimensions.So to avoid this all images are resized into standard dimension of 256x256.

1.4. Appendix: Sample Images

Figure 1. Example Image 1 from the resized dataset

Figure 2. Example Image 2 from the resized dataset

Figure 3. Example Image 3 from the resized dataseta

2. Mask Types and Their Design Rationale

[2] [4] The training script employs four distinct types of masks, each designed to simulate different patterns of image occlusion. These masks help improve the model's robustness by exposing it to a variety of corruption types.

2.1. Narrowed Masks (`narrowed`)

- **Purpose:** Simulates narrow structural corruptions such as cracks or scratches.

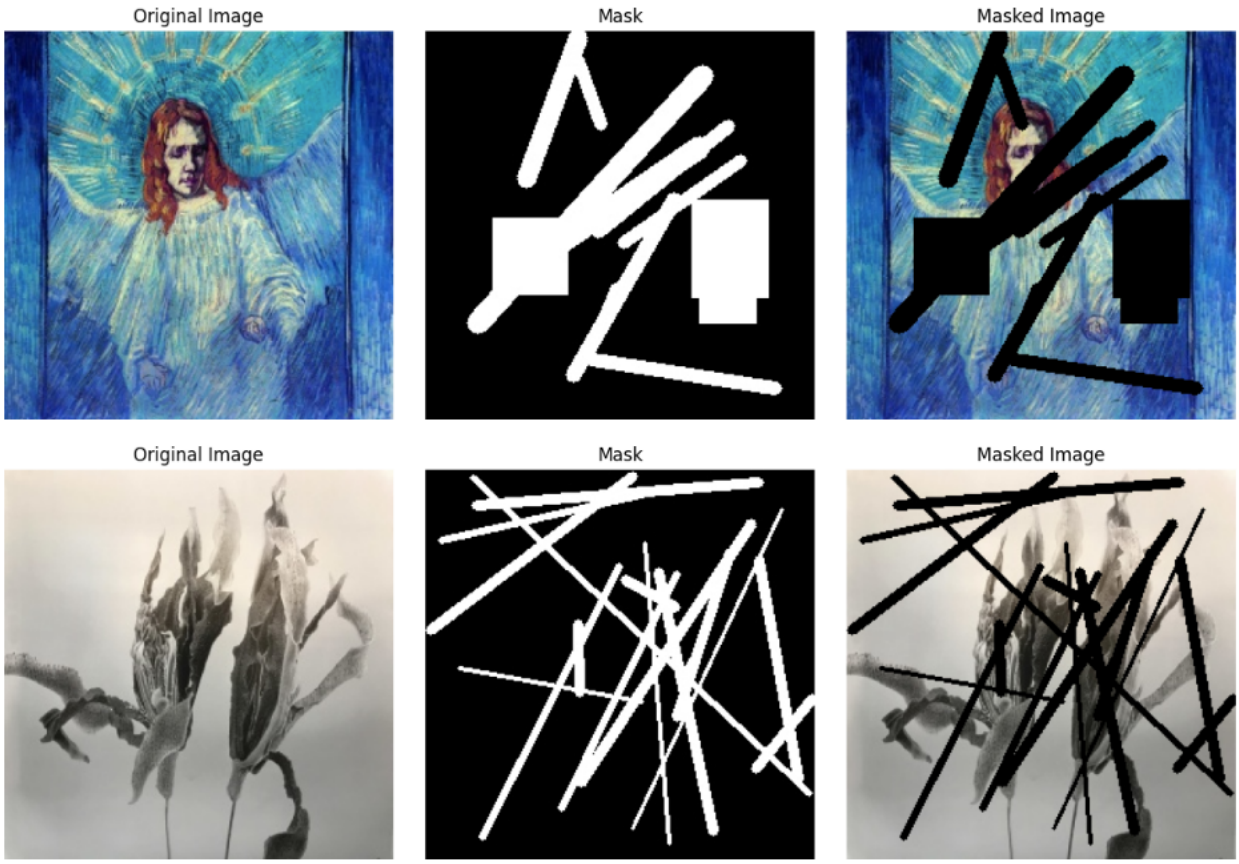


Figure 4. Your descriptive caption goes here.

- **Design:** Consists of 1 to 3 randomly positioned lines (vertical or horizontal), each with a random thickness ranging from 5 to 15 pixels.
- **Use Case:** Effective for training models to handle thin, elongated occlusions, commonly seen in damaged documents or scanned artworks.

2.2. Wide Masks (`large_wide`)

- **Purpose:** Simulates large vertical occlusions such as folds, columns, or overlays.
- **Design:** A single vertical block covering approximately 50% of the image width, placed at a random horizontal position.
- **Use Case:** Helps the model learn to reconstruct large missing content, particularly in structured regions like buildings or columns of text.

2.3. Box Masks (`large_box`)

- **Purpose:** Mimics rectangular damage such as torn-out sections or removed labels.
- **Design:** A randomly positioned rectangular or square region, occupying roughly 50% of the image area.
- **Use Case:** Trains the model to handle medium- to large-scale occlusions and aids in structural reconstruction.

2.4. Free-form Masks (`deepfillv2`)

- **Purpose:** Emulates complex and irregular occlusions inspired by the DeepFillv2 masking strategy.
- **Design:** A composition of 4 to 8 randomly drawn geometric shapes (e.g., rectangles, circles, lines) with varied sizes and orientations.
- **Use Case:** Introduces high variation and realism in occlusion patterns, promoting better generalization to real-world scenarios.

we assigned these masks to the input paintings randomly selected and created a dataset which gives **maksed_image, real_image, mask**

3. LaMa: Image Inpainting with Fourier Convolutions

3.1. Overview

[3] **LaMa (Large Mask Inpainting)** is a state-of-the-art architecture developed by **Samsung AI Center (SAI)** for

high-quality *image inpainting*, especially effective for large and irregular holes. It was introduced in the paper:

> *"Resolution-robust Large Mask Inpainting with Fourier Convolutions"* – Suvorov et al., 2021.

LaMa achieves realistic and semantically plausible inpainting by leveraging **Fast Fourier Convolutions (FFC)** and training strategies that encourage **global coherence** and **sharp details**.

3.2. Key Architectural Components

- DownScale Block(downscales 3x)
- Spectral Transform Block
- Fast fourier convolutions block
- inpainting network (9x)
- Upscale Block(upscale 3x)

4. DownscaleBlock and UpscaleBlock

The **DownscaleBlock** and **UpscaleBlock** are fundamental components employed to modify the spatial resolution of feature maps in a neural network, particularly in tasks such as image inpainting or restoration.

The **DownscaleBlock** performs a *downsampling* operation by applying a 3×3 convolution with a stride of 2, which reduces the spatial dimensions of the input tensor. This operation is sequentially followed by *batch normalization* and a *ReLU activation* function. Such a configuration allows the model to capture abstract, high-level features at a lower resolution, thereby enabling more efficient processing.

Conversely, the **UpscaleBlock** is designed to *upscale* the feature maps, effectively increasing their spatial resolution. This is achieved through *nearest-neighbor upsampling* (a non-learnable method) by a factor of 2, followed by a 3×3 convolution with a stride of 1. The convolutional layer is also followed by *batch normalization* and a *ReLU activation* function. This block serves to restore the spatial dimensions while refining the learned features via the convolutional operation.

Together, these blocks enable the network to learn hierarchical representations of the input data, facilitating the progressive extraction and reconstruction of fine-grained details from low-resolution to high-resolution outputs.

5. SpectralTransform

5.1. Overview

The **SpectralTransform** module is a neural network component that integrates both **spatial** and **frequency domain** processing to enhance feature extraction and manipulation. By leveraging the **Fast Fourier Transform (FFT)**, it enables the model to operate in the frequency domain, where both **high-frequency details** and **low-frequency structures** can be captured more explicitly and efficiently. This hybrid approach improves the model's ability to represent complex patterns across multiple spatial scales.

5.2. Architectural Components

The architecture is composed of several key processing stages, combining spatial-domain convolutions with frequency-domain transformations.

5.2.1. Spatial Convolution and Activation (Pre-FFT Processing)

Given an input tensor $x \in \mathbb{R}^{B\times C\times H\times W}$ — where B is the batch size, C is the number of channels, and H, W are spatial dimensions — the tensor first undergoes convolution in the spatial domain. This step refines local features before transformation into the frequency domain.

- **1x1 Convolution:** A 1×1 convolution maintains the number of channels while preserving spatial dimensions.
- **Batch Normalization and ReLU:** Applied to stabilize and non-linearly activate features.

5.2.2. Frequency Domain Transformation (FFT-based Operations)

- **Real 2D FFT:** The processed tensor is transformed to the frequency domain using a 2D real FFT:

$$X_{\text{fft}} = \mathcal{F}(x) \in \mathbb{C}^{B\times C\times H\times \left(\frac{W}{2}+1\right)}$$

- **Splitting Complex Components:** The complex output is decomposed into its real and imaginary parts, both of shape $B \times C \times H \times \left(\frac{W}{2}+1\right)$. These are concatenated along the channel dimension, resulting in a real-valued tensor of shape $B \times 2C \times H \times \left(\frac{W}{2}+1\right)$.

5.2.3. Frequency Domain Convolution

- **1x1 Convolution:** Applied to the concatenated real and imaginary parts to refine frequency features.
- **Batch Normalization and ReLU:** Further normalize and activate the frequency-domain features.

5.2.4. Reconstructing the Complex Tensor

After convolution in the frequency domain, the modified real and imaginary components are recombined to form a complex tensor:

$$\hat{X}_{\text{fft}} = \text{Re} + i \cdot \text{Im}$$

5.2.5. Inverse FFT and Spatial Reconstruction

- **Inverse FFT:** The complex tensor is transformed back to the spatial domain using the inverse FFT:

$$x_{\text{irfft}} = \mathcal{F}^{-1}(\hat{X}_{\text{fft}}) \in \mathbb{R}^{B\times C\times H\times W}$$

- **Residual Connection:** The result is added to the pre-FFT spatial features:

$$x_{\text{out}} = x_{\text{irfft}} + x$$

This residual connection preserves both spatial and frequency information.

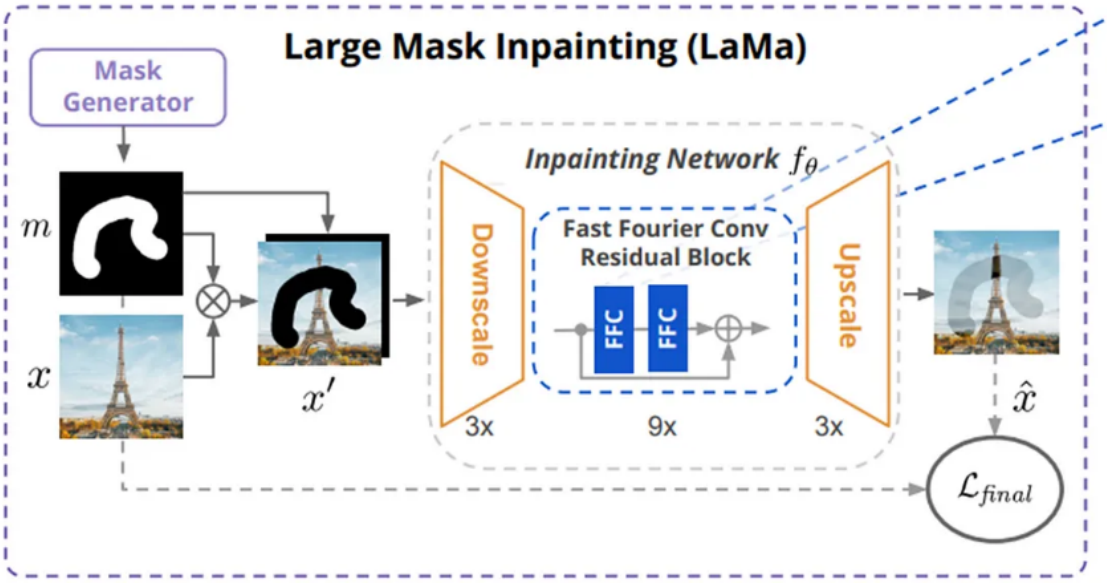


Figure 5. Model Architecture

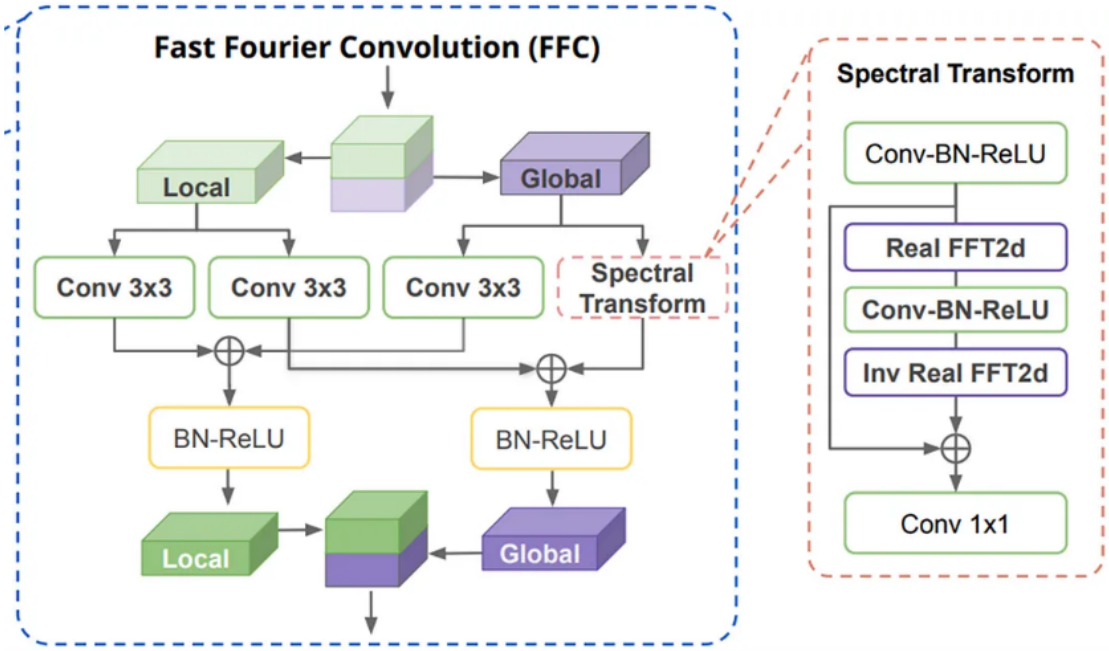


Figure 6. Fast Fourier Convolutions

5.2.6. Final Convolution (1x1 Refinement)

A final 1×1 convolution is applied to adjust the number of channels and refine the final representation.

5.3. Theoretical Motivation

- **Multiresolution Analysis:** Spatial convolutions capture fine-grained local details, while spectral operations encode global contextual patterns.
- **Efficient Frequency Processing:** The frequency domain allows selective enhancement or suppression of specific

signal bands, improving denoising and structure modeling.
- **Compact Representations:** Spectral operations often lead to more compact and informative feature maps, boosting model efficiency.

5.4. Conclusion

The **SpectralTransform** module effectively combines spatial and frequency-domain techniques to facilitate robust feature learning. This dual-path design enables the network to incorporate both local and global information, leading to superior performance in inpainting and other image restoration tasks.

6. Fast Fourier Convolutions

This module significantly improves the network's ability to recover high-frequency patterns and fine image details, making it well-suited for image restoration tasks such as inpainting.

6.1. FFC (Fast Fourier Convolution)

The **FFC** block utilizes a dual-path structure that blends both *local* and *global* feature extraction to improve the richness of the learned representations.

- **Local Pathway:** Employs standard spatial convolutions to process fine-grained local features. This includes a pair of convolutions followed by ReLU activation to maintain locality and nonlinearity.
- **Global Pathway:** Processes global features using the *SpectralTransform*, which captures high-level contextual patterns in the frequency domain.

After separate processing, the local and global outputs are fused:

- The local output is passed through another convolution layer.
- The global output undergoes a spectral transformation.
- Then both outputs are merged to combine spatially fine and contextually rich information.

This hybrid approach allows the network to simultaneously leverage both detailed and contextual information, which is especially beneficial for complex tasks involving structure recovery or image generation.

7. InpaintingNetwork Architecture

The **InpaintingNetwork** is a deep learning architecture tailored for the task of *image inpainting*, where the goal is to reconstruct missing or masked regions in an image. The network adopts a multi-stage design comprising three primary phases: *downscaling*, *feature transformation*, and *upscaling*.The inpainting network consists of 9 FFC residual blocks

- The dataloader provides both the original image and a pre-generated mask.
- masked image is created by applying the mask to the original image.
- The masked image and the corresponding mask are concatenated and passed as input to the inpainting network.
- The 4-channel input is downsampled by a factor of 3 and processed through 9 residual FFC blocks. The output is then upsampled by a factor of 3 using an upsampling block.
- The final output is a 3-channel inpainted image.

Summary

The **InpaintingNetwork** leverages a three-stage architecture to address the challenges of image inpainting:

- **Downscaling** helps extract abstract features efficiently.
- **FFC blocks** combine spatial and frequency-domain processing to capture diverse features.
- **Upscaling** reconstructs the image while preserving learned details.

This design enables the network to effectively fill in missing regions while maintaining coherence with the surrounding context, making it suitable for a wide range of image inpainting scenarios.

8. Loss Architecture and Discriminator Design

The training of the proposed inpainting model is driven by a comprehensive loss structure that combines multiple loss components to guide the generator toward producing visually realistic and semantically consistent results. This section outlines the core modules involved in the loss formulation and the discriminator design.

8.1. Adversarial Loss (`AdvLoss`)

The `AdvLoss` module defines the adversarial loss using a **PatchGAN** discriminator. It comprises both *discriminator* and *generator* loss terms:

- **Discriminator Loss ($\mathcal{L}_D$)**: Encourages the discriminator to correctly classify real images as real and generated (fake) images as fake by minimizing the negative log-likelihood.
- **Generator Loss ($\mathcal{L}_G$)**: Encourages the generator to fool the discriminator, i.e., to produce outputs that are classified as real.

This adversarial dynamic improves the realism of the generated outputs.

8.2. HRF Perceptual Loss (`HRFPerceptualLoss`)

This module uses a pre-trained **VGG19** network to compute the *perceptual similarity* between real and generated images.

- The loss is calculated as the ℓ_2 distance between feature maps extracted from selected layers of VGG19.
- This encourages high-level visual similarity, beyond simple pixel-wise differences.

8.3. Discriminator Perceptual Loss (Feature Matching) (`DiscPerceptualLoss`)

The `DiscPerceptualLoss` compares feature maps of real and generated images as extracted from the intermediate layers of the PatchGAN discriminator.

- This feature matching strategy aligns real and fake feature distributions, stabilizing training and preserving fine image structure.

8.4. Gradient Penalty (`gradient_penalty`)

A regularization term known as the **R1 Gradient Penalty** is applied to the discriminator to prevent overfitting and instability.

- It encourages the gradient norm of the discriminator's output with respect to real samples to be close to 1:

$$\mathcal{L}_{\text{gp}} = \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_r} \left[\|\nabla_{\hat{x}} D(\hat{x})\|^2 \right]$$

8.5. Final Inpainting Loss (`FinalInpaintingLoss`)

This class aggregates all individual losses into a single total objective function:

$$\mathcal{L}_{\text{total}} = \kappa \mathcal{L}_{\text{adv}} + \alpha \mathcal{L}_{\text{vgg}} + \beta \mathcal{L}_{\text{fm}} + \gamma \mathcal{L}_{\text{gp}}$$

- $\mathcal{L}_{\text{adv}}$: Adversarial loss
- $\mathcal{L}_{\text{vgg}}$: VGG-based perceptual loss
- $\mathcal{L}_{\text{fm}}$: Feature matching loss
- $\mathcal{L}_{\text{gp}}$: Gradient penalty

Hyperparameters $\kappa, \alpha, \beta, \gamma$ control the relative importance of each term.

8.6. Patch Discriminator (`PatchDiscriminator`)

The `PatchDiscriminator` implements a **PatchGAN**-style discriminator which focuses on local image patches rather than the whole image.

- Composed of stacked convolutional layers with `LeakyReLU` activations and Batch Normalization.
- Outputs a probability map indicating the realism of individual image patches.
- Provides intermediate feature maps for use in `DiscPerceptualLoss`.

8.7. Summary

This loss framework facilitates stable and effective training of the inpainting model by integrating:

- Local realism via adversarial loss
- High-level similarity via VGG perceptual loss
- Structural consistency through feature matching
- Discriminator regularization via gradient penalty

The use of a **PatchGAN** discriminator ensures the model pays attention to local structures and textures, leading to more visually coherent inpainted results.

9. Training Pipeline Overview

The training script orchestrates the end-to-end learning process of the inpainting system, involving both the generator (`InpaintingNetwork`) and the discriminator (`PatchDiscriminator`). The key components of this pipeline are outlined below:

1. **Checkpoint Loading:** The script attempts to load previously saved model weights from a designated checkpoint directory. If checkpoints exist, training resumes from the latest saved epoch. Otherwise, training starts from scratch.
2. **Optimizer Setup:** Both the generator and the discriminator are optimized using the `Adam` optimizer with a learning rate of 1×10^{-4}. This choice balances convergence speed and training stability.
3. **Training Loop:** The model is trained over 50 epochs. Each epoch involves the following steps:
   - The **discriminator** is trained using real and generated images. The loss includes the R1 gradient penalty to enforce smoothness and regularity.
   - The **generator** is updated by minimizing the total inpainting loss, which integrates adversarial loss, VGG-based perceptual loss, and discriminator feature matching loss.
4. **Model Saving:** After each epoch, the weights of both the generator and discriminator are saved. Filenames include the epoch number to facilitate version tracking and checkpoint recovery.
5. **Loss Weights:** The total loss is a weighted sum of multiple components with the following hyperparameters: adversarial loss ($\kappa = 10$), perceptual loss ($\alpha = 30$), feature matching loss ($\beta = 100$), and R1 gradient penalty ($\gamma = 0.001$). These values balance reconstruction quality, perceptual fidelity, and training stability.

Table 1. Average losses by epoch (without R1 grad penalty).

Epoch	L_{adv}	L_{hrfpl}	L_{discpl}
5	2.0520	2.1677	0.9677
15	2.0519	1.3646	0.7252
24	2.0521	1.1568	0.7255
34	2.0518	1.0629	0.6542
44	2.0518	0.9843	0.6412
54	2.0517	0.8858	0.6420
75	2.0515	0.8319	0.6227

6. **Loss Monitoring:** The script utilizes the `tqdm` library to provide a real-time progress bar, displaying current values of the loss components during training.

10. Data preparation for SwinIR

10.1. Background and Objective

Super-resolution networks like SwinIR require paired low-resolution (LR) and high-resolution (HR) images during supervised training. The dataset we chose didn't have perfectly aligned LR–HR pairs. To address this, we employ a two-stage pipeline:

1. **Stage 1:** Use a pretrained Swin2SR transformer model to up-sample raw 256×256 paintings to 512×512 and 1024×1024
2. **Stage 2:** Treat these super-resolved outputs as "ground-truth" HR targets, pairing them with downsampled or original LR inputs to train a SwinIR model.

The result is a self-generated, high-quality training corpus enabling SwinIR to learn 2× up-sampling mappings tailored to a specific painting dataset.

10.2. Data and Preprocessing

- **Input Set:** Source images are 256×256 jpg files of painting excerpts, stored in a read-only input directory.
- **Intensity Normalization:** Convert 8-bit pixel values (0–255) into floating-point [0,1] to stabilize transformer inference.
- **Spatial Padding:** Ensure both height and width are multiples of the model's attention window size (8 px) to avoid boundary artifacts in shifted-window self-attention.

10.3. Super-Resolution Model (Swin2SR ×2)

- **Model Choice:** A Swin Transformer adapted for classical image super-resolution at a fixed 2× scale.
- **Key Features:**
  - Local self-attention within non-overlapping windows that shift each layer, capturing fine and contextual details.
  - Reconstruction head mapping transformer features back to the image domain, doubling spatial resolution.
- **Inference Mode:** Model is loaded in evaluation mode (disabling dropout and batch-norm updates) and executed on GPU, achieving throughput of a few seconds per 512×512 image.

10.4. HR Generation Workflow

1. **Image Loading:** Each 256×256 painting is read in RGB format to standardize color channels.
2. **Processor Pipeline:**
   - *Rescale*: Pixel intensities brought into [0,1].
   - *Pad to multiple of 8*: Ensures dimensions are compatible with the model's window size.
3. **Forward Pass:** The preprocessed tensor is fed into Swin2SR, producing a tensor of shape $3\times512\times512$. Outputs are clamped to [0,1].
4. **Conversion and Saving:** The tensor is converted back to image format (e.g. PNG) and saved in the designated output directory, preserving original filenames for easy pairing.

10.5. Constructing SwinIR Training Pairs

- **HR Targets:** The 512×512 images generated above.
- **LR Inputs:**
  - Option A: The original 256×256 files (ideal alignment, no extra degradation).
- By keeping filenames consistent, perfectly aligned $(I_{\text{LR}}, I_{\text{HR}})$ pairs are obtained for supervised training.

10.6. Quality Assessment

10.6.1. Visual Inspection

- Sharpness of edges (brush strokes, contours) improves markedly over naive bicubic upsampling.
- Color fidelity and texture continuity are preserved, reducing checkerboard or ringing artifacts.

10.7. Quantitative Metrics (Optional)

- **PSNR** was computed between Swin2SR outputs and any available HR ground truth, for relative assessment, which aligned with the result given for the dataset.

11. Conclusion

By leveraging a state-of-the-art Swin Transformer super-resolution model, this pipeline automates the creation of a bespoke $256\times256 \rightarrow 512\times512$ training dataset. It ensures pixel-wise aligned LR–HR pairs for supervised learning, harnesses transformer-based global and local feature modeling, and provides flexibility in defining LR degradation strategies—ultimately enabling a fine-tuned SwinIR model to excel at 2× super-resolution on painting imagery.

12. Model Architecture

The SwinIR-based model is composed of three main stages: shallow feature extraction, deep feature extraction with transformer-based residual blocks, and image reconstruction. This section describes the architecture, as implemented in Python.

12.1. Implementation Overview

[1]

The architecture is implemented in PyTorch using custom modules for each key component: shallow feature extractor, deep transformer-based backbone (consisting of Residual Swin Transformer Blocks), and a final image reconstruction head. The model design allows flexibility in

the number of transformer layers, feature dimensions, and residual blocks.

12.2. Shallow Feature Extraction

The input image $I_{LQ} \in \mathbb{R}^{H \times W \times C}$ is passed through a single 3×3 convolutional layer that increases the channel dimension to the internal embedding size C:

$$F_0 = \text{Conv}_{3\times3}(I_{LQ}) \tag{1}$$

This operation is implemented via:

```
nn.Conv2d(in_channels, embed_dim,
kernel_size=3, stride=1, padding=1)
```

12.3. Deep Feature Extraction: Residual Swin Transformer Blocks

The extracted shallow features are processed through a sequence of K Residual Swin Transformer Blocks (RSTBs). Each RSTB is composed of:

- L Swin Transformer Layers (STLs), each with window-based multi-head self-attention (W-MSA) and shifted windows (SW-MSA).
- A convolutional layer that refines features within the block.
- A residual connection from the block's input to its output.

This is captured in Python as:

```
class ResidualSwinTransformerBlock(nn.Module):
    def __init__(...):
        self.blocks = nn.ModuleList([
            SwinTransformerLayer(...)
            for _ in range(depth)
        ])
        self.conv =
        nn.Conv2d(embed_dim, embed_dim, kernel_size=3, padding=1)
```

Each STL performs self-attention in local windows, using the shifted window technique to capture cross-window interactions. This alternation is handled using 'shift size=0' and 'shift size=M//2' for successive layers.

12.4. Window-Based Self-Attention

Each Swin Transformer Layer uses window-based attention to reduce complexity. Attention is computed independently in $M \times M$ windows. The shifted window approach ensures connections across windows.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}} + B\right)V \tag{2}$$

Relative position biases are added using a learnable parameter table indexed by the relative coordinates of patches in the window.

12.5. Reconstruction Head

After feature aggregation via a final convolutional layer, the model optionally upsamples the features depending on the task. For super-resolution tasks, the upsampling is done using PixelShuffle:

$$I_{HQ} = \text{PixelShuffle}(\text{Conv}_{3\times3}(F_{\text{DF}} + F_0)) \tag{3}$$

Implemented as:

```
nn.Conv2d(embed_dim, upscale_factor^2 * in_channel
nn.PixelShuffle(upscale_factor)
```

12.6. Model Parameters and Configuration

The architecture is parameterized using the following hyperparameters (example values from your code):

- **embed_dim:** 96
- **depths:** [6, 6, 6, 6] (number of STLs per stage)
- **num_heads:** [6, 6, 6, 6]
- **window_size:** 8
- **residual_blocks (num_blocks):** 6
- **mlp_ratio:** 4

These parameters can be modified to scale the model's capacity or efficiency.

12.7. Loss Function and Training

The model is trained using the mean absolute error (MAE or ℓ_1 loss) between the predicted high-resolution image and the ground truth:

$$\mathcal{L}_{\ell_1} = \|\hat{I}_{HQ} - I_{HQ}\|_1 \tag{4}$$

The training loop includes support for checkpointing, patch-based training (via unfolding), and various dataset backends (e.g., DIV2K, RealSR).

13. Model Overview and Preliminary Testing

The SwinIR (Swin Transformer for Image Restoration) model implemented in this work is based on the hierarchical vision Transformer framework, designed specifically for low-level vision tasks such as image super-resolution. It utilizes window-based self-attention and shifted windows to efficiently balance local detail and global context.

The model architecture comprises:

- A shallow feature extraction block (convolution layer)
- Multiple Residual Swin Transformer Blocks (RSTBs), each composed of Swin Transformer Layers (STLs)
- A reconstruction block that outputs the high-resolution image

The model was trained on the DIV2K dataset for 25 epochs using ℓ_1 loss as the objective function. PSNR (Peak Signal-to-Noise Ratio) was used to evaluate reconstruction quality after each epoch.

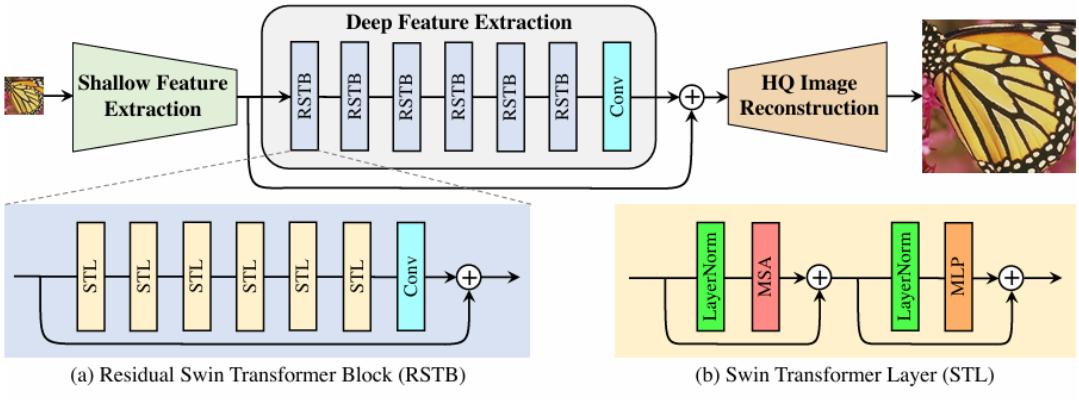


Figure 2: The architecture of the proposed SwinIR for image restoration.

Figure 7. Model Architecture

Observations from Training

Throughout training, the loss consistently decreased, and the PSNR metric steadily improved from an initial value of 23.40 dB to a final peak of 29.04 dB, indicating a significant enhancement in image reconstruction quality.

The model checkpoints were saved at epochs where new best PSNR values were achieved. Notably:

- Early training (Epochs 1–5) showed rapid improvements, with PSNR rising from 23.40 to 25.56.
- Mid-training (Epochs 6–15) maintained stable progress, achieving PSNR values above 27 dB.
- Final epochs (Epochs 20–25) saw the highest PSNR gains, peaking at 29.04 in Epoch 24.

Peak Signal-to-Noise Ratio (PSNR)

PSNR is a widely-used objective metric to quantify image quality by measuring the ratio between the maximum possible power of a clean (reference) signal and the power of corrupting noise that affects its representation. It is defined as:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right)$$

where MAX is the maximum possible pixel value (255 for 8-bit images) and MSE is the Mean Squared Error between the original high-resolution image and the super-resolved output.

Higher PSNR values imply that the reconstructed image is closer to the ground truth. In the context of image super-resolution:

- PSNR between 20–25 dB indicates noticeable artifacts or blur.
- PSNR between 25–28 dB suggests acceptable to good visual quality.
- PSNR above 28 dB typically reflects high-fidelity restoration, often visually indistinguishable from the original image.

Although PSNR does not always perfectly correlate with perceptual quality, it serves as a useful and objective baseline for comparing reconstruction performance across epochs or model variants.

SwinIR Training Report

1. Implementation Overview

A supervised image super-resolution pipeline was constructed to convert 256×256 low-resolution (LR) painting inputs into 512×512 high-resolution (HR) outputs. The principal components are as follows:

Data Preparation A `DIV2KDataset` class pairs LR and HR images from separate directories. Center-cropping ensures exact spatial alignment: each HR image is cropped to 256×256, and each LR image to 128×128, prior to conversion into PyTorch tensors.

Network Architecture

- **Shallow Embedding:** A 1×1 convolution (PatchEmbed) projects RGB input into a 96-dimensional feature space.
- **Residual Swin Transformer Body:** Two Residual Swin Transformer Blocks (RSTBs), each comprising six Swin Transformer layers (`depths=[6,6]`) with six attention heads (`num_heads=[6,6]`) and an 8×8 window. Each RSTB concludes with a 3×3 convolution that fuses features and incorporates a residual connection.
- **Upsampling Head:** A 3×3 convolution expands channel count by 4×, followed by `PixelShuffle(2)` to double spatial dimensions ($256 \rightarrow 512$). A final 3×3 convolution projects back to a three-channel RGB output.

Figure 8. Example output of the SwinIR model after 25 epochs of training.

Training Configuration • **Loss Function:** L_1 (mean absolute error), chosen for edge sharpness preservation and outlier robustness.
- **Optimizer:** Adam with fixed learning rate 2×10^{-4}.
- **Batch Size:** 8
- **Validation Protocol:** PSNR is computed on 10 randomly sampled training images at each epoch's end; best-performing model is checkpointed.
- **Epochs:** 40

2. Experimental Iterations

2.1 Modularization Attempts

Initial experiments modularized dataset and model code into separate files via Jupyter's `%%writefile`, but this approach was abandoned in favor of an integrated notebook format to streamline iterative development.

2.2 Prototype SwinIR Variant

- **Structure:** `depths=[4,4]`, `num_heads=[4,4]`, `window_size=4`.
- **Optimizer & Scheduler:** AdamW with weight decay 1×10^{-4} and OneCycleLR (10% warmup to peak LR $= 2 \times 10^{-4}$, cosine annealing).
- **Loss Exploration:** Charbonnier loss for robust gradients, then switched to MSE to directly optimize PSNR.
- **Outcome:** Rapid early convergence but late-epoch instability motivated simplification.

2.3 Final Architecture & Simplified Training

- **Increased Capacity:** Upgraded to `depths=[6,6]`, `window_size=8`.
- **Optimizer Simplification:** Plain Adam (no weight decay), constant LR $= 2 \times 10^{-4}$.
- **Validation Shortcut:** Lightweight PSNR evaluation on 10 samples instead of full validation.

3. Scheduler and Loss Function Ablations

Notes:
- *OneCycleLR* provided strong initial updates but introduced scheduling complexity.

Configuration	Optimizer & Scheduler	Loss Function
Prototype	AdamW + OneCycleLR	Charbonnier $\rightarrow$ MSE
Final	Adam (fixed LR 2×10^{-4})	L_1

- L_1 *loss* delivered sharper reconstructions and correlated well with PSNR improvements.

4. Training Results (Epochs 1{40)

Epoch	Avg. L_1	Avg. PSNR	Best PSNR
1	0.0273	31.45 dB	31.45 dB
4	0.0164	33.10 dB	33.10 dB
11	0.0148	33.40 dB	33.40 dB
17	0.0136	34.61 dB	34.61 dB
29	0.0128	35.06 dB	35.06 dB
32	0.0130	35.42 dB	35.42 dB
38	0.0134	35.68 dB	35.68 dB
40	0.0124	34.86 dB	35.68 dB

Loss decreased steadily from 0.0273 to 0.0124, and PSNR peaked at 35.68 dB on epoch 38, triggering the best-model checkpoint.

This can also be seen in the following image9

5. Next Steps

1. Expanded Validation: Allocate a larger held-out dataset for more robust PSNR evaluation.
2. Data Augmentation: Incorporate rotations, flips, and color jitter to improve generalization.
3. Lightweight Scheduling: Evaluate simple step-decay or plateau-based LR adjustments.
4. Model Scaling: Explore additional RSTB stages or increased embedding dimension if resources permit.

Note: All experiments were restricted to the above configurations due to GPU memory and runtime limitations.

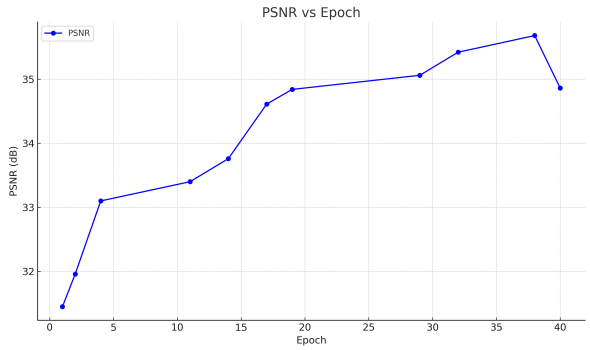


Figure 9. PSNR vs. Epoch during training

14. Training Results and Testing results of Super-Resolution Model

Below are the qualitative comparisons of Low-Resolution (LR), Super-Resolved (SR), and High-Resolution (HR) images for three samples:

14.1. Discussion

The training began with a high loss (0.0273) and relatively low PSNR (31.45) in Epoch 1. However, the model rapidly improved within the first few epochs, surpassing 33 dB PSNR by Epoch 4.

From Epoch 11 onward, the model steadily improved its reconstruction quality, achieving over 35 dB PSNR by Epoch 29. The best PSNR of 35.68 was observed at Epoch 38, showing that the model continued learning subtle details late into training. Loss values concurrently decreased to as low as 0.0124 by Epoch 40, suggesting convergence.

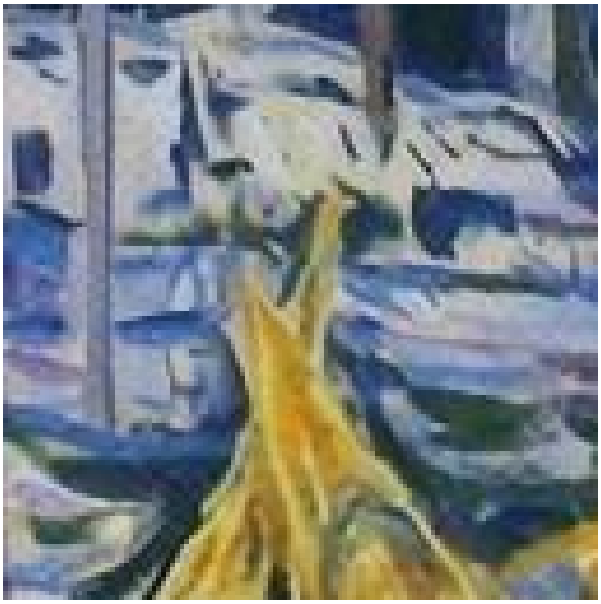
These trends suggest the model not only learned efficiently but continued improving even in later epochs, likely due to effective optimization and architectural choices.

14.2. Testing Results

Figure 10 shows the image obtained from the trained SwinIR model.

14.3. Results of the pipeline

The following the outputs of the images that are obtaned from the lama model that are passed into the swinir trained above.The resolutions of lama output is 256 which is raised to 1024 by passing it twice through the same model(Images of 512 cannot be well distinguished by eye at this size so).As we can resolution of image improves.This can be seen in the following images11



Low-Resolution (LR)

Super-Resolved (SR)

Figure 10. Testing result for one sample: LR on the left and the corresponding SR output on the right.

15. LaMa Inpainting Results

15.1. Quantitative Evaluation

We evaluate the inpainting performance using two standard metrics: Fréchet Inception Distance (FID) and Inception Score (IS). Lower FID indicates closer similarity to real images, while higher IS reflects better quality and diversity of generated samples. The results are summarized in Table 2.

Figure 11. Example LR (left) vs. SR (right) outputs for three samples.

Inpainting Type	FID ↓	Inception Score ↑
Large Wide (Only)	132.45	2.55 ± 0.37
All Combined	100.28	2.82 ± 0.55
DeepFill (Only)	126.96	2.39 ± 0.26
Narrow (Only)	83.0953	2.9565 ± 0.4017

Table 2. FID and Inception Scores across different inpainting mask types. Lower FID and higher IS indicate better performance.

From the results, we observe that combining various inpainting types yields the best overall performance, achieving the lowest FID and highest Inception Score. Large wide masks show the highest FID, indicating more visible inpainting artifacts. Results for narrow masks will be added once computation completes.

References

[1] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 1833–1844, 2021. 8

[2] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 85–100, 2018. 2

[3] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21335–21345, 2022. 3

[4] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 341–349, 2012. 2

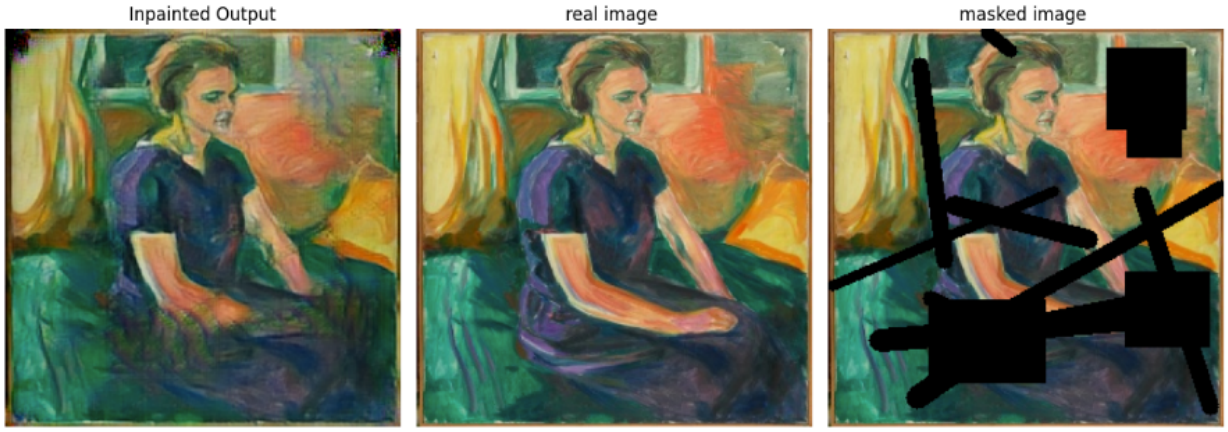


Figure 12. LaMa inpainting result showing original, masked image, and inpainted output. (DeepFill)

Figure 13. Narrow masks

Table 3. Summary of inpainting performance across mask types (epoch 75).

Mask Type	Scope	Key Behaviors	Artifacts
Large scribbles (deep-fill blobs)	Global	Guesses broad textures (e.g. foliage)	Blurry averaged fills; color/texture shift; repetitive "ripple" patterns
Mid-sized blobs over semantic regions	Semi-global	Recovers coarse shapes (torso, arm)	Soft, blurry edges; color banding; loss of fine brush-stroke detail
Thin scratches (narrow masks)	Local	Near-perfect interpolation of lines & tones	None—smooth, seamless blending
Wide rectangular hole	Global	Hallucinates rough forms (branches, fountain)	Warped geometry; unwanted color speckles; blurred cross-hatching

Figure 14. Large box

Figure 15. Large wide