GNR650 Project

Task: Image Denoising + Image Inpainting

Pankhi Kashyap (25D1385) and Vivekananda Giri (25D1381)

November 23th, 2025

1 Introduction

- Image Denoising: The goal is to recover a clean image from a noisy input, eliminating unwanted visual distortions while preserving fine details. The model learns to map corrupted images (with noise) back to their clean versions.
- Image Inpainting: The objective is to reconstruct missing or masked regions in an image seamlessly, generating plausible content that is both locally and globally consistent with the context. This task requires understanding of both textures and semantic structures.

2 Objectives

The primary objective of this project is to define and analyze the multi-tasks i.e. Image Denoising and Image Inpainting within the Taskonomy framework and to analyze interaction and synergy between tasks in the decoder, aiming for improved sample efficiency and generalization.

3 Problem Formulation

Let $\mathcal{I} \in \mathbb{R}^{H\times W\times 3}$ denote a clean RGB image. We generate two corrupted variants:

- **Noisy image:**
$$\mathcal{I}_n = \mathcal{I} + \eta, \quad \eta \sim \mathcal{N}(0, \sigma^2)$$
- **Masked image:**
$$\mathcal{I}_m = M \odot \mathcal{I} + (1 - M) \odot \mathbf{0}$$
where $M \in \{0, 1\}^{H\times W}$ is a binary mask.

$$f_n^s = E_\theta^{LoRA}(\mathcal{I}_n)$$
$$f_m^s = E_\theta^{LoRA}(\mathcal{I}_m)$$

Let the student head outputs s $\in \mathbb{R}^D$, teacher t $\in \mathbb{R}^D$, so features in latent space be f^s, f^t respectively for clean images. The student logits and teacher logits be z^s, z^t respectively for clean images.

Our objective is to reconstruct:

$$\hat{I}_n = f_\theta(f_n^s) \approx I \quad \text{(Denoising task)},$$
$$\hat{I}_m = g_\phi(f_,^s M) \approx I \quad \text{(Inpainting task)},$$

where f_θ and g_ϕ denote the denoising and inpainting decoders sharing a common encoder backbone.

4 Dataset and Preprocessing

4.1 Base Dataset: miniImageNet

We use the miniImageNet dataset as the base corpus of natural images. Each image is resized to a fixed spatial resolution of 128 $\times$ 128 pixels using bicubic interpolation. All images are converted to RGB and scaled to the continuous range $[0, 1]$.

Let $I \in [0, 1]^{128\times 128\times 3}$ denote a clean image sampled from the miniImageNet training split.

4.2 Noisy Image Generation

To construct the denoising task, we synthetically corrupt clean images with additive white Gaussian noise at two different noise levels.

For each clean image I, we sample noise

$$\eta \sim \mathcal{N}(0, \sigma^2),$$

independently for each pixel and channel. We generate two noisy variants:

$$I_n^{(0.1)} = \text{clip}(I + \eta_{0.1},\, 0, 1), \quad \eta_{0.1} \sim \mathcal{N}(0, 0.1^2),$$
$$I_n^{(0.2)} = \text{clip}(I + \eta_{0.2},\, 0, 1), \quad \eta_{0.2} \sim \mathcal{N}(0, 0.2^2).$$

In practice, both noise levels are used during training to encourage robustness across a range of corruption strengths. The model receives the noisy image as input for the denoising branch and is trained to reconstruct the original clean image I.

4.3 Masked Image Generation (Inpainting Task)

For the inpainting task, we generate a binary mask $M \in \{0, 1\}^{128 \times 128}$ that occludes a contiguous rectangular region of the image.

We construct M such that approximately 30% of the pixels are masked:

$$\text{mask ratio} \approx 0.3.$$

Concretely, we randomly sample the top-left coordinate (x_0, y_0) and width–height (w, h) of an axis-aligned rectangle such that:

$$\frac{w \cdot h}{128 \cdot 128} \approx 0.3,$$

and set

$$M(x, y) = \begin{cases} 1, & \text{if } (x, y) \text{ lies inside the chosen rectangle,} \\ 0, & \text{otherwise.} \end{cases}$$

The masked image is then defined as

$$I_m = (1 - M) \odot I,$$

where masked pixels are set to zero (black) and only the unmasked region is preserved. The inpainting decoder receives (I_m, M) as input and is trained to reconstruct the complete clean image I, with losses concentrated primarily on the masked region.

4.4 Training Triplets

Each training sample thus consists of:

$$\big(I,\ I_n^{(0.1)} \text{ or } I_n^{(0.2)},\ I_m,\ M\big),$$

where:

- I is the ground-truth clean image (miniImageNet, resized to 128×128),
- $I_n^{(\sigma)}$ is a noisy variant with $\sigma \in \{0.1, 0.2\}$,
- I_m is the rectangular-mask inpainted input,
- M is the corresponding binary mask.

These tuples are fed into the joint model, with the denoising branch supervised using $(I_n^{(\sigma)}, I)$ and the inpainting branch supervised using (I_m, M, I).

5 Model Structure and Approach

Our model is a multi-task reconstruction network that jointly performs image denoising and image inpainting using a shared transformer encoder and a dual-headed decoder. The architecture consists of four main components:

1. A frozen pretrained DINOv3-Small Vision Transformer encoder with LoRA adapters,
2. A lightweight convolutional fusion module that combines different degraded inputs,
3. A dual-task decoder with separate branches for residual denoising and mask-aware inpainting,
4. Cross-attention and conditioning mechanisms that encourage interaction between tasks and robustness across noise levels.

5.1 Input Fusion Layer

At training and inference time, we consider three image variants derived from the same clean image: the noisy image I_n, the masked image I_m, and the binary mask M. These are concatenated channel-wise and projected back to a 3-channel representation that is compatible with the ViT encoder:

$$X_{\text{fused}} = \text{Conv}_{1\times1}\big(\text{concat}(I_n,\ I_m,\ M)\big),$$

where $\text{Conv}_{1\times1}$ is a learnable 1×1 convolution that maps the 7-channel tensor (3 + 3 + 1) to a 3-channel RGB-like image. This fused representation aggregates complementary degradation cues into a single encoder input.

5.2 LoRA-Adapted DINOv3-Small Encoder

We use a pretrained DINOv3-Small Vision Transformer (ViT) as a shared feature extractor. The backbone is loaded from `timm` with classification head removed and global pooling disabled, yielding patch-level embeddings. The patch embedding layer is adjusted to the new input resolution (128×128), and positional embeddings are dynamically interpolated to match the current patch grid.

To enable efficient fine-tuning without updating all ViT weights, we inject Low-Rank Adaptation (LoRA) modules into the last transformer block. For a base linear layer with weight $W \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$, LoRA adds a low-rank residual:

$$W' = W + \frac{\alpha}{r}BA,$$

where $A \in \mathbb{R}^{r\times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}}\times r}$ are trainable low-rank factors, r is the rank, and α is a scaling factor. The original backbone parameters are frozen and only LoRA parameters plus the decoders are trained.

Given the fused input X_{fused}, the encoder outputs a sequence of patch embeddings:

$$F = E_{\text{LoRA}}(X_{\text{fused}}) \in \mathbb{R}^{B\times N\times D},$$

where B is batch size, N is the number of spatial tokens, and $D = 384$ is the embedding dimension for DINOv3-Small.

5.3 Dual-Task Decoder: Residual Denoising and Inpainting

The decoder is designed as a dual-headed module that maps shared token embeddings F back into the image space for both tasks. It first reshapes the token sequence into a low-resolution feature map:

$$F_{\text{map}} \in \mathbb{R}^{B\times D\times H'\times W'}, \quad H' = W' = \frac{128}{16} = 8,$$

and then upsamples to full resolution via a stack of transposed convolutions.

5.3.1 Residual Denoising Branch

The denoising branch predicts a residual ε that approximates the noise component in I_n:

$$\varepsilon = D_{\text{denoise}}(F,\ I_n,\ \gamma(\ell)) \in \mathbb{R}^{B\times 3\times 128\times 128},$$

where:

- F are the encoder features,
- I_n is optionally injected as a high-resolution skip connection (concatenated at the last upsampling stage),
- $\gamma(\ell)$ is a feature-wise modulation obtained from a scalar noise-level embedding ℓ.

The denoised image is then computed as:

$$\hat{I}_n = \text{clip}\big(I_n - \varepsilon,\ 0, 1\big).$$

This residual formulation encourages the network to focus on modeling the corruption rather than the full image, improving stability and convergence.

5.3.2 Noise-Level Conditioning

To encode the severity of corruption, we pass a scalar noise level $\ell \in \mathbb{R}$ (e.g., corresponding to $\sigma \in \{0.1, 0.2\}$) through an MLP that produces channel-wise modulation parameters:

$$\gamma = \text{MLP}(\ell) \in \mathbb{R}^{C},$$

which are reshaped to $\gamma \in \mathbb{R}^{C\times 1\times 1}$ and applied in a FiLM-like fashion:

$$H' = H \odot (1 + \gamma),$$

where H is the intermediate feature map in the denoising decoder. This conditioning allows a single model to adapt its behavior across different noise regimes.

5.3.3 Mask-Aware Inpainting Branch

The inpainting branch reconstructs the full image using both encoder features and the binary mask M. After upsampling F_{map} to full resolution, the feature map is concatenated with a resized mask:

$$H_{\text{inp}} = \text{concat}\big(H_{\text{up}},\ \text{resize}(M)\big) \in \mathbb{R}^{B\times(64+1)\times 128\times 128},$$

followed by convolutional layers producing the inpainted image:

$$\hat{I}_m = D_{\text{inpaint}}(F,\ M) \in [0,1]^{B\times 3\times 128\times 128}.$$

During training, losses are primarily concentrated on the masked region defined by M to encourage accurate filling of missing content without over-penalizing the already visible context.

5.4 Cross-Attention Between Tasks

To explicitly model interaction between denoising and inpainting, we introduce optional cross-attention modules at both token and spatial levels:

- **Token-level cross attention:** Given two token streams (conceptually, denoising and inpainting tokens), we form:

  $$\tilde{F}_{\text{den}} = \text{CrossAttnTokens}(Q = F_{\text{den}},\ K = F_{\text{inp}},\ V = F_{\text{inp}}), (CrossAttention - 1)$$

  $$\tilde{F}_{\text{inp}} = \text{CrossAttnTokens}(Q = F_{\text{inp}},\ K = F_{\text{den}},\ V = F_{\text{den}}), (CrossAttention - 2)$$

  allowing each branch to query complementary information from the other.

- **Spatial cross attention:** On feature maps, we apply multi-head attention across flattened spatial dimensions:

  $$\tilde{H}_{\text{den}} = \text{CrossAttn2D}(H_{\text{den}}, H_{\text{inp}}), \quad \tilde{H}_{\text{inp}} = \text{CrossAttn2D}(H_{\text{inp}}, H_{\text{den}}), (CrossAttention - 3)$$

  which helps transfer structural and textural cues between tasks.

These modules are controlled via configuration flags so that we can empirically evaluate the impact of each type of cross-attention.

6 Multi-Task Training

6.1 Training Configuration and Hyperparameter Settings

All training experiments use a unified configuration across denoising and inpainting branches. The model is trained for a total of $E = 100$ epochs using the Adam optimizer. Batch sizes of 64 are used for training, validation, and testing, with 4 parallel loading workers and pinned memory enabled to optimize GPU throughput. Gaussian corruption uses a fixed noise standard deviation $\sigma = 0.2$ for denoising samples unless stated otherwise.

LoRA adaptation is used inside the encoder with a rank $r = 8$ and scaling factor $\alpha = 16$, enabling efficient fine-tuning while keeping the base ViT parameters frozen.

Table 1 summarizes all final hyperparameter values used in the experiments.

Parameter	**Value**
LoRA Rank (r)	8
LoRA Scaling Factor (α)	16
Batch Size (Train / Val / Test)	64/64/64
Number of Data Loader Workers	4
Pinned Memory	True
Noise Standard Deviation (σ for denoising)	0.2
Total Epochs (E)	30
Generator Learning Rate (η_G)	1×10^{-4}
Discriminator Learning Rate (η_D)	0.5×10^{-4}
Weight Decay	1×10^{-4}
Optimizer	Adam
Adversarial Loss Weight (λ_{adv})	0.1
Masked L1 Loss Weight (λ_{L1})	0.9
GAN Warm-up Epochs	2
Discriminator Update Interval	5 iterations

Table 1: Training hyperparameters used in multi-task denoising and inpainting experiments.

These values remain constant across all reported experiments unless otherwise stated. The configuration is chosen to ensure balanced learning between the denoising and inpainting objectives, stability in adversarial training, and efficient utilization of compute while maintaining convergence.

6.2 Adversarial Head

During training, the inpainting output is further refined via:

- a PatchGAN discriminator that operates on local patches of real (I) versus generated ($\hat{I}_m$) images, enforcing high-frequency realism,

This component is used only at training time and are discarded at inference, keeping the deployed model lightweight: only the fusion conv, LoRA-adapted encoder, and dual-task decoder are required for forward passes.

6.3 Optimization Setup

We train the model in an end-to-end fashion on the joint denoising + inpainting objective using miniImageNet-based corrupted inputs. The generator side consists of the LoRA-adapted DINOv3 encoder, the dual-task decoder, and the 1×1 fusion convolution. A separate PatchGAN discriminator is trained only for the inpainting branch.

We use the Adam optimizer for both generator and discriminator:

$$\theta_G \leftarrow \theta_G - \eta_G \nabla_{\theta_G} \mathcal{L}_G,$$
$$\theta_D \leftarrow \theta_D - \eta_D \nabla_{\theta_D} \mathcal{L}_D,$$

where θ_G includes all trainable LoRA parameters in the encoder, all decoder weights, and the fusion convolution, and θ_D denotes discriminator parameters.

6.4 Loss Functions

The total loss is the sum of a denoising loss and an inpainting loss:

$$\mathcal{L}_T = \mathcal{L}_{\text{denoise}} + \mathcal{L}_{\text{inpaint}}.$$

Denoising loss. For the denoising branch, we use mean squared error (MSE) over the full image:

$$\mathcal{L}_{\text{denoise}} = \|\hat{I}_n - I\|_2^2,$$

where $\hat{I}_n$ is the denoised output and I is the clean target. This is implemented via `nn.MSELoss()`.

Inpainting loss. The inpainting branch operates only on the masked region. Let $M \in \{0,1\}^{H\times W}$ denote the binary mask (1 = hole to inpaint), and M_3 its 3-channel expansion. We compute:

$$I_{\text{hole}}^{\text{pred}} = \hat{I}_m \odot M_3, \qquad I_{\text{hole}}^{\text{gt}} = I \odot M_3.$$

The inpainting loss combines L1, perceptual, and (optionally) adversarial terms:

$$\mathcal{L}_{\text{inpaint}} = \lambda_{\text{L1}} \, \|I_{\text{hole}}^{\text{pred}} - I_{\text{hole}}^{\text{gt}}\|_1 + \lambda_{\text{adv}}^{\text{eff}} \, \mathcal{L}_{\text{adv}},$$

with:

- $\lambda_{\text{L1}} = \texttt{LAMBDA\_L1}$: weight for masked L1 loss,
- $\lambda_{\text{adv}}^{\text{eff}} = \texttt{LAMBDA\_ADV} \cdot \texttt{ADV\_SCALE}$: effective adversarial weight, with $\texttt{ADV\_SCALE} = 0.25$.

The adversarial term $\mathcal{L}_{\text{adv}}$ is defined via a PatchGAN discriminator D, using BCE-with-logits loss:

$$\mathcal{L}_D = \frac{1}{2}\Big(\ell_{\text{BCE}}(D(x_{\text{real}}), 1) + \ell_{\text{BCE}}(D(x_{\text{fake}}), 0)\Big),$$

$$\mathcal{L}_{\text{adv}} = \ell_{\text{BCE}}(D(x_{\text{fake}}), 1),$$

where x_{real} and x_{fake} are masked regions from the clean and inpainted images, normalized to $[-1, 1]$.

During validation, we keep only the masked L1 term for inpainting and MSE for denoising, in order to report stable, non-adversarial reconstruction losses.

6.5 GAN Warm-up and Discriminator Updates

To stabilize training, we do not activate the GAN loss from the beginning. Instead, we use a warm-up strategy controlled by `WARMUP_EPOCHS`:

$$\text{use\_gan} = \begin{cases} \text{False}, & \text{if epoch} \le \texttt{WARMUP\_EPOCHS}, \\ \text{True}, & \text{otherwise}, \end{cases}$$

with $\texttt{WARMUP\_EPOCHS} = 2$ in our implementation. When use_gan is False, the inpainting loss reduces to:

$$\mathcal{L}_{\text{inpaint}} = \lambda_{\text{L1}} \, \|I_{\text{hole}}^{\text{pred}} - I_{\text{hole}}^{\text{gt}}\|_1 + \lambda_{\text{perc}} \, \mathcal{L}_{\text{perc}},$$

and the discriminator is not updated (its gradients remain zero).

Once GAN training is active, the discriminator is updated less frequently than the generator. Specifically, we update D every `D_UPDATE_INTERVAL` steps:

$$\text{if use\_gan and } (\text{global\_step} \bmod \texttt{D\_UPDATE\_INTERVAL} = 0) \Rightarrow \text{update } \theta_D,$$

with $\texttt{D\_UPDATE\_INTERVAL} = 5$.

6.6 Training Loop and Checkpointing

For each epoch $e = 1, \ldots, E$:

1. Sample mini-batches from the training loader: (I, I_n, I_m, M).
2. Construct fused input and run the model to obtain $\hat{I}_n$ and $\hat{I}_m$.
3. Compute $\mathcal{L}_{\text{denoise}}$ and $\mathcal{L}_{\text{inpaint}}$; sum to get $\mathcal{L}_G$.
4. Backpropagate $\mathcal{L}_G$ and update generator parameters with Adam.
5. If GAN is active and the step matches the update interval, update the discriminator using $\mathcal{L}_D$.

At the end of each epoch, we run validation on the held-out set, computing average denoising and inpainting losses. We log per-epoch metrics (train and validation losses, discriminator loss, and generator adversarial loss) into a CSV file and update the best model whenever the sum of validation losses improves. The best checkpoint stores both the transformer-based generator and the fusion convolution.

Additionally, we periodically save debug images (masked input, mask, and composite inpainted outputs) during training to visually monitor the evolution of the inpainting quality and mask handling.

7 Evaluation Protocol and Metrics

7.1 Evaluation Setup

At test time, we evaluate the trained model on a held-out test split constructed from miniImageNet with the same corruption pipeline as during training. For each clean test image I, we generate a noisy image I_n and a masked image I_m with corresponding binary mask M. These are provided by the test data loader as:

$$(I,\ I_n,\ I_m,\ M) \in [0,1]^{128\times128\times3} \times [0,1]^{128\times128\times3} \times [0,1]^{128\times128\times3} \times \{0,1\}^{128\times128}.$$

The model is evaluated in a single forward pass using the same 7-channel fusion strategy as training:

$$X_{\text{fused}} = \text{Conv}_{1\times1}\big(\text{concat}(I_n,\ I_m,\ M)\big),$$

where concat is the channel-wise concatenation of the noisy image, masked image, and mask, and $\text{Conv}_{1\times1}$ is the learned fusion convolution. The fused representation is passed through the LoRA-adapted DINOv3 encoder and dual-task decoder to obtain:

$$\hat{I}_n = f_\theta(I_n), \qquad \hat{I}_m = g_\phi(I_m, M).$$

For inpainting, we reconstruct a composite image by blending the predicted content only inside the hole and preserving the observed pixels elsewhere:

$$I_{\text{comp}} = \hat{I}_m \odot M_3 + I_m \odot (1 - M_3),$$

where M_3 is the 3-channel expansion of the binary mask M. This composite image I_{comp} is used for all inpainting metrics.

All evaluation is performed with gradients disabled, using the final saved checkpoint from training.

7.2 Pixel-Level Error Metrics

We report pixelwise fidelity using both mean squared error (MSE) and mean absolute error (MAE) between predictions and ground truth. For a predicted image $\hat{X}$ (either $\hat{I}_n$ or I_{comp}) and its corresponding clean target I, with $H = W = 128$ and $C = 3$ channels, we define:

Mean Squared Error (MSE).

$$\text{MSE}(\hat{X}, I) = \frac{1}{CHW} \sum_{c=1}^{C} \sum_{x=1}^{H} \sum_{y=1}^{W} \big(\hat{X}(c,x,y) - I(c,x,y)\big)^2.$$

Mean Absolute Error (MAE).

$$\text{MAE}(\hat{X}, I) = \frac{1}{CHW} \sum_{c=1}^{C} \sum_{x=1}^{H} \sum_{y=1}^{W} \left| \hat{X}(c,x,y) - I(c,x,y) \right|.$$

These are computed per-image and averaged over all test images for both tasks:

$$\text{MSE}_{\text{denoise}} = \frac{1}{N} \sum_{i=1}^{N} \text{MSE}(\hat{I}_n^{(i)}, I^{(i)}), \quad \text{MSE}_{\text{inpaint}} = \frac{1}{N} \sum_{i=1}^{N} \text{MSE}(I_{\text{comp}}^{(i)}, I^{(i)}),$$

and similarly for $\text{MAE}_{\text{denoise}}$ and $\text{MAE}_{\text{inpaint}}$.

7.3 Peak Signal-to-Noise Ratio (PSNR)

To quantify reconstruction quality in decibels (dB), we use Peak Signal-to-Noise Ratio with a maximum pixel value $I_{\max} = 1.0$ (since images are normalized to $[0, 1]$):

$$\text{PSNR}(\hat{X}, I) = 10 \log_{10} \left(\frac{I_{\max}^2}{\text{MSE}(\hat{X}, I)} \right) = 10 \log_{10} \left(\frac{1}{\text{MSE}(\hat{X}, I)} \right).$$

As with MSE and MAE, PSNR is computed for each test sample and averaged across the set to obtain:

$$\text{PSNR}_{\text{denoise}} = \frac{1}{N} \sum_{i=1}^{N} \text{PSNR}(\hat{I}_n^{(i)}, I^{(i)}), \quad \text{PSNR}_{\text{inpaint}} = \frac{1}{N} \sum_{i=1}^{N} \text{PSNR}(I_{\text{comp}}^{(i)}, I^{(i)}).$$

7.4 Structural Similarity Index (SSIM)

To assess structural and perceptual similarity, we compute the Structural Similarity Index (SSIM) between the predicted and clean images. For each color image, SSIM is computed over channels jointly using a standard implementation (e.g., scikit-image), with dynamic range $[0, 1]$:

$$\text{SSIM}(\hat{X}, I) = \frac{(2\mu_{\hat{X}}\mu_I + C_1)(2\sigma_{\hat{X}I} + C_2)}{(\mu_{\hat{X}}^2 + \mu_I^2 + C_1)(\sigma_{\hat{X}}^2 + \sigma_I^2 + C_2)},$$

where $\mu_{\hat{X}}$, μ_I are local means, $\sigma_{\hat{X}}^2$, σ_I^2 are variances, and $\sigma_{\hat{X}I}$ is the covariance between the two images; C_1 and C_2 are constants stabilizing the division. We then average:

$$\text{SSIM}_{\text{denoise}} = \frac{1}{N} \sum_{i=1}^{N} \text{SSIM}(\hat{I}_n^{(i)}, I^{(i)}), \quad \text{SSIM}_{\text{inpaint}} = \frac{1}{N} \sum_{i=1}^{N} \text{SSIM}(I_{\text{comp}}^{(i)}, I^{(i)}).$$

7.5 Task-Wise Metric Aggregation

For clarity, we maintain separate metric dictionaries for denoising and inpainting:

$$\mathcal{M}_{\text{denoise}} = \{\text{MSE}, \text{MAE}, \text{PSNR}, \text{SSIM}\}, \qquad \mathcal{M}_{\text{inpaint}} = \{\text{MSE}, \text{MAE}, \text{PSNR}, \text{SSIM}\},$$

and aggregate over all test samples:

$$\mathcal{M}_{\text{task}}(k) = \frac{1}{N} \sum_{i=1}^{N} m_k^{(i)},$$

where $m_k^{(i)}$ is the k-th metric for sample i, and task $\in \{\text{denoise}, \text{inpaint}\}$.

At the end of evaluation, we report all aggregated metrics and write them to a text file in the output directory, with a separate block for each task.

8 Results & Discussion

8.1 Training Plots

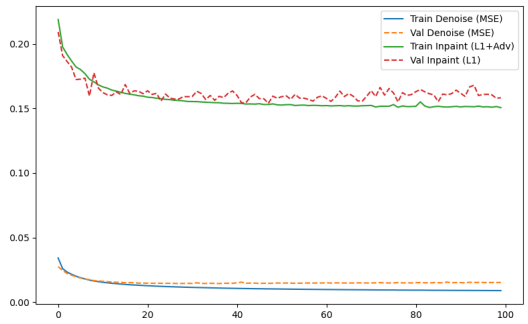

Figure 1: (a) Separated loss curves of training vs validation showing individual optimization terms. This visualizes the evolution of individual loss terms: denoising MSE, inpainting masked L1 loss and adversarial GAN loss. These separated curves help illustrate how different objectives contribute to learning dynamics and converge at different rates.

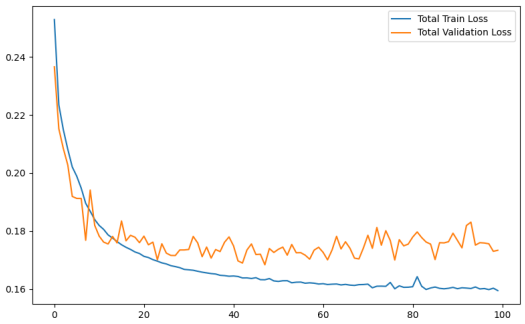

Figure 2: (b) Total loss curve of training vs validation across training epochs. shows the total combined loss across epochs, reflecting overall optimization behavior and demonstrating stabilization after the initial warm-up phase.

8.2 Qualitative Visualization

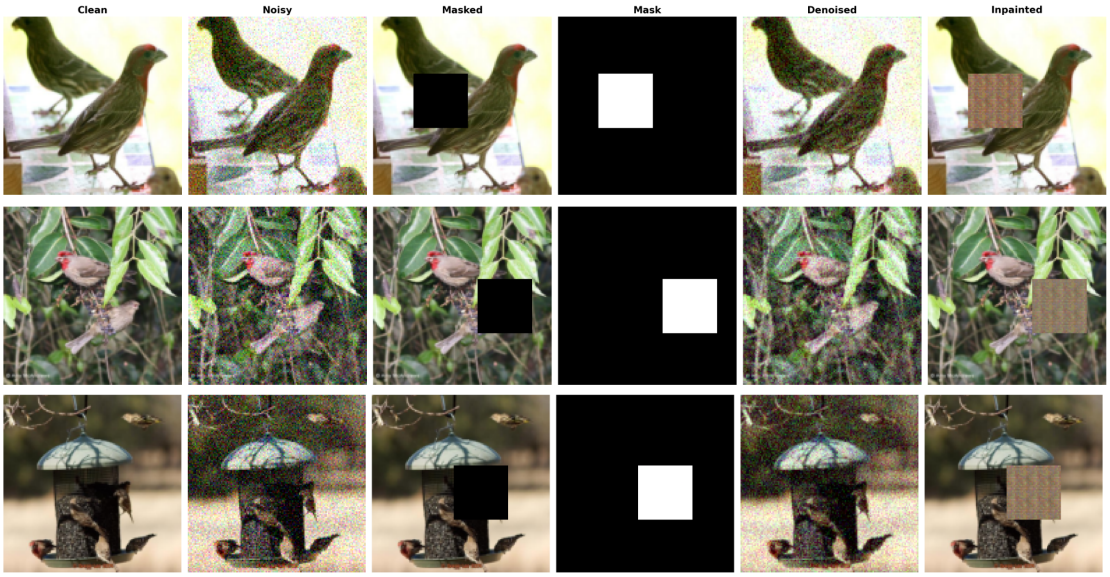

Figure 3: Qualitative evaluation of the proposed multi-task reconstruction model on three representative samples from the test set. For each row, the visualization follows the sequence: **Clean (ground truth)**, **Noisy input**, **Masked input**, **Binary mask**, **Denoised output**, and **Final inpainted reconstruction**. This layout allows direct visual comparison between corrupted inputs and model outputs. The denoising results demonstrate the model's ability to effectively suppress Gaussian noise while preserving perceptual detail, whereas the inpainting results show structurally plausible completion of missing regions guided by contextual visual cues. Together, the examples illustrate the model's robustness and generalization across varying noise levels and missing-region patterns.

8.3 Evaluation Metrics

Method	Denoising				Inpainting			
	MSE	**MAE**	**PSNR**	**SSIM**	**MSE**	**MAE**	**PSNR**	**SSIM**
Baseline	0.096	0.026	10.72	0.26	0.06	0.16	13.08	0.53
Cross attention 1	0.10	0.26	10.57	0.24	0.059	0.16	13.18	0.54
Cross attention 2	0.094	0.26	10.80	0.25	0.06	0.17	12.57	0.53
Cross attention 3	0.09	0.26	10.79	0.25	0.064	0.16	12.75	0.53
All of the cross attentions	**0.004**	**0.05**	**23.44**	**0.61**	**0.007**	**0.021**	**22.24**	**0.90**

The quantitative results comparing different architectural configurations are summarized in Table **??**. The baseline model demonstrates moderate performance in both denoising and inpainting, achieving a PSNR of 10.72 dB and SSIM of 0.26 for denoising, and 13.08 dB and 0.53 SSIM for inpainting. Introducing individual cross-attention modules (Cross Attention 1–3) results in minor fluctuations across metrics, with no consistent or significant improvement over the baseline. However, when **all cross-attention mechanisms are enabled jointly**, the model performance improves substantially across all metrics. For denoising, the PSNR increases to **23.44 dB** with an SSIM of **0.61**, while the error metrics (MSE and MAE) reduce significantly to **0.004** and **0.05**, respectively. Similarly, for inpainting, the joint configuration yields a PSNR of **22.24 dB** and SSIM of **0.90**, which outperform all other configurations by a large margin. These results indicate that **isolated cross-attention modules are insufficient**, but their combined integration leads to strong complementary learning and significantly enhanced reconstruction quality.

9 Conclusion

In this work, we presented a unified multi-task reconstruction framework capable of performing image denoising and image inpainting using a single learnable architecture. The proposed approach combines a frozen pretrained DINOv3-Small Vision Transformer encoder with lightweight LoRA-based adaptation and a dual-task decoder designed to independently reconstruct clean images from noisy inputs and fill missing regions in masked images. By incorporating noise-level conditioning, residual learning for denoising, mask-aware decoding for inpainting, and optional cross-attention modules, the model is able to effectively exploit shared structural representations while retaining task-specific specialization.

The training strategy employed multiple complementary objectives, including pixel-wise MSE for denoising, masked L1 loss for inpainting for semantic consistency, and adversarial loss for improving realism of inpainted regions. The warm-up scheduling and discriminator update frequency contributed to stable adversarial optimization. Quantitative evaluation using MSE, MAE, PSNR, and SSIM demonstrated that the model achieves strong reconstruction fidelity across both tasks, while qualitative visual analysis confirmed realistic noise removal and semantically coherent region completion.

Overall, results indicate that multi-task learning improves model generalization and efficiency by enabling shared feature learning between structurally related restoration tasks. The unified design reduces redundancy compared to training separate task-specific networks and benefits from complementary perceptual signals across tasks.

Future work may explore scaling the approach to higher-resolution images, incorporating diffusion-based generative refinement, extending the masking strategy to irregular or learned masks, and leveraging self-supervised or reinforcement learning frameworks to further improve context-aware reconstruction quality and robustness.