

# Prior-Guided Art Inpainting with Local Edge-Tuned Texture Enhancement

## 1. Problem Context and Project Summary

Museums, archives, and media teams often work with artworks that are scratched, cracked, or partially missing, and quick patch tools tend to blur texture and break the original style of the piece. This project aims to restore these damaged regions with style-consistent, curator-friendly results by using prior-guided inpainting and local edge-tuned enhancement. Concretely, the system learns from a corpus of damaged and undamaged artworks to fill missing areas in a way that preserves brushwork, color harmony, and surface grain. It blends masked reconstruction and perceptual texture losses with edge-aware guidance, so seams look natural, and boundaries stay sharp. A lightweight review interface lets a user upload an image, provide or auto-suggest a mask, preview the restoration, and inspect a change heatmap before export. The goal is to reduce manual retouch time while increasing authenticity and transparency, turning damaged assets into publishable, archivally respectful images.

## 2. Dataset

### Primary dataset: Damaged & Undamaged Artworks (Kaggle)

**Source and type:** A curated collection of RGB images of artworks labeled as Damaged or Undamaged. This provides real artwork content with a binary damage label for supervised tasks.

**Expected size:** Hundreds to a few thousand images depending on the current release.

Data format and access: Images are JPG or PNG organized in two folders (Damaged, Undamaged). Access via the Kaggle API:

```
kaggle datasets download -d pes1ug22am047/damaged-and-undamaged-artworks -p data/kaggle_art_damage --unzip
```

**Intended use in this project:** This is the main dataset for training and evaluation. I will use it to (1) build a simple damage classifier to understand dataset difficulty and (2) drive inpainting experiments by pairing images with synthetic or edge-guided masks to emulate cracks and tears.

### Auxiliary dataset: Art Images — Clear and Distorted (Kaggle)

**Source and type:** Artwork images accompanied by a large set of synthetic distortions. This helps with robustness, style diversity, and pretext tasks.

**Expected size:** Approximately 17,000 clear images and a large number of distorted variants produced by multiple distortion types.

Data format and access: Images are JPG or PNG. Access via the Kaggle API:

```
kaggle datasets download -d sankarmechengg/art-images-clear-and-distorted -p data/kaggle_art_clear_distorted --unzip
```

**Intended use in this project:** I will use the clear images for self-supervised inpainting pretraining with synthetic masks and the distorted set to test robustness and guide the design of damage-like mask shapes.

### 3. Preprocessing

#### Architecture and Algorithms to Test

##### 1. Baseline Classifier

I will first train a simple image classifier to confirm that the dataset contains a clear signal for distinguishing damaged from undamaged artworks. A compact CNN such as ResNet-18 or EfficientNet-B0 is sufficient for this sanity check and will also provide useful by-products like Grad-CAM saliency maps. These maps can highlight regions the model finds most informative and will help inspire realistic damage-like mask shapes for later inpainting experiments. Training will use cross-entropy loss, with optional inverse-frequency class weights if the Damaged/Undamaged split is imbalanced. This baseline establishes performance expectations, validates the data pipeline, and gives quick feedback on preprocessing choices before I invest in the restoration model.

##### 2. Semantic Inpainting

The core restoration system uses prior-guided inpainting to fill missing or damaged regions in a way that matches the artwork's style and texture. A frozen generative prior provides realism: either a StyleGAN-like prior for natural image statistics or a diffusion prior used as a denoiser/conditioner on masked inputs. A small trainable latent encoder maps the masked image (plus mask) to the prior's latent space for one-shot initialization, followed by a brief refinement to improve quality at inference time. To keep seams sharp and avoid blur at hole boundaries, a lightweight PatchGAN discriminator operates on local boundary patches. The loss blends masked L1/Charbonnier on known pixels and a boundary band, perceptual (LPIPS/VGG) to preserve structure and texture, style/Gram to maintain brushstroke and palette statistics, adversarial hinge loss at boundaries, and a mild total-variation term inside holes. Training uses two mask strategies: diverse irregular strokes/patches for coverage and edge/ridge-guided masks (Canny + Meijering/Fangi) that approximate real crack patterns.

##### 3. Ablations and Alternatives

I will run focused ablations to understand which components matter most and to balance quality with speed. These include enabling vs. disabling the PatchGAN boundary head, keeping or removing the style/Gram loss, and comparing irregular masks against damage-like masks to test realism vs. diversity trade-offs. I will also compare encoder-only inference (fastest) against encoder plus short refinement (higher quality) to determine a practical runtime target. Finally, I will test resolution trade-offs at  $256 \times 256$  and  $512 \times 512$  to measure gains in texture fidelity versus compute cost, and to decide a default resolution for the interactive UI.

#### Frameworks and Key Libraries

The project will be implemented primarily in PyTorch for modeling and training, with TorchVision supplying standard architectures and image transforms. To keep experiments

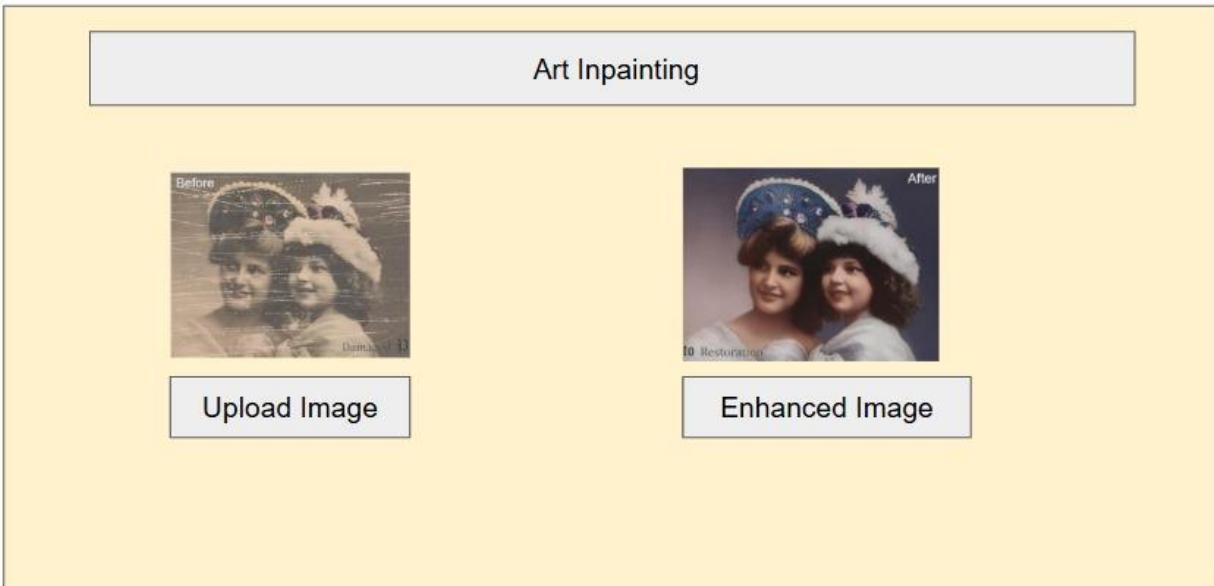
organized, I may use PyTorch Lightning for cleaner training loops and built-in logging. For perceptual similarity, I will rely on LPIPS, while OpenCV and scikit-image will support mask heuristics such as edge and ridge detection. NumPy and Matplotlib will handle exploratory data analysis and plotting. For evaluating generative quality, I may also include pytorch-fid to compute FID as an optional metric.

## Inference Pathway

1. Load trained weights (encoder + prior hook; discriminator off at inference).
2. Inputs: user artwork image and a binary mask. Optionally, generate a proposed mask automatically using edge/ridge detectors.
3. Latent encoder produces an initial latent; run a short refinement (few gradient steps) to minimize masked L1 + perceptual + style terms.
4. Produce restored image and a change heatmap (absolute difference or SSIM-based).
5. Add a small watermark or embed metadata noting restoration.
6. Return results to the UI for human review and export.

## 4. User interface plan

1. **User Inputs** — The UI accepts a JPG/PNG artwork upload and a damaged-area mask provided in one of three ways: (1) brush directly on the image to paint the region to restore, (2) upload a binary mask (white = hole), or (3) toggle Auto-Mask to propose a mask from edge/ridge detectors (user can refine with the brush). Optional controls let the user set refinement steps, edge-aware strength, style-preservation weight, and watermark on/off.
2. **Outputs** — The interface displays a side-by-side preview with a before/after slider, the restored image, and an optional change-heatmap overlay (with opacity control) that highlights edited pixels. It also shows lightweight reference indicators such as an LPIPS score and a simple style-similarity signal to suggest how well texture and palette were preserved. Export options include the restored image (watermarked by default) and a JSON log capturing mask, parameters, and timestamp.
3. **Feedback & Interpretability** — The brush + Auto-Mask workflow gives precise control over *where* restoration occurs, while the before/after slider and heatmap make edits transparent and easy to audit. Numeric indicators (LPIPS and style similarity) provide quick, interpretable cues about perceptual quality without replacing human judgment. Together these elements support responsible, curator-friendly decision-making.
4. **Implementation** — Built first in Streamlit for speed (file uploader, sidebar sliders, and an in-app mask editor via a drawable canvas component), with an optional Gradio demo for shareable prototypes. The app calls a PyTorch inference function that loads model weights, applies the user/auto mask, performs one-shot latent initialization plus short refinement, and returns the restored image, heatmap, and metrics. All exports embed provenance (settings + hash) and apply a subtle watermark by default.



## 5. Innovation and Anticipated Challenges

The approach is distinctive because it targets **artworks** rather than generic photos, combining a **prior-guided inpainting** pipeline (StyleGAN/diffusion prior kept frozen for realism) with **local edge-tuned guidance** so seams are crisp and repairs respect brushwork and canvas grain. Two ingredients make this technically challenging and novel in combination: (1) a **boundary-focused discriminator** (PatchGAN) that only “polices” the hole edges to prevent blur without heavy compute, and (2) a **style-aware objective** (perceptual + style/Gram + mild TV) that keeps palette, texture statistics, and stroke patterns consistent with the surrounding paint. Practically, a small **latent encoder** gives one-shot initialization and a **short refinement loop** balances quality with runtime, while the UI surfaces a **change heatmap** and **watermarked exports** to keep edits transparent and curator friendly.

1. **Key technical risk 1 — Style mismatch or visible seams.** Restorations may look photographic or blurry at boundaries. *Mitigation:* emphasize style/Gram loss and boundary-band L1, keep a lightweight PatchGAN on seam patches, add color/palette matching, and run ablations at  $256^2$  vs  $512^2$  to find the best fidelity—compute trade-off.
2. **Key technical risk 2 — Poor or overbroad masks (auto-mask errors).** Edge/ridge detectors can highlight texture that isn’t damage. *Mitigation:* keep auto-mask **optional** with a brush editor for correction, tune conservative thresholds, mix **irregular** and **damage-like** masks during training (curriculum), and provide instant previews plus the change heatmap so users can quickly adjust the region.
3. **Key technical risk 3 — Runtime and compute budget.** Diffusion/GAN priors can be slow or memory-heavy. *Mitigation:* freeze the prior, use a **trainable latent encoder** for rapid initialization, limit refinement steps, enable mixed precision, cache features where possible, and default to  $256 \times 256$  in the UI with an advanced  $512 \times 512$  mode for final exports.

4. **(Responsible-use risk, operational)** — Hallucination and provenance. *Mitigation:* watermark outputs by default, embed a provenance log (mask + settings + timestamp), show change heatmaps, and require human approval before export.

## 6. Implementation Timeline

| Week           | Focus  | Example  |
|----------------|--|--|
| Oct 20 - 26    | Data pipeline, EDA, and mask generators (irregular + edge/ridge)   | Clean train/val/test splits; loader verified; thumbnails, class counts, and masked-image demos saved to results/.                          |
| Oct 27 – Nov 2 | Baseline classifier (ResNet-18/EfficientNet-B0) + basic Streamlit UI   | Baseline accuracy logged; UI accepts image + mask and shows preview; LPIPS utility tested on sample pairs.                                 |
| Nov 3 – 16     | Inpainting v1: latent encoder + frozen prior; core losses (masked L1, perceptual, style/Gram, boundary band) | First believable restorations; qualitative panels (before/mask/after) and quick ablation notes saved.                                      |
| Nov 17 – 30    | Seam sharpening + UX: PatchGAN, change heatmap, watermark; UI sliders (style/edge/steps) and Auto-Mask       | Stable interactive flow (upload/brush/automatic mask → restore → slider/heatmap); cleaner seams; export with watermark + JSON provenance.  |
| Dec 1 – 11     | Metrics, polish, and packaging   | Best checkpoint selected; LPIPS/ SSIM/ PSNR (optional FID) reported; README/ run steps complete; short demo script and final report ready. |

## 7. Responsible AI Reflection

- **Fairness & Representation.** Inpainting can unintentionally alter culturally significant details (e.g., motifs, pigments, inscriptions) or bias toward certain artistic styles. I'll validate on a diverse set of artworks (periods, media, palettes), review failures with side-by-side panels and a change heatmap, and keep conservative defaults that minimize overediting. When edits could distort meaning, a “**Restore-only (no recolor/textured shift)**” mode will restrict changes to the smallest necessary region.

- **Privacy & Consent.** These datasets are digitized artworks, so PII is not expected. For user uploads, the app will process images in memory where possible, **strip metadata (EXIF)** on export, and avoid retaining files after download. If an image contains living persons or sensitive content, the user can opt to **mask/skip** those regions; the tool will not auto-inpaint unmasked areas.
- **Data Licensing & Provenance.** I'll respect the Kaggle dataset terms, keep attributions/links in the repo, and **avoid redistributing raw data**. Exports will include a **watermark** and a **provenance JSON** (dataset/source, mask, parameters, timestamp). Only the results derived allowed by the licenses will be shared; experiments will log dataset and version to ensure traceability.