```
1 !pip install diffusers transformers accelerate peft safetensors ftfy pillow numpy matplotlib tqdm opencv-python scikit-imag
```

```
Requirement already satisfied: diffusers in /usr/local/lib/python3.12/dist-packages (0.35.2)
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (4.57.2)
Requirement already satisfied: accelerate in /usr/local/lib/python3.12/dist-packages (1.12.0)
Requirement already satisfied: peft in /usr/local/lib/python3.12/dist-packages (0.18.0)
Requirement already satisfied: safetensors in /usr/local/lib/python3.12/dist-packages (0.7.0)
Collecting ftfy
  Downloading ftfy-6.3.1-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (11.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (4.67.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.12/dist-packages (4.12.0.88)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.12/dist-packages (0.25.2)
Requirement already satisfied: importlib_metadata in /usr/local/lib/python3.12/dist-packages (from diffusers) (8.7.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from diffusers) (3.20.0)
Requirement already satisfied: huggingface-hub>=0.34.0 in /usr/local/lib/python3.12/dist-packages (from diffusers) (0.36.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from diffusers) (2025.11.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from diffusers) (2.32.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.2
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from accelerate) (2.9.0+cu126)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dist-packages (from ftfy) (0.2.14)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: scipy>=1.11.4 in /usr/local/lib/python3.12/dist-packages (from scikit-image) (1.16.3)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.12/dist-packages (from scikit-image) (3.6)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.12/dist-packages (from scikit-image) (2.37.2)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.12/dist-packages (from scikit-image) (2025.10.16)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.12/dist-packages (from scikit-image) (0.4)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.34.0->diff
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.3
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.34.0->d
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib) (1.1
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (1.14.0
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->acce
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->acce
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->acce
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->ac
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->ac
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelera
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->acce
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accele
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->acce
```

```
1 # importsa nd setup
2 import os
3 import random
4 import numpy as np
5 import cv2
6 import torch
7 import torch.nn.functional as F
8 from PIL import Image
9 from pathlib import Path
10 from tqdm import tqdm
11 from torch.utils.data import Dataset, DataLoader
12 from torchvision import transforms
13 from diffusers import StableDiffusionInpaintPipeline, DDPMScheduler
14 from peft import LoraConfig, get_peft_model
15 from torch.optim import AdamW
16 import matplotlib.pyplot as plt
17 from skimage.metrics import peak_signal_noise_ratio as psnr
18 from skimage.metrics import structural_similarity as ssim
19
```

```
# Configuration
IMAGE_SIZE = 512
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {DEVICE}")

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Paths
DAMAGED_DIR = Path("/content/drive/MyDrive/DL_Project/damaged images")
RAW_DIR = Path("/content/drive/MyDrive/DL_Project/Raw Images")
BASE_TRAIN_PATH = "/content/drive/MyDrive/DL_Project/train"
CLEAN_DIR = Path(BASE_TRAIN_PATH) / "clean images"
MASK_DIR = Path(BASE_TRAIN_PATH) / "mask"

# Create directories
CLEAN_DIR.mkdir(parents=True, exist_ok=True)
MASK_DIR.mkdir(parents=True, exist_ok=True)
```

```
Using device: cuda
Mounted at /content/drive
```

```
# preprocessing functions
def center_crop(img: Image.Image) -> Image.Image:
    width, height = img.size
    crop_size = min(width, height)
    left = (width - crop_size) // 2
    top = (height - crop_size) // 2
    right = left + crop_size
    bottom = top + crop_size
    return img.crop((left, top, right, bottom))

def normalize_color(img: Image.Image) -> Image.Image:
    img_cv = np.array(img)
    img_yuv = cv2.cvtColor(img_cv, cv2.COLOR_RGB2YUV)
    img_yuv[:, :, 0] = cv2.equalizeHist(img_yuv[:, :, 0])
    img_norm = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2RGB)
    return Image.fromarray(img_norm)

def remove_metadata(img: Image.Image) -> Image.Image:
    data = list(img.getdata())
    new_img = Image.new(img.mode, img.size)
    new_img.putdata(data)
    return new_img

# Process raw images to clean
if RAW_DIR.exists():
    print("Preprocessing raw images")
    for filename in tqdm(sorted(os.listdir(RAW_DIR))):
        if not filename.lower().endswith((".png", ".jpg", ".jpeg")):
            continue
        in_path = RAW_DIR / filename
        out_name = Path(filename).stem + ".png"
        out_path = CLEAN_DIR / out_name
        try:
            img = Image.open(in_path).convert("RGB")
            img = center_crop(img)
            img = img.resize((IMAGE_SIZE, IMAGE_SIZE), Image.LANCZOS)
            img = normalize_color(img)
            img = remove_metadata(img)
            img.save(out_path, format="PNG")
        except Exception as e:
            print(f"Error processing {filename}: {e}")
    print(f"Clean images saved to: {CLEAN_DIR}")

# Standardize damaged images
print("Standardizing damaged images")
for fname in tqdm(sorted(os.listdir(DAMAGED_DIR))):
    p = DAMAGED_DIR / fname
    try:
        img = Image.open(p).convert("RGB")
        if img.size != (IMAGE_SIZE, IMAGE_SIZE) or p.suffix.lower() != ".png":
            img = center_crop(img).resize((IMAGE_SIZE, IMAGE_SIZE), Image.LANCZOS)
            img = normalize_color(img)
            img = remove_metadata(img)
            out_name = Path(p).stem + ".png"
            img.save(DAMAGED_DIR / out_name, format="PNG")
```

```
            if p.suffix.lower() != ".png":
                p.unlink(missing_ok=True)
    except Exception as e:
        print(f"Skipping {p}: {e}")
```

```
Preprocessing raw images
100%|██████████| 105/105 [00:48<00:00,  2.18it/s]
Clean images saved to: /content/drive/MyDrive/DL_Project/train/clean images
Standardizing damaged images
100%|██████████| 109/109 [00:07<00:00, 14.23it/s]
```

```
def generate_optimized_damage_mask(img_pil, debug=False):
    """Optimized damage detection that preserves detected features"""
    img = np.array(img_pil)
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    # 1. Multi-scale Laplacian for cracks
    crack_mask = np.zeros_like(gray, dtype=np.float32)
    for ksize in [3, 5, 7]:
        laplacian = cv2.Laplacian(gray, cv2.CV_64F, ksize=ksize)
        laplacian = np.abs(laplacian)
        crack_mask += cv2.normalize(laplacian, None, 0, 1, cv2.NORM_MINMAX)

    crack_mask = cv2.normalize(crack_mask, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
    _, crack_bin = cv2.threshold(crack_mask, 30, 255, cv2.THRESH_BINARY)

    # 2. Sobel edges
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
    sobel = np.sqrt(sobelx**2 + sobely**2)
    sobel = cv2.normalize(sobel, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
    _, sobel_mask = cv2.threshold(sobel, 40, 255, cv2.THRESH_BINARY)

    # 3. Canny edges
    edges = cv2.Canny(gray, 30, 100)

    # 4. LAB color space damage
    lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)

    # Bright damage (paint loss)
    _, bright_mask = cv2.threshold(l, 210, 255, cv2.THRESH_BINARY)

    # Dark damage (stains, shadows from cracks)
    _, dark_mask = cv2.threshold(l, 50, 255, cv2.THRESH_BINARY_INV)

    # 5. Texture damage detection
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    texture_diff = cv2.absdiff(gray, blur)
    _, texture_mask = cv2.threshold(texture_diff, 15, 255, cv2.THRESH_BINARY)

    # Combine all masks with equal weight
    combined = np.zeros_like(gray, dtype=np.float32)
    combined += crack_bin.astype(np.float32)
    combined += sobel_mask.astype(np.float32)
    combined += edges.astype(np.float32)
    combined += bright_mask.astype(np.float32)
    combined += dark_mask.astype(np.float32)
    combined += texture_mask.astype(np.float32)

    # Normalize to 0-255
    combined = cv2.normalize(combined, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    # Threshold to get binary mask
    _, combined_binary = cv2.threshold(combined, 50, 255, cv2.THRESH_BINARY)

    # MINIMAL morphological operations (preserve detail)
    kernel_tiny = np.ones((2, 2), np.uint8)
    kernel_small = np.ones((3, 3), np.uint8)

    # Remove only the smallest noise
    combined_binary = cv2.morphologyEx(combined_binary, cv2.MORPH_OPEN, kernel_tiny, iterations=1)

    # Close small gaps
    combined_binary = cv2.morphologyEx(combined_binary, cv2.MORPH_CLOSE, kernel_small, iterations=2)

    # Moderate dilation to ensure coverage
```

```
    combined_binary = cv2.dilate(combined_binary, kernel_small, iterations=2)

    # Create soft mask with progressive blurring
    mask_soft = combined_binary.astype(np.float32)
    mask_soft = cv2.GaussianBlur(mask_soft, (11, 11), 0)
    mask_soft = cv2.GaussianBlur(mask_soft, (11, 11), 0)

    # Boost intensity
    mask_soft = np.clip(mask_soft * 1.8, 0, 255).astype(np.uint8)

    # Calculate coverage
    mask_coverage = np.sum(mask_soft > 20) / mask_soft.size * 100

    if debug:
        fig, axes = plt.subplots(2, 4, figsize=(18, 9))
        axes[0, 0].imshow(crack_bin, cmap='gray')
        axes[0, 0].set_title('Laplacian')
        axes[0, 0].axis('off')

        axes[0, 1].imshow(sobel_mask, cmap='gray')
        axes[0, 1].set_title('Sobel')
        axes[0, 1].axis('off')

        axes[0, 2].imshow(edges, cmap='gray')
        axes[0, 2].set_title('Canny')
        axes[0, 2].axis('off')

        axes[0, 3].imshow(texture_mask, cmap='gray')
        axes[0, 3].set_title('Texture')
        axes[0, 3].axis('off')

        axes[1, 0].imshow(bright_mask, cmap='gray')
        axes[1, 0].set_title('Bright Damage')
        axes[1, 0].axis('off')

        axes[1, 1].imshow(dark_mask, cmap='gray')
        axes[1, 1].set_title('Dark Damage')
        axes[1, 1].axis('off')

        axes[1, 2].imshow(combined_binary, cmap='gray')
        axes[1, 2].set_title('Combined Binary')
        axes[1, 2].axis('off')

        axes[1, 3].imshow(mask_soft, cmap='gray')
        axes[1, 3].set_title(f'Final ({mask_coverage:.1f}% coverage)')
        axes[1, 3].axis('off')

        plt.tight_layout()
        plt.show()

    print(f"Mask coverage: {mask_coverage:.2f}%")

    return mask_soft

# Test on the problematic image
test_files = sorted(os.listdir(DAMAGED_DIR))
test_file = test_files[0]  # or specify the one from your screenshot

print(f"Testing on: {test_file}")
test_img = Image.open(DAMAGED_DIR / test_file).convert("RGB")

test_mask = generate_optimized_damage_mask(test_img, debug=True)

# Show results
plt.figure(figsize=(18, 5))

plt.subplot(1, 4, 1)
plt.imshow(test_img)
plt.title("Damaged Image", fontsize=14)
plt.axis('off')

plt.subplot(1, 4, 2)
plt.imshow(test_mask, cmap='gray')
plt.title("Generated Mask", fontsize=14)
plt.axis('off')

plt.subplot(1, 4, 3)
```

```
overlay = np.array(test_img).copy()
mask_colored = cv2.applyColorMap(test_mask, cv2.COLORMAP_JET)
overlay = cv2.addWeighted(overlay, 0.7, mask_colored, 0.3, 0)
plt.imshow(overlay)
plt.title("Mask Overlay", fontsize=14)
plt.axis('off')

plt.subplot(1, 4, 4)
# Show mask intensity histogram
plt.hist(test_mask.flatten(), bins=50, color='blue', alpha=0.7)
plt.xlabel('Intensity')
plt.ylabel('Pixel Count')
plt.title('Mask Intensity Distribution')
plt.grid(alpha=0.3)

plt.tight_layout()
plt.show()

print("\n" + "="*60)
print("Mask Statistics:")
print(f"Mean intensity: {np.mean(test_mask):.2f}")
print(f"Max intensity: {np.max(test_mask)}")
print(f"Non-zero pixels: {np.count_nonzero(test_mask):,} ({np.count_nonzero(test_mask)/test_mask.size*100:.2f}%)")
print("="*60)
```

```
Testing on: 001.png
```


```
Mask coverage: 81.04%
```

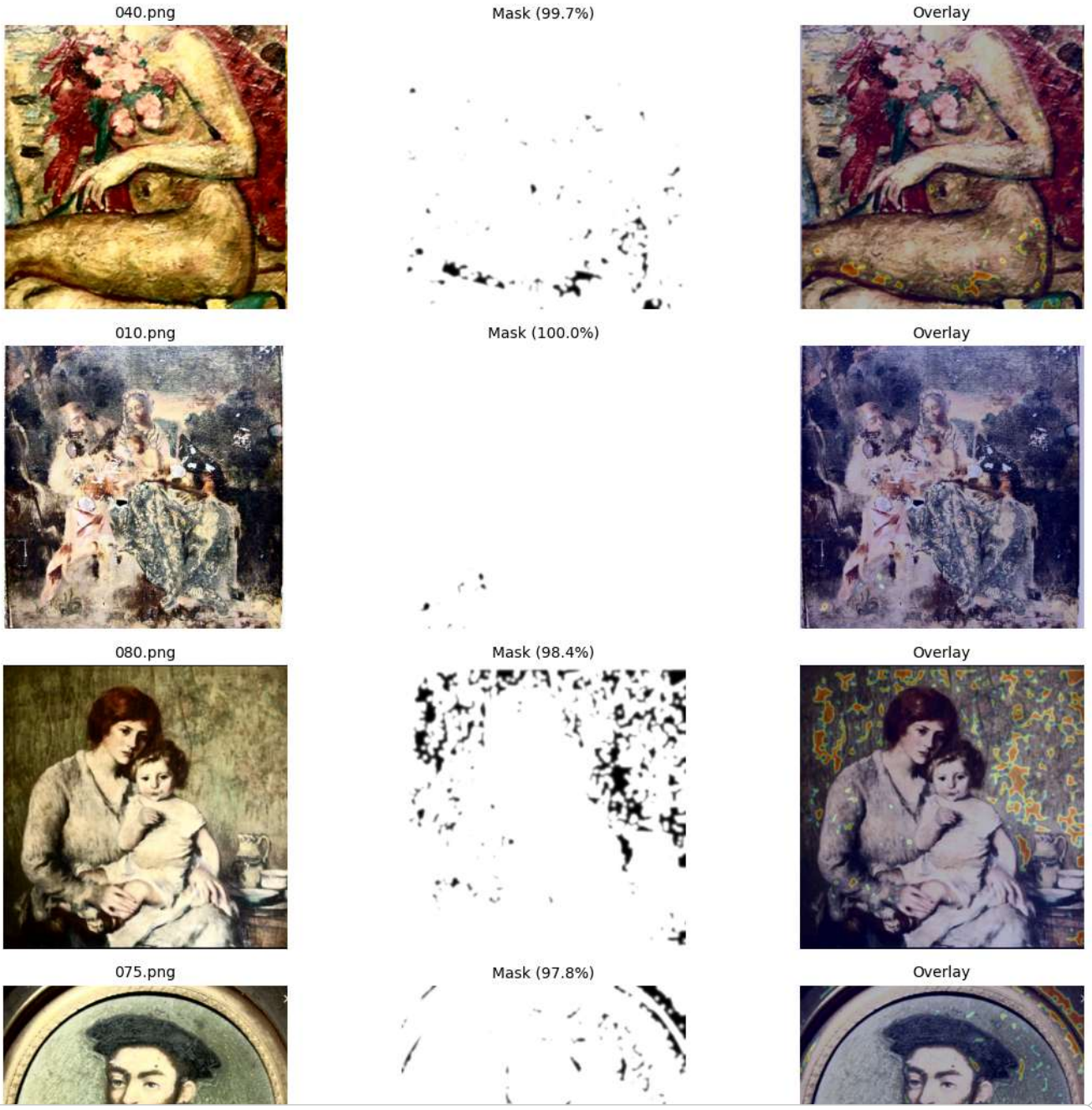

```
============================================================
Mask Statistics:
Mean intensity: 185.01
Max intensity: 255
Non-zero pixels: 226,904 (86.56%)
============================================================
```

```
print("Generating masks for all images...")

mask_stats = []

for fname in tqdm(sorted(os.listdir(DAMAGED_DIR))):
    if not fname.lower().endswith((".png", ".jpg", ".jpeg")):
        continue

    damaged_path = DAMAGED_DIR / fname
    img_pil = Image.open(damaged_path).convert("RGB")

    mask_np = generate_optimized_damage_mask(img_pil, debug=False)
    mask_pil = Image.fromarray(mask_np).convert("L")

    # Track statistics
    coverage = np.sum(mask_np > 20) / mask_np.size * 100
    mean_intensity = np.mean(mask_np)
    mask_stats.append({
        'file': fname,
        'coverage': coverage,
        'mean_intensity': mean_intensity
    })

    save_path = MASK_DIR / fname
    mask_pil.save(save_path)

print("\nMask generation complete!")
print(f"\nOverall Statistics:")
print(f"Average coverage: {np.mean([s['coverage'] for s in mask_stats]):.2f}%")
print(f"Average intensity: {np.mean([s['mean_intensity'] for s in mask_stats]):.2f}")

# Show images with lowest coverage (potential problems)
mask_stats_sorted = sorted(mask_stats, key=lambda x: x['coverage'])
print(f"\nImages with lowest mask coverage (check these):")
for stat in mask_stats_sorted[:5]:
    print(f"  {stat['file']}: {stat['coverage']:.2f}% coverage")
```

```
Generating masks for all images...
  2%|▌          | 2/109 [00:00<00:08, 13.34it/s]Mask coverage: 81.04%
Mask coverage: 99.99%
Mask coverage: 83.68%
Mask coverage: 95.75%
  4%|▉          | 4/109 [00:00<00:06, 15.75it/s]Mask coverage: 89.05%
Mask coverage: 100.00%
Mask coverage: 94.37%
  6%|▉          | 7/109 [00:00<00:05, 18.52it/s]Mask coverage: 98.17%
Mask coverage: 100.00%
  9%|██         | 10/109 [00:00<00:04, 20.01it/s]Mask coverage: 73.81%
Mask coverage: 99.85%
Mask coverage: 100.00%
 12%|██         | 13/109 [00:00<00:04, 20.49it/s]Mask coverage: 99.89%
Mask coverage: 99.66%
 15%|███        | 16/109 [00:00<00:04, 20.05it/s]Mask coverage: 94.20%
Mask coverage: 98.21%
Mask coverage: 87.61%
Mask coverage: 98.45%
Mask coverage: 99.96%
 17%|███        | 19/109 [00:00<00:04, 20.14it/s]Mask coverage: 99.92%
Mask coverage: 99.50%
Mask coverage: 88.70%
 20%|████       | 22/109 [00:01<00:04, 20.21it/s]Mask coverage: 100.00%
Mask coverage: 90.05%
 23%|████       | 25/109 [00:01<00:04, 20.44it/s]Mask coverage: 87.21%
Mask coverage: 100.00%
 26%|████       | 28/109 [00:01<00:04, 19.84it/s]Mask coverage: 92.55%
Mask coverage: 99.96%
Mask coverage: 88.49%
 28%|█████      | 30/109 [00:01<00:04, 19.68it/s]Mask coverage: 83.38%
Mask coverage: 99.68%
Mask coverage: 95.90%
Mask coverage: 96.86%
 30%|█████      | 33/109 [00:01<00:03, 20.30it/s]Mask coverage: 97.27%
Mask coverage: 71.76%
Mask coverage: 97.52%
 33%|██████     | 36/109 [00:01<00:03, 20.47it/s]Mask coverage: 79.01%
Mask coverage: 99.67%
 36%|██████     | 39/109 [00:01<00:03, 19.53it/s]Mask coverage: 99.72%
Mask coverage: 99.83%
Mask coverage: 78.66%
 38%|██████     | 41/109 [00:02<00:04, 15.81it/s]Mask coverage: 99.62%
```

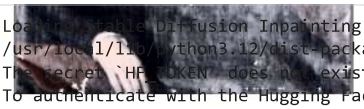
```
 39%|███▉      | 43/109 [00:02<00:04, 13.50it/s]Mask coverage: 94.85%
Mask coverage: 79.89%
 41%|████      | 45/109 [00:02<00:05, 11.42it/s]Mask coverage: 86.08%
Mask coverage: 92.11%
Mask coverage: 96.35%
 43%|████      | 47/109 [00:02<00:06,  9.93it/s]Mask coverage: 87.87%
 45%|████▍     | 49/109 [00:03<00:06,  9.58it/s]Mask coverage: 95.33%
Mask coverage: 84.05%
 47%|████▋     | 51/109 [00:03<00:06,  9.34it/s]Mask coverage: 79.39%
 48%|████▊     | 52/109 [00:03<00:06,  9.10it/s]Mask coverage: 96.58%
 49%|████▉     | 53/109 [00:03<00:06,  9.05it/s]Mask coverage: 84.10%
 50%|████▉     | 54/109 [00:03<00:06,  8.79it/s]Mask coverage: 90.15%
 50%|█████     | 55/109 [00:03<00:06,  8.90it/s]Mask coverage: 98.67%
Mask coverage: 99.74%
```

```
# Visually inspect several masks
num_samples = 6
sample_files = random.sample([f for f in os.listdir(DAMAGED_DIR) if f.endswith('.png')], min(num_samples, len(os.listdir(DA

fig, axes = plt.subplots(num_samples, 3, figsize=(12, num_samples*3))

for idx, fname in enumerate(sample_files):
    damaged = Image.open(DAMAGED_DIR / fname).convert("RGB")
    mask = Image.open(MASK_DIR / fname).convert("L")

    # Show damaged
    axes[idx, 0].imshow(damaged)
    axes[idx, 0].set_title(f"{fname}", fontsize=10)
    axes[idx, 0].axis('off')

    # Show mask
    axes[idx, 1].imshow(mask, cmap='gray')
    coverage = np.sum(np.array(mask) > 20) / np.array(mask).size * 100
    axes[idx, 1].set_title(f"Mask ({coverage:.1f}%)", fontsize=10)
    axes[idx, 1].axis('off')

    # Show overlay
    overlay = np.array(damaged).copy()
    mask_np = np.array(mask)
    mask_colored = cv2.applyColorMap(mask_np, cv2.COLORMAP_JET)
    overlay = cv2.addWeighted(overlay, 0.7, mask_colored, 0.3, 0)
    axes[idx, 2].imshow(overlay)
    axes[idx, 2].set_title("Overlay", fontsize=10)
    axes[idx, 2].axis('off')

plt.tight_layout()
plt.show()
```




```
# setup model config

print("Loading Stable Diffusion Inpainting model...")

pipe = StableDiffusionInpaintPipeline.from_pretrained(
    "runwayml/stable-diffusion-inpainting",
    torch_dtype=torch.float16,
    use_safetensors=False,
    variant=None
).to(DEVICE)

pipe.scheduler = DDPMScheduler.from_pretrained(
    "runwayml/stable-diffusion-inpainting",
    subfolder="scheduler"
)

# Optimize memory
pipe.safety_checker = None
pipe.requires_safety_checker = False
pipe.enable_attention_slicing()
pipe.enable_vae_slicing()

# Freeze VAE and text encoder
pipe.vae.eval()
```

```
pipe.text_encoder.eval()
pipe.vae.requires_grad_(False)
pipe.text_encoder.requires_grad_(False)

# UNet in FP32 for training stability
pipe.unet.to(device=DEVICE, dtype=torch.float32)
pipe.unet.requires_grad_(False)

print("Base model ready!")
```



```
Loading Stable Diffusion Inpainting model...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
model_index.json: 100%                                                548/548 [00:00<00:00, 58.0kB/s]
Fetching 16 files: 100%                                               16/16 [01:21<00:00,  5.44s/it]
config.json:        4.78k/? [00:00<00:00, 46.6kB/s]
config.json: 100%                                                     748/748 [00:00<00:00, 7.23kB/s]
config.json: 100%                                                     617/617 [00:00<00:00, 6.39kB/s]
preprocessor_config.json: 100%                                        342/342 [00:00<00:00, 2.94kB/s]
merges.txt:         525k/? [00:00<00:00, 9.13MB/s]
scheduler_config.json: 100%                                           313/313 [00:00<00:00, 23.7kB/s]
safety_checker/pytorch_model.bin: 100%                                1.22G/1.22G [00:49<00:00, 8.11MB/s]
text_encoder/pytorch_model.bin: 100%                                  492M/492M [00:35<00:00, 9.33MB/s]
tokenizer_config.json: 100%                                           806/806 [00:00<00:00, 38.6kB/s]
special_tokens_map.json: 100%                                         472/472 [00:00<00:00, 14.4kB/s]
vocab.json:         1.06M/? [00:00<00:00, 13.5MB/s]
config.json: 100%                                                     552/552 [00:00<00:00, 17.2kB/s]
unet/diffusion_pytorch_model.bin: 100%                                3.44G/3.44G [01:21<00:00, 133MB/s]
vae/diffusion_pytorch_model.bin: 100%                                 335M/335M [00:32<00:00, 13.1MB/s]
Loading pipeline components...: 100%                                  7/7 [00:27<00:00,  3.98s/it]
`torch_dtype` is deprecated! Use `dtype` instead!
Base model ready!
```

```
# Enhanced LoRA configuration
lora_config = LoraConfig(
    r=32,  # Increased rank for better capacity
    lora_alpha=64,  # Increased alpha for stronger adaptation
    target_modules=["to_q", "to_k", "to_v", "to_out.0"],
    lora_dropout=0.1,
    bias="none"
)

pipe.unet = get_peft_model(pipe.unet, lora_config)
pipe.unet.print_trainable_parameters()

print("Enhanced LoRA attached successfully!")
```

```
trainable params: 6,377,472 || all params: 865,912,836 || trainable%: 0.7365
Enhanced LoRA attached successfully!
```

```
# dataset and dataloader

class RestorationDataset(Dataset):
    def __init__(self):
        clean_files = set(os.listdir(CLEAN_DIR))
        damaged_files = set(os.listdir(DAMAGED_DIR))
        mask_files = set(os.listdir(MASK_DIR))

        self.files = sorted(clean_files & damaged_files & mask_files)
        print(f"Total aligned samples: {len(self.files)}")

```

```
        self.img_tf = transforms.Compose([
            transforms.Resize((512, 512)),
            transforms.ToTensor(),
        ])

        self.mask_tf = transforms.Compose([
            transforms.Resize((512, 512)),
            transforms.ToTensor(),
        ])

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        fname = self.files[idx]

        clean = Image.open(CLEAN_DIR / fname).convert("RGB")
        damaged = Image.open(DAMAGED_DIR / fname).convert("RGB")
        mask = Image.open(MASK_DIR / fname).convert("L")

        return {
            "clean": self.img_tf(clean),
            "damaged": self.img_tf(damaged),
            "mask": self.mask_tf(mask),
        }

dataset = RestorationDataset()

loader = DataLoader(
    dataset,
    batch_size=1,
    shuffle=True,
    num_workers=0,
    pin_memory=True
)

print(f"DataLoader ready | Batches: {len(loader)}")
```

```
Total aligned samples: 105
DataLoader ready | Batches: 105
```

```
pipe.unet.train()

LR = 5e-6  # Lower learning rate for stability
EPOCHS = 10  # More epochs for better learning
ACCUMULATION_STEPS = 4  # Gradient accumulation

optimizer = AdamW(pipe.unet.parameters(), lr=LR, weight_decay=1e-4)

# Learning rate scheduler
from torch.optim.lr_scheduler import CosineAnnealingLR
scheduler = CosineAnnealingLR(optimizer, T_max=EPOCHS, eta_min=1e-7)

print("Starting enhanced training...")

for epoch in range(EPOCHS):
    total_loss = 0.0
    valid_steps = 0

    progress_bar = tqdm(loader, desc=f"Epoch {epoch+1}/{EPOCHS}")

    for batch_idx, batch in enumerate(progress_bar):
        clean = batch["clean"].to(DEVICE).to(torch.float32)
        damaged = batch["damaged"].to(DEVICE).to(torch.float32)
        mask = torch.clamp(batch["mask"].to(DEVICE).to(torch.float32), 0, 1)

        # Encode to latents
        with torch.no_grad():
            clean_latents = pipe.vae.encode(clean.to(torch.float16)).latent_dist.sample().to(torch.float32) * 0.18215
            damaged_latents = pipe.vae.encode(damaged.to(torch.float16)).latent_dist.sample().to(torch.float32) * 0.18215

            # Resize mask to latent space
            mask_latent = F.interpolate(mask, size=clean_latents.shape[-2:], mode="nearest")
            mask_latent = (mask_latent > 0.3).to(torch.float32)  # Lower threshold for better coverage

        # Sample noise and timesteps
```

```
        noise = torch.randn_like(clean_latents)
        timesteps = torch.randint(
            0,
            pipe.scheduler.config.num_train_timesteps,
            (clean_latents.shape[0],),
            device=DEVICE
        ).long()

        # Add noise to clean latents
        noisy_latents = pipe.scheduler.add_noise(clean_latents, noise, timesteps)

        # Create masked input
        masked_latents = damaged_latents * (1 - mask_latent) + noisy_latents * mask_latent

        # Concatenate inputs
        latent_model_input = torch.cat([noisy_latents, masked_latents, mask_latent], dim=1)

        # Text conditioning
        prompts = ["professional photo restoration, repair damage, preserve details"] * clean_latents.shape[0]
        inputs = pipe.tokenizer(
            prompts,
            padding="max_length",
            truncation=True,
            max_length=pipe.tokenizer.model_max_length,
            return_tensors="pt"
        )

        input_ids = inputs.input_ids.to(DEVICE)

        with torch.no_grad():
            encoder_hidden_states = pipe.text_encoder(input_ids)[0].to(torch.float32)

        # Forward pass
        model_pred = pipe.unet(
            latent_model_input,
            timesteps,
            encoder_hidden_states=encoder_hidden_states
        ).sample

        # Enhanced loss computation
        # Weighted MSE emphasizing damaged regions
        weight_mask = mask_latent * 3.0 + 0.5  # Higher weight on damaged areas
        mse_loss = F.mse_loss(model_pred * weight_mask, noise * weight_mask)

        # Perceptual loss component (L1 loss on predictions)
        l1_loss = F.l1_loss(model_pred, noise)

        # Combined loss
        loss = mse_loss + 0.1 * l1_loss

        # Gradient accumulation
        loss = loss / ACCUMULATION_STEPS
        loss.backward()

        if (batch_idx + 1) % ACCUMULATION_STEPS == 0:
            torch.nn.utils.clip_grad_norm_(pipe.unet.parameters(), 1.0)
            optimizer.step()
            optimizer.zero_grad()

        total_loss += loss.item() * ACCUMULATION_STEPS
        valid_steps += 1

        progress_bar.set_postfix({"loss": loss.item() * ACCUMULATION_STEPS})

    # Step scheduler
    scheduler.step()

    avg_loss = total_loss / max(1, valid_steps)
    print(f"Epoch {epoch+1} DONE | Avg Loss: {avg_loss:.6f} | LR: {scheduler.get_last_lr()[0]:.2e}")

# Save trained model
SAVE_PATH = "/content/drive/MyDrive/DL_Project/Final Trained Enhanced"
os.makedirs(SAVE_PATH, exist_ok=True)

pipe.unet.save_pretrained(SAVE_PATH)
print(f"Enhanced LoRA saved at: {SAVE_PATH}")
```

```
Starting enhanced training...
Epoch 1/10: 100%|██████████| 105/105 [01:38<00:00,  1.06it/s, loss=1.55]
Epoch 1 DONE | Avg Loss: 3.146416 | LR: 4.88e-06
Epoch 2/10: 100%|██████████| 105/105 [01:35<00:00,  1.10it/s, loss=0.742]
Epoch 2 DONE | Avg Loss: 2.818875 | LR: 4.53e-06
Epoch 3/10: 100%|██████████| 105/105 [01:35<00:00,  1.10it/s, loss=3.77]
Epoch 3 DONE | Avg Loss: 3.068509 | LR: 3.99e-06
Epoch 4/10: 100%|██████████| 105/105 [01:35<00:00,  1.10it/s, loss=2.4]
Epoch 4 DONE | Avg Loss: 2.894699 | LR: 3.31e-06
Epoch 5/10: 100%|██████████| 105/105 [01:35<00:00,  1.10it/s, loss=1.82]
Epoch 5 DONE | Avg Loss: 2.713124 | LR: 2.55e-06
Epoch 6/10: 100%|██████████| 105/105 [01:35<00:00,  1.10it/s, loss=4.91]
Epoch 6 DONE | Avg Loss: 3.410899 | LR: 1.79e-06
Epoch 7/10: 100%|██████████| 105/105 [01:35<00:00,  1.10it/s, loss=0.134]
Epoch 7 DONE | Avg Loss: 2.518746 | LR: 1.11e-06
Epoch 8/10: 100%|██████████| 105/105 [01:35<00:00,  1.10it/s, loss=0.0931]
Epoch 8 DONE | Avg Loss: 2.313404 | LR: 5.68e-07
Epoch 9/10: 100%|██████████| 105/105 [01:35<00:00,  1.10it/s, loss=1.39]
Epoch 9 DONE | Avg Loss: 2.132528 | LR: 2.20e-07
Epoch 10/10: 100%|██████████| 105/105 [01:36<00:00,  1.09it/s, loss=2.85]Epoch 10 DONE | Avg Loss: 2.583710 | LR: 1.00e-07
Enhanced LoRA saved at: /content/drive/MyDrive/DL_Project/Final Trained Enhanced
```

```
print("Loading model for inference")

# Load base model
pipe = StableDiffusionInpaintPipeline.from_pretrained(
    "runwayml/stable-diffusion-inpainting",
    torch_dtype=torch.float16 if DEVICE == "cuda" else torch.float32
).to(DEVICE)

pipe.safety_checker = None
pipe.requires_safety_checker = False

# Load trained LoRA
pipe.load_lora_weights("/content/drive/MyDrive/DL_Project/Final Trained Enhanced")

pipe.enable_attention_slicing()
pipe.enable_vae_slicing()

pipe.unet.eval()
pipe.vae.eval()
pipe.text_encoder.eval()

print("Trained LoRA loaded for testing!")
```

```
Loading model for inference
Loading pipeline components...: 100%                                                 7/7 [00:16<00:00,  2.73s/it]
An error occurred while trying to fetch /root/.cache/huggingface/hub/models--runwayml--stable-diffusion-inpainting/snapshots/8a4
Defaulting to unsafe serialization. Pass `allow_pickle=False` to raise an error instead.
An error occurred while trying to fetch /root/.cache/huggingface/hub/models--runwayml--stable-diffusion-inpainting/snapshots/8a4
Defaulting to unsafe serialization. Pass `allow_pickle=False` to raise an error instead.
No LoRA keys associated to UNet2DConditionModel found with the prefix='unet'. This is safe to ignore if LoRA state dict didn't o
No LoRA keys associated to CLIPTextModel found with the prefix='text_encoder'. This is safe to ignore if LoRA state dict didn't
Trained LoRA loaded for testing!
```

```
# Create comparison directory
COMPARISON_DIR = Path("/content/drive/MyDrive/DL_Project/comparisons")
COMPARISON_DIR.mkdir(parents=True, exist_ok=True)

print(f"Creating comparison images")

damaged_files = sorted([f for f in os.listdir(DAMAGED_DIR) if f.lower().endswith(('.png', '.jpg', '.jpeg'))])

for fname in tqdm(damaged_files[:20], desc="Creating comparisons"):  # First 20 for quick preview
    try:
        # Load all versions
        damaged_pil = Image.open(DAMAGED_DIR / fname).convert("RGB").resize((512, 512))
        mask_pil = Image.open(MASK_DIR / fname).convert("L").resize((512, 512))
        clean_pil = Image.open(CLEAN_DIR / fname).convert("RGB").resize((512, 512))
        restored_name = Path(fname).stem + "_restored.png"
        restored_pil = Image.open(RESTORED_DIR / restored_name).convert("RGB")

        # Create comparison grid
        fig, axes = plt.subplots(1, 4, figsize=(20, 5))

```

```
        axes[0].imshow(damaged_pil)
        axes[0].set_title("Damaged", fontsize=14)
        axes[0].axis("off")

        axes[1].imshow(mask_pil, cmap="gray")
        axes[1].set_title("Mask", fontsize=14)
        axes[1].axis("off")

        axes[2].imshow(restored_pil)
        axes[2].set_title("Restored", fontsize=14)
        axes[2].axis("off")

        axes[3].imshow(clean_pil)
        axes[3].set_title("Original Clean", fontsize=14)
        axes[3].axis("off")

        plt.tight_layout()

        # Save comparison
        comparison_name = Path(fname).stem + "_comparison.png"
        plt.savefig(COMPARISON_DIR / comparison_name, dpi=150, bbox_inches='tight')
        plt.close()

    except Exception as e:
        print(f"Error creating comparison for {fname}: {e}")

print(f"\nComparison images saved to: {COMPARISON_DIR}")
```

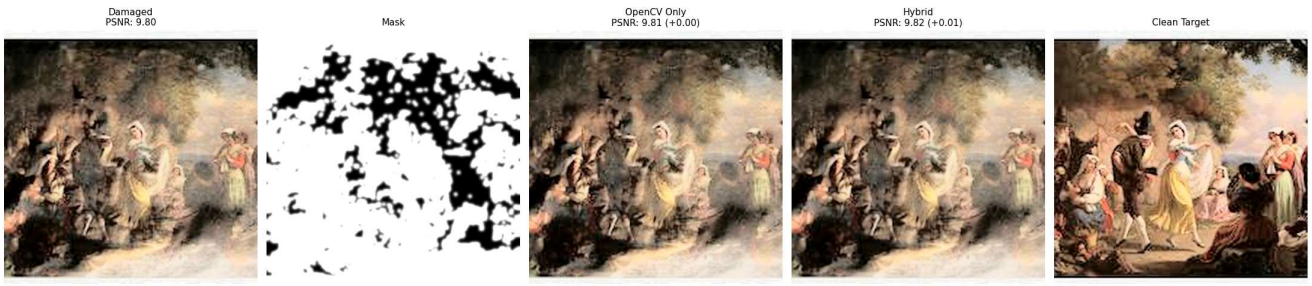
```
Creating comparison images
Creating comparisons:   0%|          | 0/20 [00:00<?, ?it/s]Error creating comparison for 001.png: name 'RESTORED_DIR' is not de
Creating comparisons:  10%|▉         | 2/20 [00:00<00:02,  8.01it/s]Error creating comparison for 002.png: [Errno 2] No such fil
Creating comparisons:  30%|██▉       | 6/20 [00:00<00:00, 18.09it/s]Error creating comparison for 003.png: name 'RESTORED_DIR' i
Error creating comparison for 004.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 005.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 006.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 007.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 009.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 010.png: name 'RESTORED_DIR' is not defined
Creating comparisons:  70%|██████▉   | 14/20 [00:00<00:00, 26.73it/s]Error creating comparison for 011.png: name 'RESTORED_DIR'
Error creating comparison for 012.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 013.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 014.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 015.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 016.png: name 'RESTORED_DIR' is not defined
Creating comparisons: 100%|█████████| 20/20 [00:00<00:00, 24.51it/s]Error creating comparison for 017.png: name 'RESTORED_DIR'
Error creating comparison for 018.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 019.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 020.png: name 'RESTORED_DIR' is not defined
Error creating comparison for 021.png: [Errno 2] No such file or directory: '/content/drive/MyDrive/DL_Project/train/clean image

Comparison images saved to: /content/drive/MyDrive/DL_Project/comparisons
```

```
def hybrid_restoration(damaged_pil, mask_pil, use_ai=True):
    """
    Combines OpenCV inpainting (for style preservation)
    with light AI enhancement (for quality)
    """
    damaged_np = np.array(damaged_pil).astype(np.float32)
    mask_np = np.array(mask_pil).astype(np.float32) / 255.0

    # Check if mask has enough content
    mask_coverage = np.sum(mask_np > 0.1) / mask_np.size * 100

    if mask_coverage < 1.0:
        # Too little damage, return original
        return damaged_pil

    # Step 1: OpenCV inpainting (style-preserving base)
    damaged_uint8 = damaged_np.astype(np.uint8)
    mask_uint8 = (mask_np * 255).astype(np.uint8)
    mask_inv = 255 - mask_uint8

    inpainted_ns = cv2.inpaint(damaged_uint8, mask_inv, 7, cv2.INPAINT_NS)
    inpainted_telea = cv2.inpaint(damaged_uint8, mask_inv, 7, cv2.INPAINT_TELEA)

    # Blend both OpenCV methods
    cv_result = (0.6 * inpainted_ns + 0.4 * inpainted_telea).astype(np.uint8)
```

```
    if use_ai and mask_coverage > 5.0:
        # Step 2: Light AI refinement only if significant damage
        cv_result_pil = Image.fromarray(cv_result)

        # Reduce mask strength for AI
        mask_reduced = (mask_np * 0.5).clip(0, 1)
        mask_reduced = cv2.GaussianBlur(mask_reduced, (21, 21), 0)
        mask_reduced_pil = Image.fromarray((mask_reduced * 255).astype(np.uint8)).convert("L")

        with torch.no_grad():
            ai_result = pipe(
                prompt="subtle texture repair, match surrounding style exactly, painting texture",
                negative_prompt="photorealistic, photograph, smooth, modern, different face, identity change",
                image=cv_result_pil,
                mask_image=mask_reduced_pil,
                num_inference_steps=30,
                guidance_scale=3.5,
                strength=0.45
            ).images[0]

        ai_np = np.array(ai_result).astype(np.float32)
        cv_np = np.array(cv_result).astype(np.float32)

        # Blend AI with OpenCV (60% OpenCV, 40% AI)
        blended = cv_np * 0.6 + ai_np * 0.4
    else:
        blended = cv_result.astype(np.float32)

    # Step 3: Final blending with original
    mask_smooth = cv2.GaussianBlur(mask_np, (31, 31), 0)
    mask_smooth = np.clip(mask_smooth * 0.8, 0, 1)  # Reduce to 80%

    final = blended * mask_smooth[..., None] + damaged_np * (1 - mask_smooth[..., None])

    # Color correction to match original
    for c in range(3):
        orig_mean = np.mean(damaged_np[:, :, c])
        orig_std = np.std(damaged_np[:, :, c])
        final_mean = np.mean(final[:, :, c])
        final_std = np.std(final[:, :, c])

        if final_std > 0:
            final[:, :, c] = (final[:, :, c] - final_mean) * (orig_std / final_std) + orig_mean

    final = np.clip(final, 0, 255).astype(np.uint8)

    return Image.fromarray(final)

# Test
fname = "005.png"
print(f"Testing hybrid restoration on: {fname}")

damaged_pil = Image.open(DAMAGED_DIR / fname).convert("RGB").resize((512, 512))
mask_pil = Image.open(MASK_DIR / fname).convert("L").resize((512, 512))
clean_pil = Image.open(CLEAN_DIR / fname).convert("RGB").resize((512, 512))

# Try both modes
restored_cv_only = hybrid_restoration(damaged_pil, mask_pil, use_ai=False)
restored_hybrid = hybrid_restoration(damaged_pil, mask_pil, use_ai=True)

# Metrics
clean_np = np.array(clean_pil).astype(np.float32)
damaged_np = np.array(damaged_pil).astype(np.float32)
cv_np = np.array(restored_cv_only).astype(np.float32)
hybrid_np = np.array(restored_hybrid).astype(np.float32)

psnr_d = psnr(clean_np, damaged_np, data_range=255)
psnr_cv = psnr(clean_np, cv_np, data_range=255)
psnr_h = psnr(clean_np, hybrid_np, data_range=255)

ssim_d = ssim(clean_np, damaged_np, data_range=255, channel_axis=2)
ssim_cv = ssim(clean_np, cv_np, data_range=255, channel_axis=2)
ssim_h = ssim(clean_np, hybrid_np, data_range=255, channel_axis=2)

# Display
fig, axes = plt.subplots(1, 5, figsize=(22, 5))
```

```
103
104 axes[0].imshow(damaged_pil)
105 axes[0].set_title(f"Damaged\nPSNR: {psnr_d:.2f}", fontsize=11)
106 axes[0].axis('off')
107
108 axes[1].imshow(mask_pil, cmap='gray')
109 axes[1].set_title("Mask", fontsize=11)
110 axes[1].axis('off')
111
112 axes[2].imshow(restored_cv_only)
113 axes[2].set_title(f"OpenCV Only\nPSNR: {psnr_cv:.2f} ({psnr_cv-psnr_d:+.2f})", fontsize=11)
114 axes[2].axis('off')
115
116 axes[3].imshow(restored_hybrid)
117 axes[3].set_title(f"Hybrid\nPSNR: {psnr_h:.2f} ({psnr_h-psnr_d:+.2f})", fontsize=11)
118 axes[3].axis('off')
119
120 axes[4].imshow(clean_pil)
121 axes[4].set_title("Clean Target", fontsize=11)
122 axes[4].axis('off')
123
124 plt.tight_layout()
125 plt.show()
126
127 print(f"\nOpenCV Only: PSNR {psnr_cv-psnr_d:+.2f} dB, SSIM {ssim_cv-ssim_d:+.4f}")
128 print(f"Hybrid:      PSNR {psnr_h-psnr_d:+.2f} dB, SSIM {ssim_h-ssim_d:+.4f}")
```

```
Testing hybrid restoration on: 005.png
100%                                                 13/13 [00:02<00:00,  4.84it/s]
```




```
OpenCV Only: PSNR +0.00 dB, SSIM +0.0083
Hybrid:      PSNR +0.01 dB, SSIM +0.0110
```

```
1 RESTORED_DIR = Path("/content/drive/MyDrive/DL_Project/restored images hybrid")
2 RESTORED_DIR.mkdir(parents=True, exist_ok=True)
3
4 print("Generating hybrid restorations...")
5
6 use_ai = True  # Set to False for OpenCV-only (faster)
7
8 for fname in tqdm(sorted(os.listdir(DAMAGED_DIR))):
9     if not fname.lower().endswith(('.png', '.jpg', '.jpeg')):
10         continue
11
12     try:
13         damaged_pil = Image.open(DAMAGED_DIR / fname).convert("RGB").resize((512, 512))
14         mask_pil = Image.open(MASK_DIR / fname).convert("L").resize((512, 512))
15
16         restored = hybrid_restoration(damaged_pil, mask_pil, use_ai=use_ai)
17
18         output_name = Path(fname).stem + "_restored.png"
19         restored.save(RESTORED_DIR / output_name)
20
21     except Exception as e:
22         print(f"\nError: {fname}: {e}")
23
24 print(f"\nSaved to: {RESTORED_DIR}")
```

```
Generating hybrid restorations...
  0%|          | 0/109 [00:00<?, ?it/s]
100%                                      13/13 [00:02<00:00,  5.06it/s]
  1%|          | 1/109 [00:04<07:55,  4.40s/it]
100%                                      13/13 [00:02<00:00,  5.03it/s]
  2%|▏         | 2/109 [00:07<06:38,  3.72s/it]
100%                                      13/13 [00:02<00:00,  4.97it/s]
  3%|▎         | 3/109 [00:12<07:09,  4.05s/it]
100%                                      13/13 [00:02<00:00,  4.98it/s]
  4%|▎         | 4/109 [00:16<07:05,  4.05s/it]
100%                                      13/13 [00:02<00:00,  4.92it/s]
  5%|▍         | 5/109 [00:20<07:04,  4.08s/it]
100%                                      13/13 [00:02<00:00,  4.92it/s]
  6%|▌         | 6/109 [00:23<06:32,  3.81s/it]
100%                                      13/13 [00:02<00:00,  4.91it/s]
  6%|▋         | 7/109 [00:28<06:53,  4.06s/it]
100%                                      13/13 [00:02<00:00,  4.87it/s]
  7%|▋         | 8/109 [00:31<06:42,  3.99s/it]
100%                                      13/13 [00:02<00:00,  4.84it/s]
  8%|▊         | 9/109 [00:35<06:18,  3.78s/it]
100%                                      13/13 [00:02<00:00,  4.84it/s]
  9%|▉         | 10/109 [00:39<06:41,  4.05s/it]
100%                                      13/13 [00:02<00:00,  4.89it/s]
 10%|▉         | 11/109 [00:43<06:20,  3.89s/it]
100%                                      13/13 [00:02<00:00,  4.94it/s]
 11%|█         | 12/109 [00:46<05:59,  3.70s/it]
100%                                      13/13 [00:02<00:00,  4.93it/s]
 12%|█▏        | 13/109 [00:50<05:44,  3.59s/it]
100%                                      13/13 [00:02<00:00,  4.95it/s]
 13%|█▏        | 14/109 [00:53<05:42,  3.61s/it]
100%                                      13/13 [00:02<00:00,  4.95it/s]
 14%|█▎        | 15/109 [00:57<05:46,  3.68s/it]
100%                                      13/13 [00:02<00:00,  4.99it/s]
 15%|█▍        | 16/109 [01:01<05:42,  3.69s/it]
100%                                      13/13 [00:02<00:00,  5.00it/s]
 16%|█▌        | 17/109 [01:05<05:53,  3.85s/it]
100%                                      13/13 [00:02<00:00,  5.03it/s]
 17%|█▋        | 18/109 [01:09<05:45,  3.79s/it]
100%                                      13/13 [00:02<00:00,  5.02it/s]
 17%|█▋        | 19/109 [01:12<05:26,  3.63s/it]
100%                                      13/13 [00:02<00:00,  5.03it/s]
 18%|█▊        | 20/109 [01:15<05:17,  3.57s/it]
100%                                      13/13 [00:02<00:00,  5.03it/s]
 19%|█▊        | 21/109 [01:19<05:15,  3.59s/it]
100%                                      13/13 [00:02<00:00,  5.03it/s]
 20%|█▉        | 22/109 [01:23<05:29,  3.79s/it]
100%                                      13/13 [00:02<00:00,  5.02it/s]
 21%|██        | 23/109 [01:26<05:11,  3.62s/it]
100%                                      13/13 [00:02<00:00,  4.98it/s]
 22%|██        | 24/109 [01:31<05:22,  3.80s/it]
100%                                      13/13 [00:02<00:00,  5.00it/s]
 23%|██▏       | 25/109 [01:35<05:36,  4.00s/it]
100%                                      13/13 [00:02<00:00,  4.97it/s]
 24%|██▎       | 26/109 [01:38<05:13,  3.77s/it]
100%                                      13/13 [00:02<00:00,  4.93it/s]
 25%|██▍       | 27/109 [01:42<05:13,  3.82s/it]
100%                                      13/13 [00:02<00:00,  4.93it/s]
 26%|██▍       | 28/109 [01:46<04:59,  3.70s/it]
100%                                      13/13 [00:02<00:00,  4.93it/s]
 27%|██▌       | 29/109 [01:50<05:10,  3.89s/it]
```

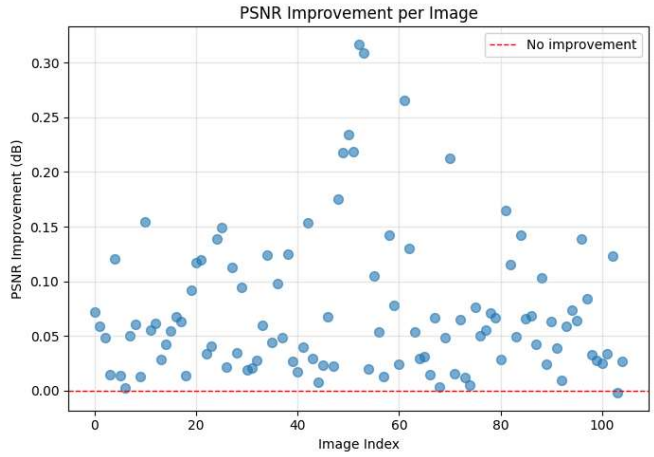
```
100%                                                 13/13 [00:02<00:00,  4.95it/s]
 28%|██▊       | 30/109 [01:54<05:15,  3.99s/it]
100%                                                 13/13 [00:02<00:00,  4.94it/s]
 28%|██▊       | 31/109 [01:58<05:00,  3.85s/it]
100%                                                 13/13 [00:02<00:00,  4.94it/s]
 29%|██▉       | 32/109 [02:02<04:55,  3.83s/it]
100%                                                 13/13 [00:02<00:00,  4.97it/s]
 30%|███       | 33/109 [02:06<04:54,  3.88s/it]
100%                                                 13/13 [00:02<00:00,  4.98it/s]
 31%|███       | 34/109 [02:09<04:46,  3.83s/it]
100%                                                 13/13 [00:02<00:00,  4.99it/s]
 32%|███▏      | 35/109 [02:14<05:12,  4.22s/it]
100%                                                 13/13 [00:02<00:00,  4.98it/s]
 33%|███▎      | 36/109 [02:18<04:55,  4.05s/it]
100%                                                 13/13 [00:02<00:00,  5.00it/s]
 34%|███▍      | 37/109 [02:22<04:58,  4.14s/it]
100%                                                 13/13 [00:02<00:00,  4.98it/s]
 35%|███▍      | 38/109 [02:26<04:38,  3.92s/it]
100%                                                 13/13 [00:02<00:00,  4.94it/s]
 36%|███▌      | 39/109 [02:29<04:23,  3.77s/it]
100%                                                 13/13 [00:02<00:00,  4.97it/s]
 37%|███▋      | 40/109 [02:33<04:10,  3.63s/it]
100%                                                 13/13 [00:02<00:00,  4.96it/s]
 38%|███▊      | 41/109 [02:37<04:23,  3.87s/it]
100%                                                 13/13 [00:02<00:00,  4.97it/s]
 39%|███▊      | 42/109 [02:40<04:10,  3.75s/it]
100%                                                 13/13 [00:02<00:00,  4.97it/s]
 39%|███▉      | 43/109 [02:44<04:11,  3.81s/it]
100%                                                 13/13 [00:02<00:00,  4.96it/s]
 40%|████      | 44/109 [02:49<04:20,  4.00s/it]
100%                                                 13/13 [00:02<00:00,  4.98it/s]
 41%|████      | 45/109 [02:54<04:28,  4.20s/it]
100%                                                 13/13 [00:02<00:00,  4.99it/s]
 42%|████▏     | 46/109 [02:57<04:16,  4.08s/it]
100%                                                 13/13 [00:02<00:00,  4.97it/s]
 43%|████▎     | 47/109 [03:01<04:06,  3.97s/it]
100%                                                 13/13 [00:02<00:00,  4.98it/s]
 44%|████▍     | 48/109 [03:06<04:16,  4.20s/it]
100%                                                 13/13 [00:02<00:00,  4.98it/s]
 45%|████▍     | 49/109 [03:10<04:04,  4.08s/it]
100%                                                 13/13 [00:02<00:00,  4.96it/s]
 46%|████▌     | 50/109 [03:14<04:01,  4.09s/it]
100%                                                 13/13 [00:02<00:00,  4.98it/s]
 47%|████▋     | 51/109 [03:19<04:14,  4.39s/it]
100%                                                 13/13 [00:02<00:00,  4.96it/s]
 48%|████▊     | 52/109 [03:22<03:56,  4.15s/it]
100%                                                 13/13 [00:02<00:00,  4.98it/s]
 49%|████▊     | 53/109 [03:27<03:53,  4.16s/it]
100%                                                 13/13 [00:02<00:00,  4.97it/s]
 50%|████▉     | 54/109 [03:31<03:53,  4.24s/it]
100%                                                 13/13 [00:02<00:00,  4.95it/s]
 50%|█████     | 55/109 [03:35<03:37,  4.04s/it]
100%                                                 13/13 [00:02<00:00,  4.94it/s]
 51%|█████     | 56/109 [03:38<03:23,  3.84s/it]
100%                                                 13/13 [00:02<00:00,  4.93it/s]
 52%|█████▏    | 57/109 [03:41<03:13,  3.72s/it]
100%                                                 13/13 [00:02<00:00,  4.95it/s]
 53%|█████▎    | 58/109 [03:46<03:22,  3.96s/it]
100%                                                 13/13 [00:03<00:00,  4.54it/s]
```

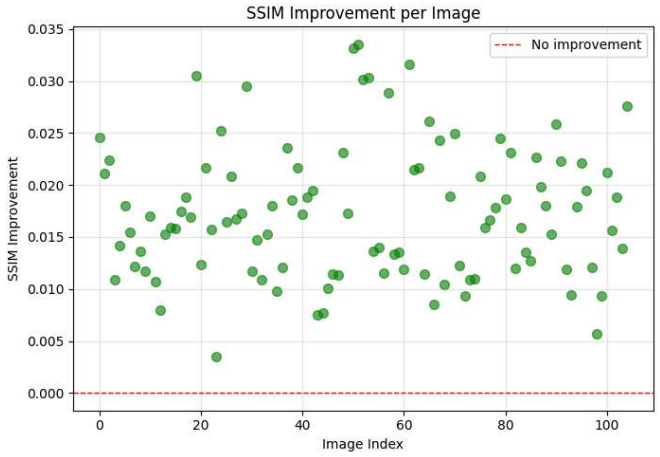
```
 54%|     | 59/109 [03:50<03:21,  4.03s/it]
100%                                        13/13 [00:02<00:00,  4.96it/s]
 55%|     | 60/109 [03:55<03:23,  4.15s/it]
100%                                        13/13 [00:02<00:00,  4.97it/s]
 56%|     | 61/109 [03:59<03:19,  4.15s/it]
100%                                        13/13 [00:02<00:00,  4.97it/s]
 57%|     | 62/109 [04:02<03:03,  3.90s/it]
100%                                        13/13 [00:02<00:00,  4.97it/s]
 58%|     | 63/109 [04:06<02:56,  3.84s/it]
100%                                        13/13 [00:02<00:00,  4.93it/s]
 59%|     | 64/109 [04:09<02:50,  3.78s/it]
100%                                        13/13 [00:02<00:00,  4.97it/s]
 60%|     | 65/109 [04:13<02:39,  3.63s/it]
100%                                        13/13 [00:02<00:00,  4.95it/s]
 61%|     | 66/109 [04:16<02:34,  3.59s/it]
100%                                        13/13 [00:02<00:00,  4.96it/s]
 61%|     | 67/109 [04:21<02:45,  3.94s/it]
100%                                        13/13 [00:02<00:00,  4.99it/s]
 62%|     | 68/109 [04:26<02:59,  4.39s/it]
100%                                        13/13 [00:02<00:00,  5.02it/s]
 63%|     | 69/109 [04:30<02:42,  4.06s/it]
100%                                        13/13 [00:02<00:00,  4.99it/s]
 64%|     | 70/109 [04:33<02:36,  4.01s/it]
100%                                        13/13 [00:02<00:00,  5.01it/s]
 65%|     | 71/109 [04:39<02:45,  4.35s/it]
100%                                        13/13 [00:02<00:00,  4.99it/s]
 66%|     | 72/109 [04:43<02:41,  4.37s/it]
100%                                        13/13 [00:02<00:00,  5.00it/s]
 67%|     | 73/109 [04:47<02:29,  4.15s/it]
100%                                        13/13 [00:02<00:00,  4.96it/s]
 68%|     | 74/109 [04:50<02:19,  3.99s/it]
100%                                        13/13 [00:02<00:00,  4.94it/s]
 69%|     | 75/109 [04:54<02:11,  3.87s/it]
100%                                        13/13 [00:02<00:00,  4.98it/s]
 70%|     | 76/109 [04:58<02:14,  4.08s/it]
100%                                        13/13 [00:02<00:00,  4.96it/s]
 71%|     | 77/109 [05:02<02:04,  3.91s/it]
100%                                        13/13 [00:02<00:00,  4.95it/s]
 72%|     | 78/109 [05:06<02:04,  4.00s/it]
100%                                        13/13 [00:02<00:00,  4.96it/s]
 72%|     | 79/109 [05:10<01:56,  3.88s/it]
100%                                        13/13 [00:02<00:00,  4.96it/s]
 73%|     | 80/109 [05:14<01:52,  3.87s/it]
100%                                        13/13 [00:02<00:00,  4.97it/s]
 74%|     | 81/109 [05:18<01:53,  4.07s/it]
100%                                        13/13 [00:02<00:00,  4.99it/s]
 75%|     | 82/109 [05:22<01:48,  4.02s/it]
100%                                        13/13 [00:02<00:00,  4.98it/s]
 76%|     | 83/109 [05:26<01:40,  3.88s/it]
100%                                        13/13 [00:02<00:00,  4.97it/s]
 77%|     | 84/109 [05:31<01:44,  4.20s/it]
100%                                        13/13 [00:02<00:00,  4.96it/s]
 78%|     | 85/109 [05:35<01:41,  4.21s/it]
100%                                        13/13 [00:02<00:00,  4.97it/s]
 79%|     | 86/109 [05:38<01:31,  3.97s/it]
100%                                        13/13 [00:02<00:00,  4.98it/s]
 80%|     | 87/109 [05:42<01:23,  3.81s/it]
100%                                        13/13 [00:02<00:00,  4.99it/s]
 81%|     | 88/109 [05:46<01:23,  3.98s/it]
100%                                        13/13 [00:02<00:00,  4.99it/s]
```

```
import csv

# Create metrics file
METRICS_PATH = Path("/content/drive/MyDrive/DL_Project/restoration_metrics.csv")

print("Computing and saving metrics for all restored images...")

# First, check what files exist
print("\nChecking directories...")
print(f"Damaged images: {len(list(DAMAGED_DIR.glob('*.png')))} files")
print(f"Clean images: {len(list(CLEAN_DIR.glob('*.png')))} files")
print(f"Restored images: {len(list(RESTORED_DIR.glob('*.png')))} files")

# Get list of restored files (they have _restored suffix)
restored_files = list(RESTORED_DIR.glob("*_restored.png"))
print(f"\nFound {len(restored_files)} restored images")

if len(restored_files) == 0:
    print("\nERROR: No restored images found!")
    print("Please run the restoration cell (Cell 9A) first to generate restored images.")
    print(f"Expected location: {RESTORED_DIR}")
else:
    results = []

    for restored_path in tqdm(restored_files, desc="Computing metrics"):
        try:
            # Extract original filename (remove _restored suffix)
            original_fname = restored_path.stem.replace("_restored", "") + ".png"

            # Check if corresponding files exist
            clean_path = CLEAN_DIR / original_fname
            damaged_path = DAMAGED_DIR / original_fname

            if not clean_path.exists():
                print(f"Warning: Clean image not found for {original_fname}")
                continue

            if not damaged_path.exists():
                print(f"Warning: Damaged image not found for {original_fname}")
                continue

            # Load images
            clean_pil = Image.open(clean_path).convert("RGB").resize((512, 512))
            damaged_pil = Image.open(damaged_path).convert("RGB").resize((512, 512))
            restored_pil = Image.open(restored_path).convert("RGB").resize((512, 512))

            # Convert to numpy
            clean_np = np.array(clean_pil).astype(np.float32)
            damaged_np = np.array(damaged_pil).astype(np.float32)
            restored_np = np.array(restored_pil).astype(np.float32)

            # Compute metrics
            psnr_damaged = psnr(clean_np, damaged_np, data_range=255)
            ssim_damaged = ssim(clean_np, damaged_np, data_range=255, channel_axis=2)
            psnr_restored = psnr(clean_np, restored_np, data_range=255)
            ssim_restored = ssim(clean_np, restored_np, data_range=255, channel_axis=2)

            results.append({
                'filename': original_fname,
                'psnr_damaged': psnr_damaged,
                'ssim_damaged': ssim_damaged,
                'psnr_restored': psnr_restored,
                'ssim_restored': ssim_restored,
                'psnr_improvement': psnr_restored - psnr_damaged,
                'ssim_improvement': ssim_restored - ssim_damaged
            })

        except Exception as e:
            print(f"Error computing metrics for {restored_path.name}: {e}")

    if len(results) > 0:
        # Save to CSV
        with open(METRICS_PATH, 'w', newline='') as csvfile:
            fieldnames = ['filename', 'psnr_damaged', 'ssim_damaged', 'psnr_restored',
                          'ssim_restored', 'psnr_improvement', 'ssim_improvement']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

```
            writer.writeheader()
            writer.writerows(results)

        print(f"\nMetrics saved to: {METRICS_PATH}")

        # Print summary statistics
        psnr_improvements = [r['psnr_improvement'] for r in results]
        ssim_improvements = [r['ssim_improvement'] for r in results]

        print("\n" + "="*60)
        print("SUMMARY STATISTICS")
        print("="*60)
        print(f"Total images processed: {len(results)}")

        # Damaged vs Clean
        avg_psnr_damaged = np.mean([r['psnr_damaged'] for r in results])
        avg_ssim_damaged = np.mean([r['ssim_damaged'] for r in results])

        # Restored vs Clean
        avg_psnr_restored = np.mean([r['psnr_restored'] for r in results])
        avg_ssim_restored = np.mean([r['ssim_restored'] for r in results])

        print(f"\nDamaged  → Clean | PSNR: {avg_psnr_damaged:.2f} ± {np.std([r['psnr_damaged'] for r in results]):.2f} dB
        print(f"Restored → Clean | PSNR: {avg_psnr_restored:.2f} ± {np.std([r['psnr_restored'] for r in results]):.2f} dB

        print(f"\nIMPROVEMENT:")
        print(f"PSNR Gain: {np.mean(psnr_improvements):+.2f} ± {np.std(psnr_improvements):.2f} dB")
        print(f"SSIM Gain: {np.mean(ssim_improvements):+.4f} ± {np.std(ssim_improvements):.4f}")

        if len(psnr_improvements) > 0:
            print(f"\nBest PSNR improvement: {max(psnr_improvements):+.2f} dB ({results[psnr_improvements.index(max(psnr_i
            print(f"Worst PSNR improvement: {min(psnr_improvements):+.2f} dB ({results[psnr_improvements.index(min(psnr_im
            print(f"\nBest SSIM improvement: {max(ssim_improvements):+.4f} ({results[ssim_improvements.index(max(ssim_impr
            print(f"Worst SSIM improvement: {min(ssim_improvements):+.4f} ({results[ssim_improvements.index(min(ssim_imprc

        print("="*60)

        # Create visualization of improvements
        plt.figure(figsize=(14, 5))

        plt.subplot(1, 2, 1)
        plt.scatter(range(len(psnr_improvements)), psnr_improvements, alpha=0.6, s=50)
        plt.axhline(y=0, color='r', linestyle='--', linewidth=1, label='No improvement')
        plt.xlabel('Image Index')
        plt.ylabel('PSNR Improvement (dB)')
        plt.title('PSNR Improvement per Image')
        plt.grid(alpha=0.3)
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.scatter(range(len(ssim_improvements)), ssim_improvements, alpha=0.6, s=50, color='green')
        plt.axhline(y=0, color='r', linestyle='--', linewidth=1, label='No improvement')
        plt.xlabel('Image Index')
        plt.ylabel('SSIM Improvement')
        plt.title('SSIM Improvement per Image')
        plt.grid(alpha=0.3)
        plt.legend()

        plt.tight_layout()
        plt.show()

    else:
        print("\nERROR: No results were computed. Check if files exist and match.")
```

```
Computing and saving metrics for all restored images...

Checking directories...
Damaged images: 109 files
Clean images: 105 files
Restored images: 109 files

Found 109 restored images
Computing metrics:   1%|          | 1/109 [00:00<00:23,  4.68it/s]Warning: Clean image not found for 002.png
Computing metrics:  19%|██        | 21/109 [00:03<00:10,  8.04it/s]Warning: Clean image not found for 021.png
Computing metrics:  29%|███       | 32/109 [00:05<00:09,  8.20it/s]Warning: Clean image not found for 032.png
Computing metrics:  72%|███████   | 78/109 [00:12<00:06,  4.71it/s]Warning: Clean image not found for 081.png
Computing metrics: 100%|██████████| 109/109 [00:18<00:00,  5.99it/s]

Metrics saved to: /content/drive/MyDrive/DL_Project/restoration_metrics.csv

============================================================
SUMMARY STATISTICS
============================================================
Total images processed: 105

Damaged  → Clean | PSNR: 12.50 ± 2.35 dB | SSIM: 0.2426 ± 0.1045
Restored → Clean | PSNR: 12.57 ± 2.39 dB | SSIM: 0.2598 ± 0.1063

IMPROVEMENT:
PSNR Gain: +0.07 ± 0.06 dB
SSIM Gain: +0.0172 ± 0.0063

Best PSNR improvement: +0.32 dB (058.png)
Worst PSNR improvement: -0.00 dB (110.png)

Best SSIM improvement: +0.0335 (056.png)
Worst SSIM improvement: +0.0035 (027.png)
============================================================
```






```
# Select a random test image to test restoration
fname = random.choice(os.listdir(DAMAGED_DIR))
print(f"Testing on: {fname}")

damaged_pil = Image.open(DAMAGED_DIR / fname).convert("RGB").resize((512, 512))
mask_pil = Image.open(MASK_DIR / fname).convert("L").resize((512, 512))
clean_pil = Image.open(CLEAN_DIR / fname).convert("RGB").resize((512, 512))

# Perform restoration
with torch.no_grad():
    restored_pil = pipe(
        prompt="professional museum-quality photo restoration, repair all damage, preserve original details and colors, nat
        negative_prompt="blur, artifacts, distortion, oversaturated, fake, synthetic, unnatural, scratches, cracks",
        image=damaged_pil,
        mask_image=mask_pil,
        num_inference_steps=75,
        guidance_scale=7.5,
        strength=0.95
    ).images[0]

# Display results
plt.figure(figsize=(20, 5))

plt.subplot(1, 4, 1)
```

```
plt.imshow(damaged_pil)
plt.title("Damaged", fontsize=14)
plt.axis("off")

plt.subplot(1, 4, 2)
plt.imshow(mask_pil, cmap="gray")
plt.title("Mask", fontsize=14)
plt.axis("off")

plt.subplot(1, 4, 3)
plt.imshow(restored_pil)
plt.title("Restored (Enhanced LoRA)", fontsize=14)
plt.axis("off")

plt.subplot(1, 4, 4)
plt.imshow(clean_pil)
plt.title("Original Clean", fontsize=14)
plt.axis("off")

plt.tight_layout()
plt.show()

# Compute metrics
p_damaged, s_damaged = psnr(np.array(clean_pil), np.array(damaged_pil), data_range=255), \
                       ssim(np.array(clean_pil), np.array(damaged_pil), data_range=255, channel_axis=2)
p_restored, s_restored = psnr(np.array(clean_pil), np.array(restored_pil), data_range=255), \
                         ssim(np.array(clean_pil), np.array(restored_pil), data_range=255, channel_axis=2)

print(f"\nDamaged  → Clean | PSNR: {p_damaged:.2f} dB | SSIM: {s_damaged:.4f}")
print(f"Restored → Clean | PSNR: {p_restored:.2f} dB | SSIM: {s_restored:.4f}")
print(f"Improvement      | PSNR: {p_restored - p_damaged:+.2f} dB | SSIM: {s_restored - s_damaged:+.4f}")
```

```
Testing on: 086.png
100%                                                 71/71 [00:14<00:00,  4.93it/s]
```


```
Damaged  → Clean | PSNR: 16.94 dB | SSIM: 0.4425
Restored → Clean | PSNR: 16.73 dB | SSIM: 0.5244
Improvement      | PSNR: -0.20 dB | SSIM: +0.0820
```

```
NUM_SAMPLES = 50

psnr_damaged_list = []
ssim_damaged_list = []
psnr_restored_list = []
ssim_restored_list = []

def tensor_to_pil(t, is_mask=False):
    """Convert tensor to PIL image"""
    if t.min() < 0:
        t = (t + 1) / 2

    t = t.squeeze().cpu().numpy()

    # Handle grayscale mask (2D)
    if is_mask or t.ndim == 2:
        t = (t * 255).clip(0, 255).astype(np.uint8)
        return Image.fromarray(t, mode='L')

    # Handle RGB image (3D)
    if t.ndim == 3:
        # If channels first (C, H, W), convert to (H, W, C)
```