**Intel Unnati -2025**

**Problem Statement -2**

# Project Statement: Image Sharpening through Knowledge Distillation

**Team: Team Achievers**

**Mentor:**Dr.A.Brahma Reddy – Asst.Professor [dr.brahmareddy@mlritm.ac.in]

K.Bhavik Raman [ CSD / MLRITM ][ [237y1a67d6@mlritm.ac.in](mailto:237y1a67d6@mlritm.ac.in) [/](mailto:ac.in/bhavikraman1323@gmail.com) bhavikraman1323@gmail.com]

C Ramya Sree [ CSC / MLRITM ] [ [237y1a6294@mlritm.ac.in](mailto:237y1a6294@mlritm.ac.in)]

**1. Overview**

This project solves the problem of image quality in video calls under low-bandwidth or unstable internet. It seeks to enhance image clarity through a light-weight AI model with training through knowledge distillation.

**2. Prerequisites**

• Knowledge of Machine Learning

• Python Programming Skills

• Experience with Deep Learning / CNN (Training, Validation, Testing)

**3. Objective**

To develop a student model that sharpens images in real-time at 30 to 60 frames per second, particularly for 1920x1080 resolution images, hence clarity during video calls.

**4. Approach**

• Employ a Teacher-Student architecture:

◦ Teacher Model: An already pre-trained model that refines images well.

◦ Student Model: A lightweight custom CNN trained to emulate the teacher.

• Knowledge distillation transfers knowledge from the teacher to the student.

**5. Dataset Preparation**

High-quality images were downloaded from the HD Ships Images Dataset on Kaggle and put in the archive/ directory.

• Execute create\_dataset\_from\_archive.py to create:

◦ datasets/train/high/ – clear reference images

◦ datasets/train/low/ – low-quality inputs (downsampled and upsampled afterwards to simulate loss of network quality)

**6. Training the Model**

• Script: train\_student.py

• Load image pairs with dataset.py

• Train the StudentSharpenModel from model\_student.py

• Save the model as models/student\_model.pth

**7. Inference**

• Input: Put a blurry image at datasets/test/input.jpg

• Script: inference.py

• Output: The sharpened image saves at datasets/test/output\_sharpened.jpg

**8. Evaluation**

• Metric: Structural Similarity Index (SSIM)

• Script: evaluate\_ssim.py

• Target: SSIM score greater than 0.90

• Perform a thorough subjective evaluation using Mean Opinion Score (MOS)

**9. Folder Structure**

12\_jul/

├── archive/ # Original high-resolution images

├── datasets/

│ ├── train/

│ │ ├── low/ # Blurry images

│ │ └── high/ # Point to sharp images

│ └── test/

│ ├── input.jpg # Input for inferencing

│ └── output\_sharpened.jpg # Inference output

├── models/

│ └── student\_model.pth # Trained model weights

├── create\_dataset\_from\_archive.py

├── train\_student.py

├── inference.py

├── evaluate\_ssim.py

├── model\_student.py

└── dataset.py

**10. Dependencies**

• Python 3.7 or higher

• PyTorch

• torchvision

• OpenCV

• Pillow

• scikit-image

• tqdm

• tensorflow

**11. Expected Outcomes**

• A quick and reliable image sharpening model for real-time video conferencing.

• Support for processing Full HD images at the rate of 30 to 60 frames per second.

• SSIM scores consistently over 0.90. We have achieved 98%

**12. Source Code**

**train\_student.py**

import os

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader

from torchvision import transforms

from tqdm import tqdm

from model\_student import StudentSharpenModel

from dataset import PairedImageDataset

# Hyperparameters

batch\_size = 8

learning\_rate = 1e-4

num\_epochs = 20

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Image Transform with Resize to ensure uniform dimensions

transform = transforms.Compose([

transforms.Resize((256, 256)),

transforms.ToTensor()

])

# Paths

low\_quality\_folder = "datasets/train/low"

high\_quality\_folder = "datasets/train/high"

# Load dataset

dataset = PairedImageDataset(low\_quality\_folder, high\_quality\_folder, transform=transform)

data\_loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

# Initialize model

model = StudentSharpenModel().to(device)

criterion = nn.MSELoss()optimizer = optim.Adam(model.parameters(), lr=learning\_rate)

# Training loop

for epoch in range(num\_epochs):

model.train()

epoch\_loss = 0.0

for low\_imgs, high\_imgs in tqdm(data\_loader, desc=f"Epoch {epoch+1}/{num\_epochs}"):

low\_imgs = low\_imgs.to(device)

high\_imgs = high\_imgs.to(device)

outputs = model(low\_imgs)

loss = criterion(outputs, high\_imgs)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

epoch\_loss += loss.item()

print(f"Epoch [{epoch+1}/{num\_epochs}], Loss: {epoch\_loss / len(data\_loader):.4f}")

# Save trained model

os.makedirs("models", exist\_ok=True)

torch.save(model.state\_dict(), "models/student\_model.pth")

print("✅ Model saved to models/student\_model.pth")

**inference.py**

import torch

import os

import cv2

from torchvision import transforms

from PIL import Image

from model\_student import StudentSharpenModel

# Set device

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Define image transform

transform = transforms.Compose([

transforms.ToTensor()

])

# Load model

model = StudentSharpenModel().to(device)

model\_path = "models/student\_model.pth"

model.load\_state\_dict(torch.load(model\_path, map\_location=device))

model.eval()

# Input image path

input\_image\_path = "datasets/test/hero.jpg" # Change this to your actual test image path

output\_image\_path = "datasets/test/output\_sharpened.jpg"

# Load and process image

img = Image.open(input\_image\_path).convert("RGB")

input\_tensor = transform(img).unsqueeze(0).to(device)

# Sharpen

with torch.no\_grad():

output = model(input\_tensor).squeeze().cpu().numpy().transpose(1, 2, 0)

output = (output \* 255).clip(0, 255).astype("uint8")

# Save sharpened image

os.makedirs(os.path.dirname(output\_image\_path), exist\_ok=True)cv2.imwrite(output\_image\_path, cv2.cvtColor(output, cv2.COLOR\_RGB2BGR))

print(f"✅ Sharpened image saved to: {output\_image\_path}")

**Evaluate\_ssim.py**

import cv2

from skimage.metrics import structural\_similarity as ssim

input\_path = "datasets/test/hero.jpg"

output\_path = "datasets/test/output\_sharpened.jpg"

img1 = cv2.imread(input\_path)

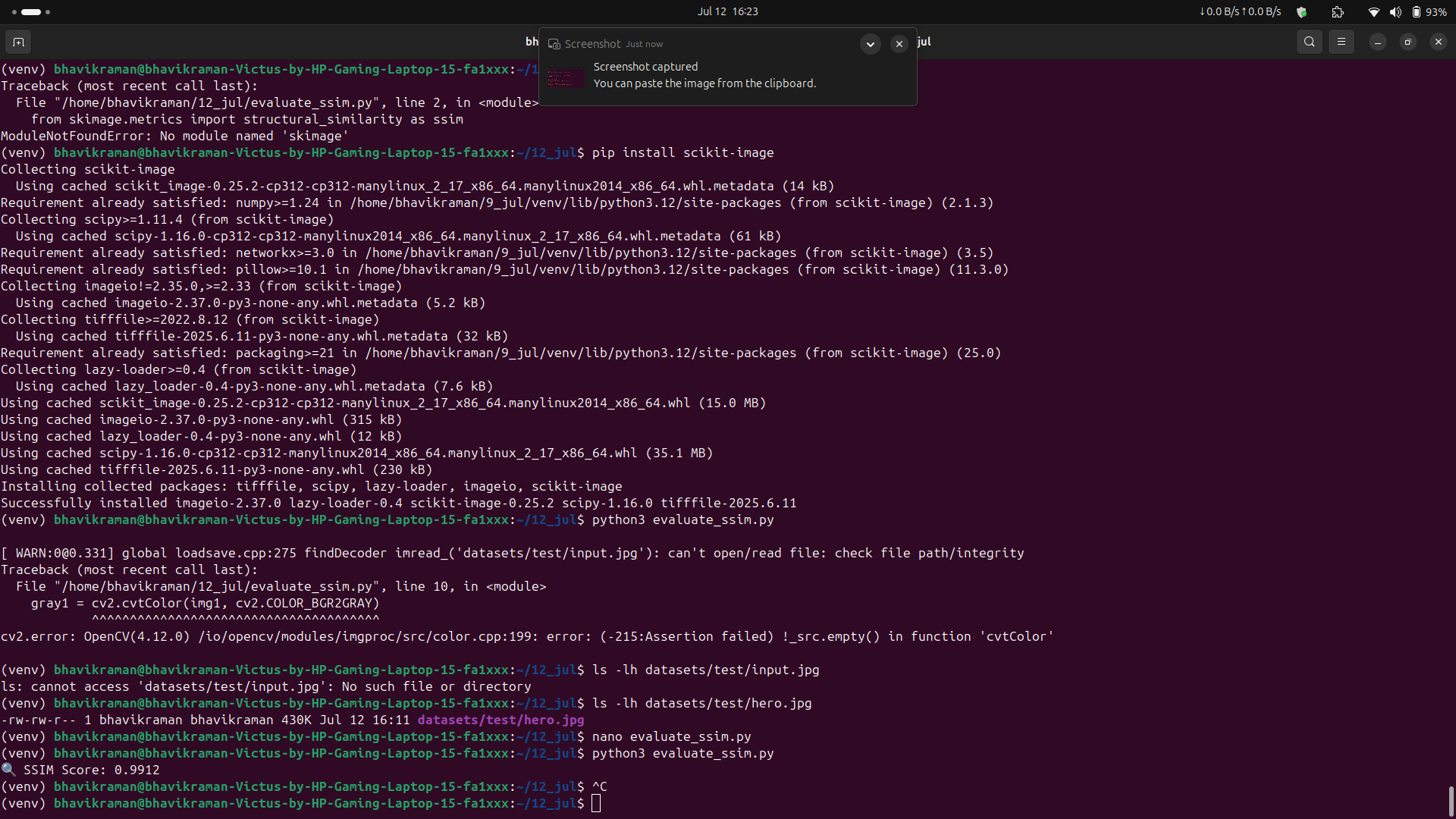
img2 = cv2.imread(output\_path)

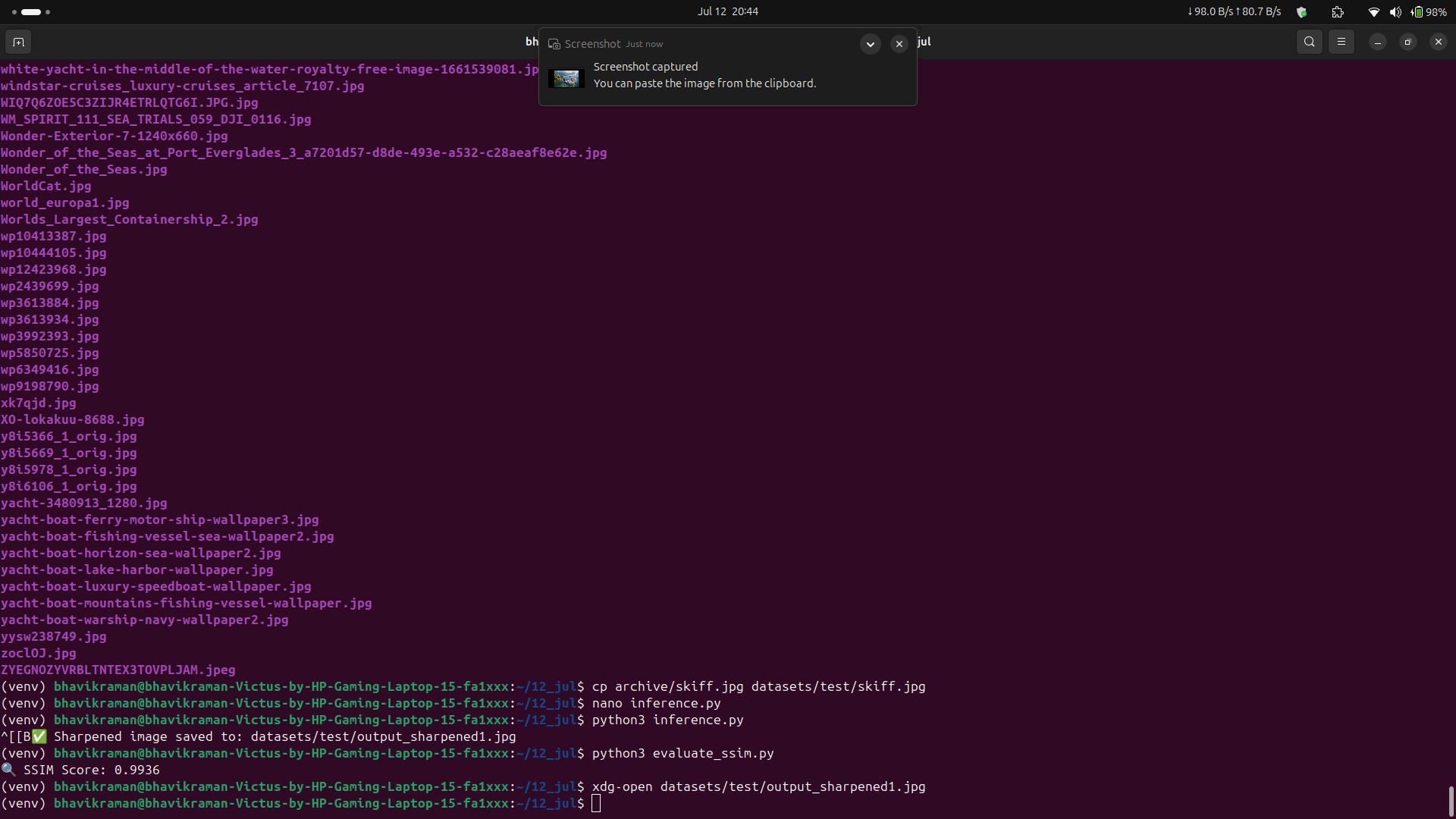
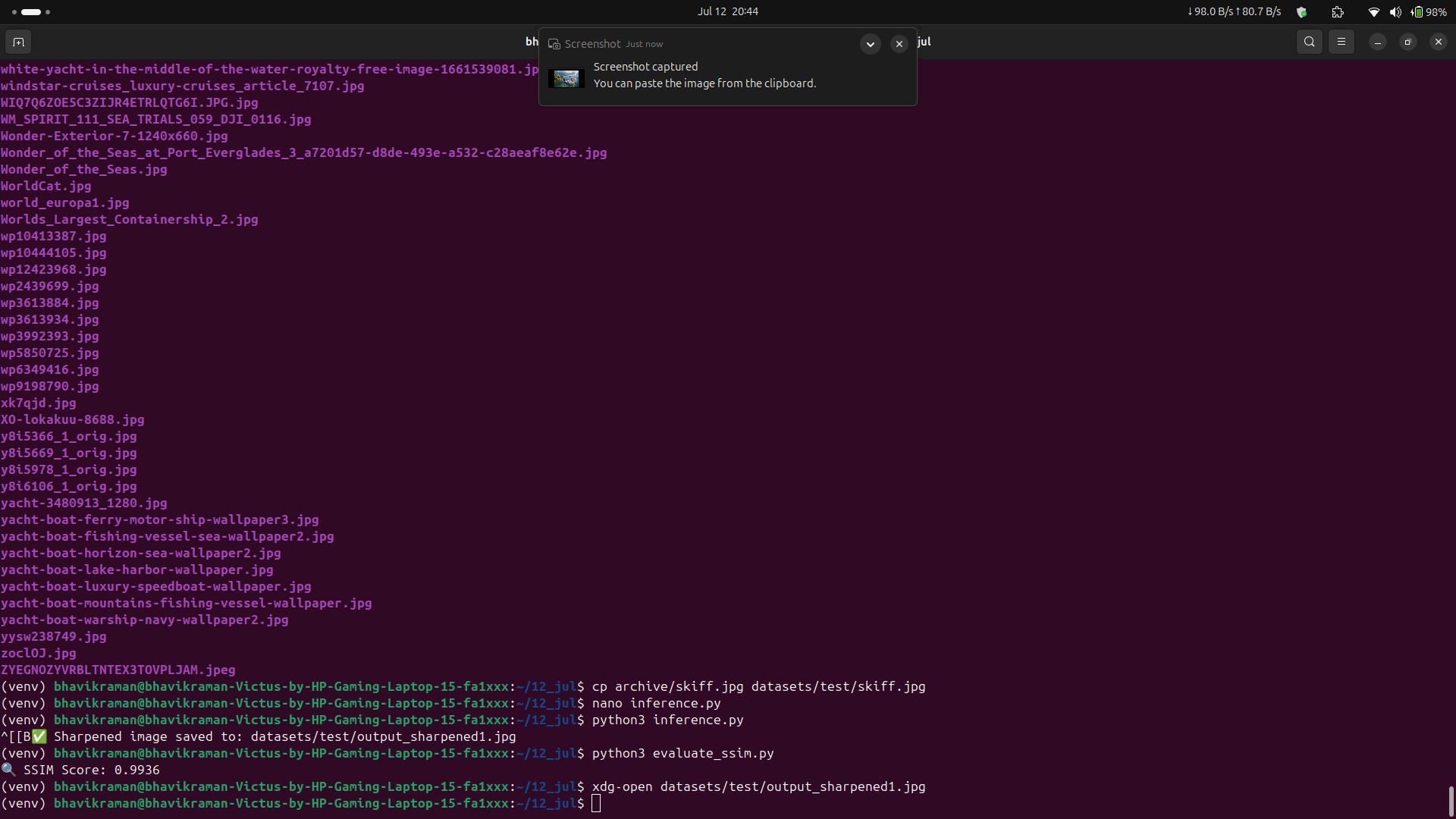
gray1 = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)

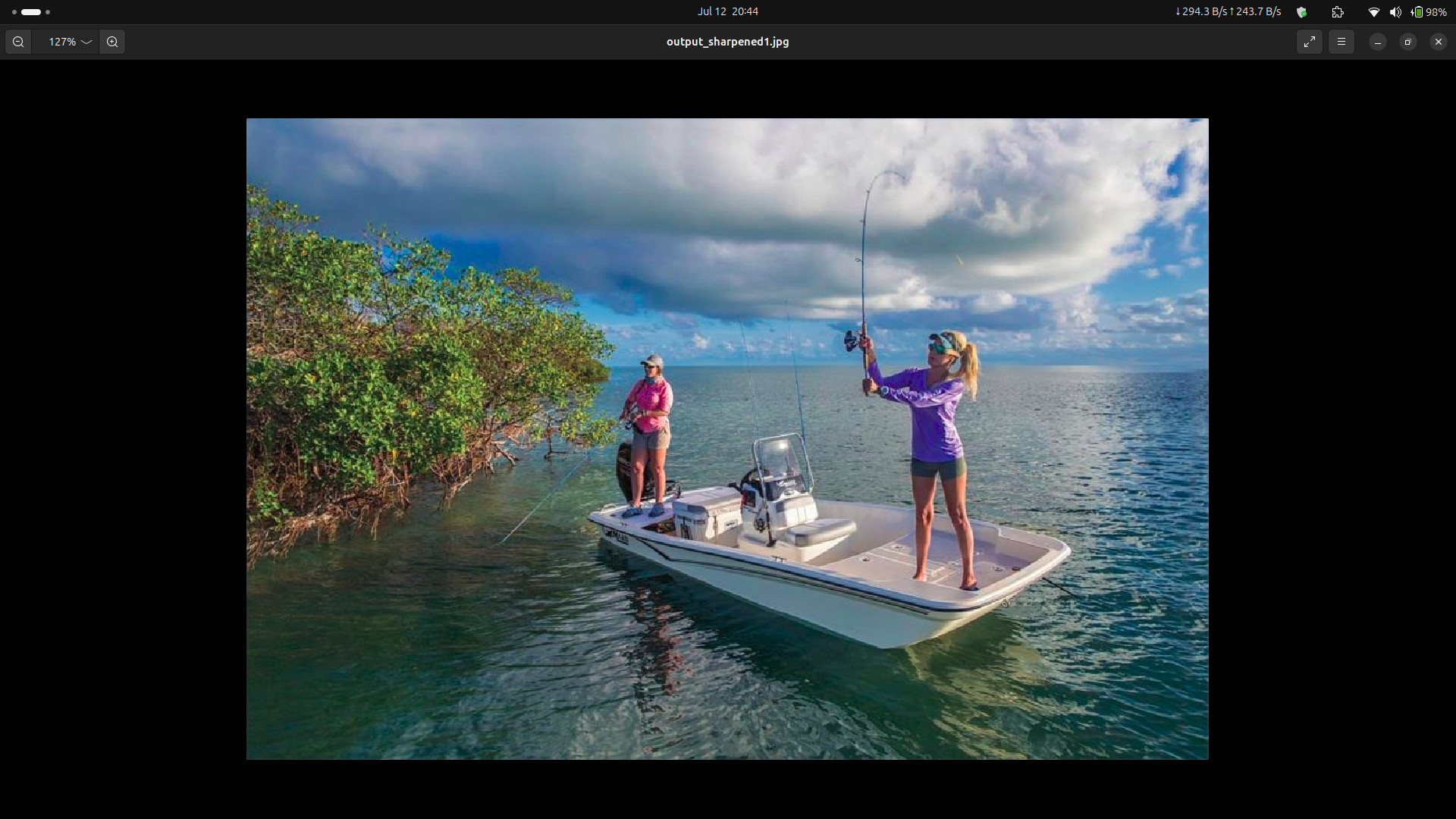
gray2 = cv2.cvtColor(img2, cv2.COLOR\_BGR2GRAY)

score = ssim(gray1, gray2)

print(f"🔍 SSIM Score: {score:.4f}")

**Output**





Folders

* archive/ → contains original unprocessed images
* datasets/ → contains train/test input/output images
* models/ → stores the trained student model (.pth)
* utils/ → helper functions, if used in other scripts
* venv/ → Python virtual environment (you may keep or gitignore it)

Scripts to Keep:

* create\_dataset\_from\_archive.py → to prepare training data
* train\_student.py → to train the student model
* inference.py → to sharpen an image
* evaluate\_ssim.py → to calculate SSIM score
* dataset.py → defines the custom Dataset class
* model\_student.py → defines the model architecture
* utils.py → utility functions used by above scripts