INTEL-UNNATI AI&ML JOURNAL
Number 2 (2025), pp. 1–10

Code:IUIT-2025

MACHINE LEARNING

# Image Sharpening Using Knowledge Distillation

**Team Luminous**

Fahim Shafeek, Angela Mary Thomas, Lakshmi Krishna V R

Saintgits Group of Institutions, Kottayam, Kerala

**Abstract:** In the age of video calls and live streaming, image quality often takes a hit due to compression, motion blur, or a weak Internet connection. In this project, we solve the problem using knowledge distillation, a smart deep learning-based image sharpening solution. We use images that have been intentionally blurred to mimic real-world scenarios. Here an ultra-lightweight student model is trained to mimic the large, high-performance teacher Model. When tested with the Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR), the student model shows that it can restore sharpness effectively, even with fewer resources. Overall, this project shows how knowledge distillation training a smaller model to learn from a larger model can produce high-quality results without a heavy processing load.

**Keywords:** Image sharpening, Knowledge distillation, Convolutional neural networks, Teacher-student model.

## 1 Introduction

In today's world, especially with the surge in remote work and online education, video conferencing has become an essential part of daily life. However, one of the most common challenges faced in live video transmission is blurred image quality due to motion blur or bandwidth limitations. This significantly reduces the clarity of visuals, affecting the user experience. To address this, we propose a project that enhances image sharpness in real-time video streams using knowledge distillation, a powerful technique in machine learning [4]. The model leverages a high-capacity teacher network to train a lightweight student model that can efficiently restore image quality with minimal computational cost [3]. A high-quality dataset, DIV2K, is used to support deep learning techniques. Our project aims to improve the SSIM and PSNR scores and real-time performance while offering a practical guide for deploying AI-powered image enhancement tools in low-bandwidth scenarios.

## 2 Related Works

| S.No. | Authors | Key Findings |
|---|---|---|
| 1 | Geoffrey Hinton, Oriol Vinyals, Jeff Dean [4] | Introduces knowledge distillation where a student model learns from the soft outputs of a larger trained teacher model. |
| 2 | Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner [6] | Provides the architectural foundation (CNNs) for both the teacher and student model and demonstrated end-to-end learning with convolutional layers for image recognition. |
| 3 | Bharat Bhusan Sau, Vineeth N. Balasubramanian [9] | Focuses on learning from noisy teachers for compressing models into ultra-lightweight student versions. |
| 4 | Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, Manmohan Chandraker [1] | Applies distillation for efficient vision models contributing to lightweight real-time image processing. |

Table 1: Reference papers used for Image sharpening using Knowledge Distillation

## 3 Libraries Used

In the project, the following Python libraries are used:

```
    NumPy
    PyTorch
    Seaborn
    PIL
    Matplotlib
    TQDM
    Scikit-learn
```

## 4 Methodology

In this work, two different kinds of deep learning models are employed. The first is a Teacher Model with high capacity, trained for optimal image restoration. The second is a lightweight Student Model, trained using knowledge distillation to mimic the teacher's performance while being efficient for real-time use. The overall methodology can be broken down into the following key steps:

**Data Loading:** The high-resolution DIV2K dataset is used [7], consisting of 800 images. Out of these, 700 are used for training and 100 for testing.

**Pre-processing:** High-quality images are artificially degraded to simulate real-world blurry conditions by applying bicubic downscaling, Gaussian blur, and upscaling operations.

**Dataset Preparation:** For each image, a random 256x256 patch is extracted to serve as high-quality ground truth. The corresponding low-quality version is created using degradation techniques as mentioned above.

**Teacher Model Training:** A deep convolutional neural network is taken and trained using L1 loss, which helps preserve sharp image structures. This model serves as a high-performing reference.

**Student Model Training:** A compact CNN is trained using a composite loss L1 loss with the ground truth image and MSE loss with the teacher model's output. This allows the student to mimic both the output and internal behavior of the teacher.

**Model Evaluation:** The teacher and student models are evaluated on the test dataset using SSIM and PSNR to quantify their restoration performance.

**Visualization:** Visual comparisons are made between the blurry input, student output, teacher output, and ground truth images to assess perceptual improvements.

**SSIM & PSNR:** Structural Similarity Index Measure [10]is a perceptual metric which measures the similarity between the two images. It just compares pixel differences. SSIM considers image structure, luminance, and contrast, which are more aligned with how humans perceive images.

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{1}$$

$\mu_x$ Mean (average intensity) of the image patch $x$.
$\mu_y$ Mean of the image patch $y$.
$\sigma_x^2$ Variance of $x$, measuring contrast in image patch $x$.
$\sigma_y^2$ Variance of $y$, measuring contrast in image patch $y$.
$\sigma_{xy}$ Covariance between $x$ and $y$, showing how their variations correspond.
$C_1$, $C_2$ Small constants to stabilize the division.

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

$MAX_I$ Maximum possible pixel value of the image.
$MSE$ Mean Squared Error between the original and the distorted image.
$\log_{10}$ Base-10 logarithm.
$10$ scaling factor.

## 5 Implementation

The project starts by importing the necessary libraries that are needed for the project, after which a CUDA-enabled GPU is used to speed up training. The next step was the preparation of the dataset, which was done by dividing the dataset into 700 images for training and 100 images for testing. Creating the blurred sharp image pairs involves extracting patches of 256x256 from the high-resolution images, downscaling it using bicubic interpolation, applying a bit of Gaussian blur, and then upscaling them [3]. Once this is done, these samples are loaded into Pytorch dataloader objects for proper shuffling and batching.

To implement the concept of knowledge distillation involves the use of two models, a teacher model and a student model, which is a lightweight model, whereas the teacher model is a large-capacity heavy model with more than 2.5 million parameters and its goal is to achieve the best performance. In contrast, the Student is an ultra-lightweight model with only around 13,000 parameters, making it over 180 times smaller and significantly faster for increasing training stability, both the teacher and the student model make use of residual connections [7]. To sharpen the blurry inputs, the teacher model is trained over 200 epochs. The student is then trained using a combination of MSE loss, which is against the teacher's output and L1 loss, which is against the ground truth, helping it to learn well from the teacher model. Alongside, the student also tries to imitate the internal layer structure of the teacher model. SSIM and PSNR are used for measuring their performance and visual comparisons are produced to display the blurry input, the teacher's output, student's output and the ground truth.

## 6 Results & Discussion

The Training was conducted on local machine, with the teacher trained for 200 epochs and the student for 300. The training result is summarized in the table 2.

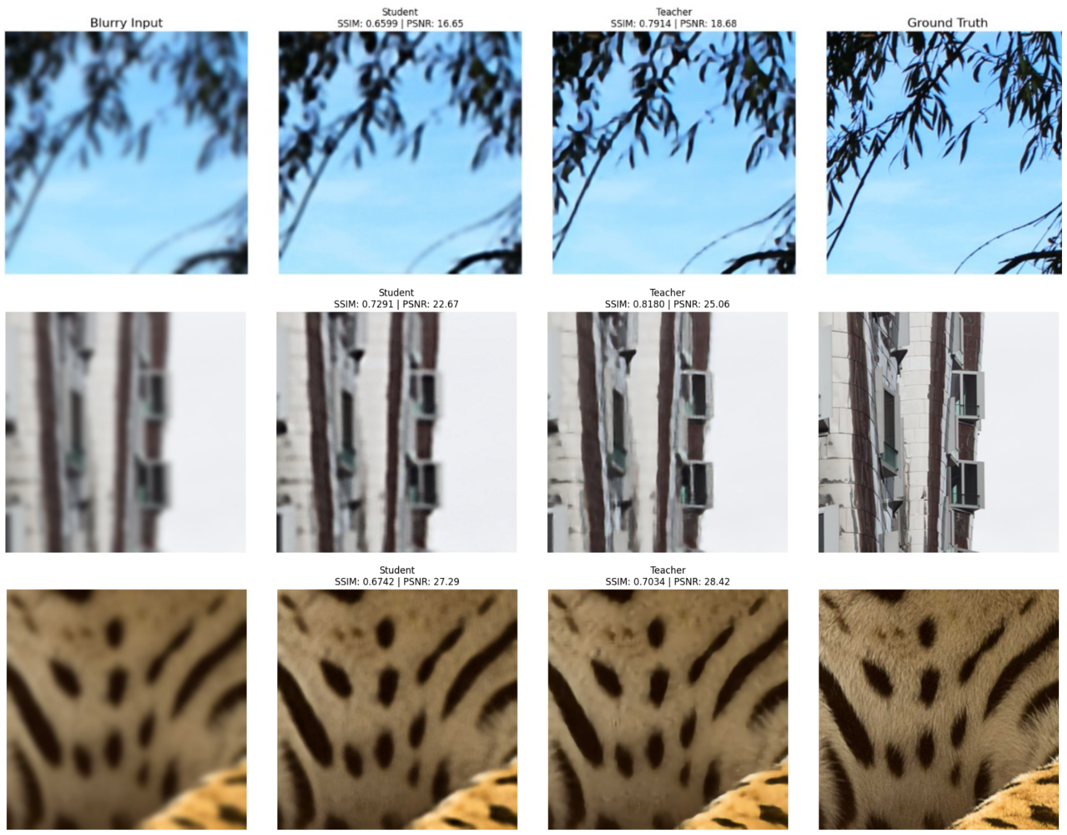
Blurry Input
Student
SSIM: 0.6599 | PSNR: 16.65
Teacher
SSIM: 0.7914 | PSNR: 18.68
Ground Truth
Student
SSIM: 0.7291 | PSNR: 22.67
Teacher
SSIM: 0.8180 | PSNR: 25.06
Student
SSIM: 0.6742 | PSNR: 27.29
Teacher
SSIM: 0.7034 | PSNR: 28.42

Figure 1: Comparison of blurry input, student output, teacher output, and ground truth images for sample test cases.

| Model | SSIM vs Ground Truth | PSNR (dB) |
|---|---|---|
| **Teacher Model** | **0.7943** | **27.48** |
| **Student Model** | **0.7119** | **25.78** |

Table 2: SSIM and PSNR scores of Teacher and Student Models on 100 DIV2K images.

| Model | SSIM vs Teacher Model | PSNR vs Teacher Model |
|---|---|---|
| **Student Model** | **0.9137** | **34.09** |

Table 3: Performance of Student Model Compared to Teacher Model

## 7 Evaluation & Conclusions

This project introduced a smart and efficient way to sharpen images using deep learning and knowledge distillation. The powerful teacher model, which was trained on high-quality image data, delivered strong restoration results with an SSIM of **0.7943** and a PSNR of **27.48** dB against the original images. We then trained a much smaller student model to learn from the teacher's output using a mix of L1 and MSE loss functions. Despite its lightweight design, the student model performed impressively, reaching an SSIM of **0.7119** and a PSNR of **25.78** dB compared to the ground truth, and an even higher SSIM of **0.9137** and PSNR **34.09** dB when compared to the teacher model's output. This successful balance of good results and efficient performance proves our project's main goal. It clearly highlights the benefit of transferring knowledge from larger AI models to smaller, practical ones. These result shows that the student model is not only fast and efficient, but also highly capable of restoring sharpness.

## 8 Acknowledgment

We would like to express our sincere thanks of gratitude to **Intel© Corporation** for granting the opportunity to work on this project the *Intel© - Unnati Program*. Their support, direction, and technical resources were very helpful for the successful execution of our work. We also would like to extend deep gratitude to our mentor **Mr. Siju Swamy**, whose constant feedback helped to us rectify our mistakes and improve our project, which eventually helped us to submit the project within the given time frame. We wish to recognize **Saintgits College of Engineering and Technology** for providing a nurturing academic environment with domain knowledge along with tools related to machine learning, which were foundational in setting up the project. To conclude, we thank all other academic and industry partners who collaborated with us on this project, helping us complete the project within the given time.

## References

[1] CHEN, G., CHOI, W., YU, X., HAN, T., AND CHANDRAKER, M. Learning efficient object detection models with knowledge distillation. In *Advances in Neural Information Processing Systems (NeurIPS)* (2017), pp. 742–751.

[2] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016.

[3] GOU, J., YU, B., MAYBANK, S. J., AND TAO, D. Knowledge distillation: A survey. *International Journal of Computer Vision 129*, 6 (2021), 1789–1819. 10.1007/s11263-021-01453-z.

[4] HINTON, G., VINYALS, O., AND DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[5] HUYNH-THU, Q., AND GHANBARI, M. The accuracy of psnr in predicting video quality for different video scenes and frame rates. *Telecommunication Systems 47*, 3–4 (2012), 247–262. 10.1007/s11235-010-9351-x.

[6] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324. 10.1109/5.726791.

[7] LIU, J., TANG, J., AND WU, G. Residual feature distillation network for lightweight image super-resolution. *arXiv preprint arXiv:2009.11551* (2020).

[8] PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., ET AL. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)* (2019), vol. 32.

[9] SAU, B. B., AND BALASUBRAMANIAN, V. N. Deep model compression: Distilling knowledge from noisy teachers. In *arXiv preprint arXiv:1610.09650* (2016).

[10] WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (Apr. 2004), 600–612. 10.1109/TIP.2004.824411.

## A Main code sections for the solution

### A.1 Loading and splitting the dataset

```
# Download and unzip the DIV2K dataset
url = "http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_HR.zip"
print(" Downloading DIV2K HR images...")
r = requests.get(url)
z = zipfile.ZipFile(io.BytesIO(r.content))
z.extractall("DIV2K_train_HR")
print(" Dataset downloaded and extracted!")
# Data Splitting
# Split the dataset for training and testing
DATASET_PATH = 'DIV2K_train_HR/DIV2K_train_HR'
all_files = sorted(glob.glob(os.path.join(DATASET_PATH, "*.png")))
# Use 200 images for training and 100 for testing
train_files = all_files[:700]
test_files = all_files[700:800]
print(f"Found {len(train_files)} training images.")
print(f"Found {len(test_files)} testing images.")
```

## A.2 Generating Blurry Dataset

```
class SharpeningDataset(Dataset):
    # Custom Dataset for creating blurry/sharp image pairs.
    def __init__(self, image_paths, patch_size=256):
        self.image_paths = image_paths
        self.patch_size = patch_size
        self.to_tensor = ToTensor()
        self.scale_factor = 2 # Defines the initial downscaling factor
    def __len__(self):
        return len(self.image_paths)
    def __getitem__(self, idx):
        try:
            # Load the high-resolution (HR) ground truth image
            hr_image = Image.open(self.image_paths[idx]).convert('RGB')
            w, h = hr_image.size
            if w < self.patch_size or h < self.patch_size:
                hr_image = hr_image.resize((self.patch_size, self.patch_size),
                                          Image.BICUBIC)
                w, h = hr_image.size
            # A random crop for data augmentation
            rand_w = np.random.randint(0, w - self.patch_size + 1)
            rand_h = np.random.randint(0, h - self.patch_size + 1)
            hr_patch = hr_image.crop((rand_w, rand_h, rand_w + self.patch_size,
                                      rand_h + self.patch_size))
            # Create the blurry (LR) input image
            # 1. Downscale the image
            lr_size = (self.patch_size // self.scale_factor, self.patch_size //
                                      self.scale_factor)
            lr_patch = hr_patch.resize(lr_size, Image.BICUBIC)
            # 2. Apply a strong Gaussian blur
            lr_patch = lr_patch.filter(ImageFilter.GaussianBlur(radius=1.5))
            # 3. Upscale it back to the original patch size
            lr_patch = lr_patch.resize((self.patch_size, self.patch_size), Image.
                                      BICUBIC)
            # Convert images to PyTorch tensors
            hr_tensor = self.to_tensor(hr_patch)
            lr_tensor = self.to_tensor(lr_patch)
            return {'lr': lr_tensor, 'hr': hr_tensor}
        except Exception as e:
            print(f"[Dataset Error] at index {idx}: {e}")
            raise e
patch_size = 256
batch_size = 5
train_dataset = SharpeningDataset(train_files, patch_size=patch_size)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
                                          num_workers=0)
test_dataset = SharpeningDataset(test_files, patch_size=patch_size)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False, num_workers=0)
                                           # Batch size 1 for testing
```

## A.3 Training Student Model

```
   def train_student_with_distillation(student_model, teacher_model, loader,
                                      epochs=300):
    print("\n Training Student Model with Knowledge Distillation ")
    # The student model will be trained
    student_model.to(device)
    optimizer = optim.Adam(student_model.parameters(), lr=1e-4)
    teacher_model.to(device)
    teacher_model.eval()
    # Loss functions
    loss_vs_truth = nn.L1Loss()
    loss_vs_teacher = nn.MSELoss()
    alpha = 0.3
    for epoch in range(epochs):
        student_model.train()
        epoch_loss = 0
        pbar = tqdm(loader, desc=f"Epoch {epoch+1}/{epochs}")
        for batch in pbar:
            lr_imgs, hr_imgs = batch['lr'].to(device), batch['hr'].to(device)
            optimizer.zero_grad()
            with torch.no_grad():
                teacher_output = teacher_model(lr_imgs)
            # Get the student's prediction
            student_output = student_model(lr_imgs)
            # Calculate the two loss components
            l1 = loss_vs_truth(student_output, hr_imgs)
            l2 = loss_vs_teacher(student_output, teacher_output)
            # Combine them into the final distillation loss
            distillation_loss = alpha * l1 + (1 - alpha) * l2
            distillation_loss.backward()
            optimizer.step()
            epoch_loss += distillation_loss.item()
            pbar.set_postfix({'Distill Loss': distillation_loss.item(), 'L1': l1.
                                                      item(), 'L2': l2.item()})
    # Save the final student model
    torch.save(student_model.state_dict(), "student_model.pth")
    print("\nStudent model training complete and saved to /content/drive/MyDrive/
                                              student_model.pth")
# --- Execute Student Training ---
teacher_model = TeacherModel()
teacher_model.load_state_dict(torch.load("teacher_model.pth"))
teacher_model.eval()
student_model = StudentModel()
train_student_with_distillation(student_model, teacher_model, train_loader)
```

## A.4 Evaluation 1 (Student,Teacher)

```
def evaluate_model(student_model, reference_model, loader):
    #Evaluates how the student model matches the teacher model.
    student_model.to(device)
    reference_model.to(device)
    student_model.eval()
    reference_model.eval()
    total_ssim = 0
    count = 0
    with torch.no_grad():
        for batch in tqdm(loader, desc="Evaluating"):
            lr_imgs = batch['lr'].to(device)
            student_output = student_model(lr_imgs).clamp(0.0, 1.0)
            reference_output = reference_model(lr_imgs).clamp(0.0, 1.0)
            # Convert to (H, W, C) format
            student_np = student_output.squeeze().cpu().numpy().transpose(1, 2, 0)
            reference_np = reference_output.squeeze().cpu().numpy().transpose(1, 2
                                                  , 0)
            current_ssim = ssim(student_np, reference_np, data_range=1.0,
                                                  channel_axis=2)
            total_ssim += current_ssim
            count += 1
    return total_ssim / count
# Evaluate both models
print("\n Evaluating Model Performance on Test Set ")
# Load saved models for evaluation
final_teacher = TeacherModel()
final_teacher.load_state_dict(torch.load("teacher_model.pth"))
final_student = StudentModel()
final_student.load_state_dict(torch.load("student_model.pth"))
# Calculate and print SSIM scores
student_ssim = evaluate_model(final_student, final_teacher, test_loader)
print(f"\n--- Final Results ---")
print(f"Student Model SSIM: {student_ssim:.4f}")
```

## A.5 Evaluation 2(Student,Teacher,Ground Truth)

```
def evaluate_model(model, loader):
    #Evaluates the model on the test set and returns the average SSIM.
    model.to(device)
    model.eval()
    total_ssim = 0
    count = 0
    with torch.no_grad():
        for batch in tqdm(loader, desc="Evaluating"):
            lr_imgs, hr_imgs = batch['lr'].to(device), batch['hr'].to(device)
            output = model(lr_imgs).clamp(0.0, 1.0) # Clamp output to valid range
                                                  [0,1]
            # SSIM expects images in (H, W, C) format
            output_np = output.squeeze().cpu().numpy().transpose(1, 2, 0)
            hr_np = hr_imgs.squeeze().cpu().numpy().transpose(1, 2, 0)
            # Calculate SSIM for the current image and add to total
            current_ssim = ssim(output_np, hr_np, data_range=1.0, channel_axis=2)
            total_ssim += current_ssim
            count += 1
    return total_ssim / count
print("\n--- Evaluating Model Performance on Test Set ---")
# Load saved models for evaluation
final_teacher = TeacherModel()
final_teacher.load_state_dict(torch.load("teacher_model.pth"))
final_student = StudentModel()
final_student.load_state_dict(torch.load("student_model.pth"))
# Calculate and print SSIM scores
teacher_ssim = evaluate_model(final_teacher, test_loader)
student_ssim = evaluate_model(final_student, test_loader)
print(f"\n--- Final Results ---")
print(f"Teacher Model SSIM: {teacher_ssim:.4f}")
print(f"Student Model SSIM: {student_ssim:.4f}")
if student_ssim > 0.90:
    print("\n SUCCESS: Student model achieved an SSIM score above 90%!")
else:
    print("\n NOTE:Student model SSIM is below 90%.Consider more training epochs
                                        or a slightly larger student model.")
```

## A.6 Visualisation of Data Results

```
from skimage.metrics import peak_signal_noise_ratio as psnr
def visualize_results_with_ssim_psnr(model, loader, num_images=25):
    model.to(device)
    model.eval()
    to_pil = ToPILImage()
    teacher = TeacherModel()
    teacher.load_state_dict(torch.load("teacher_model.pth"))
    teacher.to(device)
    teacher.eval()
    fig, axes = plt.subplots(num_images, 4, figsize=(20, 5 * num_images))
    if num_images == 1: axes = [axes]
    axes[0, 0].set_title("Blurry Input", fontsize=16)
    axes[0, 1].set_title("Student Output\n(SSIM | PSNR)", fontsize=16)
    axes[0, 2].set_title("Teacher Output\n(SSIM | PSNR)", fontsize=16)
    axes[0, 3].set_title("Ground Truth", fontsize=16)
    with torch.no_grad():
        for i, batch in enumerate(loader):
            if i >= num_images: break
            lr_img, hr_img = batch['lr'].to(device), batch['hr'].to(device)
            student_output = model(lr_img).clamp(0.0, 1.0)
            teacher_output = teacher(lr_img).clamp(0.0, 1.0)
            s_out_np = student_output.squeeze().cpu().numpy().transpose(1,2,0)
            t_out_np = teacher_output.squeeze().cpu().numpy().transpose(1,2,0)
            hr_np = hr_img.squeeze().cpu().numpy().transpose(1,2,0)
            # Compute SSIM & PSNR
            ssim_student = ssim(s_out_np, hr_np, data_range=1.0, channel_axis=2)
            ssim_teacher = ssim(t_out_np, hr_np, data_range=1.0, channel_axis=2)
            psnr_student = psnr(hr_np, s_out_np, data_range=1.0)
            psnr_teacher = psnr(hr_np, t_out_np, data_range=1.0)
            images = [
                to_pil(lr_img.squeeze().cpu()),
                to_pil(student_output.squeeze().cpu()),
                to_pil(teacher_output.squeeze().cpu()),
                to_pil(hr_img.squeeze().cpu())]
            for j, (ax, img) in enumerate(zip(axes[i], images)):
                ax.imshow(img)
                ax.axis('off')
                if j == 1:
                    ax.set_title(f"Student\nSSIM: {ssim_student:.4f} | PSNR: {
                                                          psnr_student:.2f}",
                                                          fontsize=12)
                elif j == 2:
                    ax.set_title(f"Teacher\nSSIM: {ssim_teacher:.4f} | PSNR: {
                                                          psnr_teacher:.2f}",
                                                          fontsize=12)
    plt.tight_layout()
    plt.show()
print("\n Visualizing Sample Results with per-image SSIM & PSNR ")
visualize_results_with_ssim_psnr(final_student, test_loader, num_images=5)
```