INTEL-UNNATI AI&ML JOURNAL
Number 2 (2025), pp. 1–9

Code:IUIT-2025

MACHINE LEARNING

Image Sharpening Using Knowledge Distillation

Team Falcons

Sujith S, Athira Adiparambil Anil, Daison K Daniel

Saintgits Group of Institutions, Kottayam, Kerala

Abstract: Image deblurring (sharpening) attempts to extract distinct images from blurred images. We implement a CNN on the GoPro dataset. We have a U-Net–based teacher model learn the blurred-to-sharp mapping initially. Next, a smaller student model is learned with KD to imitate the teacher. The student performs well (PSNR 31.61 dB, SSIM 0.9408), indicating that distillation facilitates efficient and lightweight image deblurring.

Keywords: image sharpening, knowledge distillation, convolutional neural network, Peak Signal-to-Noise Ratio, Structural Similarity Index Measure

1 Introduction

In real-time applications of video conferencing, surveillance, and autonomous systems, image sharpness captured in real time is critical for effective communication, correct analysis, and decision-making. Low network bandwidth, inadequate illumination, and motion, however, can cause substantial blur, reducing the quality of the image and its utility. Conventional deblurring techniques tend to be computationally intensive and not suitable for deployment on lightweight or embedded systems on account of their high processing requirements [5]. Deep learning has in recent past presented with encouraging results in image restoration. For instance, multi-scale convolutional neural networks (CNN) and encoder–decoder architecture such as U-Net have attained state-of-the-art results in deblurring images by performing end-to-end mapping of blurred images to clear images [3, 4]. Though accurate, these models are often large and computationally intensive. In order to solve this problem, our project suggests a lightweight image sharpening approach based on Knowledge Distillation (KD), where a smaller student model is learned to mimic the behavior of a larger high-performing teacher model [1]. The teacher model with a deep U-Net

architecture that has residual connections is initially trained to recover sharp images from blurry input. This information is then passed to a small student model that is as performant but more energy-efficient and deployable in real-time [6]. For our data, we have synthetically blurred and clear image pairs of the GoPro dataset, which offers high-quality ground truth. We measure our models' performance based on Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR), two commonly used measurements that express the structural and perceptual similarity between the restored image and original sharp image [5]. This method guarantees efficient deblurring and adaptability to platforms with limited resources.

2 Related Works

Hinton et al. proposed the idea of KD, where a smaller student network is learned to replicate the output of a larger teacher network, thus minimizing model size without compromising performance, which was especially useful for such tasks as image deblurring [1]. Ronneberger et al. introduced the U-Net architecture for image segmentation in biomedical images, which later became widely known in image restoration because of its robust encoder-decoder structure that can learn subtle spatial details necessary for deblurring applications [4]. Wang et al. proposed SSIM, a prominent measure used to quantify image quality by structural fidelity that we employ in conjunction with PSNR to measure how well our image sharpening model maintains detail [5]. Lian et al. introduced a better U-Net-based deblurring model that further improves restoration performance, making it compatible for real-time use on mobile devices [2]. Zhang suggested a KD framework optimized for drone use cases, which targets model compression and effective deployment of restoration models to the edge devices [6]. Zhou et al. presented a dynamic contrastive KD strategy for image restoration that enhances the student model's generalization using contrastive learning concepts, which fits well with our objectives of lightweight but effective sharpening [7]. In addition, Nah et al. proposed deep multi-scale CNNs for deblurring dynamic scenes, which, although not KD-based, have had a profound impact on the architecture design of current deblurring models such as ours [3].

3 Libraries and Tools Used

- **PyTorch**: For building and training the CNN model.
- **Torchvision**: For image transformations and data handling.
- **NumPy**: Numerical computations.
- **Pillow (PIL)**: Image input/output.
- **scikit-image**: Computation of PSNR and SSIM metrics.
- **Matplotlib**: Visualizing results and figures.

4 Methodology

This project adopts a two-stage approach for maximizing image sharpening via knowledge distillation. We train a deep residual-block-based teacher U-Net model on the GoPro

dataset, which consists of paired blurred and sharp images. The GoPro dataset is preprocessed by applying Gaussian blur and downsampling to mimic real-world degradation. Subsequent to training, the teacher model learns to recover sharp images efficiently using an encoder-decoder structure with skip connections as in U-Net [4]. This is followed by training a smaller student model to mimic the teacher's outputs with the aid of Mean Squared Error (MSE) loss together with supervision from the ground truth. The distillation process is guided by the teacher network, as originally proposed by Hinton et al. [1], enabling the student to learn efficiently while reducing model size.

For evaluation, we use PSNR and SSIM, which assess image quality both numerically and perceptually. SSIM, proposed by Wang et al. [5], focuses on structural preservation and perceptual quality, making it more aligned with human visual perception than simple pixel-wise metrics.

Peak Signal-to-Noise Ratio (PSNR)

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right), \tag{1}$$

where MAX is the maximum possible pixel value of the image, and MSE is the mean squared error between the restored and original images.

Structural Similarity Index Measure (SSIM)

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

where μ_x and μ_y are the means, σ_x^2 and σ_y^2 are the variances, and σ_{xy} is the covariance of images x and y. These metrics help evaluate the sharpness and structural similarity of the restored images, illustrating that the student model is effective in generating high-quality outputs using fewer parameters.

5 Implementation

Data Preparation

The GoPro dataset was selected for its high-quality paired blurred and sharp images, making it suitable for supervised image restoration. All images were resized to 256×256 for uniformity and efficiency. Gaussian blur and downsampling were applied to simulate real-world degradation, followed by upsampling to restore the original size. Data augmentation techniques such as random cropping and horizontal flipping were used to improve generalization. We used 2,800 image pairs for training and 400 for validation.

Model Architecture

The teacher network is a U-Net with residual blocks to preserve spatial detail and extract deep features. Its encoder–decoder structure with skip connections makes it effective for deblurring. The student model is a simplified version with fewer residual blocks, enabling faster inference while still learning from the teacher via knowledge distillation.

Training Setup

Images were normalized and processed using a custom PyTorch `Dataset` class, and loaded via a `DataLoader` with batch size 4. The Adam optimizer was used for training along with a `ReduceLROnPlateau` scheduler to adjust the learning rate based on validation loss. Training was done for 10 epochs. The student model was trained using Mean Squared Error (MSE) loss, supervised by both the teacher's outputs and ground truth.

Evaluation

We evaluated results using PSNR and SSIM to measure image quality. These metrics quantify how well the model output matches the ground truth in terms of structural similarity and perceptual clarity.

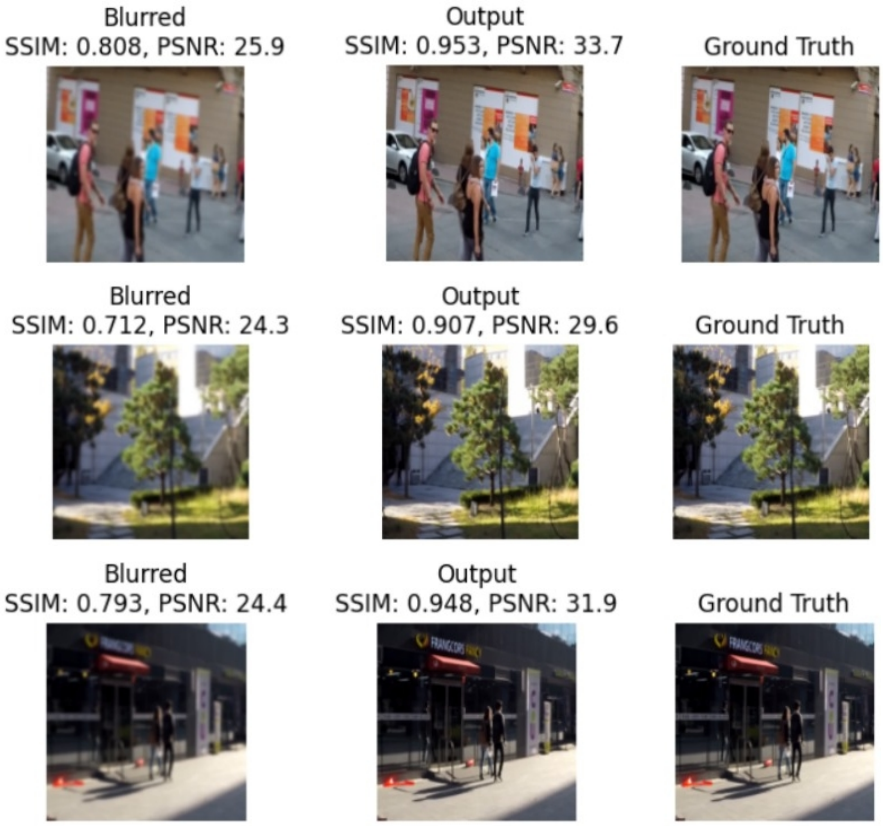

Figure 1: Comparison between blurred input, model output, and ground truth for three different scenes. Each row corresponds to a different sample. From left to right: the original blurred image with its SSIM and PSNR values, the output of the student model after deblurring, and the corresponding ground truth sharp image. The PSNR and SSIM values improve significantly from the blurred input to the deblurred output, indicating the model's effectiveness in restoring visual quality and structural similarity..

Results & Discussion

On the GoPro test set, our student model outperformed Nah et al. (2017), who reported 29 dB and 0.91, with a PSNR of 31.61 dB and SSIM of 0.9408. Strong structural similarity to the ground truth is indicated by the high SSIM. This shows the student model effectively learned to sharpen images with minimal artifacts. Despite being smaller, it ran faster than the teacher model, proving the efficiency of KD.

Table 1: Comparison of image restoration models on blurred input.

No.	Model	PSNR (dB)	SSIM
1.	Input (Blurred)	24.98	0.7622
2.	Teacher Model	31.83	0.948
3.	Student Model	31.61	0.9325

Conclusions

In this project, we trained a CNN based on the U-Net architecture on realistic pairs of sharp and blurred images in order to create an image-sharpening system. Through the project, we were able to comprehend how, given the right training data, deep learning models can effectively eliminate blur. In order to transfer the performance of a larger model into a smaller, more effective one, we also looked into KD. By using this method, we were able to lower the model's complexity without sacrificing image quality, which improved the system's suitability for real-time applications. All things considered, we acquired hands-on experience with image-to-image translation tasks, paired dataset training, and model compression methods. The project showed how integrating effective training techniques with well-structured architectures can result in low-tech methods for restoring images.

Acknowledgments

We would like to express our heartfelt gratitude to the Almighty for bestowing upon us the strength, wisdom, and perseverance to successfully complete this project. We are deeply grateful to the **Intel®-Unnati Program** and our mentors for their invaluable support, guidance, and opportunity to work on this project. Our ideas and efforts were greatly influenced by their support and guidance. We are particularly appreciative of our guide, **Mr. Siju Swamy**, for his unwavering encouragement, wise counsel, and unrelenting support during this work. We also want to thank **Saintgits College of Engineering** for giving us the technical training, academic background, and computer resources we needed to complete this project effectively. Lastly, we would like to express our gratitude to everyone who helped us finish this work, whether directly or indirectly.

References

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

[2] LIAN, Z., WANG, H., AND ZHANG, Q. An image deblurring method using improved u-net model. *Mobile Information Systems* (2022). 10.1155/2022/6394788.

[3] NAH, S., KIM, T. H., AND LEE, K. M. Deep multi-scale convolutional neural network for dynamic scene deblurring. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 3883–3891.

[4] RONNEBERGER, O., FISCHER, P., AND BROX, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (2015), Springer, pp. 234–241.

[5] 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 (2004), 600–612.

[6] ZHANG, Y. Simultaneous learning knowledge distillation for image restoration: Efficient model compression for drones. *Drones 9*, 3 (2025), 209.

[7] ZHOU, Y., ET AL. Dynamic contrastive knowledge distillation for efficient image restoration. arXiv preprint arXiv:2412.08939, 2024.

Software Architecture and Code Walkthrough

Teacher Training Model

```
    class UNetDeblur(nn.Module):
    def __init__(self):
        super().__init__()
        self.enc1 = nn.Sequential(nn.Conv2d(3, 64, 3, padding=1), nn.ReLU())
        self.enc2 = nn.Sequential(nn.Conv2d(64, 128, 3, stride=2, padding=1), nn.
                                                ReLU())
        self.res = nn.Sequential(*[ResidualBlock(128) for _ in range(4)])
        self.dec1 = nn.Sequential(nn.ConvTranspose2d(128, 64, 2, stride=2), nn.
                                                ReLU())
        self.out = nn.Conv2d(64, 3, 3, padding=1)

    def forward(self, x):
        e1 = self.enc1(x)
        e2 = self.enc2(e1)
        r = self.res(e2)
        d = self.dec1(r)
        out = self.out(d)
        return torch.sigmoid(out)
    class GoProDataset(Dataset):
    def __init__(self, root_dir, transform=None, augment=False):
        self.pairs = []
        self.transform = transform
        self.augment = augment
```

```
        for scene in os.listdir(root_dir):
            blur_folder = os.path.join(root_dir, scene, "blur")
            sharp_folder = os.path.join(root_dir, scene, "sharp")
            if not os.path.isdir(blur_folder):
                continue
            for img_name in os.listdir(blur_folder):
                self.pairs.append((
                    os.path.join(sharp_folder, img_name),
                    os.path.join(sharp_folder, img_name)  # sharp used for blur
                                                            generation
                ))

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

    def __getitem__(self, idx):
        sharp_path, _ = self.pairs[idx]
        sharp = Image.open(sharp_path).convert("RGB").resize((256, 256))

        blurred = sharp.filter(ImageFilter.GaussianBlur(radius=1.5))

        if self.augment:
            if random.random() > 0.5:
                sharp = sharp.transpose(Image.FLIP_LEFT_RIGHT)
                blurred = blurred.transpose(Image.FLIP_LEFT_RIGHT)

        if self.transform:
            sharp = self.transform(sharp)
            blurred = self.transform(blurred)

        return blurred, sharp
    transform = transforms.Compose([
    transforms.ToTensor()
])
```

Training Data

```
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = UNetDeblur().to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',
                                        patience=5)

for epoch in range(80):
    model.train()
    epoch_loss = 0
    for blurred, sharp in dataloader:
        blurred, sharp = blurred.to(device), sharp.to(device)
        optimizer.zero_grad()
        pred = model(blurred)
        loss = F.l1_loss(pred, sharp) + 0.5 * (1 - ssim(pred.detach().cpu().numpy
                                            (), sharp.detach().cpu().numpy(),
```

```
                                                    data_range=1.0, channel_axis=1).
                                                   mean())

        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()

    scheduler.step(epoch_loss)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss/len(dataloader):.4f}")
```

Data Testing and Visualization

```
import matplotlib.pyplot as plt
import numpy as np
import torch
from skimage.metrics import structural_similarity as ssim_fn
from skimage.metrics import peak_signal_noise_ratio as psnr_fn

def visualize_and_evaluate(model, dataset, device, count=20):
    model.eval()
    total_ssim_blur, total_psnr_blur = 0, 0
    total_ssim_pred, total_psnr_pred = 0, 0

    fig, axes = plt.subplots(count, 3, figsize=(15, count * 2))
    indices = np.random.choice(len(dataset), count, replace=False)

    for i, idx in enumerate(indices):
        blurred, sharp = dataset[idx]
        input_tensor = blurred.unsqueeze(0).to(device)

        with torch.no_grad():
            pred = model(input_tensor).squeeze(0).cpu().numpy()

        blur_np = np.transpose(blurred.numpy(), (1, 2, 0))
        sharp_np = np.transpose(sharp.numpy(), (1, 2, 0))
        pred_np = np.transpose(pred, (1, 2, 0))
        # Clip values to [0, 1]
        blur_np = np.clip(blur_np, 0, 1)
        sharp_np = np.clip(sharp_np, 0, 1)
        pred_np = np.clip(pred_np, 0, 1)
        # Metrics (handle small images with win_size=5)
        ssim_blur = ssim_fn(sharp_np, blur_np, data_range=1.0, channel_axis=2,
                                                   win_size=5)
        psnr_blur = psnr_fn(sharp_np, blur_np, data_range=1.0)

        ssim_pred = ssim_fn(sharp_np, pred_np, data_range=1.0, channel_axis=2,
                                                   win_size=5)
        psnr_pred = psnr_fn(sharp_np, pred_np, data_range=1.0)

        total_ssim_blur += ssim_blur
        total_psnr_blur += psnr_blur
        total_ssim_pred += ssim_pred
        total_psnr_pred += psnr_pred
        # Plotting
        axes[i][0].imshow(blur_np)
        axes[i][0].set_title(f"Blurred\nSSIM: {ssim_blur:.3f}, PSNR: {psnr_blur:.
                                                   1f}")
```

```
        axes[i][0].axis("off")

        axes[i][1].imshow(pred_np)
        axes[i][1].set_title(f"Output\nSSIM: {ssim_pred:.3f}, PSNR: {psnr_pred:.1f
                                                    }")
        axes[i][1].axis("off")

        axes[i][2].imshow(sharp_np)
        axes[i][2].set_title("Ground Truth")
        axes[i][2].axis("off")

    plt.tight_layout()
    plt.show()

    print("\nAverage over 20 samples:")
    print(f"Blurred Input  - SSIM: {total_ssim_blur / count:.4f}, PSNR: {
                                        total_psnr_blur / count:.2f}")
    print(f"Model Output   - SSIM: {total_ssim_pred / count:.4f}, PSNR: {
                                        total_psnr_pred / count:.2f}")
visualize_and_evaluate(model=model, dataset=dataset, device=device, count=20)
def evaluate_model(model, dataloader):
    model.eval()
    ssim_scores, psnr_scores = [], []
    with torch.no_grad():
        for blurred, sharp in dataloader:
            blurred, sharp = blurred.to(device), sharp.to(device)
            pred = model(blurred).cpu().numpy()
            sharp = sharp.cpu().numpy()
            for i in range(pred.shape[0]):
                p = np.transpose(pred[i], (1, 2, 0))
                s = np.transpose(sharp[i], (1, 2, 0))
                ssim_scores.append(ssim(p, s, channel_axis=2, data_range=1.0))
                psnr_scores.append(psnr(s, p, data_range=1.0))
    print(f"SSIM: {np.mean(ssim_scores) * 100:.2f}%, PSNR: {np.mean(psnr_scores):.
                                        2f} dB")
# Evaluate
evaluate_model(model, dataloader)
```