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

Code:IUIT-2025

MACHINE LEARNING

Image Sharpening Using Knowledge Distillation

Team Pixel

Arjun A, Ashil George James, Dona Elsa Jose

Saintgits Group of Institutions, Kottayam, Kerala

Abstract: Image blurring is a prevalent issue that impairs the efficacy and clarity of visual communication in real-time applications such as autonomous navigation, video conferencing, and surveillance. We suggest a knowledge distillation-based real-time image deblurring framework to solve this. The technique trains a smaller student model with three residual blocks using an 8-block teacher model. For training and testing, real-world scenarios are simulated using the artificially blurred BSD500 dataset. While preserving computational efficiency and satisfying real-time performance requirements, the student model produced notable gains in video quality as indicated by PSNR and SSIM scores.

Keywords: Image Deblurring, Knowledge Distillation, Residual Learning, U-Net, PSNR, SSIM, HIDE Model.

1 Introduction

In real-time applications such as video conferencing, surveillance the clarity of the image plays a vital role in ensuring effective communication and user experience. But factors like low network bandwidth, poor lighting conditions, and motion blur often degrade the quality of image. The deblurring methods that currently exist are not suitable to be run on modest devices because of the high computational expense [5]. Our project addresses this issue by proposing a lightweight image sharpening solution based on the Knowledge Distillation technique [2, 5]. In Knowledge Distillation there are two models, a teacher model and a student model. A teacher model is a powerful deep neural network which learns the complex task of image restoration. This knowledge is transferred to a smaller model that mimics the teacher model and requires a significantly lower computational resource [2]. The training of both the images are done with the help of synthetically blurred images and their performance is measured using Structural Similarity Index (SSIM) and Peak Signal

Noise Ratio (PSNR) [5]. The light weight student model offers high-quality image sharpening, which makes it suitable for real-time application.

2 Literature Review

The developments that had happened in the recent past in the filed of image enhancement and knowledge distillation have greatly influenced our work. Chen et al [1] using real-time spatial attention and dual-branch network introduced a low-light enhancement method. This approach enables fast processing which is highly suitable for mobile platforms. Li et al [8] in his work has suggested teacher student model for low-light enhancement which focuses on attention-map-based distillation for preserving structutal information.

Hinton et al. [2] laid the groundwork for knowledge distillation by tranining smaller networks for memorizing the outputs of larger ones, which allows them to function as stand alone classifiers while compressing without losing performance. Lozano [5] in his work has done a detailed survery of distillation methods such as response, feature and realtion-based techniques. This survery influenced our choice of response-based distillation for this work. Tao has explored knowledge distillation for YOLO and emphasized effective model compression for edge deployment which greatly impacted our strategy.

The Keras Team [8] by capturing moments of students during their work using a visual network, implemented practical vision knowledge distillation. They applied the knowledge to predict happiness. A CNN built from ResNet teacher informed the deblurring pipeline. Shaziya and Sharma [6] showcased U-Net's encoder-decoder architecture which we have used in our work for the traning of the model, its use in structural recovery is crucial for deblurring. Together, these studies support our approach: a lightweight, dual-branch U-Net followed by a discriminator that responds to knowledge distillation for real-time low-light deblurring on edge devices.

3 Libraries Used

In the project for various tasks, following packages are used.

```
    PyTorch: Deep learning model construction and training
    torchvision: Image transformation and data loading utilities
    Numpy: Numerical computation support
    Pillow: Image manipulation and preprocessing
    Matplotlib: Graphical visualization of results
    Scikit-learn: Image quality assessment tools
    tqdm: Visualization of training progress
```

4 Methodology

To implement and evaluate sharpening of image using Knowledge Distillation technique, the methodology is divided into distinct phases. The first step is Dataset Preparation. The BSD500 (Berkeley Segmentation Dataset) is used here for training and evaluation [6]. The High Resolution images were cropped into patches of 256x256 then the patches were downscaled by a factor of two by bilinear interpolation to patches of 128x128 , applied gaussian blur and then upscaled back to original resolution of 256x256 which simulates the blurry image seen in low bandwidth video conferencing [4]. For the teacher and the student models a U–Net–based architecture was used [6]. The Teacher Model is a complex U–Net model while the student is lighter version which mimic the teacher's output [2, 5]. The teacher model is trained using supervised learning with the loss between the predicted output and ground truth high resolution images having optimizer Adam with learning rate $1e^{-4}$. The student model is trained on a combination of the L1 loss between student output and the distillation loss between student and teacher output and ground truth. The final trained models are tested on unseen data and evaluated using the two key metrics SSIM and PSNR

5 Implementation

Dataset Preparation and Degradation

The DataSet we used for training and evaluation is BSD500 [6]. It was splitted into training and validation sets. Each image from the dataset underwent many processing steps. A crop of 256x256 was taken initially from each of the image, downscaled it using bicubic interpolation to 128x128, followed by adding a Gaussian Blur of radius 1. Finally, the image was upscaled to 256x256 using bicubic interpolation. The cropped image is the ground truth and the degraded image is the input to the models.

Model Architecture

Our Model Architecture is based on a lightweight variant of U–Net–Model called the HIDE Model [6]. It starts with an convolutional input layer followed by two downsampling layers. At the core we have implemented multiple residual blocks which helps to preserve the important features. The downsampling layer is followed by two upsampling layers which restores the resolution, and an output layer which produces the sharpened images. What helps the model to retain the fine details is the skip connections between the encoder and decoder stages [6]. The teacher model for higher accuracy has used 8 residual blocks while the student on the other hand has used 3 blocks for reducing the complexity.

Training Procedure

The teacher model was trained first using only the mean absolute error (L1 Loss) between the prediction of the teacher and the ground truth. The Adam optimizer was used with a learning rate of 1×10^{-4} [6] and a scheduler that reduced the learning rate every 25 epochs. The teacher was trained with a batch size of 8 for 75 epochs.

The student model was trained using the knowledge distillation technique. The student model learns from both the output of the teacher and the ground truth using a weighted L1 Loss. The distillation loss is defined by:

$$\text{Loss} = \alpha \cdot \mathcal{L}_1(\text{student}, \text{teacher}) + (1 - \alpha) \cdot \mathcal{L}_1(\text{student}, \text{ground\_truth})$$

with $\alpha = 0.75$, emphasizing the learning from the teacher [2, 5]

Evaluation Setup

The trained models were evaluated on a set of unseen images that were not used during the training phase. To measure image quality and structural similarity, SSIM and PSNR metrics were used [4, 6]. The outputs of the teacher model, student model, the degraded input, and the ground truth were displayed side by side for visual inspection. Each of the images has its SSIM and PSNR values labeled, which helps confirm that the student model is able to mimic the performance of the teacher model while being lighter and faster. The formulas used for SSIM and PSNR are given below [3, 9]:

Structural Similarity Index (SSIM):

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

Where:

- μ_x, μ_y are the means of images x and y
- σ_x^2, σ_y^2 are the variances of x and y
- σ_{xy} is the covariance of x and y
- $C_1 = (K_1L)^2, C_2 = (K_2L)^2$ are constants to stabilize the division
- L is the dynamic range of pixel values (typically 255)

Peak Signal-to-Noise Ratio (PSNR):

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

Where:

- $\text{MSE} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}[I(i, j) - K(i, j)]^2$
- MAX_I is the maximum possible pixel value (e.g., 255)
- I is the original image, and K is the reconstructed image

6 Results & Discussion

In this work, we have evaluated the performance of a lightweight student model trained via Knowledge Distillation from a deeper and complex teacher model for image restoration on blurred and downsampled images from the BSD500 dataset [6]. The results are summarized in Table 1.

Table 1: Comparison of image restoration models on blurred input

No.	Model	PSNR (dB)	SSIM
1.	Input (Blurred)	24.84	0.6536
2.	Teacher Model	28.82	0.8322
3.	Student Model	28.77	0.8310

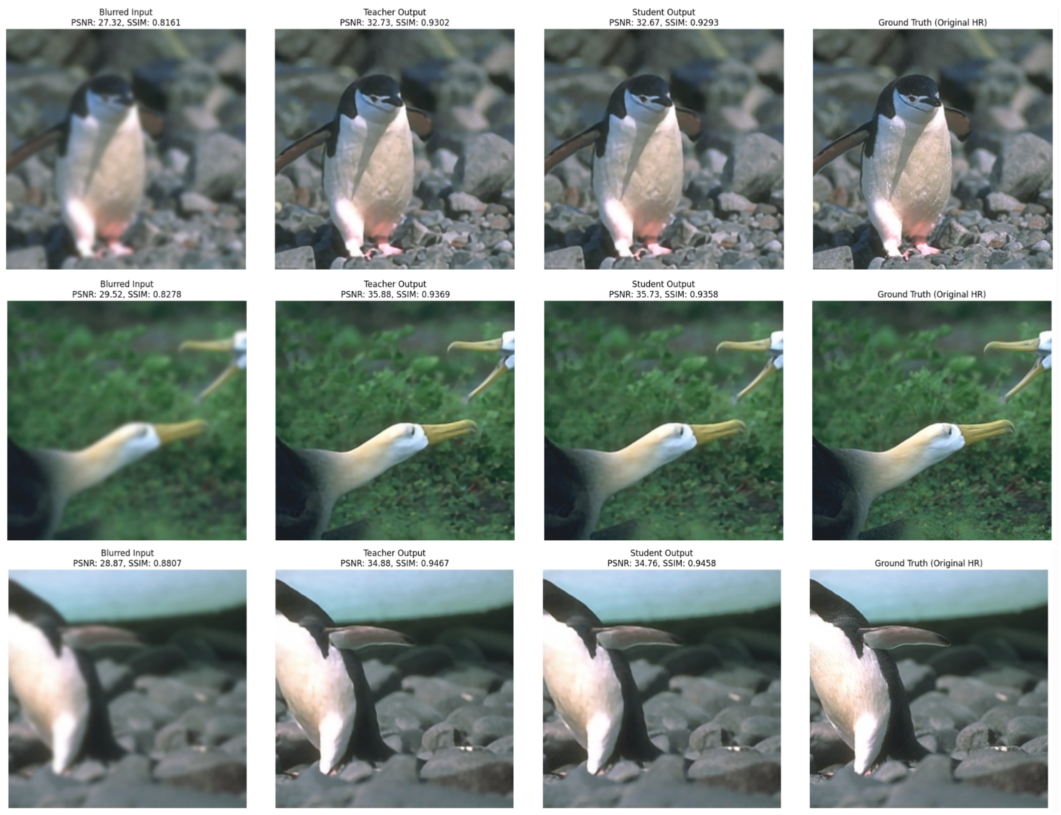
Blurred Input
PSNR: 27.32, SSIM: 0.8161
Teacher Output
PSNR: 32.73, SSIM: 0.9302
Student Output
PSNR: 32.67, SSIM: 0.9293
Ground Truth (Original HR)
Blurred Input
PSNR: 29.52, SSIM: 0.8278
Teacher Output
PSNR: 35.88, SSIM: 0.9369
Student Output
PSNR: 35.73, SSIM: 0.9358
Ground Truth (Original HR)
Blurred Input
PSNR: 28.87, SSIM: 0.8807
Teacher Output
PSNR: 34.88, SSIM: 0.9467
Student Output
PSNR: 34.76, SSIM: 0.9458
Ground Truth (Original HR)

Figure 1: Comparison of degraded input, student output, teacher output, and ground truth with SSIM and PSNR metrics.

7 Conclusions

In this project, we have successfully implemented and evaluated a knowledge distillation model for image restoration. A high capacity teacher model was trained first to map from blurred inputs to sharpened images as output. A more lightweight student model was then trained using combination of the ground truth and teacher's prediction [2,4]. The outputs show that the student model despite being lightweight and having fewer parameters was able to mimic the teacher model in terms of both ssim and psnr and offered higher computational efficiency. Upon side by side comparison of the input and the outputs there is noticeable difference between the Ground Truth and the Output of the student model. The student model was able to restore fine details and structural similarity effectively [4,6].

Overall, this project highlights the practical benefits of knowledge distillation which enables the design of compact and high performing models [2,5].

Acknowledgments

We extend our sincere thanks of gratitude to **Intel© Corporation** for allowing us to work on this project through the *Intel© - Unnati Program*. We are especially thankful to our mentor, Mr. Siju Swamy, for his valuable guidance and unwavering support. His time to time feedback and support was a key element which helped us to compelete this project within this limited time.

We would also like to thank Saintgits College of Engineering and Technology for providing crucial training and resources in machine learning. We are also grateful to the researchers and practitioners from which our work is inspired from. Finally, we would like to thank all those who contributed to our project either directly or indirectly.

References

[1] CHEN, Y., ZHOU, M., ZHANG, Y., LI, X., ZHAO, L., WANG, H., AND LIU, W. Real-time low-light image enhancement with spatial attention and dual-branch network. *Scientific Reports 13*, 1 (2023), 7263. 10.1038/s41598-023-34379-2.

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

[3] 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.

[4] LI, Z., WANG, Y., AND ZHANG, J. Low-light image enhancement with knowledge distillation. *Neurocomputing 518* (2023), 332–343. 10.1016/j.neucom.2022.10.083.

[5] LOZANO, P. Knowledge distillation: A survey. *arXiv preprint arXiv:2501.09268* (2024).

[6] SHAZIYA, H., AND SHARMA, M. Pulmonary ct images segmentation using cnn and unet models of deep learning. *International Journal of Advanced Trends in Computer Science and Engineering 10*, 1 (2021), 174–179.

[7] TAO, J. Knowledge distillation for yolo, 2022. Medium article.

[8] TEAM, K. Knowledge distillation for image classification. Keras documentation.

[9] 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 Downloading and extracting the dataset

```
# Download and Extract Dataset
TRAIN_PATH = 'BSR/BSDS500/data/images/train/'
TEST_PATH = 'BSR/BSDS500/data/images/test/'
if not os.path.exists('BSR'):
    print("--- Downloading BSD500 dataset ---")
    # These shell commands work in Colab/Jupyter
    !wget -q https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/
                                                BSR/BSR_bsds500.tgz
    !tar -xzf BSR_bsds500.tgz
    print("Dataset extracted successfully.")
else:
    print("Dataset already exists.")
```

A.2 Downscaling the Image

```
    def __getitem__(self, idx):
        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
        rand_w, rand_h = np.random.randint(0, w - self.patch_size + 1), 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))
        lr_w, lr_h = self.patch_size // self.scale_factor, self.patch_size // self
                                                 .scale_factor
        # Degrade: Downsample -> Blur -> Upsample
        degraded_patch = hr_patch.resize((lr_w, lr_h), Image.BICUBIC)
        if self.blur_radius > 0:
            degraded_patch = degraded_patch.filter(ImageFilter.GaussianBlur(radius
                                                     =self.blur_radius))
        degraded_patch = degraded_patch.resize((self.patch_size, self.patch_size),
                                                  Image.BICUBIC)
        hr_tensor, degraded_tensor = self.to_tensor(hr_patch), self.to_tensor(
                                                 degraded_patch)
        return {'lr': degraded_tensor, 'hr': hr_tensor}
```

A.3 Calculating SSIM and PSNR

```
def calculate_ssim(img1_tensor, img2_tensor):
    """Calculate SSIM between two tensors."""
    img1, img2 = to_numpy_image(img1_tensor), to_numpy_image(img2_tensor)
    # The 'multichannel' argument is deprecated; use 'channel_axis' instead
    win_size = min(min(img1.shape[0], img1.shape[1]), 7)
    if win_size < 3: return 0.0
    if win_size % 2 == 0: win_size -=1
    return structural_similarity(img1, img2, win_size=win_size, channel_axis=-1,
                                                data_range=1.0)
def calculate_psnr(img1_tensor, img2_tensor):
    """Calculate PSNR between two tensors."""
    mse = F.mse_loss(img1_tensor, img2_tensor)
    return 20 * torch.log10(1.0 / torch.sqrt(mse)).item() if mse > 0 else float('
                                              inf')
```

A.4 Teacher Model Training

```
#Teacher Model Training
print("\n--- Training Teacher Model (8 ResBlocks) ---")
teacher_model_path = "teacher_model_blurred.pth"
teacher_model = HIDEModel(n_feats=64, n_resblocks=8).to(device)
optimizer = optim.Adam(teacher_model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=25, gamma=0.5)
criterion = nn.L1Loss()
num_epochs_teacher = 75 # Increased epochs
# Create dataloaders
train_dataset = BSD500RestorationDataset(TRAIN_PATH, scale_factor=2, blur_radius=1
                                         )
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True, num_workers=2
                                         , pin_memory=True)
start_time = time.time()
for epoch in range(num_epochs_teacher):
    teacher_model.train()
    epoch_loss = 0
    progress_bar = tqdm(train_loader, desc=f"Teacher Epoch {epoch+1}/{
                                             num_epochs_teacher}", leave=False)
    for batch in progress_bar:
        degraded, sharp = batch['lr'].to(device), batch['hr'].to(device)
        optimizer.zero_grad()
        loss = criterion(teacher_model(degraded), sharp)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    scheduler.step()
    print(f"Teacher Epoch [{epoch+1}/{num_epochs_teacher}], Avg Loss: {epoch_loss/
                                             len(train_loader):.6f}, LR: {
                                             scheduler.get_last_lr()[0]:.1e}")
print(f"--- Teacher Training Finished in {(time.time() - start_time)/60:.2f}
                                         minutes ---")
torch.save(teacher_model.state_dict(), teacher_model_path)
print(f"Teacher model saved to {teacher_model_path}")
```

A.5 Student Model Training

```
#Student Model Training (Knowledge Distillation)
print("\n--- Training Student Model (3 ResBlocks) via Distillation ---")
teacher_model.eval() # Freeze the teacher model
student_model = HIDEModel(n_feats=64, n_resblocks=3).to(device)
student_model_path = "student_model_blurred.pth"
optimizer = optim.Adam(student_model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=25, gamma=0.5)
alpha = 0.75 # Distillation loss weight
num_epochs_student = 75
start_time = time.time()
for epoch in range(num_epochs_student):
    student_model.train()
    epoch_loss = 0
    progress_bar = tqdm(train_loader, desc=f"Student Epoch {epoch+1}/{
                                             num_epochs_student}", leave=False)
    for batch in progress_bar:
        degraded, sharp = batch['lr'].to(device), batch['hr'].to(device)
        with torch.no_grad():
            teacher_pred = teacher_model(degraded)

        student_pred = student_model(degraded)
        loss = alpha * criterion(student_pred, teacher_pred) + (1 - alpha) *
                                                 criterion(student_pred, sharp)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    scheduler.step()
    print(f"Student Epoch [{epoch+1}/{num_epochs_student}], Avg Loss: {epoch_loss/
                                             len(train_loader):.6f}, LR: {
                                             scheduler.get_last_lr()[0]:.1e}")
print(f"--- Distillation Training Finished in {(time.time() - start_time)/60:.2f}
                                         minutes ---")
torch.save(student_model.state_dict(), student_model_path)
print(f"Student model saved to {student_model_path}")
```

A.6 Visualization Of The Output

```
# --- Visualize the first N results ---
num_images_to_visualize = 15
print(f"\n--- Visualizing First {num_images_to_visualize} Test Images ---")
for i in range(min(num_images_to_visualize, len(results))):
    current_result = results[i]
    # Convert tensors to numpy images for plotting
    images_to_plot = {k: to_numpy_image(v) for k, v in current_result['images'].
                                             items()}
    # Create the plot
    fig, axes = plt.subplots(1, 4, figsize=(22, 6))
    # Define titles for this specific result
    titles = [
      f"Blurred Input\nPSNR: {current_result['input_psnr']:.2f}, SSIM: {
                                               current_result['input_ssim']:.4f}",
      f"Teacher Output\nPSNR: {current_result['teacher_psnr']:.2f}, SSIM: {
                                               current_result['teacher_ssim']:.4f}
                                               ",
      f"Student Output\nPSNR: {current_result['student_psnr']:.2f}, SSIM: {
                                               current_result['student_ssim']:.4f}
                                               ",
      "Ground Truth (Original HR)"
    ]
    # Set titles and images
    for ax, img_key, title in zip(axes, images_to_plot.keys(), titles):
        ax.imshow(images_to_plot[img_key])
        ax.set_title(title, fontsize=12)
        ax.axis('off')
    fig.suptitle(f"Image Restoration Comparison (Test Image {i+1})", fontsize=16)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
```