

Intel Unnati Industrial Training 2025

Image Sharpening Using Knowledge Distillation

Team:

Mentor: Mrs. Rama Mamidala – Asst.Professor [m_rama@mlritm.ac.in]

Pinnapureddy Jahnavi – 237Y1A05H7[237y1a05h7@mlritm.ac.in]

Puneeth -- 237Y1A0585[237y1a0585@mlritm.ac.in]

1. Introduction

Image sharpening enhances the contrast between neighboring pixels to emphasize edges and details, improving the perceptual quality of images. Traditional methods like Laplacian filters or unsharp masking can enhance edges but often amplify noise and generate undesirable artifacts.

Deep learning models, such as convolutional neural networks (CNNs), have shown superior performance by learning complex image enhancement mappings from data. However, these models tend to be large and resource-intensive, limiting their deployment on edge devices.

Knowledge distillation (KD), introduced by Hinton et al., is a technique where a large, complex model (teacher) transfers its knowledge to a smaller, simpler model (student). This process can compress models without significant loss of accuracy.

In this project, we apply knowledge distillation to the problem of image sharpening, where the teacher model performs high-quality sharpening, and the student learns to approximate the teacher's output with fewer parameters. This enables efficient and effective sharpening suitable for real-time applications.

2. Problem Statement

Image Sharpening using knowledge distillation

Prerequisites:

Concepts in Machine Learning

Programming Skills (Python)

Deep Learning / CNN - Train/Validate/Test with Data

3. Abstract

Image sharpening is a fundamental task in image processing aimed at enhancing edges and fine details to improve visual clarity. Traditional sharpening techniques often lead to noise amplification and artifacts. Recent advances in deep learning have introduced more effective approaches but require large models, which are

computationally expensive. Knowledge distillation offers a way to compress large, powerful "teacher" models into smaller "student" models with comparable performance. This project explores the use of knowledge distillation to develop a lightweight image sharpening model that achieves high-quality results with reduced computational cost. We implement a teacher-student framework, train the models on a dataset of blurred and sharp images, and demonstrate the effectiveness of the approach through quantitative metrics and visual results.

4. Objective:

To develop a lightweight image sharpening model using knowledge distillation that achieves high-quality results with reduced computational cost, enabling efficient deployment on resource-limited devices.

5. Hardware and Software Setup

Hardware

- **Processor:** Intel Core i7 or equivalent
- **GPU:** NVIDIA GTX 1060 or higher (for faster training)
- **RAM:** 16 GB or more
- **Storage:** SSD with sufficient space for datasets and models

Software

- **Operating System:** Windows 10 / Ubuntu 18.04 or later
- **Programming Language:** Python 3.7+
- **Deep Learning Framework:** PyTorch 1.8+
- **Libraries:**
 - torchvision
 - numpy
 - PIL (Pillow)
- **Development Environment:** Jupyter Notebook or VS Code

6. Source code:


Below is an example implementation using Google Collab:

```
!pip install torch torchvision scikit-image opencv-python matplotlib
```

```
[ ] from google.colab import files
import os

uploaded = files.upload()
os.makedirs("images", exist_ok=True)

for fname in uploaded:
    os.rename(fname, f"images/{fname}")
```

 Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving butterfly.jpg to butterfly.jpg

```
[ ] import torch
import torch.nn as nn

class StudentModel(nn.Module):
    def __init__(self):
        super(StudentModel, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 3, 3, padding=1),
        )

    def forward(self, x):
        return self.net(x)
```

```
[ ] from torchvision import transforms
from PIL import Image
from torch.utils.data import Dataset, DataLoader
import cv2

class SharpnessDataset(Dataset):
    def __init__(self, path="images"):
        self.files = os.listdir(path)
        self.path = path
        self.transform = transforms.ToTensor()

    def __getitem__(self, idx):
        img_path = os.path.join(self.path, self.files[idx])
        hr = cv2.imread(img_path)
        hr = cv2.resize(hr, (384, 384))
        lr = cv2.resize(hr, (128, 128), interpolation=cv2.INTER_LINEAR)
        upscaled = cv2.resize(lr, (384, 384), interpolation=cv2.INTER_LINEAR)

        return self.transform(Image.fromarray(upscaled)), self.transform(Image.fromarray(hr))

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

```
[ ] device = "cuda" if torch.cuda.is_available() else "cpu"
    student = StudentModel().to(device)

    optimizer = torch.optim.Adam(student.parameters(), lr=1e-3)
    criterion = nn.MSELoss()

    dataset = SharpnessDataset()
    loader = DataLoader(dataset, batch_size=1, shuffle=True)

    for epoch in range(5):
        student.train()
        running_loss = 0
        for input_img, target_img in loader:
            input_img, target_img = input_img.to(device), target_img.to(device)

            output = student(input_img)
            loss = criterion(output, target_img)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        print(f"Epoch {epoch+1} Loss: {running_loss/len(loader):.4f}")
```

```
⇒ Epoch 1 Loss: 0.2569
Epoch 2 Loss: 0.2099
Epoch 3 Loss: 0.1639
Epoch 4 Loss: 0.1189
Epoch 5 Loss: 0.0772
```

```
[ ] from skimage.metrics import structural_similarity as ssim
    import numpy as np

    def compute_ssim(img1, img2):
        img1 = img1.permute(1, 2, 0).cpu().numpy()
        img2 = img2.permute(1, 2, 0).cpu().numpy()
        return ssim(img1, img2, channel_axis=2, data_range=1)

    student.eval()
    total_ssim = 0

    for input_img, target_img in loader:
        input_img, target_img = input_img.to(device), target_img.to(device)
        with torch.no_grad():
            output = student(input_img)

            total_ssim += compute_ssim(output[0], target_img[0])

    print(f"Average SSIM:", total_ssim / len(loader))
```

```
⇒ Average SSIM: 0.63354695
```

```

import matplotlib.pyplot as plt

student.eval()

# Iterate through a few batches
for i, (input_img, target_img) in enumerate(loader):
    if i >= 2: # Display images for first 2 batches
        break

    input_img, target_img = input_img.to(device), target_img.to(device)
    with torch.no_grad():
        output = student(input_img)

    # Process each image in the batch
    for j in range(input_img.size(0)):
        input_np = input_img[j].permute(1, 2, 0).cpu().numpy()
        target_np = target_img[j].permute(1, 2, 0).cpu().numpy()
        output_np = output[j].permute(1, 2, 0).cpu().numpy()

        fig, axes = plt.subplots(1, 2, figsize=(15, 5))

        axes[0].imshow(input_np)
        axes[0].set_title("original Image")
        axes[0].axis('off')

        axes[1].imshow(target_np)
        axes[1].set_title("sharpened output")
        axes[1].axis('off')
    plt.show()

```

7. OUTPUT



original Image



sharpened output



[] Start coding or [generate](#) with AI.

8. Applications

- Real-Time Image Enhancement: Deployment on mobile or embedded devices where computational resources are limited.
- Video Processing: Sharpening frames in real-time streaming or video editing applications.
- Medical Imaging: Enhancing details in scans where fine edge preservation is critical.
- Photography & Surveillance: Improving image clarity in low-quality or compressed images.

9. Conclusion

This project demonstrates that knowledge distillation can effectively compress a deep image sharpening model into a lightweight student network while retaining most of its performance. The distilled model is suitable for resource-constrained environments without a significant trade-off in image quality. Future work includes exploring more advanced architectures and distillation methods, and extending the approach to other image restoration tasks.