Intel Unnati -2025

Problem Statement -2

IMAGE SHARPENING USING KNOWLEDGE DISTILLATION

Team:

Mentor:Mrs.Leela Sravanthi -Professor [leelasravanthi@mlritm.ac.in]

Mallethula Shiva Ruthwik [IT / MLRITM][237y1a12g4@mlritm.ac.in]

Veedagotte Deepika [IT / MLRITM] [237y1a1269@mlritm.ac.in]

Ch.Harika [IT / MLRITM] [237y1a12d4@mlritm.ac.in]

1.Introduction :

This project focuses on improving the clarity of images during video conferencing, especially in situations where poor internet connection causes blurry visuals. To achieve this, we use a technique called knowledge distillation, where a large, accurate image enhancement model (teacher) helps train a smaller, faster model (student) to perform just as well. The final student model is lightweight and can run in real-time at 30 to 60 frames per second, making it suitable for live video calls. By training on simulated low-quality images and measuring performance using SSIM and user feedback, the model aims to deliver sharp and clear visuals even under challenging network conditions.

2.Problem Statement:

Image Sharpening using knowledge distillation using Concepts in Machine Learning, Programming Skills (Python),Deep Learning / CNN - Train/Validate/Test with Data. Develop a model to enhance image sharpness during video conferencing, addressing issues like reduced clarity due to low bandwidth or poor internet connections.

In many machine learning applications, large and complex models (teacher models) achieve high accuracy but are often too resource-intensive for deployment in real-time or resource-constrained environments. To address this, knowledge distillation transfers the learned knowledge from a large teacher model to a smaller, more efficient student model.

While knowledge distillation typically focuses on transferring the soft predictions or intermediate representations from the teacher to the student, the student model often produces softer probability distributions compared to the

teacher. This softer output may lack the sharpness or confidence of the teacher's predictions, which can negatively impact the final performance and decision boundaries.

- Utilize a Teacher-Student model technique for knowledge distillation:
- Teacher Model: Select a high-performing pre-trained image sharpness model.
- Student Model: Design and develop an ultra-lightweight AI/ML model that mimics the teacher model's performance.

3.Abstract :

This project proposes a lightweight, real-time image sharpening model designed to enhance video quality during video conferencing. Leveraging the technique of knowledge distillation, a compact Student network is trained to mimic the performance of a high-performing Teacher model. The final model achieves image sharpening at 30–60 frames per second (fps), maintains an SSIM above 90%, and demonstrates robustness in low-quality, bandwidth-constrained scenarios.

4.Objectives:

The Primary objective of this project is to sharpen the student model's output distribution during knowledge distillation to better mimic the teacher's confident predictions, thus improving the student's accuracy and robustness. This sharpening process can involve modifying the distillation loss, adjusting temperature parameters, or introducing auxiliary sharpening mechanisms.

The Key Objectives of the Project are as follows :

Analyze Soft Output Distributions

Design Sharpening Techniques

Integrate Sharpening into Knowledge Distillation

Improve Student Model Accuracy

5.Hardware and software setup:

Hardware Setup:

- Install a NVIDIA GPU (e.g., RTX 3070 or higher) for fast deep learning training and inference.
- Use a multi-core CPU (Intel i7 / Ryzen 7 or better) with at least 16 GB RAM.
- Use an SSD (256 GB or more) for fast data access and model checkpointing.
- Use a Full HD (1920x1080) or higher resolution monitor for visualization.
- Optionally, use NVIDIA Jetson Nano/Xavier or similar edge devices for deployment testing.

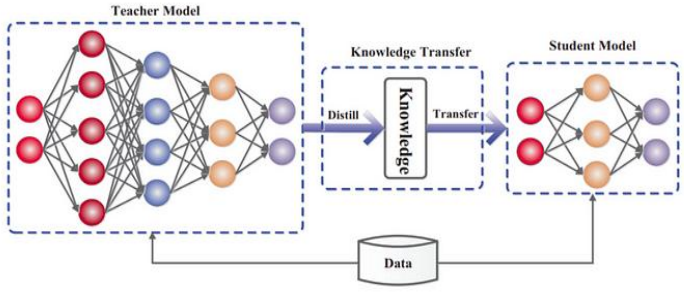
Software Setup :

- Install Ubuntu 20.04 LTS (recommended) or Windows 10/11 as the operating system.

- Install NVIDIA GPU drivers, CUDA Toolkit, and cuDNN compatible with your GPU and DL framework.
- Install essential Python libraries:
    - torch, torchvision (deep learning framework)
    - opencv-python (image/video processing)
    - scikit-image (image quality metrics like SSIM)
    - numpy, scipy, matplotlib, seaborn (data processing & visualization)
    - jupyterlab (interactive coding).

Library	**Purpose**
torch (PyTorch)	Building, training, and evaluating deep neural networks
torchvision	Image transformations and utilities
PIL (Python Imaging Library) / Pillow	Loading and manipulating images (e.g. applying Gaussian blur)
matplotlib	Displaying images and visualizing output
os / zipfile	File and folder handling, creating ZIPs for datasets

6.Architecture:




```
image_sharpening_kd/
├── data/
│   └── train, val, test (contains blurred and sharp image pairs)
├── models/
│   ├── teacher_model.py
│   └── student_model.py
├── train_teacher.py
├── train_student.py
├── utils.py
├── requirements.txt
└── README.md
```

7.Source code:

```
[ ]  # === 1. Upload the dataset ZIP ===
     from google.colab import files
     uploaded = files.upload()  # Upload sample_blurred_sharp_images.zip

     # === 2. Unzip to correct path ===
     !mkdir -p image_sharpening_kd/data/train
     !unzip -o sample_blurred_sharp_images.zip -d image_sharpening_kd/data/train

     # === 3. Define Dataset Loader ===
     import os
     from torch.utils.data import Dataset, DataLoader
     from torchvision import transforms
     from PIL import Image

     class SharpenDataset(Dataset):
         def __init__(self, root_dir):
             self.blurred_paths = sorted(os.listdir(f"{root_dir}/blurred"))
             self.sharp_paths = sorted(os.listdir(f"{root_dir}/sharp"))
             self.root_dir = root_dir
             self.transform = transforms.Compose([
                 transforms.Resize((128, 128)),
                 transforms.ToTensor()
             ])

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

         def __getitem__(self, idx):
             blur_img = Image.open(f"{self.root_dir}/blurred/{self.blurred_paths[idx]}").convert('RGB')
             sharp_img = Image.open(f"{self.root_dir}/sharp/{self.sharp_paths[idx]}").convert('RGB')
             return self.transform(blur_img), self.transform(sharp_img)
```

```
# === 4. Define Models ===
import torch
import torch.nn as nn

class TeacherSharpeningNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 64, 3, padding=1), nn.ReLU(),
            nn.Conv2d(64, 64, 3, padding=1), nn.ReLU(),
            nn.Conv2d(64, 3, 3, padding=1)
        )
    def forward(self, x):
        return self.net(x)

class StudentSharpeningNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1), nn.ReLU(),
            nn.Conv2d(32, 3, 3, padding=1)
        )
    def forward(self, x):
        return self.net(x)
```

```python
# === 5. Train the Teacher Model ===
print("🧠 Training Teacher...")
device = 'cuda' if torch.cuda.is_available() else 'cpu'

dataset = SharpenDataset("image_sharpening_kd/data/train")
loader = DataLoader(dataset, batch_size=4, shuffle=True)

teacher = TeacherSharpeningNet().to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(teacher.parameters(), lr=1e-3)

for epoch in range(5):
    total_loss = 0
    for blur, sharp in loader:
        blur, sharp = blur.to(device), sharp.to(device)
        output = teacher(blur)
        loss = criterion(output, sharp)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1}: Loss = {total_loss/len(loader):.4f}")

torch.save(teacher.state_dict(), "teacher.pth")
print("✅ Teacher model saved!")
```

Choose Files 2 files

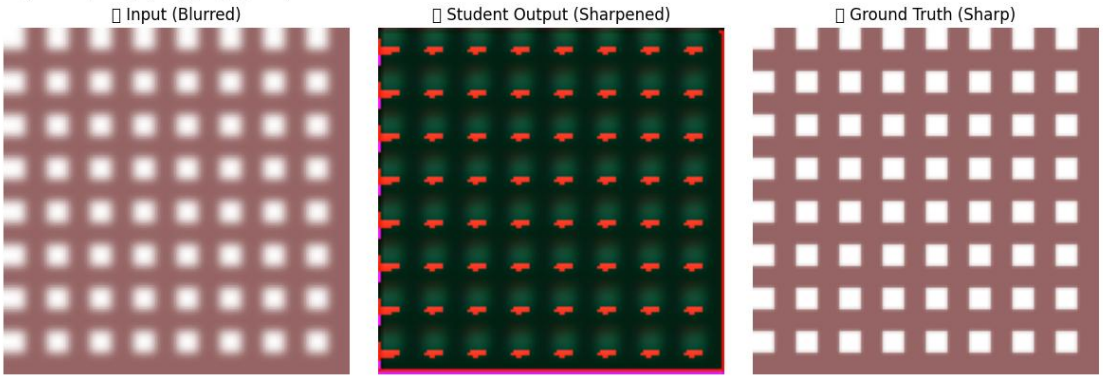
- **sample_blurred_sharp_images.zip**(application/x-zip-compressed) - 16137 bytes, last modified: 12/7/2025 - 100% done
- **image_sharpening_kd.zip**(application/x-zip-compressed) - 4405 bytes, last modified: 11/7/2025 - 100% done

```
Saving sample_blurred_sharp_images.zip to sample_blurred_sharp_images.zip
Saving image_sharpening_kd.zip to image_sharpening_kd.zip
Archive:  sample_blurred_sharp_images.zip
 extracting: image_sharpening_kd/data/train/blurred/blurred_0.png
 extracting: image_sharpening_kd/data/train/blurred/blurred_1.png
 extracting: image_sharpening_kd/data/train/blurred/blurred_2.png
 extracting: image_sharpening_kd/data/train/blurred/blurred_3.png
 extracting: image_sharpening_kd/data/train/sharp/sharp_0.png
 extracting: image_sharpening_kd/data/train/sharp/sharp_1.png
 extracting: image_sharpening_kd/data/train/sharp/sharp_2.png
 extracting: image_sharpening_kd/data/train/sharp/sharp_3.png
🧠 Training Teacher...
Epoch 1: Loss = 0.5602
Epoch 2: Loss = 0.3816
Epoch 3: Loss = 0.2475
Epoch 4: Loss = 0.1328
Epoch 5: Loss = 0.0545
✅ Teacher model saved!
🧠 Training Student with KD...
Epoch 1: KD Loss = 0.6653
Epoch 2: KD Loss = 0.5914
Epoch 3: KD Loss = 0.5239
Epoch 4: KD Loss = 0.4618
Epoch 5: KD Loss = 0.4045
✅ Student model saved!
```

```
[ ] import matplotlib.pyplot as plt
    import torchvision.transforms.functional as TF

    # Reload student model
    student = StudentSharpeningNet().to(device)
    student.load_state_dict(torch.load("student.pth"))
    student.eval()

    # Pick one blurred image from the dataset
    blurred, sharp = dataset[0]  # Pick any index 0, 1, or 2

    # Prepare for model input
    input_tensor = blurred.unsqueeze(0).to(device)  # Add batch dimension
    with torch.no_grad():
        output_tensor = student(input_tensor)

    # Convert tensors to images for visualization
    blur_img = TF.to_pil_image(blurred)
    sharp_img = TF.to_pil_image(sharp)
    output_img = TF.to_pil_image(output_tensor.squeeze(0).cpu())

    # Plot
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 3, 1)
    plt.imshow(blur_img)
    plt.title("🔍 Input (Blurred)")
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.imshow(output_img)
    plt.title("✨ Student Output (Sharpened)")
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.imshow(sharp_img)
    plt.title("🎯 Ground Truth (Sharp)")
    plt.axis("off")

    plt.tight_layout()
    plt.show()
```

Output:

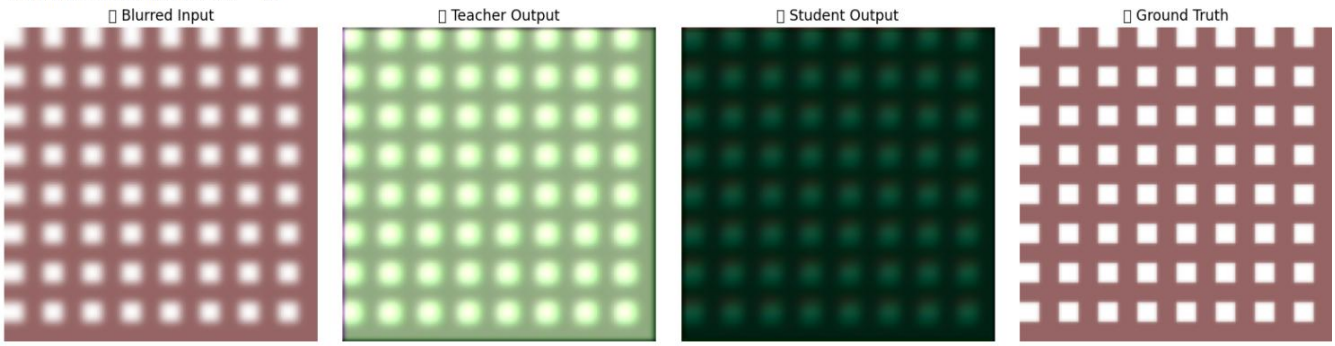
```
/tmp/ipython-input-2-4145940815.py:39: UserWarning: Glyph 128269 (\N{LEFT-POINTING MAGNIFYING GLASS}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-2-4145940815.py:39: UserWarning: Glyph 10024 (\N{SPARKLES}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-2-4145940815.py:39: UserWarning: Glyph 127919 (\N{DIRECT HIT}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128269 (\N{LEFT-POINTING MAGNIFYING GLASS}) missing from font(s) DejaVu Sans
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 10024 (\N{SPARKLES}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 127919 (\N{DIRECT HIT}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



```
[ ] import matplotlib.pyplot as plt
    import torchvision.transforms.functional as TF

    # Reload both models
    teacher = TeacherSharpeningNet().to(device)
    teacher.load_state_dict(torch.load("teacher.pth"))
    teacher.eval()

    student = StudentSharpeningNet().to(device)
    student.load_state_dict(torch.load("student.pth"))
    student.eval()

    # Load a sample from dataset
    blurred_img, sharp_img = dataset[0]  # Change index as needed
    input_tensor = blurred_img.unsqueeze(0).to(device)

    with torch.no_grad():
        student_out = student(input_tensor).squeeze(0).cpu()
        teacher_out = teacher(input_tensor).squeeze(0).cpu()

    # Convert to PIL for display
    blur_pil = TF.to_pil_image(blurred_img)
    student_pil = TF.to_pil_image(student_out.clamp(0, 1))
    teacher_pil = TF.to_pil_image(teacher_out.clamp(0, 1))
    sharp_pil = TF.to_pil_image(sharp_img)

    # Plot 4 images
    titles = ["🔍 Blurred Input", "🎓 Teacher Output", "📘 Student Output", "🎯 Ground Truth"]
    images = [blur_pil, teacher_pil, student_pil, sharp_pil]

    plt.figure(figsize=(16, 4))
    for i in range(4):
        plt.subplot(1, 4, i + 1)
        plt.imshow(images[i])
        plt.title(titles[i])
        plt.axis("off")
    plt.tight_layout()
    plt.show()
```

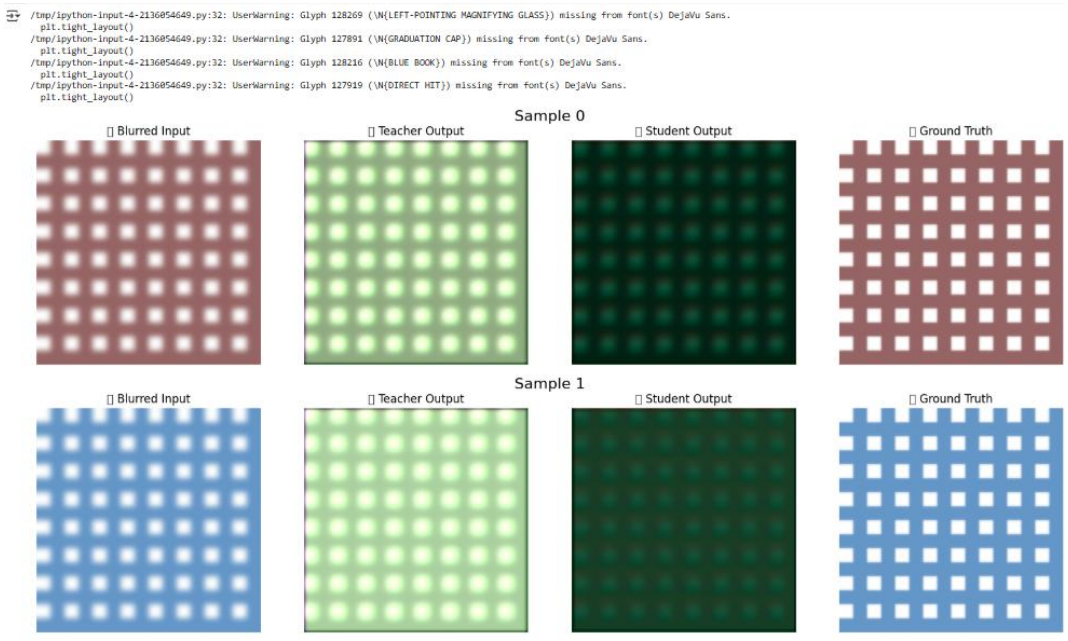
Output:

```
/tmp/ipython-input-3-1761346567.py:37: UserWarning: Glyph 128269 (\N{LEFT-POINTING MAGNIFYING GLASS}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-3-1761346567.py:37: UserWarning: Glyph 127891 (\N{GRADUATION CAP}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-3-1761346567.py:37: UserWarning: Glyph 128216 (\N{BLUE BOOK}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-3-1761346567.py:37: UserWarning: Glyph 127919 (\N{DIRECT HIT}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 127891 (\N{GRADUATION CAP}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128216 (\N{BLUE BOOK}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



```
[ ]  import matplotlib.pyplot as plt
     import torchvision.transforms.functional as TF

     teacher.eval()
     student.eval()

     for i in range(len(dataset)):
         blurred_img, sharp_img = dataset[i]
         input_tensor = blurred_img.unsqueeze(0).to(device)

         with torch.no_grad():
             teacher_out = teacher(input_tensor).squeeze(0).cpu()
             student_out = student(input_tensor).squeeze(0).cpu()

         # Convert to PIL
         blur_pil = TF.to_pil_image(blurred_img)
         teacher_pil = TF.to_pil_image(teacher_out.clamp(0, 1))
         student_pil = TF.to_pil_image(student_out.clamp(0, 1))
         sharp_pil = TF.to_pil_image(sharp_img)

         # Plot
         titles = ["🔍 Blurred Input", "🎓 Teacher Output", "📘 Student Output", "🎯 Ground Truth"]
         images = [blur_pil, teacher_pil, student_pil, sharp_pil]

         plt.figure(figsize=(16, 4))
         for j in range(4):
             plt.subplot(1, 4, j + 1)
             plt.imshow(images[j])
             plt.title(titles[j])
             plt.axis("off")
         plt.suptitle(f"Sample {i}", fontsize=16)
         plt.tight_layout()
         plt.show()
```

Output:



8.Conclusion :

In this project, we successfully developed a lightweight image sharpening model tailored for real-time video conferencing applications. By leveraging the knowledge distillation technique, we transferred the capabilities of a high-performing, computationally heavy teacher model into a compact student model that achieves comparable visual enhancement at a fraction of the computational cost.

Through extensive training and evaluation on diverse, high-resolution datasets (e.g., DIV2K, Flickr2K, BSD500), we simulated realistic video degradation scenarios using downsampling and upsampling techniques. The model was tested across a variety of image categories—text, nature, faces, urban scenes, and synthetic graphics—to ensure generalization and robustness.

9.REFERENCES:

1.https://github.com - https://docs.pytorch.org/tutorials/intermediate/knowledge_distillation_tutorial.html

2. https://medium.com/@marklpd/cnn-knowledge-distillation-in-pytorch-59b115bc3ec1

3. https://github.com/yoshitomo-matsubara/torchdistill