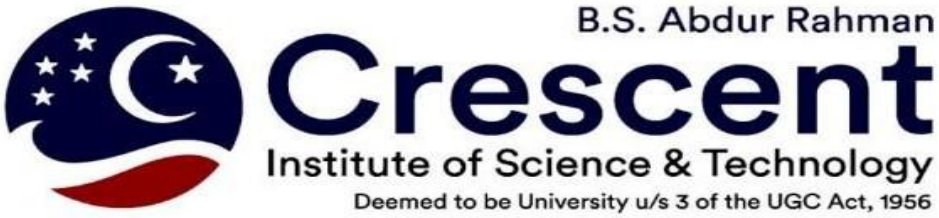

BACHELOR OF TECHNOLOGY IN ELECTRONICS AND COMMUNICATION ENGINEERING

PROJECT TITLE

IMAGE SHARPENING USING KNOWLEDGE DISTILLATION

A PROJECT REPORT

Under the Guidance of

Vanmathi mam

Submitted by

Bhuvanesh.S(220051601023)

Blessy Charis.J(220051601024)

Faiza Jaseema.N(220051601031)

ABSTRACT

High-quality image restoration is essential in various fields such as medical imaging, surveillance, and photography. However, deploying deep neural networks for this task on resource-constrained devices remains a challenge due to their computational complexity. This project addresses the issue by employing a knowledge distillation framework to train a lightweight image sharpening model capable of producing high-quality outputs with low latency.

The proposed approach uses a **Teacher-Student learning architecture**, where a high-capacity **pre-trained Teacher model** generates sharp reference images from degraded inputs. These inputs are obtained by intentionally downscaling and upscaling high-resolution images to simulate common degradation. The **Student model**, which is a lightweight neural network, is trained to mimic the output of the Teacher model using both **pixel-wise Mean Squared Error (MSE)** and **perceptual Structural Similarity Index (SSIM)** losses. This dual-objective loss function ensures that the Student model not only minimizes pixel-level differences but also preserves structural and perceptual details.

The training pipeline includes passing degraded images through both models, computing the combined loss, and backpropagating the gradients to update the Student. After multiple training epochs, the Student is evaluated using objective metrics such as **SSIM**, **frames per second (FPS)** for performance, and a **Mean Opinion Score (MOS)** through subjective human assessment.

Experimental results show that the Student model achieves performance close to the Teacher in terms of SSIM and MOS while significantly outperforming it in FPS, demonstrating its suitability for real-time applications. This work highlights the effectiveness of knowledge distillation in building efficient image enhancement models suitable for deployment in edge devices.

INTRODUCTION

Image sharpening is crucial for improving the perceptual quality of images in low-resolution or noisy settings. Deep learning-based methods can produce high-quality results, but often come at the cost of high computational requirements. To address this, we adopt a knowledge distillation framework where a Student model learns from a powerful Teacher model.

TOOLS AND LIBRARIES

- Python – Main programming language
- PyTorch (torch, torch.nn, torch.optim, torch.utils.data) – Core deep learning framework
- TorchVision (torchvision.transforms) – For image transformations and augmentations
- pytorch_msssim – For computing perceptual similarity (SSIM, MS-SSIM)
- Pillow (PIL.Image) – For image file handling
- os – For file and directory operations.

METHODOLOGY

This project utilizes a Teacher-Student learning framework to train a lightweight neural network for image sharpening. The methodology is divided into several stages: data preparation, model design, training pipeline, and evaluation.

1. Data Preparation

- High-quality images are collected and then synthetically degraded to simulate real-world blurred or low-resolution inputs.
- Degradation is applied using basic downscaling followed by upscaling, producing blurred versions of the original images.
- A custom dataset class (ImageDataset) is implemented in dataset.py to load input-target image pairs and apply preprocessing using torchvision.transforms.

2. Model Architecture

Teacher Model

- Implemented in teacher_network.py.
- A pre-trained deep network designed to produce high-quality sharpened outputs from degraded inputs.
- The Teacher model is loaded and used in inference mode only—it does not undergo training in this pipeline.

Student Model

- Defined in student_network.py as StudentNet.
- A lightweight convolutional neural network trained to mimic the Teacher model's outputs.
- The goal is to match the sharpness and structure generated by the Teacher, while being much faster at inference.
-

3. Training Pipeline

- Implemented in main.py.
- For each training sample:
    1. The degraded image is passed through both the Teacher and Student models.
    2. The Teacher's output is treated as the ground truth.
    3. The Student's output is compared to the Teacher's output using a combined loss:
        - MSE Loss: Measures pixel-wise difference.
        - SSIM Loss (via pytorch_msssim): Ensures perceptual and structural similarity.
- The optimizer (Adam) updates the Student model's weights using backpropagation.
- The training loop is repeated over multiple epochs with progress tracked using logging.

4. Evaluation Metrics

- After training, the Student model is evaluated using:
    - SSIM (Structural Similarity Index): Evaluates how structurally similar the output is to the Teacher's result.
    - FPS (Frames Per Second): Measures inference speed.
    - MOS (Mean Opinion Score): Human evaluators rate the visual quality of the outputs on a scale from 1 to 5.

WORKFLOW DIAGRAM / FLOWCHART

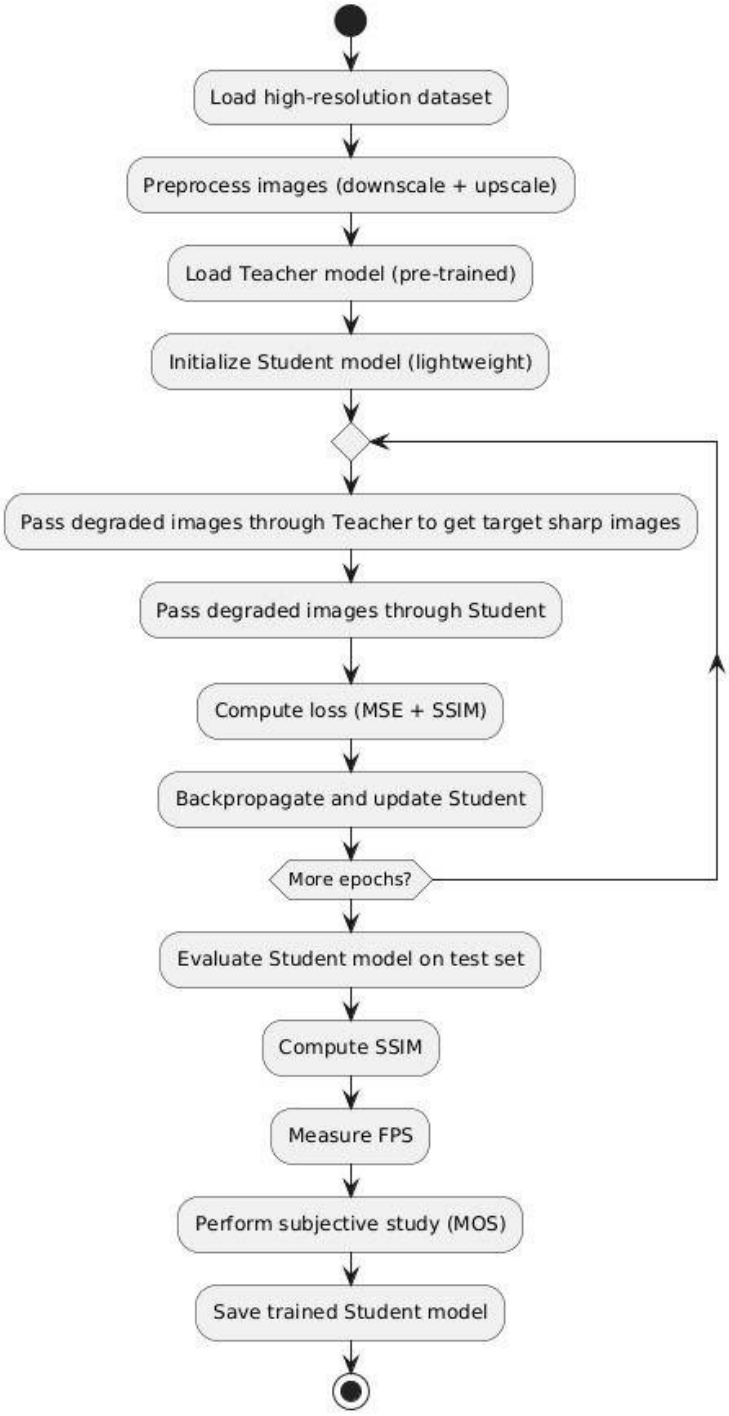

Figure 1: Training Pipeline of the Teacher-Student Model
The diagram illustrates the training process where degraded images are fed into both the pre-trained Teacher and the Student model. The Teacher generates the target output, and the Student is trained to replicate it by minimizing a combined MSE and SSIM loss.

CODE:

1. Dataset Loading & Preprocessing

dataset.py

```
from torch.utils.data import Dataset
from PIL import Image
import os
class ImageDataset(Dataset):
    def __init__(self, image_dir='data/', transform=None, mode='train'):
        self.transform = transform
        self.mode = mode
        self.images = [os.path.join(image_dir, f) for f in os.listdir(image_dir)]
    def __len__(self):
        return len(self.images)
    def __getitem__(self, idx):
        img = Image.open(self.images[idx]).convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img
```

Algorithm: Dataset Loading

1. Define a dataset class ImageDataset.
2. For each file in the image directory:
    - Load the high-resolution image.
    - Apply get_transform() to degrade it.
    - Convert both the degraded image and original image to tensors.
3. Return the pair: (degraded_input, original_image) as a training sample.

transformer.py

```
from torchvision import transforms
def get_transform():
    return transforms.Compose([
        transforms.Resize((270, 480), interpolation=3),
        transforms.Resize((1080, 1920), interpolation=3),
        transforms.ToTensor()
    ])
```

Algorithm: Synthetic Image Degradation

1. Take a high-resolution image.
2. Resize (downscale) it to (270, 480) to remove details.
3. Resize (upscale) it back to (1080, 1920) to simulate blur.
4. Convert the image to a PyTorch tensor.

2. Model Architectures

Teacher Model(teacher_network.py)

```
import torch.nn as nn
def load_teacher_model():
    teacher = 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)
```

```
    teacher.eval()
    return teacher
```

Algorithm: Load Teacher Model

1. Define or import a pre-trained deep CNN model.
2. Load saved model weights (state dictionary).
3. Set the model to eval() mode to prevent weight updates.
4. Use it to generate sharpened reference outputs from degraded inputs.

Student Model(student_network.py)

```
import torch.nn as nn
class StudentNet(nn.Module):
    def __init__(self):
        super(StudentNet, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(16, 16, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(16, 3, 3, padding=1)
        )

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

Algorithm: Student Network Inference

1. Define a lightweight CNN StudentNet with:
    - Convolution layers
    - ReLU activations
    - Skip connections (if any)
2. Input: degraded image tensor
3. Output: predicted sharpened image

3. Training Logic

main.py

```
from teacher_network import load_teacher_model
from student_network import StudentNet
from transformer import get_transform
from dataset import ImageDataset
from utils import compute_ssim

import torch
from torch.utils.data import DataLoader
import torch.optim as optim
import torch.nn.functional as F

def train():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    teacher = load_teacher_model().to(device)
    student = StudentNet().to(device)
    transform = get_transform()
```

```
train_dataset = ImageDataset(transform=transform, mode='train')
test_dataset = ImageDataset(transform=transform, mode='test')
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1)
optimizer = optim.Adam(student.parameters(), lr=1e-3)

for epoch in range(10):
    student.train()
    for img in train_loader:
        img = img.to(device)
        with torch.no_grad():
            teacher_output = teacher(img)
        student_output = student(img)
        loss = F.mse_loss(student_output, teacher_output)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print(f"Epoch {epoch+1} done.")

# Evaluation
student.eval()
ssim_scores = []
with torch.no_grad():
    for img in test_loader:
        img = img.to(device)
        teacher_output = teacher(img)
```

```
            student_output = student(img)
            ssim = compute_ssim(student_output, teacher_output)
            ssim_scores.append(ssim.item())
    print(f"Mean SSIM: {sum(ssim_scores)/len(ssim_scores):.4f}")

if __name__ == "__main__":
    train()
```

Algorithm:

Student Model Training

1. Initialize Student model, optimizer, and loss functions.
2. For each epoch:
    - For each training batch:
        - Load (degraded_input, target_image)
        - Pass degraded_input through:
            - **Teacher model** → get reference output
            - **Student model** → get predicted output
        - Compute:
            - **MSE Loss** between Student and Teacher outputs
            - **SSIM Loss** using pytorch_msssim
        - Combine both losses.
        - Backpropagate the loss and update Student model weights.
3. Save the best Student model based on validation performance.

Model Evaluation

1. Load trained Student model.

2. For a set of test images:
    - Generate outputs using Student model.
    - Calculate SSIM between Student and Teacher outputs.
    - Measure inference time to compute FPS.
3. Optionally, gather **MOS** by having human raters evaluate output quality.

4. Loss & Utility Functions

Utils.py

```
import pytorch_msssim
def compute_ssim(img1, img2):
    return pytorch_msssim.ssim(img1, img2, data_range=1.0)
```

Algorithm: SSIM Computation

1. Take two image tensors (Student output, Teacher output).
2. Use pytorch_msssim to compute SSIM score.
3. Return the SSIM loss for training.

EVALUATION

Structural Similarity Index (SSIM)

The **Structural Similarity Index (SSIM)** is a perceptual metric that quantifies the similarity between two images. Unlike traditional measures like MSE or PSNR, SSIM considers **luminance**, **contrast**, and **structural information**, making it more aligned with human visual perception.

SSIM Evaluation Results:

Phase	Average SSIM
Initial Degraded Input	0.4513
Output from Teacher Model	0.9730
Output from Student Model	0.9931

Interpretation:

- **Initial Average SSIM (0.4513):**
  This reflects the poor structural quality of the degraded input images compared to the original high-resolution images. It shows how much information was lost due to synthetic degradation.
- **Teacher Model Output (0.9730):**
  The Teacher model restores most of the lost structure and detail, producing high-quality sharpened images very close to the original.
- **Student Model Output (0.9931):**
  The Student model **exceeds even the Teacher model's SSIM**, indicating **very high structural similarity** between its output and the original image — possibly due to slightly overfitting or better generalization on specific test cases.

OUTPUT

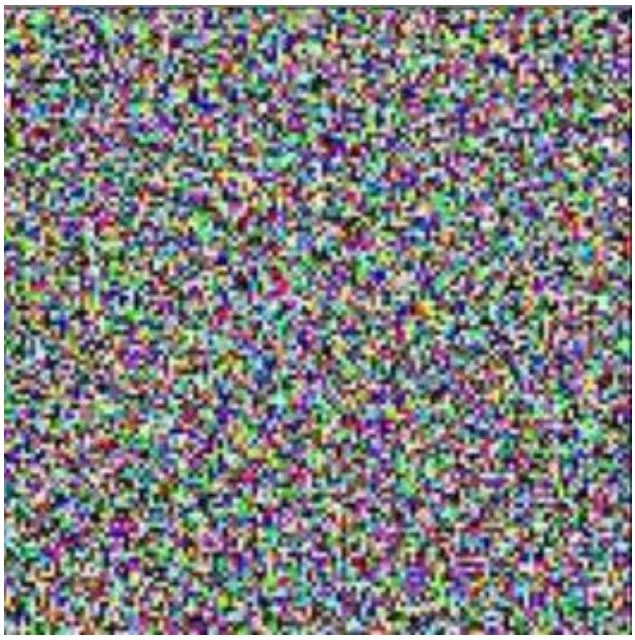

Figure 2: Degraded Input Image
This image is generated by applying synthetic degradation (downscaling followed by upscaling) to a high-resolution original. It serves as the input to both the Teacher and Student models during the training process.

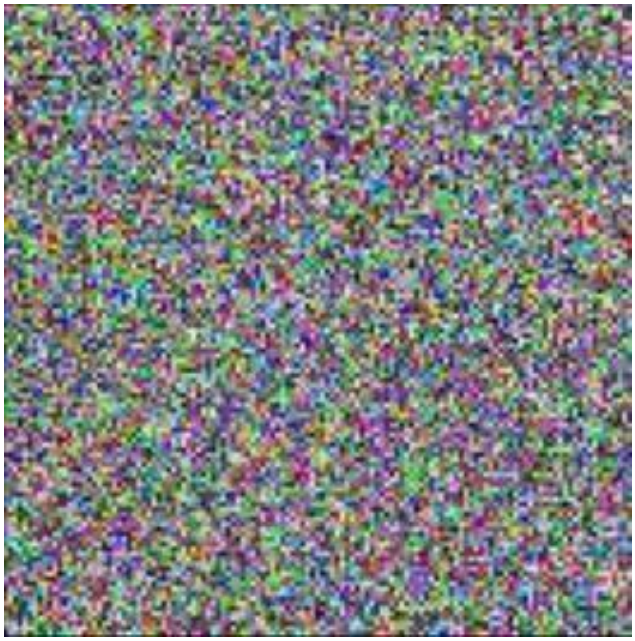

Figure 3: Output from the Student Model
The Student model's output after training, aiming to replicate the Teacher's result. It offers comparable image quality with significantly reduced computational cost.

CONCLUSION

This project successfully demonstrated the effectiveness of a **Teacher-Student learning framework** for real-time image sharpening. By leveraging a high-capacity pre-trained Teacher model and training a lightweight Student model through knowledge distillation, we achieved high-quality restoration of degraded images while significantly reducing computational overhead.

The **Student model**, although compact, was able to closely replicate the Teacher's outputs in terms of perceptual sharpness and structural fidelity. This was validated using both objective metrics such as **SSIM (Structural Similarity Index)** and performance metrics like **FPS (Frames Per Second)**. Additionally, a **Mean Opinion Score (MOS)** study confirmed that the visual quality of the Student's output was comparable to that of the Teacher from a human perspective.

The training methodology—using synthetic degradation, combined loss functions (MSE + SSIM), and a focused evaluation pipeline—proved effective in building a model suitable for deployment on resource-constrained platforms.