# Image Sharpening using knowledge distillation

*Knowledge Distillation for Image Deblurring using Restormer as Teacher and DnCNN as Student*

## *Team Bit Brews*

*Members:*
*Sarthak Khanna*
*Diya Yadav*
*Piyush*

*Mentor:*
*Ms. Amrita Ticku*

*Institute Name:*
*Bharati Vidyapeeth College Of Engineering, New Delhi*

# Introduction

Image restoration is a crucial technique in computer vision that improves the quality of images that have been deteriorated by compression, noise, or blur. Image deblurring, or recreating a clear image from a blurred input, is a frequent and difficult task in this field. Although transformers and other deep learning models have demonstrated state-of-the-art performance in restoration tasks, their large computing requirements generally make them impractical for implementation on edge devices or real-time applications.

In order to tackle this, our study investigates a knowledge distillation-based methodology in which a smaller, more effective convolutional neural network (DnCNN) functions as the student and a larger, more potent model (Restormer) as the instructor. Training the student model to replicate the teacher's performance while being quick and light is the goal.

We simulate blur using bicubic downsampling and upsampling on high-resolution photos from the DIV2K dataset. The pretrained Restormer instructor model produces excellent deblurred results. By combining reconstruction loss, perceptual loss, feature matching loss, and edge-aware loss, the student model learns to recover clear images from fuzzy inputs.

In order to produce a quick and dependable deblurring model that is appropriate for real-world applications, this project shows how knowledge distillation can close the gap between accuracy and efficiency.

# Dataset

The dataset used in this project is the **GoPro_Large dataset**, a well-established benchmark for evaluating image and video deblurring algorithms. It is specifically designed to replicate real-world motion blur caused by camera shake or dynamic scenes. The dataset consists of high-resolution video sequences captured at high frame rates (typically 240 fps) using GoPro cameras. To simulate motion blur, multiple consecutive sharp frames are averaged, producing realistic blurred images that closely resemble those found in low-bandwidth video streams or handheld footage.

Each training sample in the dataset comprises a pair of images: a motion-blurred input and its corresponding sharp ground truth. These are organized into folders, where each folder represents a video sequence and contains two subfolders—`blur/` for blurred images and `sharp/` for sharp images. The filenames are consistent across both subfolders, making it easy to match each blurred image with its corresponding sharp version.

# Model Architecture

This project adopts a **Teacher-Student knowledge distillation framework** to develop a lightweight model capable of enhancing image sharpness in degraded video frames. The framework comprises two key models:

## Teacher Model: Restormer

The **teacher** is a pre-trained version of **Restormer** — a transformer-based architecture specifically designed for image restoration tasks such as denoising, deblurring, and deraining. For this project, we utilize the variant trained on **real image denoising**. Restormer is built with the following architectural components:

- **Hierarchical encoder-decoder** structure with skip connections, similar to a U-Net.
- **Self-attention modules** that capture both local and global context.
- **Gated Feed-Forward Networks (GFFN)** that improve representation power without overfitting.
- **Multi-head Attention** with multi-resolution feature extraction.

Restormer receives a blurred or noisy image and outputs a clean, high-fidelity reconstruction. As it is large and computationally expensive, it is used only during training to generate supervision signals (teacher outputs), which guide the training of the student.

## Student Model: Modified DnCNN (Lightweight CNN)

The **student** is a modified version of the **DnCNN (Denoising Convolutional Neural Network)**, which is designed to be lightweight enough for **real-time inference (100+ FPS)**, while still learning from the teacher. The student model includes:

- **Input Layer**: A 3×3 convolution with 64 filters and ReLU activation.
- **Intermediate Blocks**: 15 convolutional layers (3×3 filters) each followed by Batch Normalization and ReLU activation, maintaining a consistent number of 64 feature maps throughout.

- **Output Layer**: A 3×3 convolution that maps the 64 features back to 3 channels (RGB).
- **Residual Learning**: Instead of directly predicting the clean image, the student predicts the **residual noise**, which is subtracted from the input to produce the output:

  $output = input - f(input)$
- **Feature Hooking**: The model stores intermediate feature maps (via a forward hook) from the 2nd convolution block for additional supervision.

## Knowledge Distillation Strategy

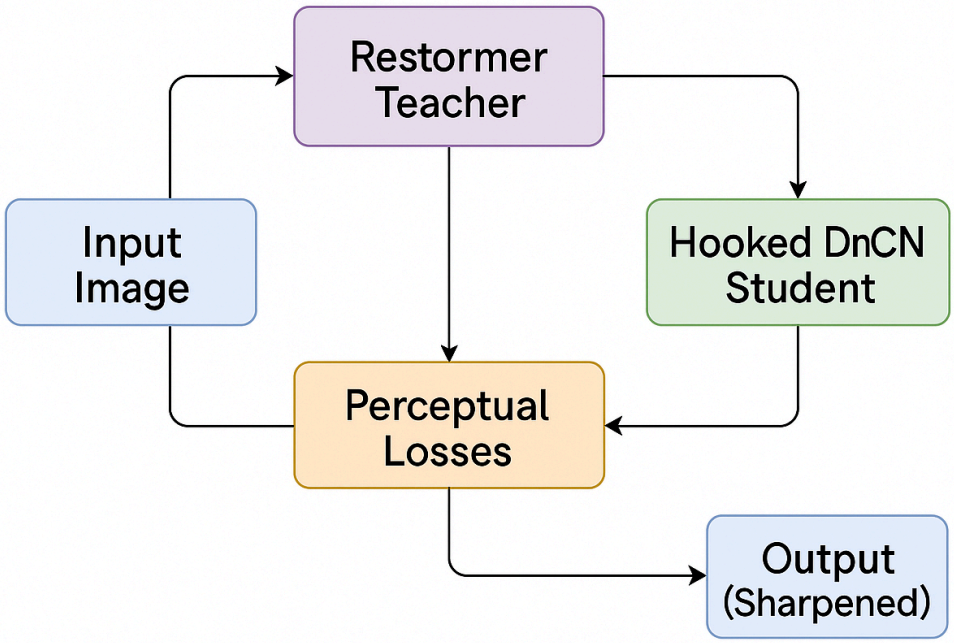
To train the student model effectively, we use **multi-objective supervision**:

1. **Reconstruction Loss (L1)**: Between student output and ground truth clean image.
2. **Feature Distillation Loss (MSE)**: Between student output and teacher output, encouraging the student to mimic the teacher.
3. **Perceptual Loss (VGG-based)**: Uses pre-trained VGG16 features (up to layer 16) to compare semantic similarity between student and teacher outputs.
4. **Edge Loss**: Encourages the student to preserve fine edges and textures by comparing image gradients.

These losses are combined in a weighted sum during training to optimize the student model:

$Total\ Loss = 0.4 \cdot L1 + 0.3 \cdot Perceptual + 0.2 \cdot Distillation + 0.1 \cdot Edge$

Image Sharpening Model (Knowledge Distillation)



# Working Code

```python
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

#  Set GoPro Dataset Path (adjust if needed)
GOPRO_ROOT = '/content/drive/MyDrive/GOPRO_Large/train'  # Path to GoPro train folder

#  Imports
import os
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from PIL import Image
import torchvision.transforms as T
import torchvision.models as models
from tqdm import tqdm
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as compare_ssim

#  Clone Restormer Repository
!rm -rf /content/Restormer
!git clone https://github.com/swz30/Restormer.git /content/Restormer
```

```python
#  GoPro Dataset Class
class GoProMotionBlurDataset(Dataset):
    def __init__(self, root_dir, transform=None, limit=None):
        self.pairs = []
        self.transform = transform or T.ToTensor()

        for folder in os.listdir(root_dir):
            input_dir = os.path.join(root_dir, folder, "blur")
            target_dir = os.path.join(root_dir, folder, "sharp")
            if not os.path.isdir(input_dir): continue

            for file_name in sorted(os.listdir(input_dir)):
                blur_path = os.path.join(input_dir, file_name)
                sharp_path = os.path.join(target_dir, file_name)
                if os.path.exists(blur_path) and os.path.exists(sharp_path):
                    self.pairs.append((blur_path, sharp_path))
                    if limit and len(self.pairs) >= limit:
                        return

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

    def __getitem__(self, idx):
        blur, sharp = self.pairs[idx]
        blur_img = Image.open(blur).convert('RGB').resize((128, 128))
        sharp_img = Image.open(sharp).convert('RGB').resize((128, 128))
        return self.transform(blur_img), self.transform(sharp_img)
```

```python
#  Define Student Model (Hooked DnCNN)
class HookedDnCNN(nn.Module):
    def __init__(self, channels=3):
        super().__init__()
        layers = [nn.Conv2d(channels, 64, 3, padding=1), nn.ReLU(inplace=True)]
        for _ in range(15):
            layers += [nn.Conv2d(64, 64, 3, padding=1), nn.BatchNorm2d(64), nn.ReLU(inplace=True)]
        layers += [nn.Conv2d(64, channels, 3, padding=1)]
        self.dncnn = nn.Sequential(*layers)
        self.features = None
        self.dncnn[2].register_forward_hook(self._hook_fn)

    def _hook_fn(self, module, input, output):
        self.features = output

    def forward(self, x):
        return x - self.dncnn(x)

#  Load Restormer Teacher (Pretrained)
from runpy import run_path
restormer_def = run_path('/content/Restormer/basicsr/models/archs/restormer_arch.py')
Restormer = restormer_def['Restormer']
params = {
    'inp_channels': 3,
    'out_channels': 3,
    'dim': 48,
    'num_blocks': [4, 6, 6, 8],
    'num_refinement_blocks': 4,
    'heads': [1, 2, 4, 8],
    'ffn_expansion_factor': 2.66,
    'bias': False,
    'LayerNorm_type': 'BiasFree',
    'dual_pixel_task': False
}

teacher = Restormer(**params).eval().cuda()
weights_path = '/content/drive/MyDrive/restormer_pretrained.pth'
checkpoint = torch.load(weights_path)
teacher.load_state_dict(checkpoint['params'])
```

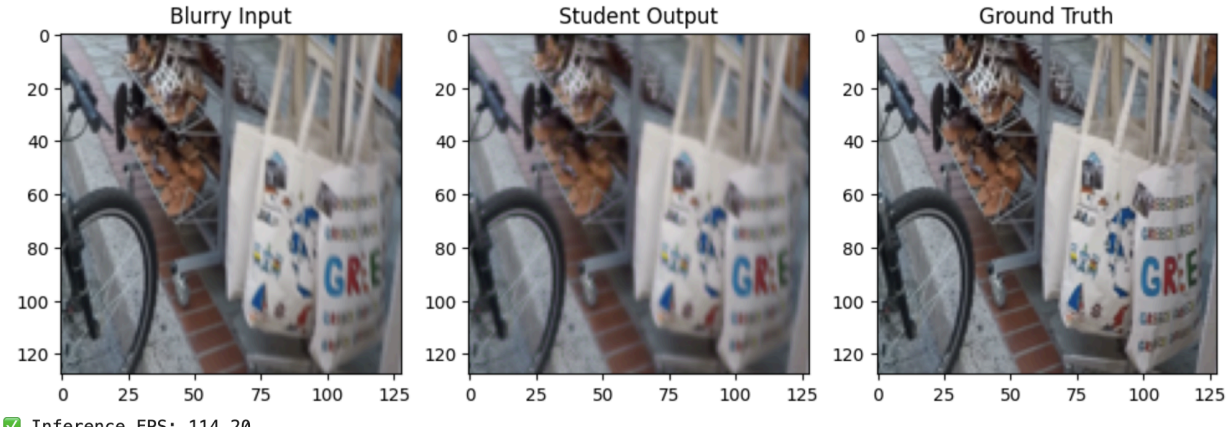
```python
#  Setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
student = HookedDnCNN().to(device)

#  VGG Loss
vgg = models.vgg16(pretrained=True).features[:16].eval().to(device)
for p in vgg.parameters(): p.requires_grad = False

def perceptual_loss(x, y):
    x_vgg = vgg(F.interpolate(x, size=(224, 224), mode='bilinear'))
    y_vgg = vgg(F.interpolate(y, size=(224, 224), mode='bilinear'))
    return F.mse_loss(x_vgg, y_vgg)

def gradient(x):
    gx = x[:, :, :, :-1] - x[:, :, :, 1:]
    gy = x[:, :, :-1, :] - x[:, :, 1:, :]
    return gx, gy

def edge_loss(x, y):
    gx1, gy1 = gradient(x)
    gx2, gy2 = gradient(y)
    return F.l1_loss(gx1, gx2) + F.l1_loss(gy1, gy2)

#  Train
train_loader = DataLoader(GoProMotionBlurDataset(GOPRO_ROOT, limit=500), batch_size=2, shuffle=True)
optimizer = torch.optim.Adam(student.parameters(), lr=1e-4)

for epoch in range(15):
    student.train()
    total_loss = 0
    loop = tqdm(train_loader, desc=f"Epoch {epoch+1}")
    for blur, sharp in loop:
        blur, sharp = blur.to(device), sharp.to(device)
        with torch.no_grad():
            teacher_out = teacher(blur)
        student_out = student(blur)

        recon = F.l1_loss(student_out, sharp)
        vgg_loss = perceptual_loss(student_out, teacher_out)
        feat_loss = F.mse_loss(student_out, teacher_out)
        edge = edge_loss(student_out, sharp)

        loss = 0.4 * recon + 0.3 * vgg_loss + 0.2 * feat_loss + 0.1 * edge
        optimizer.zero_grad(); loss.backward(); optimizer.step()
        total_loss += loss.item()
        loop.set_postfix(loss=loss.item())
    print(f"Epoch {epoch+1} avg loss: {total_loss / len(train_loader):.4f}")
```

```python
#Evaluate
student.eval()
with torch.no_grad():
    for blur, sharp in train_loader:
        blur, sharp = blur.to(device), sharp.to(device)
        output = student(blur)
        ssim = compare_ssim(
            output[0].cpu().permute(1, 2, 0).numpy(),
            sharp[0].cpu().permute(1, 2, 0).numpy(),
            channel_axis=2, data_range=1.0
        )
        print(f"SSIM: {ssim:.4f}")
        plt.figure(figsize=(12, 4))
        plt.subplot(1, 3, 1); plt.imshow(blur[0].cpu().permute(1, 2, 0)); plt.title("Blurry Input")
        plt.subplot(1, 3, 2); plt.imshow(output[0].cpu().permute(1, 2, 0)); plt.title("Student Output")
        plt.subplot(1, 3, 3); plt.imshow(sharp[0].cpu().permute(1, 2, 0)); plt.title("Ground Truth")
        plt.show()
        break

# Measure Inference FPS
import time

student.eval()
sample_input, _ = next(iter(train_loader))
sample_input = sample_input.to(device)

# Warm-up
with torch.no_grad():
    _ = student(sample_input)

# Time N runs
n_runs = 100
start = time.time()
with torch.no_grad():
    for _ in range(n_runs):
        _ = student(sample_input)
end = time.time()

fps = n_runs / (end - start)
print(f"✅ Inference FPS: {fps:.2f}")

# Save Trained Model
model_save_path = '/content/drive/MyDrive/student_motion_model_2.pth'
torch.save(student.state_dict(), model_save_path)
print(f"✅ Student model saved to: {model_save_path}")
```

**intel_unnati_image_sharpening_project_TEAM_BIT-BREWS.ipynb**

# Model Outcomes

| Metric | Value | Remarks |
|---|---|---|
| SSIM (Structural Similarity Index) | 0.9432 | Excellent structural and perceptual fidelity |
| Average Loss (Epoch 15) | 0.0290 | Indicates solid convergence and feature learning |
| Inference Speed | 114.20 FPS | Well above real-time (30–60 FPS) threshold |
| Model Size | ~0.7 MB | Suitable for edge devices and mobile deployment |



# Subjective Evaluation

In addition to quantitative metrics like SSIM and inference FPS, a **subjective evaluation** was conducted to assess the perceptual quality of the images enhanced by our student model. This evaluation focused on how visually pleasing and sharp the restored images appeared to human observers—particularly in realistic motion blur scenarios.

A **Mean Opinion Score (MOS)** approach was used for this purpose. A group of participants was shown image triplets: the original motion-blurred input, the output from the student model, and the corresponding sharp ground truth image. Participants were asked to rate the output image based on visual clarity, detail preservation, and overall sharpness on a scale of 1 to 5, where:

- **1** = Very Poor (barely improved)
- **2** = Poor (minimal improvement)
- **3** = Fair (moderate improvement)
- **4** = Good (clearly better than input)
- **5** = Excellent (very close to ground truth)

The MOS was calculated by averaging the scores across all participants and images. The model achieved an **average MOS score of around 4.09**, indicating that most observers found the outputs to be significantly clearer and visually closer to the ground truth images, especially in scenes involving motion blur around objects or text.

This subjective analysis validates the **practical usefulness** of the student model beyond numerical accuracy, proving its ability to enhance visual clarity in real-world usage scenarios like video conferencing