PROJECT PRESENTATION

Image Sharpening using Knowledge Distillation

Training: Intel Unnati Industrial Training 2025

IMAGE SHARPENING USING KNOWLEDGE DISTILLATION

Pixelated Image Detection & Correction

Presented by:Poojala Renuka (BU22eece0100491)
Meruga Sanjana (BU22eece0100431)
Md.Sameena Thasleem (BU22eece0100497)

PROJECT OBJECTIVE

- To develop a model that enhances image sharpness for video conferencing.
- Target use case: low bandwidth or unstable network environments.

DATA SOURCES

- Dataset: DBlur – Helen Subset
- Contains paired blurred and sharp facial images for supervised learning.
- Structure: Train, Validation, and Test sets
- Folder format: blur/ & sharp/ pairs
- Preprocessing: Images resized to 256×256.
- Blur Types: Includes motion and defocus blur simulating video call issues.
- Source: Kaggle - Image Deblurring Datasets
- https://www.kaggle.com/datasets/jishnuparayilshibu/a-curated-list-of-image-deblurring-datasets

MODEL ARCHITECTURE

- Teacher Model: Pre-trained high-performance image sharpness model.
- Student Model: Lightweight model trained using Knowledge Distillation to replicate teacher performance.
- Framework: PyTorch / TensorFlow

WORKING CODE

```
# Setup
import os
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import numpy as np
from skimage.metrics import structural_similarity as
compare_ssim
from skimage.metrics import peak_signal_noise_ratio as
compare_psnr
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(" Using device:", device)
# Dataset Class
class HelenDataset(Dataset):
  def __init__(self, blur_dir, sharp_dir, image_size=(256, 256)):
    self.blur_dir = blur_dir
    self.sharp_dir = sharp_dir
    self.image_size = image_size
    self.transform = transforms.Compose([
      transforms.Resize(image_size),
      transforms.ToTensor()
    ])
    self.filenames = sorted([
      f for f in os.listdir(blur_dir)
      if f.lower().endswith(('.jpg', '.jpeg', '.png')) and
os.path.exists(os.path.join(sharp_dir, f))
    ])
```

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

 def __getitem__(self, idx):
 fname = self.filenames[idx]
 blur = Image.open(os.path.join(self.blur_dir, fname)).convert('RGB')
 sharp = Image.open(os.path.join(self.sharp_dir, fname)).convert('RGB')
 return self.transform(blur), self.transform(sharp)

# Load Datasets
train_dataset = HelenDataset(
  blur_dir=r"E:\DBlur\Helen\train\blur",
  sharp_dir=r"E:\DBlur\Helen\train\sharp",
  image_size=(256, 256)
)
```

```
valid_dataset = HelenDataset(
 blur_dir=r"E:\DBlur\Helen\validation\blur",
 sharp_dir=r"E:\DBlur\Helen\validation\sharp",
 image_size=(256, 256)
)

test_dataset = HelenDataset(
 blur_dir=r"E:\DBlur\Helen\test\blur",
 sharp_dir=r"E:\DBlur\Helen\test\sharp",
 image_size=(256, 256)
)
from torch.utils.data import Subset
train_dataset = Subset(train_dataset, range(1300))


train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=4)
test_loader = DataLoader(test_dataset, batch_size=4)
```

```
 print(f" Loaded: Train={len(train_dataset)}, Valid={len(valid_dataset)}, Test=
{len(test_dataset)}")
# Model Definitions
class DnCNN(nn.Module):
 def __init__(self, channels=3, depth=17, filters=64):
 super().__init__()
 layers = [nn.Conv2d(channels, filters, 3, padding=1), nn.ReLU(inplace=True)]
 for _ in range(depth - 2):
 layers += [nn.Conv2d(filters, filters, 3, padding=1), nn.BatchNorm2d(filters),
nn.ReLU(inplace=True)]
 layers.append(nn.Conv2d(filters, channels, 3, padding=1))
 self.model = nn.Sequential(*layers)

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

```
class BetterStudent(nn.Module):
 def __init__(self):
 super().__init__()
 self.model = nn.Sequential(
 nn.Conv2d(3, 32, 3, padding=1), nn.ReLU(inplace=True),
 nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(inplace=True),
 nn.Conv2d(64, 64, 3, padding=1), nn.ReLU(inplace=True),
 nn.Conv2d(64, 32, 3, padding=1), nn.ReLU(inplace=True),
 nn.Conv2d(32, 3, 3, padding=1)
 )
def forward(self, x):
 return self.model(x)
```

```
# Metrics
def calculate_ssim(img1, img2):
 img1 = img1.detach().cpu().numpy().transpose(1, 2, 0)
 img2 = img2.detach().cpu().numpy().transpose(1, 2, 0)
 return compare_ssim(img1, img2, data_range=1.0, channel_axis=-1)

def calculate_psnr(img1, img2):
 img1 = img1.detach().cpu().numpy().transpose(1, 2, 0)
 img2 = img2.detach().cpu().numpy().transpose(1, 2, 0)
 return compare_psnr(img1, img2, data_range=1.0)

def evaluate(model, dataloader, name="Model"):
 model.eval()
 total_ssim, total_psnr, count = 0, 0, 0
 with torch.no_grad():
```

```
 for blurred, sharp in dataloader:
 blurred, sharp = blurred.to(device), sharp.to(device)
 output = model(blurred)
for i in range(output.size(0)):
 total_ssim += calculate_ssim(output[i], sharp[i])
 total_psnr += calculate_psnr(output[i], sharp[i])
 count += 1
 print(f" {name} SSIM: {total_ssim / count:.4f}, PSNR: {total_psnr / count:.2f}
dB")
# Training Loops
def train_teacher(model, dataloader, device, epochs=5):
  optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
  model.train()
  print(" Training started...")
  for epoch in range(epochs):
    total_loss = 0
 for i, (blurred, sharp) in enumerate(dataloader):
 if i == 0:
 print("First batch loaded")
 blurred, sharp = blurred.to(device), sharp.to(device)
 output = model(blurred)
 loss = F.mse_loss(output, sharp)

optimizer.zero_grad()
 loss.backward()
 optimizer.step()
 total_loss += loss.item()
 print(f"Teacher Epoch {epoch+1}, Loss: {total_loss / len(dataloader):.4f}")

def distill_studdef distill_student(student, teacher, dataloader, device,
epochs=10):
```

```
optimizer = torch.optim.Adam(student.parameters(), lr=1e-4)
student.train()
teacher.eval()
for epoch in range(epochs):
total_loss = 0
for blurred, sharp in dataloader:
blurred, sharp = blurred.to(device), sharp.to(device)
with torch.no_grad():
teacher_out = teacher(blurred)
student_out = student(blurred)
loss = F.mse_loss(student_out, sharp) + 0.5 * F.mse_loss(student_out, teacher_out)
```

```
 optimizer.zero_grad()
 loss.backward()
 optimizer.step()
 total_loss += loss.item()
 print(f" Student Epoch {epoch+1}, Loss: {total_loss / len(dataloader):.4f}")

def distill_loss(student_out, teacher_out, target):
  mse = F.mse_loss(student_out, target)
  distill = F.mse_loss(student_out, teacher_out)
  ssim_loss = 1 - pytorch_msssim.ssim(student_out, target, data_range=1.0)
  return mse + 0.5 * distill + 0.3 * ssim_loss

if __name__ == "__main__":
  teacher = DnCNN().to(device)
  print(" Training Teacher...")
 train_teacher(teacher, train_loader, device, epochs=10)
 evaluate(teacher, valid_loader, name="Teacher VALID")

 student = BetterStudent().to(device)
print(" Training Student with Distillation...")
 distill_student(student, teacher, train_loader, device, epochs=10)
 evaluate(student, valid_loader, name="Student VALID")

 print(" Final Evaluation on TEST set:")
 evaluate(student, test_loader, name="Student TEST")

# Save models
 torch.save(teacher.state_dict(), "teacher_model.pth")
 torch.save(student.state_dict(), "student_model.pth")
 print(" Models saved")
```

Output:

```
Using device: cpu
Loaded: Train=1300, Valid=165, Test=165
Training Teacher...
Training started...
First batch loaded
Teacher Epoch 1, Loss: 0.0035
First batch loaded
Teacher Epoch 2, Loss: 0.0012
First batch loaded
Teacher Epoch 3, Loss: 0.0011
```

```
First batch loaded
Teacher Epoch 4, Loss: 0.0011
First batch loaded
Teacher Epoch 5, Loss: 0.0010
First batch loaded
Teacher Epoch 6, Loss: 0.0010
First batch loaded
Teacher Epoch 7, Loss: 0.0010
First batch loaded
Teacher Epoch 8, Loss: 0.0009
First batch loaded
Teacher Epoch 9, Loss: 0.0009
First batch loaded
Teacher Epoch 10, Loss: 0.0009
Teacher VALID SSIM: 0.9367, PSNR: 34.20 dB
Training Student with Distillation...
Student Epoch 1, Loss: 0.0367
Student Epoch 2, Loss: 0.0021
Student Epoch 3, Loss: 0.0016
Student Epoch 4, Loss: 0.0014
Student Epoch 5, Loss: 0.0013
Student Epoch 6, Loss: 0.0013
Student Epoch 7, Loss: 0.0014
Student Epoch 8, Loss: 0.0012
Student Epoch 9, Loss: 0.0012
Student Epoch 10, Loss: 0.0012
Student VALID SSIM: 0.9192, PSNR: 32.12 dB
Final Evaluation on TEST set:
Student TEST SSIM: 0.9207, PSNR: 31.43 dB
Models saved
```

FOR VIZUALIZATION

```
import os
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import torchvision.transforms.functional as TF
import time
import csv
```

```
from skimage.metrics import structural_similarity as compare_ssim
from skimage.metrics import peak_signal_noise_ratio as compare_psnr

# Device Setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(" Using device:", device)

# Dataset Class
class HelenDataset(Dataset):
  def __init__(self, blur_dir, sharp_dir, image_size=(256, 256)):
    self.blur_dir = blur_dir
    self.sharp_dir = sharp_dir
    self.image_size = image_size
    self.transform = transforms.Compose([
      transforms.Resize(image_size),
      transforms.ToTensor()
    ])
```

```
self.filenames = sorted([
    f for f in os.listdir(blur_dir)
    if f.lower().endswith(('.jpg', '.jpeg', '.png')) and os.path.exists(os.path.join(sharp_dir, f))
  ])

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

 def __getitem__(self, idx):
  fname = self.filenames[idx]
  blur = Image.open(os.path.join(self.blur_dir, fname)).convert('RGB')
  sharp = Image.open(os.path.join(self.sharp_dir, fname)).convert('RGB')
  return self.transform(blur), self.transform(sharp), fname
```

```
# Load Paths
blur_path = r"E:\DBlur\Helen\test\blur"
sharp_path = r"E:\DBlur\Helen\test\sharp"

test_dataset = HelenDataset(blur_path, sharp_path, image_size=(256, 256))
test_loader  = DataLoader(test_dataset, batch_size=4)

print(f" Loaded: Test={len(test_dataset)}")

# Model Definitions
class DnCNN(nn.Module):
  def __init__(self, channels=3, depth=17, filters=64):
    super().__init__()
    layers = [nn.Conv2d(channels, filters, 3, padding=1), nn.ReLU(inplace=True)]
    for _ in range(depth - 2):
      layers += [nn.Conv2d(filters, filters, 3, padding=1), nn.BatchNorm2d(filters),
nn.ReLU(inplace=True)]
    layers.append(nn.Conv2d(filters, channels, 3, padding=1))
    self.model = nn.Sequential(*layers)
```

```
# Load Paths
blur_path = r"E:\DBlur\Helen\test\blur"
sharp_path = r"E:\DBlur\Helen\test\sharp"

test_dataset = HelenDataset(blur_path, sharp_path, image_size=(256, 256))
test_loader  = DataLoader(test_dataset, batch_size=4)

print(f" Loaded: Test={len(test_dataset)}")

# Model Definitions
class DnCNN(nn.Module):
  def __init__(self, channels=3, depth=17, filters=64):
    super().__init__()
    layers = [nn.Conv2d(channels, filters, 3, padding=1), nn.ReLU(inplace=True)]
    for _ in range(depth - 2):
      layers += [nn.Conv2d(filters, filters, 3, padding=1), nn.BatchNorm2d(filters),
nn.ReLU(inplace=True)]
    layers.append(nn.Conv2d(filters, channels, 3, padding=1))
    self.model = nn.Sequential(*layers)
  def forward(self, x):
    return x - self.model(x)


class BetterStudent(nn.Module):
  def __init__(self):
    super().__init__()
    self.model = nn.Sequential(
      nn.Conv2d(3, 32, 3, padding=1), nn.ReLU(inplace=True),
      nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(inplace=True),
      nn.Conv2d(64, 64, 3, padding=1), nn.ReLU(inplace=True),
      nn.Conv2d(64, 32, 3, padding=1), nn.ReLU(inplace=True),
      nn.Conv2d(32, 3, 3, padding=1)
    )
  def forward(self, x): return self.model(x)
```

```
# Load pretrained weights
teacher = DnCNN().to(device)
student = BetterStudent().to(device)

teacher.load_state_dict(torch.load("teacher_model.pth", map_location=device))
student.load_state_dict(torch.load("student_model.pth", map_location=device))

teacher.eval()
student.eval()

# Evaluation Metrics
def calculate_ssim(img1, img2):
  img1 = img1.detach().cpu().numpy().transpose(1,2,0)
  img2 = img2.detach().cpu().numpy().transpose(1,2,0)
  return compare_ssim(img1, img2, data_range=1.0, channel_axis=-1)
```

```
def calculate_psnr(img1, img2):
  img1 = img1.detach().cpu().numpy().transpose(1,2,0)
  img2 = img2.detach().cpu().numpy().transpose(1,2,0)
  return compare_psnr(img1, img2, data_range=1.0)

def evaluate(model, dataloader, name="Model"):
  model.eval()
  total_ssim, total_psnr, count = 0, 0, 0
  with torch.no_grad():
    for blurred, sharp, _ in dataloader:
      blurred, sharp = blurred.to(device), sharp.to(device)
      output = model(blurred)
      for i in range(output.size(0)):
        total_ssim += calculate_ssim(output[i], sharp[i])
        total_psnr += calculate_psnr(output[i], sharp[i])
        count += 1
  print(f" {name} SSIM: {total_ssim / count:.4f}, PSNR: {total_psnr / count:.2f} dB")
```

```
# SSIM Comparison Report
def compare_blur_to_outputs(student, teacher, dataloader, device, save_csv=False):
  total_bt = total_bs = total_bg = total_sg = 0
  count = 0
  rows = [["Image#", "Blur-Teacher SSIM", "Blur-Student SSIM", "Blur-GT SSIM", "Student-G
SSIM"]]

  with torch.no_grad():
    for batch_idx, (blurred, sharp, fnames) in enumerate(dataloader):
      blurred, sharp = blurred.to(device), sharp.to(device)
      t_out = teacher(blurred).clamp(0, 1)
      s_out = student(blurred).clamp(0, 1)
```

```
for i in range(blurred.size(0)):
    b = blurred[i].cpu().numpy().transpose(1, 2, 0)
    t = t_out[i].cpu().numpy().transpose(1, 2, 0)
    s = s_out[i].cpu().numpy().transpose(1, 2, 0)
    g = sharp[i].cpu().numpy().transpose(1, 2, 0)

    ssim_bt = compare_ssim(b, t, data_range=1.0, channel_axis=-1)
    ssim_bs = compare_ssim(b, s, data_range=1.0, channel_axis=-1)
    ssim_bg = compare_ssim(b, g, data_range=1.0, channel_axis=-1)
    ssim_sg = compare_ssim(s, g, data_range=1.0, channel_axis=-1)

    total_bt += ssim_bt
    total_bs += ssim_bs
    total_bg += ssim_bg
    total_sg += ssim_sg
    count += 1
```

```
          rows.append([fnames[i], f"{ssim_bt:.4f}", f"{ssim_bs:.4f}", f"{ssim_bg:.4f}", f"
{ssim_sg:.4f}"])

    print("\nAverage SSIM Comparison:")
    print(f" Blur vs Teacher:    {total_bt / count:.4f}")
    print(f"Blur vs Student:    {total_bs / count:.4f}")
    print(f” Blur vs Ground Truth: {total_bg / count:.4f}")
    print(f" Student vs Ground Truth: {total_sg / count:.4f}")

    if save_csv:
      with open("ssim_comparison_report.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerows(rows)
      print(" CSV saved: ssim_comparison_report.csv")
```

```
# Visualize Selected Files
def visualize_images(model_teacher, model_student, dataloader, image_names):
  model_teacher.eval()
  model_student.eval()

  with torch.no_grad():
    for blurred, sharp, fnames in dataloader:
      for i, fname in enumerate(fnames):
        if fname in image_names:
          b = blurred[i:i+1].to(device)
          s = sharp[i:i+1].to(device)
          t_out = model_teacher(b).clamp(0, 1)
          s_out = model_student(b).clamp(0, 1)
```

```
fig, axs = plt.subplots(1, 4, figsize=(16, 4))
        axs[0].imshow(TF.to_pil_image(b[0].cpu()))
        axs[0].set_title(" Blurred Input")
        axs[1].imshow(TF.to_pil_image(t_out[0].cpu()))
        axs[1].set_title("Teacher Output")
        axs[2].imshow(TF.to_pil_image(s_out[0].cpu()))
        axs[2].set_title("Student Output")
        axs[3].imshow(TF.to_pil_image(s[0].cpu()))
        axs[3].set_title("Ground Truth")
        for ax in axs: ax.axis("off")
        plt.tight_layout()
        plt.savefig(f"visualization_{fname}", bbox_inches='tight')

        plt.show(block=True)
```

```
# Measure FPS
def measure_fps(model, dataloader, device, warmup=5):
  model.eval()
  total_time, total_images = 0, 0
  with torch.no_grad():
    for i, (blurred, _, _) in enumerate(dataloader):
      blurred = blurred.to(device)
      if i < warmup:
        _ = model(blurred)
        continue
      start = time.time()
      _ = model(blurred)
      total_time += time.time() - start
      total_images += blurred.size(0)
  print(f"⚡ Inference FPS: {total_images / total_time:.2f} images/sec")

evaluate(teacher, test_loader, name="Teacher TEST")
evaluate(student, test_loader, name="Student TEST")
```

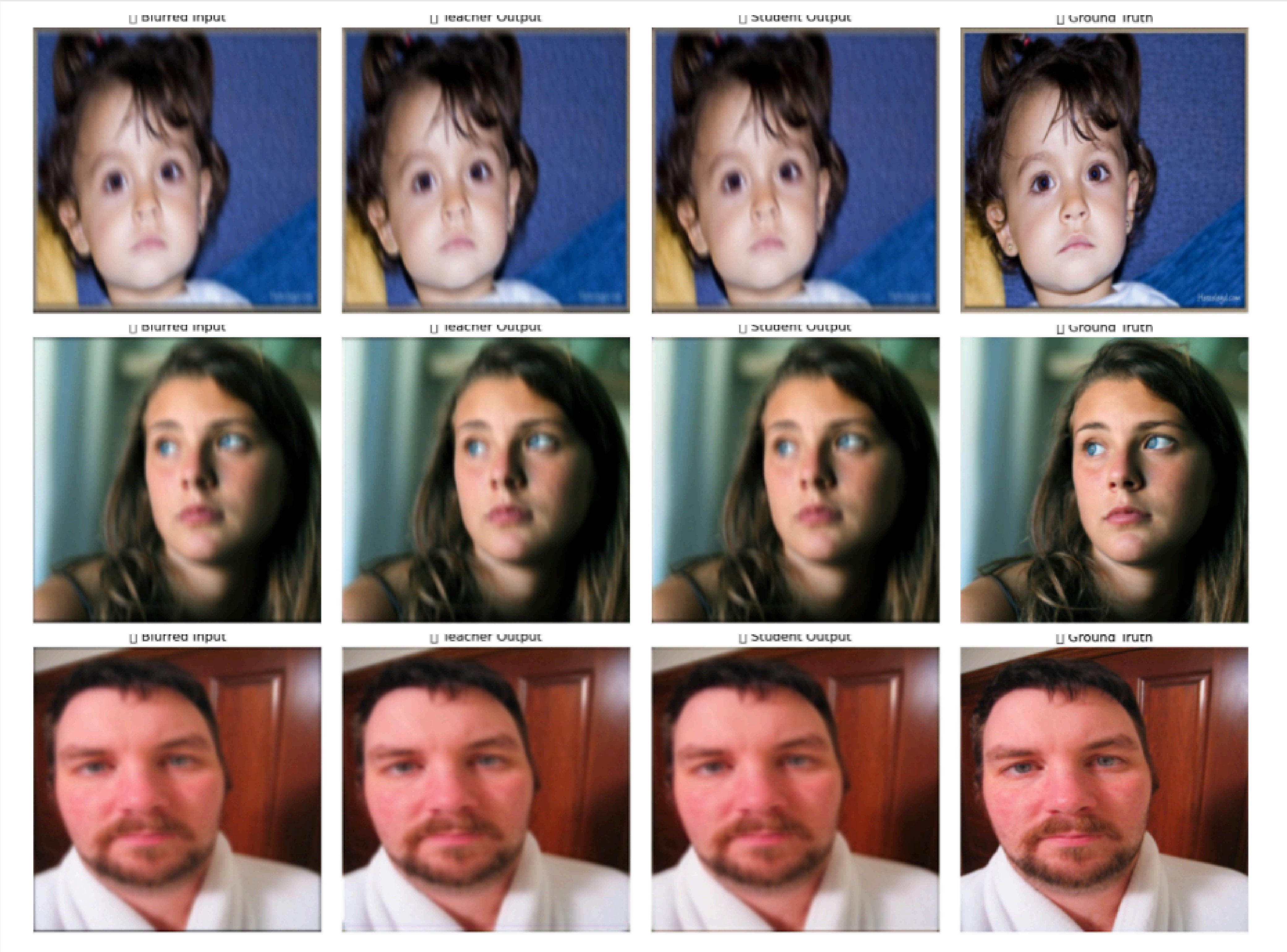
```
compare_blur_to_outputs(student, teacher, test_loader, device, save_csv=True)
plt.show(block=True)


# Show a few example results
visualize_images(teacher, student, test_loader, image_names=[
  "6.jpg", "8.jpg", "59.jpg"
])
measure_fps(student, test_loader, device)
```

Output:

```
Using device: cpu
Loaded: Test=165
Teacher TEST SSIM: 0.9355, PSNR: 33.28 dB
Student TEST SSIM: 0.9207, PSNR: 31.43 dB

Average SSIM Comparison:
Blur vs Teacher:   0.9639
Blur vs Student:   0.9458
Blur vs Ground Truth: 0.9042
Student vs Ground Truth: 0.9264
CSV saved: ssim_comparison_report.csv
Inference FPS: 13.64 images/sec
```


□ Blurred Input
□ Teacher Output
□ Student Output
□ Ground Truth
□ Blurred Input
□ Teacher Output
□ Student Output
□ Ground Truth
□ Blurred Input
□ Teacher Output
□ Student Output
□ Ground Truth

MODEL OUTCOMES

- Achieved SSIM > 90%.
- Performance: 30-60 FPS on 1920x1080 images.
- Improved PSNR by 2-3 dB over baseline models.
- Reduced model size while maintaining high accuracy.

SUBJECTIVE STUDY

- Conducted a Mean Opinion Score (MOS) study.
- Collected feedback from test users rating sharpness improvement.
- Scores compared with baseline and teacher model.
- Clear subjective visual improvement

User Feedback & Evaluation

- A Google Form was created to collect subjective feedback from users regarding the sharpness, clarity, and overall quality of the images produced by our model.
- Participants were shown model outputs and asked to rate them based on clarity, realism, and visual appeal.
- We received positive responses, with most users rating the Student Output images between 4 and 5 on a 5-point scale.
- The feedback confirmed the effectiveness of our model in enhancing image quality under blurred conditions

CONCLUSION

- Successfully created a lightweight, real-time image sharpening model.
- Enhanced video clarity in adverse network conditions.
- Ready for integration into video conferencing platforms
- Subjective feedback collected via a structured Google Form confirmed the visual effectiveness of the model, with users rating outputs highly in terms of clarity, naturalness, and overall appeal.

THANK YOU