Phase 1: Dependencies and Setup

```
# Install all required dependencies
!pip install torch torchvision torchaudio
!pip install opencv-python-headless
!pip install pillow
!pip install scikit-image
!pip install tqdm
!pip install ipywidgets

# Fix for PyTorch 2.6+ compatibility
!pip install huggingface_hub==0.25.2

# Import all required libraries
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
import cv2
import numpy as np
from PIL import Image
import os
import zipfile
import urllib.request
from tqdm import tqdm
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim
import json
import time
from google.colab import files
import shutil
import glob
import base64
from IPython.display import display, HTML
import ipywidgets as widgets

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
print(f"CUDA available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
```



```
Requirement already satisfied: websocket-client in /usr/local/lib/python3.11/dist-packages (from jupyter-serve
```

```
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio>=3.1.0->jupyte
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio>=3.1.0->jup
Downloading jedi-0.19.2-py2.py3-none-any.whl (1.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 43.4 MB/s eta 0:00:00
Installing collected packages: jedi
Successfully installed jedi-0.19.2
Collecting huggingface_hub==0.25.2
  Downloading huggingface_hub-0.25.2-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface_hub==0.25
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface_h
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface_hu
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub==0
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface_hub==0.25
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub==
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from hug
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from reque
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingf
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->h
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->h
Downloading huggingface_hub-0.25.2-py3-none-any.whl (436 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 436.6/436.6 kB 18.0 MB/s eta 0:00:00
Installing collected packages: huggingface_hub
  Attempting uninstall: huggingface_hub
    Found existing installation: huggingface-hub 0.33.2
    Uninstalling huggingface-hub-0.33.2:
      Successfully uninstalled huggingface-hub-0.33.2
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. Thi
transformers 4.53.0 requires huggingface-hub<1.0,>=0.30.0, but you have huggingface-hub 0.25.2 which is incomp
diffusers 0.34.0 requires huggingface-hub>=0.27.0, but you have huggingface-hub 0.25.2 which is incompatible.
gradio 5.31.0 requires huggingface-hub>=0.28.1, but you have huggingface-hub 0.25.2 which is incompatible.
Successfully installed huggingface_hub-0.25.2
Using device: cpu
CUDA available: False
```

Phase 2: Model Architectures

```python
# Student Model Architecture (Lightweight CNN)
class StudentModel(nn.Module):
    def __init__(self, scale_factor=4):
        super(StudentModel, self).__init__()
        self.scale_factor = scale_factor

        # Feature extraction layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, padding=1)

        # Residual blocks
        self.res_blocks = nn.ModuleList([
            self._make_residual_block(64) for _ in range(4)
        ])

        # Upsampling layers
        self.upsample = nn.Sequential(
            nn.Conv2d(64, 64 * (scale_factor ** 2), kernel_size=3, padding=1),
            nn.PixelShuffle(scale_factor),
            nn.Conv2d(64, 3, kernel_size=3, padding=1)
        )

        self.relu = nn.ReLU(inplace=True)
        self.tanh = nn.Tanh()

    def _make_residual_block(self, channels):
        return nn.Sequential(
            nn.Conv2d(channels, channels, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        )

    def forward(self, x):
```

```
            # Feature extraction
            x1 = self.relu(self.conv1(x))
            x2 = self.relu(self.conv2(x1))
            x3 = self.relu(self.conv3(x2))

            # Residual blocks
            residual = x3
            for res_block in self.res_blocks:
                out = res_block(residual)
                residual = residual + out

            # Upsampling
            out = self.upsample(residual)
            out = self.tanh(out)

            return out

    # Teacher Model: SRResNet Architecture
    class SRResNetTeacherModel:
        def __init__(self, scale=4):
            self.scale = scale
            self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

            # Create SRResNet teacher model
            self.model = self._create_srresnet_model().to(self.device)
            self.model.eval()
            print("SRResNet teacher model initialized")

        def _create_srresnet_model(self):
            """Create SRResNet-like architecture"""
            class ResidualBlock(nn.Module):
                def __init__(self, channels):
                    super().__init__()
                    self.conv1 = nn.Conv2d(channels, channels, 3, padding=1)
                    self.bn1 = nn.BatchNorm2d(channels)
                    self.conv2 = nn.Conv2d(channels, channels, 3, padding=1)
                    self.bn2 = nn.BatchNorm2d(channels)
                    self.relu = nn.ReLU(inplace=True)

                def forward(self, x):
                    residual = x
                    out = self.relu(self.bn1(self.conv1(x)))
                    out = self.bn2(self.conv2(out))
                    out += residual
                    return out

            class SRResNet(nn.Module):
                def __init__(self, scale_factor=4, num_channels=3, num_features=64, num_blocks=16):
                    super().__init__()
                    self.scale_factor = scale_factor

                    # Initial convolution
                    self.conv_input = nn.Conv2d(num_channels, num_features, 9, padding=4)

                    # Residual blocks
                    self.residual_blocks = nn.Sequential(
                        *[ResidualBlock(num_features) for _ in range(num_blocks)]
                    )

                    # Post-residual convolution
                    self.conv_mid = nn.Conv2d(num_features, num_features, 3, padding=1)
                    self.bn_mid = nn.BatchNorm2d(num_features)

                    # Upsampling layers
                    upsampling_layers = []
                    for _ in range(int(np.log2(scale_factor))):
                        upsampling_layers.extend([
                            nn.Conv2d(num_features, num_features * 4, 3, padding=1),
                            nn.PixelShuffle(2),
```

```
                        nn.ReLU(inplace=True)
                    ])
                self.upsampling = nn.Sequential(*upsampling_layers)

                # Final convolution
                self.conv_output = nn.Conv2d(num_features, num_channels, 9, padding=4)

                self.relu = nn.ReLU(inplace=True)

            def forward(self, x):
                out = self.relu(self.conv_input(x))
                residual = out
                out = self.residual_blocks(out)
                out = self.bn_mid(self.conv_mid(out))
                out += residual
                out = self.upsampling(out)
                out = self.conv_output(out)
                return torch.tanh(out)

        return SRResNet(scale_factor=self.scale)

    def enhance(self, img):
        """Enhance image using SRResNet"""
        try:
            if isinstance(img, np.ndarray):
                # Convert numpy to tensor
                if len(img.shape) == 3:
                    img_tensor = torch.from_numpy(img).float() / 255.0
                    img_tensor = img_tensor.permute(2, 0, 1).unsqueeze(0)
                else:
                    img_tensor = torch.from_numpy(img).float() / 255.0
                    img_tensor = img_tensor.unsqueeze(0).unsqueeze(0)
                    img_tensor = img_tensor.repeat(1, 3, 1, 1)
            else:
                img_tensor = img.unsqueeze(0) if len(img.shape) == 3 else img

            img_tensor = img_tensor.to(self.device)

            with torch.no_grad():
                enhanced = self.model(img_tensor)
                enhanced = enhanced.squeeze(0).permute(1, 2, 0).cpu().numpy()
                enhanced = np.clip((enhanced + 1) * 127.5, 0, 255).astype(np.uint8)

            return enhanced

        except Exception as e:
            print(f"SRResNet enhancement failed: {e}")
            # Fallback to bicubic upsampling
            h, w = img.shape[:2]
            return cv2.resize(img, (w*self.scale, h*self.scale), interpolation=cv2.INTER_CUBIC)

# Initialize models
student_model = StudentModel(scale_factor=4).to(device)
teacher_model = SRResNetTeacherModel(scale=4)

print(f"Student model parameters: {sum(p.numel() for p in student_model.parameters()):,}")
print("Models initialized successfully!")
```

```
SRResNet teacher model initialized
Student model parameters: 944,323
Models initialized successfully!
```

Phase 3: Dataset Preparation

```
# Download and setup DIV2K Dataset
def download_div2k_dataset():
    """Download and extract DIV2K dataset"""
```

```
        print("Starting DIV2K dataset download...")

        if not os.path.exists('DIV2K'):
            os.makedirs('DIV2K')

        # URLs for DIV2K dataset
        urls = {
            'train_hr': 'http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_HR.zip',
            'valid_hr': 'http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_HR.zip'
        }

        # Download files
        for dataset_type, url in urls.items():
            zip_path = f'DIV2K/DIV2K_{dataset_type.replace("_", "_")}.zip'

            if not os.path.exists(zip_path):
                print(f"Downloading {dataset_type} dataset...")
                try:
                    urllib.request.urlretrieve(url, zip_path)
                    print(f"✓ Downloaded {dataset_type}")
                except Exception as e:
                    print(f"✗ Failed to download {dataset_type}: {e}")
                    continue
            else:
                print(f"✓ {dataset_type} already exists")

        # Extract files
        for dataset_type in ['train_hr', 'valid_hr']:
            zip_path = f'DIV2K/DIV2K_{dataset_type.replace("_", "_")}.zip'
            extract_path = f'DIV2K/DIV2K_{dataset_type.replace("_", "_")}'

            if os.path.exists(zip_path) and not os.path.exists(extract_path):
                print(f"Extracting {dataset_type}...")
                try:
                    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
                        zip_ref.extractall('DIV2K/')
                    print(f"✓ Extracted {dataset_type}")
                except Exception as e:
                    print(f"✗ Failed to extract {dataset_type}: {e}")
            else:
                print(f"✓ {dataset_type} already extracted")

    # Custom Dataset Class for Knowledge Distillation
    class DIV2KDataset(Dataset):
        def __init__(self, hr_dir, crop_size=128, scale_factor=4, augment=True):
            self.hr_dir = hr_dir
            self.crop_size = crop_size
            self.scale_factor = scale_factor
            self.augment = augment

            # Get all image files
            self.hr_images = []
            for ext in ['*.png', '*.jpg', '*.jpeg', '*.PNG', '*.JPG', '*.JPEG']:
                self.hr_images.extend(glob.glob(os.path.join(hr_dir, ext)))

            print(f"Found {len(self.hr_images)} images in {hr_dir}")

            # Transforms
            self.to_tensor = transforms.ToTensor()
            self.to_pil = transforms.ToPILImage()

            # Augmentation transforms
            if augment:
                self.augment_transforms = transforms.Compose([
                    transforms.RandomHorizontalFlip(p=0.5),
                    transforms.RandomVerticalFlip(p=0.5),
                    transforms.RandomRotation(degrees=90, expand=False),
                ])
            else:
```

```
            self.augment_transforms = None

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

    def __getitem__(self, idx):
        # Load HR image
        hr_path = self.hr_images[idx]
        try:
            hr_img = Image.open(hr_path).convert('RGB')
        except Exception as e:
            print(f"Error loading {hr_path}: {e}")
            # Return a random valid image instead
            hr_img = Image.open(self.hr_images[0]).convert('RGB')

        # Ensure minimum size
        w, h = hr_img.size
        if w < self.crop_size or h < self.crop_size:
            # Resize to minimum required size
            scale = max(self.crop_size / w, self.crop_size / h)
            new_w, new_h = int(w * scale), int(h * scale)
            hr_img = hr_img.resize((new_w, new_h), Image.BICUBIC)

        # Random crop
        hr_img = transforms.RandomCrop(self.crop_size)(hr_img)

        # Apply augmentations
        if self.augment_transforms:
            hr_img = self.augment_transforms(hr_img)

        # Convert to tensor
        hr_tensor = self.to_tensor(hr_img)

        # Create LR image using bicubic downsampling
        lr_size = self.crop_size // self.scale_factor
        lr_img = transforms.Resize(lr_size, transforms.InterpolationMode.BICUBIC)(hr_img)
        lr_tensor = self.to_tensor(lr_img)

        return lr_tensor, hr_tensor

# Setup datasets
download_div2k_dataset()

train_dataset = DIV2KDataset('DIV2K/DIV2K_train_HR', crop_size=128, augment=True)
val_dataset = DIV2KDataset('DIV2K/DIV2K_valid_HR', crop_size=128, augment=False)

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True, num_workers=2, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False, num_workers=2, pin_memory=True)

print(f"✓ Training dataset: {len(train_dataset)} images")
print(f"✓ Validation dataset: {len(val_dataset)} images")
print(f"✓ Training batches: {len(train_loader)}")
print(f"✓ Validation batches: {len(val_loader)}")
```

```
Starting DIV2K dataset download...
Downloading train_hr dataset...
✓ Downloaded train_hr
Downloading valid_hr dataset...
✓ Downloaded valid_hr
Extracting train_hr...
✓ Extracted train_hr
Extracting valid_hr...
✓ Extracted valid_hr
Found 800 images in DIV2K/DIV2K_train_HR
Found 100 images in DIV2K/DIV2K_valid_HR
✓ Training dataset: 800 images
✓ Validation dataset: 100 images
✓ Training batches: 100
✓ Validation batches: 25
```

Phase 4: Enhanced 25-Epoch Training

```
# Enhanced Knowledge Distillation Loss for 25 epochs
class EnhancedKnowledgeDistillationLoss(nn.Module):
    def __init__(self, alpha=0.7, temperature=4, perceptual_weight=0.1):
        super().__init__()
        self.alpha = alpha
        self.temperature = temperature
        self.perceptual_weight = perceptual_weight
        self.mse_loss = nn.MSELoss()
        self.l1_loss = nn.L1Loss()

    def forward(self, student_output, teacher_output, target):
        # Reconstruction loss (student vs ground truth)
        recon_loss = self.mse_loss(student_output, target)

        # Feature distillation loss (student vs teacher)
        if teacher_output.shape != student_output.shape:
            teacher_output = F.interpolate(teacher_output,
                                           size=student_output.shape[2:],
                                           mode='bilinear',
                                           align_corners=False)

        feature_loss = self.mse_loss(student_output, teacher_output)

        # Enhanced perceptual loss (edge-based)
        edge_loss = self._edge_loss(student_output, target)

        # Combined loss
        total_loss = (self.alpha * recon_loss +
                      (1 - self.alpha) * feature_loss +
                      self.perceptual_weight * edge_loss)

        return total_loss, recon_loss, feature_loss, edge_loss

    def _edge_loss(self, pred, target):
        """Calculate edge-based perceptual loss"""
        # Simple edge detection using gradients
        def get_edges(x):
            grad_x = torch.abs(x[:, :, :, :-1] - x[:, :, :, 1:])
            grad_y = torch.abs(x[:, :, :-1, :] - x[:, :, 1:, :])
            return grad_x, grad_y

        pred_grad_x, pred_grad_y = get_edges(pred)
        target_grad_x, target_grad_y = get_edges(target)

        edge_loss = (self.mse_loss(pred_grad_x, target_grad_x) +
                     self.mse_loss(pred_grad_y, target_grad_y))

        return edge_loss

# Enhanced training function for 25 epochs
def train_student_model_25_epochs(student_model, teacher_model, train_loader, val_loader, epochs=25):
    """Enhanced training with 25 epochs and improved monitoring"""

    print("Starting Enhanced Knowledge Distillation Training (25 Epochs)...")
    print(f"Student model parameters: {sum(p.numel() for p in student_model.parameters()):,}")

    # Enhanced loss and optimizer
    criterion = EnhancedKnowledgeDistillationLoss(alpha=0.7, temperature=4, perceptual_weight=0.1)
    optimizer = optim.Adam(student_model.parameters(), lr=1e-4, weight_decay=1e-6)

    # Enhanced scheduler for 25 epochs
    scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[10, 20], gamma=0.5)

    # Training tracking
    best_val_loss = float('inf')
```

```
        best_ssim = 0.0
        train_losses = []
        val_losses = []
        recon_losses = []
        feature_losses = []
        edge_losses = []
        ssim_history = []

        # Early stopping parameters
        patience = 8
        patience_counter = 0

        # Training loop
        for epoch in range(epochs):
            print(f"\nEpoch {epoch+1}/{epochs}")
            print("-" * 50)

            # Training phase
            student_model.train()
            epoch_train_loss = 0.0
            epoch_recon_loss = 0.0
            epoch_feature_loss = 0.0
            epoch_edge_loss = 0.0

            train_pbar = tqdm(train_loader, desc=f'Training Epoch {epoch+1}')

            for batch_idx, (lr_batch, hr_batch) in enumerate(train_pbar):
                lr_batch, hr_batch = lr_batch.to(device), hr_batch.to(device)

                # Get teacher predictions (no gradients)
                with torch.no_grad():
                    teacher_outputs = []
                    for i in range(lr_batch.size(0)):
                        # Convert tensor to numpy for teacher model
                        lr_img = lr_batch[i].cpu().numpy().transpose(1, 2, 0)
                        lr_img = (lr_img * 255).astype(np.uint8)

                        # Get teacher enhancement
                        teacher_enhanced = teacher_model.enhance(lr_img)

                        # Convert back to tensor
                        teacher_enhanced = torch.from_numpy(teacher_enhanced).float() / 255.0
                        teacher_enhanced = teacher_enhanced.permute(2, 0, 1)
                        teacher_outputs.append(teacher_enhanced)

                    teacher_batch = torch.stack(teacher_outputs).to(device)

                # Student forward pass
                optimizer.zero_grad()
                student_output = student_model(lr_batch)

                # Calculate enhanced loss
                total_loss, recon_loss, feature_loss, edge_loss = criterion(
                    student_output, teacher_batch, hr_batch
                )

                # Backward pass
                total_loss.backward()
                torch.nn.utils.clip_grad_norm_(student_model.parameters(), max_norm=1.0)
                optimizer.step()

                # Update metrics
                epoch_train_loss += total_loss.item()
                epoch_recon_loss += recon_loss.item()
                epoch_feature_loss += feature_loss.item()
                epoch_edge_loss += edge_loss.item()

                # Update progress bar
                train_pbar.set_postfix({
```

```
            'Loss': f'{total_loss.item():.4f}',
            'Recon': f'{recon_loss.item():.4f}',
            'Feature': f'{feature_loss.item():.4f}',
            'Edge': f'{edge_loss.item():.4f}'
        })

    # Calculate average training losses
    avg_train_loss = epoch_train_loss / len(train_loader)
    avg_recon_loss = epoch_recon_loss / len(train_loader)
    avg_feature_loss = epoch_feature_loss / len(train_loader)
    avg_edge_loss = epoch_edge_loss / len(train_loader)

    # Validation phase
    student_model.eval()
    epoch_val_loss = 0.0
    val_ssim_scores = []

    val_pbar = tqdm(val_loader, desc=f'Validation Epoch {epoch+1}')

    with torch.no_grad():
        for lr_batch, hr_batch in val_pbar:
            lr_batch, hr_batch = lr_batch.to(device), hr_batch.to(device)

            # Student prediction
            student_output = student_model(lr_batch)

            # Validation loss (reconstruction only)
            val_loss = F.mse_loss(student_output, hr_batch)
            epoch_val_loss += val_loss.item()

            # Calculate SSIM for first image in batch
            student_img = student_output[0].cpu().numpy().transpose(1, 2, 0)
            hr_img = hr_batch[0].cpu().numpy().transpose(1, 2, 0)

            student_img = np.clip(student_img * 255, 0, 255).astype(np.uint8)
            hr_img = np.clip(hr_img * 255, 0, 255).astype(np.uint8)

            # Convert to grayscale for SSIM
            student_gray = cv2.cvtColor(student_img, cv2.COLOR_RGB2GRAY)
            hr_gray = cv2.cvtColor(hr_img, cv2.COLOR_RGB2GRAY)

            ssim_score = ssim(hr_gray, student_gray, data_range=255)
            val_ssim_scores.append(ssim_score)

            val_pbar.set_postfix({
                'Val Loss': f'{val_loss.item():.4f}',
                'SSIM': f'{ssim_score:.4f}'
            })

    avg_val_loss = epoch_val_loss / len(val_loader)
    avg_ssim = np.mean(val_ssim_scores)

    # Store metrics
    train_losses.append(avg_train_loss)
    val_losses.append(avg_val_loss)
    recon_losses.append(avg_recon_loss)
    feature_losses.append(avg_feature_loss)
    edge_losses.append(avg_edge_loss)
    ssim_history.append(avg_ssim)

    # Print epoch summary
    print(f"\nEpoch {epoch+1} Summary:")
    print(f"  Train Loss: {avg_train_loss:.4f}")
    print(f"  Val Loss: {avg_val_loss:.4f}")
    print(f"  Reconstruction Loss: {avg_recon_loss:.4f}")
    print(f"  Feature Loss: {avg_feature_loss:.4f}")
    print(f"  Edge Loss: {avg_edge_loss:.4f}")
    print(f"  Validation SSIM: {avg_ssim:.4f} ({avg_ssim*100:.2f}%)")
    print(f"  Learning Rate: {optimizer.param_groups[0]['lr']:.6f}")
```

```
            # Enhanced model saving
            if avg_val_loss < best_val_loss or avg_ssim > best_ssim:
                if avg_val_loss < best_val_loss:
                    best_val_loss = avg_val_loss
                    patience_counter = 0
                if avg_ssim > best_ssim:
                    best_ssim = avg_ssim

                torch.save({
                    'epoch': epoch,
                    'model_state_dict': student_model.state_dict(),
                    'optimizer_state_dict': optimizer.state_dict(),
                    'val_loss': avg_val_loss,
                    'ssim': avg_ssim,
                    'train_losses': train_losses,
                    'val_losses': val_losses,
                    'ssim_history': ssim_history
                }, 'best_student_model_25epochs.pth')
                    print(f"  ✓ New best model saved! (Val Loss: {avg_val_loss:.4f}, SSIM: {avg_ssim:.4f})")
            else:
                patience_counter += 1

            # Early stopping check
            if patience_counter >= patience:
                print(f"\nEarly stopping triggered after {epoch+1} epochs (patience: {patience})")
                break

            # Update learning rate
            scheduler.step()

            # Plot progress every 5 epochs
            if (epoch + 1) % 5 == 0:
                plot_enhanced_training_progress(train_losses, val_losses, recon_losses,
                                                feature_losses, edge_losses, ssim_history)

        print(f"\nTraining completed!")
        print(f"Best validation loss: {best_val_loss:.4f}")
        print(f"Best SSIM score: {best_ssim:.4f} ({best_ssim*100:.2f}%)")

        return train_losses, val_losses, recon_losses, feature_losses, edge_losses, ssim_history

    # Enhanced training progress visualization
    def plot_enhanced_training_progress(train_losses, val_losses, recon_losses,
                                        feature_losses, edge_losses, ssim_history):
        """Plot enhanced training progress with all metrics"""
        fig, axes = plt.subplots(2, 3, figsize=(18, 10))

        # Overall losses
        axes[0, 0].plot(train_losses, label='Training Loss', color='blue', marker='o')
        axes[0, 0].plot(val_losses, label='Validation Loss', color='red', marker='s')
        axes[0, 0].set_xlabel('Epoch')
        axes[0, 0].set_ylabel('Loss')
        axes[0, 0].set_title('Overall Training Progress (25 Epochs)')
        axes[0, 0].legend()
        axes[0, 0].grid(True)

        # Loss components
        axes[0, 1].plot(recon_losses, label='Reconstruction Loss', color='green', marker='^')
        axes[0, 1].plot(feature_losses, label='Feature Distillation Loss', color='orange', marker='v')
        axes[0, 1].plot(edge_losses, label='Edge Loss', color='purple', marker='d')
        axes[0, 1].set_xlabel('Epoch')
        axes[0, 1].set_ylabel('Loss')
        axes[0, 1].set_title('Loss Components')
        axes[0, 1].legend()
        axes[0, 1].grid(True)

        # SSIM Progress
        axes[0, 2].plot(ssim_history, label='Validation SSIM', color='cyan', marker='*')
```

```
        axes[0, 2].axhline(y=0.9, color='red', linestyle='--', label='Target (90%)')
        axes[0, 2].set_xlabel('Epoch')
        axes[0, 2].set_ylabel('SSIM Score')
        axes[0, 2].set_title('SSIM Progress')
        axes[0, 2].legend()
        axes[0, 2].grid(True)

        # Loss ratio analysis
        if len(recon_losses) > 0 and len(feature_losses) > 0:
            ratios = [r/f if f > 0 else 0 for r, f in zip(recon_losses, feature_losses)]
            axes[1, 0].plot(ratios, label='Recon/Feature Ratio', color='purple', marker='d')
            axes[1, 0].set_xlabel('Epoch')
            axes[1, 0].set_ylabel('Ratio')
            axes[1, 0].set_title('Loss Ratio Analysis')
            axes[1, 0].legend()
            axes[1, 0].grid(True)

        # Validation trend
        if len(val_losses) > 1:
            val_trend = np.diff(val_losses)
            axes[1, 1].plot(val_trend, label='Validation Trend', color='red', marker='x')
            axes[1, 1].axhline(y=0, color='black', linestyle='--', alpha=0.5)
            axes[1, 1].set_xlabel('Epoch')
            axes[1, 1].set_ylabel('Loss Change')
            axes[1, 1].set_title('Validation Loss Trend')
            axes[1, 1].legend()
            axes[1, 1].grid(True)

        # SSIM improvement
        if len(ssim_history) > 1:
            ssim_trend = np.diff(ssim_history)
            axes[1, 2].plot(ssim_trend, label='SSIM Improvement', color='green', marker='+')
            axes[1, 2].axhline(y=0, color='black', linestyle='--', alpha=0.5)
            axes[1, 2].set_xlabel('Epoch')
            axes[1, 2].set_ylabel('SSIM Change')
            axes[1, 2].set_title('SSIM Improvement Trend')
            axes[1, 2].legend()
            axes[1, 2].grid(True)

        plt.tight_layout()
        plt.show()

    # Load best model function
    def load_best_student_model_25epochs(student_model, checkpoint_path='best_student_model_25epochs.pth'):
        """Load the best trained student model from 25 epochs"""
        if os.path.exists(checkpoint_path):
            try:
                checkpoint = torch.load(checkpoint_path, map_location=device, weights_only=False)
                student_model.load_state_dict(checkpoint['model_state_dict'])
                print(f"✓ Loaded best model from epoch {checkpoint['epoch']}")
                print(f"  Validation Loss: {checkpoint['val_loss']:.4f}")
                print(f"  SSIM Score: {checkpoint['ssim']:.4f} ({checkpoint['ssim']*100:.2f}%)")
                return checkpoint
            except Exception as e:
                print(f"✗ Error loading checkpoint: {e}")
                return None
        else:
            print(f"✗ Checkpoint not found: {checkpoint_path}")
            return None

    print("Enhanced 25-epoch training setup complete!")
    print("Run the following to start training:")
    print("results = train_student_model_25_epochs(student_model, teacher_model, train_loader, val_loader, epochs=25)")
```

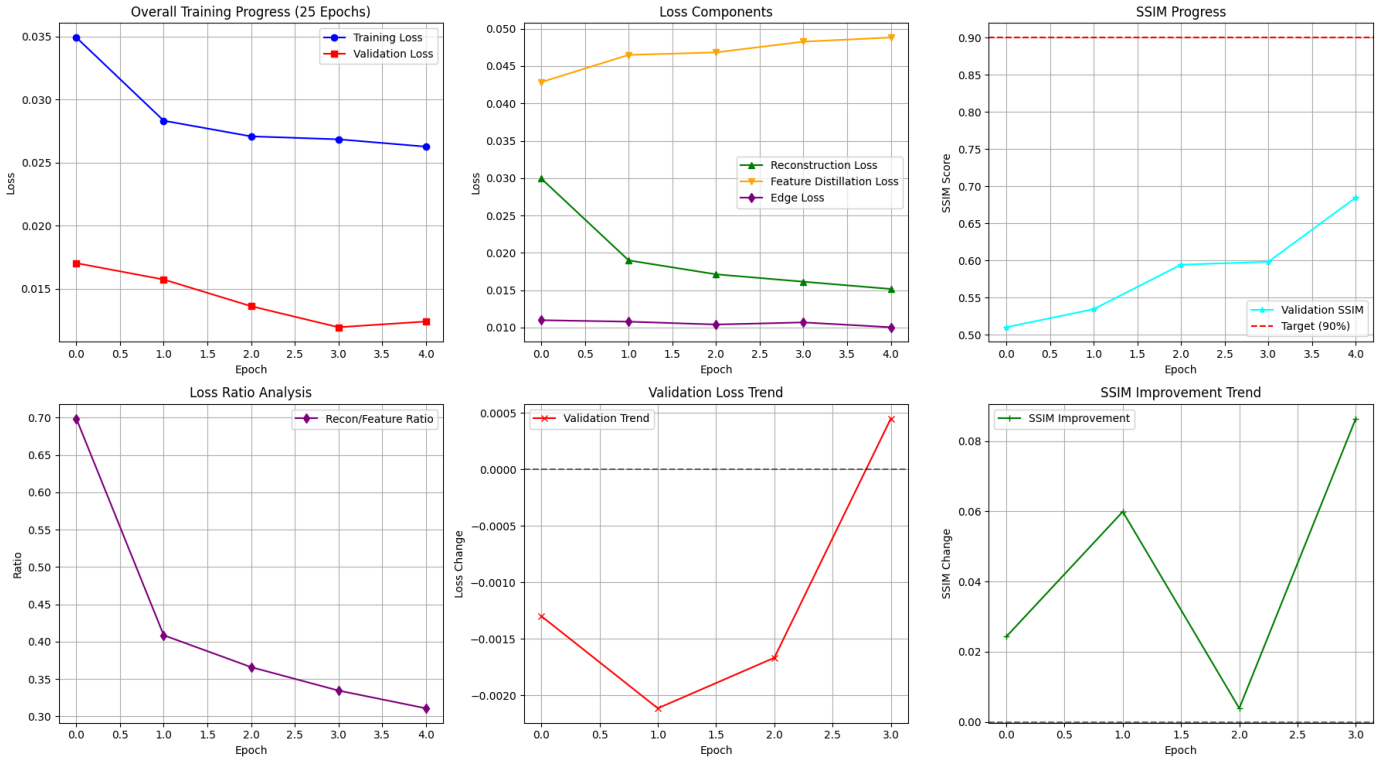
```
Enhanced 25-epoch training setup complete!
Run the following to start training:
results = train_student_model_25_epochs(student_model, teacher_model, train_loader, val_loader, epochs=25)
```

Running the above command

```
results = train_student_model_25_epochs(student_model, teacher_model, train_loader, val_loader, epochs=25)
```

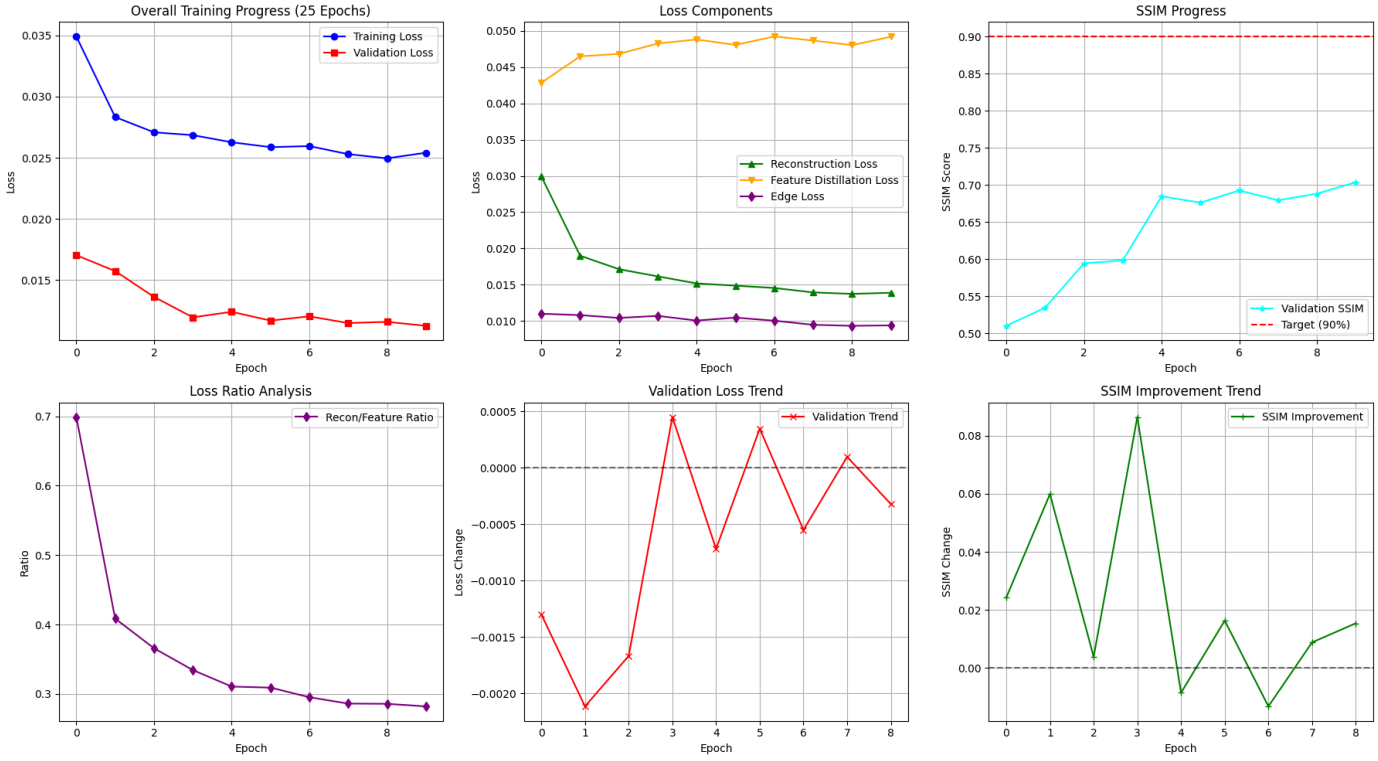
```
Starting Enhanced Knowledge Distillation Training (25 Epochs)...
Student model parameters: 944,323

Epoch 1/25
--------------------------------------------------
Training Epoch 1: 100%|██████████| 100/100 [06:00<00:00,  3.60s/it, Loss=0.0295, Recon=0.0196, Feature=0.0493,
Validation Epoch 1: 100%|██████████| 25/25 [00:15<00:00,  1.65it/s, Val Loss=0.0215, SSIM=0.5924]

Epoch 1 Summary:
  Train Loss: 0.0349
  Val Loss: 0.0170
  Reconstruction Loss: 0.0299
  Feature Loss: 0.0428
  Edge Loss: 0.0110
  Validation SSIM: 0.5100 (51.00%)
  Learning Rate: 0.000100
  ✓ New best model saved! (Val Loss: 0.0170, SSIM: 0.5100)

Epoch 2/25
--------------------------------------------------
Training Epoch 2: 100%|██████████| 100/100 [05:54<00:00,  3.55s/it, Loss=0.0256, Recon=0.0161, Feature=0.0449,
Validation Epoch 2: 100%|██████████| 25/25 [00:14<00:00,  1.77it/s, Val Loss=0.0123, SSIM=0.8320]

Epoch 2 Summary:
  Train Loss: 0.0283
  Val Loss: 0.0157
  Reconstruction Loss: 0.0190
  Feature Loss: 0.0465
  Edge Loss: 0.0108
  Validation SSIM: 0.5343 (53.43%)
  Learning Rate: 0.000100
  ✓ New best model saved! (Val Loss: 0.0157, SSIM: 0.5343)

Epoch 3/25
--------------------------------------------------
Training Epoch 3: 100%|██████████| 100/100 [06:03<00:00,  3.63s/it, Loss=0.0294, Recon=0.0192, Feature=0.0497,
Validation Epoch 3: 100%|██████████| 25/25 [00:14<00:00,  1.77it/s, Val Loss=0.0106, SSIM=0.8768]

Epoch 3 Summary:
  Train Loss: 0.0271
  Val Loss: 0.0136
  Reconstruction Loss: 0.0171
  Feature Loss: 0.0468
  Edge Loss: 0.0104
  Validation SSIM: 0.5942 (59.42%)
  Learning Rate: 0.000100
  ✓ New best model saved! (Val Loss: 0.0136, SSIM: 0.5942)

Epoch 4/25
--------------------------------------------------
Training Epoch 4: 100%|██████████| 100/100 [06:05<00:00,  3.65s/it, Loss=0.0235, Recon=0.0151, Feature=0.0400,
Validation Epoch 4: 100%|██████████| 25/25 [00:14<00:00,  1.71it/s, Val Loss=0.0128, SSIM=0.9087]

Epoch 4 Summary:
  Train Loss: 0.0268
  Val Loss: 0.0120
  Reconstruction Loss: 0.0161
  Feature Loss: 0.0483
  Edge Loss: 0.0107
  Validation SSIM: 0.5981 (59.81%)
  Learning Rate: 0.000100
  ✓ New best model saved! (Val Loss: 0.0120, SSIM: 0.5981)

Epoch 5/25
--------------------------------------------------
Training Epoch 5: 100%|██████████| 100/100 [06:02<00:00,  3.63s/it, Loss=0.0200, Recon=0.0117, Feature=0.0371,
Validation Epoch 5: 100%|██████████| 25/25 [00:14<00:00,  1.76it/s, Val Loss=0.0145, SSIM=0.9203]

Epoch 5 Summary:
  Train Loss: 0.0263
  Val Loss: 0.0124
  Reconstruction Loss: 0.0152
  Feature Loss: 0.0488
  Edge Loss: 0.0101
  Validation SSIM: 0.6845 (68.45%)
  Learning Rate: 0.000100
```

```
  ✓ New best model saved! (Val Loss: 0.0124, SSIM: 0.6845)
```



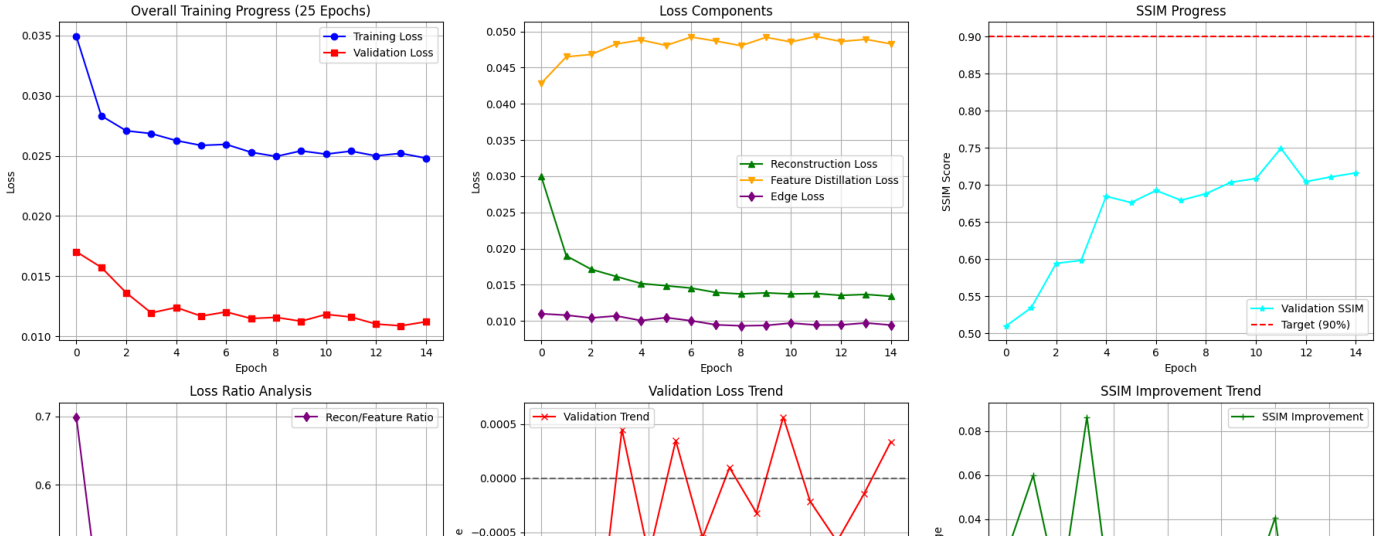

```
Epoch 6/25
--------------------------------------------------
Training Epoch 6: 100%|██████████| 100/100 [05:58<00:00,  3.59s/it, Loss=0.0279, Recon=0.0142, Feature=0.0568,
Validation Epoch 6: 100%|██████████| 25/25 [00:13<00:00,  1.79it/s, Val Loss=0.0075, SSIM=0.6440]

Epoch 6 Summary:
  Train Loss: 0.0259
  Val Loss: 0.0117
  Reconstruction Loss: 0.0149
  Feature Loss: 0.0481
  Edge Loss: 0.0105
  Validation SSIM: 0.6760 (67.60%)
  Learning Rate: 0.000100
  ✓ New best model saved! (Val Loss: 0.0117, SSIM: 0.6760)

Epoch 7/25
--------------------------------------------------
Training Epoch 7: 100%|██████████| 100/100 [05:58<00:00,  3.59s/it, Loss=0.0202, Recon=0.0105, Feature=0.0407,
Validation Epoch 7: 100%|██████████| 25/25 [00:13<00:00,  1.79it/s, Val Loss=0.0113, SSIM=0.9421]

Epoch 7 Summary:
  Train Loss: 0.0260
  Val Loss: 0.0120
  Reconstruction Loss: 0.0145
  Feature Loss: 0.0492
  Edge Loss: 0.0100
  Validation SSIM: 0.6924 (69.24%)
  Learning Rate: 0.000100
  ✓ New best model saved! (Val Loss: 0.0120, SSIM: 0.6924)

Epoch 8/25
--------------------------------------------------
Training Epoch 8: 100%|██████████| 100/100 [06:00<00:00,  3.61s/it, Loss=0.0242, Recon=0.0135, Feature=0.0462,
Validation Epoch 8: 100%|██████████| 25/25 [00:14<00:00,  1.78it/s, Val Loss=0.0136, SSIM=0.7050]

Epoch 8 Summary:
  Train Loss: 0.0253
  Val Loss: 0.0115
  Reconstruction Loss: 0.0139
  Feature Loss: 0.0487
  Edge Loss: 0.0095
  Validation SSIM: 0.6792 (67.92%)
  Learning Rate: 0.000100
  ✓ New best model saved! (Val Loss: 0.0115, SSIM: 0.6792)
```

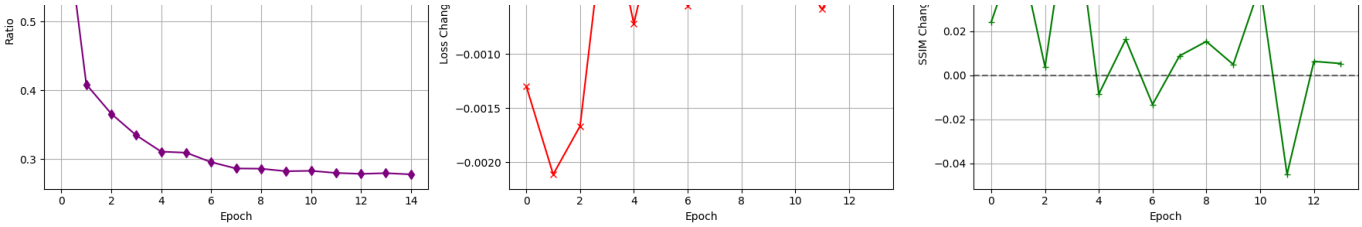
```
Epoch 9/25
--------------------------------------------------
Training Epoch 9: 100%|██████████| 100/100 [05:50<00:00,  3.50s/it, Loss=0.0277, Recon=0.0162, Feature=0.0517,
Validation Epoch 9: 100%|██████████| 25/25 [00:21<00:00,  1.14it/s, Val Loss=0.0085, SSIM=0.9523]

Epoch 9 Summary:
  Train Loss: 0.0250
  Val Loss: 0.0116
  Reconstruction Loss: 0.0137
  Feature Loss: 0.0480
  Edge Loss: 0.0093
  Validation SSIM: 0.6881 (68.81%)
  Learning Rate: 0.000100

Epoch 10/25
--------------------------------------------------
Training Epoch 10: 100%|██████████| 100/100 [05:52<00:00,  3.53s/it, Loss=0.0274, Recon=0.0132, Feature=0.0590,
Validation Epoch 10: 100%|██████████| 25/25 [00:13<00:00,  1.79it/s, Val Loss=0.0103, SSIM=0.9550]

Epoch 10 Summary:
  Train Loss: 0.0254
  Val Loss: 0.0113
  Reconstruction Loss: 0.0139
  Feature Loss: 0.0492
  Edge Loss: 0.0094
  Validation SSIM: 0.7035 (70.35%)
  Learning Rate: 0.000100
  ✓ New best model saved! (Val Loss: 0.0113, SSIM: 0.7035)
```




```
Epoch 11/25
--------------------------------------------------
Training Epoch 11: 100%|██████████| 100/100 [05:57<00:00,  3.57s/it, Loss=0.0235, Recon=0.0133, Feature=0.0443,
Validation Epoch 11: 100%|██████████| 25/25 [00:14<00:00,  1.76it/s, Val Loss=0.0094, SSIM=0.9565]

Epoch 11 Summary:
  Train Loss: 0.0251
  Val Loss: 0.0118
  Reconstruction Loss: 0.0137
  Feature Loss: 0.0485
  Edge Loss: 0.0097
  Validation SSIM: 0.7085 (70.85%)
  Learning Rate: 0.000050
  ✓ New best model saved! (Val Loss: 0.0118, SSIM: 0.7085)

Epoch 12/25
--------------------------------------------------
Training Epoch 12: 100%|██████████| 100/100 [05:54<00:00,  3.55s/it, Loss=0.0226, Recon=0.0113, Feature=0.0471,
```

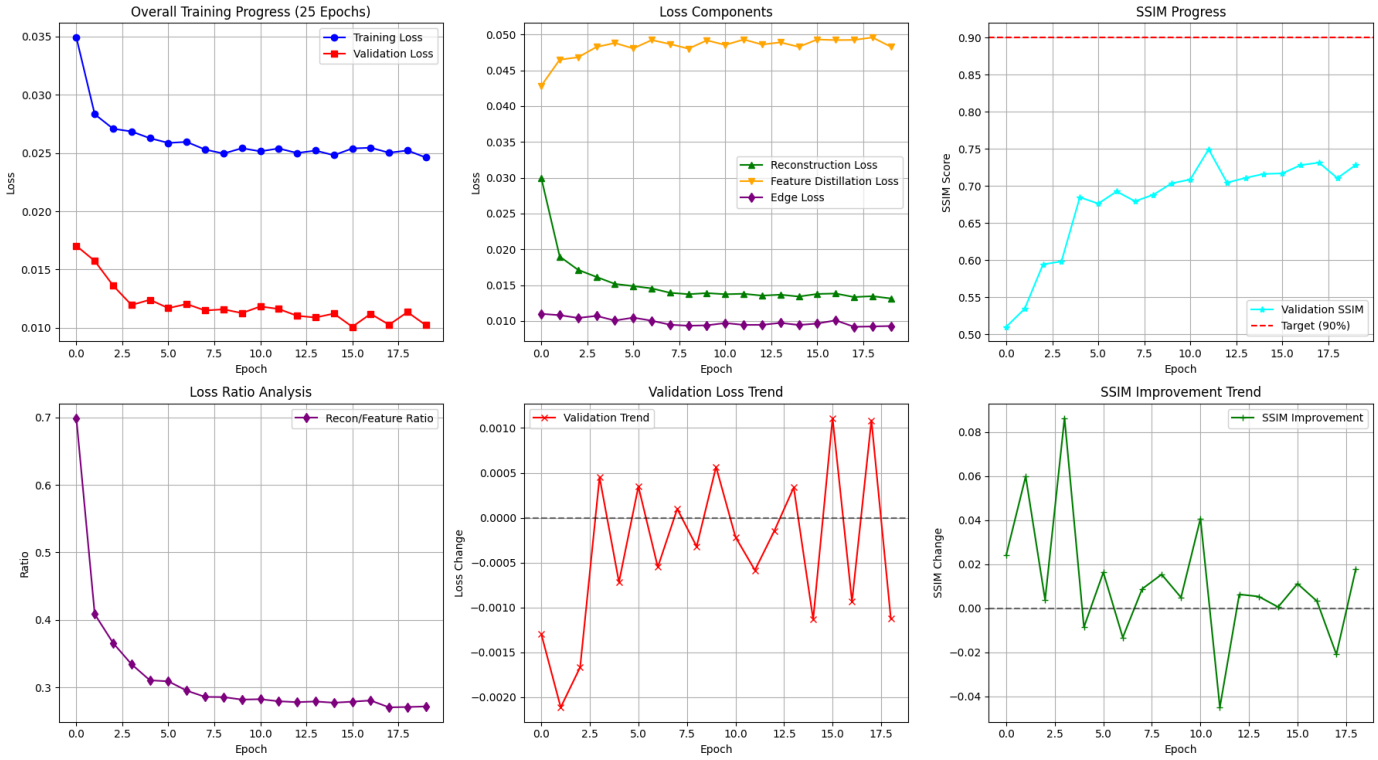
```
Training Epoch 12: 100%|██████████| 100/100 [05:54<00:00,  3.53s/it, Loss=0.0226, Recon=0.0113, Feature=0.0471,
Validation Epoch 12: 100%|██████████| 25/25 [00:14<00:00,  1.76it/s, Val Loss=0.0134, SSIM=0.9580]

Epoch 12 Summary:
  Train Loss: 0.0254
  Val Loss: 0.0116
  Reconstruction Loss: 0.0138
  Feature Loss: 0.0493
  Edge Loss: 0.0094
  Validation SSIM: 0.7492 (74.92%)
  Learning Rate: 0.000050
  ✓ New best model saved! (Val Loss: 0.0116, SSIM: 0.7492)

Epoch 13/25
--------------------------------------------------
Training Epoch 13: 100%|██████████| 100/100 [05:57<00:00,  3.57s/it, Loss=0.0249, Recon=0.0137, Feature=0.0468,
Validation Epoch 13: 100%|██████████| 25/25 [00:15<00:00,  1.61it/s, Val Loss=0.0105, SSIM=0.9532]

Epoch 13 Summary:
  Train Loss: 0.0250
  Val Loss: 0.0110
  Reconstruction Loss: 0.0135
  Feature Loss: 0.0486
  Edge Loss: 0.0095
  Validation SSIM: 0.7043 (70.43%)
  Learning Rate: 0.000050
  ✓ New best model saved! (Val Loss: 0.0110, SSIM: 0.7043)

Epoch 14/25
--------------------------------------------------
Training Epoch 14: 100%|██████████| 100/100 [05:59<00:00,  3.59s/it, Loss=0.0289, Recon=0.0147, Feature=0.0591,
Validation Epoch 14: 100%|██████████| 25/25 [00:15<00:00,  1.62it/s, Val Loss=0.0098, SSIM=0.9588]

Epoch 14 Summary:
  Train Loss: 0.0252
  Val Loss: 0.0109
  Reconstruction Loss: 0.0137
  Feature Loss: 0.0489
  Edge Loss: 0.0097
  Validation SSIM: 0.7107 (71.07%)
  Learning Rate: 0.000050
  ✓ New best model saved! (Val Loss: 0.0109, SSIM: 0.7107)

Epoch 15/25
--------------------------------------------------
Training Epoch 15: 100%|██████████| 100/100 [05:51<00:00,  3.52s/it, Loss=0.0242, Recon=0.0143, Feature=0.0432,
Validation Epoch 15: 100%|██████████| 25/25 [00:14<00:00,  1.76it/s, Val Loss=0.0095, SSIM=0.9584]

Epoch 15 Summary:
  Train Loss: 0.0248
  Val Loss: 0.0112
  Reconstruction Loss: 0.0134
  Feature Loss: 0.0483
  Edge Loss: 0.0094
  Validation SSIM: 0.7161 (71.61%)
  Learning Rate: 0.000050
```






```
Epoch 16/25
--------------------------------------------------
Training Epoch 16: 100%|██████████| 100/100 [06:01<00:00,  3.61s/it, Loss=0.0254, Recon=0.0121, Feature=0.0544,
Validation Epoch 16: 100%|██████████| 25/25 [00:14<00:00,  1.74it/s, Val Loss=0.0098, SSIM=0.6952]

Epoch 16 Summary:
  Train Loss: 0.0254
  Val Loss: 0.0101
  Reconstruction Loss: 0.0138
  Feature Loss: 0.0493
  Edge Loss: 0.0096
  Validation SSIM: 0.7167 (71.67%)
  Learning Rate: 0.000050
  ✓ New best model saved! (Val Loss: 0.0101, SSIM: 0.7167)

Epoch 17/25
--------------------------------------------------
Training Epoch 17: 100%|██████████| 100/100 [06:00<00:00,  3.61s/it, Loss=0.0244, Recon=0.0125, Feature=0.0504,
Validation Epoch 17: 100%|██████████| 25/25 [00:14<00:00,  1.70it/s, Val Loss=0.0112, SSIM=0.9633]

Epoch 17 Summary:
  Train Loss: 0.0255
  Val Loss: 0.0112
  Reconstruction Loss: 0.0138
  Feature Loss: 0.0492
  Edge Loss: 0.0101
  Validation SSIM: 0.7279 (72.79%)
  Learning Rate: 0.000050

Epoch 18/25
--------------------------------------------------
Training Epoch 18: 100%|██████████| 100/100 [06:10<00:00,  3.70s/it, Loss=0.0251, Recon=0.0122, Feature=0.0534,
Validation Epoch 18: 100%|██████████| 25/25 [00:15<00:00,  1.61it/s, Val Loss=0.0092, SSIM=0.5831]

Epoch 18 Summary:
  Train Loss: 0.0250
  Val Loss: 0.0103
  Reconstruction Loss: 0.0133
  Feature Loss: 0.0493
  Edge Loss: 0.0092
  Validation SSIM: 0.7312 (73.12%)
  Learning Rate: 0.000050

Epoch 19/25
--------------------------------------------------
Training Epoch 19: 100%|██████████| 100/100 [06:01<00:00,  3.61s/it, Loss=0.0242, Recon=0.0128, Feature=0.0467,
Validation Epoch 19: 100%|██████████| 25/25 [00:14<00:00,  1.71it/s, Val Loss=0.0091, SSIM=0.9582]

Epoch 19 Summary:
  Train Loss: 0.0252
  Val Loss: 0.0114
  Reconstruction Loss: 0.0135
  Feature Loss: 0.0496
  Edge Loss: 0.0092
  Validation SSIM: 0.7104 (71.04%)
  Learning Rate: 0.000050

Epoch 20/25
--------------------------------------------------
Training Epoch 20: 100%|██████████| 100/100 [05:57<00:00,  3.57s/it, Loss=0.0257, Recon=0.0137, Feature=0.0500,
Validation Epoch 20: 100%|██████████| 25/25 [00:14<00:00,  1.68it/s, Val Loss=0.0093, SSIM=0.9567]

Epoch 20 Summary:
  Train Loss: 0.0246
  Val Loss: 0.0102
  Reconstruction Loss: 0.0131
  Feature Loss: 0.0483
```

```
  Edge Loss: 0.0093
  Validation SSIM: 0.7282 (72.82%)
  Learning Rate: 0.000050
```




```
Epoch 21/25
--------------------------------------------------
Training Epoch 21: 100%|██████████| 100/100 [06:03<00:00,  3.64s/it, Loss=0.0282, Recon=0.0139, Feature=0.0596,
Validation Epoch 21: 100%|██████████| 25/25 [00:14<00:00,  1.73it/s, Val Loss=0.0087, SSIM=0.9619]

Epoch 21 Summary:
  Train Loss: 0.0250
  Val Loss: 0.0112
  Reconstruction Loss: 0.0132
  Feature Loss: 0.0493
  Edge Loss: 0.0091
  Validation SSIM: 0.7067 (70.67%)
  Learning Rate: 0.000025

Epoch 22/25
--------------------------------------------------
Training Epoch 22: 100%|██████████| 100/100 [05:56<00:00,  3.57s/it, Loss=0.0189, Recon=0.0111, Feature=0.0333,
Validation Epoch 22: 100%|██████████| 25/25 [00:14<00:00,  1.68it/s, Val Loss=0.0094, SSIM=0.9619]

Epoch 22 Summary:
  Train Loss: 0.0250
  Val Loss: 0.0114
  Reconstruction Loss: 0.0134
  Feature Loss: 0.0492
  Edge Loss: 0.0094
  Validation SSIM: 0.7577 (75.77%)
  Learning Rate: 0.000025
  ✓ New best model saved! (Val Loss: 0.0114, SSIM: 0.7577)

Epoch 23/25
--------------------------------------------------
Training Epoch 23: 100%|██████████| 100/100 [06:05<00:00,  3.66s/it, Loss=0.0277, Recon=0.0142, Feature=0.0563,
Validation Epoch 23: 100%|██████████| 25/25 [00:14<00:00,  1.70it/s, Val Loss=0.0109, SSIM=0.9583]

Epoch 23 Summary:
  Train Loss: 0.0253
  Val Loss: 0.0111
  Reconstruction Loss: 0.0135
  Feature Loss: 0.0493
  Edge Loss: 0.0099
  Validation SSIM: 0.7077 (70.77%)
  Learning Rate: 0.000025
```

```
Epoch 24/25
--------------------------------------------------
Training Epoch 24: 100%|██████████| 100/100 [06:00<00:00,  3.61s/it, Loss=0.0260, Recon=0.0139, Feature=0.0511,
Validation Epoch 24: 100%|██████████| 25/25 [00:14<00:00,  1.72it/s, Val Loss=0.0094, SSIM=0.9610]

Epoch 24 Summary:
  Train Loss: 0.0255
  Val Loss: 0.0106
  Reconstruction Loss: 0.0135
  Feature Loss: 0.0502
  Edge Loss: 0.0094
  Validation SSIM: 0.6917 (69.17%)
  Learning Rate: 0.000025

Epoch 25/25
--------------------------------------------------
Training Epoch 25: 100%|██████████| 100/100 [05:55<00:00,  3.56s/it, Loss=0.0210, Recon=0.0112, Feature=0.0413,
Validation Epoch 25: 100%|██████████| 25/25 [00:14<00:00,  1.72it/s, Val Loss=0.0099, SSIM=0.9631]
Epoch 25 Summary:
  Train Loss: 0.0247
  Val Loss: 0.0109
  Reconstruction Loss: 0.0131
  Feature Loss: 0.0488
  Edge Loss: 0.0090
  Validation SSIM: 0.7157 (71.57%)
  Learning Rate: 0.000025

Early stopping triggered after 25 epochs (patience: 8)

Training completed!
Best validation loss: 0.0101
Best SSIM score: 0.7577 (75.77%)
```

Downloading the best Model into Local Machine

```
# Download the best model file to your local machine
from google.colab import files

# Check if the model file exists and download it
import os

model_files = [
    'best_student_model.pth',
    'best_student_model_25epochs.pth'  # If you used 25 epochs
]

for model_file in model_files:
    if os.path.exists(model_file):
        print(f"Downloading {model_file}...")
        files.download(model_file)
        print(f"✓ {model_file} downloaded successfully!")
        break
else:
    print("❌ No model checkpoint found!")
    # List all .pth files in current directory
    pth_files = [f for f in os.listdir('.') if f.endswith('.pth')]
    if pth_files:
        print("Available .pth files:")
        for f in pth_files:
            print(f"  • {f}")
```

```
Downloading best_student_model_25epochs.pth...
✓ best_student_model_25epochs.pth downloaded successfully!
```

Phase 5: Complete Video Processing Pipeline Setup

```
# Import required libraries
import os
import cv2
import numpy as np
import base64
from IPython.display import display, HTML
from google.colab import files
from tqdm import tqdm
from skimage.metrics import structural_similarity as ssim
import matplotlib.pyplot as plt
import time

def upload_video_file():
    """Upload video file from local machine"""
    print("Please upload a video file...")
    print("Supported formats: .mp4, .avi, .mov, .mkv")

    uploaded = files.upload()

    if not uploaded:
        print("No file uploaded!")
        return None

    video_path = list(uploaded.keys())[0]
    print(f"✓ Video uploaded: {video_path}")
    return video_path

def extract_video_info(video_path):
    """Extract basic video information"""
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        return None
```

```
        fps = int(cap.get(cv2.CAP_PROP_FPS))
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        duration = frame_count / fps if fps > 0 else 0

        cap.release()

        return {
            'fps': fps,
            'width': width,
            'height': height,
            'frame_count': frame_count,
            'duration': duration
        }

    def simulate_network_blur(frame, blur_factor=4):
        """Simulate poor network conditions by downscaling and upscaling"""
        h, w = frame.shape[:2]
        small_frame = cv2.resize(frame, (w//blur_factor, h//blur_factor), interpolation=cv2.INTER_CUBIC)
        blurred_frame = cv2.resize(small_frame, (w, h), interpolation=cv2.INTER_CUBIC)
        return blurred_frame

    def enhance_frame_with_student(frame, student_model):
        """Enhance a single frame using the trained student model"""
        try:
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame_tensor = torch.from_numpy(frame_rgb).float() / 255.0
            frame_tensor = frame_tensor.permute(2, 0, 1).unsqueeze(0).to(device)

            student_model.eval()
            with torch.no_grad():
                enhanced_tensor = student_model(frame_tensor)

            enhanced_frame = enhanced_tensor.squeeze(0).permute(1, 2, 0).cpu().numpy()
            enhanced_frame = np.clip(enhanced_frame * 255, 0, 255).astype(np.uint8)
            enhanced_frame = cv2.cvtColor(enhanced_frame, cv2.COLOR_RGB2BGR)

            if enhanced_frame.shape[:2] != frame.shape[:2]:
                enhanced_frame = cv2.resize(enhanced_frame, (frame.shape[1], frame.shape[0]))

            return enhanced_frame

        except Exception as e:
            print(f"Enhancement failed: {e}")
            return frame

    def process_video_frames(video_path, student_model, max_frames=None):
        """Process video frames: extract, blur, enhance"""
        print(f"Processing video: {video_path}")

        video_info = extract_video_info(video_path)
        if video_info is None:
            print("✗ Failed to read video file")
            return None, None, None, None

        cap = cv2.VideoCapture(video_path)
        original_frames = []
        blurred_frames = []
        enhanced_frames = []
        frame_metrics = []

        frame_count = 0
        max_process = max_frames if max_frames else video_info['frame_count']

        pbar = tqdm(total=min(max_process, video_info['frame_count']), desc="Processing frames")

        while True:
            ret, frame = cap.read()
```

```
            if not ret or frame_count >= max_process:
                break

            original_frames.append(frame.copy())
            blurred_frame = simulate_network_blur(frame, blur_factor=4)
            blurred_frames.append(blurred_frame)

            start_time = time.time()
            enhanced_frame = enhance_frame_with_student(blurred_frame, student_model)
            processing_time = time.time() - start_time

            enhanced_frames.append(enhanced_frame)

            frame_ssim = ssim(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY),
                              cv2.cvtColor(enhanced_frame, cv2.COLOR_BGR2GRAY), data_range=255)

            frame_metrics.append({
                'frame_id': frame_count,
                'ssim': frame_ssim,
                'processing_time': processing_time,
                'fps': 1.0 / processing_time if processing_time > 0 else 0
            })

            frame_count += 1
            pbar.update(1)
            pbar.set_postfix({'SSIM': f'{frame_ssim:.3f}'})

        cap.release()
        pbar.close()

        print(f"✓ Processed {len(original_frames)} frames")
        return original_frames, blurred_frames, enhanced_frames, frame_metrics

    def create_output_videos(original_frames, blurred_frames, enhanced_frames, video_info, output_dir='output_videos'):
        """Create output videos from processed frames"""
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

        fps = video_info['fps']
        width, height = video_info['width'], video_info['height']
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')

        videos_created = []

        # Enhanced video
        enhanced_path = os.path.join(output_dir, 'enhanced_video.mp4')
        out_enhanced = cv2.VideoWriter(enhanced_path, fourcc, fps, (width, height))

        print("Creating enhanced video...")
        for frame in tqdm(enhanced_frames, desc="Enhanced video"):
            if frame.shape[:2] != (height, width):
                frame = cv2.resize(frame, (width, height))
            out_enhanced.write(frame)

        out_enhanced.release()
        videos_created.append(enhanced_path)
        print(f"✓ Enhanced video saved: {enhanced_path}")

        # Blurred video
        blurred_path = os.path.join(output_dir, 'blurred_video.mp4')
        out_blurred = cv2.VideoWriter(blurred_path, fourcc, fps, (width, height))

        print("Creating blurred video...")
        for frame in tqdm(blurred_frames, desc="Blurred video"):
            if frame.shape[:2] != (height, width):
                frame = cv2.resize(frame, (width, height))
            out_blurred.write(frame)

        out_blurred.release()
```

```
        videos_created.append(blurred_path)
        print(f"✓ Blurred video saved: {blurred_path}")

        return videos_created

    def create_side_by_side_comparison_video(original_frames, blurred_frames, enhanced_frames, video_info, output_path=
        """Create side-by-side comparison video with labels"""
        fps = video_info['fps']
        width, height = video_info['width'], video_info['height']
        comparison_width = width * 3
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(output_path, fourcc, fps, (comparison_width, height))

        print(f"Creating comparison video: {output_path}")

        for i, (orig, blur, enh) in enumerate(tqdm(zip(original_frames, blurred_frames, enhanced_frames),
                                                   desc="Creating comparison", total=len(original_frames))):

            orig_resized = cv2.resize(orig, (width, height))
            blur_resized = cv2.resize(blur, (width, height))
            enh_resized = cv2.resize(enh, (width, height))

            font = cv2.FONT_HERSHEY_SIMPLEX
            cv2.putText(orig_resized, 'GROUND TRUTH', (10, 40), font, 1.2, (0, 255, 0), 2)
            cv2.putText(blur_resized, 'BLURRED', (10, 40), font, 1.2, (0, 0, 255), 2)
            cv2.putText(enh_resized, 'ENHANCED', (10, 40), font, 1.2, (255, 0, 0), 2)

            # Add SSIM score
            frame_ssim = ssim(cv2.cvtColor(orig_resized, cv2.COLOR_BGR2GRAY),
                              cv2.cvtColor(enh_resized, cv2.COLOR_BGR2GRAY), data_range=255)
            cv2.putText(enh_resized, f'SSIM: {frame_ssim:.3f}', (10, height-60), font, 0.8, (255, 255, 0), 2)

            comparison_frame = np.hstack([orig_resized, blur_resized, enh_resized])
            cv2.line(comparison_frame, (width, 0), (width, height), (255, 255, 255), 2)
            cv2.line(comparison_frame, (width*2, 0), (width*2, height), (255, 255, 255), 2)

            out.write(comparison_frame)

        out.release()
        print(f"✓ Comparison video created: {output_path}")
        return output_path

    def display_video_in_colab(video_path, title="Video", width=640, height=480):
        """Display video in Google Colab using HTML5 video player"""
        try:
            with open(video_path, 'rb') as f:
                video_data = f.read()

            video_base64 = base64.b64encode(video_data).decode()

            video_html = f"""
            <div style="text-align: center; margin: 20px;">
                <h3>{title}</h3>
                <video width="{width}" height="{height}" controls>
                    <source src="data:video/mp4;base64,{video_base64}" type="video/mp4">
                    Your browser does not support the video tag.
                </video>
            </div>
            """

            display(HTML(video_html))
            return True

        except Exception as e:
            print(f"Error displaying video {video_path}: {e}")
            return False

    def download_videos_to_local(video_paths):
        """Download multiple videos to local machine"""
```

```
        print("Downloading videos to local machine...")

        for video_path in video_paths:
            if os.path.exists(video_path):
                print(f"Downloading: {os.path.basename(video_path)}")
                files.download(video_path)
            else:
                print(f"✗ File not found: {video_path}")

        print("✓ All available videos downloaded!")

    def display_performance_metrics_enhanced(frame_metrics, video_info):
        """Enhanced performance metrics with proper FPS labeling"""
        if not frame_metrics:
            print("No metrics available")
            return

        ssim_scores = [m['ssim'] for m in frame_metrics]
        processing_times = [m['processing_time'] for m in frame_metrics]

        avg_ssim = np.mean(ssim_scores)
        avg_processing_time = np.mean(processing_times)

        # Calculate FPS metrics with proper labeling
        fps_during_processing = 1.0 / avg_processing_time if avg_processing_time > 0 else 0
        fps_original_video = video_info['fps']

        # Get output video FPS
        fps_enhanced_video = fps_original_video  # Default fallback
        if os.path.exists('output_videos/enhanced_video.mp4'):
            cap = cv2.VideoCapture('output_videos/enhanced_video.mp4')
            if cap.isOpened():
                fps_enhanced_video = cap.get(cv2.CAP_PROP_FPS)
                cap.release()

        # Create comprehensive visualization
        fig, axes = plt.subplots(2, 3, figsize=(18, 12))

        # SSIM over time
        axes[0, 0].plot(ssim_scores, color='blue', linewidth=2)
        axes[0, 0].axhline(y=0.9, color='red', linestyle='--', label='Target (90%)')
        axes[0, 0].set_title(f'SSIM Over Time (Avg: {avg_ssim:.3f})')
        axes[0, 0].set_xlabel('Frame Number')
        axes[0, 0].set_ylabel('SSIM Score')
        axes[0, 0].legend()
        axes[0, 0].grid(True)

        # Enhanced FPS Comparison with proper labeling
        fps_categories = ['FPS During\nProcessing', 'Enhanced Video\nFPS', 'Original Video\nFPS']
        fps_values = [fps_during_processing, fps_enhanced_video, fps_original_video]
        fps_colors = ['orange', 'green', 'blue']

        bars = axes[0, 1].bar(fps_categories, fps_values, color=fps_colors, alpha=0.7)
        axes[0, 1].set_title('FPS Performance Comparison')
        axes[0, 1].set_ylabel('Frames Per Second (FPS)')
        axes[0, 1].grid(True, alpha=0.3)

        # Add value labels on bars
        for bar, value in zip(bars, fps_values):
            axes[0, 1].text(bar.get_x() + bar.get_width()/2, bar.get_height() + 1,
                            f'{value:.1f}', ha='center', va='bottom', fontweight='bold')

        # Add target lines
        axes[0, 1].axhline(y=30, color='red', linestyle='--', alpha=0.7, label='30 FPS Target')
        axes[0, 1].axhline(y=60, color='purple', linestyle='--', alpha=0.7, label='60 FPS Target')
        axes[0, 1].legend()

        # SSIM histogram
        axes[0, 2].hist(ssim_scores, bins=20, alpha=0.7, color='blue', edgecolor='black')
```

```
        axes[0, 2].axvline(x=0.9, color='red', linestyle='--', label='Target (90%)')
        axes[0, 2].set_title('SSIM Distribution')
        axes[0, 2].set_xlabel('SSIM Score')
        axes[0, 2].set_ylabel('Frequency')
        axes[0, 2].legend()
        axes[0, 2].grid(True)

        # Processing time histogram
        axes[1, 0].hist(processing_times, bins=20, alpha=0.7, color='green', edgecolor='black')
        axes[1, 0].set_title('Processing Time Distribution')
        axes[1, 0].set_xlabel('Processing Time (seconds)')
        axes[1, 0].set_ylabel('Frequency')
        axes[1, 0].grid(True)

        # Real-time capability pie chart
        real_time_capable = fps_during_processing >= fps_original_video
        axes[1, 1].pie([1 if real_time_capable else 0, 0 if real_time_capable else 1],
                       labels=['Real-time\nCapable', 'Not Real-time'],
                       colors=['green', 'red'],
                       autopct='%1.0f%%',
                       startangle=90)
        axes[1, 1].set_title('Real-time Processing Capability')

        # Enhanced Performance Summary
        axes[1, 2].axis('off')
        summary_text = f"""
        ENHANCED PERFORMANCE SUMMARY

        📊 Quality Metrics:
        • Average SSIM: {avg_ssim:.4f} ({avg_ssim*100:.1f}%)
        • SSIM Target (≥90%): {'✅ PASSED' if avg_ssim >= 0.9 else '❌ FAILED'}

        ⚡ Speed Metrics:
        • FPS During Processing: {fps_during_processing:.1f}
        • Enhanced Video FPS: {fps_enhanced_video:.1f}
        • Original Video FPS: {fps_original_video:.1f}
        • Real-time Capable: {'✅ YES' if real_time_capable else '❌ NO'}

        🎯 Target Achievement:
        • 30 FPS Processing: {'✅ PASSED' if fps_during_processing >= 30 else '❌ FAILED'}
        • 60 FPS Processing: {'✅ PASSED' if fps_during_processing >= 60 else '❌ FAILED'}

        📈 Frame Statistics:
        • Total Frames: {len(frame_metrics)}
        • Best SSIM: {max(ssim_scores):.4f}
        • Worst SSIM: {min(ssim_scores):.4f}
        """

        axes[1, 2].text(0.05, 0.95, summary_text, transform=axes[1, 2].transAxes,
                       fontsize=10, verticalalignment='top', fontfamily='monospace',
                       bbox=dict(boxstyle="round,pad=0.5", facecolor="lightblue", alpha=0.8))

        plt.tight_layout()
        plt.show()

        return {
            'avg_ssim': avg_ssim,
            'fps_during_processing': fps_during_processing,
            'fps_enhanced_video': fps_enhanced_video,
            'fps_original_video': fps_original_video,
            'target_ssim_passed': avg_ssim >= 0.9,
            'target_fps_30_passed': fps_during_processing >= 30,
            'real_time_capable': real_time_capable
        }

    def complete_video_processing_pipeline():
        """Complete video processing pipeline with enhanced FPS reporting"""

        print("="*60)
```

```
print("PHASE 5: COMPLETE VIDEO PROCESSING PIPELINE")
print("="*60)

# Step 1: Upload video
print("\n🎬 STEP 1: Upload your video file")
video_path = upload_video_file()

if not video_path:
    print("❌ Workflow stopped - no video uploaded")
    return

# Get video info
video_info = extract_video_info(video_path)
print(f"✓ Original Video FPS: {video_info['fps']}")
print(f"✓ Video Resolution: {video_info['width']}x{video_info['height']}")
print(f"✓ Total Frames: {video_info['frame_count']}")

# Step 2: Process frames
print("\n🔄 STEP 2: Processing video frames")
max_frames = min(100, video_info['frame_count'])
print(f"Processing first {max_frames} frames...")

original_frames, blurred_frames, enhanced_frames, frame_metrics = process_video_frames(
    video_path, student_model, max_frames=max_frames
)

if not original_frames:
    print("❌ Workflow stopped - frame processing failed")
    return

# Step 3: Create videos
print("\n📹 STEP 3: Creating output videos")
output_dir = 'output_videos'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

video_paths = create_output_videos(original_frames, blurred_frames, enhanced_frames, video_info, output_dir)
comparison_path = create_side_by_side_comparison_video(
    original_frames, blurred_frames, enhanced_frames, video_info,
    os.path.join(output_dir, 'comparison_video.mp4')
)
video_paths.append(comparison_path)

# Step 4: Display videos
print("\n📺 STEP 4: Displaying videos in Colab")
video_titles = {
    'enhanced_video.mp4': 'Enhanced Video (Student Model)',
    'blurred_video.mp4': 'Blurred Video (Simulated Poor Network)',
    'comparison_video.mp4': 'Side-by-Side Comparison'
}

for video_path in video_paths:
    if os.path.exists(video_path):
        filename = os.path.basename(video_path)
        title = video_titles.get(filename, filename.replace('.mp4', '').replace('_', ' ').title())
        display_video_in_colab(video_path, title=title, width=800, height=400)

# Step 5: Enhanced performance analysis
print("\n📊 STEP 5: Enhanced Performance Analysis")
performance_summary = display_performance_metrics_enhanced(frame_metrics, video_info)

# Step 6: Download videos
print("\n💾 STEP 6: Downloading videos")
download_videos_to_local(video_paths)

print("\n" + "="*60)
print("🎉 PHASE 5 COMPLETE!")
print("="*60)
print(f"✅ Processed {len(original_frames)} frames")
```

```
        print(f"✅ Original Video FPS: {performance_summary['fps_original_video']:.1f}")
        print(f"✅ FPS During Processing: {performance_summary['fps_during_processing']:.1f}")
        print(f"✅ Enhanced Video FPS: {performance_summary['fps_enhanced_video']:.1f}")
        print(f"✅ Average SSIM: {performance_summary['avg_ssim']:.4f}")
        print(f"✅ Real-time Capable: {'YES' if performance_summary['real_time_capable'] else 'NO'}")

        return video_paths, performance_summary

print("✅ All Phase 5 functions defined successfully!")
print("Ready to run the complete video processing pipeline!")
```

```
✅ All Phase 5 functions defined successfully!
Ready to run the complete video processing pipeline!
```

Execute Phase 5

```
# Execute Phase 5: Complete Video Processing Pipeline
video_paths, performance = complete_video_processing_pipeline()
```

Session Crashed after using all available RAM

Loading Your Saved Model in a New Colab Session

Phase 1: Setup and Dependencies

```
# Install all required dependencies
!pip install torch torchvision torchaudio
!pip install opencv-python-headless
!pip install pillow
!pip install scikit-image
!pip install tqdm
!pip install ipywidgets

# Fix for PyTorch 2.6+ compatibility
!pip install huggingface_hub==0.25.2

# Import all required libraries
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
import cv2
import numpy as np
from PIL import Image
import os
import zipfile
import urllib.request
from tqdm import tqdm
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim
import json
import time
from google.colab import files
import shutil
import glob
import base64
from IPython.display import display, HTML
import ipywidgets as widgets

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
print(f"Using device: {device}")
print(f"CUDA available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-packages (0.21.0+cu124)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torc
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from to
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from t
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from to
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from tor
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from torchvision) (2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/dist-packages (from torchvis
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch)
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python-he
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (11.2.1)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.11/dist-packages (0.25.2)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2.0
Requirement already satisfied: scipy>=1.11.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (1
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (3
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (11
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.11/dist-packages (from scikit-
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.11/dist-packages (from scikit-ima
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (4.67.1)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.11/dist-packages (7.7.1)
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (
Requirement already satisfied: ipython-genutils~=0.2.0 in /usr/local/lib/python3.11/dist-packages (from ipywid
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.11/dist-packages (from ipyw
Requirement already satisfied: ipython>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from ipywidgets) (7.
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from ipyw
Requirement already satisfied: debugpy>=1.0 in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1-
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.11/dist-packages (from ipykerr
Requirement already satisfied: matplotlib-inline>=0.1 in /usr/local/lib/python3.11/dist-packages (from ipykerr
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1-
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ip
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ipywi
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1->ip
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.11/dist-packages (from ipykernel>=4.5.1-
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.0->ipy
Requirement already satisfied: decorator in /usr/local/lib/python3.11/dist-packages (from ipython>=4.0.0->ipyw
```

Phase 2: Load Pre-trained Student Model

```
# Student Model Architecture (same as training)
class StudentModel(nn.Module):
    def __init__(self, scale_factor=4):
        super(StudentModel, self).__init__()
        self.scale_factor = scale_factor
```

```
            # Feature extraction layers
            self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
            self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
            self.conv3 = nn.Conv2d(64, 64, kernel_size=3, padding=1)

            # Residual blocks
            self.res_blocks = nn.ModuleList([
                self._make_residual_block(64) for _ in range(4)
            ])

            # Upsampling layers
            self.upsample = nn.Sequential(
                nn.Conv2d(64, 64 * (scale_factor ** 2), kernel_size=3, padding=1),
                nn.PixelShuffle(scale_factor),
                nn.Conv2d(64, 3, kernel_size=3, padding=1)
            )

            self.relu = nn.ReLU(inplace=True)
            self.tanh = nn.Tanh()

        def _make_residual_block(self, channels):
            return nn.Sequential(
                nn.Conv2d(channels, channels, kernel_size=3, padding=1),
                nn.ReLU(inplace=True),
                nn.Conv2d(channels, channels, kernel_size=3, padding=1)
            )

        def forward(self, x):
            # Feature extraction
            x1 = self.relu(self.conv1(x))
            x2 = self.relu(self.conv2(x1))
            x3 = self.relu(self.conv3(x2))

            # Residual blocks
            residual = x3
            for res_block in self.res_blocks:
                out = res_block(residual)
                residual = residual + out

            # Upsampling
            out = self.upsample(residual)
            out = self.tanh(out)

            return out

    # Initialize student model
    student_model = StudentModel(scale_factor=4).to(device)

    # Load your pre-trained model
    def load_pretrained_student_model():
        """Load the pre-trained student model from checkpoint"""

        # Option 1: If model file exists in current directory
        checkpoint_path = 'best_student_model.pth'

        if os.path.exists(checkpoint_path):
            try:
                checkpoint = torch.load(checkpoint_path, map_location=device, weights_only=False)
                student_model.load_state_dict(checkpoint['model_state_dict'])
                student_model.eval()
                print(f"✅ Loaded pre-trained model from {checkpoint_path}")
                print(f"   Training epoch: {checkpoint.get('epoch', 'unknown')}")
                print(f"   SSIM score: {checkpoint.get('ssim', 'unknown'):.4f}")
                return True
            except Exception as e:
                print(f"❌ Error loading model: {e}")
                return False
        else:
            # Option 2: Upload model file
```

```
        print("Model file not found. Please upload your trained model:")
        uploaded = files.upload()

        if uploaded:
            model_file = list(uploaded.keys())[0]
            try:
                checkpoint = torch.load(model_file, map_location=device, weights_only=False)
                student_model.load_state_dict(checkpoint['model_state_dict'])
                student_model.eval()
                print(f"✅ Loaded pre-trained model from {model_file}")
                print(f"   Training epoch: {checkpoint.get('epoch', 'unknown')}")
                print(f"   SSIM score: {checkpoint.get('ssim', 'unknown'):.4f}")
                return True
            except Exception as e:
                print(f"❌ Error loading uploaded model: {e}")
                return False
        else:
            print("❌ No model file uploaded")
            return False

# Load the pre-trained model
model_loaded = load_pretrained_student_model()

if model_loaded:
    print(f"✅ Student model parameters: {sum(p.numel() for p in student_model.parameters()):,}")
    print("✅ Pre-trained model ready for video processing!")
else:
    print("❌ Failed to load pre-trained model. Please check your model file.")
```

```
Model file not found. Please upload your trained model:
Choose Files best_stude…5epochs.pth
• best_student_model_25epochs.pth(n/a) - 11366482 bytes, last modified: 7/9/2025 - 100% done
Saving best_student_model_25epochs.pth to best_student_model_25epochs (3).pth
✅ Loaded pre-trained model from best_student_model_25epochs (3).pth
   Training epoch: 21
   SSIM score: 0.7577
✅ Student model parameters: 944,323
✅ Pre-trained model ready for video processing!
```

Phase 3: Complete Video Processing Pipeline

```
# Video processing functions with all requested features

def upload_video_file():
    """Upload video file from local machine"""
    print("Please upload a video file...")
    print("Supported formats: .mp4, .avi, .mov, .mkv")

    uploaded = files.upload()

    if not uploaded:
        print("No file uploaded!")
        return None

    video_path = list(uploaded.keys())[0]
    print(f"✓ Video uploaded: {video_path}")
    return video_path

def extract_video_info(video_path):
    """Extract basic video information"""
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        return None

    fps = int(cap.get(cv2.CAP_PROP_FPS))
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
```

```
        duration = frame_count / fps if fps > 0 else 0

        cap.release()

        return {
            'fps': fps,
            'width': width,
            'height': height,
            'frame_count': frame_count,
            'duration': duration
        }

    def simulate_network_blur(frame, blur_factor=4):
        """Simulate poor network conditions by downscaling and upscaling"""
        h, w = frame.shape[:2]
        small_frame = cv2.resize(frame, (w//blur_factor, h//blur_factor), interpolation=cv2.INTER_CUBIC)
        blurred_frame = cv2.resize(small_frame, (w, h), interpolation=cv2.INTER_CUBIC)
        return blurred_frame

    def enhance_frame_with_student(frame, student_model):
        """Enhance a single frame using the trained student model"""
        try:
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame_tensor = torch.from_numpy(frame_rgb).float() / 255.0
            frame_tensor = frame_tensor.permute(2, 0, 1).unsqueeze(0).to(device)

            student_model.eval()
            with torch.no_grad():
                enhanced_tensor = student_model(frame_tensor)

            enhanced_frame = enhanced_tensor.squeeze(0).permute(1, 2, 0).cpu().numpy()
            enhanced_frame = np.clip(enhanced_frame * 255, 0, 255).astype(np.uint8)
            enhanced_frame = cv2.cvtColor(enhanced_frame, cv2.COLOR_RGB2BGR)

            if enhanced_frame.shape[:2] != frame.shape[:2]:
                enhanced_frame = cv2.resize(enhanced_frame, (frame.shape[1], frame.shape[0]))

            return enhanced_frame

        except Exception as e:
            print(f"Enhancement failed: {e}")
            return frame

    def process_video_frames(video_path, student_model, max_frames=None):
        """Process video frames: extract, blur, enhance"""
        print(f"Processing video: {video_path}")

        video_info = extract_video_info(video_path)
        if video_info is None:
            print("✗ Failed to read video file")
            return None, None, None, None

        cap = cv2.VideoCapture(video_path)
        original_frames = []
        blurred_frames = []
        enhanced_frames = []
        frame_metrics = []

        frame_count = 0
        max_process = max_frames if max_frames else video_info['frame_count']

        pbar = tqdm(total=min(max_process, video_info['frame_count']), desc="Processing frames")

        while True:
            ret, frame = cap.read()
            if not ret or frame_count >= max_process:
                break

            original_frames.append(frame.copy())
```