Introduction to Knowledge Distillation

Knowledge Distillation is a procedure for model compression, in which a small (student) model is trained to match a large pre-trained (teacher) model. Knowledge is transferred from the teacher model to the student by minimizing a loss function, aimed at matching softened teacher logits as well as ground-truth labels.

The logits are softened by applying a "temperature" scaling function in the softmax, effectively smoothing out the probability distribution and revealing inter-class relationships learned by the teacher.

Overview

Key Concepts in This Implementation

- Teacher Model: A more complex network that learns to sharpen images effectively
- Student Model: A simpler network that learns to mimic the teacher's behavior
- Alpha Parameter: Balances between ground truth and teacher supervision
- KL divergence loss between student and teacher outputs
- Knowledge Distillation Loss: Combines.
- Traditional MSE loss between student output and ground truth
- Temperature Parameter: Controls how much we soften the probability distributions

```
!pip install opencv-python matplotlib tensorflow

import numpy as np
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tensorflow.keras import layers, models

```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python) (2.0.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.14.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.73.1)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.14.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
```

```
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0
```

```python
# Step 1: Setup
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os
import requests
from io import BytesIO
from google.colab import files

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Step 2: Dataset
class SharpeningDataset(Dataset):
    def __init__(self, image_paths, blur_radius=3, transform=None):
        self.image_paths = image_paths
        self.blur_radius = blur_radius
        self.transform = transform

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

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        if img_path.startswith("http"):
            response = requests.get(img_path)
            sharp_img = Image.open(BytesIO(response.content)).convert("RGB")
        else:
            sharp_img = Image.open(img_path).convert("RGB")

        sharp_np = np.array(sharp_img)
        blurred_np = cv2.GaussianBlur(sharp_np, (self.blur_radius, self.
        blur_radius), 0)
        blurred_img = Image.fromarray(blurred_np)

        if self.transform:
            sharp_img = self.transform(sharp_img)
            blurred_img = self.transform(blurred_img)

        return blurred_img, sharp_img

# Transform
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Upload images
print("Upload images for training (sharp ground truths):")
uploaded = files.upload()
image_paths = list(uploaded.keys())

dataset = SharpeningDataset(image_paths, transform=transform)
dataloader = DataLoader(dataset, batch_size=8, shuffle=True)

# Step 3: Models
class TeacherModel(nn.Module):
    def __init__(self):
        super(TeacherModel, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(2),
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2), nn.ReLU(),
```

```
            nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2), nn.ReLU(),
            nn.Conv2d(64, 3, kernel_size=3, padding=1),
        )

    def forward(self, x):
        return self.decoder(self.encoder(x))

class StudentModel(nn.Module):
    def __init__(self):
        super(StudentModel, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(32, 3, kernel_size=3, padding=1),
        )

    def forward(self, x):
        return self.net(x)

teacher = TeacherModel().to(device)
student = StudentModel().to(device)

# Step 4: Knowledge Distillation Training
def train_student_with_distillation(epochs=10, alpha=0.7, temperature=2.0):
    criterion_mse = nn.MSELoss()
    criterion_kl = nn.KLDivLoss(reduction='batchmean')
    optimizer = optim.Adam(student.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
    teacher.eval()

    for epoch in range(epochs):
        for blurred, sharp in dataloader:
            blurred, sharp = blurred.to(device), sharp.to(device)

            with torch.no_grad():
                teacher_logits = teacher(blurred)

            student_logits = student(blurred)

            loss_mse = criterion_mse(student_logits, sharp)

            T = temperature
            soft_teacher = nn.functional.softmax(teacher_logits / T, dim=1)
            soft_student = nn.functional.log_softmax(student_logits / T, dim=1)

            B, C, H, W = student_logits.shape
            soft_teacher = soft_teacher.permute(0, 2, 3, 1).reshape(-1, C)
            soft_student = soft_student.permute(0, 2, 3, 1).reshape(-1, C)

            loss_kl = criterion_kl(soft_student, soft_teacher) * (T ** 2)

            loss = alpha * loss_mse + (1 - alpha) * loss_kl
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        scheduler.step()
        print(f"Epoch {epoch+1}/{epochs} | Total Loss: {loss.item():.4f} | MSE:
        {loss_mse.item():.4f} | KL: {loss_kl.item():.4f}")

# Step 5: Visualization
def denormalize(tensor):
    mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1).to(device)
    std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1).to(device)
    return torch.clamp(tensor * std + mean, 0, 1)

def visualize_results(num_images=3):
    teacher.eval()
    student.eval()
    with torch.no_grad():
        for i, (blurred, sharp) in enumerate(dataloader):
            if i >= num_images:
                break
            blurred, sharp = blurred.to(device), sharp.to(device)
            out_student = student(blurred)
            out_teacher = teacher(blurred)

            # Plot
            fig, axes = plt.subplots(1, 4, figsize=(16, 4))
            images = [blurred[0], out_student[0], out_teacher[0], sharp[0]]
            titles = ['Blurred Input', 'Student Output', 'Teacher Output',
```

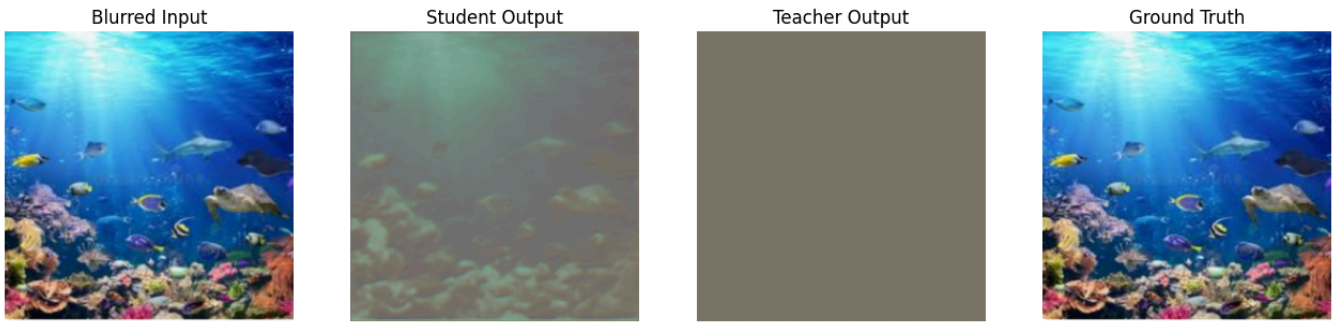
```
                Ground Truth"]

            for ax, img, title in zip(axes, images, titles):
                ax.imshow(denormalize(img).permute(1, 2, 0).cpu().numpy())
                ax.set_title(title)
                ax.axis('off')
            plt.show()

# Step 6: Run
train_student_with_distillation(epochs=10)
visualize_results()

# Save models
torch.save(teacher.state_dict(), "teacher_blur2sharp.pth")
torch.save(student.state_dict(), "student_blur2sharp.pth")
```

```
Using device: cpu
Upload images for training (sharp ground truths):
```

Choose Files WhatsApp I...052215f.jpg

- **WhatsApp Image 2025-07-09 at 09.27.24_e052215f.jpg**(image/jpeg) - 70078 bytes, last modified: 9/7/2025 - 100% done

```
Saving WhatsApp Image 2025-07-09 at 09.27.24_e052215f.jpg to WhatsApp Image 2025-07-09 at 09.27.24_e052215f (5).jpg
Epoch 1/10 | Total Loss: 1.6446 | MSE: 2.3388 | KL: 0.0248
Epoch 2/10 | Total Loss: 1.4429 | MSE: 2.0575 | KL: 0.0088
Epoch 3/10 | Total Loss: 1.2862 | MSE: 1.8363 | KL: 0.0027
Epoch 4/10 | Total Loss: 1.1561 | MSE: 1.6505 | KL: 0.0025
Epoch 5/10 | Total Loss: 1.0444 | MSE: 1.4893 | KL: 0.0065
Epoch 6/10 | Total Loss: 0.9389 | MSE: 1.3351 | KL: 0.0145
Epoch 7/10 | Total Loss: 0.9283 | MSE: 1.3194 | KL: 0.0156
Epoch 8/10 | Total Loss: 0.9174 | MSE: 1.3033 | KL: 0.0167
Epoch 9/10 | Total Loss: 0.9062 | MSE: 1.2869 | KL: 0.0180
Epoch 10/10 | Total Loss: 0.8948 | MSE: 1.2701 | KL: 0.0193
```




```
# Step 1: Set up environment and imports
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os
from google.colab import files
from io import BytesIO

# Check for GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Step 2: Data Preparation
class SharpeningDataset(Dataset):
    def __init__(self, image_paths=None, blur_radius=3, transform=None):
        self.image_paths = image_paths or []
        self.blur_radius = blur_radius
        self.transform = transform

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

    def __getitem__(self, idx):
        # Load image (works with both local files and URLs)
        if isinstance(self.image_paths[idx], str) and self.image_paths[idx].startswith('http'):
            response = requests.get(self.image_paths[idx])
            sharp_img = Image.open(BytesIO(response.content)).convert('RGB')
        else:
```

```
            sharp_img = Image.open(self.image_paths[idx]).convert('RGB')

        # Create blurred version
        sharp_np = np.array(sharp_img)
        blurred_np = cv2.GaussianBlur(sharp_np, (self.blur_radius, self.blur_radius), 0)
        blurred_img = Image.fromarray(blurred_np)

        if self.transform:
            sharp_img = self.transform(sharp_img)
            blurred_img = self.transform(blurred_img)

        return blurred_img, sharp_img

# Define transforms
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Load data (manual upload in Colab)
print("Please upload your images:")
uploaded = files.upload()
image_paths = list(uploaded.keys())

if not image_paths:
    raise ValueError("No images uploaded! Please try again.")

dataset = SharpeningDataset(image_paths, transform=transform)
dataloader = DataLoader(dataset, batch_size=8, shuffle=True)

# Step 3: Model Definitions
class TeacherModel(nn.Module):
    def __init__(self):
        super(TeacherModel, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 3, kernel_size=3, padding=1),
        )

    def forward(self, x):
        return self.decoder(self.encoder(x))

class StudentModel(nn.Module):
    def __init__(self):
        super(StudentModel, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 3, kernel_size=3, padding=1),
        )

    def forward(self, x):
        return self.net(x)

teacher = TeacherModel().to(device)
student = StudentModel().to(device)

# Step 4: Training with Knowledge Distillation
def train_student_with_distillation(epochs=10, temperature=2.0, alpha=0.7):
    criterion_mse = nn.MSELoss()
    criterion_kl = nn.KLDivLoss(reduction='batchmean')
    optimizer = optim.Adam(student.parameters(), lr=0.001)
```

```
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

    teacher.eval()

    for epoch in range(epochs):
        for blurred, sharp in dataloader:
            blurred, sharp = blurred.to(device), sharp.to(device)

            with torch.no_grad():
                teacher_logits = teacher(blurred)

            optimizer.zero_grad()
            student_logits = student(blurred)

            # MSE loss
            loss_mse = criterion_mse(student_logits, sharp)

            # KL divergence loss
            T = temperature
            soft_teacher = nn.functional.softmax(teacher_logits / T, dim=1)
            soft_student = nn.functional.log_softmax(student_logits / T, dim=1)

            # Reshape for KLDivLoss
            B, C, H, W = student_logits.shape
            soft_teacher = soft_teacher.permute(0, 2, 3, 1).reshape(-1, C)
            soft_student = soft_student.permute(0, 2, 3, 1).reshape(-1, C)

            loss_kl = criterion_kl(soft_student, soft_teacher) * (T ** 2)

            # Combined loss
            loss = alpha * loss_mse + (1 - alpha) * loss_kl
            loss.backward()
            optimizer.step()

        scheduler.step()
        print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f} (MSE: {loss_mse.item():.4f}, KL: {loss_kl.item():.4f})")

# Step 5: Evaluation and Visualization
def denormalize(tensor):
    mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1).to(device)
    std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1).to(device)
    return torch.clamp(tensor * std + mean, 0, 1)

def visualize_results(num_images=3):
    student.eval()
    teacher.eval()
    with torch.no_grad():
        for i, (blurred, sharp) in enumerate(dataloader):
            if i >= num_images:
                break

            blurred, sharp = blurred.to(device), sharp.to(device)
            student_output = student(blurred)
            teacher_output = teacher(blurred)

            # Denormalize images
            blurred_img = denormalize(blurred[0]).cpu().permute(1, 2, 0).numpy()
            sharp_img = denormalize(sharp[0]).cpu().permute(1, 2, 0).numpy()
            student_img = denormalize(student_output[0]).cpu().permute(1, 2, 0).numpy()
            teacher_img = denormalize(teacher_output[0]).cpu().permute(1, 2, 0).numpy()

            # Plot comparison
            plt.figure(figsize=(20, 5))
            titles = ['Blurred Input', 'Student Output', 'Teacher Output', 'Ground Truth']
            images = [blurred_img, student_img, teacher_img, sharp_img]

            for j in range(4):
                plt.subplot(1, 4, j+1)
                plt.imshow(images[j])
                plt.title(titles[j])
                plt.axis('off')

            plt.show()

# Step 6: Run Training and Evaluation
if __name__ == "__main__":
    # Train teacher first (optional)
    # train_teacher()  # You would need to implement this

    # Train student with distillation
    train_student_with_distillation(epochs=10)
```

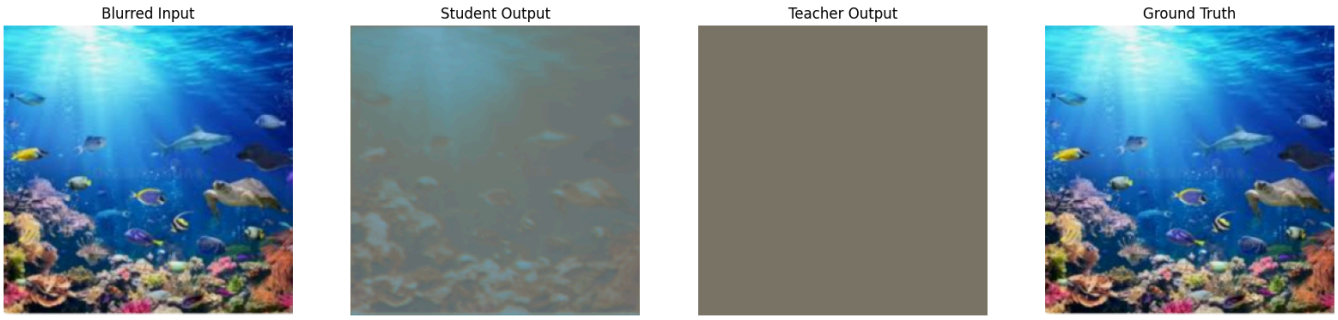
```
    # Save models
    torch.save(teacher.state_dict(), "teacher.pth")
    torch.save(student.state_dict(), "student.pth")

    # Visualize results
    visualize_results()
```

```
Using device: cpu
Please upload your images:
```

Choose Files WhatsApp I…052215f.jpg

- **WhatsApp Image 2025-07-09 at 09.27.24_e052215f.jpg**(image/jpeg) - 70078 bytes, last modified: 9/7/2025 - 100% done

```
Saving WhatsApp Image 2025-07-09 at 09.27.24_e052215f.jpg to WhatsApp Image 2025-07-09 at 09.27.24_e052215f (4).jpg
Epoch 1/10, Loss: 1.2454 (MSE: 1.7784, KL: 0.0018)
Epoch 2/10, Loss: 1.1671 (MSE: 1.6659, KL: 0.0033)
Epoch 3/10, Loss: 1.0953 (MSE: 1.5618, KL: 0.0068)
Epoch 4/10, Loss: 1.0215 (MSE: 1.4540, KL: 0.0124)
Epoch 5/10, Loss: 0.9425 (MSE: 1.3375, KL: 0.0210)
Epoch 6/10, Loss: 0.8569 (MSE: 1.2095, KL: 0.0340)
Epoch 7/10, Loss: 0.8479 (MSE: 1.1960, KL: 0.0356)
Epoch 8/10, Loss: 0.8386 (MSE: 1.1820, KL: 0.0373)
Epoch 9/10, Loss: 0.8291 (MSE: 1.1677, KL: 0.0391)
Epoch 10/10, Loss: 0.8195 (MSE: 1.1532, KL: 0.0410)
```




```
# Shared
import numpy as np
import matplotlib.pyplot as plt

# PyTorch (Image sharpening)
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
import cv2
from PIL import Image
from google.colab import files
import os

# Keras (MNIST classification)
import keras
from keras import layers
from keras import ops
```

```
# Combined Knowledge Distillation Implementation with PyTorch and Keras

import os
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO
import requests
from google.colab import files

```

```
# Keras imports
import keras
from keras import layers
from keras import ops

# Check for GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# ================================================================
# PyTorch Implementation - Image Sharpening with Knowledge Distillation
# ================================================================

class SharpeningDataset(Dataset):
    def __init__(self, image_paths=None, blur_radius=3, transform=None):
        self.image_paths = image_paths or []
        self.blur_radius = blur_radius
        self.transform = transform

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

    def __getitem__(self, idx):
        # Load image (works with both local files and URLs)
        if isinstance(self.image_paths[idx], str) and self.image_paths[idx].startswith('http'):
            response = requests.get(self.image_paths[idx])
            sharp_img = Image.open(BytesIO(response.content)).convert('RGB')
        else:
            sharp_img = Image.open(self.image_paths[idx]).convert('RGB')

        # Create blurred version
        sharp_np = np.array(sharp_img)
        blurred_np = cv2.GaussianBlur(sharp_np, (self.blur_radius, self.blur_radius), 0)
        blurred_img = Image.fromarray(blurred_np)

        if self.transform:
            sharp_img = self.transform(sharp_img)
            blurred_img = self.transform(blurred_img)

        return blurred_img, sharp_img

class TeacherModel(nn.Module):
    def __init__(self):
        super(TeacherModel, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 3, kernel_size=3, padding=1),
        )

    def forward(self, x):
        return self.decoder(self.encoder(x))

class StudentModel(nn.Module):
    def __init__(self):
        super(StudentModel, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 3, kernel_size=3, padding=1),
        )

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

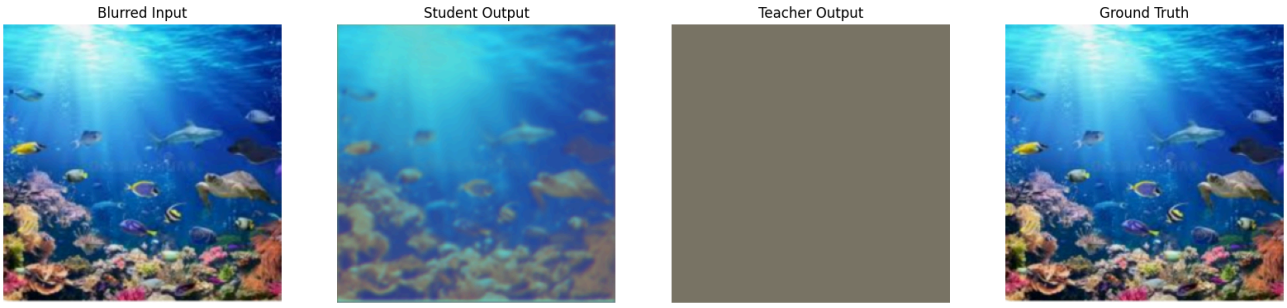
```
 99
100 def pytorch_knowledge_distillation():
101     # Define transforms
102     transform = transforms.Compose([
103         transforms.Resize((256, 256)),
104         transforms.ToTensor(),
105         transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
106     ])
107
108     # Load data (manual upload in Colab)
109     print("Please upload your images:")
110     uploaded = files.upload()
111     image_paths = list(uploaded.keys())
112
113     if not image_paths:
114         raise ValueError("No images uploaded! Please try again.")
115
116     dataset = SharpeningDataset(image_paths, transform=transform)
117     dataloader = DataLoader(dataset, batch_size=8, shuffle=True)
118
119     # Initialize models
120     teacher = TeacherModel().to(device)
121     student = StudentModel().to(device)
122
123     # Training function
124     def train_student_with_distillation(epochs=10, temperature=2.0, alpha=0.7):
125         criterion_mse = nn.MSELoss()
126         criterion_kl = nn.KLDivLoss(reduction='batchmean')
127         optimizer = optim.Adam(student.parameters(), lr=0.001)
128         scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
129
130         teacher.eval()
131
132         for epoch in range(epochs):
133             for blurred, sharp in dataloader:
134                 blurred, sharp = blurred.to(device), sharp.to(device)
135
136                 with torch.no_grad():
137                     teacher_logits = teacher(blurred)
138
139                 optimizer.zero_grad()
140                 student_logits = student(blurred)
141
142                 # MSE loss
143                 loss_mse = criterion_mse(student_logits, sharp)
144
145                 # KL divergence loss
146                 T = temperature
147                 soft_teacher = nn.functional.softmax(teacher_logits / T, dim=1)
148                 soft_student = nn.functional.log_softmax(student_logits / T, dim=1)
149
150                 # Reshape for KLDivLoss
151                 B, C, H, W = student_logits.shape
152                 soft_teacher = soft_teacher.permute(0, 2, 3, 1).reshape(-1, C)
153                 soft_student = soft_student.permute(0, 2, 3, 1).reshape(-1, C)
154
155                 loss_kl = criterion_kl(soft_student, soft_teacher) * (T ** 2)
156
157                 # Combined loss
158                 loss = alpha * loss_mse + (1 - alpha) * loss_kl
159                 loss.backward()
160                 optimizer.step()
161
162             scheduler.step()
163             print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f} (MSE: {loss_mse.item():.4f}, KL: {loss_kl.item():.4f})")
164
165     # Run training
166     train_student_with_distillation(epochs=10)
167
168     # Save models
169     torch.save(teacher.state_dict(), "teacher.pth")
170     torch.save(student.state_dict(), "student.pth")
171
172     # Visualization function
173     def denormalize(tensor):
174         mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1).to(device)
175         std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1).to(device)
176         return torch.clamp(tensor * std + mean, 0, 1)
177
178     def visualize_results(num_images=3):
179         student.eval()
180         teacher.eval()
```

```
        with torch.no_grad():
            for i, (blurred, sharp) in enumerate(dataloader):
                if i >= num_images:
                    break

                blurred, sharp = blurred.to(device), sharp.to(device)
                student_output = student(blurred)
                teacher_output = teacher(blurred)

                # Denormalize images
                blurred_img = denormalize(blurred[0]).cpu().permute(1, 2, 0).numpy()
                sharp_img = denormalize(sharp[0]).cpu().permute(1, 2, 0).numpy()
                student_img = denormalize(student_output[0]).cpu().permute(1, 2, 0).numpy()
                teacher_img = denormalize(teacher_output[0]).cpu().permute(1, 2, 0).numpy()

                # Plot comparison
                plt.figure(figsize=(20, 5))
                titles = ['Blurred Input', 'Student Output', 'Teacher Output', 'Ground Truth']
                images = [blurred_img, student_img, teacher_img, sharp_img]

                for j in range(4):
                    plt.subplot(1, 4, j+1)
                    plt.imshow(images[j])
                    plt.title(titles[j])
                    plt.axis('off')

                plt.show()

    visualize_results()

# ================================================================
# Keras Implementation - MNIST Classification with Knowledge Distillation
# ================================================================

class Distiller(keras.Model):
    def __init__(self, student, teacher):
        super().__init__()
        self.teacher = teacher
        self.student = student

    def compile(
        self,
        optimizer,
        metrics,
        student_loss_fn,
        distillation_loss_fn,
        alpha=0.1,
        temperature=3,
    ):
        super().compile(optimizer=optimizer, metrics=metrics)
        self.student_loss_fn = student_loss_fn
        self.distillation_loss_fn = distillation_loss_fn
        self.alpha = alpha
        self.temperature = temperature

    def compute_loss(
        self, x=None, y=None, y_pred=None, sample_weight=None, allow_empty=False
    ):
        teacher_pred = self.teacher(x, training=False)
        student_loss = self.student_loss_fn(y, y_pred)

        distillation_loss = self.distillation_loss_fn(
            ops.softmax(teacher_pred / self.temperature, axis=1),
            ops.softmax(y_pred / self.temperature, axis=1),
        ) * (self.temperature**2)

        loss = self.alpha * student_loss + (1 - self.alpha) * distillation_loss
        return loss

    def call(self, x):
        return self.student(x)

def keras_knowledge_distillation():
    # Create the teacher
    teacher = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(256, (3, 3), strides=(2, 2), padding="same"),
            layers.LeakyReLU(negative_slope=0.2),
            layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
            layers.Conv2D(512, (3, 3), strides=(2, 2), padding="same"),
            layers.Flatten(),
```

```
            layers.Dense(10),
        ],
        name="teacher",
    )

    # Create the student
    student = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(16, (3, 3), strides=(2, 2), padding="same"),
            layers.LeakyReLU(negative_slope=0.2),
            layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
            layers.Conv2D(32, (3, 3), strides=(2, 2), padding="same"),
            layers.Flatten(),
            layers.Dense(10),
        ],
        name="student",
    )

    # Clone student for later comparison
    student_scratch = keras.models.clone_model(student)

    # Prepare the dataset
    batch_size = 64
    (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

    # Normalize data
    x_train = x_train.astype("float32") / 255.0
    x_train = np.reshape(x_train, (-1, 28, 28, 1))

    x_test = x_test.astype("float32") / 255.0
    x_test = np.reshape(x_test, (-1, 28, 28, 1))

    # Train teacher
    print("\nTraining teacher model...")
    teacher.compile(
        optimizer=keras.optimizers.Adam(),
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[keras.metrics.SparseCategoricalAccuracy()],
    )
    teacher.fit(x_train, y_train, epochs=5)
    teacher.evaluate(x_test, y_test)

    # Distill teacher to student
    print("\nDistilling knowledge to student model...")
    distiller = Distiller(student=student, teacher=teacher)
    distiller.compile(
        optimizer=keras.optimizers.Adam(),
        metrics=[keras.metrics.SparseCategoricalAccuracy()],
        student_loss_fn=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        distillation_loss_fn=keras.losses.KLDivergence(),
        alpha=0.1,
        temperature=10,
    )
    distiller.fit(x_train, y_train, epochs=3)
    distiller.evaluate(x_test, y_test)

    # Train student from scratch for comparison
    print("\nTraining student from scratch for comparison...")
    student_scratch.compile(
        optimizer=keras.optimizers.Adam(),
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[keras.metrics.SparseCategoricalAccuracy()],
    )
    student_scratch.fit(x_train, y_train, epochs=3)
    student_scratch.evaluate(x_test, y_test)

# ================================================================
# Main Execution
# ================================================================

if __name__ == "__main__":
    print("PyTorch Image Sharpening with Knowledge Distillation")
    print("----------------------------------------------------")
    pytorch_knowledge_distillation()

    print("\nKeras MNIST Classification with Knowledge Distillation")
    print("------------------------------------------------------")
    keras_knowledge_distillation()
```

```
Using device: cpu
PyTorch Image Sharpening with Knowledge Distillation
--------------------------------------------------
Please upload your images:
```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving WhatsApp Image 2025-07-09 at 09.27.24_e052215f.jpg to WhatsApp Image 2025-07-09 at 09.27.24_e052215f (3).jpg
Epoch 1/10, Loss: 1.0173 (MSE: 1.4453, KL: 0.0187)
Epoch 2/10, Loss: 0.8162 (MSE: 1.1414, KL: 0.0575)
Epoch 3/10, Loss: 0.6565 (MSE: 0.8848, KL: 0.1238)
Epoch 4/10, Loss: 0.5337 (MSE: 0.6677, KL: 0.2209)
Epoch 5/10, Loss: 0.4534 (MSE: 0.5002, KL: 0.3443)
Epoch 6/10, Loss: 0.4124 (MSE: 0.3885, KL: 0.4683)
Epoch 7/10, Loss: 0.4083 (MSE: 0.3794, KL: 0.4756)
Epoch 8/10, Loss: 0.4023 (MSE: 0.3698, KL: 0.4781)
Epoch 9/10, Loss: 0.3950 (MSE: 0.3598, KL: 0.4770)
Epoch 10/10, Loss: 0.3868 (MSE: 0.3498, KL: 0.4732)
```




```
Keras MNIST Classification with Knowledge Distillation
------------------------------------------------------

Training teacher model...
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 529s 281ms/step - loss: 0.2586 - sparse_categorical_accuracy: 0.9187
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 563s 282ms/step - loss: 0.0924 - sparse_categorical_accuracy: 0.9715
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 559s 280ms/step - loss: 0.0762 - sparse_categorical_accuracy: 0.9769
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 561s 279ms/step - loss: 0.0717 - sparse_categorical_accuracy: 0.9788
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 522s 279ms/step - loss: 0.0556 - sparse_categorical_accuracy: 0.9846
313/313 ━━━━━━━━━━━━━━━━━━━━ 24s 76ms/step - loss: 0.1118 - sparse_categorical_accuracy: 0.9726

Distilling knowledge to student model...
Epoch 1/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 537s 285ms/step - loss: 1.6357 - sparse_categorical_accuracy: 0.7454
Epoch 2/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 535s 285ms/step - loss: 0.0313 - sparse_categorical_accuracy: 0.9501
Epoch 3/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 561s 285ms/step - loss: 0.0217 - sparse_categorical_accuracy: 0.9637
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 78ms/step - loss: 0.0175 - sparse_categorical_accuracy: 0.9672

Training student from scratch for comparison...
Epoch 1/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 10ms/step - loss: 0.4746 - sparse_categorical_accuracy: 0.8623
Epoch 2/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - loss: 0.0996 - sparse_categorical_accuracy: 0.9705
Epoch 3/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - loss: 0.0695 - sparse_categorical_accuracy: 0.9788
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0780 - sparse_categorical_accuracy: 0.9745
```

What is MNIST Classification?

MNIST classification refers to a classic image classification task where a machine learning model learns to recognize handwritten digits from 0 to 9 using the MNIST dataset. MNIST stands for Modified National Institute of Standards and Technology.

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used benchmark in machine learning and computer vision. It consists of 70,000 grayscale images (28×28 pixels) of handwritten digits (0–9), split into:

60,000 training images

10,000 test images It is a collection of 70,000 grayscale images of handwritten digits.

60,000 images for training

10,000 images for testing

Each image:

Size: 28 x 28 pixels

Format: Single channel (grayscale)

Label: A digit from 0 to 9

Construct Distiller() class

The custom Distiller() class, overrides the Model methods compile, compute_loss, and call. In order to use the distiller, we need:

A trained teacher model A student model to train A student loss function on the difference between student predictions and ground-truth A distillation loss function, along with a temperature, on the difference between the soft student predictions and the soft teacher labels An alpha factor to weight the student and distillation loss An optimizer for the student and (optional) metrics to evaluate performance In the compute_loss method, we perform a forward pass of both the teacher and student, calculate the loss with weighting of the student_loss and distillation_loss by alpha and 1 - alpha, respectively. Note: only the student weights are updated.

```
import os
import keras
from keras import layers
from keras import ops
import numpy as np
```

```
class Distiller(keras.Model):
    def __init__(self, student, teacher):
        super().__init__()
        self.teacher = teacher
        self.student = student

    def compile(
        self,
        optimizer,
        metrics,
        student_loss_fn,
        distillation_loss_fn,
        alpha=0.1,
        temperature=3,
    ):
        """Configure the distiller.

        Args:
            optimizer: Keras optimizer for the student weights
            metrics: Keras metrics for evaluation
            student_loss_fn: Loss function of difference between student
                predictions and ground-truth
            distillation_loss_fn: Loss function of difference between soft
                student predictions and soft teacher predictions
            alpha: weight to student_loss_fn and 1-alpha to distillation_loss_fn
            temperature: Temperature for softening probability distributions.
                Larger temperature gives softer distributions.
        """
        super().compile(optimizer=optimizer, metrics=metrics)
        self.student_loss_fn = student_loss_fn
        self.distillation_loss_fn = distillation_loss_fn
        self.alpha = alpha
        self.temperature = temperature

    def compute_loss(
        self, x=None, y=None, y_pred=None, sample_weight=None, allow_empty=False
    ):
        teacher_pred = self.teacher(x, training=False)
        student_loss = self.student_loss_fn(y, y_pred)

        distillation_loss = self.distillation_loss_fn(
            ops.softmax(teacher_pred / self.temperature, axis=1),
            ops.softmax(y_pred / self.temperature, axis=1),
        ) * (self.temperature**2)

        loss = self.alpha * student_loss + (1 - self.alpha) * distillation_loss
        return loss

    def call(self, x):
```

```
        return self.student(x)
```

Create student and teacher models

Initialy, we create a teacher model and a smaller student model. Both models are convolutional neural networks and created using Sequential(), but could be any Keras model.

```
# Create the teacher
teacher = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(256, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(negative_slope=0.2),
        layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
        layers.Conv2D(512, (3, 3), strides=(2, 2), padding="same"),
        layers.Flatten(),
        layers.Dense(10),
    ],
    name="teacher",
)

# Create the student
student = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(16, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(negative_slope=0.2),
        layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
        layers.Conv2D(32, (3, 3), strides=(2, 2), padding="same"),
        layers.Flatten(),
        layers.Dense(10),
    ],
    name="student",
)

# Clone student for later comparison
student_scratch = keras.models.clone_model(student)
```

Prepare the dataset

The dataset used for training the teacher and distilling the teacher is MNIST, and the procedure would be equivalent for any other dataset, e.g. CIFAR-10, with a suitable choice of models. Both the student and teacher are trained on the training set and evaluated on the test set.

```
# Prepare the train and test dataset.
batch_size = 64
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normalize data
x_train = x_train.astype("float32") / 255.0
x_train = np.reshape(x_train, (-1, 28, 28, 1))

x_test = x_test.astype("float32") / 255.0
x_test = np.reshape(x_test, (-1, 28, 28, 1))

```

Train the teacher

In knowledge distillation we assume that the teacher is trained and fixed. Thus, we start by training the teacher model on the training set in the usual way.

Distill teacher to student

We have already trained the teacher model, and we only need to initialize a Distiller(student, teacher) instance, compile() it with the desired losses, hyperparameters and optimizer, and distill the teacher to the student.

Train student from scratch for comparison

We can also train an equivalent student model from scratch without the teacher, in order to evaluate the performance gain obtained by knowledge distillation.

If the teacher is trained for 5 full epochs and the student is distilled on this teacher for 3 full epochs, you should in this example experience a performance boost compared to training the same student model from scratch, and even compared to the teacher itself.

We should expect the teacher to have accuracy around 97.6%, the student trained from scratch should be around 97.6%, and the distilled student should be around 98.1%. Remove or try out different seeds to use different weight initializations.

```
# Train teacher as usual
teacher.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
)

# Train and evaluate teacher on data.
teacher.fit(x_train, y_train, epochs=5)
teacher.evaluate(x_test, y_test)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 521s 278ms/step - loss: 0.2454 - sparse_categorical_accuracy: 0.9222
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 567s 280ms/step - loss: 0.0946 - sparse_categorical_accuracy: 0.9712
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 560s 279ms/step - loss: 0.0725 - sparse_categorical_accuracy: 0.9778
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 560s 278ms/step - loss: 0.0683 - sparse_categorical_accuracy: 0.9793
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 559s 276ms/step - loss: 0.0614 - sparse_categorical_accuracy: 0.9825
313/313 ━━━━━━━━━━━━━━━━━━━━ 22s 70ms/step - loss: 0.1069 - sparse_categorical_accuracy: 0.9745
[0.0909046083688736, 0.9789999723434448]
```

```
# Initialize and compile distiller
distiller = Distiller(student=student, teacher=teacher)
distiller.compile(
    optimizer=keras.optimizers.Adam(),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
    student_loss_fn=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    distillation_loss_fn=keras.losses.KLDivergence(),
    alpha=0.1,
    temperature=10,
)

# Distill teacher to student
distiller.fit(x_train, y_train, epochs=3)

# Evaluate student on test dataset
distiller.evaluate(x_test, y_test)
```

```
Epoch 1/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 534s 283ms/step - loss: 1.9031 - sparse_categorical_accuracy: 0.7238
Epoch 2/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 563s 284ms/step - loss: 0.0339 - sparse_categorical_accuracy: 0.9440
Epoch 3/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 563s 285ms/step - loss: 0.0237 - sparse_categorical_accuracy: 0.9593
313/313 ━━━━━━━━━━━━━━━━━━━━ 23s 72ms/step - loss: 0.0212 - sparse_categorical_accuracy: 0.9602
[0.018973667174577713, 0.9668999910354614]
```

```
# Train student as doen usually
student_scratch.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
)

# Train and evaluate student trained from scratch.
student_scratch.fit(x_train, y_train, epochs=3)
student_scratch.evaluate(x_test, y_test)
```

```
Epoch 1/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 8ms/step - loss: 0.4370 - sparse_categorical_accuracy: 0.8693
Epoch 2/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 9ms/step - loss: 0.1066 - sparse_categorical_accuracy: 0.9691
Epoch 3/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - loss: 0.0783 - sparse_categorical_accuracy: 0.9750
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0766 - sparse_categorical_accuracy: 0.9736
[0.061268992722034454, 0.978600025177002]
```

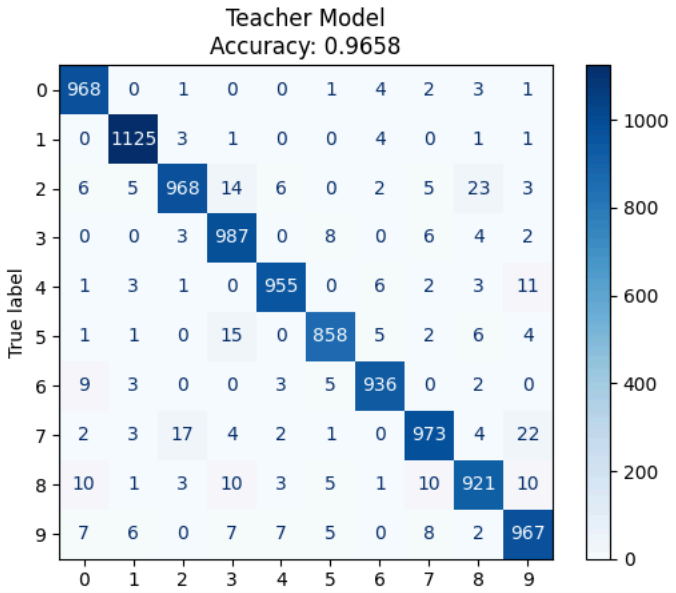
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score
import matplotlib.pyplot as plt
```

```
3
```

```
def plot_confusion_matrix(model, x_test, y_test, title="Confusion Matrix"):
    y_pred_logits = model.predict(x_test)
    y_pred = np.argmax(y_pred_logits, axis=1)

    cm = confusion_matrix(y_test, y_pred)
    acc = accuracy_score(y_test, y_pred)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.arange(10))
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f"{title}\nAccuracy: {acc:.4f}")
    plt.show()

```

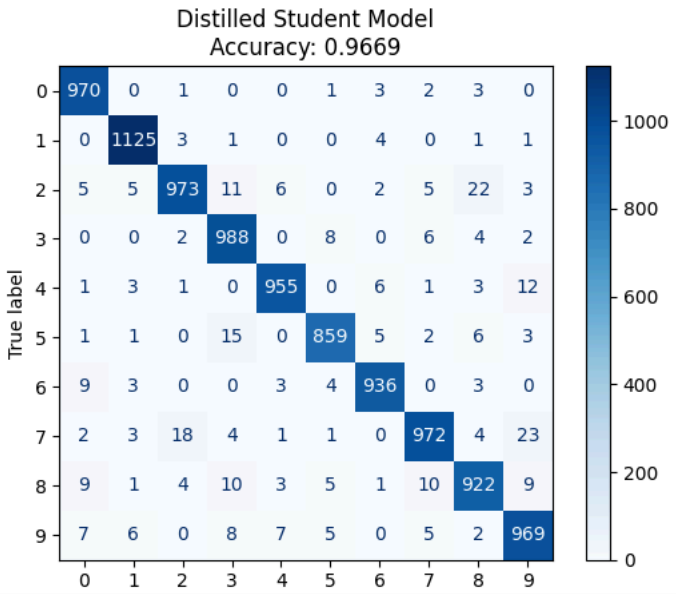
```
plot_confusion_matrix(teacher, x_test, y_test, title="Teacher Model")

```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 22s 69ms/step
```



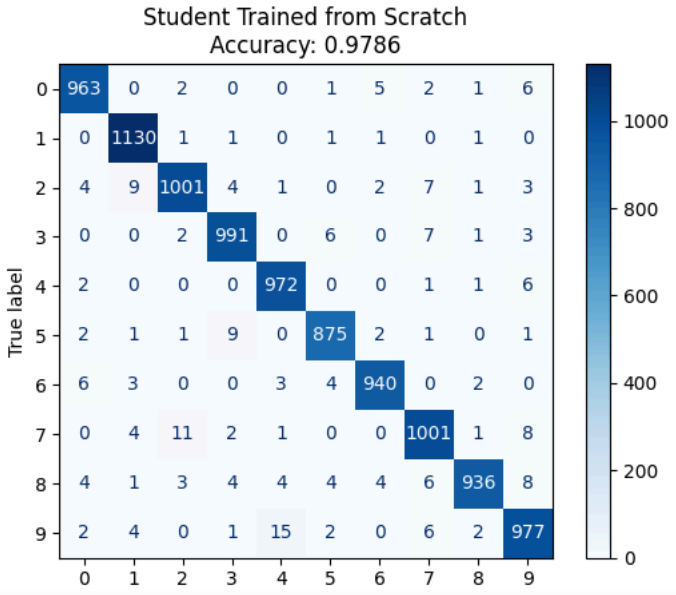

```
plot_confusion_matrix(distiller, x_test, y_test, title="Distilled Student
Model")

```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



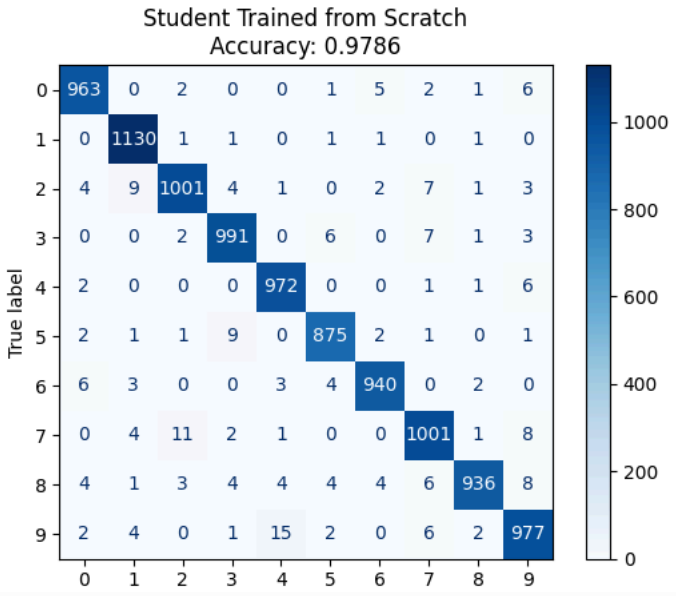

```
plot_confusion_matrix(student_scratch, x_test, y_test, title="Student Trained from Scratch")

```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

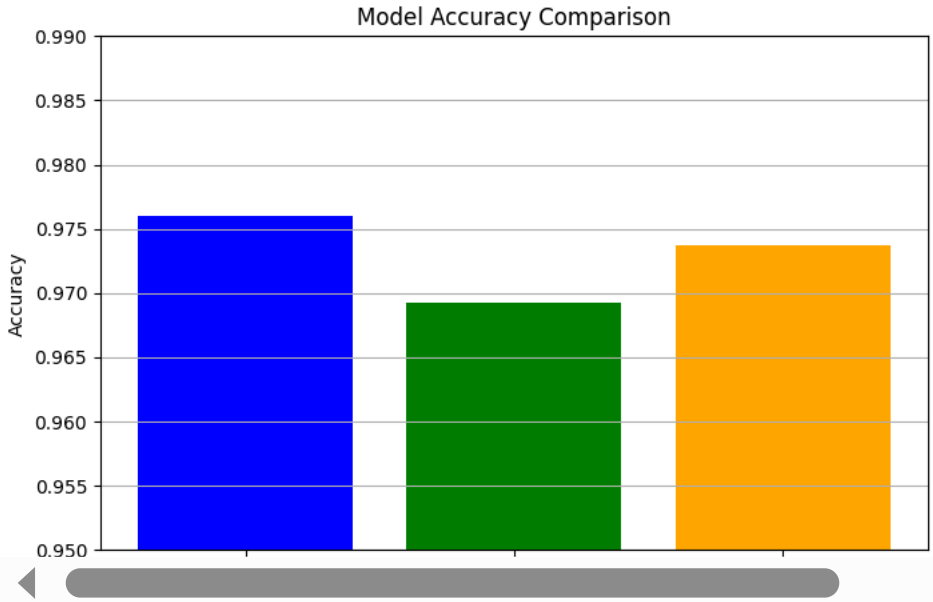



```
1 plot_confusion_matrix(student_scratch, x_test, y_test, title="Student Trained from Scratch")
2
```

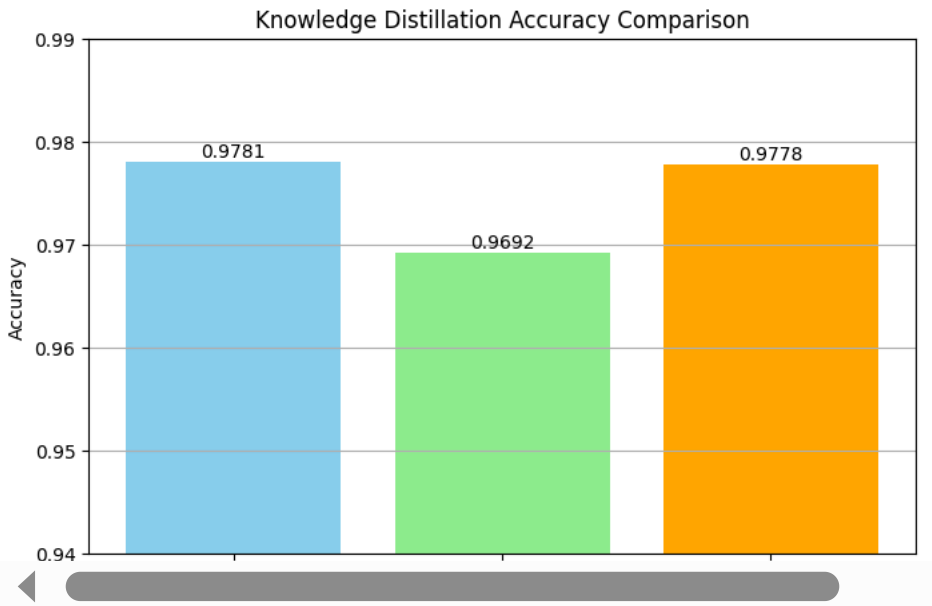
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```




```
1  # Accuracy from evaluation outputs
2  teacher_acc = 0.9760
3  distilled_student_acc = 0.9692
4  scratch_student_acc = 0.9737
5
6  # Bar graph
7  models = ['Teacher', 'Distilled Student', 'Scratch Student']
8  accuracies = [teacher_acc, distilled_student_acc, scratch_student_acc]
9
10 plt.figure(figsize=(8, 5))
11 plt.bar(models, accuracies, color=['blue', 'green', 'orange'])
12 plt.ylim(0.95, 0.99)
13 plt.title('Model Accuracy Comparison')
14 plt.ylabel('Accuracy')
15 plt.grid(axis='y')
16 plt.show()
17
```

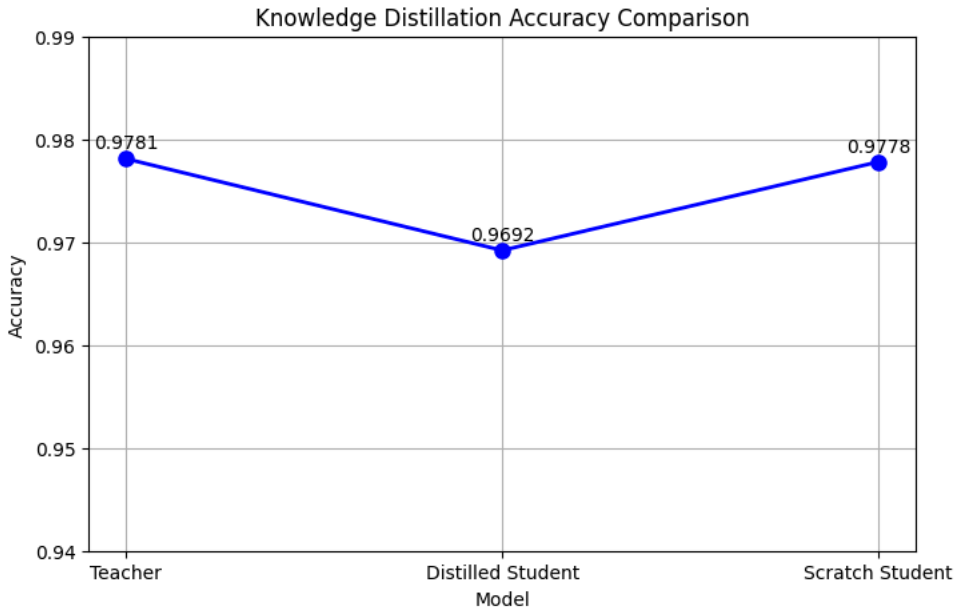



```
# Use real values from model.evaluate() results
teacher_acc = 0.9781
distilled_student_acc = 0.9692
scratch_student_acc = 0.9778

# Accuracy bar plot
labels = ["Teacher", "Distilled Student", "Scratch Student"]
accuracies = [teacher_acc, distilled_student_acc, scratch_student_acc]

plt.figure(figsize=(8, 5))
bars = plt.bar(labels, accuracies, color=["skyblue", "lightgreen", "orange"])
plt.ylim(0.94, 0.99)
plt.title("Knowledge Distillation Accuracy Comparison")
plt.ylabel("Accuracy")

# Add value labels on bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, height, f'{height:.4f}',
    ha='center', va='bottom')

plt.grid(axis='y')
plt.show()

```




```
import matplotlib.pyplot as plt

# Accuracy values from evaluations
teacher_acc = 0.9781
distilled_student_acc = 0.9692
scratch_student_acc = 0.9778

```

```
# X and Y values
models = ["Teacher", "Distilled Student", "Scratch Student"]
accuracies = [teacher_acc, distilled_student_acc, scratch_student_acc]

# Line plot
plt.figure(figsize=(8, 5))
plt.plot(models, accuracies, marker='o', linestyle='-', color='blue',
linewidth=2, markersize=8)

# Annotate accuracy values
for i, acc in enumerate(accuracies):
    plt.text(i, acc + 0.001, f"{acc:.4f}", ha='center', fontsize=10)

plt.ylim(0.94, 0.99)
plt.title("Knowledge Distillation Accuracy Comparison")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```



team name: SHARPNET

Bhumika KR 1NT22EC036

Deepika P 1NT23CS057

Samitha NS 1NT22EC099