

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

```
1 !pip install opencv-python matplotlib tensorflow
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import cv2
6 import tensorflow as tf
7 from tensorflow.keras import layers, models
8
```

```
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) (0.6.0)
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/python3.11/dist-packages (from tensorflow) (2.32.3)
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) (0.37.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.43.0)
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->tensorflow) (3.10.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10.0)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.1.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.7.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (0.7.0)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.0.6)
```

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->tensorflow
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->tensorflow)

```
1  # Step 1: Setup
2  import torch
3  import torch.nn as nn
4  import torch.optim as optim
5  import torchvision.transforms as transforms
6  from torch.utils.data import Dataset, DataLoader
7  import cv2
8  import numpy as np
9  import matplotlib.pyplot as plt
10 from PIL import Image
11 import os
12 import requests
13 from io import BytesIO
14 from google.colab import files
15
16 # Device
17 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
18 print(f"Using device: {device}")
19
20 # Step 2: Dataset
21 class SharpeningDataset(Dataset):
22     def __init__(self, image_paths, blur_radius=3, transform=None):
23         self.image_paths = image_paths
24         self.blur_radius = blur_radius
25         self.transform = transform
26
27     def __len__(self):
28         return len(self.image_paths)
29
30     def __getitem__(self, idx):
31         img_path = self.image_paths[idx]
32         if img_path.startswith("http"):
33             response = requests.get(img_path)
34             sharp_img = Image.open(BytesIO(response.content)).convert("RGB")
35         else:
36             sharp_img = Image.open(img_path).convert("RGB")
37
38         sharp_np = np.array(sharp_img)
39         blurred_np = cv2.GaussianBlur(sharp_np, (self.blur_radius, self.
40             blur_radius), 0)
41         blurred_img = Image.fromarray(blurred_np)
42
43         if self.transform:
44             sharp_img = self.transform(sharp_img)
45             blurred_img = self.transform(blurred_img)
46
47         return blurred_img, sharp_img
48
49 # Transform
50 transform = transforms.Compose([
51     transforms.Resize((256, 256)),
52     transforms.ToTensor(),
53     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
54 ])
55
56 # Upload images
57 print("Upload images for training (sharp ground truths):")
58 uploaded = files.upload()
59 image_paths = list(uploaded.keys())
60
61 dataset = SharpeningDataset(image_paths, transform=transform)
62 dataloader = DataLoader(dataset, batch_size=8, shuffle=True)
63
64 # Step 3: Models
65 class TeacherModel(nn.Module):
66     def __init__(self):
67         super(TeacherModel, self).__init__()
68         self.encoder = nn.Sequential(
69             nn.Conv2d(3, 64, kernel_size=3, padding=1), nn.ReLU(),
70             nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.ReLU(),
71             nn.MaxPool2d(2),
72             nn.Conv2d(128, 256, kernel_size=3, padding=1), nn.ReLU(),
73             nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.ReLU(),
74             nn.MaxPool2d(2),
75         )
76         self.decoder = nn.Sequential(
77             nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2), nn.ReLU(),
```

```

77         nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.ReLU(),
78         nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2), nn.ReLU(),
79         nn.Conv2d(64, 3, kernel_size=3, padding=1),
80     )
81
82     def forward(self, x):
83         return self.decoder(self.encoder(x))
84
85     class StudentModel(nn.Module):
86         def __init__(self):
87             super(StudentModel, self).__init__()
88             self.net = nn.Sequential(
89                 nn.Conv2d(3, 32, kernel_size=3, padding=1), nn.ReLU(),
90                 nn.Conv2d(32, 32, kernel_size=3, padding=1), nn.ReLU(),
91                 nn.Conv2d(32, 3, kernel_size=3, padding=1),
92             )
93
94         def forward(self, x):
95             return self.net(x)
96
97     teacher = TeacherModel().to(device)
98     student = StudentModel().to(device)
99
100    # Step 4: Knowledge Distillation Training
101    def train_student_with_distillation(epochs=10, alpha=0.7, temperature=2.0):
102        criterion_mse = nn.MSELoss()
103        criterion_kl = nn.KLDivLoss(reduction='batchmean')
104        optimizer = optim.Adam(student.parameters(), lr=0.001)
105        scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
106        teacher.eval()
107
108        for epoch in range(epochs):
109            for blurred, sharp in dataloader:
110                blurred, sharp = blurred.to(device), sharp.to(device)
111
112                with torch.no_grad():
113                    teacher_logits = teacher(blurred)
114
115                student_logits = student(blurred)
116
117                loss_mse = criterion_mse(student_logits, sharp)
118
119                T = temperature
120                soft_teacher = nn.functional.softmax(teacher_logits / T, dim=1)
121                soft_student = nn.functional.log_softmax(student_logits / T, dim=1)
122
123                B, C, H, W = student_logits.shape
124                soft_teacher = soft_teacher.permute(0, 2, 3, 1).reshape(-1, C)
125                soft_student = soft_student.permute(0, 2, 3, 1).reshape(-1, C)
126
127                loss_kl = criterion_kl(soft_student, soft_teacher) * (T ** 2)
128
129                loss = alpha * loss_mse + (1 - alpha) * loss_kl
130                optimizer.zero_grad()
131                loss.backward()
132                optimizer.step()
133
134            scheduler.step()
135            print(f"Epoch {epoch+1}/{epochs} | Total Loss: {loss.item():.4f} | MSE: {loss_mse.item():.4f} | KL: {loss_kl.item():.4f}")
136
137    # Step 5: Visualization
138    def denormalize(tensor):
139        mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1).to(device)
140        std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1).to(device)
141        return torch.clamp(tensor * std + mean, 0, 1)
142
143    def visualize_results(num_images=3):
144        teacher.eval()
145        student.eval()
146        with torch.no_grad():
147            for i, (blurred, sharp) in enumerate(dataloader):
148                if i >= num_images:
149                    break
150                blurred, sharp = blurred.to(device), sharp.to(device)
151                out_student = student(blurred)
152                out_teacher = teacher(blurred)
153
154                # Plot
155                fig, axes = plt.subplots(1, 4, figsize=(16, 4))
156                images = [blurred[0], out_student[0], out_teacher[0], sharp[0]]
157                titles = ['Blurred Input', 'Student Output', 'Teacher Output',

```

```

158         ground_truth_j
159     for ax, img, title in zip(axes, images, titles):
160         ax.imshow(denormalize(img).permute(1, 2, 0).cpu().numpy())
161         ax.set_title(title)
162         ax.axis('off')
163     plt.show()
164
165 # Step 6: Run
166 train_student_with_distillation(epochs=10)
167 visualize_results()
168
169 # Save models
170 torch.save(teacher.state_dict(), "teacher_blur2sharp.pth")
171 torch.save(student.state_dict(), "student_blur2sharp.pth")
172

```



Using device: cpu

Upload images for training (sharp ground truths):

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

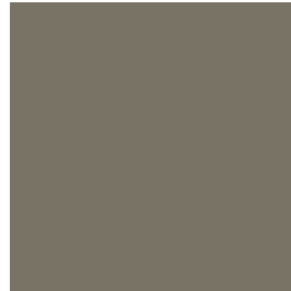
Blurred Input



Student Output



Teacher Output



Ground Truth



```

1 # Step 1: Set up environment and imports
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 import torchvision.transforms as transforms
6 from torch.utils.data import Dataset, DataLoader
7 import cv2
8 import numpy as np
9 import matplotlib.pyplot as plt
10 from PIL import Image
11 import os
12 from google.colab import files
13 from io import BytesIO
14
15 # Check for GPU
16 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
17 print(f"Using device: {device}")
18
19 # Step 2: Data Preparation
20 class SharpeningDataset(Dataset):
21     def __init__(self, image_paths=None, blur_radius=3, transform=None):
22         self.image_paths = image_paths or []
23         self.blur_radius = blur_radius
24         self.transform = transform
25
26     def __len__(self):
27         return len(self.image_paths)
28
29     def __getitem__(self, idx):
30         # Load image (works with both local files and URLs)
31         if isinstance(self.image_paths[idx], str) and self.image_paths[idx].startswith('http'):
32             response = requests.get(self.image_paths[idx])
33             sharp_img = Image.open(BytesIO(response.content)).convert('RGB')
34         else:

```

```

35         sharp_img = Image.open(self.image_paths[idx]).convert('RGB')
36
37         # Create blurred version
38         sharp_np = np.array(sharp_img)
39         blurred_np = cv2.GaussianBlur(sharp_np, (self.blur_radius, self.blur_radius), 0)
40         blurred_img = Image.fromarray(blurred_np)
41
42         if self.transform:
43             sharp_img = self.transform(sharp_img)
44             blurred_img = self.transform(blurred_img)
45
46         return blurred_img, sharp_img
47
48 # Define transforms
49 transform = transforms.Compose([
50     transforms.Resize((256, 256)),
51     transforms.ToTensor(),
52     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
53 ])
54
55 # Load data (manual upload in Colab)
56 print("Please upload your images:")
57 uploaded = files.upload()
58 image_paths = list(uploaded.keys())
59
60 if not image_paths:
61     raise ValueError("No images uploaded! Please try again.")
62
63 dataset = SharpeningDataset(image_paths, transform=transform)
64 dataloader = DataLoader(dataset, batch_size=8, shuffle=True)
65
66 # Step 3: Model Definitions
67 class TeacherModel(nn.Module):
68     def __init__(self):
69         super(TeacherModel, self).__init__()
70         self.encoder = nn.Sequential(
71             nn.Conv2d(3, 64, kernel_size=3, padding=1),
72             nn.ReLU(),
73             nn.Conv2d(64, 128, kernel_size=3, padding=1),
74             nn.ReLU(),
75             nn.MaxPool2d(2),
76             nn.Conv2d(128, 256, kernel_size=3, padding=1),
77             nn.ReLU(),
78             nn.Conv2d(256, 256, kernel_size=3, padding=1),
79             nn.ReLU(),
80             nn.MaxPool2d(2),
81         )
82         self.decoder = nn.Sequential(
83             nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2),
84             nn.ReLU(),
85             nn.Conv2d(128, 128, kernel_size=3, padding=1),
86             nn.ReLU(),
87             nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2),
88             nn.ReLU(),
89             nn.Conv2d(64, 3, kernel_size=3, padding=1),
90         )
91
92     def forward(self, x):
93         return self.decoder(self.encoder(x))
94
95 class StudentModel(nn.Module):
96     def __init__(self):
97         super(StudentModel, self).__init__()
98         self.net = nn.Sequential(
99             nn.Conv2d(3, 32, kernel_size=3, padding=1),
100             nn.ReLU(),
101             nn.Conv2d(32, 32, kernel_size=3, padding=1),
102             nn.ReLU(),
103             nn.Conv2d(32, 3, kernel_size=3, padding=1),
104         )
105
106     def forward(self, x):
107         return self.net(x)
108
109 teacher = TeacherModel().to(device)
110 student = StudentModel().to(device)
111
112 # Step 4: Training with Knowledge Distillation
113 def train_student_with_distillation(epochs=10, temperature=2.0, alpha=0.7):
114     criterion_mse = nn.MSELoss()
115     criterion_kl = nn.KLDivLoss(reduction='batchmean')
116     optimizer = optim.Adam(student.parameters(), lr=0.001)

```

```


117 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
118
119 teacher.eval()
120
121 for epoch in range(epochs):
122     for blurred, sharp in dataloader:
123         blurred, sharp = blurred.to(device), sharp.to(device)
124
125         with torch.no_grad():
126             teacher_logits = teacher(blurred)
127
128         optimizer.zero_grad()
129         student_logits = student(blurred)
130
131         # MSE loss
132         loss_mse = criterion_mse(student_logits, sharp)
133
134         # KL divergence loss
135         T = temperature
136         soft_teacher = nn.functional.softmax(teacher_logits / T, dim=1)
137         soft_student = nn.functional.log_softmax(student_logits / T, dim=1)
138
139         # Reshape for KLDivLoss
140         B, C, H, W = student_logits.shape
141         soft_teacher = soft_teacher.permute(0, 2, 3, 1).reshape(-1, C)
142         soft_student = soft_student.permute(0, 2, 3, 1).reshape(-1, C)
143
144         loss_kl = criterion_kl(soft_student, soft_teacher) * (T ** 2)
145
146         # Combined loss
147         loss = alpha * loss_mse + (1 - alpha) * loss_kl
148         loss.backward()
149         optimizer.step()
150
151     scheduler.step()
152     print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f} (MSE: {loss_mse.item():.4f}, KL: {loss_kl.item():.4f})")
153
154 # Step 5: Evaluation and Visualization
155 def denormalize(tensor):
156     mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1).to(device)
157     std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1).to(device)
158     return torch.clamp(tensor * std + mean, 0, 1)
159
160 def visualize_results(num_images=3):
161     student.eval()
162     teacher.eval()
163     with torch.no_grad():
164         for i, (blurred, sharp) in enumerate(dataloader):
165             if i >= num_images:
166                 break
167
168             blurred, sharp = blurred.to(device), sharp.to(device)
169             student_output = student(blurred)
170             teacher_output = teacher(blurred)
171
172             # Denormalize images
173             blurred_img = denormalize(blurred[0]).cpu().permute(1, 2, 0).numpy()
174             sharp_img = denormalize(sharp[0]).cpu().permute(1, 2, 0).numpy()
175             student_img = denormalize(student_output[0]).cpu().permute(1, 2, 0).numpy()
176             teacher_img = denormalize(teacher_output[0]).cpu().permute(1, 2, 0).numpy()
177
178             # Plot comparison
179             plt.figure(figsize=(20, 5))
180             titles = ['Blurred Input', 'Student Output', 'Teacher Output', 'Ground Truth']
181             images = [blurred_img, student_img, teacher_img, sharp_img]
182
183             for j in range(4):
184                 plt.subplot(1, 4, j+1)
185                 plt.imshow(images[j])
186                 plt.title(titles[j])
187                 plt.axis('off')
188
189             plt.show()
190
191 # Step 6: Run Training and Evaluation
192 if __name__ == "__main__":
193     # Train teacher first (optional)
194     # train_teacher() # You would need to implement this
195
196     # Train student with distillation
197     train_student_with_distillation(epochs=10)
198

```

```

199 # Save models
200 torch.save(teacher.state_dict(), "teacher.pth")
201 torch.save(student.state_dict(), "student.pth")
202
203 # Visualize results
204 visualize_results()

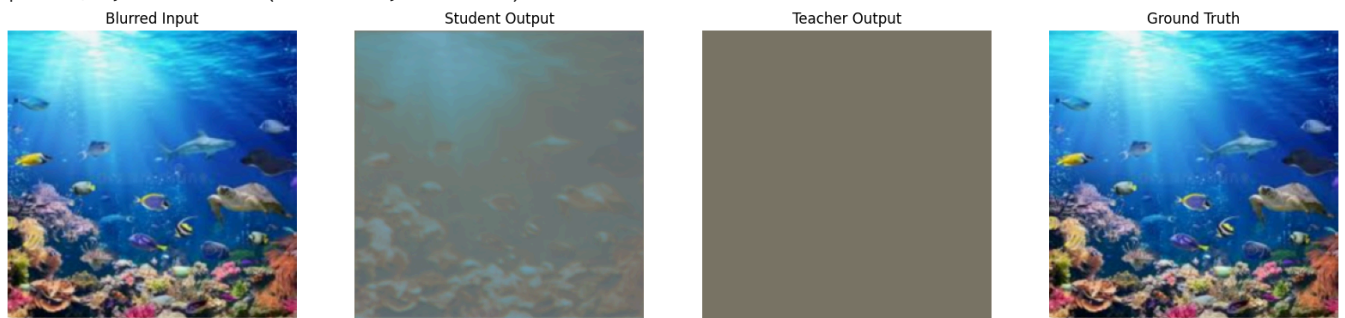
```

 Using device: cpu
 Please upload your images:

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)



```

1 # Shared
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # PyTorch (Image sharpening)
6 import torch
7 import torch.nn as nn
8 import torch.optim as optim
9 import torchvision.transforms as transforms
10 from torch.utils.data import Dataset, DataLoader
11 import cv2
12 from PIL import Image
13 from google.colab import files
14 import os
15
16 # Keras (MNIST classification)
17 import keras
18 from keras import layers
19 from keras import ops
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

17 # Keras imports
18 import keras
19 from keras import layers
20 from keras import ops
21
22 # Check for GPU
23 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
24 print(f"Using device: {device}")
25
26 # =====
27 # PyTorch Implementation - Image Sharpening with Knowledge Distillation
28 # =====
29
30 class SharpeningDataset(Dataset):
31     def __init__(self, image_paths=None, blur_radius=3, transform=None):
32         self.image_paths = image_paths or []
33         self.blur_radius = blur_radius
34         self.transform = transform
35
36     def __len__(self):
37         return len(self.image_paths)
38
39     def __getitem__(self, idx):
40         # Load image (works with both local files and URLs)
41         if isinstance(self.image_paths[idx], str) and self.image_paths[idx].startswith('http'):
42             response = requests.get(self.image_paths[idx])
43             sharp_img = Image.open(BytesIO(response.content)).convert('RGB')
44         else:
45             sharp_img = Image.open(self.image_paths[idx]).convert('RGB')
46
47         # Create blurred version
48         sharp_np = np.array(sharp_img)
49         blurred_np = cv2.GaussianBlur(sharp_np, (self.blur_radius, self.blur_radius), 0)
50         blurred_img = Image.fromarray(blurred_np)
51
52         if self.transform:
53             sharp_img = self.transform(sharp_img)
54             blurred_img = self.transform(blurred_img)
55
56         return blurred_img, sharp_img
57
58 class TeacherModel(nn.Module):
59     def __init__(self):
60         super(TeacherModel, self).__init__()
61         self.encoder = nn.Sequential(
62             nn.Conv2d(3, 64, kernel_size=3, padding=1),
63             nn.ReLU(),
64             nn.Conv2d(64, 128, kernel_size=3, padding=1),
65             nn.ReLU(),
66             nn.MaxPool2d(2),
67             nn.Conv2d(128, 256, kernel_size=3, padding=1),
68             nn.ReLU(),
69             nn.Conv2d(256, 256, kernel_size=3, padding=1),
70             nn.ReLU(),
71             nn.MaxPool2d(2),
72         )
73         self.decoder = nn.Sequential(
74             nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2),
75             nn.ReLU(),
76             nn.Conv2d(128, 128, kernel_size=3, padding=1),
77             nn.ReLU(),
78             nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2),
79             nn.ReLU(),
80             nn.Conv2d(64, 3, kernel_size=3, padding=1),
81         )
82
83     def forward(self, x):
84         return self.decoder(self.encoder(x))
85
86 class StudentModel(nn.Module):
87     def __init__(self):
88         super(StudentModel, self).__init__()
89         self.net = nn.Sequential(
90             nn.Conv2d(3, 32, kernel_size=3, padding=1),
91             nn.ReLU(),
92             nn.Conv2d(32, 32, kernel_size=3, padding=1),
93             nn.ReLU(),
94             nn.Conv2d(32, 3, kernel_size=3, padding=1),
95         )
96
97     def forward(self, x):
98         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()

```

```

181         with torch.no_grad():
182             for i, (blurred, sharp) in enumerate(dataloader):
183                 if i >= num_images:
184                     break
185
186                 blurred, sharp = blurred.to(device), sharp.to(device)
187                 student_output = student(blurred)
188                 teacher_output = teacher(blurred)
189
190                 # Denormalize images
191                 blurred_img = denormalize(blurred[0]).cpu().permute(1, 2, 0).numpy()
192                 sharp_img = denormalize(sharp[0]).cpu().permute(1, 2, 0).numpy()
193                 student_img = denormalize(student_output[0]).cpu().permute(1, 2, 0).numpy()
194                 teacher_img = denormalize(teacher_output[0]).cpu().permute(1, 2, 0).numpy()
195
196                 # Plot comparison
197                 plt.figure(figsize=(20, 5))
198                 titles = ['Blurred Input', 'Student Output', 'Teacher Output', 'Ground Truth']
199                 images = [blurred_img, student_img, teacher_img, sharp_img]
200
201                 for j in range(4):
202                     plt.subplot(1, 4, j+1)
203                     plt.imshow(images[j])
204                     plt.title(titles[j])
205                     plt.axis('off')
206
207                 plt.show()
208
209         visualize_results()
210
211 # =====
212 # Keras Implementation - MNIST Classification with Knowledge Distillation
213 # =====
214
215 class Distiller(keras.Model):
216     def __init__(self, student, teacher):
217         super().__init__()
218         self.teacher = teacher
219         self.student = student
220
221     def compile(
222         self,
223         optimizer,
224         metrics,
225         student_loss_fn,
226         distillation_loss_fn,
227         alpha=0.1,
228         temperature=3,
229     ):
230         super().compile(optimizer=optimizer, metrics=metrics)
231         self.student_loss_fn = student_loss_fn
232         self.distillation_loss_fn = distillation_loss_fn
233         self.alpha = alpha
234         self.temperature = temperature
235
236     def compute_loss(
237         self, x=None, y=None, y_pred=None, sample_weight=None, allow_empty=False
238     ):
239         teacher_pred = self.teacher(x, training=False)
240         student_loss = self.student_loss_fn(y, y_pred)
241
242         distillation_loss = self.distillation_loss_fn(
243             ops.softmax(teacher_pred / self.temperature, axis=1),
244             ops.softmax(y_pred / self.temperature, axis=1),
245         ) * (self.temperature**2)
246
247         loss = self.alpha * student_loss + (1 - self.alpha) * distillation_loss
248         return loss
249
250     def call(self, x):
251         return self.student(x)
252
253 def keras_knowledge_distillation():
254     # Create the teacher
255     teacher = keras.Sequential(
256         [
257             keras.Input(shape=(28, 28, 1)),
258             layers.Conv2D(256, (3, 3), strides=(2, 2), padding="same"),
259             layers.LeakyReLU(negative_slope=0.2),
260             layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
261             layers.Conv2D(512, (3, 3), strides=(2, 2), padding="same"),
262             layers.Flatten(),

```

```

263         layers.Dense(10),
264     ],
265     name="teacher",
266 )
267
268 # Create the student
269 student = keras.Sequential(
270     [
271         keras.Input(shape=(28, 28, 1)),
272         layers.Conv2D(16, (3, 3), strides=(2, 2), padding="same"),
273         layers.LeakyReLU(negative_slope=0.2),
274         layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
275         layers.Conv2D(32, (3, 3), strides=(2, 2), padding="same"),
276         layers.Flatten(),
277         layers.Dense(10),
278     ],
279     name="student",
280 )
281
282 # Clone student for later comparison
283 student_scratch = keras.models.clone_model(student)
284
285 # Prepare the dataset
286 batch_size = 64
287 (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
288
289 # Normalize data
290 x_train = x_train.astype("float32") / 255.0
291 x_train = np.reshape(x_train, (-1, 28, 28, 1))
292
293 x_test = x_test.astype("float32") / 255.0
294 x_test = np.reshape(x_test, (-1, 28, 28, 1))
295
296 # Train teacher
297 print("\nTraining teacher model...")
298 teacher.compile(
299     optimizer=keras.optimizers.Adam(),
300     loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
301     metrics=[keras.metrics.SparseCategoricalAccuracy()],
302 )
303 teacher.fit(x_train, y_train, epochs=5)
304 teacher.evaluate(x_test, y_test)
305
306 # Distill teacher to student
307 print("\nDistilling knowledge to student model...")
308 distiller = Distiller(student=student, teacher=teacher)
309 distiller.compile(
310     optimizer=keras.optimizers.Adam(),
311     metrics=[keras.metrics.SparseCategoricalAccuracy()],
312     student_loss_fn=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
313     distillation_loss_fn=keras.losses.KLDivergence(),
314     alpha=0.1,
315     temperature=10,
316 )
317 distiller.fit(x_train, y_train, epochs=3)
318 distiller.evaluate(x_test, y_test)
319
320 # Train student from scratch for comparison
321 print("\nTraining student from scratch for comparison...")
322 student_scratch.compile(
323     optimizer=keras.optimizers.Adam(),
324     loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
325     metrics=[keras.metrics.SparseCategoricalAccuracy()],
326 )
327 student_scratch.fit(x_train, y_train, epochs=3)
328 student_scratch.evaluate(x_test, y_test)
329
330 # =====
331 # Main Execution
332 # =====
333
334 if __name__ == "__main__":
335     print("PyTorch Image Sharpening with Knowledge Distillation")
336     print("-----")
337     pytorch_knowledge_distillation()
338
339     print("\nKeras MNIST Classification with Knowledge Distillation")
340     print("-----")
341     keras_knowledge_distillation()

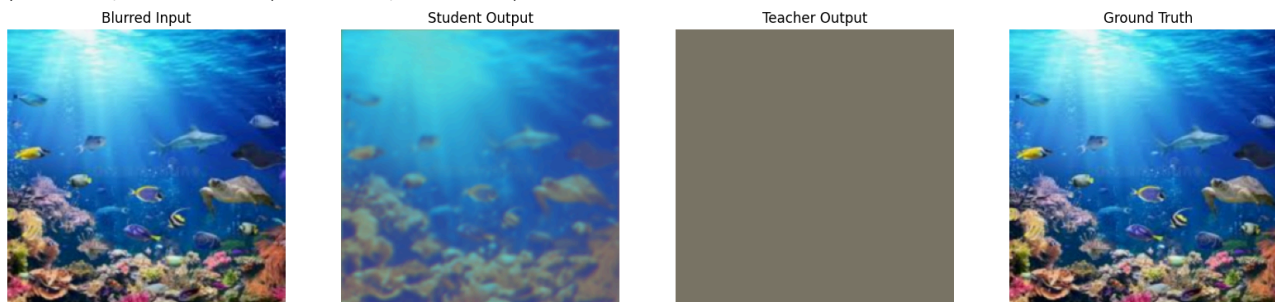
```

Using device: cpu
PyTorch Image Sharpening with Knowledge Distillation

Please upload your images:
 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.

```
1 import os
2 import keras
3 from keras import layers
4 from keras import ops
5 import numpy as np

1 class Distiller(keras.Model):
2     def __init__(self, student, teacher):
3         super().__init__()
4         self.teacher = teacher
5         self.student = student
6
7     def compile(
8         self,
9         optimizer,
10        metrics,
11        student_loss_fn,
12        distillation_loss_fn,
13        alpha=0.1,
14        temperature=3,
15    ):
16        """Configure the distiller.
17
18        Args:
19            optimizer: Keras optimizer for the student weights
20            metrics: Keras metrics for evaluation
21            student_loss_fn: Loss function of difference between student
22                predictions and ground-truth
23            distillation_loss_fn: Loss function of difference between soft
24                student predictions and soft teacher predictions
25            alpha: weight to student_loss_fn and 1-alpha to distillation_loss_fn
26            temperature: Temperature for softening probability distributions.
27                Larger temperature gives softer distributions.
28        """
29        super().compile(optimizer=optimizer, metrics=metrics)
30        self.student_loss_fn = student_loss_fn
31        self.distillation_loss_fn = distillation_loss_fn
32        self.alpha = alpha
33        self.temperature = temperature
34
35    def compute_loss(
36        self, x=None, y=None, y_pred=None, sample_weight=None, allow_empty=False
37    ):
38        teacher_pred = self.teacher(x, training=False)
39        student_loss = self.student_loss_fn(y, y_pred)
40
41        distillation_loss = self.distillation_loss_fn(
42            ops.softmax(teacher_pred / self.temperature, axis=1),
43            ops.softmax(y_pred / self.temperature, axis=1),
44        ) * (self.temperature**2)
45
46        loss = self.alpha * student_loss + (1 - self.alpha) * distillation_loss
47        return loss
48
49    def call(self, x):
```

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

✓ Create student and teacher models

Initially, 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.

```
1  # Create the teacher
2  teacher = keras.Sequential(
3      [
4          keras.Input(shape=(28, 28, 1)),
5          layers.Conv2D(256, (3, 3), strides=(2, 2), padding="same"),
6          layers.LeakyReLU(negative_slope=0.2),
7          layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
8          layers.Conv2D(512, (3, 3), strides=(2, 2), padding="same"),
9          layers.Flatten(),
10         layers.Dense(10),
11     ],
12     name="teacher",
13 )
14
15 # Create the student
16 student = keras.Sequential(
17     [
18         keras.Input(shape=(28, 28, 1)),
19         layers.Conv2D(16, (3, 3), strides=(2, 2), padding="same"),
20         layers.LeakyReLU(negative_slope=0.2),
21         layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
22         layers.Conv2D(32, (3, 3), strides=(2, 2), padding="same"),
23         layers.Flatten(),
24         layers.Dense(10),
25     ],
26     name="student",
27 )
28
29 # Clone student for later comparison
30 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.

```
1 # Prepare the train and test dataset.
2 batch_size = 64
3 (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
4
5 # Normalize data
6 x_train = x_train.astype("float32") / 255.0
7 x_train = np.reshape(x_train, (-1, 28, 28, 1))
8
9 x_test = x_test.astype("float32") / 255.0
10 x_test = np.reshape(x_test, (-1, 28, 28, 1))
11
```

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.

```
1 # Train teacher as usual
2 teacher.compile(
3     optimizer=keras.optimizers.Adam(),
4     loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
5     metrics=[keras.metrics.SparseCategoricalAccuracy()],
6 )
7
8 # Train and evaluate teacher on data.
9 teacher.fit(x_train, y_train, epochs=5)
10 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]
```

```
1 # Initialize and compile distiller
2 distiller = Distiller(student=student, teacher=teacher)
3 distiller.compile(
4     optimizer=keras.optimizers.Adam(),
5     metrics=[keras.metrics.SparseCategoricalAccuracy()],
6     student_loss_fn=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
7     distillation_loss_fn=keras.losses.KLDivergence(),
8     alpha=0.1,
9     temperature=10,
10 )
11
12 # Distill teacher to student
13 distiller.fit(x_train, y_train, epochs=3)
14
15 # Evaluate student on test dataset
16 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]
```

```
1 # Train student as doen usually
2 student_scratch.compile(
3     optimizer=keras.optimizers.Adam(),
4     loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
5     metrics=[keras.metrics.SparseCategoricalAccuracy()],
6 )
7
8 # Train and evaluate student trained from scratch.
9 student_scratch.fit(x_train, y_train, epochs=3)
10 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]
```

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

```

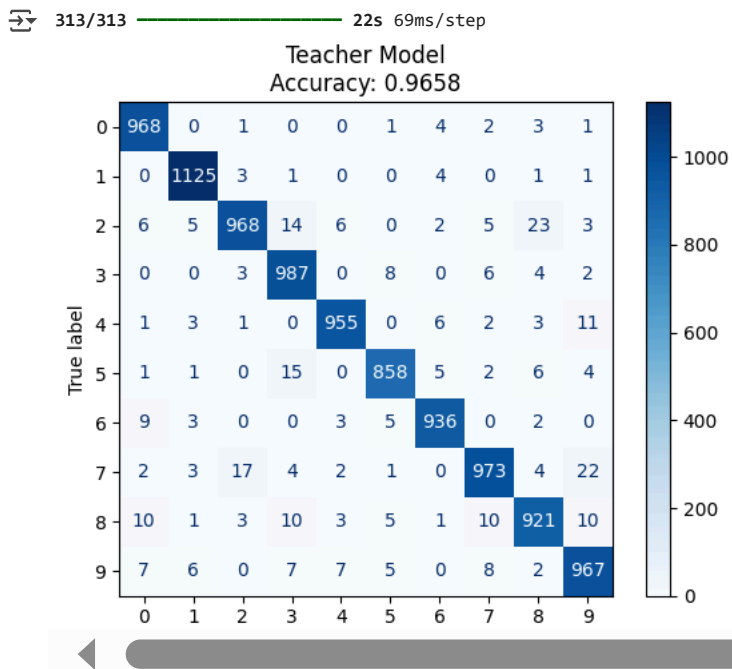
1 def plot_confusion_matrix(model, x_test, y_test, title="Confusion Matrix"):
2     y_pred_logits = model.predict(x_test)
3     y_pred = np.argmax(y_pred_logits, axis=1)
4
5     cm = confusion_matrix(y_test, y_pred)
6     acc = accuracy_score(y_test, y_pred)
7
8     disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.arange(10))
9     disp.plot(cmap=plt.cm.Blues)
10    plt.title(f"{title}\nAccuracy: {acc:.4f}")
11    plt.show()
12

```

```

1 plot_confusion_matrix(teacher, x_test, y_test, title="Teacher Model")
2

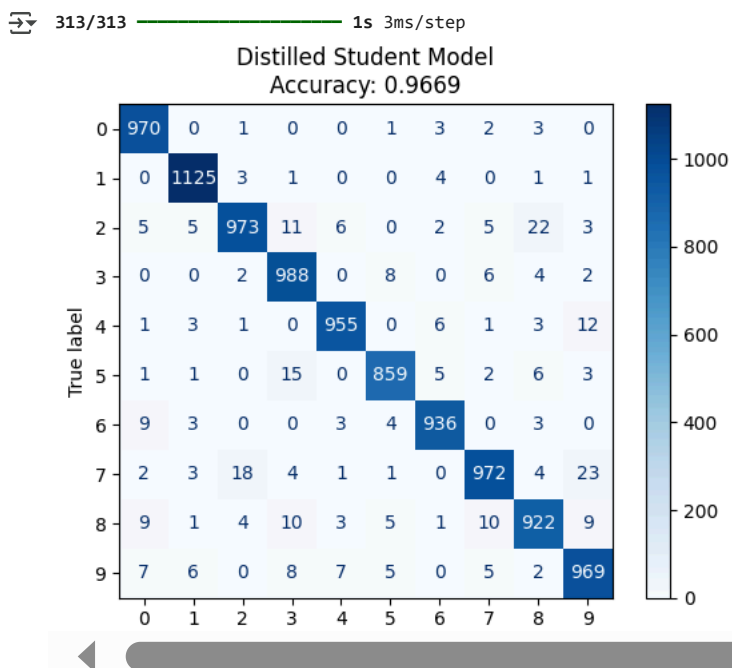
```



```

1 plot_confusion_matrix(distiller, x_test, y_test, title="Distilled Student
  Model")
2

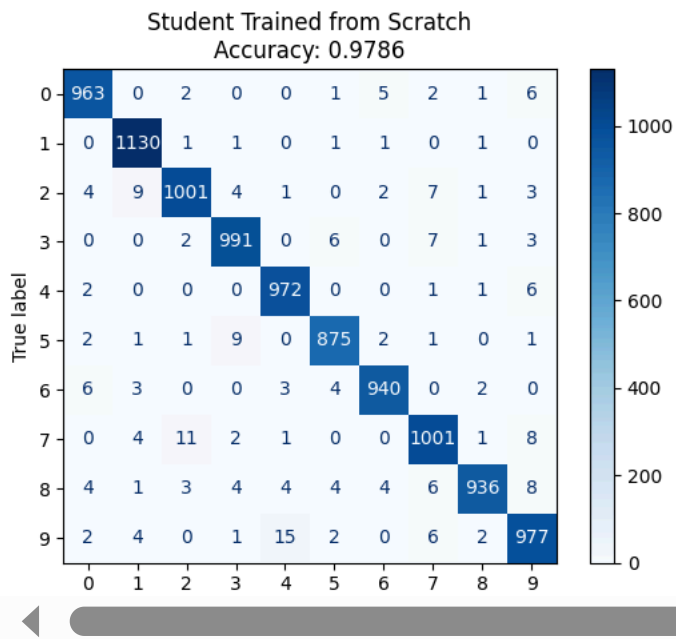
```



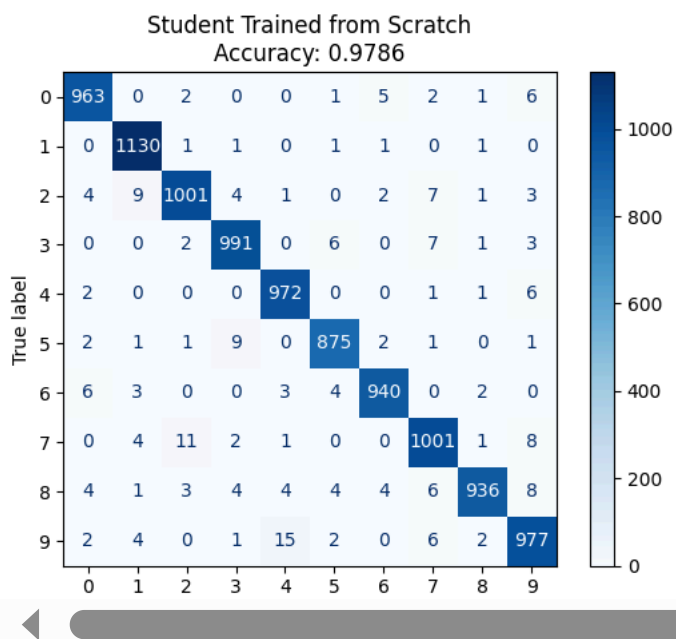
```

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

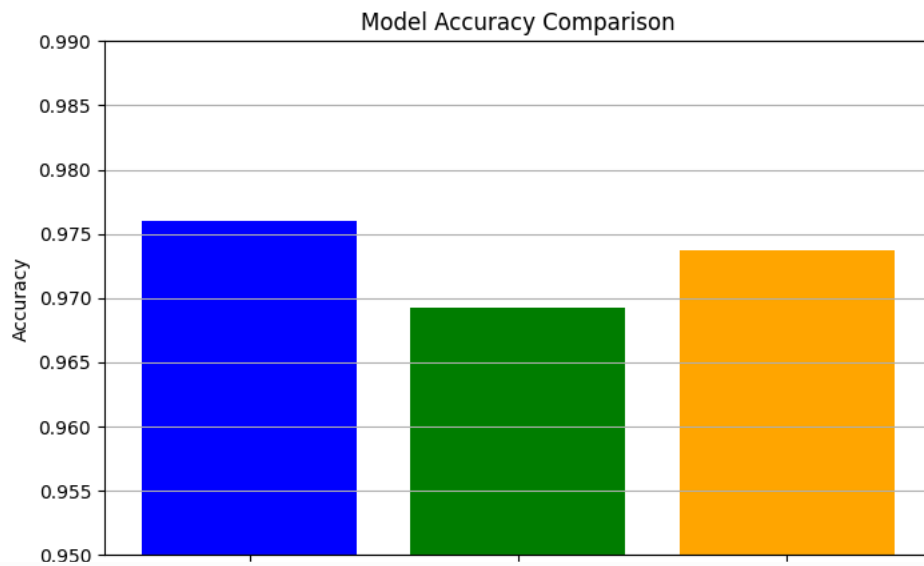
```

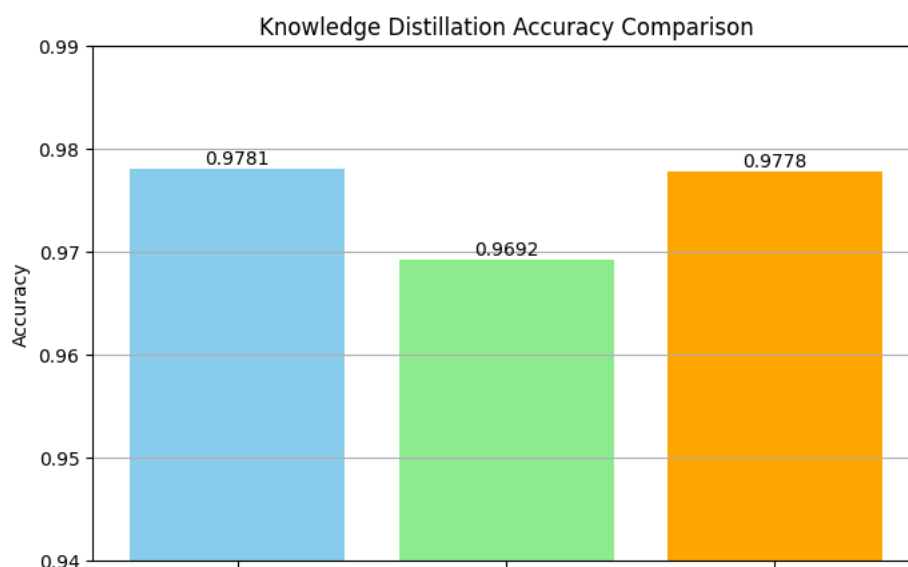
```
1 plot_confusion_matrix(student_scratch, x_test, y_test, title="Student Trained from Scratch")
2
```



```
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
```



```
1 # Use real values from model.evaluate() results
2 teacher_acc = 0.9781
3 distilled_student_acc = 0.9692
4 scratch_student_acc = 0.9778
5
6 # Accuracy bar plot
7 labels = ["Teacher", "Distilled Student", "Scratch Student"]
8 accuracies = [teacher_acc, distilled_student_acc, scratch_student_acc]
9
10 plt.figure(figsize=(8, 5))
11 bars = plt.bar(labels, accuracies, color=["skyblue", "lightgreen", "orange"])
12 plt.ylim(0.94, 0.99)
13 plt.title("Knowledge Distillation Accuracy Comparison")
14 plt.ylabel("Accuracy")
15
16 # Add value labels on bars
17 for bar in bars:
18     height = bar.get_height()
19     plt.text(bar.get_x() + bar.get_width()/2.0, height, f'{height:.4f}',
20             ha='center', va='bottom')
21
22 plt.grid(axis='y')
23 plt.show()
```

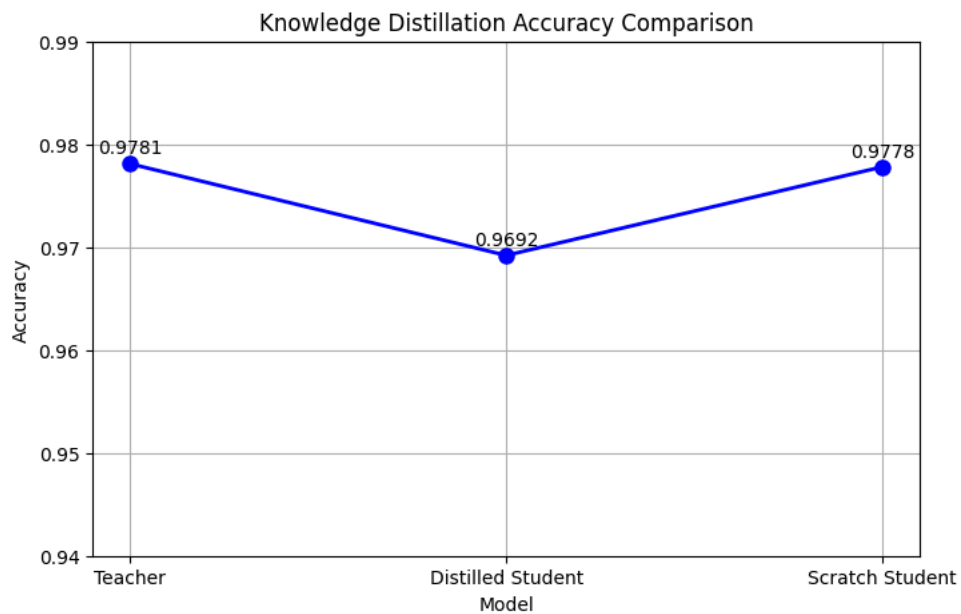


```
1 import matplotlib.pyplot as plt
2
3 # Accuracy values from evaluations
4 teacher_acc = 0.9781
5 distilled_student_acc = 0.9692
6 scratch_student_acc = 0.9778
7
```

```

8 # X and Y values
9 models = ["Teacher", "Distilled Student", "Scratch Student"]
10 accuracies = [teacher_acc, distilled_student_acc, scratch_student_acc]
11
12 # Line plot
13 plt.figure(figsize=(8, 5))
14 plt.plot(models, accuracies, marker='o', linestyle='-', color='blue',
15          linewidth=2, markersize=8)
16
17 # Annotate accuracy values
18 for i, acc in enumerate(accuracies):
19     plt.text(i, acc + 0.001, f"{acc:.4f}", ha='center', fontsize=10)
20
21 plt.ylim(0.94, 0.99)
22 plt.title("Knowledge Distillation Accuracy Comparison")
23 plt.xlabel("Model")
24 plt.ylabel("Accuracy")
25 plt.grid(True)
26 plt.show()

```



team name: SHARPNET

Bhumika KR 1NT22EC036

Deepika P 1NT23CS057

Samitha NS 1NT22EC099