Deep Learning - Training a Simple Convolution Neural Network Model

Transfer Learning via Feature Extraction

Transfer Learning via Fine-Tuning: Training ResNet18 Using a PreTrained ResNet18 Model as the starting point

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader, random_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC

# Step 1: Define directories
train_dir = "/content/drive/MyDrive/TrainingImages/"
test_dir = "/content/drive/MyDrive/TestingImages/"

# Step 2: Define image dimensions, batch size, and number of classes
image_height = 100
image_width = 100
batch_size = 32
num_epochs = 20
num_classes = 10

# Step 3: Define dataset transformation
transform = transforms.Compose([
    transforms.Resize((image_height, image_width)),
    transforms.ToTensor()
])

# Step 4: Define dataset
train_dataset = ImageFolder(train_dir, transform=transform)
test_dataset = ImageFolder(test_dir, transform=transform)

# Step 5: Split dataset into training and validation
train_size = int(0.9 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])

# Step 6: Create data loaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

# Step 7: Define the CNN model architecture
class CNNModel(nn.Module):
    def __init__(self, num_classes):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 3)
        self.conv2 = nn.Conv2d(8, 8, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(8 * 23 * 23, 16)
        self.fc2 = nn.Linear(16, 16)
        self.fc3 = nn.Linear(16, num_classes)

    def forward(self, x):
        x = self.pool(nn.functional.relu(self.conv1(x)))
        x = self.pool(nn.functional.relu(self.conv2(x)))
        x = x.view(-1, 8 * 23 * 23)
        x = nn.functional.relu(self.fc1(x))
        x = nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

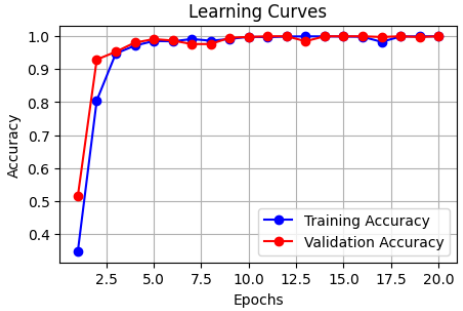
```
# Step 8: Initialize the model, criterion, and optimizer
model = CNNModel(num_classes)  # Fixing num_classes to 10
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Step 9: Train the CNN model for 20 epochs
train_accuracies, val_accuracies = [], []

for epoch in range(num_epochs):
    running_loss = 0.0
    correct_train, total_train = 0, 0
    model.train()
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    train_accuracy = correct_train / total_train
    train_accuracies.append(train_accuracy)

    correct_val, total_val = 0, 0
    model.eval()
    with torch.no_grad():
        for images, labels in val_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

    val_accuracy = correct_val / total_val
    val_accuracies.append(val_accuracy)

    print("Epoch: {}/{}.. ".format(epoch + 1, num_epochs),
          "Training Loss: {:.3f}.. ".format(running_loss / len(train_loader)),
          "Training Accuracy: {:.3f}.. ".format(train_accuracy),
          "Validation Accuracy: {:.3f}".format(val_accuracy))

# Step 10: Plot learning curves
epochs = range(1, num_epochs + 1)
plt.figure(figsize=(5, 3))
plt.plot(epochs, train_accuracies, marker='o', label='Training Accuracy', color='blue')
plt.plot(epochs, val_accuracies, marker='o', label='Validation Accuracy', color='red')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Learning Curves')
plt.grid(True)
plt.legend()
plt.show()

# Step 11: Transfer Learning via Feature Extraction with ResNet18
resnet18 = torchvision.models.resnet18(pretrained=True)
for param in resnet18.parameters():
    param.requires_grad = False

num_ftrs = resnet18.fc.in_features
resnet18.fc = nn.Linear(num_ftrs, num_classes)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(resnet18.parameters(), lr=0.001)

# Step 12: Train the model for 10 epochs
train_accuracies_resnet, val_accuracies_resnet = [], []

for epoch in range(10):
    running_loss = 0.0
    correct_train, total_train = 0, 0
    resnet18.train()
    for images, labels in train_loader:
        optimizer.zero_grad()
```

```
        outputs = resnet18(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    train_accuracy = correct_train / total_train
    train_accuracies_resnet.append(train_accuracy)

    correct_val, total_val = 0, 0
    resnet18.eval()
    with torch.no_grad():
        for images, labels in val_loader:
            outputs = resnet18(images)
            _, predicted = torch.max(outputs.data, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

    val_accuracy = correct_val / total_val
    val_accuracies_resnet.append(val_accuracy)

    print("Epoch: {}/{}.. ".format(epoch + 1, 10),
          "Training Loss: {:.3f}.. ".format(running_loss / len(train_loader)),
          "Training Accuracy: {:.3f}.. ".format(train_accuracy),
          "Validation Accuracy: {:.3f}".format(val_accuracy))

# Step 13: Plot learning curves for ResNet18
epochs_resnet = range(1, 11)
plt.figure(figsize=(5, 3))
plt.plot(epochs_resnet, train_accuracies_resnet, marker='o', label='Training Accuracy', color='blue')
plt.plot(epochs_resnet, val_accuracies_resnet, marker='o', label='Validation Accuracy', color='red')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('ResNet18 Learning Curves')
plt.grid(True)
plt.legend()
plt.show()

# Step 14: Feature Extraction using ResNet18
class FeatureExtractor:
    def __init__(self, model):
        self.model = model
        self.features = None
        self.hook = self.model.layer4.register_forward_hook(self.hook_fn)

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

    def extract(self, x):
        self.model(x)
        return self.features

train_feature_extractor = FeatureExtractor(resnet18)
test_feature_extractor = FeatureExtractor(resnet18)

train_features = []
train_labels = []

test_features = []
test_labels = []

# Extract features and labels from the training dataset
for images, labels in train_loader:
    features_batch = train_feature_extractor.extract(images)
    train_features.append(features_batch.detach().cpu().numpy())
    train_labels.append(labels.numpy())

# Extract features and labels from the testing dataset
for images, labels in test_loader:
    features_batch = test_feature_extractor.extract(images)
    test_features.append(features_batch.detach().cpu().numpy())
    test_labels.append(labels.numpy())
```

```
train_features = np.concatenate(train_features)
train_labels = np.concatenate(train_labels)
test_features = np.concatenate(test_features)
test_labels = np.concatenate(test_labels)

# Step 15: Train SVM prediction model using the features extracted
svm_model = SVC(kernel='rbf', C=10)
svm_model.fit(train_features.reshape(train_features.shape[0], -1), train_labels)

# Step 16: Evaluate SVM model
test_predictions = svm_model.predict(test_features.reshape(test_features.shape[0], -1))

test_accuracy_svm = accuracy_score(test_labels, test_predictions)

print("SVM Test Accuracy:", test_accuracy_svm)

# Step 17: Transfer Learning via Fine-Tuning
# Define the model, criterion, and optimizer for fine-tuning
resnet18_ft = torchvision.models.resnet18(pretrained=True)

# Modify the last layer for the new task
num_ftrs = resnet18_ft.fc.in_features
resnet18_ft.fc = nn.Linear(num_ftrs, num_classes)

# Define criterion and optimizer
criterion_ft = nn.CrossEntropyLoss()
optimizer_ft = optim.Adam(resnet18_ft.parameters(), lr=0.001)

# Train the fine-tuned model for 10 epochs
train_accuracies_ft, val_accuracies_ft = [], []

for epoch in range(10):
    running_loss = 0.0
    correct_train, total_train = 0, 0
    resnet18_ft.train()
    for images, labels in train_loader:
        optimizer_ft.zero_grad()
        outputs = resnet18_ft(images)
        loss = criterion_ft(outputs, labels)
        loss.backward()
        optimizer_ft.step()
        running_loss += loss.item()

        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    train_accuracy = correct_train / total_train
    train_accuracies_ft.append(train_accuracy)

    correct_val, total_val = 0, 0
    resnet18_ft.eval()
    with torch.no_grad():
        for images, labels in val_loader:
            outputs = resnet18_ft(images)
            _, predicted = torch.max(outputs.data, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

    val_accuracy = correct_val / total_val
    val_accuracies_ft.append(val_accuracy)

    print("Epoch: {}/{}.. ".format(epoch + 1, 10),
          "Training Loss: {:.3f}.. ".format(running_loss / len(train_loader)),
          "Training Accuracy: {:.3f}.. ".format(train_accuracy),
          "Validation Accuracy: {:.3f}".format(val_accuracy))

# Plot learning curves for fine-tuning
epochs_ft = range(1, 11)
plt.figure(figsize=(5, 3))
plt.plot(epochs_ft, train_accuracies_ft, marker='o', label='Training Accuracy', color='blue')
plt.plot(epochs_ft, val_accuracies_ft, marker='o', label='Validation Accuracy', color='red')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Fine-Tuning ResNet18 Learning Curves')
plt.grid(True)
```

```
    plt.legend()
    plt.show()
```

```
Epoch: 1/20..  Training Loss: 1.628..  Training Accuracy: 0.346..  Validation Accuracy: 0.514
Epoch: 2/20..  Training Loss: 0.485..  Training Accuracy: 0.805..  Validation Accuracy: 0.929
Epoch: 3/20..  Training Loss: 0.161..  Training Accuracy: 0.948..  Validation Accuracy: 0.953
Epoch: 4/20..  Training Loss: 0.100..  Training Accuracy: 0.971..  Validation Accuracy: 0.981
Epoch: 5/20..  Training Loss: 0.050..  Training Accuracy: 0.986..  Validation Accuracy: 0.991
Epoch: 6/20..  Training Loss: 0.048..  Training Accuracy: 0.985..  Validation Accuracy: 0.987
Epoch: 7/20..  Training Loss: 0.027..  Training Accuracy: 0.992..  Validation Accuracy: 0.976
Epoch: 8/20..  Training Loss: 0.038..  Training Accuracy: 0.987..  Validation Accuracy: 0.976
Epoch: 9/20..  Training Loss: 0.030..  Training Accuracy: 0.992..  Validation Accuracy: 0.996
Epoch: 10/20..  Training Loss: 0.008..  Training Accuracy: 0.999..  Validation Accuracy: 0.998
Epoch: 11/20..  Training Loss: 0.010..  Training Accuracy: 0.997..  Validation Accuracy: 1.000
Epoch: 12/20..  Training Loss: 0.004..  Training Accuracy: 0.999..  Validation Accuracy: 1.000
Epoch: 13/20..  Training Loss: 0.002..  Training Accuracy: 1.000..  Validation Accuracy: 0.985
Epoch: 14/20..  Training Loss: 0.004..  Training Accuracy: 1.000..  Validation Accuracy: 1.000
Epoch: 15/20..  Training Loss: 0.004..  Training Accuracy: 0.999..  Validation Accuracy: 1.000
Epoch: 16/20..  Training Loss: 0.004..  Training Accuracy: 0.999..  Validation Accuracy: 1.000
Epoch: 17/20..  Training Loss: 0.055..  Training Accuracy: 0.983..  Validation Accuracy: 0.998
Epoch: 18/20..  Training Loss: 0.005..  Training Accuracy: 0.999..  Validation Accuracy: 1.000
Epoch: 19/20..  Training Loss: 0.001..  Training Accuracy: 1.000..  Validation Accuracy: 0.998
Epoch: 20/20..  Training Loss: 0.001..  Training Accuracy: 1.000..  Validation Accuracy: 1.000
```




```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The curren
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 118MB/s]
Epoch: 1/10..  Training Loss: 0.429..  Training Accuracy: 0.909..  Validation Accuracy: 1.000
Epoch: 2/10..  Training Loss: 0.045..  Training Accuracy: 0.999..  Validation Accuracy: 1.000
Epoch: 3/10..  Training Loss: 0.022..  Training Accuracy: 1.000..  Validation Accuracy: 1.000
Epoch: 4/10..  Training Loss: 0.015..  Training Accuracy: 1.000..  Validation Accuracy: 1.000
Epoch: 5/10..  Training Loss: 0.010..  Training Accuracy: 1.000..  Validation Accuracy: 1.000
Epoch: 6/10..  Training Loss: 0.008..  Training Accuracy: 1.000..  Validation Accuracy: 1.000
Epoch: 7/10..  Training Loss: 0.009..  Training Accuracy: 1.000..  Validation Accuracy: 1.000
Epoch: 8/10..  Training Loss: 0.007..  Training Accuracy: 1.000..  Validation Accuracy: 1.000
Epoch: 9/10..  Training Loss: 0.007..  Training Accuracy: 0.999..  Validation Accuracy: 1.000
```