EX 4 IMPLEMENTATION OF A CONVOLUTIONAL NEURAL NETWORK (CNN)

DATE: 11/09/2025

Problem Statement:

Implement a Convolutional Neural Network (CNN) from scratch using PyTorch to classify images. Train the network using a dataset of labeled images and evaluate its performance. Additionally, visualize the learned filters in the convolution layers.

Suggested Dataset: CIFAR-10

Objectives:

1. Understand the architecture and functionality of Convolutional Neural Networks (CNNs).
2. Implement CNN layers including convolution, pooling, and fully connected layers.
3. Train the model on the CIFAR-10 dataset and evaluate its performance.
4. Visualize the learned filters to interpret feature extraction at early layers.

Scope:

CNNs are powerful architectures for image classification tasks. This experiment helps students grasp key CNN concepts such as spatial feature learning, hierarchical representation, and how filters learn patterns in images. Visualizing filters bridges the gap between model architecture and interpretability.

Tools and Libraries Used:

1. Python 3.x
2. PyTorch
3. torchvision
4. Matplotlib

Implementation Steps:

Step 1: Load and Preprocess CIFAR-10 Dataset

```
import torchvision.transforms as transforms
import torchvision

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
```

```
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)

classes = trainset.classes
```

Step 2: Define CNN Architecture

```
import torch.nn as nn

class SimpleCNN(nn.Module):
  def __init__(self):
    super(SimpleCNN, self).__init__()
    self.conv1 = nn.Conv2d(3, 16, 3, padding=1)  # Output: 16x32x32
    self.pool = nn.MaxPool2d(2, 2)            # Output: 16x16x16
    self.conv2 = nn.Conv2d(16, 32, 3, padding=1)  # Output: 32x16x16 → 32x8x8 after
pooling
    self.fc1 = nn.Linear(32 * 8 * 8, 128)
    self.fc2 = nn.Linear(128, 10)
    self.relu = nn.ReLU()

  def forward(self, x):
    x = self.pool(self.relu(self.conv1(x)))
    x = self.pool(self.relu(self.conv2(x)))
    x = x.view(-1, 32 * 8 * 8)
    x = self.relu(self.fc1(x))
    x = self.fc2(x)
    return x
```

Step 3: Train the CNN

```
import torch
import torch.optim as optim

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN().to(device)

lossfn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(10):
  running_loss = 0.0
  for inputs, labels in trainloader:
    inputs, labels = inputs.to(device), labels.to(device)

    optimizer.zero_grad()
    outputs = model(inputs)
    loss = lossfn(outputs, labels)
```

```
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss / len(trainloader):.4f}")
```

Step 4: Evaluate Model Performance

```
correct, total = 0, 0
with torch.no_grad():
    for images, labels in testloader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Accuracy on test data: {100 * correct / total:.2f}%")
```

Step 5: Visualize Learned Filters

```
import matplotlib.pyplot as plt

def visualize_filters(layer, n_filters=8):
    filters = layer.weight.data.clone().cpu()
    fig, axs = plt.subplots(1, n_filters, figsize=(15, 4))
    for i in range(n_filters):
        f = filters[i]
        f = (f - f.min()) / (f.max() - f.min())  # Normalize for display
        axs[i].imshow(f.permute(1, 2, 0))
        axs[i].axis('off')
        axs[i].set_title(f'Filter {i+1}')
    plt.tight_layout()
    plt.show()

visualize_filters(model.conv1)
```

Output: