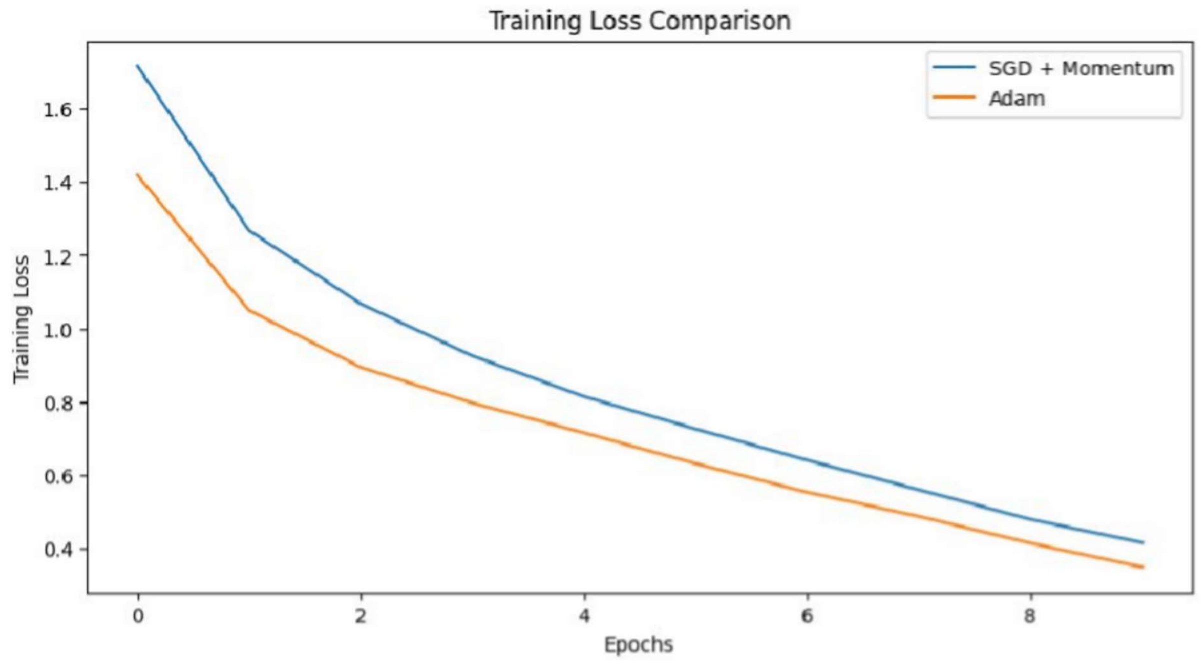
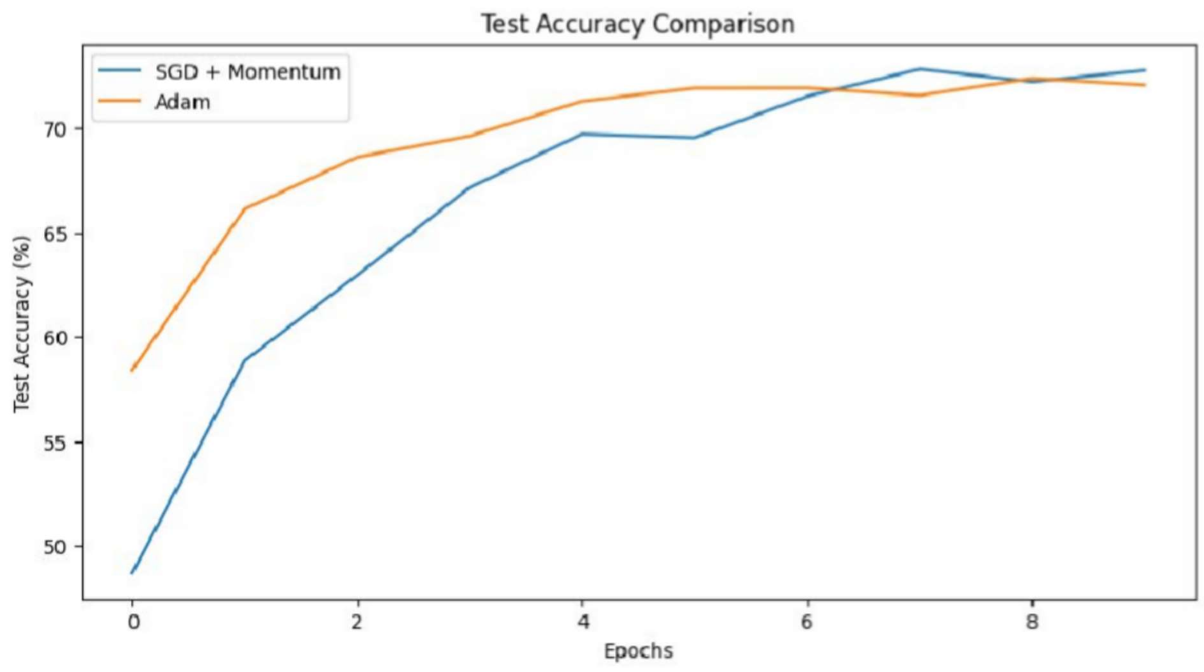
```
        train_losses.append(running_loss / len(trainloader))
        model.eval()
        correct, total = 0, 0

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

        test_accuracies.append(100 * correct / total)
        print(f"Epoch [{epoch+1}/{epochs}] | Loss: {train_losses[-1]:.4f} | Accuracy:
{test_accuracies[-1]:.2f}%")

    return train_losses, test_accuracies



# Train with both optimizers
sgd_losses, sgd_acc = train_model(optimizer_type="sgd", epochs=10, lr=0.01,
momentum=0.9)
adam_losses, adam_acc = train_model(optimizer_type="adam", epochs=10, lr=0.001)

# Plot training loss comparison
plt.figure(figsize=(10,5))
plt.plot(sgd_losses, label='SGD + Momentum')
plt.plot(adam_losses, label='Adam')

plt.xlabel('Epochs'); plt.ylabel('Training Loss')

plt.title('Training Loss Comparison')

plt.legend()

plt.show()


# Plot accuracy comparison

plt.figure(figsize=(10,5))

plt.plot(sgd_acc, label='SGD + Momentum')

plt.plot(adam_acc, label='Adam')

plt.xlabel('Epochs'); plt.ylabel('Test Accuracy (%)')

plt.title('Test Accuracy Comparison')

plt.legend()

plt.show()
```

OUTPUT:





RESULT: The Adam optimizer achieved faster convergence and higher accuracy 72.78% compared to SGD with Momentum 72.08% on CIFAR-10.

EXP NO: 04 DATE: 06/09/2025	**IMPLEMENT CNN FROM SCRATCH**

AIM: To implement a Convolutional Neural Network (CNN) from scratch for image classification using the CIFAR-10 dataset, evaluate its performance, and visualize the learned convolution filters.

ALGORITHM:

- Load the CIFAR-10 dataset and apply normalization preprocessing.
- Define a CNN model with convolution, ReLU, pooling, and fully-connected layers.
- Perform forward propagation to generate class predictions.
- Compute the classification loss using Cross-Entropy Loss.
- Perform backpropagation and update model weights using Adam optimizer.
- Evaluate classification accuracy on the test dataset.
- Visualize the learned filters from the first convolution layer.

CODE:

```
# CNN from scratch on CIFAR-10

import torch, torch.nn as nn, torch.optim as optim

import torch.nn.functional as F

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

import numpy as np

import torchvision


# Device

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


# Data Preprocessing

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))
])
trainset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
testset  = datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader  = DataLoader(testset, batch_size=64, shuffle=False)


# CNN Model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2,2)
        self.fc1 = nn.Linear(64*8*8, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))   # 32x16x16
        x = self.pool(F.relu(self.conv2(x)))   # 64x8x8
        x = x.view(-1, 64*8*8)
        x = F.relu(self.fc1(x))
        return self.fc2(x)

model = SimpleCNN().to(device)
```

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

# Training
for epoch in range(10):
    model.train()
    for imgs, labels in trainloader:
        imgs, labels = imgs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(imgs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
    print(f"Epoch {epoch+1} complete")

# Evaluation
correct, total = 0, 0
model.eval()
with torch.no_grad():
    for imgs, labels in testloader:
        imgs, labels = imgs.to(device), labels.to(device)
        outputs = model(imgs)
        _, pred = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (pred == labels).sum().item()

accuracy = 100 * correct / total
print(f"Test Accuracy: {accuracy:.2f}%")
```