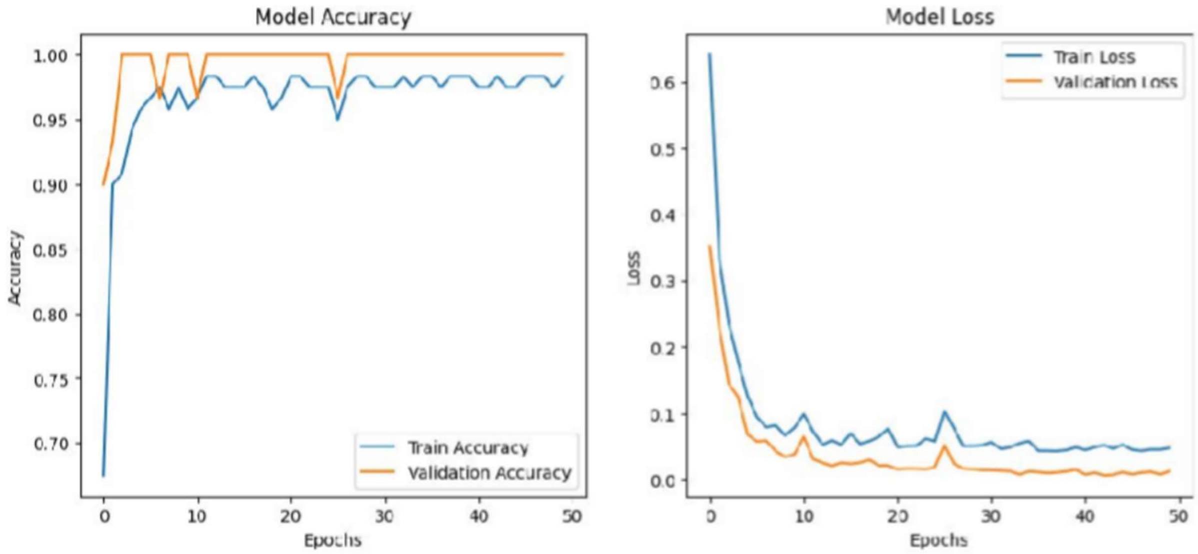
```
plt.show()
```

OUTPUT:



RESULT: The MLP model successfully classified Iris flower species with a test accuracy of 98% and demonstrated good convergence in accuracy and loss graphs.

EXP NO: 03 DATE: 30/08/2025	**SGD WITH MOMENTUM VS ADAM OPTIMIZER**

AIM: To implement a training algorithm using Stochastic Gradient Descent (SGD) with momentum and compare it with the Adam optimizer using the CIFAR-10 dataset by analysing their convergence rates and classification performance.

ALGORITHM:

- Load CIFAR-10 dataset and preprocess images with normalization.
- Define a Simple CNN model for image classification.
- Train the model twice: Using SGD with Momentum and Adam Optimizer
- Perform forward propagation to compute predictions.
- Calculate loss using Cross-Entropy Loss.
- Update weights using selected optimizer.
- Compare both models using: Training Loss Curve, Test Accuracy Curve.

CODE:

```
import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt


# Data Loading & Normalization

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=128, shuffle=True,
num_workers=2)


testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False,
num_workers=2)




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)
        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, 64 * 8 * 8)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

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


def train_model(optimizer_type="sgd", epochs=10, lr=0.01, momentum=0.9):
    model = SimpleCNN().to(device)
    criterion = nn.CrossEntropyLoss()

    if optimizer_type == "sgd":
        optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)
    elif optimizer_type == "adam":
        optimizer = optim.Adam(model.parameters(), lr=lr)

    train_losses = []
    test_accuracies = []

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for inputs, labels in trainloader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
```