```
!pip install pennylane
```

```
Collecting pennylane
  Downloading PennyLane-0.40.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy<2.1 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-pac
Collecting rustworkx>=0.14.0 (from pennylane)
  Downloading rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_
Requirement already satisfied: autograd in /usr/local/lib/python3.11/dist-pac
Collecting tomlkit (from pennylane)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Collecting appdirs (from pennylane)
  Downloading appdirs-1.4.4-py2.py3-none-any.whl.metadata (9.0 kB)
Collecting autoray>=0.6.11 (from pennylane)
  Downloading autoray-0.7.0-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: cachetools in /usr/local/lib/python3.11/dist-p
Collecting pennylane-lightning>=0.40 (from pennylane)
  Downloading PennyLane_Lightning-0.40.0-cp311-cp311-manylinux_2_28_x86_64.wh
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-pa
Collecting diastatic-malt (from pennylane)
  Downloading diastatic_malt-2.15.2-py3-none-any.whl.metadata (2.6 kB)
Collecting scipy-openblas32>=0.3.26 (from pennylane-lightning>=0.40->pennylan
  Downloading scipy_openblas32-0.3.29.0.0-py3-none-manylinux_2_17_x86_64.many
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.1/56.1 kB 2.1 MB/s eta 0:00:
Requirement already satisfied: astunparse in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: gast in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: termcolor in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.1
Requirement already satisfied: six<2.0,>=1.6.1 in /usr/local/lib/python3.11/d
Downloading PennyLane-0.40.0-py3-none-any.whl (2.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 17.8 MB/s eta 0:00:00
Downloading autoray-0.7.0-py3-none-any.whl (930 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 930.0/930.0 kB 32.4 MB/s eta 0:00
Downloading PennyLane_Lightning-0.40.0-cp311-cp311-manylinux_2_28_x86_64.whl
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.4/2.4 MB 36.5 MB/s eta 0:00:00
Downloading rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x8
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 57.4 MB/s eta 0:00:00
Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Downloading diastatic_malt-2.15.2-py3-none-any.whl (167 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 167.9/167.9 kB 9.7 MB/s eta 0:00:
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading scipy_openblas32-0.3.29.0.0-py3-none-manylinux_2_17_x86_64.manyli
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.6/8.6 MB 45.5 MB/s eta 0:00:00
Installing collected packages: appdirs, tomlkit, scipy-openblas32, rustworkx,
Successfully installed appdirs-1.4.4 autoray-0.7.0 diastatic-malt-2.15.2 penn
```

```
import pennylane as qml
from pennylane import numpy as np
```

```
x_original = [1.0, 2.0, 3.0]
y_original = [2.0, 4.0, 6.0]
```

Standardizing the input values to lie in the range -pi to +pi

```
x_scale = [xi*(np.pi/2)/max(x_original) for xi in x_original]
y_scale = [yi*(np.pi/2)/max(y_original) for yi in y_original]


dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def circuit(x, theta):
  qml.RY(x, wires=0)
  qml.RY(theta[0], wires=0) # represents w

  qml.RZ(theta[1], wires=0) # represents b


  return qml.expval(qml.PauliZ(0))



theta = np.array([0.0, 0.0], requires_grad=True)
opt = qml.GradientDescentOptimizer(stepsize=0.1)

for epoch in range(100):
  y_pred = [circuit(xi, theta) for xi in x_scale]

  loss = np.mean((np.array(y_pred) - np.array(y_scale))**2)

  theta = opt.step(lambda t: loss, theta)

  if epoch %10 == 0:
    print(f"Epoch: {epoch}, Loss: {loss}, Theta: {theta}")
```

```
Epoch: 0, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 10, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 20, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 30, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 40, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 50, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 60, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 70, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 80, Loss: 0.9613607519996069, Theta: [0. 0.]
Epoch: 90, Loss: 0.9613607519996069, Theta: [0. 0.]
```

Start coding or generate with AI.

Classification of Cats/Dogs using quantum machine learning

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms, datasets
import pennylane as qml

# ResNet setup
resnet = models.resnet18(pretrained=True)
resnet.fc = nn.Identity()  # Remove FC layer
for param in resnet.parameters():
    param.requires_grad = False

# Quantum setup
n_qubits = 4
n_layers = 2
dev = qml.device("default.qubit", wires=n_qubits)

@qml.qnode(dev, interface="torch")
def quantum_circuit(inputs, weights):
    # Batch-aware angle encoding (no decorator needed)
    for i in range(n_qubits):
        qml.RX(inputs[:, i], wires=i)  # Critical: [batch, feature] indexing

    # Variational layers
    for layer in range(n_layers):
        for i in range(n_qubits):
            qml.Rot(*weights[layer, i], wires=i)
        for i in range(n_qubits):
            qml.CNOT(wires=[i, (i+1) % n_qubits])
```

```
        return [qml.expval(qml.PauliZ(i)) for i in range(n_qubits)]

    weight_shapes = {"weights": (n_layers, n_qubits, 3)}
    qlayer = qml.qnn.TorchLayer(quantum_circuit, weight_shapes)

    # Hybrid model
    class QuantumResNet(nn.Module):
        def __init__(self, num_classes):
            super().__init__()
            self.resnet = resnet
            self.reduce = nn.Linear(512, n_qubits)  # Trainable reduction
            self.quantum = qlayer
            self.fc = nn.Linear(n_qubits, num_classes)

        def forward(self, x):
            features = self.resnet(x)
            features = torch.flatten(features, 1)  # Flatten to [batch, 512]
            reduced = self.reduce(features)  # Shape: [batch, n_qubits]
            quantum_output = self.quantum(reduced)  # Process entire batch
            return self.fc(quantum_output)

    # Training setup remains unchanged
    transform = transforms.Compose([
        transforms.Resize(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])

    train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transf
    test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transf

    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32, shuffle=Fa

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = QuantumResNet(num_classes=10).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # Training loop
    num_epochs = 30
    for epoch in range(num_epochs):
        model.train()
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

        # Validation
        model.eval()
        correct, total = 0, 0
        with torch.no_grad():
```

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

    print(f"Epoch {epoch+1}/{num_epochs}, Accuracy: {100 * correct / total:.2f}%"
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: Use
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: Use
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /
100%|██████████| 44.7M/44.7M [00:00<00:00, 94.3MB/s]
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data
100%|██████████| 170M/170M [00:02<00:00, 74.0MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
Epoch 1/30, Accuracy: 46.85%
Epoch 2/30, Accuracy: 52.85%
Epoch 3/30, Accuracy: 60.23%
```

```
torch.save(model.state_dict(), "quantum_resnet.pth")
```

```
transform = transforms.Compose([
    transforms.Resize(224),          # Resize to 224x224 (ResNet requirement)
    transforms.ToTensor(),           # Convert to tensor
    transforms.Normalize(            # Use same normalization as training
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

```
model = QuantumResNet(num_classes=10).to(device)
model.load_state_dict(torch.load("quantum_resnet.pth"))  # Load saved weights
model.eval()  # Set to evaluation mode
```

```
from PIL import Image

# Load and preprocess an image
image = Image.open("your_image.jpg")  # Replace with your image path
input_tensor = transform(image).unsqueeze(0).to(device)  # Add batch dimension

# Make prediction
with torch.no_grad():
    outputs = model(input_tensor)
    probabilities = torch.nn.functional.softmax(outputs, dim=1)
    predicted_class = torch.argmax(probabilities, dim=1).item()

# Map class index to CIFAR-10 label
```

```
cifar10_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                   'dog', 'frog', 'horse', 'ship', 'truck']
print(f"Predicted class: {cifar10_classes[predicted_class]}")


model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        preds = torch.argmax(outputs, dim=1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate accuracy
accuracy = (np.array(all_preds) == np.array(all_labels)).mean()
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```