```
import h5py
import numpy as np
from sklearn.model_selection import train_test_split
import torch
from torch.utils.data import TensorDataset, random_split
import torch.nn as nn
import torch.nn.functional as F
```

```
!wget --no-check-certificate "https://cernbox.cern.ch/remote.php/dav/public-files/AtBT8y4MiQYFcgc/SinglePhotonPt50_IMGCROPS_n249k
!wget --no-check-certificate "https://cernbox.cern.ch/remote.php/dav/public-files/FbXw3V4XNyYB3oA/SingleElectronPt50_IMGCROPS_n24
```

Show hidden output

```
def load_dataset(path):
    with h5py.File(path, "r") as f:
        X = f['X'][:]
        y = f['y'][:]
    return X, y


X_p, y_p = load_dataset("SinglePhoton.hdf5")
X_e, y_e = load_dataset("SingleElectron.hdf5")


X = np.concatenate([X_p[:150000], X_e[:150000]], axis=0)
y = np.concatenate([y_p[:150000], y_e[:150000]], axis=0)


mean = np.mean(X, axis=(0, 1, 2))
std = np.std(X, axis=(0, 1, 2))
X = (X - mean) / std


print(X.shape, y.shape)
```

```
(300000, 32, 32, 2) (300000,)
```

```
X_tensor = torch.tensor(X, dtype=torch.float32).permute(0, 3, 1, 2)  # (N, C, H, W)
y_tensor = torch.tensor(y, dtype=torch.long)

dataset = TensorDataset(X_tensor, y_tensor)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size

train_dataset, test_dataset = random_split(dataset, [train_size, test_size])


train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64)


class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, 3, stride, 1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, 3, 1, 1)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, 1, stride),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        return F.relu(out)

class ResNet15(nn.Module):
    def __init__(self):
```

```
            super().__init__()
            self.layer1 = BasicBlock(2, 16)
            self.layer2 = BasicBlock(16, 32, stride=2)
            self.layer3 = BasicBlock(32, 64, stride=2)
            self.layer4 = BasicBlock(64, 128, stride=2)
            self.layer5 = BasicBlock(128, 128)
            self.layer6 = BasicBlock(128, 128)
            self.pool = nn.AdaptiveAvgPool2d((1, 1))
            self.dropout = nn.Dropout(0.5)
            self.fc = nn.Linear(128, 2)

        def forward(self, x):
            x = self.layer1(x)
            x = self.layer2(x)
            x = self.layer3(x)
            x = self.layer4(x)
            x = self.layer5(x)
            x = self.layer6(x)
            x = self.pool(x)
            x = torch.flatten(x, 1)
            x = self.dropout(x)
            return self.fc(x)


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


    device
```

```
device(type='cuda')
```

```
    model = ResNet15().to(device)

    optimizer = torch.optim.Adam(model.parameters(), lr=0.002, weight_decay=1e-4)
    criterion = nn.CrossEntropyLoss()

    # 训练过程
    for epoch in range(100):
        model.train()
        total_loss = 0
        correct = 0
        for X_batch, y_batch in train_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)

            optimizer.zero_grad()
            outputs = model(X_batch)
            loss = criterion(outputs, y_batch)
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            pred = outputs.argmax(dim=1)
            correct += (pred == y_batch).sum().item()

        acc = correct / len(train_dataset)
        print(f"Epoch {epoch+1}, Loss: {total_loss:.4f}, Train Acc: {acc:.4f}")
```

```
Epoch 65, Loss: 2049.0884, Train Acc: 0.7314
Epoch 66, Loss: 2047.6722, Train Acc: 0.7301
Epoch 67, Loss: 2048.6692, Train Acc: 0.7304
Epoch 68, Loss: 2045.1367, Train Acc: 0.7311
Epoch 69, Loss: 2045.2457, Train Acc: 0.7313
Epoch 70, Loss: 2045.5745, Train Acc: 0.7312
Epoch 71, Loss: 2045.6029, Train Acc: 0.7313
Epoch 72, Loss: 2047.2719, Train Acc: 0.7314
Epoch 73, Loss: 2045.8736, Train Acc: 0.7320
Epoch 74, Loss: 2050.2165, Train Acc: 0.7308
Epoch 75, Loss: 2049.2465, Train Acc: 0.7306
Epoch 76, Loss: 2044.4619, Train Acc: 0.7312
Epoch 77, Loss: 2043.8266, Train Acc: 0.7316
Epoch 78, Loss: 2046.1338, Train Acc: 0.7312
Epoch 79, Loss: 2044.0491, Train Acc: 0.7308
Epoch 80, Loss: 2045.9406, Train Acc: 0.7313
Epoch 81, Loss: 2045.2247, Train Acc: 0.7313
Epoch 82, Loss: 2045.0367, Train Acc: 0.7317
Epoch 83, Loss: 2046.8210, Train Acc: 0.7308
Epoch 84, Loss: 2045.8484, Train Acc: 0.7319
Epoch 85, Loss: 2045.8704, Train Acc: 0.7311
Epoch 86, Loss: 2044.5332, Train Acc: 0.7315
Epoch 87, Loss: 2046.0285, Train Acc: 0.7315
Epoch 88, Loss: 2045.4932, Train Acc: 0.7307
Epoch 89, Loss: 2048.4170, Train Acc: 0.7305
Epoch 90, Loss: 2044.8641, Train Acc: 0.7319
Epoch 91, Loss: 2044.8397, Train Acc: 0.7321
Epoch 92, Loss: 2045.9035, Train Acc: 0.7316
Epoch 93, Loss: 2045.4703, Train Acc: 0.7318
Epoch 94, Loss: 2045.6702, Train Acc: 0.7311
Epoch 95, Loss: 2045.2959, Train Acc: 0.7308
Epoch 96, Loss: 2044.7267, Train Acc: 0.7315
Epoch 97, Loss: 2047.7539, Train Acc: 0.7316
Epoch 98, Loss: 2043.7364, Train Acc: 0.7318
Epoch 99, Loss: 2046.4649, Train Acc: 0.7311
Epoch 100, Loss: 2044.9058, Train Acc: 0.7322
```

```
model.eval()
correct = 0
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        outputs = model(X_batch)
        pred = outputs.argmax(dim=1)
        correct += (pred == y_batch).sum().item()

test_acc = correct / len(test_dataset)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
Test Accuracy: 0.7272
```