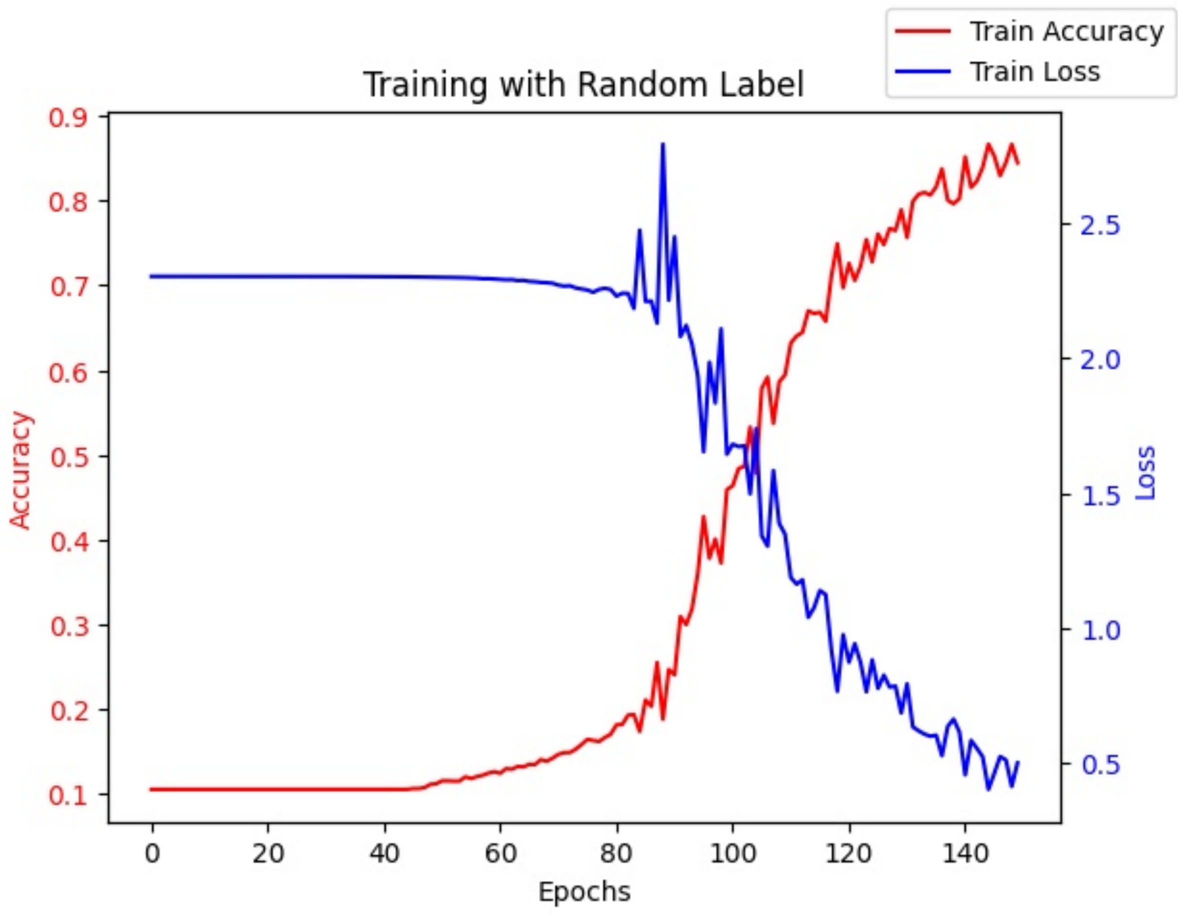
```
import torch
import torch.nn as nn
import time
import numpy as np
from torchvision import datasets
from torchvision.transforms import transforms
from torch.utils.data import Subset
import matplotlib.pyplot as plt

np.random.seed(42)
torch.manual_seed(42)

# Make sure to use only 10% of the available MNIST data.
# Otherwise, experiment will take quite long (around 90 minutes).

# (Modified version of AlexNet)
class AlexNet(nn.Module):
    def __init__(self, num_class=10):
        super(AlexNet, self).__init__()

        self.conv_layer1 = nn.Sequential(
            nn.Conv2d(1, 96, kernel_size=4),
            nn.ReLU(inplace=True),
            nn.Conv2d(96, 96, kernel_size=3),
            nn.ReLU(inplace=True)
        )
        self.conv_layer2 = nn.Sequential(
            nn.Conv2d(96, 256, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2)
        )
        self.conv_layer3 = nn.Sequential(
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2)
        )

        self.fc_layer1 = nn.Sequential(
            nn.Dropout(),
            nn.Linear(6400, 800),
            nn.ReLU(inplace=True),
            nn.Linear(800, 10)
        )

    def forward(self, x):
        output = self.conv_layer1(x)
        output = self.conv_layer2(output)
        output = self.conv_layer3(output)
        output = torch.flatten(output, 1)
```

```
        output = self.fc_layer1(output)
        return output


learning_rate = 0.1
batch_size = 64
epochs = 150

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = AlexNet().to(device)
loss_function = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

### here ###
train_dataset = datasets.MNIST(root='~/Downloads/mnist_data/',
                               train=True,
                               transform=transforms.ToTensor(),)

train_dataset = Subset(train_dataset,sorted(torch.randint(low=1, high=60001, size=(6000,), dtype=torch.int)))

new_dataset = []

for i in range(len(train_dataset)):
    image, label = train_dataset[i]
    new_dataset.append((image, np.random.randint(0,10)))

train_loader = torch.utils.data.DataLoader(dataset = new_dataset,batch_size=batch_size)
test_loader = torch.utils.data.DataLoader(dataset = new_dataset,    ### only change batch_size
                                          batch_size=batch_size*8)


### here ###

tick = time.time()

loss_list =[]
accuracy_list =[]

for epoch in range(epochs):
    print(f"\nEpoch {epoch + 1} / {epochs}")
    model.train()
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        loss = loss_function(model(images), labels)
        loss.backward()

        optimizer.step()

    model.eval()
    running_loss = 0.0
```

```
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = loss_function(outputs, labels)
            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    loss_list.append(running_loss / len(test_loader))
    accuracy_list.append(correct / total)

fig, ax1 = plt.subplots()

ax1.plot(accuracy_list, label="Train Accuracy", color='r')
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Accuracy", color='r')
ax1.tick_params(axis='y', labelcolor='r')

ax2 = ax1.twinx()

ax2.plot(loss_list, label="Train Loss", color='b')
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Loss", color='b')
ax2.tick_params(axis='y', labelcolor='b')

plt.title('Training with Random Label')
fig.legend()
plt.show()

tock = time.time()
print(f"Total training time: {tock - tick}")
```


Train Accuracy
Train Loss
Training with Random Label
Accuracy
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Loss
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
120
140
Epochs