1. Example of calculation accuracy and loss multi-label audio classification using PyTorch

```
import torch
import torch.nn.functional as F

def calculate_accuracy_and_loss(model, data_loader, device):
    model.eval()
    correct = 0
    total = 0
    total_loss = 0
    with torch.no_grad():
        for batch_idx, (data, target) in enumerate(data_loader):
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss = F.binary_cross_entropy_with_logits(output, target)
            total_loss += loss.item()
            predicted = torch.sigmoid(output) > 0.5
            correct += (predicted == target).sum().item()
            total += target.size(0)
    accuracy = 100. * correct / total
    average_loss = total_loss / len(data_loader)
    return accuracy, average_loss

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = MyModel()
model.to(device)
data_loader = DataLoader(...)
accuracy, loss = calculate_accuracy_and_loss(model, data_loader, device)
```

2. How to calculate accuracy and loss of multilabel audio classification using PyTorch in different train and test loop functions please give me an example

```
import torch
import torch.nn.functional as F

def calculate_batch_accuracy_and_loss(model, data, target, device):
    model.eval()
    data, target = data.to(device), target.to(device)
    output = model(data)
    loss = F.binary_cross_entropy_with_logits(output, target)
    predicted = torch.sigmoid(output) > 0.5
    correct = (predicted == target).sum().item()
```

```
    accuracy = 100. * correct / target.size(0)
    return accuracy, loss.item()

def train(model, train_loader, optimizer, device):
    model.train()
    train_loss = 0
    for data, target in train_loader:
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.binary_cross_entropy_with_logits(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    return train_loss / len(train_loader)

def test(model, test_loader, device):
    model.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            accuracy, loss = calculate_batch_accuracy_and_loss(model, data, target, device)
            test_loss += loss
            correct += accuracy * target.size(0) / 100.
            total += target.size(0)
    test_loss /= len(test_loader)
    test_accuracy = 100. * correct / total
    return test_loss, test_accuracy

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = MyModel()
model.to(device)
optimizer = torch.optim.Adam(model.parameters())
train_loader = DataLoader(...)
test_loader = DataLoader(...)
for epoch in range(num_epochs):
    train_loss = train(model, train_loader, optimizer, device)
    test_loss, test_accuracy = test(model, test_loader, device)
    print('Epoch {} - Train Loss: {:.4f} - Test Loss: {:.4f} - Test Accuracy: {:.2f}%
```

```
# Calculating train and test loss and accuracy

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader

class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.pool = nn.MaxPool2d(2)
        self.fc1 = nn.Linear(64 * 11 * 11, 128)
        self.fc2 = nn.Linear(128, 10)
        self.fc3 = nn.Linear(10, 6)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 11 * 11)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

def train(model, train_loader, criterion, optimizer, device):
    model.train()
    train_loss = 0
    correct = 0
    total = 0
    for data, target in train_loader:
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        predicted = torch.sigmoid(output) > 0.5
        correct += (predicted == target).sum().item()
        total += target.size(0)
    train_loss /= len(train_loader)
```

```
    train_accuracy = 100. * correct / total
    return train_loss, train_accuracy

def test(model, test_loader, criterion, device):
    model.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss = criterion(output, target)
            test_loss += loss.item()
            predicted = torch.sigmoid(output) > 0.5
            correct += (predicted == target).sum().item()
            total += target.size(0)
    test_loss /= len(test_loader)
    test_accuracy = 100. * correct / total
    return test_loss, test_accuracy

# Sample usage
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
train_dataset = MyDataset(...)
test_dataset = MyDataset(...)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
model = MyModel()
model.to(device)
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
for epoch in range(num_epochs):
    train_loss, train_accuracy = train(model, train_loader, criterion, optimizer, device)
    test_loss, test_accuracy = test(model, test_loader, criterion, device)
    print(f"Epoch {epoch+1} - Train Loss: {train_loss:.4f} - Train Accuracy: {train_accuracy:.2f}% -
Test Loss: {test_loss:.4f} - Test Accuracy: {test_accuracy:.2f}%")
```