```python
import re
import torch
from torch.nn import Module, CrossEntropyLoss
from torch.utils.data import DataLoader, Dataset
from torch.optim import Adam


def train(
,    model: Module, loss_fn: CrossEntropyLoss, optimizer: Adam, dataloader: DataLoader
,) -> tuple:
    """Train a given model.

    Args:
        model (Module): model.
        loss_fn (CrossEntropyLoss): loss function.
        optimizer (Adam): optimizer.
        dataloader (DataLoader): dataloader.

    Returns:
        tuple: loss, accuracy, model.
    """
    # Keep tracking for y and y_pred to calculate final loss and accuracy
    all_y = None
    all_y_pred = None

    # Perform batch training
    for _, (X, y) in enumerate(dataloader):

        # Forward Propagation
        y_pred = model(X)

        # Calculate loss
        loss = loss_fn(y_pred, y)

        # Calculate gradients
        loss.backward()

        # Update weights
        optimizer.step()

        # Clearn gradients in the optimizer
        optimizer.zero_grad()

        # Store y and y_pred of this batch
        with torch.no_grad():
            all_y = y if all_y == None else torch.cat((all_y, y))
            all_y_pred = (
                y_pred if all_y_pred == None else torch.cat((all_y_pred, y_pred))
            )

    # Calculate loss and accuracy
    loss, acc = calculate_loss_accuracy(all_y_pred, all_y, loss_fn)

    return loss, acc, model


def test(model: Module, loss_fn: CrossEntropyLoss, dataset: Dataset) -> tuple:
    """Test a given model.

    Args:
        model (Module): model.
        loss_fn (CrossEntropyLoss): loss function.
        dataset (Dataset): dataset.

    Returns:
        tuple: loss, accuracy, model
    """
    # Use no_grad to freeze the model.
    with torch.no_grad():
        X = dataset.X
        y = dataset.y

        # Forward Propagation
        y_pred = model(X)

        # Calculate loss and accuracy
        loss, acc = calculate_loss_accuracy(y_pred, y, loss_fn)

    return loss, acc, model


def calculate_loss_accuracy(
,    y_pred: torch.Tensor, y: torch.tensor, loss_fn: CrossEntropyLoss
,) -> tuple:
    """Calculate loss and accuracy with given labels and predicted labels.

    Args:
        y_pred (torch.Tensor): predicted labels.
        y (torch.tensor): true labels.
        loss_fn (CrossEntropyLoss): loss function.

    Returns:
        tuple: loss, accuracy
    """
    acc = torch.sum(torch.argmax(y_pred, 1) == y) / y.size()[0]
    loss = loss_fn(y_pred, y).item()
    return loss, acc
```