In [ ]:
```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, models, transforms
import torchvision.transforms as T
```

Set the device to CPU

In [ ]:
```
device = torch.device("cpu")
```

Data transformations for training and validation

In [ ]:
```
data_transforms = {
    'train': T.Compose([
        T.Resize((224, 224)),
        T.RandomHorizontalFlip(),
        T.ToTensor(),
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': T.Compose([
        T.Resize((224, 224)),
        T.ToTensor(),
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = '/content/drive/MyDrive/GreenOX'
```

Create datasets for training and validation

In [ ]:
```
image_datasets = {
    x: datasets.ImageFolder(os.path.join(data_dir, x), data_transforms[x])
    for x in ['train', 'val']
}
```

In [ ]:
```
batch_size = 8
```

Create data loaders for training and validation

In [ ]:
```
data_loaders = {
    x: DataLoader(image_datasets[x], shuffle=True, batch_size=batch_size)
    for x in ['train', 'val']
}
```

In [ ]:
```
class PetBottleDetector(nn.Module):
    def __init__(self):
```

```
        super(PetBottleDetector, self).__init__()
        self.model = models.resnet18(pretrained=True)
        self.model.fc = nn.Sequential(
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 2)  # 2 output units for binary classification
        )

    def forward(self, x):
        x = self.model(x)
        return x
```

In [ ]:
```
model = PetBottleDetector()
model = model.to(device)
```

Define a function for the training loop

In [ ]:
```
def train_model(model, criterion, optimizer, num_epochs=10):
    for epoch in range(num_epochs):
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in data_loaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(image_datasets[phase])
```

In [ ]:
```
num_epochs = 10
```

In [ ]:
```
train_model(model, criterion, optimizer, num_epochs)
```

Define a function for the training loop

In [ ]:
```
model.eval()
correct = 0
total = 0

with torch.no_grad():
```

```
    for inputs, labels in data_loaders['val']:
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```

In [ ]:

```
accuracy = 100 * correct / total
print(f'Accuracy on the validation set: {accuracy:.2f}%')
```