In [ ]:
```
import os
import csv
import shutil
from sklearn.model_selection import train_test_split

# load labels.csv for train dataset
with open('dataset/train/labels.csv', 'r') as f:
    reader = csv.reader(f)
    train_data = list(reader)

# split train data into train and validation sets
train_data, valid_data = train_test_split(train_data, test_size=0.2, random_state=

# create valid directory
os.makedirs('dataset/valid/images')
os.makedirs('dataset/valid/annotations')

# copy images and annotations to valid directory
for data in valid_data:
    filename = data[0]
    label = data[1]
    src_img = os.path.join('dataset/train/images', filename)
    dst_img = os.path.join('dataset/valid/images', filename)
    src_ann = os.path.join('dataset/train/annotations', filename[:-4] + '.xml')
    dst_ann = os.path.join('dataset/valid/annotations', filename[:-4] + '.xml')
    shutil.copy(src_img, dst_img)
    shutil.copy(src_ann, dst_ann)

# write labels.csv for valid dataset
with open('dataset/valid/labels.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['filename', 'label'])
    for data in valid_data:
        writer.writerow(data)
```

In [ ]:
```
import os
import csv
from PIL import Image
import torch
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms

# define transforms to apply to the images
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

class MyDataset(Dataset):
    def __init__(self, root, csv_file, transforms=None):
        self.root = root
        self.transforms = transforms
        self.imgs = []
        self.class_names = set()

        # read the CSV file and extract the image paths and bounding boxes
        with open(csv_file, 'r') as f:
            reader = csv.DictReader(f)
            for row in reader:
                img_path = os.path.join(root, 'images', row['filename'])
                bbox = (int(row['xmin']), int(row['ymin']), int(row['xmax']), int(
```

```
                class_name = row['label']

                # add to list of images and set of class names
                self.imgs.append((img_path, bbox, class_name))
                self.class_names.add(class_name)

    def __getitem__(self, idx):
        # load image and crop to bounding box
        img_path, bbox, class_name = self.imgs[idx]
        img = Image.open(img_path).convert('RGB')
        img = img.crop(bbox)

        # apply transforms if specified
        if self.transforms is not None:
            img = self.transforms(img)

        return img, class_name

    def __len__(self):
        return len(self.imgs)

    def num_classes(self):
        return len(self.class_names)


train_dataset = MyDataset('dataset/train', 'dataset/train/labels.csv', transforms=
valid_dataset = MyDataset('dataset/valid', 'dataset/valid/labels.csv', transforms=
test_dataset = MyDataset('dataset/test', 'dataset/test/labels.csv', transforms=tra

num_classes = train_dataset.num_classes()
print(f"Number of classes: {num_classes}")
print(train_dataset.class_names)
class_names =train_dataset.class_names

# For the train dataset
num_train_images = len(train_dataset)
num_train_classes = train_dataset.num_classes()
print(f"Number of train images: {num_train_images}")
print(f"Number of train classes: {num_train_classes}")

# For the validation dataset
num_valid_images = len(valid_dataset)
num_valid_classes = valid_dataset.num_classes()
print(f"Number of valid images: {num_valid_images}")
print(f"Number of valid classes: {num_valid_classes}")

# For the test dataset
num_test_images = len(test_dataset)
num_test_classes = test_dataset.num_classes()
print(f"Number of test images: {num_test_images}")
print(f"Number of test classes: {num_test_classes}")

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

# create train, validation, and test data loaders
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```
Number of classes: 9
{'D11', 'D10', 'D44', 'D00', 'D01', 'D50', 'D43', 'D20', 'D40'}
Number of train images: 25513
Number of train classes: 9
Number of valid images: 5103
Number of valid classes: 9
Number of test images: 6366
Number of test classes: 9
```

Yes, it appears that the data has been loaded into PyTorch tensors.

The output you provided shows the shape of the tensors for the first batch of data and labels in both the train and test loaders. The data tensors have a shape of torch.Size([32, 3, 224, 224]), which indicates that there are 32 images in the batch, each with 3 color channels (RGB), and each image is 224x224 pixels. The labels tensor has a shape of (32,), which means it is a one-dimensional tensor with 32 elements, corresponding to the labels for each image in the batch.

In [ ]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models

# Set device to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


# Define the number of epochs
num_epochs = 10

# Define class names and create a class-to-index mapping dictionary
class_names = ['D00', 'D01', 'D10', 'D11', 'D20', 'D40', 'D43', 'D44', 'D50']
class_to_idx = {class_name: i for i, class_name in enumerate(class_names)}


# Define the model and the optimizer
model = models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(class_names))
model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
# Train the model
for epoch in range(num_epochs):
    print()
    print(f'Epoch {epoch+1}/{num_epochs}')
    print('-' * 10)

    # Set to training mode
    model.train()

    train_loss = 0.0

    # Iterate over data
    for inputs, labels in train_loader:
        inputs = inputs.to(device)
        labels = torch.tensor([class_to_idx[label] for label in labels]).to(device
```

```
        # Zero the parameter gradients
        optimizer.zero_grad()

        print(str(epoch),end=" ")
        # Forward pass
        with torch.set_grad_enabled(True):
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)


            # Backward pass
            loss.backward()
            optimizer.step()

        # Statistics
        train_loss += loss.item() * inputs.size(0)

    # Calculate average loss
    train_loss /= len(train_loader.dataset)
    print()
    print(f'Training Loss: {train_loss:.4f}')

    # Set to evaluation mode
    model.eval()

    valid_loss = 0.0
    valid_acc = 0.0

    # Iterate over data
    for inputs, labels in valid_loader:
        inputs = inputs.to(device)
        labels = torch.tensor([class_to_idx[label] for label in labels]).to(device

        # Forward pass
        with torch.set_grad_enabled(False):
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)

        # Statistics
        valid_loss += loss.item() * inputs.size(0)
        valid_acc += torch.sum(preds == labels.data)

    # Calculate average loss and accuracy
    valid_loss /= len(valid_loader.dataset)
    valid_acc = valid_acc.double() / len(valid_loader.dataset)
    print(f'Validation Loss: {valid_loss:.4f}, Validation Accuracy: {valid_acc:.4f

    # Save model for current epoch
    torch.save(model.state_dict(),f'model_{epoch+1}.pth')
```

```
Epoch 1/10
----------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Training Loss: 0.6402
Validation Loss: 0.3535, Validation Accuracy: 0.8814

Epoch 2/10
----------
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Training Loss: 0.3802
Validation Loss: 0.2489, Validation Accuracy: 0.9212

Epoch 3/10
----------
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Training Loss: 0.2841
Validation Loss: 0.1620, Validation Accuracy: 0.9477

Epoch 4/10
----------
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
Training Loss: 0.2149
Validation Loss: 0.1227, Validation Accuracy: 0.9622

Epoch 5/10
----------
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
Training Loss: 0.1595
Validation Loss: 0.0739, Validation Accuracy: 0.9786

Epoch 6/10
----------
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

```
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
Training Loss: 0.1168
Validation Loss: 0.0534, Validation Accuracy: 0.9839

Epoch 7/10
----------
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
Training Loss: 0.0809
Validation Loss: 0.0300, Validation Accuracy: 0.9937

Epoch 8/10
----------
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
```

```
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
Training Loss: 0.0630
Validation Loss: 0.0302, Validation Accuracy: 0.9908

Epoch 9/10
----------
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
Training Loss: 0.0501
Validation Loss: 0.0236, Validation Accuracy: 0.9937

Epoch 10/10
----------
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[9], line 53
     49     _, preds = torch.max(outputs, 1)
     52     # Backward pass
---> 53     loss.backward()
     54     optimizer.step()
     56 # Statistics

File F:\anaconda3\lib\site-packages\torch\_tensor.py:487, in Tensor.backward(self,
gradient, retain_graph, create_graph, inputs)
    477 if has_torch_function_unary(self):
    478     return handle_torch_function(
    479         Tensor.backward,
    480         (self,),
   (...)
    485         inputs=inputs,
    486     )
--> 487 torch.autograd.backward(
    488     self, gradient, retain_graph, create_graph, inputs=inputs
    489 )

File F:\anaconda3\lib\site-packages\torch\autograd\__init__.py:200, in backward(te
nsors, grad_tensors, retain_graph, create_graph, grad_variables, inputs)
    195     retain_graph = create_graph
    197 # The reason we repeat same the comment below is that
    198 # some Python versions print out the first line of a multi-line function
    199 # calls in the traceback and some print out the last line
--> 200 Variable._execution_engine.run_backward(  # Calls into the C++ engine to r
un the backward pass
    201     tensors, grad_tensors_, retain_graph, create_graph, inputs,
    202     allow_unreachable=True, accumulate_grad=True)

KeyboardInterrupt:
```

In [ ]:
```
model
```

Out[ ]:

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=F
alse)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=F
alse)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
```

```
bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=9, bias=True)
)
```

In [ ]:

```
import torch
import torchvision.models as models
import torch
import torch.nn as nn
import torch.optim as optim
```

```
from torchvision import models

# Set up the device for running the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# Define class names and create a class-to-index mapping dictionary
class_names = ['D00', 'D01', 'D10', 'D11', 'D20', 'D40', 'D43', 'D44', 'D50']
class_to_idx = {class_name: i for i, class_name in enumerate(class_names)}
# Load the model
model = models.resnet18(pretrained=False)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(class_names))
model.load_state_dict(torch.load('F:/venkatesh/Resnet/model_8.pth', map_location=d
model.eval()

# Define the transform for the input image
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])


# Evaluate the model on the testing dataset
correct = 0
total = 0
print("testing on test data")
for data in test_loader:

    images, labels = data
    #print(labels,end=" ")
    images = images.to(device)
    labels = torch.tensor([class_to_idx[label] for label in labels]).to(device)
    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()
    print(correct,end=" ")


# Print the accuracy on the testing dataset
print('Accuracy on the testing dataset: %d %%' % (100 * correct / total))
```

```
F:\anaconda3\lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The
parameter 'pretrained' is deprecated since 0.13 and may be removed in the future,
please use 'weights' instead.
  warnings.warn(
F:\anaconda3\lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Argu
ments other than a weight enum or `None` for 'weights' are deprecated since 0.13 a
nd may be removed in the future. The current behavior is equivalent to passing `we
ights=None`.
  warnings.warn(msg)
```

```
testing on test data
27 55 84 112 140 169 198 228 258 285 313 341 371 399 424 452 481 509 538 567 595 6
21 650 676 702 728 756 780 807 836 862 893 924 944 974 1000 1026 1053 1081 1107 11
34 1162 1190 1215 1244 1272 1295 1323 1351 1376 1404 1431 1457 1483 1510 1539 1564
1593 1621 1644 1669 1697 1726 1754 1784 1814 1845 1873 1903 1932 1961 1991 2021 20
52 2079 2107 2138 2167 2197 2226 2256 2283 2312 2339 2366 2395 2423 2453 2481 2512
2542 2573 2601 2632 2661 2691 2722 2751 2779 2808 2837 2867 2896 2925 2951 2978 30
06 3036 3068 3094 3124 3153 3182 3211 3236 3264 3294 3321 3350 3380 3410 3441 3468
3498 3527 3556 3585 3615 3641 3671 3699 3727 3757 3787 3819 3847 3876 3904 3934 39
65 3995 4025 4054 4079 4108 4138 4166 4194 4226 4254 4280 4310 4341 4371 4401 4430
4461 4489 4517 4544 4574 4603 4632 4663 4694 4724 4756 4787 4819 4848 4879 4909 49
39 4971 5001 5032 5064 5095 5126 5147 5174 5198 5227 5257 5287 5316 5345 5373 5404
5429 5460 5488 5517 5545 5573 5602 5628 5654 5682 Accuracy on the testing dataset:
89 %
```

In [ ]:

```python
from sklearn.metrics import confusion_matrix
import torch
import torchvision.models as models
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models

# Set up the device for running the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# Define class names and create a class-to-index mapping dictionary
class_names = ['D00', 'D01', 'D10', 'D11', 'D20', 'D40', 'D43', 'D44', 'D50']
class_to_idx = {class_name: i for i, class_name in enumerate(class_names)}
# Load the model
model = models.resnet18(pretrained=False)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(class_names))
model.load_state_dict(torch.load('F:/venkatesh/Resnet/model_8.pth', map_location=d
model.eval()

# Define the transform for the input image
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Evaluate the model on the testing dataset
correct = 0
total = 0
predictions = []
labels = []
with torch.no_grad():

    for images, targets in test_loader:
        images, targets = images.to(device), torch.tensor([class_to_idx[label] for
        print(targets,end=" ")
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        predictions += predicted.cpu().numpy().tolist()
        labels += targets.cpu().numpy().tolist()
        total += targets.size(0)
        correct += (predicted == targets).sum().item()

# Print the accuracy on the testing dataset
print('Accuracy on the testing dataset: %d %%' % (100 * correct / total))

# Print the confusion matrix
```

```
conf_mat = confusion_matrix(labels, predictions)
print(conf_mat)
```

```
tensor([4, 2, 0, 0, 5, 4, 0, 2, 4, 0, 0, 0, 0, 0, 0, 2, 0, 5, 0, 5, 5, 5, 0, 2,
        2, 2, 2, 0, 5, 5, 0, 0]) tensor([0, 2, 0, 4, 0, 4, 2, 0, 0, 0, 0, 2, 2, 4,
4, 5, 4, 4, 5, 0, 4, 0, 2, 0,
        2, 2, 0, 0, 2, 4, 0, 5]) tensor([0, 0, 2, 5, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4,
4, 4, 0, 5, 2, 2, 0, 2, 5, 0,
        0, 2, 0, 0, 0, 0, 0, 0]) tensor([4, 0, 0, 0, 5, 0, 2, 0, 5, 0, 5, 0, 0, 0,
2, 2, 0, 0, 0, 0, 0, 0, 0, 2,
        0, 0, 0, 2, 0, 4, 2, 5]) tensor([5, 5, 5, 5, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
0, 2, 2, 0, 0, 0, 0, 2, 0, 5,
        0, 0, 0, 0, 2, 5, 2, 0]) tensor([0, 0, 2, 0, 0, 0, 2, 2, 5, 0, 2, 0, 4, 4,
4, 0, 0, 0, 0, 0, 5, 0, 0, 2,
        0, 0, 0, 0, 0, 0, 0, 2]) tensor([2, 0, 0, 0, 5, 0, 5, 5, 5, 2, 0, 2, 0, 0,
0, 0, 5, 0, 2, 2, 5, 0, 2, 0,
        0, 2, 2, 0, 0, 2, 4, 0]) tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0,
0, 0, 0, 0, 0, 2, 5, 4, 2, 2,
        0, 0, 0, 0, 0, 0, 2, 0]) tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0,
4, 2, 2, 2, 0, 2, 2, 0, 4, 0,
        0, 0, 0, 0, 0, 4, 2, 0]) tensor([2, 2, 0, 2, 0, 0, 2, 0, 2, 2, 0, 0, 5, 0,
7, 5, 5, 5, 5, 5, 5, 5, 5, 5,
        4, 5, 0, 0, 0, 4, 2, 5]) tensor([4, 5, 7, 4, 4, 0, 0, 6, 7, 7, 4, 7, 0, 5,
5, 5, 5, 5, 5, 0, 4, 5, 0, 7,
        4, 0, 7, 0, 5, 0, 4, 1]) tensor([0, 4, 4, 7, 8, 4, 5, 5, 5, 0, 0, 0, 0, 4,
5, 5, 5, 5, 5, 7, 4, 5, 4, 5,
        5, 0, 5, 5, 5, 1, 7, 4]) tensor([5, 2, 0, 7, 5, 5, 5, 5, 5, 5, 5, 4, 5, 0,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
        5, 4, 7, 4, 4, 4, 4, 0]) tensor([4, 5, 5, 4, 7, 4, 4, 6, 4, 0, 5, 4, 5, 5,
5, 5, 5, 5, 7, 0, 0, 5, 5, 0,
        0, 0, 7, 5, 5, 5, 5, 5]) tensor([5, 0, 5, 7, 1, 0, 4, 5, 4, 5, 5, 5, 0, 5,
5, 5, 5, 5, 4, 4, 4, 7, 4, 5,
        0, 4, 5, 4, 7, 4, 2, 0]) tensor([0, 5, 5, 5, 7, 8, 7, 4, 4, 4, 4, 7, 7, 4,
5, 5, 5, 7, 7, 4, 4, 5, 5, 5,
        5, 4, 5, 5, 5, 0, 0, 5]) tensor([5, 5, 4, 5, 4, 7, 7, 5, 5, 5, 4, 0, 5, 5,
5, 4, 4, 0, 1, 7, 5, 4, 5, 5,
        5, 4, 4, 0, 5, 5, 5, 5]) tensor([5, 5, 4, 0, 0, 5, 0, 0, 5, 5, 5, 5, 0, 7,
4, 5, 4, 4, 4, 5, 5, 5, 5, 5,
        5, 5, 5, 4, 5, 4, 4, 7]) tensor([0, 0, 0, 0, 4, 5, 5, 5, 5, 5, 5, 5, 4, 0,
4, 5, 5, 0, 4, 5, 7, 7, 0, 4,
        7, 7, 4, 0, 5, 5, 5, 5]) tensor([5, 5, 5, 1, 7, 4, 4, 0, 7, 4, 5, 5, 4, 0,
5, 5, 7, 5, 4, 4, 0, 4, 5, 4,
        5, 5, 5, 5, 0, 4, 5, 2]) tensor([4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 6, 1, 1, 5, 0, 4, 8,
        7, 4, 0, 5, 5, 4, 7, 5]) tensor([4, 5, 0, 0, 0, 7, 7, 5, 5, 5, 5, 5, 5, 5,
5, 4, 4, 4, 0, 4, 4, 7, 4, 7,
        5, 5, 5, 4, 5, 5, 4, 5]) tensor([4, 4, 4, 0, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5,
7, 5, 0, 4, 7, 4, 4, 4, 4, 4,
        5, 5, 5, 5, 0, 4, 7, 5]) tensor([5, 5, 5, 5, 0, 5, 4, 8, 4, 7, 0, 4, 7, 0,
7, 7, 7, 7, 5, 5, 5, 7, 7, 1,
        1, 2, 0, 4, 4, 4, 7, 5]) tensor([4, 5, 5, 5, 5, 7, 7, 5, 6, 4, 0, 5, 5, 5,
0, 5, 0, 4, 5, 4, 7, 0, 5, 4,
        0, 5, 7, 5, 4, 4, 7, 7]) tensor([7, 4, 4, 5, 5, 5, 5, 5, 7, 0, 0, 4, 8, 7,
5, 7, 4, 4, 4, 4, 4, 4, 5, 5,
        5, 5, 5, 5, 0, 0, 0, 0]) tensor([5, 0, 0, 4, 5, 5, 5, 5, 5, 5, 5, 5, 7, 5,
5, 7, 4, 0, 6, 4, 4, 7, 7, 4,
        5, 5, 6, 4, 0, 4, 4, 4]) tensor([0, 4, 7, 4, 4, 4, 0, 5, 0, 7, 4, 4, 0, 0,
5, 5, 5, 5, 5, 0, 4, 4, 7, 5,
        4, 5, 5, 5, 5, 4, 4, 5]) tensor([5, 5, 5, 5, 5, 5, 5, 5, 0, 0, 5, 5, 5, 5,
5, 5, 4, 4, 0, 0, 4, 4, 4, 5,
        7, 0, 4, 4, 7, 0, 4, 7]) tensor([4, 4, 5, 5, 4, 4, 7, 0, 4, 5, 5, 5, 0, 5,
0, 5, 5, 5, 5, 4, 7, 4, 0, 4,
        4, 0, 5, 5, 5, 4, 5, 5]) tensor([5, 5, 5, 0, 5, 4, 5, 5, 0, 0, 5, 0, 7, 7,
5, 5, 5, 5, 5, 5, 5, 5, 4, 0, 5,
        5, 5, 4, 4, 4, 5, 5, 4]) tensor([7, 5, 5, 5, 7, 0, 5, 1, 4, 0, 4, 7, 5, 5,
7, 7, 5, 5, 5, 5, 5, 4, 5, 5,
        4, 4, 4, 5, 5, 5, 4, 0]) tensor([0, 0, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
```

```
5, 5, 5, 5, 4, 5, 5, 0, 0, 4,
        4, 0, 5, 4, 5, 5, 5, 7]) tensor([4, 5, 5, 7, 4, 4, 4, 4, 5, 4, 4, 5, 5, 5,
5, 7, 4, 5, 4, 0, 7, 7, 1, 0,
        5, 5, 4, 5, 5, 5, 5, 5]) tensor([5, 5, 5, 4, 7, 7, 4, 5, 5, 5, 5, 5, 4, 0,
0, 4, 4, 4, 5, 5, 4, 4, 0, 2,
        4, 0, 0, 0, 5, 5, 5, 5]) tensor([7, 7, 1, 1, 0, 4, 4, 5, 5, 5, 0, 5, 5, 5,
5, 4, 5, 0, 4, 4, 4, 7, 5, 5,
        5, 5, 5, 0, 0, 4, 4, 5]) tensor([5, 0, 0, 7, 7, 0, 5, 4, 7, 5, 5, 0, 4, 1,
1, 3, 4, 4, 5, 5, 5, 7, 7, 4,
        5, 7, 0, 4, 4, 4, 4, 7]) tensor([5, 5, 5, 5, 5, 5, 7, 0, 0, 5, 0, 0, 0, 4,
1, 0, 0, 5, 7, 7, 5, 0, 5, 5,
        5, 5, 4, 7, 0, 4, 5, 0]) tensor([0, 7, 4, 0, 0, 5, 5, 5, 5, 5, 5, 5, 4, 5,
4, 0, 5, 7, 0, 0, 5, 5, 5, 5,
        5, 0, 0, 5, 4, 4, 5, 5]) tensor([5, 5, 5, 7, 5, 0, 4, 7, 7, 4, 4, 4, 5, 5,
6, 5, 7, 7, 0, 7, 7, 4, 0, 7,
        0, 4, 5, 4, 5, 4, 4, 7]) tensor([7, 7, 7, 5, 5, 0, 4, 4, 4, 7, 0, 5, 4, 7,
0, 7, 7, 1, 4, 7, 0, 0, 0, 0,
        0, 4, 0, 0, 7, 7, 0, 4]) tensor([4, 0, 0, 4, 4, 4, 7, 0, 4, 5, 4, 0, 5, 4,
0, 4, 4, 6, 5, 0, 4, 5, 2, 4,
        4, 5, 5, 4, 4, 5, 0, 0]) tensor([7, 7, 1, 5, 1, 5, 5, 4, 5, 5, 5, 5, 5, 4,
4, 5, 5, 0, 0, 0, 0, 0, 5, 5,
        0, 0, 4, 5, 5, 5, 5, 5]) tensor([5, 5, 5, 4, 5, 5, 7, 7, 5, 4, 5, 0, 5, 5,
2, 2, 0, 0, 0, 0, 0, 5, 0, 5,
        5, 0, 2, 5, 5, 5, 5, 5]) tensor([5, 5, 5, 7, 5, 5, 5, 0, 4, 4, 7, 7, 4, 4,
7, 7, 4, 5, 7, 0, 0, 0, 7, 0,
        4, 5, 5, 5, 5, 5, 7, 0]) tensor([5, 5, 5, 5, 5, 5, 5, 5, 0, 5, 0, 4, 4, 7,
6, 0, 4, 5, 5, 5, 5, 4, 5, 1,
        4, 4, 5, 4, 5, 5, 5, 5]) tensor([5, 5, 0, 5, 4, 7, 7, 4, 5, 5, 2, 2, 4, 4,
0, 0, 4, 0, 2, 5, 5, 5, 4, 7,
        2, 0, 0, 5, 5, 4, 7, 5]) tensor([8, 4, 4, 7, 7, 0, 0, 0, 7, 7, 5, 1, 1, 1,
4, 7, 5, 4, 7, 5, 5, 7, 7, 0,
        0, 5, 4, 5, 5, 5, 5, 5]) tensor([5, 7, 7, 4, 5, 7, 4, 4, 4, 0, 7, 5, 4, 0,
7, 0, 0, 4, 4, 7, 7, 0, 5, 5,
        7, 0, 5, 5, 0, 2, 0, 0]) tensor([0, 4, 0, 4, 4, 0, 7, 7, 5, 5, 4, 4, 0, 7,
5, 5, 2, 5, 5, 5, 5, 5, 1, 1,
        3, 4, 7, 7, 4, 4, 7, 1]) tensor([1, 1, 1, 1, 1, 3, 5, 5, 5, 5, 0, 4, 4, 4,
5, 5, 5, 5, 5, 5, 7, 5, 5, 5,
        4, 1, 7, 0, 0, 5, 5, 7]) tensor([7, 7, 7, 4, 4, 5, 7, 5, 5, 5, 5, 5, 4, 4,
5, 4, 5, 5, 4, 5, 5, 4, 4, 7,
        4, 5, 0, 4, 5, 4, 5, 4]) tensor([4, 4, 0, 0, 5, 1, 7, 5, 4, 7, 4, 7, 4, 4,
5, 5, 4, 4, 5, 7, 4, 5, 5, 5,
        5, 5, 5, 0, 4, 0, 0, 5]) tensor([5, 5, 5, 5, 5, 0, 4, 5, 1, 3, 7, 4, 5, 4,
0, 0, 5, 0, 0, 4, 4, 4, 0, 4,
        0, 5, 5, 5, 0, 1, 7, 7]) tensor([7, 5, 5, 4, 0, 0, 0, 4, 5, 4, 5, 5, 5, 4,
5, 4, 5, 5, 5, 5, 4, 4, 4, 4,
        4, 0, 5, 5, 5, 7, 5, 4]) tensor([5, 5, 5, 5, 4, 4, 0, 4, 4, 0, 0, 4, 7, 0,
4, 5, 5, 5, 5, 5, 0, 5, 0, 4,
        4, 0, 2, 4, 7, 0, 5, 2]) tensor([4, 5, 5, 5, 4, 7, 5, 5, 5, 4, 5, 5, 5, 5,
5, 4, 2, 0, 4, 5, 1, 5, 5, 4,
        4, 0, 5, 5, 5, 4, 4, 4]) tensor([0, 0, 0, 5, 5, 5, 5, 5, 5, 4, 4, 4, 0, 4,
1, 1, 1, 1, 5, 5, 5, 0, 5, 0,
        4, 7, 1, 1, 5, 3, 7, 0]) tensor([7, 4, 5, 7, 4, 4, 7, 5, 4, 4, 5, 4, 5, 5,
5, 0, 5, 1, 3, 5, 7, 4, 0, 0,
        7, 5, 4, 5, 5, 5, 4, 7]) tensor([0, 4, 5, 0, 0, 5, 0, 0, 1, 4, 0, 0, 4, 5,
8, 5, 5, 4, 7, 4, 0, 0, 4, 5,
        7, 0, 7, 4, 4, 7, 0, 7]) tensor([7, 7, 7, 4, 7, 7, 5, 7, 5, 5, 5, 4, 7, 2,
2, 7, 5, 5, 4, 4, 8, 5, 4, 8,
        7, 6, 4, 4, 4, 8, 7, 0]) tensor([2, 0, 7, 7, 2, 0, 4, 7, 8, 4, 2, 7, 8, 8,
8, 2, 7, 4, 7, 7, 7, 2, 4, 2,
        4, 6, 6, 7, 8, 4, 2, 2]) tensor([2, 0, 7, 0, 4, 2, 2, 2, 7, 4, 4, 4, 7, 4,
8, 2, 4, 0, 7, 4, 7, 0, 7, 2,
        0, 2, 2, 8, 8, 0, 8, 8]) tensor([2, 4, 7, 7, 7, 2, 4, 2, 0, 0, 4, 4, 0, 8,
4, 0, 4, 0, 4, 7, 4, 4, 5, 5,
        2, 2, 2, 2, 5, 7, 2, 2]) tensor([7, 4, 4, 7, 7, 0, 8, 5, 5, 8, 4, 8, 5, 5,
```

```
4, 8, 5, 8, 8, 4, 5, 5, 7, 0,
        4, 8, 4, 4, 5, 5, 8, 7]) tensor([0, 7, 2, 2, 2, 4, 4, 7, 8, 6, 7, 7, 4, 8,
7, 4, 7, 2, 2, 4, 7, 4, 0, 2,
        4, 7, 7, 4, 6, 4, 8, 7]) tensor([7, 4, 4, 8, 4, 7, 8, 0, 7, 8, 4, 4, 6, 5,
4, 4, 5, 5, 5, 7, 6, 0, 0, 4,
        4, 4, 7, 4, 2, 6, 0, 8]) tensor([0, 4, 4, 4, 7, 4, 2, 5, 0, 2, 2, 2, 7, 7,
5, 5, 4, 5, 4, 7, 7, 2, 2, 2,
        0, 2, 2, 7, 0, 0, 4, 7]) tensor([8, 7, 2, 2, 0, 0, 4, 8, 7, 4, 8, 4, 4, 7,
7, 2, 2, 2, 2, 0, 0, 4, 4, 4,
        0, 8, 7, 8, 0, 0, 6, 4]) tensor([7, 8, 2, 5, 5, 5, 5, 5, 5, 4, 4, 0, 4, 5,
7, 4, 0, 2, 2, 2, 0, 2, 0, 2,
        8, 8, 4, 2, 2, 2, 0, 0]) tensor([0, 2, 4, 7, 5, 4, 4, 2, 0, 0, 0, 2, 4, 2,
4, 8, 8, 5, 5, 5, 5, 5, 4, 4,
        4, 4, 8, 8, 2, 2, 0, 0]) tensor([7, 0, 7, 0, 0, 7, 0, 0, 5, 5, 5, 4, 4, 4,
2, 2, 4, 6, 8, 0, 0, 4, 4, 4,
        2, 2, 0, 0, 0, 0, 8, 8]) tensor([8, 0, 0, 2, 8, 8, 7, 8, 2, 0, 8, 8, 0, 0,
7, 0, 7, 0, 4, 8, 4, 4, 8, 5,
        4, 2, 2, 4, 4, 2, 0, 4]) tensor([8, 2, 4, 6, 8, 8, 8, 7, 8, 8, 2, 7, 7, 7,
5, 4, 4, 5, 6, 8, 4, 0, 7, 2,
        2, 0, 2, 7, 7, 2, 2, 5]) tensor([4, 5, 7, 7, 4, 2, 0, 4, 8, 4, 2, 4, 0, 8,
4, 5, 5, 5, 5, 8, 6, 2, 4, 5,
        5, 2, 8, 4, 5, 0, 7, 4]) tensor([4, 4, 8, 8, 2, 7, 5, 4, 4, 7, 0, 7, 6, 4,
6, 2, 0, 0, 0, 8, 8, 0, 4, 6,
        6, 7, 8, 4, 7, 4, 2, 2]) tensor([2, 0, 0, 0, 7, 4, 0, 0, 2, 2, 2, 4, 7, 4,
8, 4, 5, 4, 8, 4, 7, 7, 0, 8,
        5, 4, 4, 2, 0, 5, 8, 8]) tensor([8, 7, 5, 5, 7, 7, 8, 8, 7, 2, 8, 8, 4, 4,
5, 5, 4, 4, 8, 4, 5, 5, 5, 4,
        0, 6, 4, 7, 4, 7, 7, 7]) tensor([8, 8, 0, 2, 2, 2, 0, 0, 2, 7, 8, 7, 5, 5,
8, 8, 4, 8, 6, 4, 4, 8, 8, 6,
        8, 4, 8, 7, 4, 0, 4, 2]) tensor([2, 2, 0, 2, 7, 7, 4, 6, 4, 7, 0, 7, 0, 5,
5, 5, 4, 5, 8, 8, 0, 7, 0, 0,
        7, 0, 4, 4, 8, 4, 2, 7]) tensor([7, 4, 5, 5, 2, 2, 4, 2, 2, 8, 2, 2, 4, 7,
4, 4, 2, 6, 5, 4, 7, 7, 4, 4,
        7, 4, 8, 8, 4, 8, 5, 4]) tensor([8, 5, 5, 8, 8, 0, 2, 0, 2, 2, 4, 6, 7, 4,
4, 4, 4, 4, 4, 4, 5, 4, 5, 7,
        0, 5, 5, 5, 7, 8, 8, 4]) tensor([4, 7, 0, 8, 8, 8, 7, 0, 4, 7, 7, 4, 0, 4,
8, 8, 0, 0, 7, 7, 0, 4, 8, 0,
        4, 0, 4, 7, 4, 0, 7, 0]) tensor([7, 0, 8, 4, 4, 4, 7, 6, 2, 7, 7, 4, 7, 7,
0, 7, 0, 7, 8, 0, 2, 2, 2, 2,
        2, 2, 4, 8, 7, 0, 0, 8]) tensor([0, 4, 7, 4, 6, 8, 4, 7, 4, 0, 0, 0, 0, 7,
0, 7, 8, 5, 5, 4, 4, 5, 8, 4,
        4, 8, 5, 4, 5, 6, 4, 7]) tensor([4, 0, 0, 0, 0, 5, 4, 8, 7, 4, 4, 8, 0, 6,
8, 8, 8, 8, 4, 8, 8, 5, 4, 0,
        0, 7, 8, 8, 8, 8, 8, 2]) tensor([5, 0, 0, 4, 4, 4, 4, 7, 8, 7, 4, 7, 7, 7,
8, 4, 5, 4, 4, 8, 0, 5, 5, 5,
        5, 5, 4, 2, 8, 0, 4, 0]) tensor([0, 2, 2, 8, 7, 7, 2, 8, 7, 0, 0, 8, 0, 7,
0, 7, 7, 4, 8, 6, 8, 6, 8, 4,
        0, 2, 2, 2, 2, 2, 5, 5]) tensor([8, 8, 8, 6, 4, 7, 0, 4, 0, 7, 0, 0, 0, 4,
0, 0, 2, 0, 4, 0, 2, 6, 4, 5,
        5, 4, 4, 8, 5, 2, 4, 2]) tensor([0, 2, 2, 7, 7, 0, 0, 5, 4, 4, 7, 0, 4, 4,
2, 0, 4, 2, 2, 0, 7, 5, 0, 7,
        7, 2, 0, 2, 8, 4, 0, 4]) tensor([8, 5, 5, 5, 4, 7, 7, 4, 2, 2, 4, 4, 5, 2,
4, 7, 8, 7, 2, 2, 8, 7, 0, 8,
        7, 4, 7, 6, 0, 8, 0, 8]) tensor([8, 4, 8, 8, 4, 8, 8, 5, 4, 4, 8, 8, 4, 4,
2, 2, 4, 6, 8, 8, 7, 4, 4, 4,
        4, 5, 5, 5, 5, 8, 8, 0]) tensor([4, 7, 4, 7, 8, 5, 5, 0, 7, 4, 2, 0, 0, 4,
7, 0, 0, 0, 8, 4, 4, 4, 4, 4,
        4, 4, 0, 4, 2, 4, 2, 7]) tensor([7, 7, 4, 5, 7, 0, 8, 0, 0, 0, 0, 0, 0, 5,
5, 4, 4, 0, 5, 7, 6, 4, 4, 4,
        0, 8, 6, 0, 0, 4, 0, 0]) tensor([4, 7, 2, 2, 2, 2, 4, 4, 4, 4, 4, 7, 8, 8,
4, 7, 4, 7, 7, 5, 8, 5, 5, 5,
        4, 4, 6, 2, 7, 0, 8, 7]) tensor([7, 0, 4, 7, 2, 4, 5, 4, 2, 5, 4, 8, 4, 0,
0, 8, 4, 4, 0, 4, 2, 2, 8, 8,
        2, 2, 2, 2, 4, 5, 0, 0]) tensor([7, 0, 7, 4, 6, 7, 8, 5, 8, 4, 4, 8, 8, 0,
```

```
4, 7, 7, 4, 4, 4, 8, 8, 5, 8,
        4, 4, 4, 2, 2, 5, 0, 5]) tensor([4, 4, 4, 0, 0, 5, 4, 0, 2, 7, 7, 8, 8, 8,
4, 5, 8, 8, 0, 0, 7, 0, 4, 4,
        0, 2, 7, 4, 7, 0, 4, 4]) tensor([7, 4, 7, 0, 8, 4, 4, 4, 5, 2, 2, 2, 2, 2,
2, 7, 0, 0, 4, 8, 7, 7, 7, 2,
        7, 0, 4, 7, 4, 8, 7, 8]) tensor([8, 4, 4, 7, 2, 7, 8, 8, 7, 0, 4, 2, 0, 4,
2, 2, 7, 4, 5, 5, 4, 4, 5, 0,
        7, 6, 0, 6, 4, 0, 0, 2]) tensor([2, 4, 4, 2, 7, 8, 0, 0, 5, 5, 4, 8, 7, 7,
0, 8, 5, 2, 7, 7, 4, 4, 4, 0,
        0, 0, 0, 7, 2, 7, 7, 8]) tensor([2, 0, 4, 7, 4, 7, 4, 7, 4, 2, 7, 4, 8, 8,
8, 4, 7, 7, 4, 4, 7, 7, 4, 8,
        8, 6, 8, 0, 5, 5, 5, 0]) tensor([0, 5, 5, 7, 8, 2, 5, 8, 5, 8, 8, 0, 7, 5,
4, 8, 4, 8, 0, 4, 0, 2, 2, 7,
        4, 2, 4, 7, 2, 0, 0, 4]) tensor([4, 2, 7, 4, 4, 7, 8, 0, 7, 8, 4, 7, 4, 4,
5, 4, 4, 8, 8, 2, 4, 4, 4, 2,
        2, 2, 2, 0, 4, 6, 2, 4]) tensor([0, 2, 2, 0, 4, 4, 2, 2, 2, 0, 0, 0, 0, 5,
4, 8, 7, 0, 4, 8, 8, 7, 4, 5,
        5, 5, 5, 5, 5, 7, 0, 7]) tensor([0, 7, 2, 4, 2, 2, 2, 2, 2, 7, 4, 0, 5, 2,
4, 4, 2, 2, 7, 4, 6, 2, 0, 8,
        5, 7, 0, 0, 0, 0, 0, 0]) tensor([0, 0, 0, 7, 7, 2, 4, 7, 6, 8, 4, 4, 2, 2,
2, 2, 0, 0, 0, 8, 7, 0, 7, 7,
        0, 2, 0, 4, 0, 0, 8, 5]) tensor([7, 0, 7, 7, 8, 2, 2, 2, 7, 7, 0, 0, 0, 5,
5, 4, 4, 8, 4, 4, 8, 4, 4, 0,
        2, 6, 0, 0, 4, 5, 8, 2]) tensor([7, 7, 0, 7, 4, 2, 7, 4, 0, 5, 2, 2, 0, 0,
0, 0, 4, 4, 8, 2, 0, 8, 7, 4,
        2, 2, 8, 5, 5, 7, 5, 5]) tensor([5, 5, 4, 8, 4, 5, 4, 4, 6, 8, 8, 4, 2, 2,
0, 6, 8, 5, 4, 4, 2, 2, 2, 2,
        2, 2, 7, 4, 2, 4, 7, 4]) tensor([7, 4, 2, 2, 4, 7, 7, 7, 8, 4, 5, 8, 4, 8,
5, 7, 7, 7, 4, 2, 2, 2, 8, 4,
        6, 4, 7, 0, 4, 7, 7, 2]) tensor([7, 8, 2, 7, 4, 4, 4, 7, 7, 7, 7, 8, 2, 7,
5, 4, 7, 4, 8, 4, 4, 4, 6, 8,
        8, 4, 8, 4, 5, 8, 8, 4]) tensor([5, 4, 2, 0, 4, 4, 8, 5, 4, 8, 4, 4, 4, 8,
4, 8, 2, 7, 8, 7, 7, 8, 7, 7,
        7, 2, 2, 0, 0, 7, 8, 0]) tensor([5, 6, 8, 8, 4, 0, 5, 4, 4, 7, 4, 8, 8, 7,
4, 2, 2, 2, 4, 8, 8, 8, 0, 4,
        4, 4, 8, 4, 7, 4, 4, 5]) tensor([4, 4, 4, 2, 2, 4, 4, 8, 6, 4, 7, 7, 4, 4,
8, 7, 4, 4, 2, 8, 7, 4, 4, 7,
        2, 2, 2, 2, 7, 8, 8, 0]) tensor([0, 7, 8, 7, 8, 0, 4, 2, 2, 2, 7, 8, 4, 2,
4, 8, 4, 2, 2, 2, 2, 4, 2, 4,
        7, 7, 0, 0, 0, 5, 8, 4]) tensor([8, 4, 0, 8, 2, 0, 7, 4, 2, 4, 8, 0, 4, 7,
4, 7, 7, 8, 2, 5, 4, 0, 0, 4,
        4, 4, 4, 5, 4, 4, 8, 4]) tensor([5, 4, 4, 4, 4, 4, 8, 7, 0, 4, 4, 0, 7, 0,
7, 0, 0, 2, 0, 0, 0, 4, 8, 4,
        5, 4, 4, 8, 8, 5, 7, 4]) tensor([2, 7, 4, 7, 7, 6, 0, 4, 7, 2, 2, 2, 7, 2,
4, 7, 0, 2, 4, 7, 7, 7, 2, 0,
        7, 0, 2, 5, 4, 5, 5, 4]) tensor([8, 7, 7, 7, 0, 8, 4, 7, 7, 0, 6, 7, 7, 7,
7, 4, 8, 0, 7, 8, 6, 4, 7, 2,
        4, 4, 4, 2, 0, 4, 5, 0]) tensor([4, 6, 4, 2, 2, 7, 0, 8, 2, 2, 2, 0, 0, 0,
4, 4, 4, 8, 4, 7, 7, 4, 8, 7,
        0, 4, 5, 5, 4, 8, 8, 8]) tensor([8, 4, 4, 7, 0, 0, 7, 2, 7, 7, 7, 6, 8, 2,
2, 8, 8, 4, 4, 2, 4, 5, 7, 8,
        5, 7, 4, 4, 5, 5, 5, 4]) tensor([4, 2, 2, 2, 0, 7, 7, 4, 7, 0, 4, 4, 7, 2,
0, 4, 6, 4, 4, 8, 8, 4, 2, 0,
        0, 4, 4, 7, 0, 0, 8, 8]) tensor([2, 2, 4, 7, 4, 4, 0, 4, 8, 8, 5, 5, 4, 4,
4, 7, 7, 0, 0, 7, 7, 4, 7, 4,
        8, 4, 2, 2, 2, 5, 8, 7]) tensor([0, 0, 7, 4, 4, 8, 0, 8, 4, 7, 0, 7, 0, 0,
7, 0, 8, 0, 0, 0, 0, 0, 7, 7,
        6, 0, 7, 8, 0, 4, 5, 5]) tensor([4, 6, 0, 0, 7, 0, 0, 7, 4, 2, 2, 2, 0, 0,
0, 0, 0, 2, 2, 2, 7, 8, 0, 7,
        4, 0, 2, 4, 0, 0, 0, 0]) tensor([0, 2, 4, 2, 2, 2, 4, 8, 8, 4, 4, 5, 5, 4,
7, 4, 4, 7, 5, 2, 7, 7, 8, 2,
        0, 0, 6, 8, 4, 0, 8, 8]) tensor([6, 4, 4, 4, 0, 2, 4, 5, 5, 5, 5, 5, 4, 7,
0, 4, 4, 5, 4, 7, 7, 7, 0, 2,
        7, 8, 7, 4, 7, 4, 0, 4]) tensor([4, 2, 6, 7, 4, 0, 2, 0, 0, 0, 0, 2, 0, 4,
```

```
6, 7, 7, 4, 4, 4, 0, 0, 4, 7,
        2, 2, 4, 6, 7, 2, 0, 5]) tensor([4, 4, 8, 0, 0, 2, 7, 7, 5, 2, 7, 8, 2, 2,
2, 4, 7, 7, 4, 4, 7, 6, 0, 5,
        4, 5, 5, 5, 7, 6, 4, 4]) tensor([4, 4, 4, 5, 5, 4, 4, 6, 8, 8, 6, 7, 7, 4,
4, 2, 8, 4, 4, 5, 5, 4, 7, 4,
        7, 0, 7, 7, 0, 5, 5, 5]) tensor([5, 5, 4, 2, 4, 8, 0, 0, 7, 0, 8, 4, 2, 2,
0, 0, 7, 7, 4, 2, 4, 7, 4, 2,
        2, 0, 8, 8, 2, 2, 2, 6]) tensor([8, 2, 5, 4, 4, 4, 5, 7, 4, 8, 4, 4, 7, 8,
2, 2, 2, 7, 0, 4, 2, 2, 2, 2,
        2, 4, 2, 2, 2, 2, 2, 2]) tensor([4, 7, 0, 7, 0, 0, 7, 4, 4, 4, 4, 7, 4, 7,
7, 7, 0, 4, 7, 7, 0, 4, 8, 4,
        4, 4, 8, 2, 2, 2, 4, 8]) tensor([7, 2, 4, 4, 8, 2, 7, 4, 8, 7, 6, 4, 6, 8,
6, 8, 8, 7, 8, 8, 7, 4, 8, 7,
        8, 4, 2, 2, 2, 2, 4, 7]) tensor([0, 4, 8, 5, 0, 2, 0, 0, 5, 4, 7, 8, 5, 0,
6, 6, 8, 8, 4, 7, 7, 8, 2, 4,
        7, 7, 2, 0, 7, 7, 0, 4]) tensor([7, 4, 2, 7, 2, 2, 4, 5, 2, 4, 6, 8, 4, 4,
4, 7, 0, 4, 8, 8, 8, 8, 4, 0,
        2, 7, 0, 7, 7, 7, 7, 0]) tensor([5, 4, 7, 7, 7, 0, 7, 0, 4, 0, 4, 8, 4, 4,
4, 4, 4, 8, 5, 4, 5, 0, 6, 4,
        0, 7, 4, 2, 4, 2, 2, 4]) tensor([4, 8, 8, 7, 7, 7, 0, 7, 4, 4, 8, 8, 4, 4,
5, 7, 0, 7, 4, 2, 2, 8, 2, 2,
        4, 4, 8, 4, 4, 8, 4, 2]) tensor([8, 0, 8, 0, 4, 4, 4, 8, 8, 8, 2, 4, 0, 2,
2, 8, 7, 2, 0, 0, 2, 4, 4, 5,
        4, 4, 2, 4, 4, 8, 7, 0]) tensor([7, 4, 7, 4, 2, 0, 4, 0, 0, 0, 4, 7, 4, 2,
5, 8, 4, 2, 4, 0, 4, 7, 7, 4,
        8, 0, 8, 0, 8, 7, 4, 8]) tensor([4, 4, 4, 4, 0, 0, 0, 8, 8, 6, 8, 8, 4, 8,
8, 4, 2, 2, 0, 4, 4, 4, 7, 4,
        0, 4, 7, 0, 4, 8, 5, 4]) tensor([4, 4, 8, 7, 4, 5, 7, 8, 8, 4, 4, 2, 2, 0,
7, 8, 7, 7, 4, 8, 4, 7, 7, 7,
        4, 7, 4, 2, 7, 8, 8, 7]) tensor([4, 5, 5, 4, 4, 7, 4, 2, 2, 2, 8, 5, 5, 2,
2, 2, 4, 7, 7, 7, 7, 0, 4, 4,
        5, 2, 2, 5, 8, 7, 7, 4]) tensor([8, 7, 2, 4, 0, 4, 5, 7, 0, 6, 6, 8, 8, 8,
0, 8, 8, 5, 5, 4, 8, 4, 0, 0,
        4, 2, 2, 2, 2, 2, 4, 5]) tensor([5, 5, 2, 4, 8, 8, 4, 2, 2, 2, 2, 4, 7, 4,
4, 0, 4, 8, 7, 7, 8, 4, 8, 8,
        7, 7, 0, 7, 7, 0, 0, 7]) tensor([4, 0, 0, 0, 2, 0, 4, 7, 4, 7, 0, 8, 4, 2,
0, 7, 7, 4, 8, 5, 5, 4, 0, 5,
        5, 8, 5, 8, 2, 0, 0, 4]) tensor([4, 2, 7, 7, 2, 0, 0, 7, 8, 2, 2, 0, 8, 2,
4, 8, 2, 8, 4, 4, 8, 5, 5, 4,
        7, 4, 0, 0, 0, 0, 0, 0]) tensor([2, 2, 2, 5, 5, 0, 2, 2, 2, 4, 4, 8, 4, 8,
8, 4, 4, 0, 7, 4, 0, 0, 4, 0,
        2, 2, 2, 4, 8, 2, 2, 2]) tensor([2, 7, 0, 7, 4, 0, 2, 4, 4, 4, 8, 7, 6, 0,
4, 2, 2, 7, 0, 5, 4, 5, 4, 7,
        7, 7, 4, 0, 0, 0, 4, 4]) tensor([4, 2, 2, 2, 2, 4, 8, 0, 4, 2, 0, 2, 0, 7,
2, 2, 7, 4, 8, 4, 4, 8, 2, 2,
        2, 2, 5, 5, 5, 5, 8, 2]) tensor([7, 2, 0, 7, 2, 5, 8, 8, 7, 7, 4, 7, 5, 4,
4, 8, 0, 0, 0, 0, 0, 4, 7, 4,
        7, 4, 7, 7, 0, 0, 4, 7]) tensor([7, 5, 5, 8, 4, 4, 7, 7, 0, 0, 4, 7, 7, 0,
2, 0, 2, 2, 2, 0, 0, 7, 7, 7,
        4, 2, 7, 7, 4, 7, 4, 2]) tensor([4, 4, 4, 2, 7, 6, 8, 0, 0, 7, 4, 4, 7, 0,
7, 7, 6, 8, 0, 4, 2, 4, 4, 0,
        8, 4, 7, 7, 0, 0, 0, 8]) tensor([0, 4, 0, 6, 7, 8, 8, 5, 5, 8, 0, 4, 2, 7,
7, 7, 5, 5, 5, 4, 4, 8, 5, 5,
        4, 8, 4, 5, 8, 8, 5, 5]) tensor([2, 7, 4, 0, 2, 2, 4, 4, 5, 8, 8, 6, 4, 8,
8, 4, 5, 7, 7, 4, 4, 7, 7, 0,
        7, 7, 7, 4, 5, 5, 4, 0]) tensor([0, 0, 0, 8, 0, 4, 4, 4, 4, 7, 0, 8, 8, 0,
8, 4, 8, 8, 8, 7, 4, 0, 4, 4,
        0, 0, 7, 0, 0, 4, 8, 8]) tensor([4, 7, 4, 4, 8, 0, 2, 2, 0, 7, 8, 5, 5, 0,
8, 2, 4, 0, 4, 7, 8, 8, 4, 0,
        7, 0, 4, 2, 2, 2, 4, 4]) tensor([7, 6, 5, 4, 4, 4, 4, 0, 7, 2, 0, 2, 2, 8,
8, 8, 8, 2, 4, 0, 0, 0, 4, 0,
        0, 4, 4, 4, 7, 4, 7, 7]) tensor([4, 7, 7, 4, 7, 2, 8, 4, 2, 0, 0, 4, 2, 5,
4, 5, 2, 4, 6, 8, 8, 7, 0, 4,
        8, 5, 4, 4, 7, 7, 7, 2]) tensor([2, 2, 7, 0, 0, 4, 6, 2, 2, 2, 4, 5, 4, 5,
```

```
5, 8, 8, 5, 7, 4, 8, 8, 0, 4,
        7, 4, 4, 2, 7, 2, 0, 5]) tensor([5, 7, 7, 8, 2, 7, 7, 0, 0, 7, 7, 0, 0, 7,
0, 0, 0, 8, 8, 0, 0, 2, 4, 8,
        0, 7, 7, 2, 2, 7, 7, 7]) tensor([6, 7, 8, 8, 7, 8, 8, 5, 5, 5, 4, 5, 7, 7,
4, 4, 5, 4, 4, 4, 4, 5, 5, 5,
        4, 7, 7, 7, 4, 8, 5, 0]) tensor([4, 7, 4, 2, 2, 4, 4, 8, 4, 7, 5, 2, 2, 0,
0, 5, 4, 8, 8, 8, 4, 2, 2, 7,
        4, 7, 7, 8, 7, 4, 6, 0]) tensor([7, 7, 0, 0, 0, 7, 0, 7, 4, 7, 7, 0, 6, 7,
8, 0, 0, 2, 2, 7, 2, 2, 4, 7,
        0, 4, 0, 4, 8, 2, 8, 8]) tensor([0, 7, 2, 2, 4, 8, 7, 7, 0, 7, 7, 4, 4, 8,
4, 7, 0, 7, 7, 8, 4, 5, 5, 4,
        2, 2, 7, 0, 8, 8, 2, 5]) tensor([8, 4, 4, 6, 4, 8, 7, 4, 8, 4, 5, 7, 7, 2,
7, 8, 4, 4, 4, 5, 4, 2, 0, 2,
        2, 0, 5, 7, 4, 2, 4, 7]) tensor([5, 7, 4, 7, 4, 5, 4, 8, 4, 4, 8, 5, 5, 2,
2, 7, 0, 0, 0, 2, 4, 7, 7, 4,
        8, 8, 8, 4, 5, 4, 4, 2]) tensor([0, 4, 8, 4, 7, 2, 2, 2, 2, 2, 8, 0, 5, 5,
5, 7, 0, 0, 5, 5, 4, 4, 4, 5,
        4, 8, 0, 5, 4, 8, 8, 4]) tensor([4, 7, 4, 0, 0, 0, 0, 7, 8, 8, 0, 7, 7, 2,
0, 0, 0, 7, 7, 4, 4, 2, 7, 4,
        5, 5, 8, 5, 5, 4, 4, 4]) tensor([4, 0, 7, 4, 7, 4, 4, 7, 8, 2, 2, 2, 0, 6,
5, 5, 5, 5, 5, 7, 4, 6, 2, 8,
        0, 7, 0, 7, 0, 4, 7, 4]) tensor([7, 7, 0, 5, 4, 8, 4, 2, 2, 2, 2, 0, 4, 0,
0, 2, 7, 7, 4, 7, 7, 6, 4, 4,
        7, 4, 4, 0, 7, 0, 0, 0]) tensor([7, 7, 7, 2, 2, 2, 0, 4, 4, 7, 4, 5, 5, 4,
4, 2, 7, 7, 4, 8, 4, 4, 0, 2,
        0, 7, 8, 4, 4, 0, 7, 7]) tensor([0, 7, 8, 5, 8, 4, 7, 7, 7, 4, 7, 7, 8, 0,
0, 0, 6, 7, 4, 2, 5, 4, 7, 7,
        4, 7, 0, 7, 0, 7, 2, 2]) tensor([5, 5, 5, 8, 8, 4, 7, 4, 7, 0, 8, 7, 4, 4,
7, 4, 2, 0, 4, 0, 0, 0, 7, 4,
        4, 5, 5, 0, 5, 4, 2, 4]) tensor([5, 4, 8, 4, 0, 7, 0, 4, 7, 4, 4, 6, 8, 4,
4, 0, 8, 5, 4, 5, 4, 5, 4, 4,
        7, 4, 4, 4, 7, 0, 4, 0]) tensor([0, 0, 5, 5, 4, 8, 4, 4, 8, 5, 5, 5, 5, 4,
7, 7, 7, 0, 0, 2, 2, 5, 4, 7,
        4, 8, 4, 8, 0, 8, 7, 4]) tensor([7, 7, 7, 8, 8, 7, 0, 2, 7, 4, 8, 7, 2, 7,
5, 4, 2, 2, 0, 4, 2, 2, 4, 8,
        7, 7, 0, 2, 2, 0, 7, 7]) tensor([8, 8, 2, 8, 8, 4, 7, 4, 5, 5, 4, 7, 4, 2,
2, 2, 5, 4, 8, 4, 7, 7, 7, 7,
        4, 0, 6, 5, 8, 8, 4, 2]) tensor([4, 4, 7, 8, 4, 4, 7, 0, 4, 6, 7, 4, 4, 0,
4, 2, 0, 6, 4, 5, 4, 7, 5, 6,
        7, 0, 7, 2, 4, 4, 0, 4]) tensor([8, 4, 0, 4, 0, 8, 7, 7, 7, 8, 4, 8, 7, 8,
4, 4, 5, 4, 7, 7, 8, 4, 7, 6,
        8, 2, 4, 7, 4, 5, 5, 5]) tensor([5, 5, 5, 5, 4, 8, 8, 4, 2, 8, 4, 4, 5, 4,
8, 4, 4, 4, 5, 4, 4, 8, 5, 4,
        4, 4, 5, 5, 5, 0, 5, 0]) tensor([4, 4, 0, 5, 4, 2, 7, 2, 0, 0, 2, 6, 0, 0,
8, 8, 8, 7, 4, 8, 0, 2, 2, 0,
        0, 4, 7, 0, 7, 4, 8, 8]) tensor([4, 4, 8, 4, 4, 7, 7, 4, 0, 0, 0, 4, 7, 4,
2, 8, 4, 8, 8, 4, 5, 4, 2, 2,
        2, 2, 0, 8, 2, 2, 2, 2]) tensor([7, 7, 0, 0, 2, 2, 0, 0, 4, 4, 4, 2, 5, 4,
0, 6, 0, 4, 8, 0, 0, 7, 8, 4,
        2, 7, 4, 7, 2, 4, 5, 2]) tensor([4, 2, 2, 2, 2, 2, 7, 0, 4, 2, 4, 2, 5, 4,
8, 5, 4, 8, 8, 8, 2, 2, 0, 8,
        7, 7, 7, 7, 8, 7, 4, 6]) tensor([0, 2, 4, 7, 0, 8, 8, 4, 5, 5, 5, 5, 5, 4,
4, 7, 0, 5, 4, 2, 7, 7, 2, 0,
        4, 5, 8, 7, 0, 0, 2, 8]) tensor([8, 4, 7, 4, 0, 4, 2, 2, 5, 8, 7, 4, 0, 8,
0, 0, 2, 7, 0, 4, 8, 8, 7, 7,
        7, 4, 7, 7, 8, 4, 4, 8]) tensor([0, 0, 4, 4, 5, 4, 7, 4, 7, 2, 2, 8, 2, 0,
0, 0, 7, 7, 0, 0, 7, 6, 7, 7,
        0, 0, 0, 0, 2, 2, 7, 8]) tensor([7, 4, 7, 4, 5, 4, 8, 4, 7, 2, 0, 4, 4, 7,
0, 4, 2, 2, 0, 4, 0, 0, 7, 7,
        8, 4, 4, 8, 0, 8, 8, 4]) tensor([2, 2, 2, 4, 7, 0, 0, 2, 0, 7, 7, 7, 4, 4,
0, 7, 8, 8, 4, 7, 0, 4, 7, 0,
        4, 8, 4, 4, 0, 0, 0, 5]) tensor([4, 7, 4, 2, 7, 0, 4, 7, 7, 8, 2, 8, 5, 7,
8, 4, 2, 0, 2, 4, 4, 5, 7, 4,
        2, 2, 2, 2, 2, 2, 0, 0]) tensor([2, 7, 8, 5, 4, 4, 4, 5, 4, 5, 7, 0, 4, 2,
```

```
5, 7, 4, 4, 7, 2, 7, 7, 0, 7,
        0, 0, 0, 2, 0, 2, 7, 4]) tensor([7, 4, 0, 8, 0, 8, 4, 7, 4, 4, 7, 7, 8, 8,
4, 4, 7, 4, 5, 0, 0, 4, 5, 4,
        0, 0, 0, 0, 7, 8, 4, 8]) tensor([0, 2, 0, 4, 7, 4, 5, 0, 0, 2, 6, 5, 7, 4,
7, 0, 0, 7, 2, 2, 7, 0, 0, 8,
        8, 5, 4, 4, 2, 2, 4, 2]) tensor([0, 6, 5, 5, 5, 5, 7, 0, 0, 0, 2, 0, 7, 0,
0, 4, 5, 5, 4, 6, 8, 7, 2, 2,
        4, 8, 0, 4, 0, 6, 2, 8]) tensor([4, 4, 4, 8, 4, 8, 8, 7, 7, 0, 0, 4, 0, 0,
0, 7, 0, 4, 6, 2, 0, 4, 8, 8,
        8, 8, 8, 7, 6, 4, 0, 2]) tensor([2, 7, 2, 0, 4, 5, 2, 7, 2, 7, 7, 7, 0, 7,
4, 0, 0, 4, 0, 2, 2, 4, 2, 0,
        0, 8, 5, 5, 7, 0, 5, 4]) tensor([2, 2, 2, 2, 2, 4, 2, 2, 0, 2, 7, 0, 2, 4,
0, 4, 8, 8, 0, 4, 8, 4, 4, 8,
        8, 0, 7, 2, 2, 6]) Accuracy on the testing dataset: 89 %
[[1066   11   14    0   43   37    0   15    0]
 [   9   29    0    0    0    0    0    6    0]
 [  28    2  689    1    4   10    1    1    3]
 [   0    0    1    5    0    0    0    0    0]
 [ 154    0   19    0 1285   44    2   26    1]
 [  72    0   13    0   29  993    0    4   19]
 [   1    0    1    0    1    0  106   11    0]
 [  48    5    2    0   15    3    4  916    1]
 [   4    0    2    0    0   17    0    0  593]]
```

In [ ]:

```python
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate precision, recall, and F1-score
precision = precision_score(labels, predictions, average='weighted')
recall = recall_score(labels, predictions, average='weighted')
f1 = f1_score(labels, predictions, average='weighted')

# Print the results
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
```

```
Precision: 0.8975561309714356
Recall: 0.8925541941564562
F1-score: 0.8935008140164337
```

In [ ]:

```python
from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix
conf_mat = confusion_matrix(labels, predictions)

# Initialize lists to store TP, FP, TN, FN for each class
TP_list = []
FP_list = []
TN_list = []
FN_list = []

# Calculate TP, FP, TN, FN for each class
for i in range(len(class_names)):
    TP = conf_mat[i][i]
    FP = sum(conf_mat[:, i]) - TP
    FN = sum(conf_mat[i, :]) - TP
    TN = conf_mat.sum() - TP - FP - FN

    TP_list.append(TP)
    FP_list.append(FP)
    TN_list.append(TN)
    FN_list.append(FN)

# Print the results
```

```
for i in range(len(class_names)):
    print('Class:', class_names[i])
    print('True positives:', TP_list[i])
    print('False positives:', FP_list[i])
    print('True negatives:', TN_list[i])
    print('False negatives:', FN_list[i])
    print()
```

```
Class: D00
True positives: 1066
False positives: 316
True negatives: 4864
False negatives: 120

Class: D01
True positives: 29
False positives: 18
True negatives: 6304
False negatives: 15

Class: D10
True positives: 689
False positives: 52
True negatives: 5575
False negatives: 50

Class: D11
True positives: 5
False positives: 1
True negatives: 6359
False negatives: 1

Class: D20
True positives: 1285
False positives: 92
True negatives: 4743
False negatives: 246

Class: D40
True positives: 993
False positives: 111
True negatives: 5125
False negatives: 137

Class: D43
True positives: 106
False positives: 7
True negatives: 6239
False negatives: 14

Class: D44
True positives: 916
False positives: 63
True negatives: 5309
False negatives: 78

Class: D50
True positives: 593
False positives: 24
True negatives: 5726
False negatives: 23
```