Rock-Scissors-Paper detection

▼ dataset

```
data/
  train/
    paper/
      *.png
    rock/
      *.png
    scissors/
      *.png

  test/
    paper/
      *.png
    rock/
      *.png
    scissors/
      *.png
```

▼ train & test code

```
import os
from PIL import Image
import torch
from torch.utils.data import Dataset, DataLoader
from torch import nn
from torchvision import transforms


class CustomImageDataset(Dataset):
    def read_data_set(self):

        all_img_files = []
        all_labels = []

        class_names = os.walk(self.data_set_path).__next__()[1]

        for index, class_name in enumerate(class_names):
            label = index
            img_dir = os.path.join(self.data_set_path, class_name)
            img_files = os.walk(img_dir).__next__()[2]

            for img_file in img_files:
                img_file = os.path.join(img_dir, img_file)
                img = Image.open(img_file)
                if img is not None:
                    all_img_files.append(img_file)
                    all_labels.append(label)

        return all_img_files, all_labels, len(all_img_files), len(class_names)

    def __init__(self, data_set_path, transforms=None):
        self.data_set_path = data_set_path
        self.image_files_path, self.labels, self.length, self.num_classes = self.read_data_set()
        self.transforms = transforms

    def __getitem__(self, index):
        image = Image.open(self.image_files_path[index])
        image = image.convert("RGB")

        if self.transforms is not None:
            image = self.transforms(image)

        return {'image': image, 'label': self.labels[index]}

    def __len__(self):
        return self.length


class CustomConvNet(nn.Module):
    def __init__(self, num_classes):
        super(CustomConvNet, self).__init__()

        self.layer1 = self.conv_module(3, 16)
        self.layer2 = self.conv_module(16, 32)
        self.layer3 = self.conv_module(32, 64)
        self.layer4 = self.conv_module(64, 128)
        self.layer5 = self.conv_module(128, 256)
        self.gap = self.global_avg_pool(256, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.layer5(out)
        out = self.gap(out)
        out = out.view(-1, num_classes)

        return out

    def conv_module(self, in_num, out_num):
        return nn.Sequential(
            nn.Conv2d(in_num, out_num, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_num),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))

    def global_avg_pool(self, in_num, out_num):
        return nn.Sequential(
            nn.Conv2d(in_num, out_num, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_num),
            nn.LeakyReLU(),
            nn.AdaptiveAvgPool2d((1, 1)))


hyper_param_epoch = 200
hyper_param_batch = 32
hyper_param_learning_rate = 0.001

transforms_train = transforms.Compose([transforms.Resize((128, 128)),
                                       transforms.RandomRotation(10.),
                                       transforms.ToTensor()])

transforms_test = transforms.Compose([transforms.Resize((128, 128)),
                                      transforms.ToTensor()])

train_data_set = CustomImageDataset(data_set_path="./data/train", transforms=transforms_train)
train_loader = DataLoader(train_data_set, batch_size=hyper_param_batch, shuffle=True)

test_data_set = CustomImageDataset(data_set_path="./data/test", transforms=transforms_test)
test_loader = DataLoader(test_data_set, batch_size=hyper_param_batch, shuffle=True)

if not (train_data_set.num_classes == test_data_set.num_classes):
    print("error: Numbers of class in training set and test set are not equal")
    exit()

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

num_classes = train_data_set.num_classes
custom_model = CustomConvNet(num_classes=num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(custom_model.parameters(), lr=hyper_param_learning_rate)

for e in range(hyper_param_epoch):
    for i_batch, item in enumerate(train_loader):
        images = item['image'].to(device)
        labels = item['label'].to(device)

        # Forward pass
        outputs = custom_model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i_batch + 1) % hyper_param_batch == 0:
            print('Epoch [{}/{}], Loss: {:.4f}'
                .format(e + 1, hyper_param_epoch, loss.item()))

# test the model
custom_model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for item in test_loader:
        images = item['image'].to(device)
        labels = item['label'].to(device)
        outputs = custom_model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += len(labels)
        correct += (predicted == labels).sum().item()

    print('test Accuracy of the model on the {} test images: {} %'.format(total, 100 * correct / total))

torch.save(custom_model, "classify_model_200.pth")
torch.save(custom_model.state_dict(), "classify_model_state_dict.pth")
```

▼ inference code

```
import torch
from torch import nn
from torchvision import transforms
from PIL import Image

class CustomConvNet(nn.Module):
    def __init__(self):
        super(CustomConvNet, self).__init__()

        self.layer1 = self.conv_module(3, 16)
        self.layer2 = self.conv_module(16, 32)
        self.layer3 = self.conv_module(32, 64)
        self.layer4 = self.conv_module(64, 128)
        self.layer5 = self.conv_module(128, 256)
        self.gap = self.global_avg_pool(256, 3)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.layer5(out)
        out = self.gap(out)
        out = out.view(-1, 3)
        return out

    def conv_module(self, in_num, out_num):
        return nn.Sequential(
            nn.Conv2d(in_num, out_num, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_num),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))

    def global_avg_pool(self, in_num, out_num):
        return nn.Sequential(
            nn.Conv2d(in_num, out_num, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_num),
            nn.LeakyReLU(),
            nn.AdaptiveAvgPool2d((1, 1)))

class_name = ['rock', 'scissors', 'paper']

transform = transforms.Compose([
            transforms.Resize((128, 128)),
            transforms.ToTensor()
            ])

img = Image.open("./testrock01-00.PNG")
img = img.convert("RGB")
img = transform(img)
img =img.view(1, 3, 128, 128).cuda()

model = torch.load("./classify_model_200.pth")
model.eval().cuda()

with torch.no_grad():

    outputs = model(img)
    idx = torch.argmax(outputs)
    print(class_name[idx])
```