

机器视觉技术课程大作业 Part2

一、实验名称：针尖注射识别

二、实验目的

利用 pytorch 编写学习算法代码，实现斑马鱼针尖注射识别。

三、实验思路历程

在将原图像集直接放入 alexnet 进行训练后，得到了大概 70% 的正确率，这说明成功与失败的图像序列有一部分是难以区分的。在选择各图像序列的最后一张图片进行训练和验证时，发现正确率有所提高，这说明，随着时间的推移，成功与失败的图像的差别开始变大。因此，尝试对图像序列进行差分处理，选取第一张和最后一张，突出注射带来的变化，以便于区分。

四、实验原理

通过观察数据集图像组和简单图像处理后的结果，得到了以下检测思路：图像序列给出了注射的过程，使用图像 8 和图像 1 进行差分，成功的会得到注射形成的白影，而失败则几乎全黑。利用标准的 alexnet 神经网络，可以对这种区别进行有效区分。

五、实验步骤

- 1、alexnet 神经网络搭建：利用已有的标准 alexnet 神经网络进行修改，使其适应当前的分类任务。具体包括特征提取、分类器、前向传播、后向传播、权重初始化、交叉熵损失函数等部分。对应 model.py 部分。
- 2、图像序列处理：使用图像序列的最后一张图和第一张图进行差分，将处理后的差分图像保存到数据集路径中去。对应 process.py 部分。
- 3、数据集导入：利用 dataset、loader 等函数，载入处理过的图像作为训练集和测试集，划分比例是 3: 1，并对其进行预处理（缩放、转为张量、归一化），再生成类别字典（“1”为“Success”，“0”为“Fail”）。对应 train.py 前半部分。
- 4、训练与验证：训练模型并用验证集验证，通过调试，获得适宜的参数，并将训练好的模型保存为 AlexNet.pth。对应 train.py 后半部分。
- 5、测试与分析：利用整个数据集进行模拟测试，并对结果中的误差进行分析。在测试时，“1”表示注射成功，“0”表示注射失败。

六、代码

model.py

```
import torch
import torch.nn as nn
class AlexNet(nn.Module):
    def __init__(self, num_classes=2, init_weights=False):
        super(AlexNet, self).__init__()
        # 特征提取
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
```

```

        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=3, stride=2),
        nn.Conv2d(64, 128, kernel_size=5, padding=2),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=3, stride=2),
        nn.Conv2d(128, 192, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(192, 192, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(192, 128, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=3, stride=2),
    )
    # 分类器
    self.classifier = nn.Sequential(
        nn.Dropout(p=0.5),
        nn.Linear(128 * 6 * 6, 2048),
        nn.ReLU(inplace=True),
        nn.Dropout(p=0.5),
        nn.Linear(2048, 2048),
        nn.ReLU(inplace=True),
        nn.Linear(2048, num_classes),
    )
    if init_weights:
        self._initialize_weights()

    # 前向传播
    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, start_dim=1)
        x = self.classifier(x)
        return x

    # 权重初始化
    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode="fan_out",
nonlinearity="relu")
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)

```

process.py

```

import os
from PIL import Image, ImageChops
# 统计文件夹内文件夹个数
def count_folder(path):
    count = 0
    for item in os.listdir(path):
        if os.path.isdir(os.path.join(path, item)):
            count += 1
    return count
# 从输入地址获取图片进行差分，输出到输出地址
def diff(in_path, out_path):
    counts = count_folder(in_path)
    for i in range(1, counts + 1):
        img1_path = os.path.join(in_path, "MyVideo_%s" % i, "1.jpg")
        img2_path = os.path.join(in_path, "MyVideo_%s" % i, "8.jpg")
        img1 = Image.open(img1_path)
        img2 = Image.open(img2_path)
        difference = ImageChops.subtract(img1, img2)
        out_full_path = os.path.join(out_path, "%s.jpg" % i)
        difference.save(out_full_path)
# 主函数
'''
diff("data/train/Fail", "data/process/train/Fail")
diff("data/train/Success", "data/process/train/Success")
diff("data/val/Fail", "data/process/val/Fail")
diff("data/val/Success", "data/process/val/Success")
'''
diff("test_data", "test_process")

```

train.py

```

import os
import sys
import json
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, datasets
from tqdm import tqdm
from model import AlexNet
def main():
    # 选择设备
    device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
    # 图片预处理
    data_transform = {

```

```

        "transform": transforms.Compose([transforms.Resize((224, 224)),
                                         transforms.ToTensor(),
                                         transforms.Normalize((0.5, 0.5, 0.5),
(0.5, 0.5, 0.5))]))}
    # 训练集、验证集导入
    batch_size = 30
    nw = min([os.cpu_count(), batch_size if batch_size > 1 else 0, 8])
    train_dataset = datasets.ImageFolder(root="data/process/train",
transform=data_transform["transform"])
    train_num = len(train_dataset)
    train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=batch_size, shuffle=True, num_workers=nw)
    train_steps = len(train_loader)
    val_dataset = datasets.ImageFolder(root="data/process/val",
transform=data_transform["transform"])
    val_num = len(val_dataset)
    val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=4,
shuffle=False, num_workers=nw)
    # 字典{"Fail":0, "Success":1}
    result_list = train_dataset.class_to_idx
    cla_dict = dict((val, key) for key, val in result_list.items())
    json_str = json.dumps(cla_dict, indent=4)
    with open("class_indices.json", "w") as json_file:
        json_file.write(json_str)
    # 神经网络
    print("using {} images for training, {} images for
validation.".format(train_num, val_num))
    net = AlexNet(num_classes=2, init_weights=True)
    net.to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.0002)
    save_path = "AlexNet.pth"
    loss_function = nn.CrossEntropyLoss()
    best_acc = 0.0
    epochs = 20
    # 训练循环
    for epoch in range(epochs):
        net.train()
        running_loss = 0.0
        train_bar = tqdm(train_loader, file=sys.stdout)
        # 批次遍历
        for step, data in enumerate(train_bar):
            images, labels = data
            optimizer.zero_grad()
            outputs = net(images.to(device))

```

```

        loss = loss_function(outputs, labels.to(device))
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        train_bar.desc = "train epoch[{}/{}]
loss:{:.3f}".format(epoch + 1, epochs, loss)
    net.eval()
    acc = 0.0
    # 验证
    with torch.no_grad():
        val_bar = tqdm(val_loader, file=sys.stdout)
        for val_data in val_bar:
            val_images, val_labels = val_data
            outputs = net(val_images.to(device))
            predict_y = torch.max(outputs, dim=1)[1]
            acc += torch.eq(predict_y,
val_labels.to(device)).sum().item()
        val_accurate = acc / val_num
        print("[epoch %d] train_loss: %.3f  val_accuracy: %.3f" % (epoch
+ 1, running_loss / train_steps, val_accurate))
        if val_accurate > best_acc:
            best_acc = val_accurate
            torch.save(net.state_dict(), save_path)
    print("Finished Training")
if __name__ == "__main__":
    main()

```

predict.py

```

import os
import glob
import torch
from torchvision import transforms
import matplotlib.pyplot as plt
from model import AlexNet
from PIL import Image
def main():
    # 选择设备
    device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
    # 图片预处理
    data_transform = transforms.Compose([transforms.Resize((224, 224)),
transforms.ToTensor(),
transforms.Normalize((0.5, 0.5, 0.5),
(0.5, 0.5, 0.5))])
    # 加载训练好的模型

```

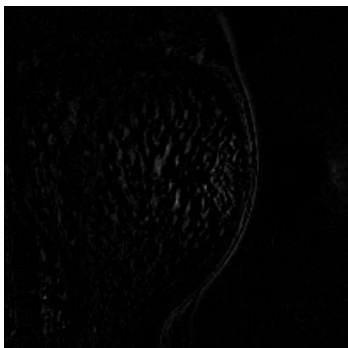
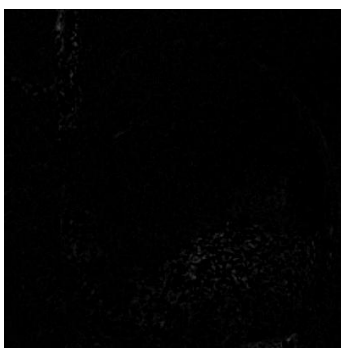
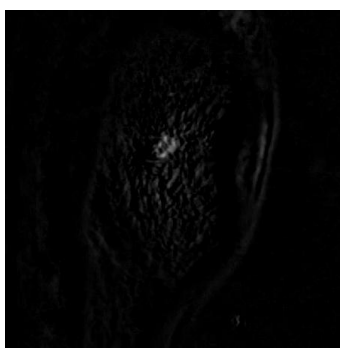
```

model = AlexNet(num_classes=2, init_weights=True).to(device)
weights_path = "AlexNet.pth"
model.load_state_dict(torch.load(weights_path))
model.eval()
# 测试集
folder_path = "test_process"
jpg_files = glob.glob(os.path.join(folder_path, "*.jpg"))
jpg_count = len(jpg_files)
for i in range(1, jpg_count+1):
    # 打开图片并预处理
    img_path = os.path.join(folder_path, "%s.jpg" % i)
    img = Image.open(img_path)
    plt.imshow(img)
    img = data_transform(img)
    img = torch.unsqueeze(img, dim=0)
    # 进行测试
    with torch.no_grad():
        output = torch.squeeze(model(img.to(device))).cpu()
        predict = torch.softmax(output, dim=0)
        predict_cla = torch.argmax(predict).numpy()
    print(predict_cla)
if __name__ == "__main__":
    main()

```

七、实验结果展示

将整体数据集用作测试，发现，除去失败的第四组图像，其他组图像都得到正确结果，正确率达到 98.75%。以下对差分图像进行对比：

组别	失败第四组	正常失败	正常成功（选取了白影较不明显的）
差分图像			

该错误可能是由拍摄图像序列时曝光度变化引起的，而在曝光度保持不变的情况下，本模型能够做到 100% 正确。

实验过程中，测试了多个 epoch 值，发现在 16 时，基本得到 100% 的测试准确率。这里将值设为 20，得到以下结果：

```
train epoch[20/20] loss:0.003: 100%|██████████| 2/2 [00:04<00:00, 2.44s/it]
100%|██████████| 5/5 [00:04<00:00, 1.07it/s]
[epoch 20] train_loss: 0.002 val_accuracy: 1.000
Finished Training
```

（文件夹里给出的模型 AlexNet.pth 即是在该条件下训练出来的）

在 epoch=50 时，最优的 train_loss 能稳定达到 0.001 的水平，而在 epoch=20 时，train_loss 可能会略大，但仍保持在 0.01 上下。

而从代码运行时间来看，整个训练过程（epoch=20，60 张训练集，20 张测试集）耗时 3 分钟左右，而在进行测试时，40 张图片仅需要 1 秒。另外，图像预处理的耗时也很短。总体而言，面对本题设环境，本模型可以给出高速、准确的结果。

八、分析

本代码利用 alexnet 神经网络对差分图像进行了分类，效果良好。

差分图像作为基础的图像组预处理手段，起到了很好的特征放大作用，这十分适应题设的要求。

alexNet 作为一个成熟的深度卷积神经网络（CNN），利用 ReLU 激活函数、Dropout、重叠最大池化等方式，提供了较为高速、准确的识别区分功能。其包含五个卷积层，三个池化层，而最后的全连接隐藏层则有效地防止过拟合。另外，使用 GPU 加速训练，也使得模型的训练更加高效。