**暨南大学本科实验报告专用纸**

课程名称 测试时增广与多模型结果集成 成绩评定

实验项目名称 数据增广 指导教师 林聪

实验项目编号 05 实验项目类型 实验地点

学生姓名 赵俊文 学号 2022104002

学院 智能科学与工程学院 系 人工智能 专业 人工智能

1. **实验目的**

了解和练习”测试时增广”（Test-Time Augmentation/TTA）和多模型结果集成(Ensemble).

1. **主要仪器设备**

**仪器：** PC

**实验环境：** Windows11,Python1.10,Pytorch11.8

1. **源程序**

源程序在实验步骤与调试中给出。

1. **实验步骤与调试**
2. **继承数据增广实验代码、模型和数据集**

**[源代码]**

#  1.1数据准备（有数据增广）

def prepare\_data\_with\_aug(data\_dir='D:/review&task/大三下/深度学习/Exp5/hotdog'):

    target\_size = (224, 224)

    torchvision\_transform = transforms.Compose([

        transforms.RandomRotation(90),

        transforms.RandomHorizontalFlip(),

        transforms.RandomVerticalFlip(),

        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),

    ])

    albumentations\_transform = A.Compose([

        A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1, p=0.5),

        A.OneOf([

            A.MotionBlur(p=0.2),

            A.MedianBlur(blur\_limit=3, p=0.1),

            A.Blur(blur\_limit=3, p=0.1),

        ], p=0.2),

        A.OneOf([

            A.OpticalDistortion(p=0.3),

            A.GridDistortion(p=0.1),

        ], p=0.2),

    ])

    train\_dataset = CombinedAugmentationDataset(

        os.path.join(data\_dir, 'train'),

        torch\_transform=torchvision\_transform,

        albu\_transform=albumentations\_transform,

        target\_size=target\_size

    )

    val\_transform = transforms.Compose([

        transforms.Resize(256),

        transforms.CenterCrop(224),

        transforms.ToTensor(),

        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

    ])

    val\_dataset = datasets.ImageFolder(

        os.path.join(data\_dir, 'val'),

        transform=val\_transform

    )

    train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True, num\_workers=4)

    val\_loader = DataLoader(val\_dataset, batch\_size=32, shuffle=False, num\_workers=4)

    return train\_loader, val\_loader, train\_dataset.classes

# 1.2 数据准备（无数据增广）

def prepare\_data\_without\_aug(data\_dir='D:/review&task/大三下/深度学习/Exp5/hotdog'):

    train\_transform = transforms.Compose([

        transforms.Resize(256),

        transforms.CenterCrop(224),

        transforms.ToTensor(),

        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

    ])

    val\_transform = transforms.Compose([

        transforms.Resize(256),

        transforms.CenterCrop(224),

        transforms.ToTensor(),

        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

    ])

    train\_dataset = datasets.ImageFolder(

        os.path.join(data\_dir, 'train'),

        transform=train\_transform

    )

    val\_dataset = datasets.ImageFolder(

        os.path.join(data\_dir, 'val'),

        transform=val\_transform

    )

    train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True, num\_workers=4)

    val\_loader = DataLoader(val\_dataset, batch\_size=32, shuffle=False, num\_workers=4)

    return train\_loader, val\_loader, train\_dataset.classes

1. **选择适合的TTA方式并在推理中实现TTA代码, 获取均值结果**

测试时增广（TTA）技术，核心思想是在测试阶段对同一图像应用多种数据增强变换并集成预测结果，从而提升模型性能。此函数的核心目标是在不改变模型的前提下，通过测试阶段的数据增强和结果融合提高分类准确率，主要步骤包括：

1. **预测流程**：

* 对原始图像进行一次预测。
* 对每个图像应用多种增强变换并分别预测。

1. **结果集成**：平均所有预测结果（原始 + 增强），取最大值作为最终预测。

**[源代码]**

3. 测试时增广(TTA)函数

def test\_time\_augmentation(model, dataloader, num\_aug=5):

    device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

    model = model.to(device)

    model.eval()

    # 定义TTA变换

    tta\_transforms = [

        transforms.Compose([

            transforms.Resize(256),

            transforms.CenterCrop(224),

            transforms.ToTensor(),

            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        ]),

        transforms.Compose([

            transforms.Resize(256),

            transforms.CenterCrop(224),

            transforms.RandomHorizontalFlip(p=1.0),

            transforms.ToTensor(),

            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        ]),

        transforms.Compose([

            transforms.Resize(256),

            transforms.CenterCrop(224),

            transforms.RandomVerticalFlip(p=1.0),

            transforms.ToTensor(),

            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        ]),

        transforms.Compose([

            transforms.Resize(256),

            transforms.CenterCrop(224),

            transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),

            transforms.ToTensor(),

            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        ]),

        transforms.Compose([

            transforms.Resize(256),

            transforms.CenterCrop(224),

            transforms.RandomRotation(30),

            transforms.ToTensor(),

            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        ])

    ]

    # 限制使用的TTA变换数量

    tta\_transforms = tta\_transforms[:num\_aug]

    correct = 0

    total = 0

    with torch.no\_grad():

        for inputs, labels in tqdm(dataloader):

            inputs, labels = inputs.to(device), labels.to(device)

            # 存储所有增强版本的预测

            all\_outputs = []

            # 原始图像预测

            outputs = model(inputs)

            all\_outputs.append(outputs)

            # 应用TTA变换

            for transform in tta\_transforms:

                # 对batch中的每个图像应用变换

                augmented\_inputs = torch.stack(

                    [transform(Image.fromarray((x.cpu().permute(1, 2, 0).numpy() \* 255).astype(np.uint8))) for x in

                     inputs])

                augmented\_inputs = augmented\_inputs.to(device)

                outputs = model(augmented\_inputs)

                all\_outputs.append(outputs)

            # 平均所有预测

            avg\_output = torch.mean(torch.stack(all\_outputs), dim=0)

            \_, predicted = torch.max(avg\_output.data, 1)

            total += labels.size(0)

            correct += (predicted == labels).sum().item()

    accuracy = 100 \* correct / total

    return accuracy

1. **分别用数据增广(DA)训练 ResNet-18和EfficientNet, 实现TTA, 并将结果集成输出**

**[源代码]**

4. 模型集成函数

def ensemble\_models(models: List[nn.Module], dataloader, use\_tta=False, num\_aug=5):

    device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

    for model in models:

        model.to(device)

        model.eval()

    if use\_tta:

        tta\_transforms = [

            transforms.Compose([

                transforms.Resize(256),

                transforms.CenterCrop(224),

                transforms.ToTensor(),

                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

            ]),

            transforms.Compose([

                transforms.Resize(256),

                transforms.CenterCrop(224),

                transforms.RandomHorizontalFlip(p=1.0),

                transforms.ToTensor(),

                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

            ]),

            transforms.Compose([

                transforms.Resize(256),

                transforms.CenterCrop(224),

                transforms.RandomVerticalFlip(p=1.0),

                transforms.ToTensor(),

                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

            ])

        ]

        tta\_transforms = tta\_transforms[:num\_aug]

    else:

        tta\_transforms = []

    correct = 0

    total = 0

    with torch.no\_grad():

        for inputs, labels in tqdm(dataloader):

            inputs, labels = inputs.to(device), labels.to(device)

            all\_outputs = []

            for model in models:

                if use\_tta:

                    model\_outputs = []

                    # 原始图像预测

                    outputs = model(inputs)

                    model\_outputs.append(outputs)

                    # TTA预测

                    for transform in tta\_transforms:

                        augmented\_inputs = torch.stack([

                            transform(Image.fromarray((x.cpu().permute(1, 2, 0).numpy() \* 255).astype(np.uint8)))

                            for x in inputs

                        ])

                        augmented\_inputs = augmented\_inputs.to(device)

                        outputs = model(augmented\_inputs)

                        model\_outputs.append(outputs)

                        # 应该在所有TTA变换完成后计算平均

                    avg\_output = torch.mean(torch.stack(model\_outputs), dim=0)

                    all\_outputs.append(avg\_output)

                else:

                    outputs = model(inputs)

                    all\_outputs.append(outputs)

            # 平均所有模型的预测

            ensemble\_output = torch.mean(torch.stack(all\_outputs), dim=0)

            \_, predicted = torch.max(ensemble\_output.data, 1)

            total += labels.size(0)

            correct += (predicted == labels).sum().item()

        accuracy = 100 \* correct / total

    return accuracy

# 3. 训练和验证函数

def train\_model(model, train\_loader, val\_loader, criterion, optimizer, num\_epochs=10, model\_name='model'):

    device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

    model = model.to(device)

    history = {

        'train\_acc': [],

        'val\_acc': [],

        'train\_loss': [],

        'val\_loss': []

    }

    best\_val\_acc = 0.0

    best\_model\_path = f'best\_{model\_name}.pth'

    for epoch in range(num\_epochs):

        print(f'Epoch {epoch + 1}/{num\_epochs}')

        print('-' \* 10)

        # 训练阶段

        model.train()

        running\_loss = 0.0

        running\_corrects = 0

        for inputs, labels in tqdm(train\_loader):

            inputs = inputs.to(device)

            labels = labels.to(device)

            optimizer.zero\_grad()

            outputs = model(inputs)

            \_, preds = torch.max(outputs, 1)

            loss = criterion(outputs, labels)

            loss.backward()

            optimizer.step()

            running\_loss += loss.item() \* inputs.size(0)

            running\_corrects += torch.sum(preds == labels.data)

        epoch\_loss = running\_loss / len(train\_loader.dataset)

        epoch\_acc = running\_corrects.double() / len(train\_loader.dataset)

        history['train\_loss'].append(epoch\_loss)

        history['train\_acc'].append(epoch\_acc.item())

        print(f'Train Loss: {epoch\_loss:.4f} Acc: {epoch\_acc:.4f}')

        # 验证阶段

        model.eval()

        val\_loss = 0.0

        val\_corrects = 0

        with torch.no\_grad():

            for inputs, labels in val\_loader:

                inputs = inputs.to(device)

                labels = labels.to(device)

                outputs = model(inputs)

                \_, preds = torch.max(outputs, 1)

                loss = criterion(outputs, labels)

                val\_loss += loss.item() \* inputs.size(0)

                val\_corrects += torch.sum(preds == labels.data)

        val\_epoch\_loss = val\_loss / len(val\_loader.dataset)

        val\_epoch\_acc = val\_corrects.double() / len(val\_loader.dataset)

        history['val\_loss'].append(val\_epoch\_loss)

        history['val\_acc'].append(val\_epoch\_acc.item())

        print(f'Val Loss: {val\_epoch\_loss:.4f} Acc: {val\_epoch\_acc:.4f}\n')

        # 保存最佳模型

        if val\_epoch\_acc > best\_val\_acc:

            best\_val\_acc = val\_epoch\_acc

            torch.save(model.state\_dict(), best\_model\_path)

    # 加载最佳模型

    model.load\_state\_dict(torch.load(best\_model\_path))

    return history, model

# 主函数

def main():

    # 参数设置

    data\_dir = 'D:/review&task/大三下/深度学习/Exp5/hotdog'

    num\_epochs = 20

    lr = 0.001

    # 准备数据

    print("\nPreparing data...")

    train\_loader\_no\_aug, val\_loader\_no\_aug, class\_names = prepare\_data\_without\_aug(data\_dir)

    train\_loader\_aug, val\_loader\_aug, \_ = prepare\_data\_with\_aug(data\_dir)

    print(f"Class names: {class\_names}")

    # 训练ResNet-18无数据增广

    print("\nTraining ResNet-18 without data augmentation...")

    resnet\_no\_aug = build\_model('resnet18')

    criterion = nn.CrossEntropyLoss()

    optimizer = optim.SGD(resnet\_no\_aug.parameters(), lr=lr, momentum=0.9)

    history\_resnet\_no\_aug, resnet\_no\_aug = train\_model(resnet\_no\_aug, train\_loader\_no\_aug, val\_loader\_no\_aug,

                                                       criterion, optimizer, num\_epochs, 'resnet18\_no\_aug')

    # 训练ResNet-18有数据增广

    print("\nTraining ResNet-18 with data augmentation...")

    resnet\_aug = build\_model('resnet18')

    optimizer = optim.SGD(resnet\_aug.parameters(), lr=lr, momentum=0.9)

    history\_resnet\_aug, resnet\_aug = train\_model(resnet\_aug, train\_loader\_aug, val\_loader\_aug,

                                                 criterion, optimizer, num\_epochs, 'resnet18\_aug')

    # 训练EfficientNet有数据增广

    print("\nTraining EfficientNet with data augmentation...")

    efficientnet\_aug = build\_model('efficientnet')

    optimizer = optim.SGD(efficientnet\_aug.parameters(), lr=lr, momentum=0.9)

    history\_efficientnet\_aug, efficientnet\_aug = train\_model(efficientnet\_aug, train\_loader\_aug, val\_loader\_aug,

                                                             criterion, optimizer, num\_epochs, 'efficientnet\_aug')

    # 测试准确率

    print("\nEvaluating models...")

    # 基本测试准确率

    resnet\_no\_aug\_acc = ensemble\_models([resnet\_no\_aug], val\_loader\_no\_aug)

    resnet\_aug\_acc = ensemble\_models([resnet\_aug], val\_loader\_aug)

    efficientnet\_aug\_acc = ensemble\_models([efficientnet\_aug], val\_loader\_aug)

    # TTA测试准确率

    resnet\_aug\_tta\_acc = ensemble\_models([resnet\_aug], val\_loader\_aug, use\_tta=True)

    efficientnet\_aug\_tta\_acc = ensemble\_models([efficientnet\_aug], val\_loader\_aug, use\_tta=True)

    # 模型集成

    ensemble\_acc = ensemble\_models([resnet\_aug, efficientnet\_aug], val\_loader\_aug, use\_tta=True)

    # 打印结果表格

    print("\nResults Table:")

    print("| Model                                   | Train Accuracy | Test Accuracy |")

    print("|-----------------------------------------|----------------|---------------|")

    print(

        f"| ResNet-18                               | {max(history\_resnet\_no\_aug['train\_acc']):.4f}         | {resnet\_no\_aug\_acc:.4f}      |")

    print(

        f"| ResNet-18 + DA                          | {max(history\_resnet\_aug['train\_acc']):.4f}         | {resnet\_aug\_acc:.4f}      |")

    print(

        f"| EfficientNet + DA                       | {max(history\_efficientnet\_aug['train\_acc']):.4f}         | {efficientnet\_aug\_acc:.4f}      |")

    print(f"| ResNet-18 + DA + TTA                    | N/A            | {resnet\_aug\_tta\_acc:.4f}      |")

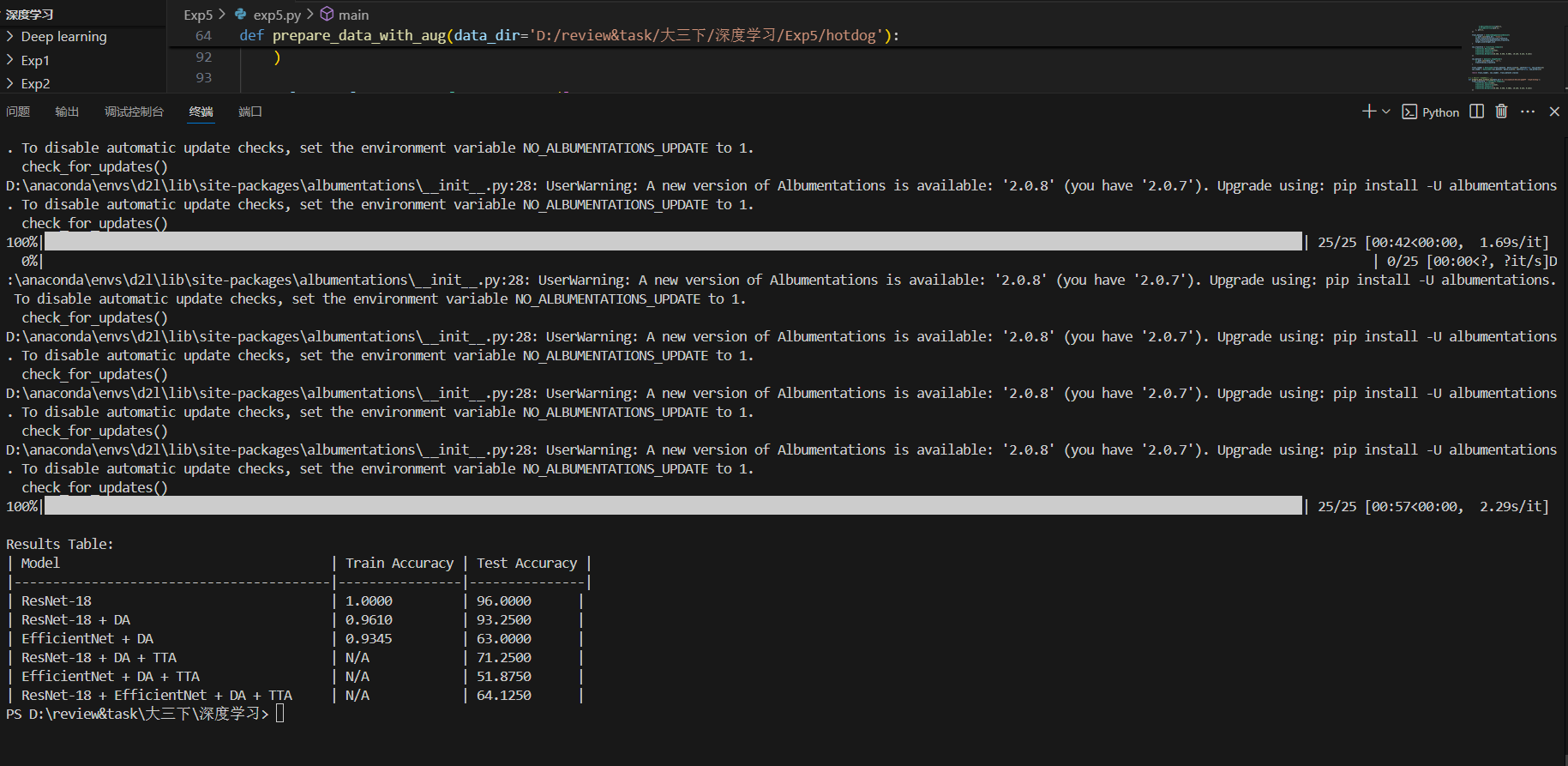
    print(f"| EfficientNet + DA + TTA                 | N/A            | {efficientnet\_aug\_tta\_acc:.4f}      |")

    print(f"| ResNet-18 + EfficientNet + DA + TTA     | N/A            | {ensemble\_acc:.4f}      |")

if \_\_name\_\_ == '\_\_main\_\_':

    main()

1. **实验结果与分析**
2. **将结果集成输出**



1. **相关实验结果填入下表**

|  |  |  |
| --- | --- | --- |
| **模型** | **训练准确率** | **测试准确率** |
| **ResNet-18** | **1.0000** | **0.9600** |
| **ResNet-18 + DA** | **0.9618** | **0.9325** |
| **EfficientNet + DA** | **0.9345** | **0.6300** |
| **ResNet-18 + DA + TTA** | **N/A** | **0.7125** |
| **EfficientNet + DA + TTA** | **N/A** | **0.51875** |
| **ResNet-18 + EfficientNet + DA + TTA** | **N/A** | **0.63125** |