中山大学计算机学院

人工智能

本科生实验报告

（2025 学年春季学期）

课程名称：Artificial Intelligence

教学班级		专业（方向）	
学号		姓名	

一、 实验题目

深度学习：中药图片分类任务

利用 pytorch 框架搭建神经网络实现中药图片分类，其中中药图片数据分为训练集 train 和测试集 test，训练集仅用于网络训练阶段，测试集仅用于模型的性能测试阶段。训练集和测试集均包含五种不同类型的中药图片：baihe、dangshen、gouqi、huaihua、jinyinhua。请合理设计神经网络架构，利用训练集完成网络训练，统计网络模型的训练准确率和测试准确率，画出模型的训练过程的 loss 曲线、准确率曲线。

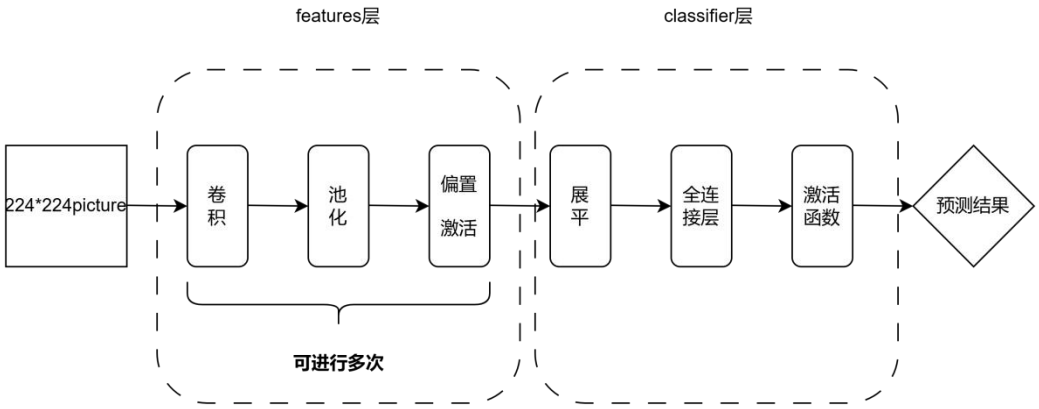
二、 实验内容

1. 算法原理

1.1 卷积神经网络 CNN

卷积神经网络（Convolutional Neural Networks，简称 CNN）是一种具有**局部连接、权值共享**特点的深层前馈神经网络（Feedforward Neural Networks），擅长处理图像特别是图像识别等相关机器学习问题，比如图像分类、目标检测、图像分割。

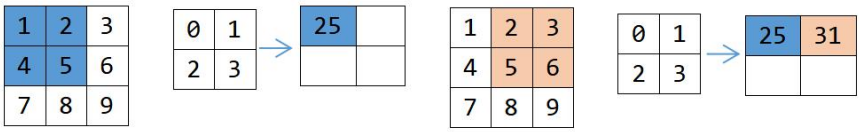
卷积神经网络的基本结构大致包括：**卷积层、激活函数、池化层、全连接层、输出层**等。



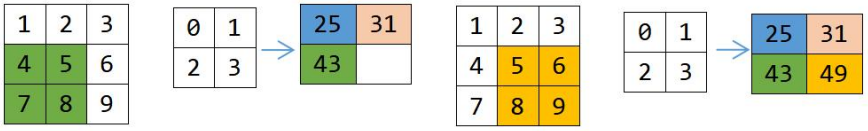
卷积神经网络通过多次的卷积、池化操作得到了原始图像的特征图，将图片一块区域中的参数结合起来，达到了减少训练参数的效果，加强了图片中像素的连续性

1.2 卷积 Convolution

根据是先给出的卷积核和步长对原始的矩阵进行卷积操作，就是将对应位置区域与卷积核的权重进行加权求和后得到新的特征值，直到原始的矩阵中没有新的卷积块为止，如下图所示，就是一个 3*3 的矩阵和 2*2 的卷积核进行卷积得到的一个更小的特征图.



0*1+1*2+2*4+3*5=25　　0*2+1*3+2*5+3*6=31

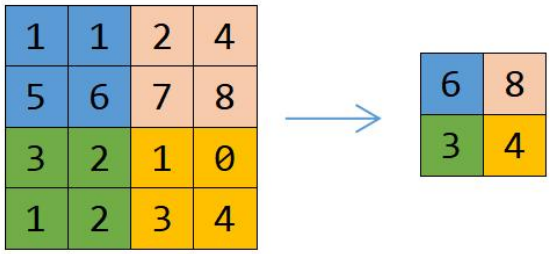


0*4+1*5+2*7+3*8=43　　0*5+1*6+2*8+3*9=49

1.3 最大池化 Max Pool

最大池化，想卷积一样，选择一个移动窗口在输入矩阵上滑动，滑动过程中去这个窗口中数据矩阵上最大值作为输出，得到了一个更小的特征图

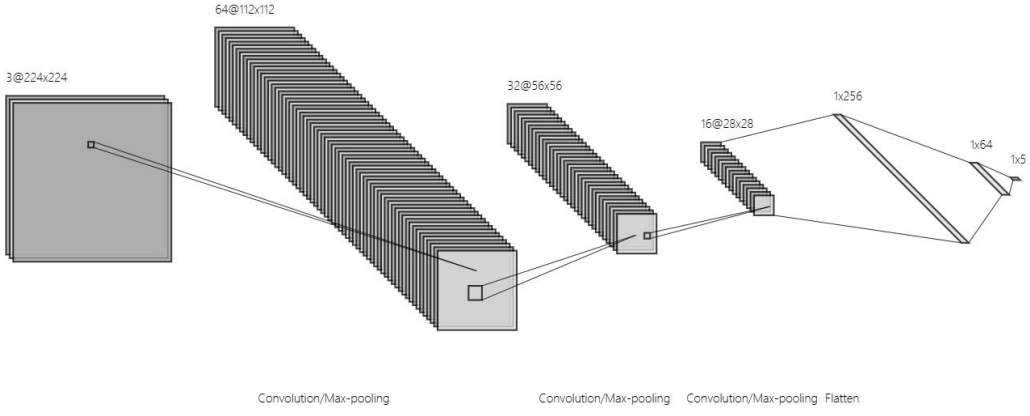
如下图所示，是用 2*2 的窗口进行最大池化操作使得新的特征图缩小了一半



Max Pool with 2×2 filters and stride 2

1.4 本次实验的卷积神经网络 CNN 的设置

示意图如下：



根据 CNN 的结构，本次实验采用了如下的参数设置：

- 使用 **3 个卷积层**:实现了如下的特征抓取
[3,224,224]->[64,112,112]->[32,56,56]->[16,28,28]
- 使用 **3 个全连接层**作为神经网络进行分类
- 使用 softmax 后的 **CrossEntropyLoss 损失函数**
- 使用 **Adam 优化器**
- 使用 **ReduceLROnPlateau 学习率调度器**

1.5 数据的预处理

本次实验选择通过随机选取的方式将数据集分成了训练集、验证集、测试集
然后对训练集进行数据增强和标准化处理的，提升模型的泛化能力：

- **随机裁剪**图像的一部分:让模型学会在图像不同局部特征下也能做出正确判断
- 以 50%的概率**水平或者垂直翻转**图像：增加数据多样性，适应物体在不同方向上的外观，尤其是左右对称或者非对称但常见方向变化的物体。
- 随机将图像进行**旋转**:提升模型对图像轻微旋转变化的适应能力，比如拍摄角度略有不同的情况。
- 随机进行**颜色抖动**:增强模型对色彩变化的鲁棒性，
- 给定的均值和标准差对每个通道（RGB）进行**标准化**，使得数据均值为 0，标准差为 1

对于验证集、测试集数据，不进行处理。

2. 伪代码

```
Algorithm: Classification of Medicine Pictures
input:train_dataset,val_dataset,test_dataset
output:trained model
Initialize: CNNmodel
            scheduler:ReduceLROnPlateau
            criterion:CrossEntropyLoss
            optimizer:Adam
for epoch=0 to epochs do
    for image,label in train_loader do
        output ← CNNmodel(image)
        loss ← criterion(output,label)
        correct ← if predicted == label
        backward
        update parameters ← optimizer
    end for
    val_acc,val_loss ← evaluate(val_loader)
    lr ← scheduler(val_acc)
end for
return model
```

3. 关键代码展示

3.1 数据预处理与加载

3.1.1 数据的获取

```
# 数据集的获取和处理

## 继承 Dataset 类用于测试集图片的加载

class TestDataset(Dataset):

    def __init__(self, root, classes, transform=None):

        self.root = root

        self.transform = transform

        self.image_files = os.listdir(root)

        self.classes = classes

        #self.classes = sorted(set([f[:-6] for f in self.image_files]))

        self.class_to_idx = {c: i for i, c in enumerate(self.classes)}

    def __len__(self):

        return len(self.image_files)

    def __getitem__(self, idx):

        if torch.is_tensor(idx):

            idx = idx.tolist()

        img_name = os.path.join(self.root, self.image_files[idx])

        image = Image.open(img_name).convert('RGB')

        if self.transform:

            image = self.transform(image)

        label = self.class_to_idx[self.image_files[idx][:-6]]

        return image, label, self.image_files[idx]

    ## 加载图片数据集

    dataset=ImageFolder(root='cnn 图片/train',transform=transform)

    ## 加载测试集的 10 张照片

    test_dataset=TestDataset(root='cnn 图片/test',

         classes=dataset.classes,

         transform=val_transform

    )
```

3.1.2 数据预处理

```
    transform = transforms.Compose([
    # 调整短边为 256 像素
    transforms.Resize(256),
    # 数据增强
    ## 随机裁剪并缩放
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    ## 50%概率水平翻转
    transforms.RandomHorizontalFlip(),
    ## 随机旋转±15 度
    transforms.RandomRotation(15),
    ## 颜色抖动
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3),
    # 转换为张量
    transforms.ToTensor(),
    # 归一化
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
])
## 验证集的数据预处理
val_transform = transforms.Compose([
    transforms.Resize(256),     # 调整短边为 256 像素
    transforms.CenterCrop(224), # 从中心裁剪 224x224 图像
    transforms.ToTensor(),      # 转换为张量
    transforms.Normalize(       # 归一化
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

3.1.3 划分数据集

```
## 划分训练集为训练集和验证集
def split_dataset(dataset,train_percent):
    # 按类别分组数据
    class_data = {}
    for i, (_, label) in enumerate(dataset):
        if label not in class_data:
            class_data[label] = []
        class_data[label].append(i)
    # 划分训练集和验证集
    train_indices = []
    val_indices = []
    for label, indices in class_data.items():
        random.shuffle(indices)
        split_index = int(len(indices) * train_percent)
        train_indices.extend(indices[:split_index])
        val_indices.extend(indices[split_index:])
    # 创建训练集和验证集的 Subset 对象
    train_dataset = Subset(dataset, train_indices)
    val_dataset = Subset(dataset, val_indices)
    return train_dataset,val_dataset
```

3.1.4 创建数据集加载器 DataLoader

```
    # 加载数据 loading data
    ## 按照比例分层划分数据集为训练集和验证集
    train_dataset,val_dataset=split_dataset(dataset,0.85)
    val_dataset.dataset.transform = val_transform
    ## 创建数据加载器 dataset_loader
    train_loader=DataLoader(train_dataset,batch_size=64,shuffle=True)
    val_loader=DataLoader(val_dataset,batch_size=16,shuffle=True)
    test_loader=DataLoader(test_dataset,batch_size=1,shuffle=True)
```

3.2 利用 pytorch 框架搭建 CNN 模型

3.2.1 继承 nn.Module 搭建 CNN 框架

```
# 使用 pytorch 框架建立 CNN 模型处理图片分类任务
class CNN(nn.Module):
    def __init__(self,classes=5):
        super(CNN,self).__init__()
        # 卷积层提取图片的特征
        self.features=nn.Sequential(
                # 第一层卷积:(3,224,224)->(64,112,112)
                nn.Conv2d(in_channels=3,
                        out_channels=64,
                        kernel_size=5,
                        stride=1,
                        padding=2
                        ),
                nn.BatchNorm2d(64),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2,
                            stride=2
                            ),
                # 第二层卷积:(64,112,112)->(32,56,56)
                nn.Conv2d(64,32,3,1,1),
                nn.BatchNorm2d(32),
                nn.ReLU(),
                nn.MaxPool2d(2,2),
                # 第三层卷积:(32,56,56)->(16,28,28)
                nn.Conv2d(32,16,3,1,1),
                nn.BatchNorm2d(16),
                nn.ReLU(),
                nn.MaxPool2d(2,2)
```

```
        )
        # 全连接分类层，总共三层全连接层进行分类
        self.classifier=nn.Sequential(
                # 第一层
                nn.Linear(16*28*28,256),
                nn.ReLU(),
                nn.Dropout(0.5),
                # 第二层
                nn.Linear(256,64),
                nn.ReLU(),
                nn.Dropout(0.3),
                # 第三层
                nn.Linear(64,classes)
        )
    # 前向传播
    def forward(self,x):
        # 卷积层提取图片的特征减少处理的特征
        x=self.features(x)
        # 展平到一维
        x=torch.flatten(x, 1)
        # 输入到全连接层分类
        x=self.classifier(x)
        return x
```

3.2.2 损失函数、优化器、学习率调度器

```
# 选择 CrossEntropyLoss 损失函数
criterion=nn.CrossEntropyLoss()
# 使用 Adam 优化器
optimizer=torch.optim.Adam(model.parameters(),lr=1e-4)
# 学习率调度器 ReduceLROnPlateau
scheduler=torch.optim.lr_scheduler.ReduceLROnPlateau(
```

```
        optimizer=optimizer,
        mode='max',
        factor=0.5,
        patience=5,
        verbose=True,
        min_lr=1e-6
        )
```

3.3 模型训练和测试

3.3.1 模型的训练

```
# 模型训练
def train(epochs,train_loader,val_loader,test_loader):
    # 记录训练过程中的 train、val 的损失值和准确率
    train_losses=[]
    val_losses=[]
    train_accs=[]
    val_accs=[]
    best_acc=0
    cnt=0
    # 训练主循环
    for epoch in range(epochs):
        epoch_loss=0
        total=0
        correct=0
        for image,label in train_loader:
            # 将数据移动至 GPU
            image,label=image.to(device),label.to(device)
            optimizer.zero_grad()
            # 前向传播
            output=model(image)
```

```
            # 计算当前批次的损失值
            loss=criterion(output,label)
            epoch_loss+=loss.item()
            # 计算当前批次的预测结果
            _,predicted=torch.max(output.data, 1)
            # 将当前批次的数量加到总数中
            total+=label.size(0)
            # 将预测正确的数量加入总数中
            correct+=(predicted == label).sum().item()
            # 向后传播
            loss.backward()
            # 更新参数
            optimizer.step()
        # 计算这次循环的 loss 和 acc
        train_losses.append(epoch_loss/len(train_loader))
        train_acc=100*correct/total
        train_accs.append(train_acc)
        # 计算 val 验证集的 loss 和 acc
        val_acc,val_loss=evaluate(val_loader)
        val_losses.append(val_loss)
        val_accs.append(val_acc)
        # 打印当前 epoch 的结果
        print(f'Epoch {epoch + 1}/{epochs}, Train_Loss: {epoch_loss/len(trai
n_loader):.4f}, Val_Loss: {val_loss/len(val_loader):.4f}, Train_Acc: {train
_acc}, Val_Acc: {val_acc}')
        # 更具 val 的 acc 调整学习率
        scheduler.step(val_acc)
        # 早停机制
        if val_acc > best_acc:
            best_acc=val_acc
            # 保存模型
```

```
            torch.save(model.state_dict(),'model.pth')
            cnt = 0
        else:
            cnt+=1
        if cnt >= 25:
            break
    #torch.save(model.state_dict(), 'last_model.pth')
    return train_losses,train_accs,val_losses,val_accs
```

3.3.2 模型的验证和评估

```
# 模型评估验证
def evaluate(val_loader):
    model.eval()
    with torch.no_grad():
        correct=0
        loss=0
        total=0
        for image,label, in val_loader:
            image,label=image.to(device), label.to(device)
            output=model(image)
            _, predicted = torch.max(output.data, 1)
            total+=label.size(0)
            correct+=(predicted == label).sum().item()
            loss+=criterion(output,label).item()
    return 100*correct/total, loss/total
```

3.3.3 使用模型预测测试集

```
# 使用模型进行预测
def predict(test_loader):
    model.load_state_dict(torch.load('model.pth'))
    model.eval()
```

```
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels, img_names in test_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            for i in range(len(predicted)):
                label_class = dataset.classes[labels[i].item()]
                predicted_class = dataset.classes[predicted[i].item()]
                if predicted_class == label_class:
                    correct += 1
                total += 1
    print(f'图片名: {img_names[i]:<15} 预测类别: {predicted_class:<9} 实际类别: {label_class}')
    accuracy = 100*correct/total
    print(f'测试集的正确率为: {accuracy:.2f}%')
    return accuracy
```

4. 创新点&优化

1．使用了**动态学习率调度器**，使用 ReduceLROnPlateau 学习率调度器

2．**早停机制**，保存在验证集（val）上 Acc 最高的模型，我们每个 epoch 对验证集的准确率 Acc 进行计算，保存 Acc 最高的一次 epoch 的模型，如果连续 epoch/2 个循环验证集的 Acc 都没有增加，那么我们就停止训练，这样有效的减少训练的时间成本和减少训练的过拟合程度

3．对训练集进行**数据增强**，增强训练的鲁棒性，对原始数据集除了分辨率的统一之外，还加入了随机裁剪缩放、随机水平或者垂直翻转、随机旋转、颜色抖动等数据增强手段，提高了数据集的鲁棒性，增强了训练模型的泛化能力

三、 实验结果及分析

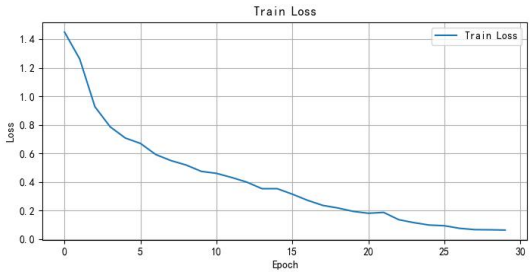
1. 实验结果展示

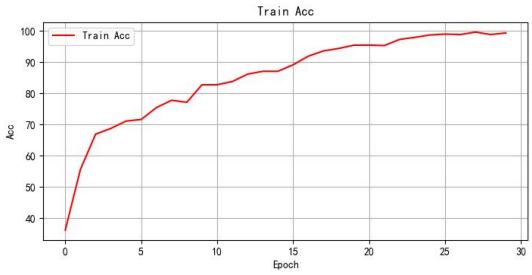
1.1 训练过程中的损失值和准确率

在**训练过程中 Loss 值不断的下降，同时训练集的准确率不断的提高**，在训练的最后，**训练集的准确率从 Acc 已经接近了 100%**，说明模型已经完全的学习了训练集的图片的特征

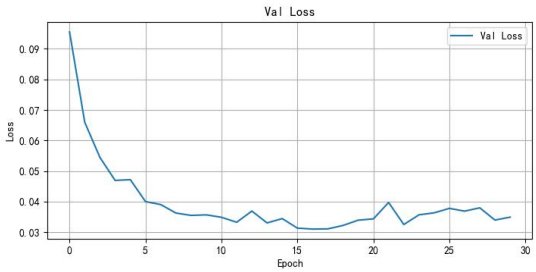
但是我们发现在验证集 val 中对模型进行验证的时候，发现最后一个 epoch 的 Loss 和 Acc 并不是最优的，这里出现了过拟合的现象，所以我们的早停机制会即使的停止训练，并且保存在训练过程中 val_acc 最优的一次 epoch 的模型参数作为我们的结果。

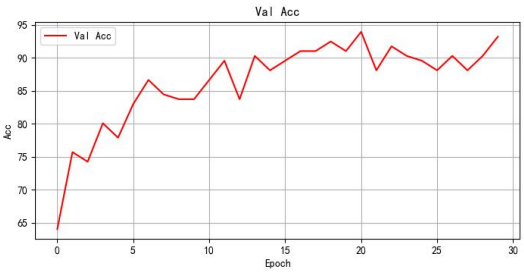
训练集的 Loss 曲线和准确率 Acc 曲线





验证集的 Loss 曲线和准确率 Acc 曲线





最后 10 次 epoch 的训练结果

```
Epoch 21/50, Train_Loss: 0.1801, Val_Loss: 0.0038, Train_Acc: 95.42483660130719, Val_Acc: 93.94160583941606
Epoch 22/50, Train_Loss: 0.1861, Val_Loss: 0.0044, Train_Acc: 95.29411764705883, Val_Acc: 88.10218978102189
Epoch 23/50, Train_Loss: 0.1348, Val_Loss: 0.0036, Train_Acc: 97.25490196078431, Val_Acc: 91.75182481751825
Epoch 24/50, Train_Loss: 0.1140, Val_Loss: 0.0040, Train_Acc: 97.90849673202614, Val_Acc: 90.2919708029197
Epoch 25/50, Train_Loss: 0.0972, Val_Loss: 0.0040, Train_Acc: 98.69281045751634, Val_Acc: 89.56204379562044
Epoch 26/50, Train_Loss: 0.0924, Val_Loss: 0.0042, Train_Acc: 98.95424836601308, Val_Acc: 88.10218978102189
Epoch 27/50, Train_Loss: 0.0743, Val_Loss: 0.0041, Train_Acc: 98.82352941176471, Val_Acc: 90.2919708029197
Epoch 28/50, Train_Loss: 0.0650, Val_Loss: 0.0042, Train_Acc: 99.6078431372549, Val_Acc: 88.10218978102189
Epoch 29/50, Train_Loss: 0.0639, Val_Loss: 0.0038, Train_Acc: 98.82352941176471, Val_Acc: 90.2919708029197
Epoch 30/50, Train_Loss: 0.0621, Val_Loss: 0.0039, Train_Acc: 99.34640522875817, Val_Acc: 93.21167883211679
```

1.2 测试集的预测结果

可以看到通过我们模型对测试集的十张照片进行预测，可以看到我们的模型能够完美的通过测试集，获得了 **100%的正确率**。

```
图片名: huaihua01.jpg   预测类别: huaihua   实际类别: huaihua
图片名: jinyinhua01.jpg 预测类别: jinyinhua 实际类别: jinyinhua
图片名: gouqi02.jpg     预测类别: gouqi     实际类别: gouqi
图片名: baihe01.jpg     预测类别: baihe     实际类别: baihe
图片名: gouqi01.jpg     预测类别: gouqi     实际类别: gouqi
图片名: dangshen01.jpg  预测类别: dangshen  实际类别: dangshen
图片名: dangshen02.jpg  预测类别: dangshen  实际类别: dangshen
图片名: huaihua02.jpg   预测类别: huaihua   实际类别: huaihua
图片名: baihe02.jpg     预测类别: baihe     实际类别: baihe
图片名: jinyinhua02.jpg 预测类别: jinyinhua 实际类别: jinyinhua
测试集的正确率为: 100.00%
```

2. 评测指标展示及分析

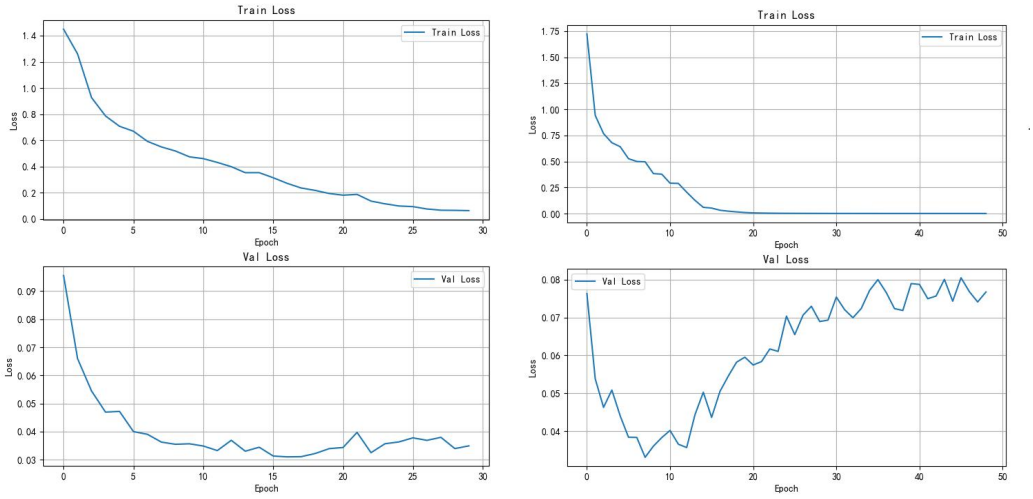
2.1 不同参数的训练效果对比

模型网络结构	学习率 lr	训练次数	测试集准确率
3 卷积*3 全连接层	**1e-4**	**30**	**100%**
3 卷积*3 全连接层	1e-3	39	90%
3 卷积*3 全连接层	1e-5	26	90%
1 卷积*3 全连接层	1e-4	36	70%
3 卷积*1 全连接层	1e-4	29	90%

2.2 消融实验

2.2.1 无早停机制

左边是本次实验代码的结果，右边是没有早停机制的结果，我们可以发现**没有早停机制的训练结果**的 val loss 值在后期上升很明显，**呈现出严重的过拟合现象**，这说明了我们**早停机制的有效性**



2.2.2 无数据增强

左边是本次实验代码的结果，右边是没有数据增强的结果，可以发现**没有数据增强的结果的验证集的准确率收敛在 80%左右，而左边的验证集准确率收敛在了 95%左右**，说明我们的**数据增强增强了模型训练的鲁棒性，提高了模型泛化能力**。