Tiny-imagenet-200 分类任务实验报告

吴晓雪 2020201516

Jan.14th 2023

1 摘要

本实验通过深度学习导论课堂中提到的图像分类任务相关的各种卷积神经网络、数据增强、MAE 等处理方法，结合相关论文，对 tiny-imagenet-200 数据集完成了两个图像分类任务。其中任务一采用了 ResNet、DenseNet 等网络结构，任务二只能使用一小部分数据，采用半监督学习的方法，通过 AutoEncoder 网络结构，结合 MAE 以及数据增强的相关技巧来完成训练，最终分别取得了 62.8% 和 24.5% 左右的分类准确率。

2 引言

本实验作为深度学习导论的作业，针对图像分类任务提出了两个任务，均围绕 Tiny-imagenet-200 数据集展开。Tiny-imagenet-200 数据集属于 Imagenet 数据集的子集，共有 200 类图片，其中训练集共有 100,000 张图片，验证集和测试集有 10,000 张图片，由于只提供了验证集的数据标签，本实验将以验证集的分类准确率作为评价的标准。**可以通过表1和2来直接查看结果，便于进行评价。**

对于两个任务，任务一使用训练集的全部数据用于模型优化，尽管如此，训练数据仍然较少，实验中采用了部分数据增强的技巧来使模型更加泛化。首先实验前期采用了典型的 CNN 网络结构 AlexNet，以其分类结果作为 baseline，再通过结合课堂知识与相关论文 [3][5]，分别构建了 ResNet、DenseNet 等网络，辅以不同的训练技巧，最终实现 62.77% 的准确率。任务中涉及的所有模型和训练均不引入其他数据，也不使用已知的预训练大模型，而是从头开始。

而任务二所使用的训练数据仅仅为任务一的十分之一，即训练数据量与测试数据量一致，这意味着将要使用数据增强以及 MAE 类似的无监督学习方法来避免模型的过拟合，加强泛化性。实验中以 ResNet 为基础构建了 AutoEncoder，经过“重构误差最小化”与“分类误差最小化”两个阶段的训练，最终得到 24.5% 的准确率。下面将依照流程细节地记录实验过程。

3 任务一实验记录

为了便于比较任务一搭建的模型效果，以及直观对实验成果查看，下面在表1中将各模型的训练轮数、参数量以及准确率展示出来。

下面分别从不同的模型角度来讨论模型设计和训练方法细节。

模型	训练轮数	参数量	正确率
AlexNet	30e	12,695,784	45.68%
ResNet	30e	**5,013,640**	55.08%
ResNet	50e	**5,013,640**	55.65%
Inception-ResNet	52e	7,156,184	48.91%
DenseNet	27e	11,809,672	60.28%
DenseNet	71e	11,809,672	**62.77**%

表 1: 不同模型的不同信息，其中同样的模型在不同训练轮数下的正确率也被列出

3.1 AlexNet

AlexNet[1] 属于卷积神经网络中比较基础且经典的模型，在实验前期主要是通过 AlexNet 的预测结果来给出一个 baseline，以便对后续的模型改进进行比较。

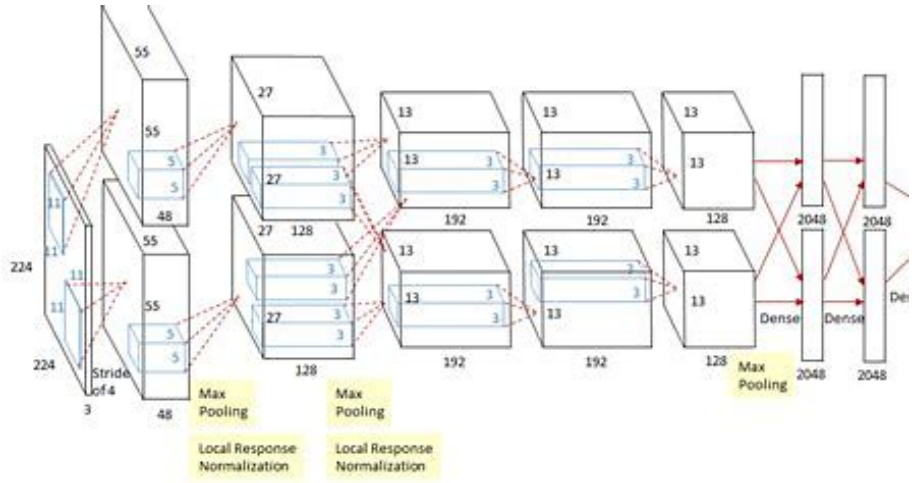

图 1: AlexNet 模型结构

经典的网络对应结构如图1所示，由于本任务数据集中图片大小为 $3 \times 64 \times 64$，模型结构需要做一定的微调，但由于全连接层的存在，模型参数量依然相对较大，可能容易导致过拟合的问题，在表1中也可以看出这一点。在训练至 loss 值稳定，正确率开始下降时，根据提前停止策略截取该模型，进行预测，得到 45% 左右的准确率，后续的任务一将这一结果作为 baseline。

值得注意的是，在该模型上，SGDM 作为优化器的效果优于 Adam，因此后续实验也将 SGDM 作为优化器中重要的备选项。训练过程的准确率图像如图2，其中虚线意味学习率的调整，从这里就已经看出，在有过拟合问题时调低学习率是一个奏效的方法。

由于 AlexNet 在任务一中仅仅作为 baseline 对应方法，此处不再详述其具体的代码设计与结果分析。

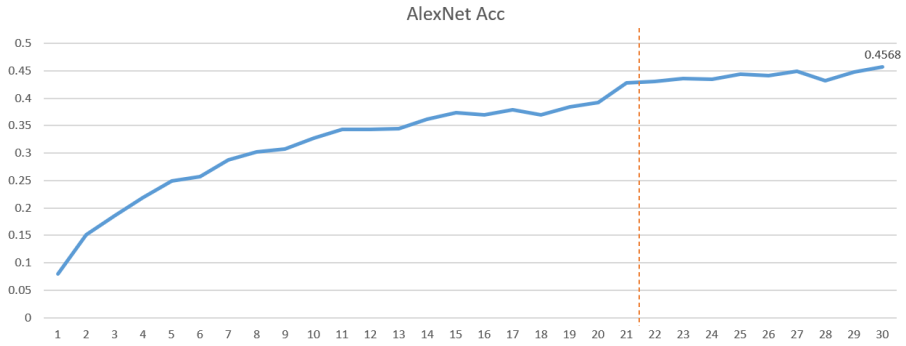

图 2: AlexNet 训练过程准确率变化

3.2 ResNet

ResNet[2] 是现代卷积神经网络中极为重要的结构，直连边的存在可以加速模型的训练，也更加能够避免大模型带来的梯度消失问题。因此为了完成典型的图像分类任务，对于 ResNet 网络的尝试也是十分必须的。

3.2.1 模型结构

首先，由于任务一尽管只是对 imagenet 子集的分类，但数据量相对与课堂练习中的任务较大，又考虑到 ResNet 本身的特性就是对于网络结构较深的模型，能够缓解梯度消失现象，发挥深度学习的作用，ResNet 网络实验前期就在 ResNet-18 的结构基础上稍加修改，并使用 SGDM 进行训练。然而结果并不理想，与作为 baseline 的 AlexNet 结果相差无几，过拟合问题非常严重，由此意识到了轻量级模型可能会有更好的效果。

通过整合课堂内容，并浏览相关论文，根据 [3] 中的 ResNet 网络结构，实验设计了新的 ResNet，共有 9 层卷积，参数量减小了许多，从表1中也可以看到这一点。网络的具体结构图如图3所示，其中的基本单元与经典的 ResNet 网络一致，但模型更加轻量，事实上在 [3] 中，这一网络结构也是以 ResNet-18 为基础搭建的。

Listing 1: **ResNet 网络中一个核心单元的代码**

```
class Residual1(nn.Module):
    def __init__(self,in_channels=64,rate=2,init_weights=False):
         super().__init__() # rate=输出通道数/输入通道数
        self.layer = nn.Sequential(
            nn.Conv2d(in_channels,rate*in_channels,kernel_size=3,padding=1,stride=2),
            nn.BatchNorm2d(rate*in_channels,affine=True),
            nn.ReLU(inplace=True), # 每个卷积操作后都有BatchNorm
            nn.Conv2d(rate*in_channels,rate*in_channels,kernel_size=3,padding=1,stride=1),
            nn.BatchNorm2d(rate*in_channels,affine=True)
        )
        self.layer1 = nn.Conv2d(in_channels,rate*in_channels,kernel_size=1,stride=2)
        if init_weights:
```

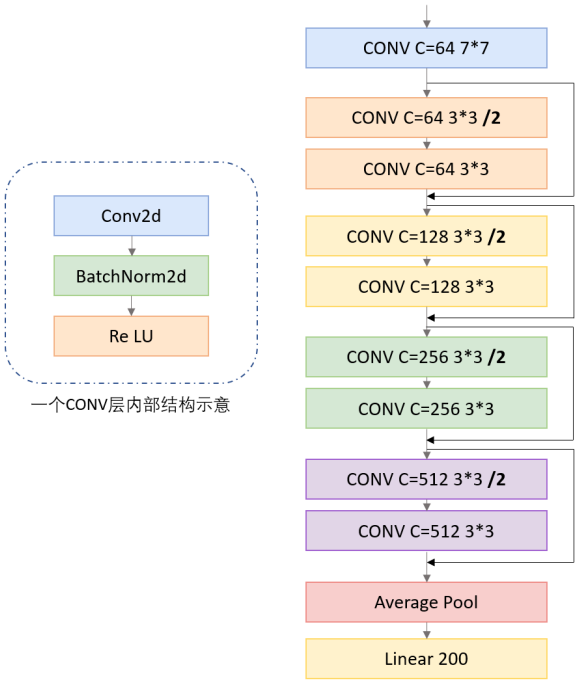

图 3: ResNet 网络结构设计

```
            self._initialize_weights()
    def forward(self,X):
        return F.relu(self.layer1(X)+self.layer(X))
    def _initialize_weights(self): # 使用kaiming_normal_初始化
        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)
```

3.2.2 数据增强

在对 AlexNet 网络结构进行尝试时，已经发现任务一的数据量仍然容易造成过拟合的问题，在训练集准确率达到 99% 左右时，验证集的准确率也无法超过 50%。因此保证模型的泛化性是本任务的关键。

首先，在模型设计部分的网络结构输入维度实际为`batch_size * 3 * 56 * 56`，并不是数据集中原图大小的`batch_size * 3 * 64 * 64`，这正是采用了数据增强以后的结果，可以在代码2中看到数据增强的具体策略。

对于训练集，读入数据为`Tensor`格式以后，通过定义的`trainCrop`方法进行数据增强。首先对`64 * 64`的图片进行随机裁剪，其大小变为`56 * 56`，再在依概率水平翻转、依概率垂直翻转、随机旋转、随机改变对比度中选择一个进行变换，这些变换的具体参数可以在代码2中看到。值得注意的是，这一数

据增强方式并不是直接在读入数据时得到一个变换后的数据集来扩充训练数据，而是在训练每一个 batch 时，对训练数据进行变换。这样的处理使训练流程中的数据量依然保持一致，模型训练时间不会骤然增长，但每一个epoch的训练数据由于随机处理的方式不同，可以保证一定的泛化性。当然，这种处理仍然可以保证一部分图片在随机裁剪操作后不再变换，与原图接近，因此也不会造成准确率的下降。

而对于验证集，由于网络结构中需要使输入图片的大小一致，因此需要使其图片大小变换，本实验采用了中心裁剪的方法。

由此，就实现了数据增强的操作，这将对模型泛化性有积极的影响，有利于验证集准确率的提升。

Listing 2: **ResNet 网络采用的数据增强代码**

```
trainList = [
        torchvision.transforms.RandomHorizontalFlip(p=0.6),# 随机水平翻转
        torchvision.transforms.RandomVerticalFlip(p=0.6), # 随机垂直翻转
        torchvision.transforms.RandomRotation(degrees=20), # 随机旋转
        torchvision.transforms.ColorJitter(brightness=0, contrast=[0.9,1.08], saturation=0, hue=0)
    ] # 随机改变对比度
valCope= torchvision.transforms.CenterCrop(56) # 对于验证集要进行中心裁剪
trainCope = torchvision.transforms.Compose([
    torchvision.transforms.RandomCrop(size=56), # 随即裁剪
    torchvision.transforms.RandomChoice(trainList), # 选择一个进行操作
    ])
def getdata_task1_aug(X,mode='train'):
    if mode == 'train':
        X = trainCope(X)
    elif mode == 'val':
        X = valCope(X)
    else:
        print('error!')
    return X
```

3.2.3 直连边相加处理

此外，在模型设计中，直连边对应的相加操作要求两个相加的 volumn 有相同的维度。而在1中也可以看到，一个核心的 ResNet 单元中两层卷积网络会将通道数变为两倍，column 的大小变为二分之一，因此直连边需要处理成相同维度以后才可以执行相加操作。在常用的 ResNet 网络中，有多种处理办法，本实验通过一个 1×1 卷积，设置步长为 2 来调整大小。可以参考1中的`self.layer1`来查看其中的细节。

3.2.4 训练细节

根据多次尝试，并同时结合 [3] 中的训练细节介绍部分，实验首先尝试了 RMProp 和 Adam 优化器，但二者的收敛速度过慢，且过拟合现象也较为严重。所以最终选用了 “SGDM+ 手动调整学

习率”的方式。

具体来说，优化器参数设置如下：

```
optimizer = torch.optim.SGD(model.parameters(),momentum=0.9,weight_decay=2e-4)
```

其中的学习率由调用训练函数时的参数设置来决定。`momentum`设置为 0.9 属于深度学习导论课堂的经验知识，`weight_decay`值参考 [3] 中对`weight_decay`设置的讨论，其中`2e-4`相比于其周围的其他值有更好的约束效果。

关于学习率的设置，总体策略为：当验证集的准确率出现基本不变或者下降的状态时，将当前的学习率变为原来的 0.1 再继续训练。初始的学习率设置为 0.1，训练近 20 轮以后开始手动调整学习率，此后的调整频率在 10 轮左右，具体可以参考图4，其中的虚线代表学习率的调整。

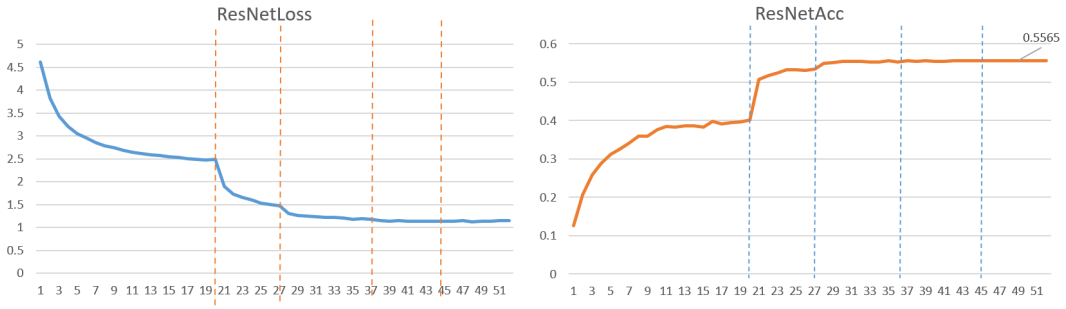

图 4: ResNet 网络过程的 loss 值和准确率变化

可以看出，在训练了 30 轮左右，loss 值和准确率都已经趋于稳定了，准确率达到了 55%，而再调整学习率，也不会有更加明显的提高。可以认为，对于该 ResNet 结构，以上的训练方法已经尽可能的在较少的训练轮数内达到了较高的准确率，发挥出了该模型的能力。

最后，ResNet 模型相比于 baseline，有较大的提升，最终最高在 50 轮达到了 55.65% 的预测准确率，表明了轻量级模型在本任务上的良好表现。

3.3 Inception-ResNet

在深度学习导论课堂的介绍中，Inception 块也是卷积神经网络中的重要结构，在 [3] 中，提出了基于 Inception-ResNet v2 model[4] 的 Modified Inception-ResNet 结构。模型将 ResNet 的关键直连边结构和 Inception 块组合起来应用，[4] 中该模型的分类效果是最好的，但本实验在试图进行复现时并没有得到优于其他模型的结果，并对原始结构进行了一部分改进。

3.3.1 模型结构

将直连边与 Inception 结合起来应用的效果可以类比为在 Inception 的多路结构下新增了一条不做任何变化的路，同时在多路合并方法上，非直连边部分将采用`concat`的方法来组合，直连边将与残差块部分直接相加，总体的结构可以从图5中看出。

每个部分的内部结构在图6中展示，其中所有的卷积层内部都是由 2D 卷积、BatchNorm2d 以及作为激活函数的 ReLU 三个部分组成。

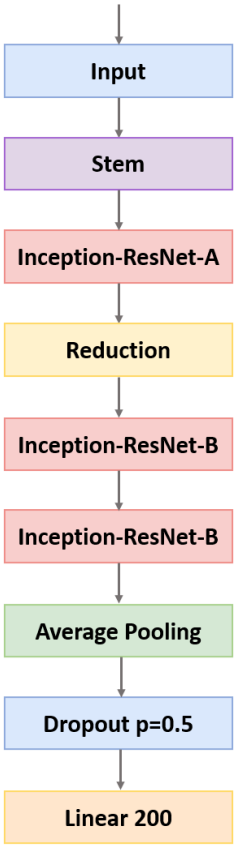

图 5: Inception-ResNet 总体结构

Stem 部分的模型结构比较复杂，与 Reduction 模块都会将输入的图像进行通道数增加、图像大小减小的操作，其中也存在着多路的操作，与 Inception 结构类似，由`concat`进行联合。而两个 Inception-ResNet 模块均含有直连边结构，是 ResNet 思想的关键，其中残差块是由 Inception 结构组合出的，残差块的结果再与原数据相加。在 Inception-ResNet 结构中，没有改变数据的维度，因而不需要特殊的处理就可以直接相加。

在整个模型中,为了减少参数量,使模型变得更加轻量级,缓解过拟合的问题,有多处用`1 * 7`和`7 * 1`的卷积来替代`7 * 7`卷积的操作，这样从一定程度上减少了卷积核的参数量。

此外，[4] 中提出了在 Inception 结构的深度模型中，随训练进行，模型的稳定性不能够保证，因而可能会导致部分神经元死亡，最后输出 0 的情况。为了缓解这种现象，论文中提出了在直连边的相加操作中为残差块添加权重的操作，将残差块的影响降低，使整个模型结构更趋向于 ResNet 结构而非 Inception 结构。[3] 表明在实验中发现这个权重值为 0.1 时模型预测的效果最好，因此在实验中也沿用了这一参数值设定。在 Inception-ResNet 模型总体对应的代码3中可以看到每一个 Inception 层都做了这一处理。

Listing 3: **Inception-ResNet 模型总体代码**

```
1 class Inception_ResNet(nn.Module):
2     def __init__(self,init_weights=False):
```

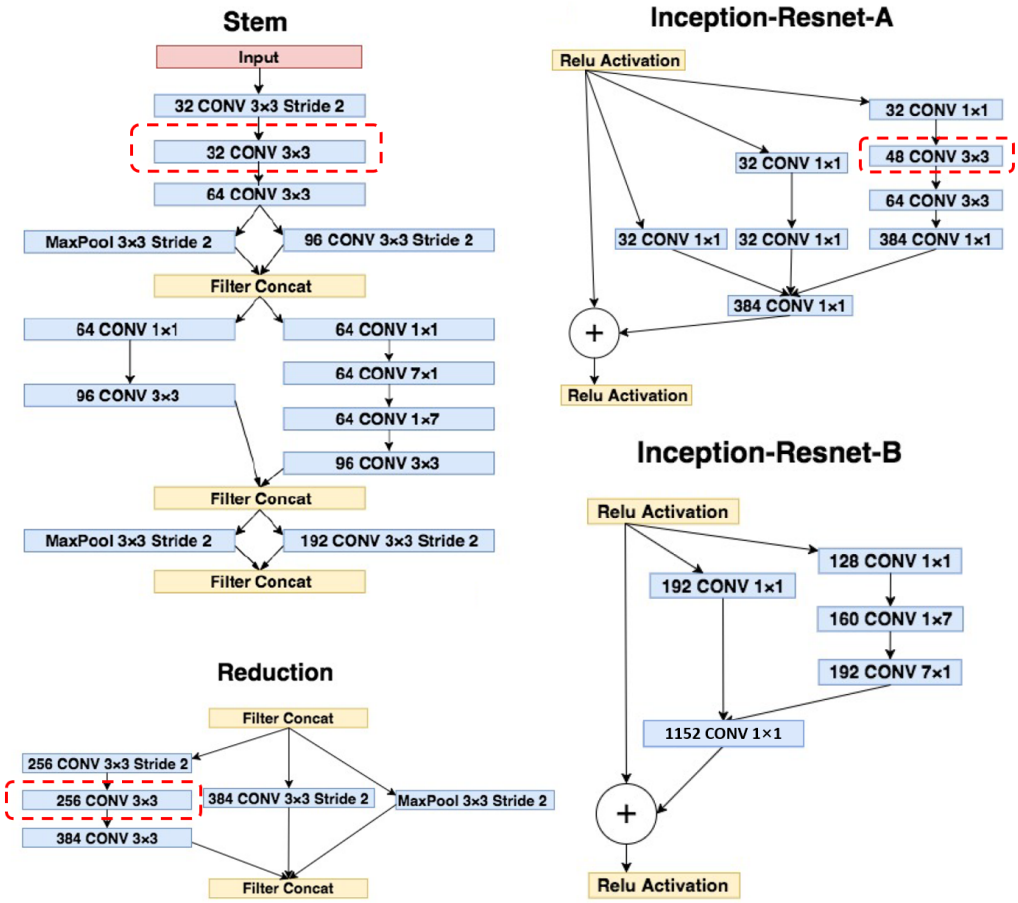

图 6: Inception-ResNet 具体结构

```
        super().__init__()
        self.layer = nn.Sequential( # 按照论文中的模型结构搭建，B的数量为A的两倍
            Stem(init_weights=init_weights),
            InceptionA(init_weights=init_weights,rate=0.1),# rate为残差块所乘的权重
            Reduction(init_weights=init_weights),
            InceptionB(init_weights=init_weights,rate=0.1),
            InceptionB(init_weights=init_weights,rate=0.1),
            nn.AvgPool2d(kernel_size=3),
            nn.Flatten(),
            nn.Dropout(p=0.5), # 使用Dropout避免过拟合
            nn.Linear(1152,200)
        )
        if init_weights:
            self._initialize_weights()
    def forward(self,X):
        return self.layer(X)
    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)
            if  isinstance(m,nn.Linear):
                nn.init.normal_(m.weight,0,0.01)
                nn.init.constant_(m.bias,0)
```

3.3.2 训练细节

本部分的模型训练同样考虑了通过数据增强来加强模型的泛化性，具体方法与 ResNet 模型训练中使用的数据增强方法相同，此处不再赘述。

而优化器的选择是多样的，在 [3] 中提出选用 RMProp 优化器，并每一轮将学习率变为原来学习率的 0.9 作为优化策略，学习率的变化是通过`torch.optim.lr_scheduler.StepLR`实现的，初始学习率为 0.05。具体来说为：

```
optimizer = RMSprop(model.parameters(),alpha=0.9,eps=1.0,lr=lr,weight_decay=2e-4)
scheduler = lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.9)
```

但实际进行训练以后，Inception-ResNet 的效果并不好。首先模型收敛的速度极慢，随着学习率的不断减小，在训练了 30 轮以后 Loss 值和预测准确率几乎没有变化，最终准确率稳定在 36.36% 附近（如图7），显然与作为 baseline 的 AlexNet 相比效果更差。因此这一训练方法是不合适的。尽管在 [3] 中这种方法能够在 25 轮内得到近 60% 的准确率，但本实验中结果却相去甚远。

即使增加两个 Inception-ResNet 模块的数量（即原始模型中取 $n > 1$），也只会使过拟合现象更加严重。

于是尝试从两个角度来改善模型：

- 改变模型的训练方案
- 修改模型结构

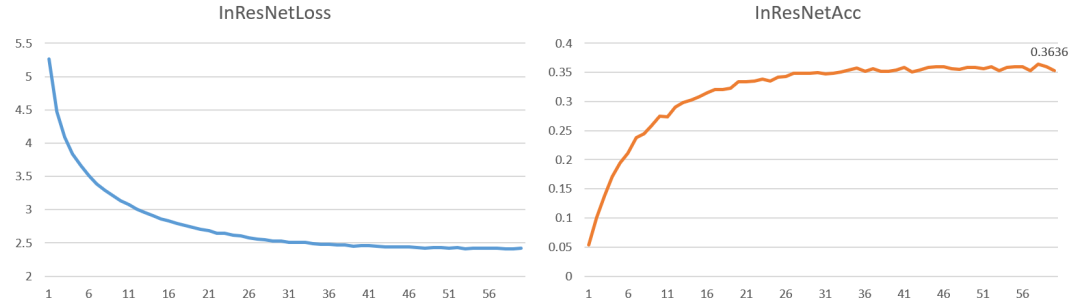

图 7: Inception-ResNet 训练过程 Loss 和 Acc 变化

首先改进训练方案，考虑到随着学习率的衰减导致训练后期的 Loss 和准确率基本无变化，尝试采用 ResNet 模型训练的方法。依然采用 RMProp 优化器（这么做的理由是在尝试了 SGD-M 和

Adam 均效果较差），但学习率在 Loss 值和准确率保持稳定后手动乘以 0.5 来降低，意图更快收敛。这种改进方法最终可以得到 48.91% 的正确率，具体可以参考图8。

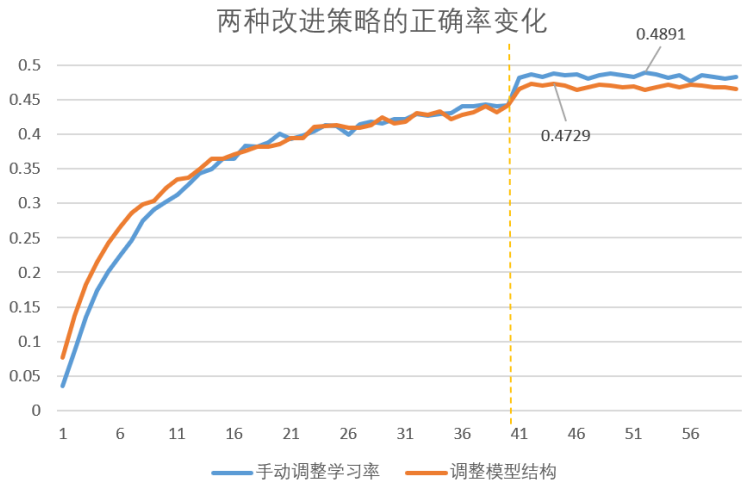

图 8: 两种改进策略的正确率变化（虚线为学习率调整）

其次尝试修改模型结构，推测原模型的结构偏复杂，可能因此更易造成过拟合。因此尝试将模型简化，在保持模型对称性设计的前提下，删减部分模块，即图6中虚线框中的模块。再通过前文中改进过的训练方案来训练，也可以得到近 47.29% 的结果（见图8）。尽管二者的结果相比于 baseline 没有较大的提升，但依然可以说明这种改进是有效的，且也加深了对模型设计的理解。

从改进的结果中也可以看出，Inception-ResNet 原模型并不一定过于复杂，而是需要选用合适的训练方法来发挥出模型的能力。尽管结果不够理想，但也可以加深对 Inception 和 ResNet 网络结构的理解。

3.4 DenseNet

DenseNet[6] 网络结构的基本思路与 ResNet 是相似的，都是将不同深度的特征结合起来，只是 ResNet 使用的是相加的方法，而 DenseNet 使用的是 Dense Connection 的方法，这也使得 DenseNet 网络的通道数增长较快。其原理结构示意图如图9所示。

而本实验的 DenseNet 更多地参考 [5] 中的模型二，其中的 DenseNet Connection 并非意味着每一模块都要与其后所有的连接节点做`concat`操作，而更倾向于将 ResNet 的结构继承，只是把相加操作改成了`concat`操作，也就是只与其后的第一个连接节点进行连接。

3.4.1 模型结构

尽管 DenseNet 在实际任务中使用的情况并不多见，但在本实验的任务一中，DenseNet 结构是预测效果最好的模型。[5] 中提出了两种基于 DenseNet 网络来解决该数据集上分类任务的方法，本实验基于第二种模型来构建网络。其具体的网络结构如图10所示。

其中的每个 CONV 模块都是由对应的卷积层、BatchNorm 以及作为激活函数的 ReLU 组成的，这一点在本实验尝试的所有模型中几乎都成立。实验中仍然尝试过去除 BatchNorm 或者使用其他

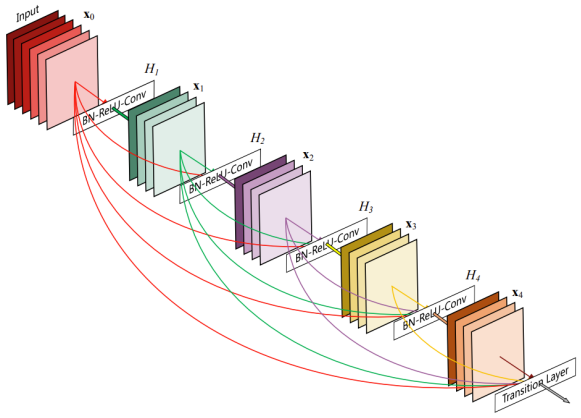

图 9: DenseNet 原理图

的激活函数，但几乎无法训练，可以看出 **BatchNorm 对模型效果的影响较大**，以及 **ReLU 的效果的确较好**。

其中的concat操作是在通道维度上做的，通道数的增加也使得模型的参数量会更大，在表1中也可以看到与效果接近的 ResNet 相比，DenseNet 网络的参数量会更多，且相差较大。但这一模型并没有因为增加的参数量而更易造成过拟合问题，说明不同模型的拟合能力有较大的不同。

可以在代码4中看到 DenseNet 的具体模型定义，包括其中的concat操作。

Listing 4: **DenseNet 模型定义代码**

```
class DenseNet(nn.Module):
    def __init__(self,init_weights=False):
         super().__init__()
        self.layer1 = MYBlock(init_weights=True)
        self.DownS = nn.MaxPool2d(kernel_size=2)
        self.layer2 = nn.Sequential(
            MYBlock(32,128,init_weights=True), # 输入和输出的通道数为参数
            MYBlock(128,128,init_weights=True),
            MYBlock(128,128,init_weights=True),
            MYBlock(128,128,init_weights=True)
        )
        self.layer3 = nn.Sequential(
            MYBlock(160,256,init_weights=True),
            MYBlock(256,256,init_weights=True),
            MYBlock(256,256,init_weights=True),
            MYBlock(256,256,init_weights=True)
        )
        self.layer4 = nn.Sequential(
            MYBlock(416,512,init_weights=True),
            MYBlock(512,512,init_weights=True),
            MYBlock(512,512,init_weights=True),
            MYBlock(512,512,init_weights=True)
```

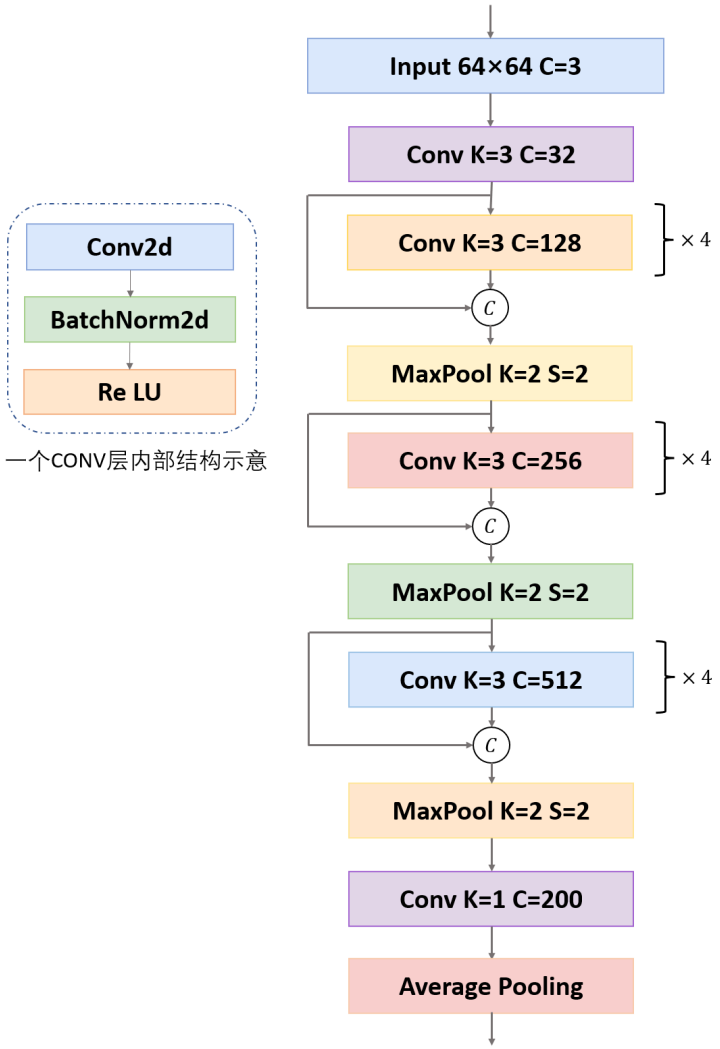

图 10: DenseNet 网络结构图

```
        )
        self.layer5 = nn.Sequential(
            nn.Conv2d(928,200,kernel_size=1),
            nn.AvgPool2d(kernel_size=8)
        )
        if init_weights:
            self._initialize_weights()
    def forward(self,X):
        X = self.layer1(X)
        X1 = self.layer2(X)
        X = torch.cat((X,X1),1) # 在通道维度上做concat操作
        X = self.DownS(X)
        X1 = self.layer3(X)
        X = torch.cat((X,X1),1)
```

```
        X = self.DownS(X)
        X1 = self.layer4(X)
        X = torch.cat((X,X1),1)
        X = self.DownS(X)
        X = self.layer5(X)
        X = X.squeeze(-1)
        X = X.squeeze(-1)
        return X
    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)
```

本实验的所有模型定义语句中都可以看出**参数的初始化**使用的是`kaiming_normal_`方法，这是多篇论文中使用的方法，效果较好。

3.4.2 数据增强

参考 [5] 中的数据增强方式，本实验的 DenseNet 模型训练采用了与前文中的两种模型不同的数据增强方法。输入图像仍然保持为`3 * 64 *64`。使用的主要操作时垂直翻转、水平翻转、高斯模糊、扩充后随机裁剪、随机仿射变化，这些操作会依照概率顺次执行，且输入的图片只有 50% 的概率会被执行这些操作。这些操作的部分具体效果可见图11。同样，这一数据增强会在每一个 batch 上作用，因此不同的 epoch 数据增强的结果也不同，从而保证了训练数据的总量不变。这一点与前文的数据增强是类似的。

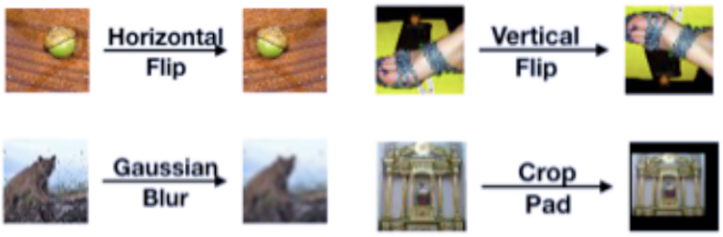

图 11: DenseNet 网络使用的数据增强方式

具体的数据增强代码可以从代码5中看到。

Listing 5: **DenseNet 网络数据增强方法**

```
dataList = [
    torchvision.transforms.RandomHorizontalFlip(p=0.5),# 随机水平翻转
    torchvision.transforms.RandomVerticalFlip(p=0.2), # 随机垂直翻转
    torchvision.transforms.GaussianBlur(kernel_size=3,sigma=(0.1,2.0)),# 高斯模糊
```

```
    torchvision.transforms.RandomCrop(64, padding=10, pad_if_needed=False, fill=0, padding_mode='
        constant'), # 扩充后随机裁剪
    torchvision.transforms.RandomAffine(degrees=45, translate=(0.2,0.2), shear=16) # 随机仿射变换
]
dataCope = torchvision.transforms.RandomApply(dataList, p=0.5)
def get_data_for_densenet(X):
    X = dataCope(X)
    return X
```

3.4.3 训练细节

在多个优化器中，尝试了 Adam 作为优化器，初始学习率设置为1e-4，取值较小，训练效果较好。而本实验中该模型训练的关键之处在于随着训练进行时的学习率的调整策略，采用了CyclicLR[7]，使得学习率在max_lr和min_lr之间来回波动变化，迭代周期由stepsize来确定。其原理见图12。

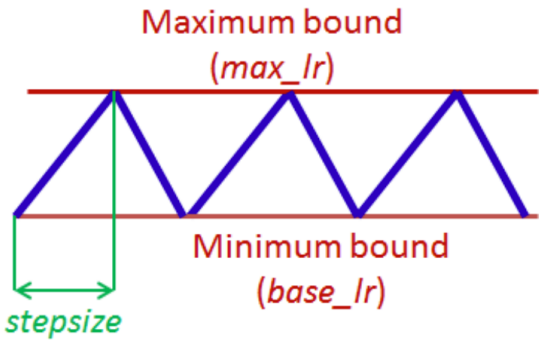

图 12: CyclicLR 原理图

这种学习率的调整策略有两个变种，本实验采用了triangular2，在每一个迭代周期后将学习率上下限差异变为原来的一半。

因此优化器的实际代码为，其中CyclicLR依然是通过lr_sheduler实现的：

```
optimizer = Adam(model.parameters(),lr=lr,eps=1e-8)
schedule = CyclicLR(optimizer,...,mode='triangular2',cycle_momentum=False)
```

参考 [5] 的策略，本实验起初也以 24 轮为界限，重新调整学习率，将学习率变为原学习率的 0.1，由于后续的学习率过低时 Loss 值和 Acc 都基本无变化，因此最后的策略是分别训练 24 轮和 48 轮，训练过程中的 Loss 值与预测准确率变化如图13。

可以看出该模型的收敛速度较快，在 27 轮就已经可以达到近 60% 的预测准确率。在 [5] 中也提到CyclicLR可以使前期的收敛速度较快，从模型实际测试效果中也验证了这一点。此外 DenseNet 网络结构也被验证为有较强的拟合能力。对于这一点，相关文章 [8] 中指出，可以认为 ResNet 的相加操作本身也可以看作是concat中的一种情况，所以 DenseNet 的拟合能力与 ResNet 是相近的，所以在本任务，这两种模型结构的拟合效果也是相近的。

总而言之，DenseNet 在任务一中是预测效果最好的模型，实验加强了对 DenseNet 的运用能力，也引入了特殊的学习率调节技巧，加深了对深度学习的理解。

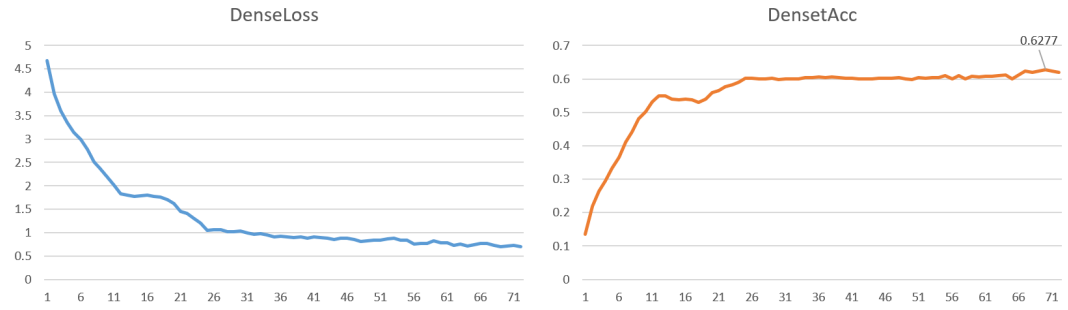

图 13: DenseNet 训练中的 Loss 值以及训练准确率变化

4 任务二实验记录

首先，将任务二的实验结果先在表格2中呈现出来，便于进行比较和结果分析。表格中的两个结果属于同一模型，但训练方法不同。

模型	训练轮数	参数量	正确率
AutoEncoder	25e+79e	4,310,059	12.83%
AutoEncoder	25e+33e	4,310,059	**24.50%**

表 2: 任务二实验结果

下面将从具体的实验流程上进行模型和训练方法的分析。

4.1 模型搭建

在任务一中，ResNet 和 DenseNet 模型的表现效果较好，因此在这个基础上，任务二的 AutoEncoder 网络结构将 ResNet 作为 Encoder 的主要组成部分。

Encoder 与 Decoder 在本实验中结构并非对称的关系，起初 Decoder 基本以反卷积层和带了直连边的卷积层组成。但考虑到对 AutoEncoder 做预训练的首要目的是使 Encoder 学到全体训练集的部分信息，Decoder 在本实验中主要起辅助的作用，故将 Decoder 的结构设置得更为简单，目的在于使 Encoder 可以在预训练阶段学习到更多的信息。

而 Encode 接上简单的 Classifier 结构后，就可以得到图像分类的结果。

最后构建的模型结构如图14所示。

Encoder 部分与图3所示的 ResNet 基本一致，出于训练数据量的骤减，将原 ResNet 结构通道数做一定的减少，缓解严重的过拟合现象。

6是 Decoder 部分的具体模型代码，其他模块的代码可以参考 ResNet 网络，或者在源码中查看。

由于数据的缺乏，训练集和验证集的数据量基本一致，可想而知模型的预测效果从绝对值上来说是不算好的。

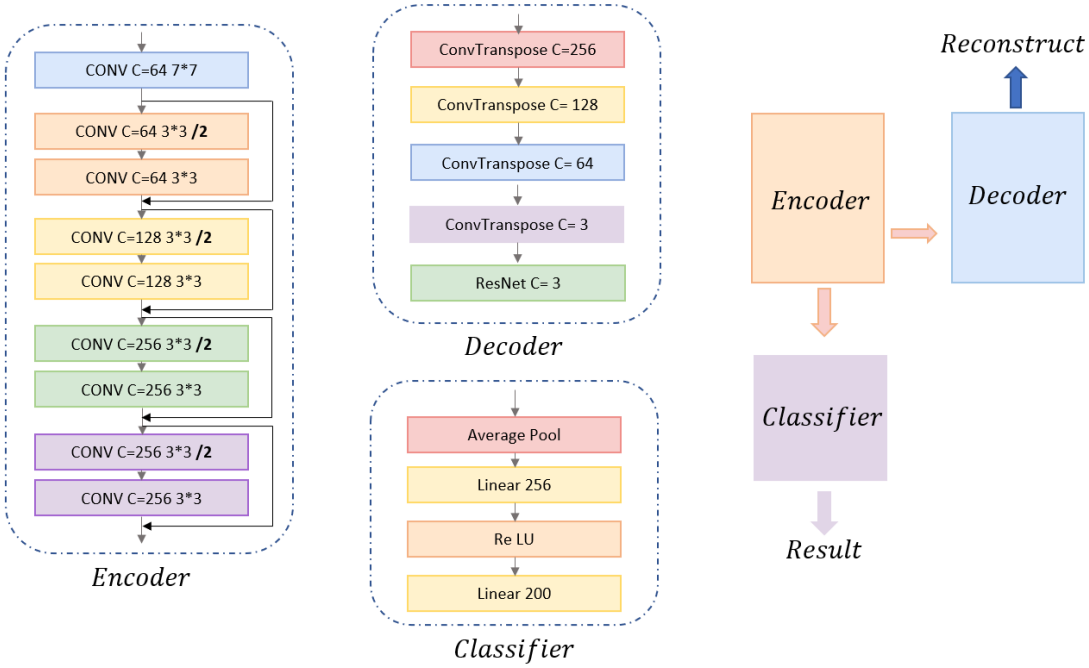

图 14: AutoEncoder 模型结构

Listing 6: **Decoder 模型代码**

```
class Decoder(nn.Module):
    def __init__(self,init_weights=False):
         super().__init__()
        self.layer = nn.Sequential(
            RevRes(256,256,init_weights=init_weights), # 主要由逆卷积层组成
            RevRes(256,128,init_weights=init_weights),
            RevRes(128,64,init_weights=init_weights),
            RevRes(64,3,init_weights=init_weights),
            Res(3,3,init_weights=init_weights)  # 最后是一个非常简单的直连边结构
        )
        if init_weights:
            self._initialize_weights()
    def forward(self,X):
        return self.layer(X)
    def _initialize_weights(self):
        for m in self.modules():
            if  isinstance(m,nn.ConvTranspose2d):
                nn.init.kaiming_normal_(m.weight,mode='fan_out',nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias,0)
```

4.2 训练细节

对于 AutoEncoder 模型，其训练流程分为两个步骤，一是预训练，利用图像重构误差来使 Encoder 学习到整个训练集图像的部分信息，但不用到图像的标签；二是分类任务训练，利用原训练集中 10% 的数据和交叉熵误差来进行分类任务的训练。

在模型训练过程中，采用的优化器均为 Adam 优化器，这是在使用其他优化器以后发现预测准确率基本没有增加时确定的。在学习率设置上，选取的值较小，因此收敛比较慢，初始值设置为`5e-4`，其后根据训练的情况手动调整学习率，一般取原学习率的 50%，若 Loss 值与预测准确率的变化不明显，也会采取暂时小幅增加学习率的操作。

首先，不做特殊的数据增强处理，按照 AutoEncoder 的训练流程进行优化，这时由于优化重构误差阶段的输入和目标输出没有较大的区别，这一阶段的训练对模型来说“难度”较低，因而 Loss 值也相对偏低，可以认为对于 Encoder 而言学习到的信息相较于有数据增强操作的训练方法要少。其两个训练过程中的重构误差值与分类准确率如图15，其中左边为重构误差阶段的 loss 值变化，右边为分类任务训练时的正确率变化。

在分类准确率达到峰值以后，由于过拟合较严重，正确率将会快速下降。

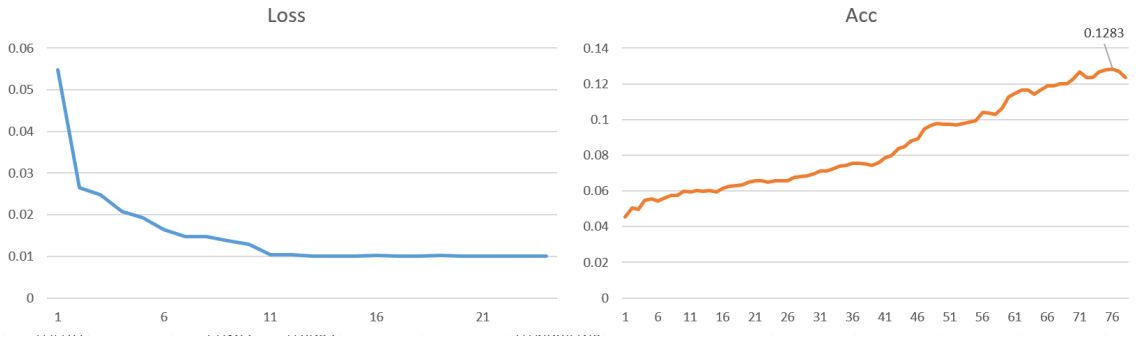

图 15: AutoEncoder 训练的两个阶段相关变化

在经过上述的初步尝试以后，发现解决过拟合是非常重要的方向，将考虑以下几个方面：

- 改善模型结构

  在初步的尝试中已经减少了原 ResNet 网络中的部分模块，降低通道数，同时 Decoder 结构极其简单，因此可以认为模型结构方面改善的空间较少

- L2 正则化

  体现在优化器中，新的训练方法中设置了更大的`weight_decay`值，具体如下：

  ```
  optimizer = Adam(model.parameters(),lr=lr,weight_decay=3e-4)
  ```

- 数据增强

  这是本实验中改进所用的主要方法，通过结合多种数据增强的操作，使预训练阶段能够学习到更多的信息，进而可以改善过拟合现象。

数据增强部分，首先使用了代码5中的dataCope方法，涉及依概率顺次进行水平翻转、垂直翻转、高斯模糊、扩充后随机裁剪以及仿射变换这几类方式，不同于之前的处理，它将数据集扩大了一倍，也就是说新构成的数据集中有 50% 是原图像，而剩下的是这些图像经过上述数据增强后的结果。

此外，受到 MAE[9] 的启发，将图像的部分 mask 以此来激发模型的学习潜力也是一个可能的方向，由于硬件设施与实验条件的原因，实验无法完全复现 MAE 工作中的处理办法，而只是通过使用albumentations库中的Cutout操作实现部分擦除，从原理上来模仿 MAE 的思路。具体的参数设置为：

```
T=A.Cutout(num_holes=12, max_h_size=16, max_w_size=16, fill_value=0,...)
```

在上述处理以后再次进行训练，预测正确率一定程度上提高了，可以参考图16。

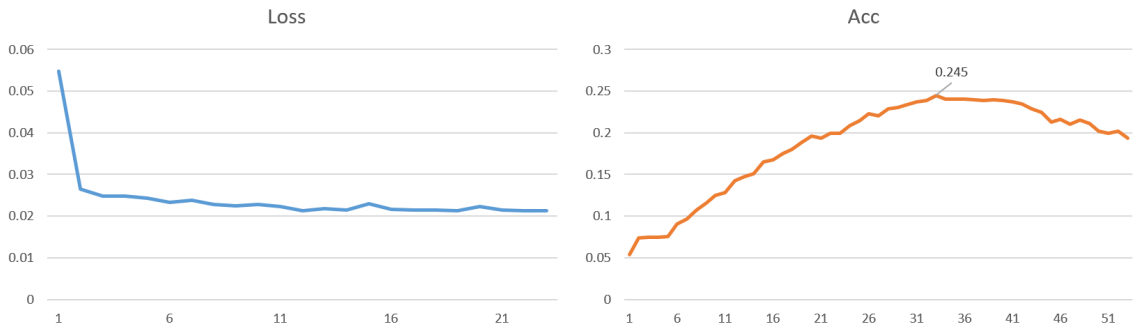

图 16: 改进训练方法后，AutoEncoder 训练的两个阶段相关变化

总而言之，从正确率的绝对值来讲，任务二的效果是比较差的，但考虑到训练数据极少的问题，这种 from scatch 的训练方式本身也比较难，因此可以认为任务二的结果在实验条件下是比较不错的了。

至此，两个任务的实验流程以及细节已经记录完毕。

5 预测误差分析

下面以整个实验流程中预测效果最好的模型为例对图像分类的预测结果进行分析。对每一类统计模型预测正确的数量，取结果最好的五类和最差的五类进行图像特点的分析，其中具体的数据见表3。

可以查看各类别的图像特征，如图17所示。可以看出，分类效果最好的模型 DenseNet 对于具有明显形状特征的类别图像分类效果较好，可以从五朔节花柱这一类别明显看出，比较特殊的形状，例如花柱、大角羊弯曲的角、带有多个按键的遥控、猫的眼睛与耳朵、以及排球相关的网与球等等，这些特征可能对分类有利。而对于分类效果较差的类别，尽管它们可能在颜色上有明显的特征，例如棕熊和列车等，由于从不同角度上看其形状可能有所不同，因而模型分类效果并不好。此外如漫画书这类在形状（正面看和侧面看不同）和颜色上均没有明显固定的特征的类别，其分类效果也不好，这是符合预期的。

因此可以认为，DenseNet 结构可能对图像的形状特征更加敏感，如果要讨论模型的进一步优化

类别编号	图片内容	分类正确数量
n06596364	漫画书	21
n02132136	棕熊、灰熊	23
n02403003	公牛	23
n02917067	高速列车、子弹	23
n04371430	沙滩裤、泳裤	24
n03733131	五朔节花柱	40
n02415577	大角羊	40
n04074963	遥控	39
n02123394	波斯猫	39
n04540053	排球	38

表 3: 任务二实验结果

方向，可能需要加强它对颜色的感知程度。

当然，事实上分类效果最好和最差的几类相差并不大，分类效果最好的结果也只能识别正确 40 张图片，而最差是 21 张，结合总体的分类正确率为 60% 左右也可以看出当前每一类的效果都是欠佳的。如果使用预训练模型结合微调，凭借模型本身对语义的感知，分类效果会更好。可以认为这类 from scratch 的训练方式由于有限的数据，是较难得到 80-90% 的结果的。

6 实验总结

总而言之，本次实验包括任务一和任务二这两个主要的任务，围绕着 tiny-imagenet-200 数据集的分类任务展开，总共 200 类图像。

任务一要求用给出的 100,000 张图训练出模型从而对 10,000 张验证集图像进行分类，由于训练数据较少，模型容易发生过拟合现象，用 from scratch 的方式难以得到绝对值上非常理想的结果。最终结合深度学习导论课堂知识以及相关论文工作，搭建了 AlexNet、ResNet、Inception-ResNet 和 DenseNet 四种模型，进行训练以后，用 DenseNet 实现了分类准确率达到 62.77% 的结果，是任务一的最好结果。此外 ResNet 以比前者更少的参数量达到了 55.65% 的正确率，也可以认为是一个比较好的结果。

任务二使用的训练数据将只有任务一训练集数据的 10%，也意味着训练集和验证集的数据量相同，使过拟合问题变得更加严重。为此使用了 AutoEncoder 的模型结构，通过首先用原训练集的所有数据的重构误差来预训练模型，再在给出的 10% 数据集上优化分类效果，结合特殊的数据增强方式，最后实现了 24.50% 的分类准确率。

Tiny-imagenet-200 数据集的分类挑战本身也是一个较难的挑战，其数据的匮乏导致很容易发生过拟合现象，from scratch 的训练方式很难得到一个比较理想的结果，但也更能够发挥出模型的潜力，是对深度学习导论课堂知识运用进行检测的合适数据集分类任务。在完成整个实验的过程中，对深度学习知识有更深的理解，也查阅了相关的论文资料，对深度学习领域比较新的研究成果有更

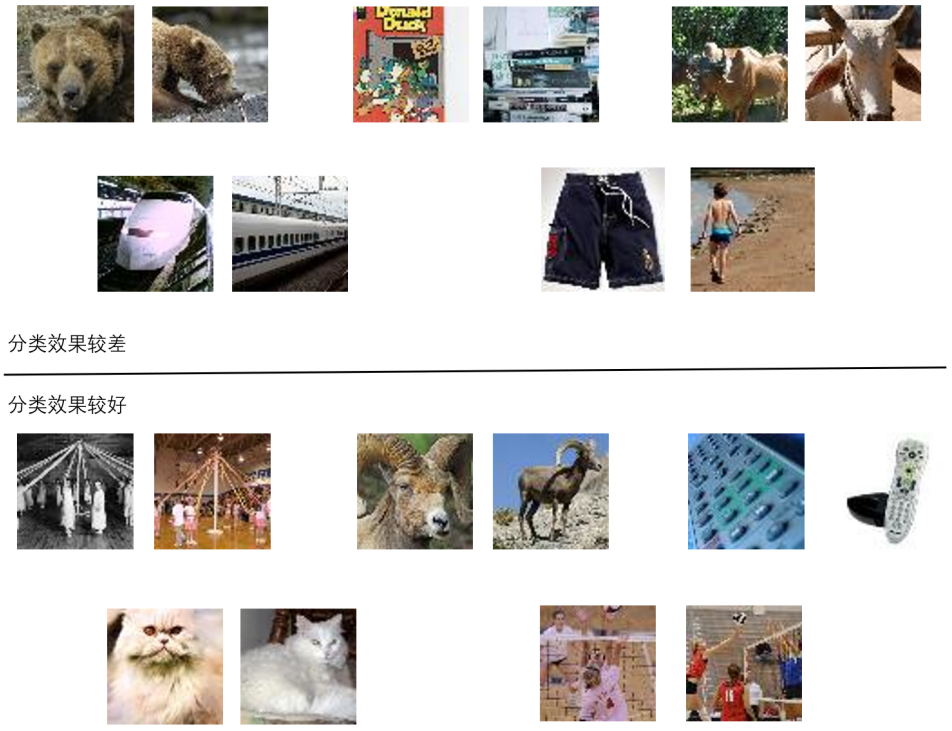

图 17: 分类效果较好与较差的类别图示

多的了解，相对完整地完成了任务。

参考文献

[1] ImageNet Classification with Deep Convolutional Neural Networks. Alex Krizhevsky , Ilya Sutskever , Geoffrey E. Hinton

[2] Deep Residual Learning for Image Recognition[Submitted on 10 Dec 2015].Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

[3] Tiny ImageNet Challenge. Jiayu Wu, Qixiang Zhang, Guoxi X

[4] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inceptionv4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261, 2016.*

[5] DenseNet Models for Tiny ImageNet Classification. Zoheb Abai, Nishad Rajmalwar

[6] G. Huang, Z. Liu, L. v. d. Maaten, K. Q. Weinberger, Densely Connected Convolutional Networks, *arXiv:1608.06993v5, 2018*

[7] Cyclical Learning Rates for Training Neural Networks. Leslie N. Smith U.S. Naval Research Laboratory, Code 5514 4555 Overlook Ave., SW., Washington, D.C. 20375 *arXiv:1506.01186v6 [cs.CV] 4 Apr 2017*

[8] Deepside: A general deep framework for sailent object detection.

[9] Masked Autoencoders Are Scalable Vision Learners. Kaiming He∗,† Xinlei Chen∗ Saining Xie Yanghao Li Piotr Dollar Ross Girshickeepside: A general deep framework for sailent object detection.*arXiv:2111.06377v3 [cs.CV] 19 Dec 2021*