MedMNIST 十项全能

项目报告

周轶凡 518030910291
王泽坤 518030910284

一、 项目背景

1.1 医学图像介绍

医学扫描越来越常见。例如，2005 年在美国大约有 57 百万个体接受了 CT 检查。到 2012 年数量超过 85 百万。大量数据中展现出来的特异性情况，非常具有统计学意义。不像自然图像，医学影像具有很强的上下文信息，例如有限数量的解剖目标，约束和结构化的背景，不同解剖结构之间的关系，强先验姿态参数信息等。

医学影像的识别（recognition）、分割（segmentation）和解析（parsing）是医学影像分析的核心任务。医学影像识别是指识别医学图像中的目标。理论上，目标的识别并不需要对目标进行检测或定位；但是实际上，通常会结合检测和定位去辅助完成目标识别。一旦完成识别，或检测，即得到了目标的最小外包矩形框（bounding box），就可以通过分割的任务寻找目标物体的精确边界。

医学影像的一大挑战是临床应用对精度、稳定性和速度的严格要求。读片和诊断通常不允许出错。尽管要求很高的精度和稳定性，速度仍不能慢，一个快速的工作流能够确保医院的高吞吐量。放射科和外科医生不会愿意话几个小时甚至几分钟去等待一个分析结果。

1.2 图像分类

分类是医学图像在计算机辅助诊断和模式识别领域的一个研究热点，精确地对人体解剖结构和病变区域进行分类能够最大程度辅助医生更加精确、更快速地诊断病情。

二、 算法原理

2.1 Resnet 分类算法

这里我们采用到了深度残差网络结构学习（Deep Residual learning），传统的神经网络存在着网络越深则会出现梯度消失或者爆炸等网络退化问题，而将传统深层网络的后面若干层学习成恒等映射$h(x) = x$，那么模型就退化成浅层网络。但是直接去学习这个恒等映射是很困难的，那么就换一种方式，把网络设计成：

$$H(x) = F(x) + x \Rightarrow F(x) = H(x) - x$$

只要$F(x) = 0$就构成了一个恒等映射$H(x) = x$，这里$F(x)$为残差。对于一个堆积层结构（几层堆积而成）现在我们希望其可以学习到残差$F(x)$，这样其实原始的学习特征是$H(x)$。之所以这样是因为残差学习相比原始特征直接学习更容易。当残差为$F(x) = 0$时，此时堆积层仅仅做了恒等映射，至少网络性能不会下降，实际上残差不会为 0，这也会使得堆积层在输入特征基础上学习到新的特征，从而拥有更好的性能。以下是残差学习单元（残差块）：

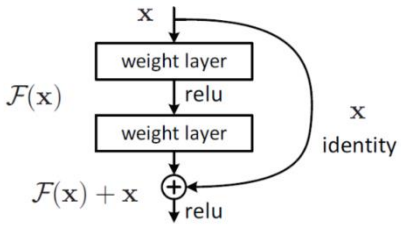

图 1.残差学习单元

为什么残差学习相对更容易，从直观上看残差学习需要学习的内容少，因为残差一般会比较小，学习难度小点。不过我们可以从数学的角度来分析这个问题，首先残差单元可以表示为：

$$y_l = h(x_l) + F(x_l, W_l)$$
$$x_{l+1} = f(y_l) = h(x_l) + F(x_l, W_l)$$

其中，x_L和x_{L+1}分别表示第L个残差单元的输入和输出，注意每个残差单元一般包含多层结构。F是残差函数，表示学习到的残差，而$h(x_L) = x_L$表示恒等映射，f是 ReLu 激活函数。基于上式，通过递归，我们求得从浅层l到深层L的学习特征表示为：

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$$

我们可以知道任何单元l和L之间都具有残差特征。

从前向传播角度，我们可以知道，对于传统的 CNN，直接堆叠的网络相当于一层层地做**仿射变换-非线性变换**，而仿射变换这一步主要是矩阵乘法。所以总体来说直接堆叠的网络相当于是乘法性质的计算。而在 ResNet 中，相对于直接堆叠的网络，因为 shortcut 的出现，计算的性质从乘法变成了加法。计算变的更加稳定，可训练的网络的层数也大大增加。

三、 代码实现

3.1 Resnet18、Resnet50 搭建

在这里我们主要完成了 Resnet18 和 Resnet50 的搭建。两者除了网络层数的不同外，在具体残差单元上也有不同的体现。两者整体框架如下：

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 2.Resnet 网络框架

3.1.1 Resnet18 细则

Resnet18 主要有两类基础块,第一种结构是在通道数不变的情况下，进行的残差结构运算，第二种的跳跃连接结构，通道数发生了改变，具体如下：

图 3 Resnet18 的两种 BasicBlocks

因此在代码实现中我们将两块分开写，第一类基础块可以看到其输入输出的尺寸大小没有发生变化，卷积核大小都为 3*3、步长都为 1、填充大小都为 1。而第二类基础块出现虚线处的降维，经过卷积层的输入输出尺寸会发生变化，且卷积核大小也会发生变化，这里需要单独列出这一虚线的卷积层，并将其和顺序往下的两个卷积层做相加，这里最后一层的输入输出尺寸也同样没有变化。

基础块搭建好后，我们就按照框架中的网络层来进行搭建，一共是：一层 7*7 核的卷积层+一个一类基础块（2 层）+一个一类基础块（2 层）+一个二类基础块（2 层）+一个一类基础块（2 层）+一个二类基础块（2 层）+一个一类基础块（2 层）+一个二类基础块（2 层）+一个一类基础块（2 层）+全连接层，共计 18 层，并且需要注意的是，每经过一次二类基础块，输出尺寸都会扩大两倍。如此我们便搭建好了自己的 Resnet18 网络。（代码过于占用篇幅，因此不在此贴出，网络代码见\medmnist\my_model.py）

3.1.2 Resnet50 细则

与 Resnet18 类似的，Resnet50 也有两类基础块（BottleNeck）：

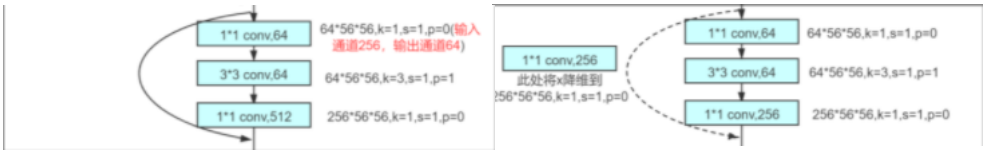

图 4 Resnet 的两种 BottleNecks

注意到两类 BottleNecks 的中间层都发生了卷积核大小和填充大小的变化，且最后一层输出变为输入尺寸的 4 倍、第一层输出变为输入尺寸的 1/4。相似的，第二类 BottleNeck 中虚线层进行降维，但输入输出尺寸保持一致。

BottleNeck 搭建好后，我们就按照框架中的网络层来进行搭建，一共是：一层 7*7 核的卷积层+一个二类基础块（3 层）+两个一类基础块（6 层）+一个二类基础块（3 层）+三个一类基础块（9 层）+一个二类基础块（3 层）+五个一类基础块（15 层）+一个二类基础块（3 层）+两个一类基础块（6 层）+全连接层，共计 50 层，如此我们便搭建好了自己的 Resnet50 网络。

3.2 train.py 修改及 loss 函数介绍

在具体的训练和测试函数中，我们沿用了源码中的大体框架，新添加了

训练网络的选择参数 argument：--resnet。在将数据集的三个部分：train_dataset,val_dataset,train_dataset 下载下来后，根据不同数据集的分类类型，选择适合的损失函数，并选择 optimizer 为最为常用的 SGD 优化，进行模型的训练，保存好每一轮训练后的模型，并作出相应的准确率 AUC 曲线，根据训练中效果最好的一轮模型，用来进行测试数据集的测试，得到最后测试的准确率 AUC 和 ACC，来反映训练的成果。

在 loss 函数的选择上，我们针对二分类的数据集，选择到 torch 中的 BCEWithLogtisLoss 函数，这个函数是 BCELoss 函数的强化版，BCE 函数可以认为是 CrossEntropyLoss 函数的特例。其分类限定为二分类，y 必须是{0,1}。而 BCEWithLogitsLoss 函数把 Sigmoid 层集成到了 BCELoss 类中。该版比用一个简单的 Sigmoid 层和 BCELoss 在数值上更稳定，因为把这两个操作合并为一个层之后,可以利用 log-sum-exp 的技巧来实现数值稳定。而对于需要做多分类和有序回归(ordinal regression)的数据集，我们采取最常用的交叉熵函数 CrossEntropyLoss：

$$L = -[y log\ \hat{y} + (1-y)\log(1-\hat{y})]$$

神经网络总是非凸的，这使得交叉熵对错误分类的惩罚力度更大，且在计算梯度时会约去复杂项使得计算更简便。故而选择交叉熵函数。

四、 结果分析

4.1 模型训练结果

以下分别是 Resnet18 和 Resnet50 对于 MedMNIST 十个数据集的训练模型情况：这里受限于硬件资源，我们并没有对每个数据集进行 100 轮次的训练，对于较小的数据集诸如 breast 和 retinamnist 等，我们训练了 50 轮并发现基本上已经收敛，而对于大一些的数据集我们则选择了 25 轮、10 轮等轮次进行训练，得到的结果用于分析已经足够。

结果记录如下表：

首先是使用 Resnet18 训练后得到的结果，其中序号指代：

（1. Pathmnist 2. Chestmnist 3. Octmnist 4. Pneumonia 5. Breastmnist 6. Dermamnist 7. Retinamnist 8. organmnist_coronal 9. organmnist_sagittal 10. organmnist_axial）

	1	2	3	4	5	6	7	8	9	10
trainAUC	0.998	0.738	0.986	1.000	1.000	1.000	1.000	1.000	0.999	1.000
trainACC	0.902	0.949	0.936	1.000	1.000	0.999	0.989	0.999	0.985	0.999
valAUC	0.990	0.693	0.965	0.994	0.958	0.897	0.749	0.999	0.988	0.998
valACC	0.852	0.949	0.895	0.968	0.885	0.736	0.508	0.957	0.866	0.957
testAUC	0.956	0.684	0.922	0.939	0.855	0.894	0.700	0.986	0.954	0.988
testACC	0.756	0.947	0.708	0.865	0.827	0.738	0.455	0.876	0.712	0.875

然后是 Resnet50 训练后得到的结果：

	1	2	3	4	5	6	7	8	9	10
trainAUC	0.998	0.720	0.986	1.000	1.000	1.000	0.956	1.000	0.996	1.000
trainACC	0.920	0.924	0.931	0.999	1.000	0.998	0.784	0.993	0.937	0.999
valAUC	0.991	0.665	0.964	0.994	0.898	0.885	0.785	0.998	0.987	0.999
valACC	0.871	0.920	0.891	0.962	0.846	0.750	0.567	0.946	0.832	0.959
testAUC	0.956	0.656	0.920	0.939	0.871	0.889	0.686	0.982	0.955	0.986
testACC	0.756	0.932	0.688	0.889	0.833	0.724	0.468	0.848	0.682	0.865

4.2 结果比较分析

通过结果图我们可以看到，针对同一个数据集和同一组参数（epochs，batchsize 等），使用 Resnet18 和 Resnet50 网络进行训练得到的 AUC 和 ACC 相差不大，说明在前者的网络层数已经足够收敛了，且随着网络层数的加深，退化情况几乎可以说没有，也可以说明 Resnet 本身的优越性。在大部分情况下，Resnet18 得到 AUC 和 ACC 也都略微优于 Resnet50 的结果。

而对于不同模型存在的差异，不管是何种 Resnet，都遵循输入部分、输出部分和中间卷积部分(若干 stages)的结构特点，网络之间的不同主要在于中间卷积部分的 block 参数和个数存在差异，这对于训练结果的影响来自学习的参数，更大一部分差异来自是数据集本身，十个数据集的质量不能同一而论，且其分类类别也不尽相同，多分类和二分类自然会导致差异。而对于改进，除了使用更大量的数据集，训练更多的轮次，一定程度的图像处理方法也可能起到一定效果，但特定数据集的处理取决于具体数据集的内容，如果不能完全理解，那么处理的效果就不能得到很好的保证。