《医学课程项目》小作业实验报告

组员：薛峥嵘 519030910349
王资 519030910345

一、作业目标

1.熟悉医学图片的构成
2.熟悉神经网络的搭建
3.掌握对医学图片的分类方法

二、作业要求

1.按照给出的代码示例，搭建 ResNet-18 与 ResNet-50 网络
2.阐述 ResNet-18 与 ResNet-50 在分类结果上的差异，并分析差异的原因
3.分析模型输入分辨率对结果产生影响的原因
4.对如何提高分类效果，提出自己的改进方案

三、数据集简介

MedMNIST 是 10 类开源医学数据集。该数据集已经做好了预处理，包括:
1.已经转换为 28*28 的轻量级，方便处理
2.数据规模从 100 到 100000，分类任务包括二分类、多分类、回归。
3.已经做了一些医学层面的预处理，使得参与该项目的人不需要有任何医学相关的背景知识。
MedMNIST 数据集总共分成十种: PathMNIST、ChestMNIST、DermaMNIST、OCTMNIST、PneumoniaMNIST、RetinaMNIST、BreastMNIST、OrganMNIST{Axial, Coronal, Sagittal}各种数据集的相关信息如下表:

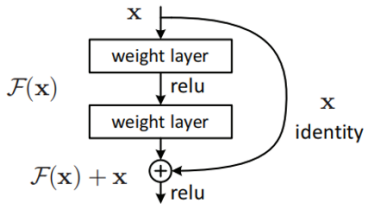
Name	Data Modality	Tasks (# Classes/Labels)	# Training	# Validation	# Test
PathMNIST	Pathology	Multi-Class (9)	89,996	10,004	7,180
ChestMNIST	Chest X-ray	Multi-Label (14) Binary-Class (2)	78,468	11,219	22,433
DermaMNIST	Dermatoscope	Multi-Class (7)	7,007	1,003	2,005
OCTMNIST	OCT	Multi-Class (4)	97,477	10,832	1,000
PneumoniaMNIST	Chest X-ray	Binary-Class (2)	4,708	524	624
RetinaMNIST	Fundus Camera	Ordinal Regression (5)	1,080	120	400
BreastMNIST	Breast Ultrasound	Binary-Class (2)	546	78	156
OrganMNIST_Axial	Abdominal CT	Multi-Class (11)	34,581	6,491	17,778
OragnMNIST_Coronal	Abdominal CT	Multi-Class (11)	13,000	2,392	8,268
OrganMNIST_Sagittal	Abdominal CT	Multi-Class (11)	13,940	2,452	8,829

四、使用模型介绍

1.Resnet 架构

1.1 简介

Resnet 的主要贡献是发现了神经网络的退化现象，并针对退化现象提出了 shortcut connection，极大的消除了深度过大的神经网络的训练困难问题。神经网络得深度首次突破 100 层。



如图所示即为残差单元的模型，可以表示为：

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

其中，x_l和x_{l+1}分别表示第 l 个残差块的输入和输出，事实上每个残差单元一般包含多层结构。F 是残差函数，h 表示恒等映射，f 是激活函数。

则从浅层 l 到深层 L 的学习特征为：

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

利用链式法则，可以求导得到反向过程的梯度。

$$\frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \frac{\partial x_L}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \left(1 + \frac{\partial}{\partial x_L} \sum_{i=l}^{L-1} F(x_i, W_i)\right)$$

不同 resnet 如下表：

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	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

在本任务中，主要使用 resnet18 和 resnet50。

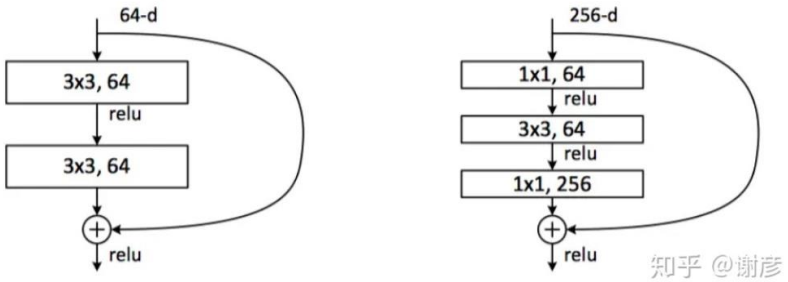
1.2 搭建过程

实现 resnet18 和 resnet50，首先需要搭建两个 block，分别是 BasicBlock 和 BottleNeck

```python
class BasicBlock(nn.Module):
    expansion = 1
    def __init__(self, in_channels, out_channels, stride = 1):
        super(BasicBlock, self).__init__()
        self.straight = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size = 3, stride = stride, padding = 1, bias = False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace = True),
            nn.Conv2d(out_channels, out_channels, kernel_size = 3, stride = 1, padding = 1, bias = False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()

        if (stride != 1) or (in_channels != out_channels):
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size = 1, stride = stride, bias = False),
                nn.BatchNorm2d(out_channels)
            )
        self.ReLU = nn.ReLU(inplace = True)

    def forward(self, x):
        output = self.straight(x) + self.shortcut(x)
        output = self.ReLU(output)

        return output
```

```python
class BottleNeck(nn.Module):
    expansion = 4
    def __init__(self, in_channels, out_channels, stride = 1):
        super(BottleNeck, self).__init__()
        self.straight = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size = 1, bias = False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace = True),
            nn.Conv2d(out_channels, out_channels, stride = stride, kernel_size = 3, padding = 1, bias = False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace = True),
            nn.Conv2d(out_channels, out_channels * BottleNeck.expansion, kernel_size = 1, bias = False),
            nn.BatchNorm2d(out_channels * BottleNeck.expansion)
        )

        self.shortcut = nn.Sequential()

        if (stride != 1) or (in_channels != out_channels * BottleNeck.expansion):
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * BottleNeck.expansion, stride = stride, kernel_size = 1, bias = False),
                nn.BatchNorm2d(out_channels * BottleNeck.expansion)
            )

        self.ReLU = nn.ReLU(inplace = True)
    def forward(self, x):
        output = self.straight(x) + self.shortcut(x)
        output = self.ReLU(output)

        return output
```

整体思路很类似，都是先构造上述的 shortcut connection 模块，以便重复利用此模块。但两者的区别在于：



左边的 BasicBlock 包含两个 3*3 的卷积层，右图的 BottleNeck 包括了三个卷积层，第一个 1*1 的卷积层用于降维，第二个 3*3 层用于处理，第三个 1*1 卷积层用于升维。这样减少了计算量。BasicBlock 主要用于 resnet18 和 resnet34，剩下更深的 resnet 都会利用 BottleNeck 结构。

然后按表中所示层顺序搭建 resnet

```
def __init__(self, block, num_blocks, in_channels, num_classes):
    super(ResNet, self).__init__()
    self.in_planes = 64

    self.conv1 = nn.Sequential(
        nn.Conv2d(in_channels, 64, kernel_size = 3,stride = 1, padding = 1, bias = False),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace = True)
    )
    self.conv2_x = self.make_layers(block, 64, num_blocks[0], 1)
    self.conv3_x = self.make_layers(block, 128, num_blocks[1], 2)
    self.conv4_x = self.make_layers(block, 256, num_blocks[2], 2)
    self.conv5_x = self.make_layers(block, 512, num_blocks[3], 2)
    self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
    self.fc = nn.Linear(512 * block.expansion, num_classes)

def make_layers(self, block, out_channels, num_block, stride):
    strides = [stride] + [1] * (num_block - 1)
    layers = []
    for stride in strides:
        layers.append(block(self.in_planes, out_channels, stride))
        self.in_planes  = out_channels * block.expansion

    return nn.Sequential(*layers)
```

这里按原论文的结构是先做了 7*7 的卷积，再做了 stride 为 2 的 maxpool，但由于 28*28 的图像本身就比较小，如果按原结构做的话就在第一层直接缩小了四倍，可能损失了不少信息，得不偿失，所以这里只做了一次加了 padding 之后的卷积。图像大小不变，但通过卷积操作提取出了一部分特征。接着就是调用 make_layers 函数，根据 BasicBlock 和 BottleNeck 类制作出整个 resnet。

前传操作就是把定义的各层都调用一遍，这里不需要赘述。

按照如下定义即可调用不同的 resnet。

```
def ResNet18(in_channels, num_classes):
    return ResNet(BasicBlock, [2, 2, 2, 2], in_channels = in_channels, num_classes = num_classes)

def ResNet50(in_channels, num_classes):
    return ResNet(BottleNeck, [3, 4, 6, 3], in_channels = in_channels, num_classes = num_classes)
```

2.loss 函数

Baseline 中主要用了两种 loss 函数，BCEwithlogitsLoss 和 CrossEntropyLoss.

对于 BECwithlogitsLoss，输入参数是(predict_result，ground_truth)，其作用是将 predict_result 先通过 sigmoid 映射到 0~1 之间的值，再通过公式

$$l_n = -w_n \left[y_n \cdot log x_n + (1 - y_n) \cdot log \left(1 - x_n \right) \right]$$

逐元素计算 Loss，最后求和。

对于 CrossEntropyLoss，在 pytorch 的设定中，该函数的主要作用是将 logsoftmax-NLLLoss 合并在一起得到的结果。将 softmax 之后的结果取 Log，减少了计算量，同时保障了函数的单调性。

其中 logsoftmax 的定义如下：

$$f_i(x) = log \frac{e^{(x_i)}}{a}, a = \sum^{j} e^{(x_j)}$$

NLLLoss 的定义如下：

$$loss(x, class) = -x[class]$$

Pytorch 中 CrossEntropyLoss 的定义如下：

$$loss(x, class) = -\log\frac{exp(x[class])}{\sum_j exp(x[j]))} = -x[class] + log(\sum_j exp(x[j]))$$

输入为(input, target)，其中 input 为模型的输出，包含每个类的得分。Shape 为 batch*n 类，target 的大小为 n，包含类别的索引（不是 one-hot 编码的形式）。

3.不同模型产生差异的原因

Resnet18 和 Resnet50 对比，Resnet18 的网络层数更少，那么其表征能力注定比不上 50 的表征能力。有了 shortcut connection 之后，在 resnet 中几乎更深的网络表征能力、泛化能力更强。所以观察到训练相同 epoch 数量的时候，resnet50 的效果往往更好一些。但也正是由于 Resnet50 的网络更加庞大，所以训练时间大约是 Resnet18 的两到三倍，这也是大型网络的一个劣势。

4.改进方案

4.1 epoch 数

我们发现 epoch==100 的时候，最优的 auc 往往在比较低的 epoch 数时找到，说明出现了过拟合的现象。所以我们减少了 epoch 数，Resnet18 大约训练 50 轮，Resnet50 大约训练 10~20 轮即可到达很好的效果，不会过拟合。

4.2 数据增强（对比度，随机旋转）

观察本数据集，发现有的数据集对比度很低，基本颜色相近。所以我们使用了 torchvision.transforms.ColorJitter 进行调整，主要改善了数据集中的亮暗、对比度。

另外，有些数据集的样本比较少，如 BreastMNIST，所以需要做一些随机旋转的操作来间接增大数据量。但这里我们做的是随机水平旋转，因为医学图像本身的特性不能被改变。如果随机竖直旋转，训练出来的结果虽然能提升一定的泛化能力，但事实上这样的数据是无效的，所以做法不可取。这里调用的是 torchvision.transforms.RandomVerticalFlip 函数。

总之，数据增强操作能够提高整个模型的泛化能力。

4.3 优化器

SGD 的收敛速度较慢，我们换成了 Adam 优化器，优化速度明显提升，这也是能够减少 epoch 数目的原因之一。

五、实验结果（与论文原文的结果进行对照）

由于训练步骤重复且冗长，在征得助教同意后，我们选取了不同数据规模、不同任务的 7 种数据进行测试，那些具有相同任务、相近数据规模的任务没有做，但并不难看出我们的模型和训练步骤的正确性。

下表为论文中的指标：

Methods	PathMNIST		ChestMNIST		DermaMNIST		OCTMNIST		PneumoniaMNIST	
	AUC	ACC	AUC	ACC	AUC	ACC	AUC	ACC	AUC	ACC
ResNet-18 (28) [6]	0.972	0.844	0.706	0.947	0.899	0.721	0.951	**0.758**	0.957	0.843
ResNet-18 (224) [6]	0.978	0.860	0.713	**0.948**	0.896	0.727	0.960	0.752	0.970	0.861
ResNet-50 (28) [6]	0.979	**0.864**	0.692	0.947	0.886	0.710	0.939	0.745	0.949	0.857
ResNet-50 (224) [6]	0.978	0.848	0.706	0.947	0.895	0.719	0.951	0.750	0.968	0.896
auto-sklearn [7]	0.500	0.186	0.647	0.642	0.906	0.734	0.883	0.595	0.947	0.865
AutoKeras [8]	0.979	**0.864**	0.715	0.939	0.921	0.756	0.956	0.736	0.970	0.918
Google AutoML Vision	**0.982**	0.811	**0.718**	0.947	**0.925**	**0.766**	**0.965**	0.732	**0.993**	**0.941**
Methods	RetinaMNIST		BreastMNIST		OrganMNIST_A		OrganMNIST_C		OrganMNIST_S	
	AUC	ACC	AUC	ACC	AUC	ACC	AUC	ACC	AUC	ACC
ResNet-18 (28) [6]	0.727	0.515	0.897	0.859	0.995	0.921	0.990	0.889	0.967	0.762
ResNet-18 (224) [6]	0.721	0.543	0.915	**0.878**	**0.997**	**0.931**	0.991	0.907	**0.974**	0.777
ResNet-50 (28) [6]	0.719	0.490	0.879	0.853	0.995	0.916	0.990	0.893	0.968	0.746
ResNet-50 (224) [6]	0.717	**0.555**	0.863	0.833	**0.997**	**0.931**	**0.992**	0.898	0.970	0.770
auto-sklearn [7]	0.694	0.525	0.848	0.808	0.797	0.563	0.898	0.676	0.855	0.601
AutoKeras [8]	0.655	0.420	0.833	0.801	0.996	0.929	**0.992**	**0.915**	0.972	**0.803**
Google AutoML Vision	**0.762**	0.530	**0.932**	0.865	0.988	0.818	0.986	0.861	0.964	0.706

下表为我们的结果:

		AUC	ACC
PathMNIST	Resnet18	0.978	0.864
	Resnet50	0.985	0.836
ChestMNIST	Resnet18	0.766	0.947
	Resnet50	0.988	0.871
DermaMNIST	Resnet18	0.915	0.741
	Resnet50	0.903	0.728
PneumoniaMNIST	Resnet18	0.960	0.837
	Resnet50	0.960	0.897
RetinaMNIST	Resnet18	0.740	0.503
	Resnet50	0.745	0.525
BreastMNIST	Resnet18	0.883	0.827
	Resnet50	0.864	0.821
OrganMNIST_C	Resnet18	0.990	0.884
	Resnet50	0.975	0.746

可见，我们的实验结果大部分与论文中的相近，还有几组实验的结果优于论文的结果。实验较为成功。