第3章

细胞图像与深度卷积

应用机器学习方法对生物显微镜生成的细胞图像进行分类识别，与一般的图像识别（如 ImageNet 图像集）相比，面临更为严峻的挑战。这是由细胞微观荧光成像特点决定的。即使采用严苛的实验条件保障同等实验环境，同样的样本在不同批次的成像，往往也会因为实验噪声而存在显著的差异。实验噪声给实验结果带来的不确定性影响，是机器学习面临的另一个挑战。本章系统介绍神经网络与深度卷积技术，探索深度卷积技术在细胞图像分类中的应用。

3.1 数据集

本章项目采用的细胞数据集源自美国生物技术公司 Recursion 发布的 RxRx1 数据集，该公司致力于运用机器学习方法基于大型生物数据集进行药物创新。RxRx1 是一个由显微镜荧光图像组成的数据集，代表 1108 个类别，细胞图像尺寸为 2048×2048×6，共 125 510 个细胞图像，约为 296GB。

本项目是 NeurIPS 2019 的竞赛项目，发布在 Kaggle 上。为了便于建模，Recursion 公司提供了缩小版的数据集，图像尺寸缩小为 512×512×6，数据集规模约为 46GB，可以通过 RxRx1 项目网站下载（https://www.rxrx.ai/）。

数据集包含的文件存放于 chapter3\dataset 目录中，相关信息如表 3.1 所示。

表 3.1 RxRx1 数据集信息描述

文 件 名	数 据 规 模	文件大小	功 能
train.csv	5 列，36 517 行	1.35MB	Id、细胞类型、实验批次、实验微孔和标签
train_controls.csv	6 列，4097 行	225KB	训练集对照组，比训练集多 well_type 列

续表

文　件　名	数 据 规 模	文件大小	功　　能
train.zip	487 356 张图片	30GB	训练集共 81 226 张 6 通道细胞图
test.csv	4 列，19 899 行	562KB	测试集，比训练集少标签列
test_controls.csv	6 列，2246 行	123KB	测试集对照组，比测试集多 well_type、sirna 两列
test.zip	265 728 张图片	16GB	测试集共 44 288 张 6 通道细胞图
pixel_stats.csv	11 列，753 084 行	58.3MB	全部图片像素的统计特征值

3.2　数据采集

RxRx1 项目数据全部来自实验，理解数据采集过程，有助于厘清模型设计的关键点。

RxRx1 项目采用四种细胞系 HUVEC、RPE、HEPG2 和 U2OS 共进行了 51 批次实验，每批次实验采用一种细胞系，实验批次与数据集划分如表 3.2 所示。

表 3.2　实验批次与数据集划分

细胞系	批次编号	训　练　集	测　试　集
HUVEC	01～24	HUVEC-01～HUVEC-16	HUVEC-17～HUVEC-24
RPE	01～11	RPE-01～RPE-07	RPE-08～RPE-11
HEPG2	01～11	HEPG2-01～HEPG2-07	HEPG2-08～HEPG2-11
U2OS	01～05	U2OS-01～U2OS-03	U2OS-04～U2OS-05
合计	51 批次	33 批次	18 批次

图像数据的来源与成像原理，请参照图 3.1 的逻辑示意，左上角方格区域表示 384 微孔板。顾名思义，384 微孔板上有 384(16 列×24 行)个微孔，每个微孔像一个微型试管。为降低实验环境的影响，384 微孔板最外面的行列被空置，因此每块实验板只用 308(14 列×22 行)个微孔。

每一批次的实验，采用四块 384 微孔板，细胞样本分别注入 308×4＝1232 个微孔中。每块实验板上的 30 个微孔添加固定相同的 30 种 siRNA 试剂作为阳性对照组，一个微孔不添加 siRNA 试剂作为阴性对照组，277 个微孔添加 277 种不同的 siRNA 试剂。四块实验板的 277×4＝1108 个微孔共添加 1108 种 siRNA 试剂，1108 种 siRNA 试剂将成为 1108 种基因表达，因此，1108 种 siRNA 将代表 1108 种标签用以标识细胞图像。

四块 384 实验板的 1232 个微孔按照用途分布如下。

(1) 30 个阳性对照微孔/板×4＝120 个(阳性对照样本)。

(2) 1 个阴性对照微孔/板×4＝4 个(阴性对照样本)。

(3) 277 个 siRNA 微孔×4＝1108 个(实验测试样本)。

在单个微孔的两个不同位置(site)采样两次，分别形成两组 512×512×6 图像，图 3.1 右下角显示了在 site1 位置上形成的 6 通道仿彩色图像以及与细胞结构的对应关系，用 6 幅通道图可以合成为一张细胞图。通道图的存储位置与命名规则如图 3.1 左下角所示。

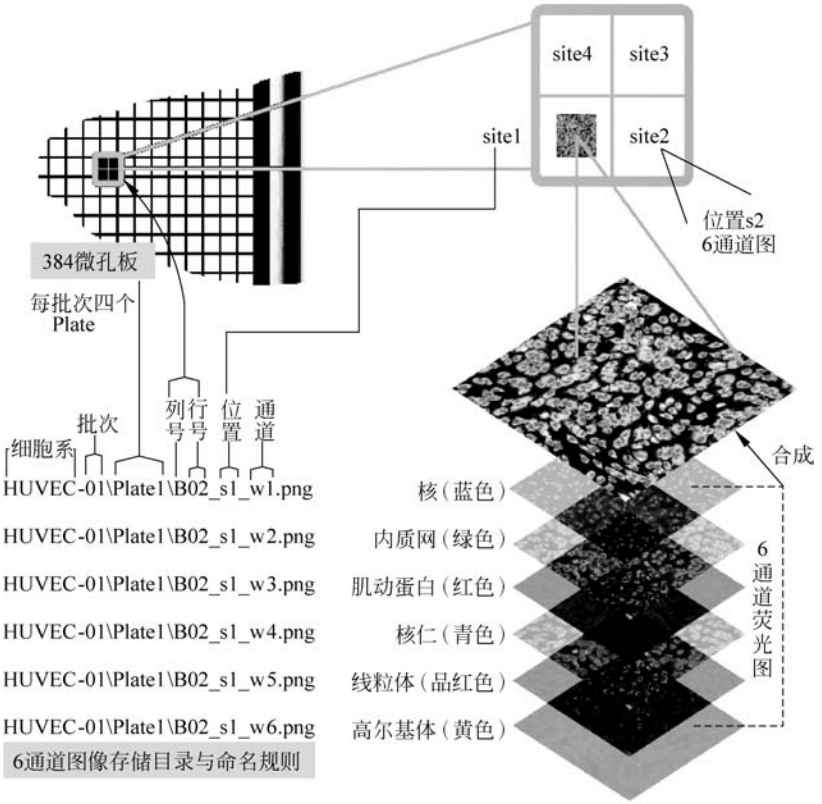


图 3.1　384 微孔板单微孔成像过程示意

单批次实验形成的通道图像数量可以计算如下。

1232(微孔)×2(site)×6(幅通道图)=14 784(幅通道图)

整个数据集包含 51 批次实验,如果实验成功率为 100%,理想情况下,通道照片数量总数为 14 784×51=753 984(幅),以 6 通道图片合成一幅图像方式计算,细胞图像数量为 753 984/6=125 664(张)。照片总数 753 984 与表 3.1 给出的 753 084 略有出入,可以理解为数据采集过程中有舍弃,实践中以实际给出的数据集为准。

3.3　数据集观察

启动 Jupyter Notebook,在 chapter3 目录下新建 Cellular_Classification.ipynb 程序。执行程序段 P3.1,完成库导入。

```
P3.1  #导入库
001   import numpy as np
002   import pandas as pd
003   import matplotlib.pyplot as plt
004   import seaborn as sns
```

```
005    import imageio
006    import h5py
007    %matplotlib inline
```

RxRx1项目的元数据分别定义在train.csv、train_controls.csv、test.csv和test_controls.csv四个文件中。执行程序段P3.2～P3.11，将四个文件合并在一起，以便于对数据做系统性观察与分析。

执行程序段P3.2，观察训练集，训练集维度为(36 517,5)，前5条记录如图3.2所示。

```
P3.2  #训练集train观察
008   train = pd.read_csv("./dataset/train.csv")
009   print('训练集维度: {0}'.format(train.shape))
010   train.head()
```

	id_code	experiment	plate	well	sirna
0	HEPG2-01_1_B03	HEPG2-01	1	B03	sirna_250
1	HEPG2-01_1_B04	HEPG2-01	1	B04	sirna_62
2	HEPG2-01_1_B05	HEPG2-01	1	B05	sirna_1115
3	HEPG2-01_1_B06	HEPG2-01	1	B06	sirna_602
4	HEPG2-01_1_B07	HEPG2-01	1	B07	sirna_529

图3.2　训练集的前5条记录

执行程序段P3.3，训练集train扩增三列，扩增后前五条记录如图3.3所示。

```
P3.3  #训练集train扩增三列
011   train['dataset'] = 'train'
012   train['well_type'] = 'treatment'
013   train['cell_type'] = [train['experiment'][i].partition('-')[0] for i in
      range(train.shape[0])]
014   train.head()
```

	id_code	experiment	plate	well	sirna	cell_type	dataset	well_type
0	HEPG2-01_1_B03	HEPG2-01	1	B03	sirna_250	HEPG2	train	treatment
1	HEPG2-01_1_B04	HEPG2-01	1	B04	sirna_62	HEPG2	train	treatment
2	HEPG2-01_1_B05	HEPG2-01	1	B05	sirna_1115	HEPG2	train	treatment
3	HEPG2-01_1_B06	HEPG2-01	1	B06	sirna_602	HEPG2	train	treatment
4	HEPG2-01_1_B07	HEPG2-01	1	B07	sirna_529	HEPG2	train	treatment

图3.3　训练集扩增后的前5条记录

执行程序段P3.4，观察训练集的实验对照集，维度为(4097,6)，前五条记录如图3.4所示。

```
P3.4  #实验对照集 train_controls 观察
015   train_controls = pd.read_csv("./dataset/train_controls.csv")
016   print('训练对照集维度: {0}'.format(train_controls.shape))
017   train_controls.head()
```

	id_code	experiment	plate	well	sirna	well_type
0	HEPG2-01_1_B02	HEPG2-01	1	B02	UNTREATED	negative_control
1	HEPG2-01_1_C03	HEPG2-01	1	C03	sirna_852	positive_control
2	HEPG2-01_1_C07	HEPG2-01	1	C07	sirna_702	positive_control
3	HEPG2-01_1_C11	HEPG2-01	1	C11	sirna_618	positive_control
4	HEPG2-01_1_C15	HEPG2-01	1	C15	sirna_272	positive_control

图 3.4　实验对照集的前五条记录

执行程序段 P3.5,训练集的实验对照集 train_controls 扩增两列,扩增后前五条记录如图 3.5 所示。

```
P3.5  #实验对照集 train_controls 扩增两列
018   train_controls['dataset'] = 'train_controls'
019   train_controls['cell_type'] = [train_controls['experiment'][i].partition('-')[0]
                for i in range(train_controls.shape[0])]
020   train_controls.head()
```

	id_code	experiment	plate	well	sirna	well_type	dataset	cell_type
0	HEPG2-01_1_B02	HEPG2-01	1	B02	UNTREATED	negative_control	train_controls	HEPG2
1	HEPG2-01_1_C03	HEPG2-01	1	C03	sirna_852	positive_control	train_controls	HEPG2
2	HEPG2-01_1_C07	HEPG2-01	1	C07	sirna_702	positive_control	train_controls	HEPG2
3	HEPG2-01_1_C11	HEPG2-01	1	C11	sirna_618	positive_control	train_controls	HEPG2
4	HEPG2-01_1_C15	HEPG2-01	1	C15	sirna_272	positive_control	train_controls	HEPG2

图 3.5　实验对照集扩增后的前五条记录

执行程序段 P3.6,观察测试集,测试集维度为(19 899,4),前五条记录如图 3.6 所示。

```
P3.6  #测试集 test 观察
021   test = pd.read_csv("./dataset/test.csv")
022   print('测试集维度: {0}'.format(test.shape))
023   test.head()
```

执行程序段 P3.7,测试集 test 扩增四列,扩增后前五条记录如图 3.7 所示。

```
P3.7  #测试集 test 扩增四列
024   test['dataset'] = 'test'
025   test['well_type'] = 'unknow'
```

	id_code	experiment	plate	well
0	HEPG2-08_1_B03	HEPG2-08	1	B03
1	HEPG2-08_1_B04	HEPG2-08	1	B04
2	HEPG2-08_1_B05	HEPG2-08	1	B05
3	HEPG2-08_1_B06	HEPG2-08	1	B06
4	HEPG2-08_1_B07	HEPG2-08	1	B07

图 3.6　测试集前五条记录

```
026    test['cell_type'] = [test['experiment'][i].partition('-')[0] for i in range(test.
       shape[0])]
027    test['sirna'] = 'unknow'
028    test.head()
```

	id_code	experiment	plate	well	dataset	well_type	cell_type	sirna
0	HEPG2-08_1_B03	HEPG2-08	1	B03	test	unknow	HEPG2	unknow
1	HEPG2-08_1_B04	HEPG2-08	1	B04	test	unknow	HEPG2	unknow
2	HEPG2-08_1_B05	HEPG2-08	1	B05	test	unknow	HEPG2	unknow
3	HEPG2-08_1_B06	HEPG2-08	1	B06	test	unknow	HEPG2	unknow
4	HEPG2-08_1_B07	HEPG2-08	1	B07	test	unknow	HEPG2	unknow

图 3.7　测试集扩增后的前五条记录

执行程序段 P3.8，观察测试集的实验对照集，维度为(2246,6)，前五条记录如图 3.8 所示。

```
P3.8   #测试实验组的对照集 test_controls 观察
029    test_controls = pd.read_csv("./dataset/test_controls.csv")
030    print('测试对照集维度: {0}'.format(test_controls.shape))
031    test_controls.head()
```

	id_code	experiment	plate	well	sirna	well_type
0	HEPG2-08_1_B02	HEPG2-08	1	B02	UNTREATED	negative_control
1	HEPG2-08_1_C03	HEPG2-08	1	C03	sirna_650	positive_control
2	HEPG2-08_1_C07	HEPG2-08	1	C07	sirna_323	positive_control
3	HEPG2-08_1_C11	HEPG2-08	1	C11	sirna_810	positive_control
4	HEPG2-08_1_C15	HEPG2-08	1	C15	sirna_577	positive_control

图 3.8　测试集的实验对照集前五条记录

执行程序段 P3.9，test_controls 扩增两列，扩增后前五条记录如图 3.9 所示。

```
P3.9   #测试实验组的对照集 test_controls 扩增两列
032    test_controls['dataset'] = 'test_controls'
033    test_controls['cell_type'] = [test_controls['experiment'][i].partition('-')[0]
                                     for i in range(test_controls.shape[0])]
034    test_controls.head()
```

	id_code	experiment	plate	well	sirna	well_type	dataset	cell_type
0	HEPG2-08_1_B02	HEPG2-08	1	B02	UNTREATED	negative_control	test_controls	HEPG2
1	HEPG2-08_1_C03	HEPG2-08	1	C03	sirna_650	positive_control	test_controls	HEPG2
2	HEPG2-08_1_C07	HEPG2-08	1	C07	sirna_323	positive_control	test_controls	HEPG2
3	HEPG2-08_1_C11	HEPG2-08	1	C11	sirna_810	positive_control	test_controls	HEPG2
4	HEPG2-08_1_C15	HEPG2-08	1	C15	sirna_577	positive_control	test_controls	HEPG2

图 3.9　测试集的实验对照集扩增后前五条记录

执行程序段 P3.10，将四个数据文件合并为一个数据集 combined，维度为(62 759，8)。

```
P3.10  #四个数据集文件合并
035    frames = [train, train_controls, test, test_controls]
036    combined = pd.concat(frames, sort = False)
037    print(combined.shape)
038    combined.head()
```

执行程序段 P3.11，检查缺失值，结果显示所有列无缺失值存在。

```
P3.11  #缺失值检查
039    combined.isnull().sum()
```

3.4　数据分布

执行程序段 P3.12，统计分析 train、test、train_controls、test_controls 四种数据集的细胞系分布情况，结果如图 3.10 所示。

```
P3.12  #细胞系类型分布统计
040    for col in ['train','test','train_controls','test_controls']:
041        x = combined[combined.dataset == col]['cell_type'].value_counts().index
042        y = combined[combined.dataset == col]['cell_type'].value_counts()
043        plt.bar(x,y,label = col,alpha = 0.7)
044        plt.legend()
```

图 3.10 显示四种细胞系的分布与实验设计保持一致。U2OS 细胞系的实验最少，HUVEC 细胞系实验最多，RPE 和 HEPG2 一样多。图 3.10 柱形图自下而上，依次为测试对照、训练对照、测试集、训练集样本的数据量。

执行程序段 3.13，统计分析 train、test、train_controls、test_controls 四种数据集的标签数量与分布情况，结果如表 3.3 所示。

```
P3.13  #标签分布统计
045    for col in ['train','test','train_controls','test_controls']:
046        labels = combined[combined.dataset == col]['sirna'].value_counts()
047        print('\n{0}标签数: {1}，重复数前五名的标签为:
                  \n{2}'.format(col,len(labels),labels.head(5)))
```

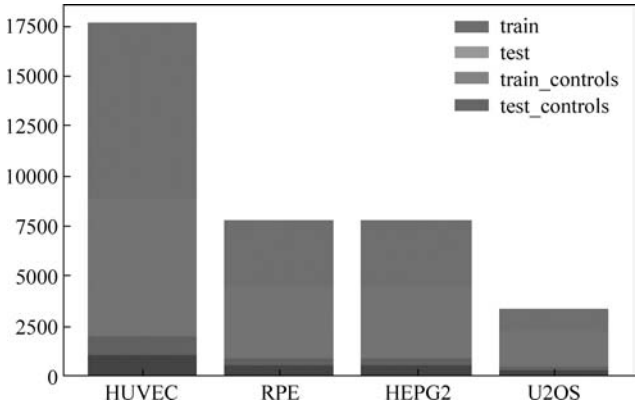


图 3.10　细胞系类型分布统计

表 3.3　标签分布统计

数　据　集	标签数	说　　明
train	1108	对应实验设计中的 1108 种 siRNA 标签
test	1	均为未知标签 unknown
train_controls	31	30 种阳性对照组标签，1 种阴性对照组标签
test_controls	31	30 种阳性对照组标签，1 种阴性对照组标签

执行程序段 P3.14，将 train 标签与 train_controls 标签做汇总统计，运行结果显示二者没有重复的 siRNA 标签，这说明对照组的 siRNA 与非对照组的 siRNA 是不同的。

```
P3.14   #train 标签与 train_controls 标签是否有重复?
048     set1 = set(list(combined[combined.dataset == 'train']['sirna'].unique())).intersection(
            set(list(combined[combined.dataset == 'train_controls']['sirna'].unique())))
049     print('标签重复数: {0}'.format(len(set1)))
```

那么训练对照组与测试对照组的 siRNA 是否有差别呢？执行程序段 P3.15，将 train_controls 与 test_controls 标签汇总统计，结果显示重复标签数为 31，证明二者完全相同。

```
P3.15   #train_controls 标签与 test_controls 标签是否有重复?
050     set2 = set(list(combined[combined.dataset == 'train_controls']['sirna'].unique())).
        intersection(
            set(list(combined[combined.dataset == 'test_controls']['sirna'].unique())))
051     print('标签重复数: {0}'.format(len(set2)))
```

基于上述分析，将 train 与 train_controls 两个数据集合并为训练集是比较合理的做法，但是一般的台式计算机难以承担其计算力需求，所以本章案例只选取实验批次数量最少的 U2OS 细胞系的数据集做模型训练与测试，将注意力放在解决复杂问题的方法步骤上。读者仍可以随时根据计算能力调整训练集规模。

3.5 筛选数据集

为了简化计算规模，执行程序段 P3.16，选取 U2OS 细胞系三个批次的实验数据，构建数据集 U2OS_train，结果显示数据集维度为(3324,2)。后面会将该数据集按照一定比例划分为训练集与验证集。

```
P3.16  #筛选用于训练和验证的数据集
all_train = combined[(combined.dataset == 'train')]
U2OS_train = all_train[all_train['experiment'].str.contains('U2OS')]
U2OS_train = U2OS_train[['id_code', 'sirna']]
U2OS_train = U2OS_train.reset_index(drop = True)
print(U2OS_train.shape)
```

程序段 P3.16 得到的数据集只保留了实验 Id 列和标签列，后面需要进一步做特征提取工作。

为了直观观察训练数据的特点，执行程序段 P3.17，完成单个实验 Id 对应的 site1 位置的 6 通道图像的特征提取与图像显示，运行结果如图 3.11 所示。

```
P3.17  #显示指定 Id 对应的 site1 位置的 6 通道图像
def load_image(basepath, id_code, site):
    images = np.zeros(shape = (6,512,512))
    path = id_code.partition('_')[0] + '/plate' + id_code.partition('_')[2][0] \
    + '/' + id_code.rpartition('_')[2]
    images[0,:,:] = imageio.imread(basepath + path + site + '_w1' + '.png')
    images[1,:,:] = imageio.imread(basepath + path + site + '_w2' + ".png")
    images[2,:,:] = imageio.imread(basepath + path + site + '_w3' + ".png")
    images[3,:,:] = imageio.imread(basepath + path + site + '_w4' + ".png")
    images[4,:,:] = imageio.imread(basepath + path + site + '_w5' + ".png")
    images[5,:,:] = imageio.imread(basepath + path + site + '_w6' + ".png")
    return images
fig, ax = plt.subplots(2,3,figsize = (18,10))
images = load_image('./dataset/train/', U2OS_train['id_code'][40], '_s1')
ax[0][0].imshow(images[0], cmap = "Blues")
ax[0][1].imshow(images[1], cmap = "Greens")
ax[0][2].imshow(images[2], cmap = "hot")
ax[1][0].imshow(images[3], cmap = "viridis")
ax[1][1].imshow(images[4], cmap = "gist_heat")
ax[1][2].imshow(images[5], cmap = "pink")
```

如图 3.11 所示为数据集 U2OS_train 中第 40 个实验 Id 对应的 site1 位置的 6 通道图像，从左到右，第一行依次为核、内质网和肌动蛋白，第二行为核仁、线粒体和高尔基体。

图 3.11 中的颜色是为了便于观察添加的滤镜效果，这些图像在 train 目录下都是灰度图像，3.23 节将介绍将其聚合为一张彩图的方法。

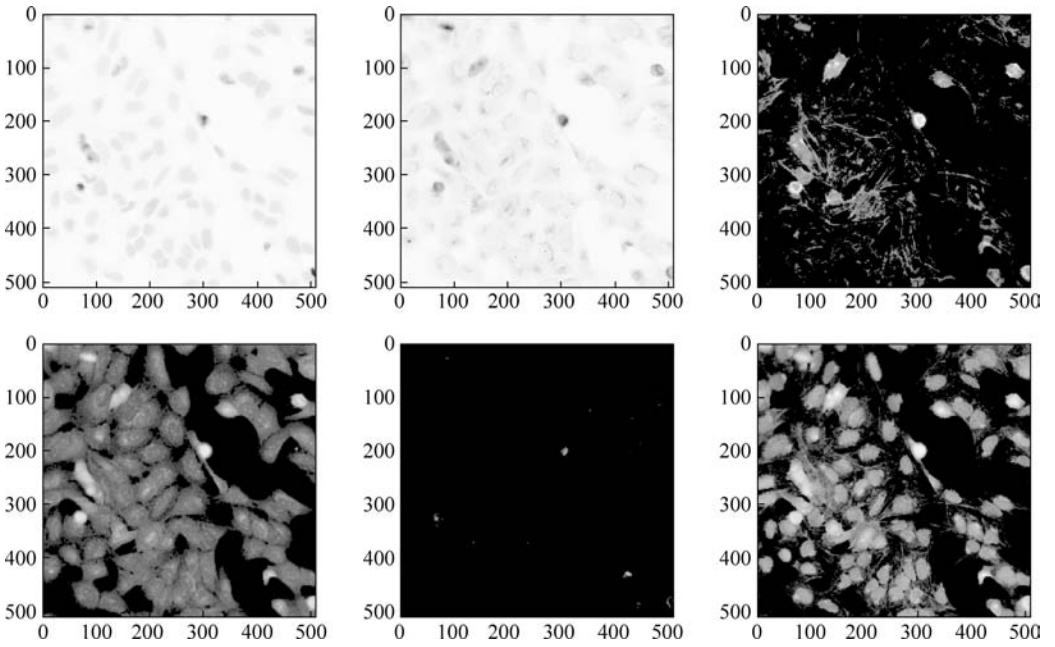


图 3.11　单个实验 Id 对应的 site1 位置的 6 通道图像

3.6　神经网络

人工神经网络(Artificial Neural Network,ANN)的灵感源于人类对生物神经网络的理解与认知,为简单起见,本书将人工神经网络简称为神经网络。

神经网络的基本构成单位是神经元,又称计算单元。

神经元按照一定的规则相互连接形成神经网络。

单个神经元可接受外部输入,完成数学计算,形成目标输出,可以视作最简单的神经网络。图 3.12 是一个能够预测房价的简单神经网络结构图。

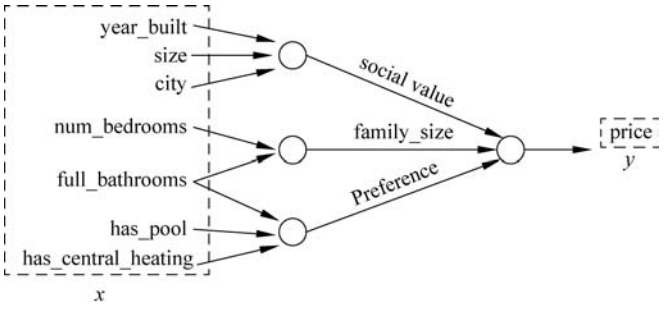


图 3.12　一个预测房价的神经网络结构

如图 3.12 所示,房价评估逻辑是:左侧虚线框里的指标称为房子特征,房子的建筑年代、面积和所在的城市,可能会决定房子的基本社会价值;卧室数量和带淋浴的洗手间

数量，可能会决定房子适宜居住的家庭人数；有的人会认为带淋浴的洗手间、小区泳池、集中供暖等便利设施会为房产带来更高的性价比。社会价值、家庭人数和性价比等因素综合决定房价。

图中的圆圈○称为神经元，神经元是能够把输入变为合理输出的功能单元。神经元是如何完成功能计算的呢？如图 3.13 所示，神经元除了输入和输出，内部结构上可以分为两部分，一部分用来对输入的特征值做线性变换，如公式(3.1)所示，称作输入函数。第二部分对线性变换的结果再做非线性变换，形成神经元的输出值，称作激励函数，激励函数有多种类型可用，此处的激励函数 σ 如公式(3.2)所示。

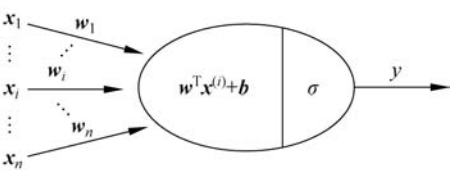


图 3.13　神经元的基本结构

$$z^{(i)} = w^{\mathrm{T}} x^{(i)} + b \tag{3.1}$$

$$y = \sigma(z) = \frac{1}{1+\mathrm{e}^{-z}} \tag{3.2}$$

其中，

x：神经元的输入。

w：参数矩阵。

b：偏差向量。

z：线性计算值。

e：自然常数。

σ：激励函数。

y：神经元的输出。

如图 3.12 所示的房价预测模型，在神经元的前后连接关系上，选用某些特征定义房子的价值，这是融入了人类的先验知识与经验判断，事实上对于更多的问题建模，人类难以预先判断各种特征对目标值的准确影响，这个时候，用一个全连接的网络(假定所有元素之间都有联系)作为初始模型，让机器在训练过程中去学习连接的权重参数，让权重参数决定元素之间的相关程度是一个不错的选择。所以，如图 3.12 所示的模型，实践中会定义为如图 3.14 所示的全连接模型。

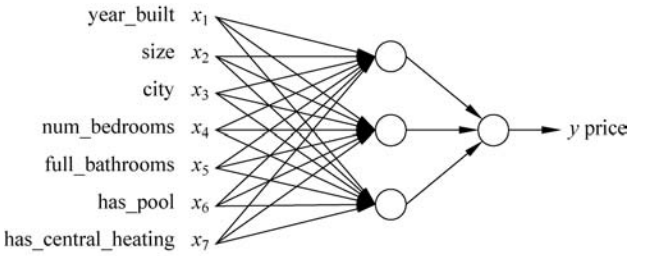


图 3.14　房价全连接神经网络模型

不难看出，神经网络是一个分层结构，如图 3.15 所示的网络可分为四层，第一层又称为输入层，最后一层称作输出层，中间的各层统称为隐藏层。由于输入层仅表示特征输

入，没有神经元，在统计神经网络层数时，一般不把输入层计算在内，所以图 3.15 是一个三层神经网络，包括两个隐藏层和一个输出层。

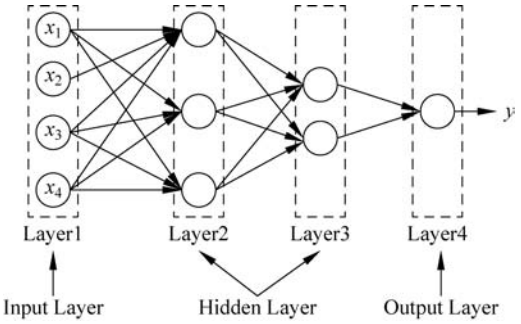


图 3.15　神经网络的分层结构

3.7　符号化表示

如图 3.16 所示是一个拥有三层结构的神经网络，这个神经网络能够解决什么问题？有多少个输入特征？有多少层？有多少个输出？这些都是必须回答的问题。

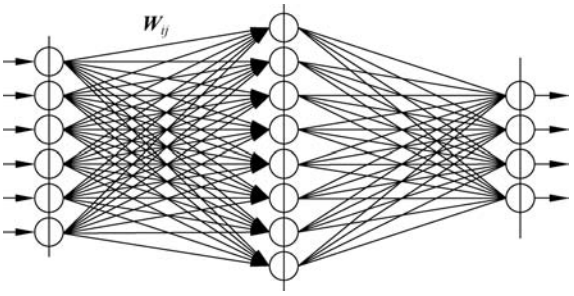


图 3.16　神经网络的图形化示意

将神经网络从图形化示意图变换到抽象的数学模型表示，离不开图形的符号化表示。如图 3.17 所示，以一个两层网络为例，输入层如何符号化？隐藏层如何符号化？输出层如何符号化？

输入层用向量 $\boldsymbol{x}=(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3)$ 表示。输入函数用 $\boldsymbol{z}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{b}$ 表示，$\boldsymbol{w}$ 和 $\boldsymbol{b}$ 是需要学习和训练的参数，$\boldsymbol{z}$ 是输入函数的输出值。非线性变换的激励函数用 $\boldsymbol{a}=g(z)$ 表示，g 表示激励函数，$\boldsymbol{a}$ 表示激励函数的输出值。输出层的输出用 $\hat{y}$ 表示。一些常见符号代表的含义如下。

（1）$n^{[l]}$：第 ℓ 层神经元的数量。

（2）$\boldsymbol{w}^{[l]}$：第 ℓ 层的权重矩阵。

（3）$\boldsymbol{b}^{[l]}$：第 ℓ 层的偏差向量。

（4）$\boldsymbol{a}^{[l]}$：第 ℓ 层的输出向量。输入层可以表示为 $\boldsymbol{a}^{[0]}$，最后一层（L 层）的输出可表示为 $\hat{y}=\boldsymbol{a}^{[L]}$。

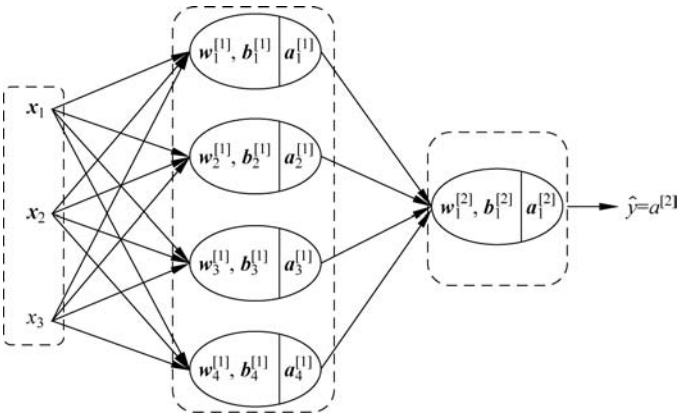


图 3.17 神经网络的符号化表示

(5) $\boldsymbol{z}^{[l]}$：第 ℓ 层线性变换的输出向量。

(6) $\boldsymbol{x}^{(i)}$：第 i 个样本的输入向量。

(7) $\boldsymbol{a}_j^{[l](i)}$：第 i 个样本在第 ℓ 层的第 j 个神经元的输出值。

(8) $\boldsymbol{w}_{jk}^{[l]}$：第 ℓ 层的第 j 个神经元对应第 k 个特征的权重参数。

(9) $\boldsymbol{b}_j^{[l]}$：第 ℓ 层的第 j 个神经元的偏差参数。

3.8 激励函数

神经元是构成神经网络的基本单位，定义神经元输出逻辑的函数，称为激励函数，也称激活函数。激励函数不但表征了单个神经元的输出逻辑，而且作为下一层的输入，也在建构着层与层之间神经元的关系，对整个神经网络的功能逻辑有重要影响。如图 3.18 所示，输入函数 $\boldsymbol{z}=\sum_{i=1}^{n}\boldsymbol{w}_i\boldsymbol{x}_i+\boldsymbol{b}$ 是线性函数，如果激励函数 $\sigma(\boldsymbol{z})$ 也是线性函数，那么不管神经网络有多少层，最后迭代的结果必然是线性的。所以，激励函数一般选用非线性函数。

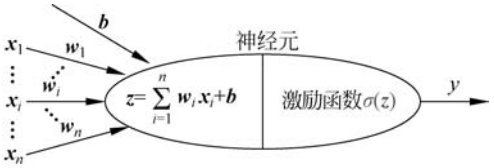


图 3.18 神经元与激励函数

历史上，感知机一度因为无法解决线性不可分问题陷入困境。以分类识别为例，如图 3.19 所示，a 是线性可分的，b 和 c 都是非线性可分的，对于 b 和 c，$\sigma(\boldsymbol{z})$ 采用线性函数则无解，此时 $\sigma(\boldsymbol{z})$ 应该表示为非线性函数。

下面给出一些常见的激励函数。

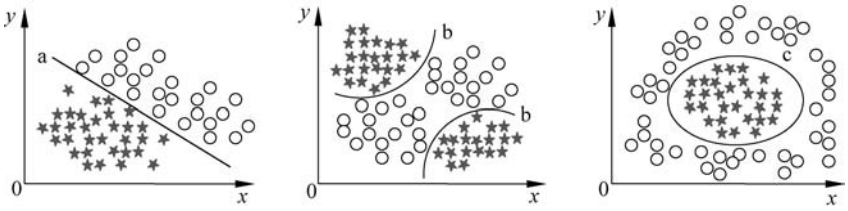


图 3.19　线性可分与非线性可分

1. Sigmoid 函数

函数形式：$\sigma(z)=\dfrac{1}{1+e^{-z}}$。导数形式：$\sigma'(z)=\sigma(z)(1-\sigma(z))$。

Sigmoid 函数能够把输入的连续实值变换为 0 和 1 之间的输出，如果是非常大的负数，那么输出则无限逼近 0；如果是非常大的正数，则输出无限逼近 1。

主要特点如下。

(1) Sigmoid 函数的值域为(0,1)，又称逻辑回归函数。

(2) Sigmoid 函数连续并严格单调递增，其反函数也单调递增。

(3) Sigmoid 函数关于点(0,0.5)对称。

(4) Sigmoid 函数的导数是以它本身为自变量的函数，即 $\sigma'(z)=F(\sigma(z))$。

(5) Sigmoid 函数源于生物学现象，其曲线被称为 S 形生长曲线。

函数图像如图 3.20 所示。

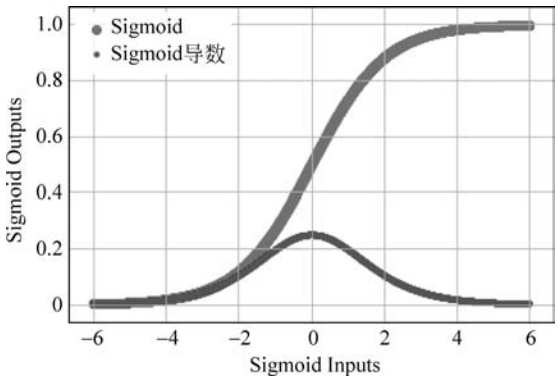


图 3.20　Sigmoid 函数及其导数

Sigmoid 函数在远离原点的两端，存在梯度消失的缺点。

2. 双曲正切(tanh)函数

函数形式：$f(z)=\tanh(z)=\dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$。导数形式：$f'(z)=1-f(z)^{2}$。

函数图像如图 3.21 所示。

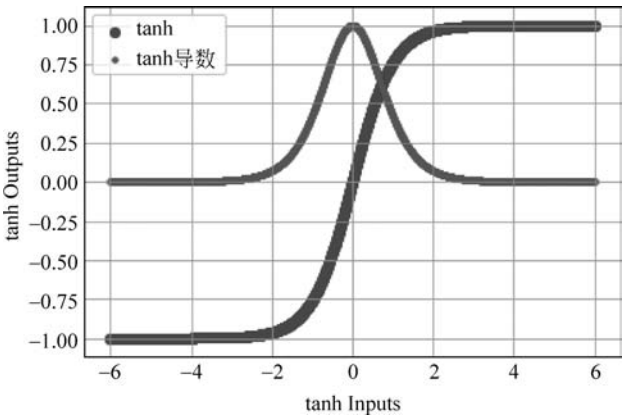


图 3.21 双曲正切函数及其导数

双曲正切函数解决了 Sigmoid 函数不是(0,0)对称的问题,但是从其导数曲线不难看出,在 z 取较大的负值或正值时,仍然存在梯度消失问题。

3. 修正线性单元(Rectified Linear Unit,ReLU)函数

ReLU 函数形式:$f(z)=\begin{cases}0, & z<0\\ z, & z\geqslant 0\end{cases}$。导数形式:$f'(z)=\begin{cases}0, & z<0\\ 1, & z\geqslant 0\end{cases}$。

函数图像如图 3.22 所示。

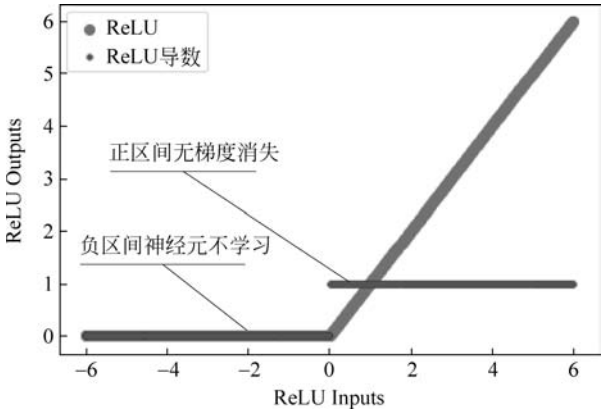


图 3.22 ReLU 函数及其导数

ReLU 函数其实是一个取最大值函数,函数形式也可以表示为 $f(z)=\max(0,z)$。ReLU 函数优点如下。

(1) 在正区间解决了梯度消失问题。

(2) 计算速度非常快,只需要判断输入是否大于 0。

(3) 收敛速度远快于 Sigmoid 函数和 tanh 函数。

其缺点如下。

(1) ReLU 的输出不是(0,0)对称的。

(2) 存在死亡神经元问题(Dead ReLU Problem)。如图 3.22 所示,导数在 x＜0 时恒为 0,这意味着某些神经元可能永远不会被激励,导致相应的权重参数可能永远不能被更新。

尽管如此,ReLU 因其显著的实践效果,是最受欢迎的激励函数之一。ReLU 有多个改进版本,例如,双极修正线性单元(Bipolar Rectified Linear Unit,BReLU)、泄漏修正线性单元(Leaky Rectified Linear Unit,Leaky ReLU)、参数修正线性单元(Parametric Rectified Linear Unit,PReLU)和随机泄漏修正线性单元(Randomized Leaky Rectified Linear Unit,RReLU)。下面单独介绍 Leaky ReLU 函数。

4. Leaky ReLU 函数

函数形式:$f(z)=\begin{cases}0.01z, & z<0\\ z, & z\geqslant 0\end{cases}$。导数形式:$f'(z)=\begin{cases}0.01, & z<0\\ 1, & z\geqslant 0\end{cases}$。

函数图像如图 3.23 所示。

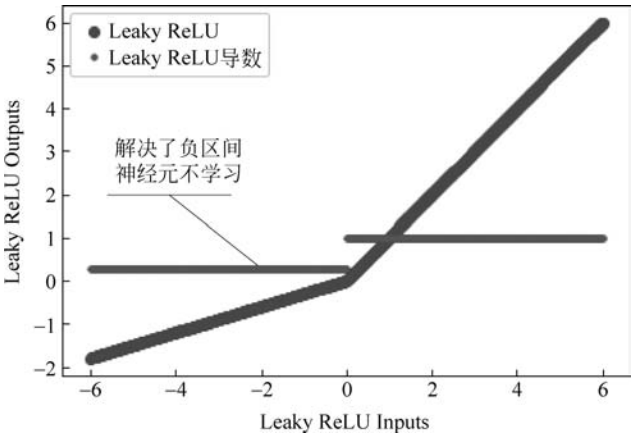


图 3.23　Leaky ReLU 函数及其导数

Leaky ReLU 将 $z<0$ 时的取值修改为 $0.01z$,这样一来,Leaky ReLU 的导数在负区间为非 0 值,从而解决了负区间存在的不学习问题。

5. 指数线性单元(Exponential Linear Unit,ELU)函数

函数形式:$f(\alpha,z)=\begin{cases}\alpha(e^z-1), & z\leqslant 0\\ z, & z>0\end{cases}$。导数形式:$f'(\alpha,z)=\begin{cases}f(\alpha,z)+\alpha, & z\leqslant 0\\ 1, & z>0\end{cases}$。

函数图像如图 3.24 所示。

ELU 函数是针对 ReLU 函数的一个改进,相比于 ReLU 函数,在输入为负数的情况下,有一定的输出且鲁棒性好。

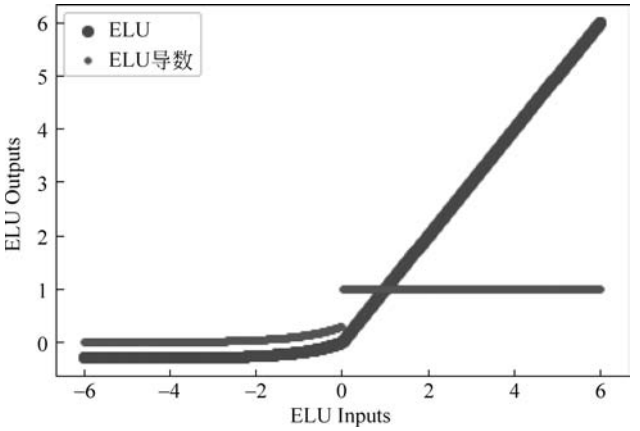


图 3.24　ELU 函数及其导数

6. Softmax 函数

Softmax 函数一般只用在输出层，在多分类应用中可将神经元的输入函数 z 归一化到(0,1)区间，所有 $f(z_i)$之和为 1。

假定有 k 个分类，则函数形式可表示为：

$$f(z_i)=\frac{e^{z_i}}{\sum_{j=1}^{k}e^{z_j}} \quad i,j=1,2,\cdots,k$$

导数形式为：

$$\frac{\partial f(z_i)}{\partial z_j}=f(z_i)(\delta_{ij}-f(z_j)) \quad \delta_{ij}=\begin{cases}0, & i\neq j\\ 1, & i=j\end{cases} \quad i,j=1,2,\cdots,k$$

函数图像如图 3.25 所示。

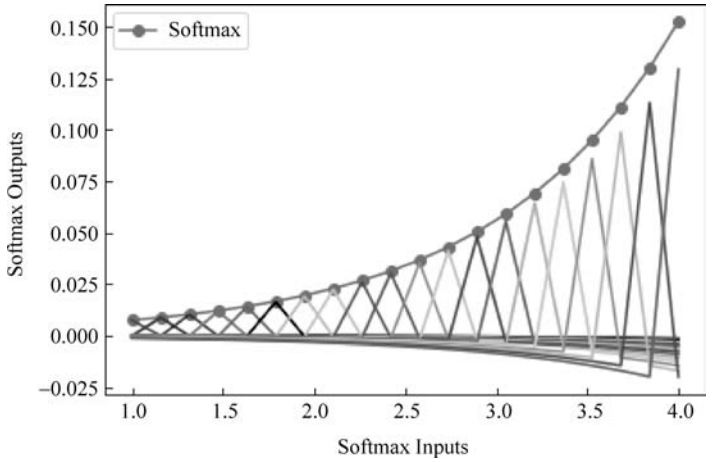


图 3.25　Softmax 函数及其偏导数

实践中常常根据应用的需要，选用不同的激励函数，也可自定义激励函数，一切以实践效果为依据。例如，如果是二分类问题，用 Sigmoid 函数作为输出层的激励函数是一个好的选择，如果是中间各层的变换，ReLU 函数的计算量较小，而且往往效果显著。

3.9　损失函数

损失函数(Lost Function)，又称成本函数、代价函数、目标函数或优化函数。损失函数不但能够衡量模型预测值与真实值偏离的程度，而且通过求解损失函数的最小值，可以实现求解模型参数、优化模型参数和评价模型学习效果的目的。

假设样本集的规模为 m，训练集表示为$(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(i)},y^{(i)}),\cdots,(x^{(m)},y^{(m)})$。

m 个样本的特征矩阵表示为：

$$\boldsymbol{X}=\begin{bmatrix}\vdots & \vdots & & \vdots\\ x^{(1)} & x^{(2)} & \cdots & x^{(m)}\\ \vdots & \vdots & & \vdots\end{bmatrix}\quad \boldsymbol{X}\in \mathrm{R}^{n\times m}$$

标签矩阵表示为：

$$\boldsymbol{Y}=[y^{(1)},y^{(2)},\cdots,y^{(m)}]\quad \boldsymbol{Y}\in \mathrm{R}^{1\times m}$$

模型的预测值表示为：

$$\hat{\boldsymbol{Y}}=[\hat{y}^{(1)},\hat{y}^{(2)},\cdots,\hat{y}^{(m)}]\quad \hat{\boldsymbol{Y}}\in \mathrm{R}^{1\times m}$$

一种理想的情况是，对于每一个样本$(x^{(i)},y^{(i)})$，都有 $\hat{y}^{(i)}\approx y^{(i)}$。为了衡量第 i 个样本预测值与真实值的误差，可用预测值与真实值差值的平方表示，即可定义单样本损失函数为：$L(\hat{y}^{(i)},y^{(i)})=(\hat{y}^{(i)}-y^{(i)})^2$。

评估 m 个样本的损失，一般用全部样本损失的均方误差表示，如公式(3.3)所示。

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})=\frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)}-y^{(i)})^2 \tag{3.3}$$

公式(3.3)是一个关于参数 w 和 b 的函数，模型求解和优化的目标，可以理解为求解损失函数 $J(w,b)$在整个样本集上的最小值。

公式(3.3)适合作为回归类问题的损失函数，对于分类问题，损失函数一般用交叉熵函数表示。以逻辑回归(Logistic Regression)为例，单样本交叉熵损失函数如公式(3.4)所示。

$$L(\hat{y}^{(i)},y^{(i)})=-(y^{(i)}\log(\hat{y}^{(i)})+(1-y^{(i)})\log(1-\hat{y}^{(i)})) \tag{3.4}$$

m 个样本的交叉熵损失函数可表示为公式(3.5)。

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})=-\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\hat{y}^{(i)})+(1-y^{(i)})\log(1-\hat{y}^{(i)})] \tag{3.5}$$

在各种流行的机器学习框架中，预定义了多种损失函数，实践中可灵活选择，也可根据问题需要自定义损失函数。

3.10 梯度下降

梯度下降，就是沿着函数的梯度（导数）方向更新自变量，使得函数的取值越来越小，直至达到全局最小或者局部最小。

先看一维梯度下降问题。假定 $J(w)$ 是关于 w 的损失函数，欲求 w 取何值时 $J(w)$ 达到最小值，可以采用如图 3.26 所示的算法。

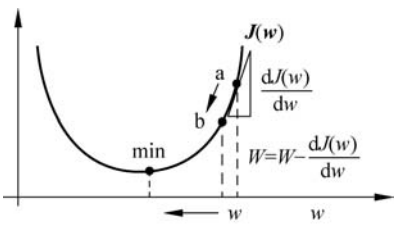


图 3.26 一维梯度下降

（1）随机选择一个 w 值，求得此处的 $J(w)$，如 a 点所示。

（2）求得 a 点的导数 $\frac{\mathrm{d}J(w)}{\mathrm{d}w}$，即 a 点的梯度。

（3）沿着梯度方向，移动点 a 到点 b，此时有 $w_b = w_a - \frac{\mathrm{d}J(w)}{\mathrm{d}w}$，称 w_b 是通过梯度下降求得的一个值。其实梯度是一个很小的值，图 3.26 是为了便于观察，拉大了 b 点与 a 点的距离。

（4）为了调整单步下降的幅度，通常对梯度项乘以一个系数 α，即有 $w_b = w_a - \alpha \frac{\mathrm{d}J(w)}{\mathrm{d}w}$，其中，$\alpha$ 被称作学习率。

（5）沿着梯度方向反复迭代，可以无限逼近 min 点，此时求得的 $w_{\min}$，即为使得 $J(w)$ 最小的最优值。

图 3.26 中的随机初值在曲线的右半部分，随着梯度下降，w 的取值越来越小，直至逼近最优值；如果初值取在左半部分，此时梯度值为负，随着梯度下降，w 的取值会越来越大，直至逼近最优值，两种情况都是梯度下降方法，与初值位置无关。

对于多维梯度下降问题，即目标函数有多个自变量，其梯度可以通过偏导数计算。以二维函数 $J(w,b)$ 为例，有 w 和 b 两个方向，需要计算沿着两个方向的梯度下降，其迭代过程如公式(3.6)和公式(3.7)所示。

$$W = W - \alpha \frac{\partial J(w,b)}{\partial(w)} \tag{3.6}$$

$$b = b - \alpha \frac{\partial J(w,b)}{\partial(b)} \tag{3.7}$$

下面以单个神经元为例，假定该神经元只有两个输入特征 x_1 和 x_2，则其单步梯度下降计算过程如图 3.27 所示。

图 3.27 的计算过程解释如下。

（1）输入函数中的 w_1、w_2 和 b 是需要学习训练的参数。

（2）激励函数采用 Sigmoid 函数，损失函数用交叉熵表示。

（3）梯度计算从损失函数开始向前倒推，先计算 $\mathrm{d}a$，然后是 $\mathrm{d}z$，最后是在三个参数方向的梯度 $\mathrm{d}w_1$、$\mathrm{d}w_2$ 和 $\mathrm{d}b$。

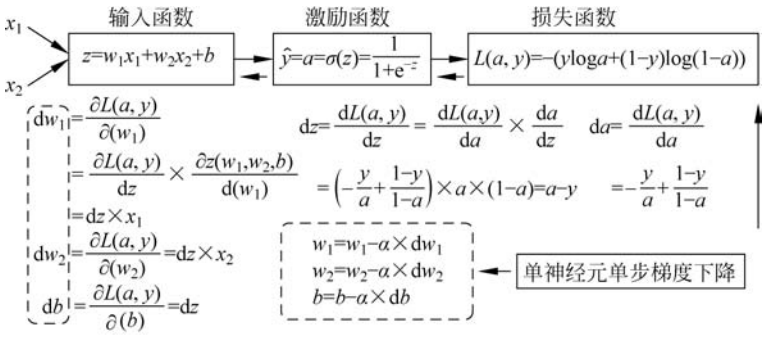


图 3.27　单个神经元的单步梯度下降计算过程

(4) 用得到的三个方向的梯度，计算得到新的 w_1、w_2 和 b 的值，如图 3.27 中虚线框中的公式所示。

图 3.27 可以理解为神经元的单样本单步梯度下降过程。

假定训练集有 m 个样本，那么梯度下降应该如何进行？如果一次输入一个样本，即完成梯度下降一步，这种方法称为随机梯度下降。如果 m 个样本同时输入网络，得到 m 个样本的梯度后取各个维度方向上的平均梯度，这种梯度下降方法称为批量梯度下降法。如果将 m 个样本划分为若干批，每次用一批样本的梯度均值决定下降的幅度，称为小批量梯度下降。

理解了单神经元的梯度下降，再来学习神经网络的梯度下降就会水到渠成。神经网络是由若干层组成的，每一层又有若干神经元，因此，神经网络的梯度下降首先以层为考量，层又以神经元为考量，梯度下降的进一步描述，请参见正向传播与反向传播。

3.11　正向传播

正向传播(Forward Propagation)，又称前向传播，计算过程是：将输入层的样本值 x，传递给第一层的输入函数得到 z 值，z 值传递给激励函数得到 a 值，a 值既是第一层的输出，也是第二层的输入，以此类推，从前向后逐层计算，直至得到输出层的估计值 $\hat{y}$，这种根据样本值从输入到输出的计算过程称作神经网络的正向传播。

以图 3.17 所示的两层网络为例，假定各层采用的激励函数为 σ，其正向传播的逐层计算过程如图 3.28 所示。

图 3.29 给出了一个四层神经网络的结构化表示与参数提示，各层正向传播的计算可分为两个步骤，一是输入函数的计算，如公式(3.8)所示；二是激励函数的计算，如公式(3.9)所示。

$$z^{[l]}=w^{[l]}a^{[l-1]}+b^{[l]} \tag{3.8}$$

$$a^{[l]}=g^{[l]}(z^{[l]}) \tag{3.9}$$

根据公式(3.8)和公式(3.9)，样本 $x^{(i)}$ 的正向传播计算过程如图 3.30 所示。

请结合微课视频，深入理解正向传播的计算过程。

$$\text{Layer1}\quad z^{[1](i)}=\begin{bmatrix}z_1^{[1](i)}\\ z_2^{[1](i)}\\ z_3^{[1](i)}\\ z_4^{[1](i)}\end{bmatrix}=\begin{bmatrix}w_1^{[1]}\\ w_2^{[1]}\\ w_3^{[1]}\\ w_4^{[1]}\end{bmatrix}x^{(i)}+b^{[1]}=\begin{bmatrix}w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]}\\ w_{21}^{[1]} & w_{22}^{[1]} & w_{23}^{[1]}\\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]}\\ w_{41}^{[1]} & w_{42}^{[1]} & w_{43}^{[1]}\end{bmatrix}\begin{bmatrix}x_1^{(i)}\\ x_2^{(i)}\\ x_3^{(i)}\end{bmatrix}+\begin{bmatrix}b_1^{[1]}\\ b_2^{[1]}\\ b_3^{[1]}\\ b_4^{[1]}\end{bmatrix}=\begin{bmatrix}w_{11}^{[1]}x_1^{(i)}+w_{12}^{[1]}x_2^{(i)}+w_{13}^{[1]}x_3^{(i)}+b_1^{[1]}\\ w_{21}^{[1]}x_1^{(i)}+w_{22}^{[1]}x_2^{(i)}+w_{23}^{[1]}x_3^{(i)}+b_2^{[1]}\\ w_{31}^{[1]}x_1^{(i)}+w_{32}^{[1]}x_2^{(i)}+w_{33}^{[1]}x_3^{(i)}+b_3^{[1]}\\ w_{41}^{[1]}x_1^{(i)}+w_{42}^{[1]}x_2^{(i)}+w_{43}^{[1]}x_3^{(i)}+b_4^{[1]}\end{bmatrix}$$

$$a^{[1](i)}=\begin{bmatrix}a_1^{[1](i)}\\ a_2^{[1](i)}\\ a_3^{[1](i)}\\ a_4^{[1](i)}\end{bmatrix}=\sigma\begin{bmatrix}z_1^{[1](i)}\\ z_2^{[1](i)}\\ z_3^{[1](i)}\\ z_4^{[1](i)}\end{bmatrix}=\sigma(z^{[1](i)})$$

$$\text{Layer2}\quad z^{[2](i)}=\left[z_1^{[2](i)}\right]=\left[w_1^{[2]}\right]a^{[1](i)}+b^{[2]}=\begin{bmatrix}w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} & w_{14}^{[2]}\end{bmatrix}\begin{bmatrix}a_1^{[1](i)}\\ a_2^{[1](i)}\\ a_3^{[1](i)}\\ a_4^{[1](i)}\end{bmatrix}+\left[b_1^{[2]}\right]=w_{11}^{[2]}a_1^{[1](i)}+w_{12}^{[2]}a_2^{[1](i)}+w_{13}^{[2]}a_3^{[1](i)}+w_{14}^{[2]}a_4^{[1](i)}+b_1^{[2]}$$

$$\hat{y}^{(i)}=a^{[2](i)}=\sigma(z^{[2](i)})$$

图 3.28　两层网络的正向传播计算过程

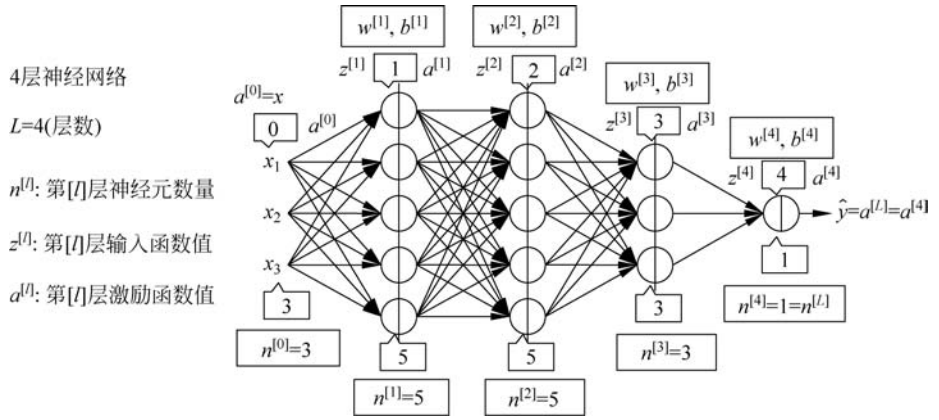


图 3.29　四层神经网络的结构化表示与参数提示

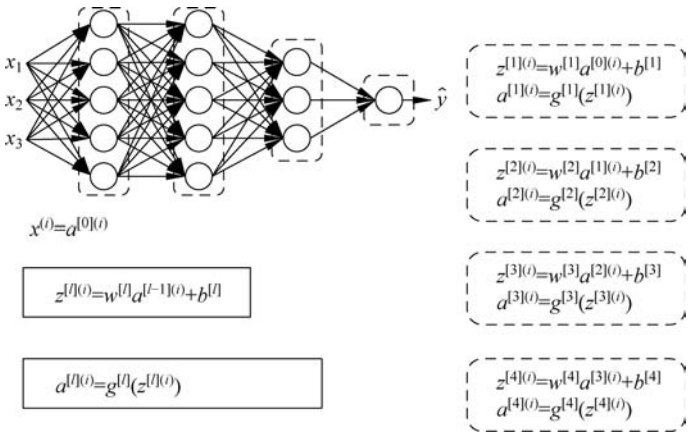


图 3.30　四层网络的正向传播计算过程

3.12　反向传播

正向传播结束后，就可以根据输出值 $\hat{y}$ 与真实值 y 来计算损失函数，根据损失函数进行反向传播。

反向传播(Backward Propagation)，又称后向传播，基本原理是根据损失函数反方向计算每一层的 z、a、w 和 b 的偏导数，得到 w 和 b 在每一层各个参数方向的梯度，用各层的梯度 $\mathrm{d}w$ 和 $\mathrm{d}b$ 逐层更新各层的参数 w 和 b，从而得到由新的 w 和 b 构成的新的网络。

反向传播与正向传播是密切联系的两个过程，正向传播是反向传播的计算基础，正向传播与反向传播迭代进行下去，直至找到最优的 w 和 b。

正向传播与反向传播的逻辑对比，如图 3.31 所示。

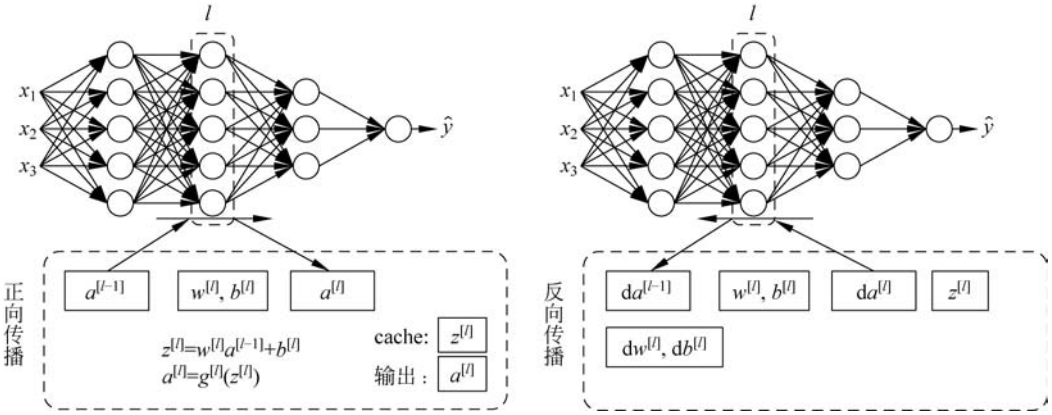


图 3.31　正向传播逻辑与反向传播逻辑对比

以第 ℓ 层的正向传播为例，其计算过程如下。

(1) 接受来自第 $\ell-1$ 层的 $a^{[l-1]}$ 作为输入。

(2) 根据 $a^{[l-1]}$、$w^{[l]}$ 和 $b^{[l]}$ 计算 $a^{[l]}$。

(3) 输出 $a^{[l]}$ 到 $\ell+1$ 层，同时缓存本层的 $w^{[l]}$、$b^{[l]}$、$z^{[l]}$ 和 $a^{[l]}$，缓存参数将用于反向传播。

第 ℓ 层的反向传播计算过程如下。

(1) 接受来自第 $\ell+1$ 层的 $\mathrm{d}a^{[l]}$ 作为输入。

(2) 根据本层缓存的 $w^{[l]}$、$b^{[l]}$、$z^{[l]}$ 和 $a^{[l]}$，计算 $\mathrm{d}w^{[l]}$、$\mathrm{d}b^{[l]}$ 和 $\mathrm{d}a^{[l-1]}$。

(3) 输出 $\mathrm{d}a^{[l-1]}$ 到第 $\ell-1$ 层，同时缓存本层的 $\mathrm{d}w^{[l]}$ 和 $\mathrm{d}b^{[l]}$，用于反向传播结束时，更新本层的参数 $w^{[l]}$ 和 $b^{[l]}$。

用模块图的方式表示第 ℓ 层的正向传播与反向传播更为简洁，如图 3.32 所示。

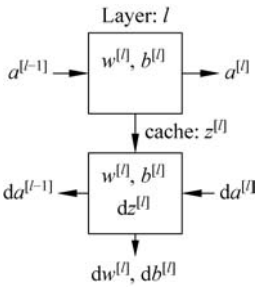


图 3.32　正向传播与反向传播模块

现在给出神经网络正向传播与反向传播的完整迭代计算过程，如图 3.33 所示。假定输出层采用 Sigmoid 函数，其他各层

激励函数为 ReLU，采用交叉熵作损失函数。

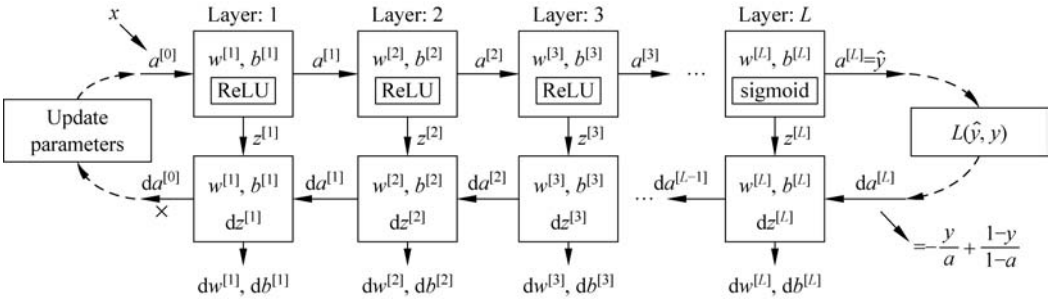


图 3.33　神经网络的正向与反向传播迭代过程

如图 3.33 所示，为了让第 1 层的正向输入与反向输出与其他各层的表示一致，用 $a^{[0]}$ 替代表示输入层的特征向量 $\boldsymbol{x}$。

神经网络反向传播的实质，是为了完成神经网络的一次梯度下降，正向传播与反向传播合在一起，就是神经网络模型的训练过程。其完整迭代计算可归纳为如下四个步骤。

(1) 正向迭代：从第 1 层到第 L 层，根据参数和激励函数，依次计算各层的输出，同时缓存相关的参数值，直至得到输出层的 $\hat{y}$，正向传播结束。

(2) 计算损失函数 L。

(3) 反向迭代：从损失函数开始，反向逐层计算各层的 $\mathrm{d}w$、$\mathrm{d}b$、$\mathrm{d}z$ 和 $\mathrm{d}a$，同时缓存各层的梯度值 $\mathrm{d}w$ 和 $\mathrm{d}b$，直至完成第 1 层的反向计算。值得注意的是，第 1 层的输出 $\mathrm{d}a^{[0]}$ 没有实际意义，因为输入层不存在激励函数。用 $a^{[0]}$ 表示特征向量 $\boldsymbol{x}$ 是符号表达的需要。

(4) 更新参数：用各层缓存的梯度值 $\mathrm{d}w$ 和 $\mathrm{d}b$，更新各层的参数 w 和 b。

重复上述步骤(1)～(4)，反复迭代计算，直至损失函数达到理想的目标值，此时各层的参数 w 和 b，即为神经网络的求解结果。

结合图 3.33 所示的传播过程，这里顺便理清两个概念。

(1) 关于迭代和梯度下降。从正向传播开始，经过损失函数计算、反向传播到参数更新结束，称为一次梯度下降或一次迭代。

(2) 关于一代(epoch)训练。训练集中的所有样本均参与了一次梯度下降，则称模型完成了一代训练。在神经网络模型训练时指定的参数 epochs，表示需要模型训练多少代。

各种深度学习框架，如 TensorFlow、Keras 等，只需要构建和定义正向传播过程，反向传播的计算过程是在函数内部自动完成的，这为神经网络模型的开发带来极大的便利。

3.13　偏差与方差

当讨论预测模型时，预测误差可以分为两个角度进行讨论："偏差"引起的误差和"方差"引起的误差。

偏差(bias)是模型的预期预测值(或平均预测值)与正确值之间的误差。偏差越大，

越偏离真实数据，偏差可用于度量模型在数据集上的拟合程度。

方差(variance)是指模型预测结果的离散程度，方差可用于度量模型在不同数据集上的稳定性，即模型在不同数据集上的泛化能力。

在机器学习领域，偏差与方差可用于指导模型的训练与优化。以一个二维数据集的分类为例，如图 3.34 所示，给出了训练集上三种不同模型的表现。

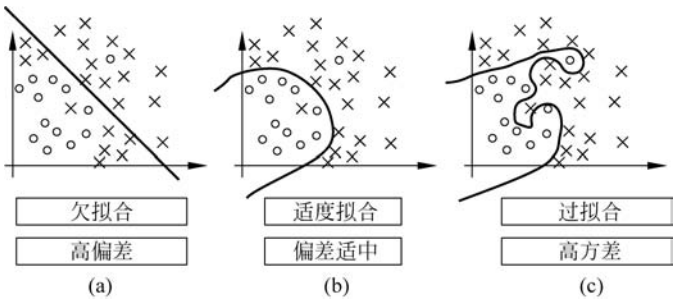


图 3.34　训练集上三种不同模型的表现

图 3.34(a)用直线模型分隔数据集过于简单，导致有较多的×被划为○类，直线无法准确表达数据关系导致偏差较高，这种情况也称为模型欠拟合。

图 3.34(c)用复杂曲线将两种数据完美分隔，与图 3.34(a)相比是另一个极端，这种情况称模型为低偏差，或者模型过拟合。因为一旦将这种模型部署于不同的数据集，极有可能导致高方差，即模型的泛化能力低。

图 3.34(b)是一个适中的方案，用相对简单的曲线分隔数据集，既照顾了数据之间的非线性关系，又没有刻意追求低偏差，没有为了几个孤立数据点或者噪声点刻意过度增加模型的复杂性，这种情况称作模型的适度拟合。

偏差与方差这两个指标可以指引模型优化的方向。如图 3.35 所示，根据一个模型实例在训练集与验证集上的误差表现，给出了模型的整体判断。

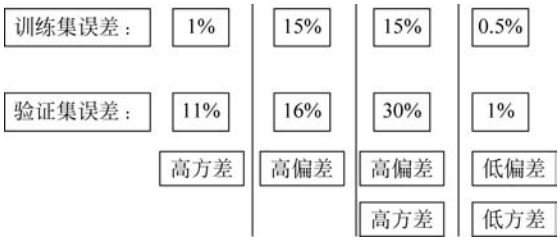


图 3.35　模型的偏差与方差判断

参照图 3.35 的数据，共有以下四个结论。

(1) 验证集的误差 11%显著高于训练集的误差 1%，证明模型存在高方差，模型过拟合，泛化能力不强。

(2) 验证集的误差 16%与训练集的误差 15%极其接近，证明模型的泛化能力较好，在两种数据集上的表现趋于一致，具有稳定性，但是偏差过高，模型的准确率不好。

(3) 训练集误差 15%，验证集误差 30%，证明模型不但偏差高，而且方差也高，图 3.36

给出了一个高偏差与高方差的直观理解。

(4) 训练集误差 0.5%,验证集误差 1%,这是比较理想的情况,偏差与方差均处于低水平。

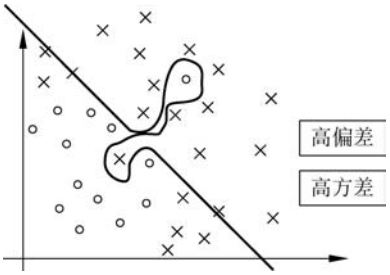


图 3.36 对高偏差与高方差模型的直观理解

理解了偏差与方差在模型优化方向的指导作用,下面给出一些具体的优化指导措施,如图 3.37 所示。

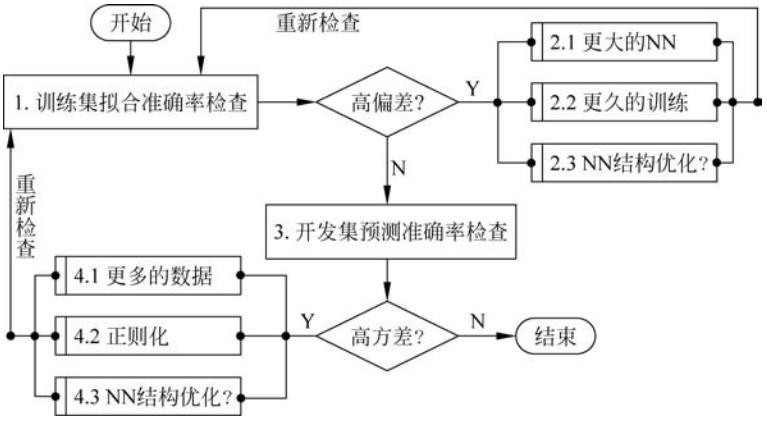


图 3.37 模型的偏差与方差优化流程

模型可按照如下步骤优化。

(1) 首先对模型在训练集上的准确率做出检测,判断是否为高偏差。

(2) 如果是高偏差,可以从以下三个方面对模型做出调整。

① 采用更大的神经网络,例如,增加层数或者增加某些层的神经元数量。

② 采用更多的迭代步数,增加模型训练时间。

③ 对神经网络的结构做出优化,例如,采用新算法等。

返回第(1)步,重新检查准确率,直至偏差降低到可以接受的程度为止。

(3) 如果是低偏差,则在开发集(验证集)上做准确率检测,判断方差高低。

(4) 如果是低方差,则模型优化结束。

(5) 如果第(3)步给出的是高方差,则从以下三个方面优化模型。

① 采用更大的数据集,增加模型的鲁棒性。

② 采用正则化方法,降低过拟合,增强模型泛化能力。

③ 修改模型结构或采用新模型，例如，采用新算法等。

(6) 返回第(1)步，重新从训练集开始逐项检查，直至优化结束。

3.14　正则化

神经网络由于密集的连接关系、深度的层数设计和众多的 w、b 参数，很容易导致过拟合现象。

正则化(Regularization)是为了防止模型过拟合而引入额外信息，对模型原有逻辑进行外部干预和修正，从而提高模型的泛化能力。

L2 正则化和 Dropout 正则化，是神经网络常用的正则化方法。

求解模型参数 w、b 的过程是以求解损失函数的最小值为目标导向的，所以改善损失函数求解逻辑，可以降低模型的过拟合。

L2 正则化是在损失函数上额外增加一个关于参数 w 的 L2 范数项(惩罚项)，这个 L2 范数项可以起到衰减权重的作用，弱化某些参数，从而降低过拟合，如公式(3.10)所示。

$$J(\boldsymbol{w},b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})+\frac{\lambda}{2m}\|\boldsymbol{w}\|_2^2 \tag{3.10}$$

其中，$\|\boldsymbol{w}\|_2^2=\sum_{j=1}^{n}\boldsymbol{w}_j^2=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$ 是参数 $\boldsymbol{w}$ 的 L2 范数的平方，λ 是超参数。

由于神经网络是分层的，每一层都有自己的权重矩阵 $\boldsymbol{w}$，所以第 ℓ 层的 L2 范数的平方可以表示为公式(3.11)。

$$\|\boldsymbol{w}^{[l]}\|_2^2=\sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(\boldsymbol{w}_{ij}^{[l]})^2,\quad \text{其中，}\boldsymbol{w}^{[l]}\text{ 的维度为}(n^{[l]},n^{[l-1]}) \tag{3.11}$$

公式 3.11 实质上是矩阵范数的平方，又称 Frobenius 范数平方，简称 F-范数平方，记做 $\|\cdot\|_F^2$ 或者 $\|\cdot\|^2$，而不是 $\|\cdot\|_2^2$。

神经网络的 L2 正则化可以表示为公式(3.12)。

$$J(\boldsymbol{w}^{[1]},b^{[1]},\boldsymbol{w}^{[2]},b^{[2]},\cdots,\boldsymbol{w}^{[L]},b^{[L]})=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})+\frac{\lambda}{2m}\sum_{l=1}^{L}\|\boldsymbol{w}^{[l]}\|^2 \tag{3.12}$$

为什么正则化可以降低过拟合？可以从如图 3.38 所示的反向传播梯度计算规律中窥见一斑。

$$\text{梯度计算：}\mathrm{d}\boldsymbol{w}^{[l]}=\left[\frac{\partial J}{\partial \boldsymbol{w}^{[l]}}\text{无正则项时的反向梯度}\right]+\frac{\lambda}{m}\boldsymbol{w}^{[l]}$$

$$\begin{aligned}\text{参数更新：}\boldsymbol{w}^{[l]}&=\boldsymbol{w}^{[l]}-\alpha\,\mathrm{d}\boldsymbol{w}^{[l]}\\&=\boldsymbol{w}^{[l]}-\alpha\left[\frac{\partial J}{\partial \boldsymbol{w}^{[l]}}\text{无正则项时的反向梯度}\right]-\alpha\frac{\lambda}{m}\boldsymbol{w}^{[l]}\\&=\boldsymbol{w}^{[l]}-\alpha\frac{\lambda}{m}\boldsymbol{w}^{[l]}-\alpha\left[\frac{\partial J}{\partial \boldsymbol{w}^{[l]}}\text{无正则项时的反向梯度}\right]\\&=\left(1-\alpha\frac{\lambda}{m}\right)\boldsymbol{w}^{[l]}-\alpha\left[\frac{\partial J}{\partial \boldsymbol{w}^{[l]}}\text{无正则项时的反向梯度}\right]\end{aligned}$$

图 3.38　加入正则项后的权重衰减规律

如图 3.38 所示，梯度计算需要考虑 F-范数正则项，参数更新时，权重 w 前面的系数是一个小于 1 的整数，这意味着随着迭代次数增加，参数更新呈衰减趋势，所以 F-范数又名权重衰减范数。这种衰减在一定程度上弱化了参数的作用，不排除参数衰减为 0 的情况出现，从而间接降低了模型复杂度，有助于弱化过拟合问题。

另一种预防神经网络模型过拟合的有效方法是 Dropout 正则化，其基本原理是对神经网络某些层的神经元施加一个舍弃概率，使得每次模型训练，只用其中一部分神经元来完成，如图 3.39 所示。

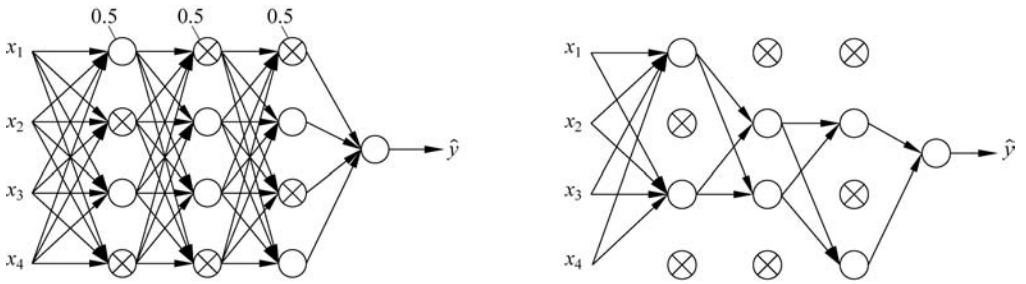


图 3.39　Dropout 正则化后网络结构对比

如图 3.39(a)所示代表 Dropout 正则化之前的网络结构，图 3.39(b)表示一次随机 Dropout 之后的网络结构。假定图 3.39(a)中的三个隐藏层均施加 0.5 的舍弃概率，图中标×的单元为舍弃的神经元，则实质上用于模型训练的神经元如图 3.39(b)所示，显然这是一个比原结构简化的网络模型。

值得注意的是，图 3.39 只是一个示意图，Dropout 正则化，每次迭代中舍弃的神经元是随机选择的，这种机制一方面使得模型得到随机简化，另一方面又使得不同的神经元得到随机重视，其效果是有助于降低原模型的过拟合，使得最后得到的模型的鲁棒性和泛化能力更好。

不难看出，Dropout 正则化与 F-范数正则化在预防过拟合方面具有类似作用。

3.15　Mini-Batch 梯度下降

梯度下降是贯穿神经网络的算法灵魂，根据样本数据参加训练的方式，可分为如下三种模式的梯度下降。

(1) 批量梯度下降(Batch Gradient Descent，BGD)：全部训练样本一起完成一次正向与反向传播，即一次梯度下降。这种方法的优点是所有样本共同决定梯度下降的方向，可以用最少的迭代步数逼近最优值，缺点是一次性装入过多样本，对内存的需求很大，对计算力要求很高。

(2) 随机梯度下降(Stochastic Gradient Descent，SGD)：一次用一个样本完成一次正向与反向传播。这种方法的优点是单个样本决定梯度的下降方向，适合在线学习，计算速

度快，内存需求小。但是单个样本决定下降的梯度方向，过于依赖样本的数据质量，容易导致下降的方向飘忽不定，需要更多的迭代，而且难以逼近最优值。

(3) 小批量梯度下降(Mini-Batch Gradient Descent，MBGD)：是对上述两种梯度下降方法的改进，将整个训练集随机划分为若干不同的组，每次输入一组数据，一次用一组数据完成一次正向与反向传播，既可以保障梯度下降的方向，又可以降低对内存与算力的过高需求。

图 3.40 给出了一个 Mini-Batch 梯度下降的数据集划分方法。

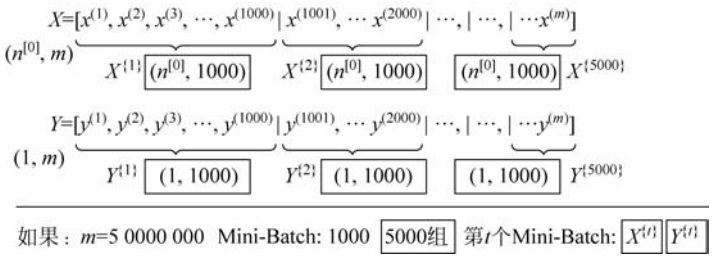


图 3.40　Mini-Batch 划分方法与符号表示示例

假定样本数据集的规模为 500 万，单个 Mini-Batch 的大小为 1000，则整个样本集可以分为 5000 组。每一组的特征矩阵用 $\boldsymbol{X}^{\{t\}}$ 表示，标签矩阵用 $\boldsymbol{Y}^{\{t\}}$ 表示。图 3.40 中标出了每一组的映射关系与维度关系。

必须强调的是，训练集的全部样本通过网络的一次训练称作一代(epoch)训练，一组 Mini-Batch 通过网络的训练是一次迭代(一次梯度下降)。一般来说，网络需要进行多代(epoch)训练，进行每一代训练之前，都要重新随机划分 Mini-Batch。Mini-Batch 的大小是固定不变的，但是每一个 Mini-Batch 的样本是随机确定的，这样做的目的是为了保证网络模型更好地适应数据随机分布带来的影响。如图 3.41 所示，在特征矩阵随机洗牌的同时，要保证标签矩阵也是同步洗牌的。

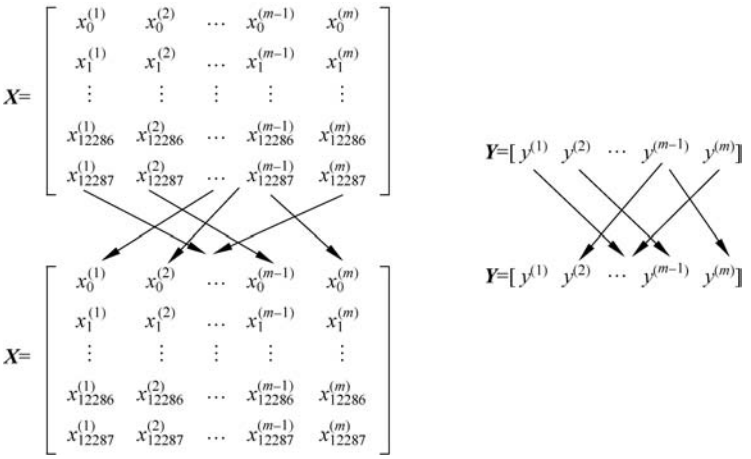


图 3.41　整个样本集的随机洗牌

图 3.42 给出了一个基于 Mini-Batch 的梯度下降算法的案例描述。

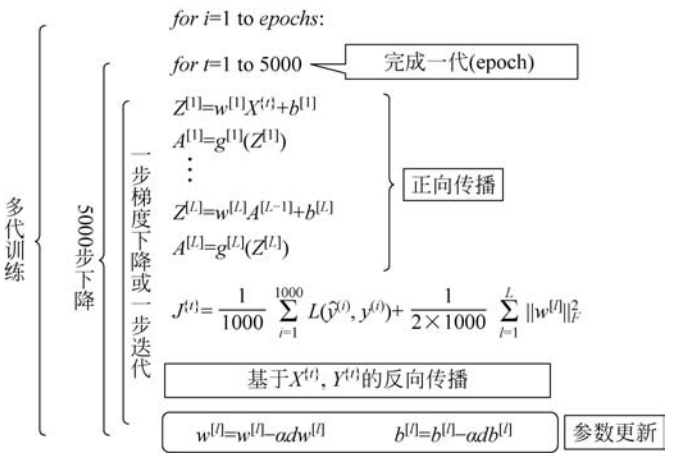


图 3.42 基于 Mini-Batch 的梯度下降算法描述

将 Mini-Batch 大小设定为 1000，只是为了问题描述的便利，实践中一般取 2 的整数次幂，例如 32、64、128 等。显然，Mini-Batch 取值为 1，相当于随机梯度下降，Mini-Batch 取值为 m（m 是样本集的大小），相当于批量梯度下降。

3.16 优化算法

本节介绍梯度下降的四种优化算法：Momentum 梯度下降法，RMSprop 梯度下降法，Adam 梯度下降法和学习率衰减法。前面三种都是基于移动指数加权平均的思想，平滑梯度的计算，进而加快模型的收敛速度。RMSprop 可以看作是对 Momentum 的改进，Adam 是对 Momentum 和 RMSprop 的综合改进。

Momentum 梯度下降法的算法描述如图 3.43 所示。

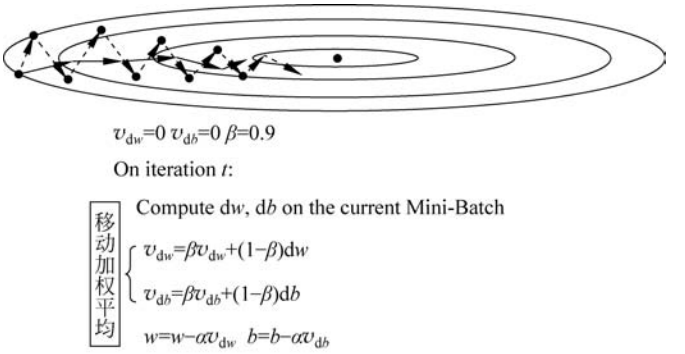
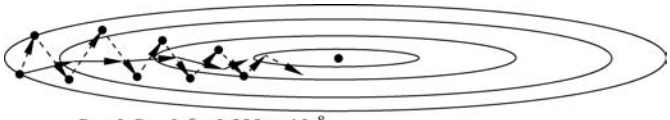


图 3.43 Momentum 梯度下降法

如图 3.43 所示，Momentum 梯度下降法是对 Mini-Batch 梯度下降法的一种改进，在 Mini-Batch 完成单步梯度计算后，没有立即更新参数，而是用移动指数加权平均的思想计算当前梯度的移动平均值 v_{dw} 和 v_{db}，然后用 v_{dw} 和 v_{db} 去更新 w 和 b。图 3.43 上方的

平面图展示了这样做的直观效果，图中虚线表示不做移动平均时的梯度下降路径，实线表示 Momentum 梯度下降路径，显然后者纵向波动更小，横向收敛速度更快。

RMSprop 是另一种能够加速梯度下降的算法，其描述与直观理解如图 3.44 所示。



$S_{dw}=0\ S_{db}=0\ \beta_2=0.999\ \varepsilon=10^{-8}$

On iteration t:

Compute dw, db on the current Mini-Batch

移动加权平均

$S_{dw}=\beta_2 S_{dw}+(1-\beta_2)dw^2$

$S_{db}=\beta_2 S_{db}+(1-\beta_2)db^2$

$w=w-\alpha\dfrac{dw}{\sqrt{S_{dw}}+\varepsilon}$　　$b=b-\alpha\dfrac{db}{\sqrt{S_{db}}+\varepsilon}$

图 3.44　RMSprop 梯度下降法

RMSprop 梯度下降法亦采用移动指数加权平均的思想，平滑梯度纵向的波动，加快横向的收敛速度。与 Momentum 不同的是，其移动平均的计算采用 dw^2 和 db^2，参数更新的策略也发生变化，分别用 dw 和 db 去除以各自的移动均方根，为了避免分母为 0，分母增加一个 ε 调节项。

Adam 梯度下降是对 Momentum 和 RMSprop 两种方法的综合改进，兼顾了二者的优点，如图 3.45 所示。

$v_{dw}=0, v_{db}=0, S_{dw}=0, S_{db}=0, \beta_1=0.9, \beta_2=0.999$

On iteration t:

Compute dw, db on the current Mini-Batch

$v_{dw}=\beta_1 v_{dw}+(1-\beta_1)dw$　　$v_{db}=\beta_1 v_{db}+(1-\beta_1)db$　　Momentum

$S_{dw}=\beta_2 S_{dw}+(1-\beta_2)dw^2$　　$S_{db}=\beta_2 S_{db}+(1-\beta_2)db^2$　　RMSprop

$v_{dw}^{corrected}=v_{dw}/(1-\beta_1^t)$　　$v_{db}^{corrected}=v_{db}/(1-\beta_1^t)$

$S_{dw}^{corrected}=S_{dw}/(1-\beta_2^t)$　　$S_{db}^{corrected}=S_{db}/(1-\beta_2^t)$　　移动平均修正

$w=w-\alpha\dfrac{v_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}}+\varepsilon}$　　$b=b-\alpha\dfrac{v_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\varepsilon}$　　参数更新

图 3.45　Adam 梯度下降法

Adam 方法同时用 Momentum 和 RMSprop 方法的移动平均去平滑各个方向的梯度，并进行移动平均修正，在参数更新阶段，分子用 Momentum 的移动均值，分母用 RMSprop 均方根，实践中 Adam 的效果往往优于前两种方法，但是 Adam 算法的计算量也要大一些。

学习率衰减也是一种常用的优化算法，从直观上理解，越是逼近目标点，参数的更新幅度应该越小，图 3.46 演示了一种基于 epoch 的学习率衰减方法。

图 3.46 左下角给出了学习率 α 的衰减计算公式，右图给出了一个衰减实例。学习率

的衰减有各种方法可用，如公式(3.13)和公式(3.14)所示，其中，α_0 表示初始学习率，k 为一个常数。

$$\alpha = 0.95^{\text{epochNum}}\alpha_0 \tag{3.13}$$

$$\alpha = \frac{k}{\sqrt{\text{epochNum}}}\alpha_0 \tag{3.14}$$

请结合视频讲解，深入学习优化算法的设计原理。特别提醒，读者在阅读中遇到的任何困惑，随时都可以查看视频讲解，文字与视频，二者总是相互补充，相得益彰。

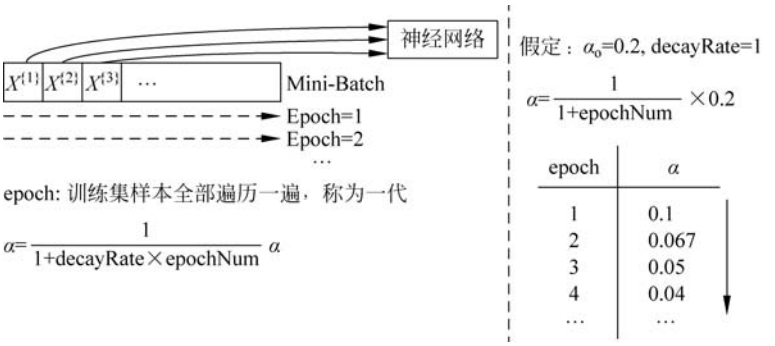


图 3.46 基于 epoch 的学习率衰减方法

3.17 参数与超参数

神经网络有两类参数，一类是分布在各层的参数 w 和 b，这是需要求解的模型参数；另一类是用来优化模型或设定模型规模的参数，称为超参数，如表 3.4 所示。

表 3.4 常见超参数

参数名称	参数说明
学习率 α	用于调整梯度下降的步幅
各层神经元数量 Units	调整网络的宽度
Mini-Batch 大小	单步迭代的效果
网络层数 Layers	调整网络的深度
过滤器参数	过滤器数量、尺寸、步长及是否填充边距等
正则化方法	是否采用正则化，采用何种正则化方法
训练代数 Epochs	决定梯度下降的迭代步数和训练持久程度
各层激励函数	影响模型各层之间的联系
优化算法	根据问题需要比较优化算法的效果
损失函数	评估模型误差的函数

学习率是最为重要的超参数，对模型的影响很大，因此，模型的优化与超参数调整往往是从学习率开始的。但是实践中往往需要多个参数同时搜索，下面介绍两种常见的参数搜索策略。

(1) 参数随机取值策略，如图 3.47 所示。

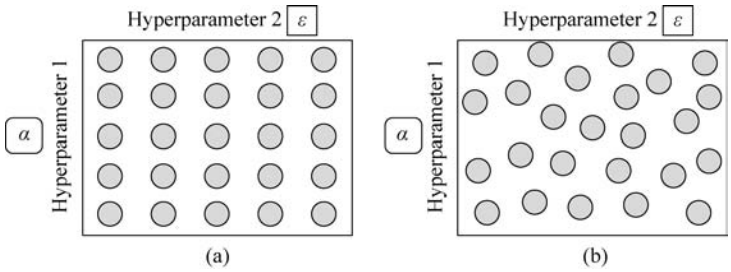


图 3.47　参数随机取值策略

假定需要同时调整的参数为 α 和 ε，显然 α 的重要性远远高于 ε，此时若采用均匀网格取值法，如图 3.47(a)所示，对寻找最佳 α 参数是不利的，因为 α 只取了 5 个不同值，而相对不重要的 ε 却取到了 25 个值。如图 3.47(b)所示的随机策略，同时兼顾了两个参数的取值范围，效果更好一些。

(2) 由粗糙到精细策略，如图 3.48 所示。

最优参数的选择往往难以一步到位，如果发现某个参数值的效果较好，如图 3.48 中的粗线参数点所示，可以围绕这个参数点画出一个范围，在这个范围内取更多的值进行检查，如阴影矩形中的实心原点所示。

在指定范围内随机取值时，应该注意取值的分布合理性。例如，假定 α 需要在[0.0001,1]随机取 10 个值，如果简单地采用随机数取值法，可能会有 90%的点落在[0.1,1]的 10 倍幅度区间，而忽略了[0.0001,0.1]的 1000 倍幅度区间，如图 3.49 所示。

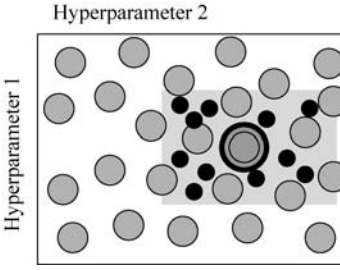


图 3.48　由粗糙到精细策略

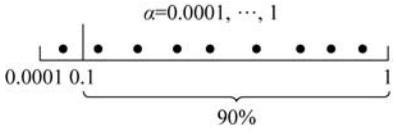


图 3.49　随机取值的不合理性

采用对数法修正取值范围，是一种更好的随机方法，如图 3.50 所示。

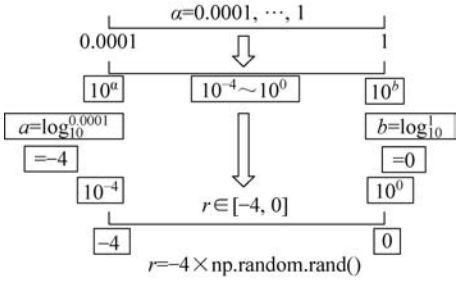


图 3.50　更好的随机取值方法

通过对数界定随机取值区间，将取值范围的起点与终点映射到同一数量级，可以保证随机取值的合理分布。如图 3.50 所示，先用对数方法将取值区间变换为[−4,0]，在此区间随机取值后，再用指数法还原为原有取值区间，可以避免如图 3.49 所示的不合理取值分布。

3.18 Softmax 回归

逻辑回归解决的是二分类问题，Softmax 回归解决的是多分类问题。对于二分类问题，神经网络的输出层一般用 Sigmoid 函数(逻辑回归)作为激励函数，对于多分类问题，神经网络输出层用 Softmax 函数(Softmax 回归)作为激励函数。

假定第 L 层为网络的输出层，包含四个神经元，采用 Softmax 作四分类器。L 层四个神经元的输入函数值 $Z^{[L]}$ 已经获得，则 Softmax 回归的输出结果与计算过程如图 3.51 所示。

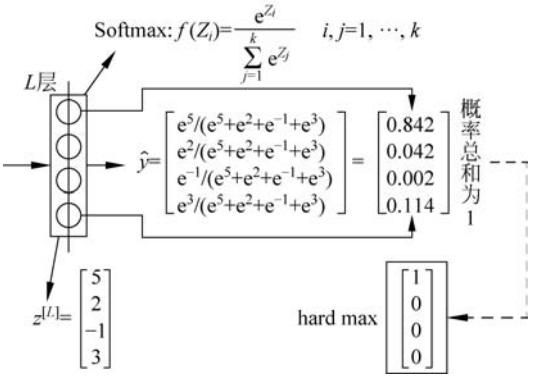


图 3.51 Softmax 回归计算过程示例

与逻辑回归的 Sigmoid 函数输出一个概率值不同，Softmax 回归输出的是一组概率值，每个概率值对应一个类别，所有概率值之和为 1。顾名思义，Softmax 是一种软编码，还需要将其转换到硬编码(hard max)形式，又称 One-Hot 编码，即将概率值最大的类别设定为 1，其他类别均设为 0。

3.19 VGG-16 卷积网络

VGG-16 是牛津大学计算机视觉组(Visual Geometry Group Network，VGG)研发的经典卷积网络(Simonyan and Zisserman，2014)，结构优美简洁，由 13 个卷积层和 3 个全链接层组成，共 16 层。不包括池化层，因为池化层不需要学习参数。卷积层过滤器大小为(3,3)，池化层过滤器大小为(2,2)，结构定义如图 3.52 所示。

VGG-16 有一个显著特点：随着网络层数增长，特征图尺寸减半，过滤器数量翻倍。这种思路，也为其他模型的设计与演化提供了借鉴与灵感。

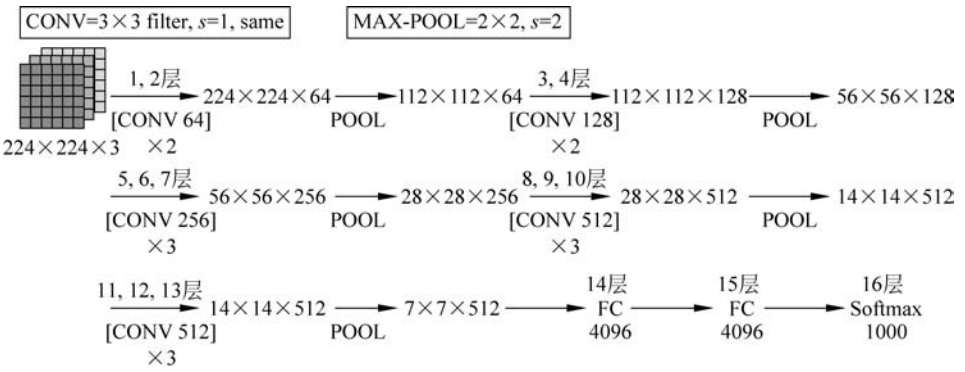


图 3.52　VGG-16 结构定义

执行程序段 P3.18，可借助 Keras 框架，完成如图 3.52 所示的 VGG-16 模型的定义。

```
P3.18  # 标准 VGG-16 模型的定义
075    from keras import Sequential
076    from keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten, Dropout
077    model = Sequential(name = 'VGG16')
       # BLOCK 1
078    model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block1_conv1', input_shape = (224, 224, 3)))
079    model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block1_conv2'))
080    model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), name = 'block1_pool'))
       # BLOCK2
081    model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block2_conv1'))
082    model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block2_conv2'))
083    model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), name = 'block2_pool'))
       # BLOCK3
084    model.add(Conv2D(filters = 256, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block3_conv1'))
085    model.add(Conv2D(filters = 256, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block3_conv2'))
086    model.add(Conv2D(filters = 256, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block3_conv3'))
087    model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), name = 'block3_pool'))
       # BLOCK4
088    model.add(Conv2D(filters = 512, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block4_conv1'))
089    model.add(Conv2D(filters = 512, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block4_conv2'))
```

```
090     model.add(Conv2D(filters = 512, kernel_size = (3, 3), activation = 'relu',
                         padding = 'same', name = 'block4_conv3'))
091     model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), name = 'block4_pool'))
        #BLOCK5
092     model.add(Conv2D(filters = 512, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block5_conv1'))
093     model.add(Conv2D(filters = 512, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block5_conv2'))
094     model.add(Conv2D(filters = 512, kernel_size = (3, 3), activation = 'relu',
                        padding = 'same', name = 'block5_conv3'))
095     model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), name = 'block5_pool'))
096     model.add(Flatten())
        #FC1
097     model.add(Dense(4096, activation = 'relu', name = 'fc1'))
098     model.add(Dropout(0.5))
        #FC2
099     model.add(Dense(4096, activation = 'relu', name = 'fc2'))
100     model.add(Dropout(0.5))
        #Softmax
101     model.add(Dense(1108, activation = 'softmax', name = 'prediction'))
102     model.summary()
```

模型结构摘要显示，VGG-16 模型包含 138 800 020 个需要学习训练的参数，计算量较大。如果使用标准的 VGG-16 模型，可以通过 Keras 中内置的 VGG-16 函数直接创建，或者采用基于 ImageNet 的预训练模型实现迁移学习。

3.20 ResNet 卷积网络

ResNet 是微软研究院的何恺明等人于 2015 年在其 *Deep Residual Learning for Image Recognition* 论文中提出的一种深度卷积学习模型(He，Zhang，et al.，2016)。通过使用残差块(Residual Unit)成功训练 152 层深的神经网络，在 ImageNet 2015 比赛中获得冠军，取得 3.57%的 top5 错误率，而且参数量较低，性能突出。

长期以来，随着层数的增加，梯度消失或梯度爆炸一直是伴随深度神经网络的一个难题，为此，何恺明等人定义了残差块，重构了神经网络的学习流程，如图 3.53 所示。

图 3.53 左侧虚线框表示的是一个残差块结构，右侧公式分步演示了其计算过程。ResNet 之前的神经网络，每一层只与其上一层和下一层有关系，而残差块则打破了这一逻辑。

如虚线框内所示，以 l 层、$l+1$ 层、$l+2$ 层这三层为例，l 层的输出 $a^{[l]}$ 是 $l+1$ 层的输入，即在传统神经网络模型中，l 层的输出信息只能通过 $l+1$ 层处理后向前传递，残差块要做的是增加一个快捷连接(跳连)，l 层的输出除了传递给 $l+1$ 层进行学习，也同时跳过

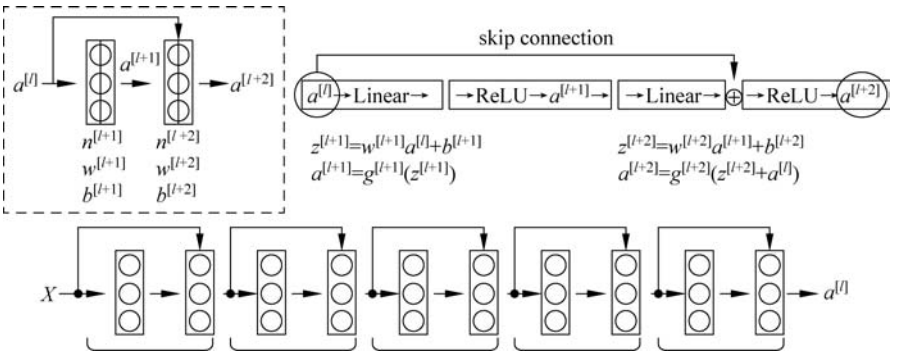


图 3.53　残差块的学习流程

$l+1$ 层传递给 $l+2$ 层(甚至更远的层),注意观察 $a^{[l]}$ 在 $l+2$ 层中的插入位置,是在 $l+2$ 层的输入函数之后,激励函数之前。也就是说,对于 $l+2$ 层的激励函数,是用 $a^{[l]}+z^{[l+2]}$ 作为输入。对于 $l+2$ 层来说,从 l 层直接复制过来的 $a^{[l]}$ 不需要经过学习训练,只有 $z^{[l+2]}$ 是学习参数得到的,即:$l+2$ 层的学习任务,有一部分是由 l 层完成的,剩余的是由 $l+1$ 层和 $l+2$ 层完成的,所以把这种网络结构称为残差块。

当然,残差块跳连(或称旁连、直连),不一定只间隔一层,也可以间隔跳连多层。

传统的卷积层或全连接层在信息传递时,或多或少会存在信息丢失、损耗等问题,残差块模型相当于一定程度弥补了信息损耗,不增加额外的参数和计算量,却可以显著提高模型的训练速度和训练效果,并且当模型的层数加深时,这个简单的结构能够很好地平滑梯度消失问题。图 3.53 展示了由若干个残差块构成的网络结构。

图 3.54 为残差网络的作者在论文中设计的实验对比。实验中以 VGGNet-19、34 层深的普通卷积神经网络和 34 层深的 ResNet 做对比,可以看到普通的卷积神经网络和 ResNet 的最大区别在于,ResNet 有很多旁路的分支将前面层的输入直连到后面的层,使得后面的层只学习残差。

实验证明,ResNet 34 层网络的效果比其他两种结构更好,而且收敛速度更快。

ResNet 34 的模型结构图中,有实线和虚线两种跳连方式,实线跳连前后的通道数相同;虚线跳连前后的通道数不一致,前面的输入需要做卷积变换才能参与到后面的计算中。模型在 ImageNet 上的训练效果如图 3.55 所示。细曲线表示训练误差,粗曲线表示验证误差。图 3.55(a)是 18 层和 34 层的普通网络对比,图 3.55(b)是 18 层和 34 层的 ResNet 对比。

论文作者在前述工作基础上,提出了 50 层、101 层、152 层的 ResNet,不仅没有出现梯度消失问题,错误率也大大降低,同时计算复杂度也保持在很低的程度。相关模型参数如表 3.5 所示。

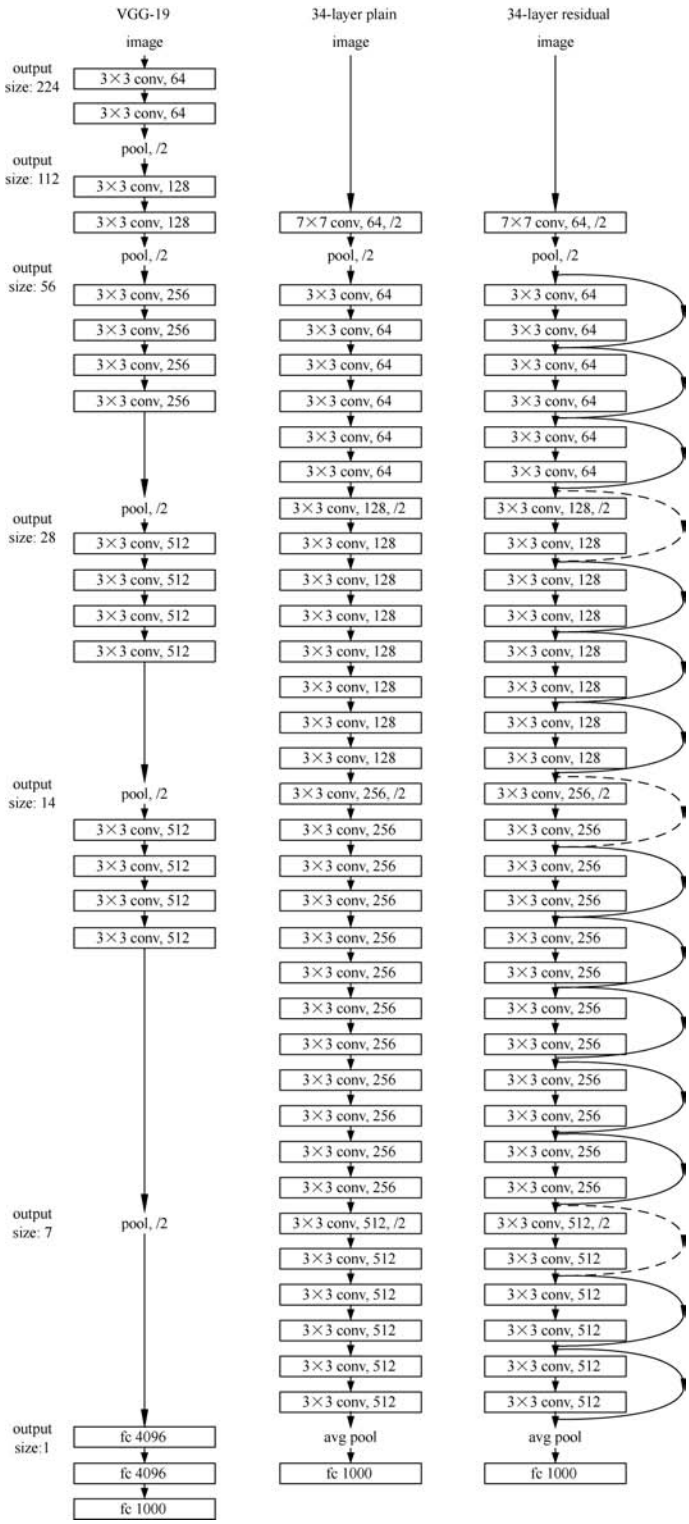


图 3.54　VGG-19、34 层普通卷积网络与 ResNet 34 对比

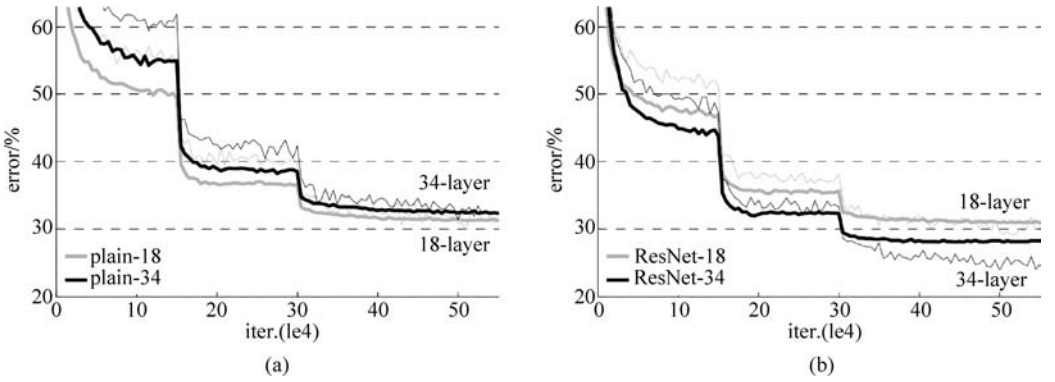


图 3.55　两种深度模型在 ImageNet 上的效果对比

表 3.5　ResNet 模型结构参数

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

表中最后一行表示模型的计算量大小，FLOPs(floating point operations)表示模型做一次前向传播需要完成的浮点运算次数。这里需要注意区别 FLOPS(floating point operations per second)，FLOPS 意指每秒浮点运算次数，是用来衡量硬件计算性能的指标。从形式上看，前者以小写 s 结尾，后者以大写 S 结尾。显然，随着网络层数的增加，ResNet 的计算量的增长并不显著。

关于 ResNet 模型的深入解析，请参见本节视频讲解。

3.21　1×1 卷积

所谓 1×1 卷积，是指卷积核大小为 1×1 的卷积运算。1×1 卷积在 ResNet 的卷积残差块、Inception 网络的 Inception 卷积块等结构中都有应用。此外，1×1 卷积可以替换全连接层，这部分知识将在第 4 章的 YOLO 算法中讲述。

先看一个简单的例子，如图 3.56 所示，假定图像维度为 6×6×1，有一个 1×1×1 的卷积核，卷积核参数值为 2，经过卷积，得到的图像维度仍然为 6×6×1，只是像素值变为原值的 2 倍，这个卷积看起来似乎没有什么实质性作用。

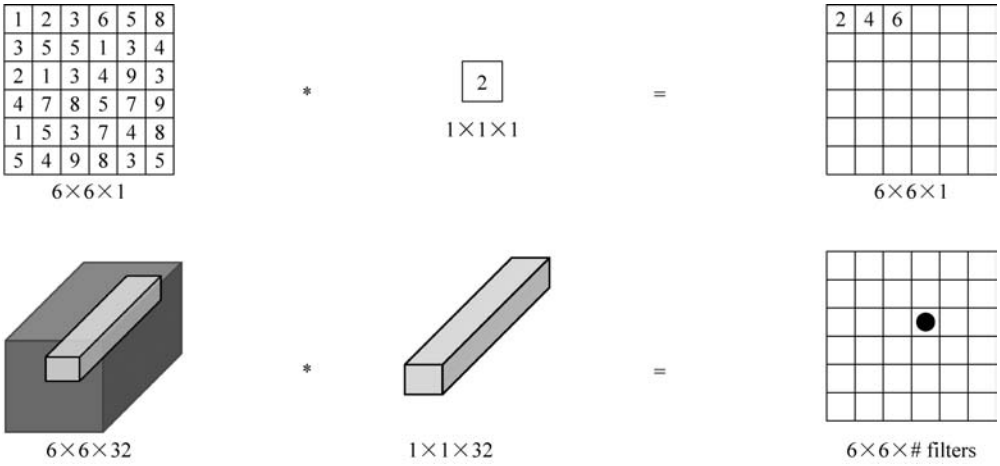


图 3.56　1×1 卷积工作逻辑示意

然而，当输入图像维度变为 6×6×32，卷积核维度为 1×1×32，则情况发生了质变。

这个时候，每次卷积操作，需要从输入图像上切出一个深度为 32 的数据条与卷积核相乘求和，然后填入右侧对应的圆点位置。

如果把 1×1×32 卷积核看成是 32 个权重 w 构成的参数集，输入图像上切出的数据条看作 1×1×32 的输入特征，那么每一个卷积操作相当于一个 $z=wx$ 线性变换过程，此时如果指定激励函数进行非线性变换，则单个 1×1 卷积核相当于扮演了一个神经元的作用。

例如，8 个 1×1×32 卷积核，则相当于有 8 个神经元，每个神经元有 32 个 w 参数，与输入层构建了一个全连接网络，如图 3.57 所示。

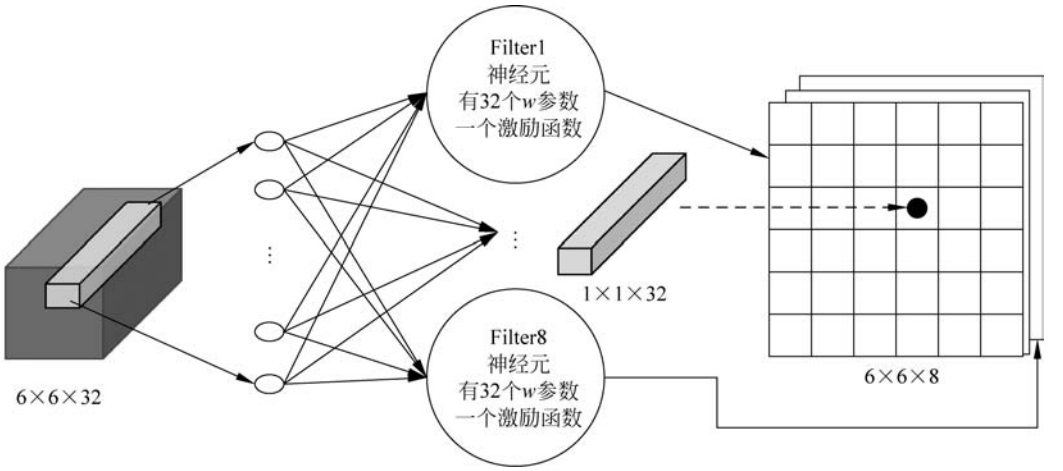


图 3.57　1×1 卷积相当于一个全连接网络

有多少个卷积核，就有多少个神经元，每个卷积核（神经元）对应一个输出通道。如图 3.57 所示，6×6×32 的输入图像经过 8 个 1×1×32 卷积，得到输出特征图的维度为 6×6×8。

1×1 卷积的作用归纳如下。

（1）通道数放缩。池化层只能改变输入图像的高度和宽度，无法改变通道数量，1×1 卷积通过控制卷积核的数量，可以实现通道数量的增加或者减少。

（2）1×1 卷积核的卷积过程相当于全连接层的计算过程，通过加入非线性激励函数，可以增加网络的非线性，使得网络可以表达更复杂的特征。

（3）1×1 卷积在模型设计中，可以起到模型优化和减少参数量的作用。3.22 节介绍的 Inception 网络，可以观察到 1×1 卷积在降低模型计算量方面带来的巨大改进。

3.22　Inception 卷积网络

Inception 又称 GoogLeNet，不写为 GoogleNet，是为了向 Yann LeCuns 设计的卷积网络先驱 LeNet 5 致敬。

Inception 是 2014 年由 Christian Szegedy 等人提出的一种全新的深度学习结构（Szegedy，Liu，et al.，2015），获得 2014 年的 ILSVRC 比赛冠军，后来又衍生出 Inception v2（Ioffe and Szegedy，2015）、Inception v3（Szegedy，Vanhoucke，et al.，2016）和 Inception v4（Szegedy，Ioffe，et al.，2017）等多个版本。

在 Inception 出现之前，大部分流行 CNN 仅仅是把卷积层堆叠得越来越多，使网络越来越深，希望能够得到更好的性能。但层数的增加会带来很多副作用，比如过拟合、梯度消失、梯度爆炸等。

构建卷积网络时，往往需要决定如何配置参数，例如，卷积核的尺寸、卷积核的个数、是否需要添加池化层等。Inception 网络的优势是在同一层上并联多种卷积方法，当然这种并联结构也增加了复杂性。如图 3.58 所示，给出了一个 Inception 卷积块的结构示例。

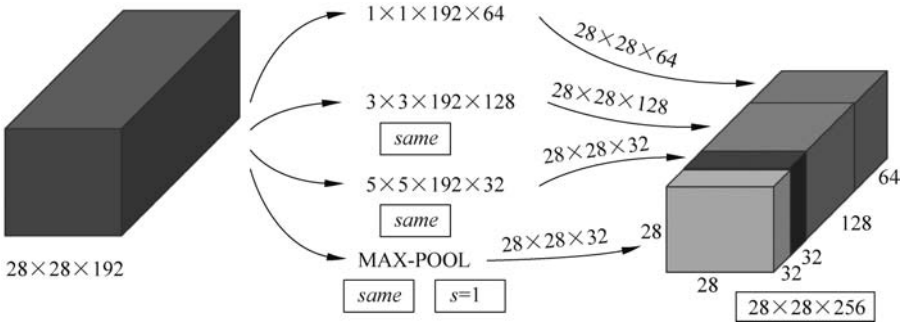


图 3.58　Inception 网络块的基本结构示例

输入图像的维度为 28×28×192，为了构建下一层特征图像的结构，分以下四步进行。

（1）定义 64 个 1×1×192 卷积核，输出特征图的维度为 28×28×64。

（2）定义 128 个 3×3×192 卷积核，same 卷积，保持图像高度与宽度尺寸不变，输出

特征图的维度为 28×28×128，将其堆叠到步骤(1)得到的特征块上。

(3) 定义 32 个 5×5×192 卷积核，same 卷积，保持图像高度与宽度尺寸不变，输出特征图的维度为 28×28×32，将其堆叠到步骤(1)、(2)得到的特征块上。

(4) 定义最大池化层，same 池化，步长为 1，保持图像高度与宽度尺寸不变，输出特征图的维度为 28×28×32，将其堆叠到步骤(1)、(2)、(3)得到的特征块上。注意池化层是不改变输入层的通道数量的，这里输出的特征图之所以为 32 通道，是因为最大池化层后面跟了一个 1×1 卷积。

图 3.58 演示的 Inception 卷积块相当于一个有 4 条并行线路的网络模块。4 条并行线路通过不同的卷积核形状来并行学习参数，发挥各种卷积核的长处，集成为一个综合目标输出，这个综合后的目标输出有可能包含更多来自上一层的特征信息。Inception 卷积块的 4 条并行线路的逻辑描述如图 3.59 所示。

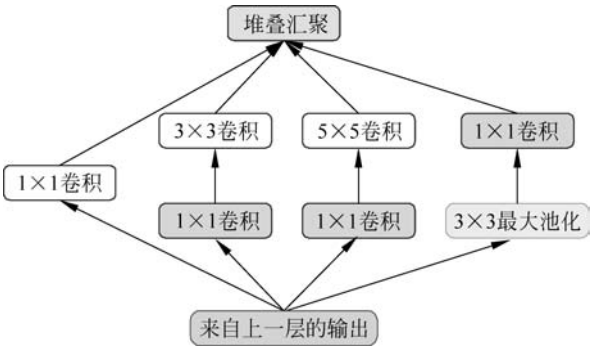


图 3.59 Inception 卷积块逻辑描述示例

3.21 节介绍过 1×1 卷积可以缩放通道数量，图 3.59 中最大池化层后面跟着 1×1 卷积是为了降维，将 192 通道降维到 32 通道。此外，如图 3.59 所示，在做 3×3 卷积、5×5 卷积之前都采用了 1×1 卷积，为什么要这样做？答案是为了降低计算量。以 5×5 卷积分支的计算为例，需要训练的参数量如图 3.60 所示。

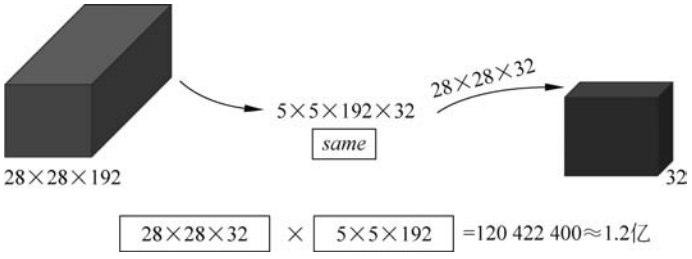


图 3.60 5×5 卷积分支需要训练的参数数量

如果在做 5×5 卷积之前，添加一个 1×1 卷积，如图 3.61 所示，则参数数量可以由 1.2 亿降为 1200 万。

不仅是降低计算量，如果输入的特征存在冗余信息，1×1 卷积层相当于为网络增加了一个特征提取层。

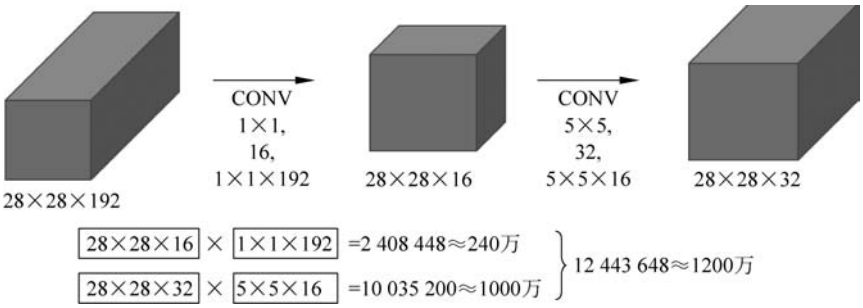


图 3.61　1×1 卷积可以大幅降低计算量

GoogLeNet 将 9 个设计精细的 Inception 卷积块和其他层串联起来。其中，Inception 卷积块的通道数分配是在 ImageNet 数据集上通过大量的实验得来的。如果只统计含有学习参数的层，GoogLeNet 网络有 22 层，如果将池化层也统计在内，则共有 27 层，如图 3.62 所示。

为了解决梯度消失问题，及时获取模型反馈，Inception 在网络中间位置引入两个辅助分类器 softmax0 和 softmax1 加强模型监督。

请参见视频讲解，加深对 Inception 网络结构的理解。

3.23　合成细胞彩色图像

RxRx1 项目组发布了一个开源辅助工具包 rxrx1-utils，提供了将 6 通道灰度图像合成为彩色图像以及图像可视化的函数库，可以通过 git 命令直接从 GitHub 下载安装到当前项目的工作目录中。

如果是 Windows 系统，切换到 MS-DOS 命令行窗口，将当前工作目录切换为 D:\MyTeaching\MyAI\chapter3（读者可根据自己的工作目录做出修改），执行如下 git 命令（需要预先安装 git 软件）。

```
git clone https://github.com/recursionpharma/rxrx1 - utils
```

git 命令完成后，当前工作目录中会新增一个名称为 rxrx1-utils 的文件夹，里面包含对细胞通道图像合并与可视化的一些方法。

如果是其他操作系统，做类似操作，将 rxrx1-utils 安装到项目的工作目录即可。

如果不采用上述 git 命令安装模式，本节的项目素材中包含 rxrx1-utils 的压缩文件，读者可自行将其解压到项目工作目录。

执行程序段 P3.19，将实验批次为 HUVEC-01，3 号实验板，K09 微孔在 s1 位置处的 6 幅灰度图像合成为细胞的彩色图像，如图 3.63 所示。

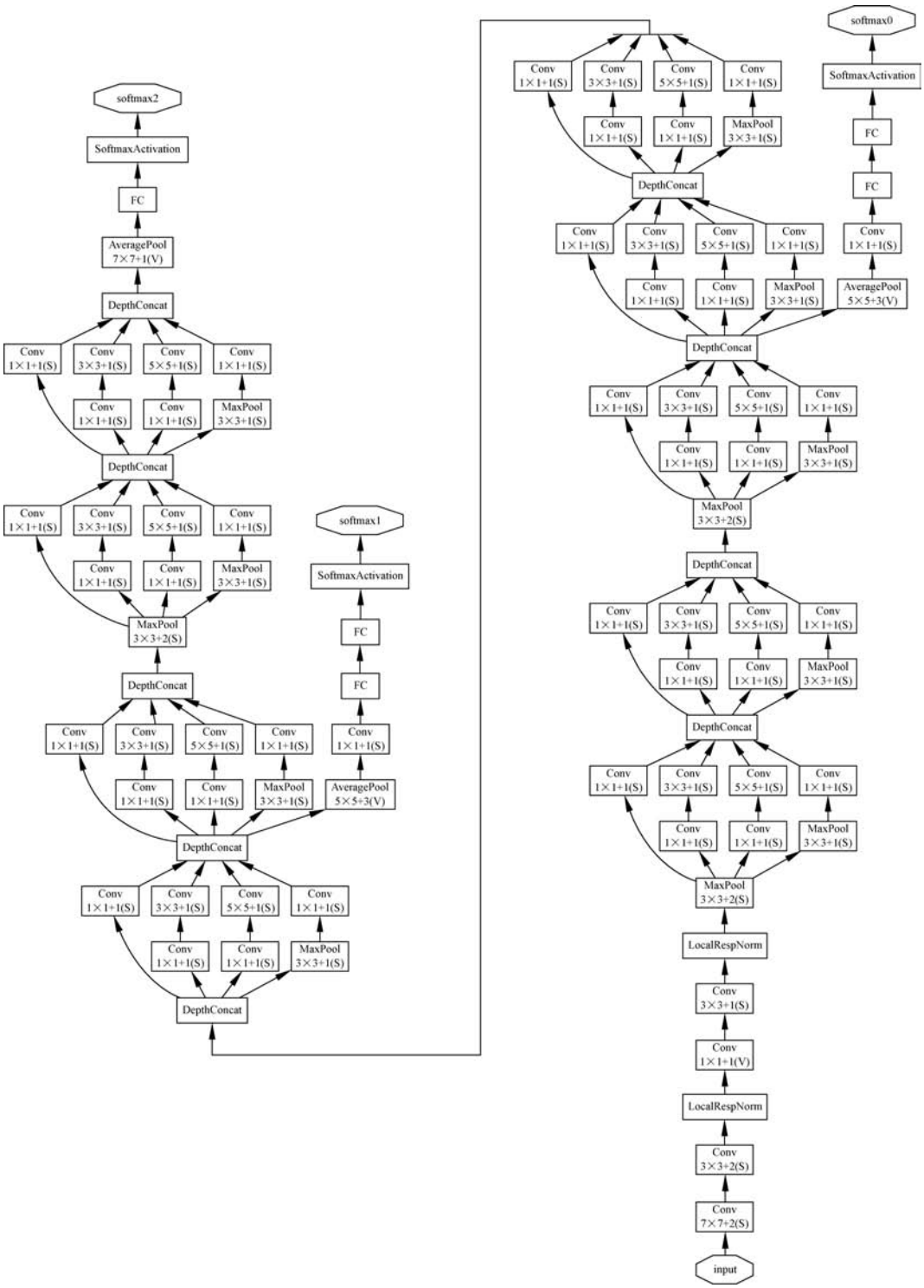


图 3.62　Inception 网络结构

```
P3.19  #将 6 通道灰度图合成细胞彩色图像
103    import sys
104    sys.path.append('rxrx1 - utils')
105    import rxrx.io as rio #导入 rxrx 工具包
       #加载并合成细胞图像,指定数据集,指定实验批次、实验板、微孔和位置参数
106    cell_image = rio.load_site_as_rgb('train', 'HUVEC - 01', 3, 'K09', 1)
107    plt.figure(figsize = (5, 5))
108    plt.imshow(cell_image)
```

合成的彩色图像尺寸与灰度图相同,高度、宽度均为 512px。

后面为了使用 VGG-16 网络做迁移学习,需要将图片尺寸缩减为 224×224×3。执行程序段 P3.20,运用 TensorFlow 的图像缩放函数,完成彩色图像的尺寸变换,如图 3.64 所示。

```
P3.20  #图片尺寸缩减为 224 x 224 x 3
109    import tensorflow as tf
110    resized = tf.image.resize(cell_image,(224,224))
111    resized = np.asarray(resized, dtype = 'uint8')
112    plt.imshow(resized)
113    plt.show()
```

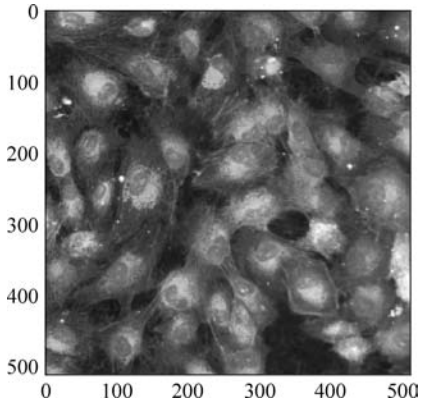


图 3.63　细胞 6 通道灰度图的合成图像

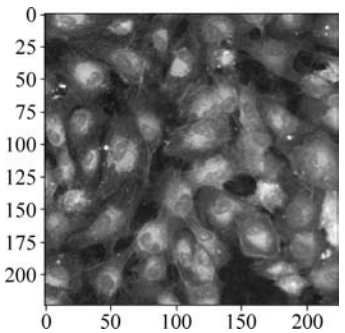


图 3.64　缩减为 224×224×3 后的细胞图像

3.24　数据集划分

3.5 节已经完成了数据的筛选,生成了 U2OS_train 数据集,并且只保留了 id_code 和 sirna 标签这两列,显然仅靠这两列数据无法完成学习建模工作。但是可以根据 id_code 去组织图像数据,这个工作放在 3.25 节完成。标签列共有 1108 种 sirna 标签,需要进行 One-Hot 编码,此工作通过执行程序段 P3.21 完成。

```
P3.21  #完成标签的 One - Hot 编码
114    sirna_code = pd.get_dummies(U2OS_train, columns = ['sirna'])
115    train_data_x = U2OS_train
116    train_data_y = sirna_code.drop(['id_code'], axis = 1)
117    print(train_data_x.shape)
118    print(train_data_y.shape)
119    train_data_y.head()
```

结果显示，train_data_x 数据集维度为(3324，2)，train_data_y 数据集维度为(3324，1108)。

执行程序段 P3.22，完成训练集与验证集的划分。

```
P3.22  #数据集划分为训练集与验证集两部分
120    from sklearn.model_selection import train_test_split
121    x_train_id, x_val_id, Y_train, Y_val = train_test_split(
           train_data_x, train_data_y, test_size = .33, random_state = 0)
122    print('训练集的特征维度: {0},标签维度: {1}'.format(x_train_id.shape,Y_train.shape))
123    print('验证集的特征维度: {0},标签维度: {1}'.format(x_val_id.shape,Y_val.shape))
124    x_train_id.reset_index(drop = True, inplace = True)
125    Y_train.reset_index(drop = True, inplace = True)
126    x_val_id.reset_index(drop = True, inplace = True)
127    Y_val.reset_index(drop = True, inplace = True)
```

运行结果如下。

```
训练集的特征维度: (2227, 2),标签维度: (2227, 1108)
验证集的特征维度: (1097, 2),标签维度: (1097, 1108)
```

执行程序段 P3.23，定义标签名称列表。类名称列表可用于后续的预测结果与真实标签的对照。

```
P3.23  #定义标签名称列表
128    classes = pd.concat([x_train_id['sirna'],x_val_id['sirna']],axis = 0).to_list()
129    for i in range(len(classes)):
130        classes[i] = classes[i].encode(encoding = 'utf - 8')
```

3.25 制作 HDF5 数据集

层次数据格式第 5 版(Hierarchical Data Format Version 5，HDF5)是一种存储与管理大型复杂数据的开源框架，提供了管理、操纵、查看和分析数据集的便捷高效方法，可优化数据访问效率和存储效率，支持大规模并行计算。

程序段 P1.1 已经用语句 import h5py 将 Python 版 HDF5 框架导入，本节直接调用即可。执行程序段 P3.24，根据训练集与验证集中 id_code 读取 6 通道灰度图像，将其合

并为彩图，重定义尺寸并保存为外部文件，存放到当前工作目录的 cell 子目录下，并同步制作 HDF5 数据集文件。

```
P3.24  #6 通道图像合并为彩图，重定义尺寸并保存为外部文件，同步制作 HDF5 数据集
       #此段程序只运行一遍即可
131    def make_dataset(x_data,y_data,x_data_id,Y_data):
132        i = 0
133        for code in x_data_id['id_code']:
134            cell_type = code.partition('_')[0]  #细胞系类型
135            plate = int(code.partition('_')[2][0])  #实验板编号
136            well = code.rpartition('_')[2]  #微孔编号
137            #读取 s1 位置的 6 通道图像，合成彩图
138            img = rio.load_site_as_rgb('train', cell_type, plate, well, 1)
139            resized = tf.image.resize(img,(224,224)) #重定义尺寸
140            resized = np.asarray(resized, dtype = 'uint8')
141            filename = './cell/' + code + '.jpg'
142            plt.imsave(filename,resized) #保存文件
143            x_data[i] = resized #图像特征
144            y_data[i] = Y_data.loc[i].ravel() #标签
145            i += 1
146    x_train_shape = (x_train_id.shape[0],224,224,3) #训练集样本空间的维度
147    x_val_shape = (x_val_id.shape[0],224,224,3) #验证集样本空间的维度
148    with h5py.File('cell.h5','w') as f:
149        x_train = f.create_dataset("x_train", x_train_shape ,'i1') #训练集特征
150            y_train = f.create_dataset("y_train", Y_train.shape ,'i1') #训练集标签
151            x_val = f.create_dataset("x_val", x_val_shape ,'i1') #验证集特征
152            y_val = f.create_dataset("y_val", Y_val.shape ,'i1') #验证集标签
153            classes = f.create_dataset('classes',data = classes) #数据集类名称
154            make_dataset(x_train,y_train,x_train_id,Y_train) #制作训练集
155            make_dataset(x_val,y_val,x_val_id,Y_val) #制作验证集
```

程序段 P3.24 需要读取 3324×6 张灰度图，并将其合并为 3324 张彩色图像，然后将尺寸缩减为 224×224×3，并同步生成含有训练集、验证集和标签名称的 HDF5 数据集文件。

运行结束后，可以将该段程序注释掉，避免反复运行，因为需要的建模数据已经保存到 cell.h5 文件中，图像文件则存放在 cell 子目录下。

3.26　迁移学习与特征提取

机器学习往往依赖大规模数据集和高昂的计算支撑，那么是否可以将别人花费数周乃至数月训练好的开源模型结构以及模型的权重参数下载下来，用于自己的实验项目呢？例如，你手里有一个图像识别分类的项目，但苦于数据集不够充分，或者计算力不能满足需要，此时完全可以借鉴那些在 ImageNet 领域表现良好的开源模型。

ImageNet 拥有来自生活领域的 1400 多万幅图像，那些能够在 ImageNet 上进行 1000 种目标分类的稳定模型，其模型结构和权重参数经过了大量的学习训练与实践检

验，因此，基于 ImageNet 上的成功模型开始你的建模，可能是一个非常不错的选择。这种将某个领域或任务上学习到的知识或模式应用到其他相关领域的方法，称为迁移学习(Transfer Learning)。

Keras 框架中不但提供了 VGG-16、VGG-19、ResNet、Inception、DenseNet 等经典模型的结构定义函数，而且提供了这些模型基于 ImageNet 训练好的权重参数，用户既可以单独使用这些模型的结构，用自己的数据集去训练自己的模型参数；也可以直接将这些经典模型的预训练权重下载下来，直接用这些模型进行预测或者特征提取等工作。

执行程序段 P3.25，用 VGG-16 的 ImageNet 预训练模型进行细胞图像的特征提取。

```
P3.25  #用 VGG-16 的 ImageNet 预训练模型进行特征提取
156    from keras.applications.vgg16 import VGG16
157    from keras.preprocessing import image
158    from keras.applications.vgg16 import preprocess_input
       #下载或读取预训练模型，首次执行时需要下载模型参数
159    model = VGG16(weights='imagenet', include_top=False)
       #单个图像的特征提取
160    def VGG16_extract_features(img):
161        x = np.expand_dims(img, axis=0)
162        features = model.predict(x)
163        return features
       #用前面的 resized 图像测试
164    features = VGG16_extract_features(resized)
165    features.shape
```

其中，第 159 行程序表示采用 ImageNet 训练的权重参数，特征提取时，不采用最后的三个全连接层，即特征提取模型只包括 VGG-16 前面的 13 层，VGG-16 的结构参见前面的图 3.52。P3.25 输出的特征图维度为(1,7,7,512)。

为了对训练集与测试集进行特征提取，执行程序段 P3.26，加载细胞图像数据集。

```
P3.26  #加载细胞图像数据集
166    def load_dataset():
167        with h5py.File('cell.h5','r') as f:
168            x_train = f['x_train'][:]     #读取训练集特征
169            y_train = f['y_train'][:]     #读取训练集标签
170            x_val = f['x_val'][:]         #读取验证集特征
171            y_val = f['y_val'][:]         #读取验证集标签
172            classes = f['classes'][:]     #类名称
173            return x_train, y_train, x_val, y_val, classes
174    X_train, Y_train, X_val, Y_val, classes = load_dataset()
```

程序段 P3.26 可能会被其他程序反复用到，所以将其单独保存为 Python 程序文件 cell_utils.py，放到项目文件夹中，便于其他程序引用。

执行程序段 P3.27，完成训练集特征提取。得到的特征集维度为(2227,7,7,512)。

```
P3.27  #用 VGG-16 对训练集进行特征提取,此段程序只运行一遍即可
175    m = X_train.shape[0]
176    x_train = np.zeros((m,7,7,512))
177    for i in range(m):
178        x_train[i] = VGG16_extract_features(X_train[i])
179    print(x_train.shape)
```

执行程序段 P3.28,完成验证集特征提取。得到的特征集维度为(1097,7,7,512)。

```
P3.28  #用 VGG-16 对验证集进行特征提取,此段程序只运行一遍即可
180    m = X_val.shape[0]
181    x_val = np.zeros((m,7,7,512))
182    for i in range(m):
183        x_val[i] = VGG16_extract_features(X_val[i])
184    print(x_val.shape)
```

执行程序段 P3.29,将训练集特征与验证集特征保存为 HDF5 特征集文件 cell_features.h5。

```
P3.29  #保存用 VGG-16 提取的特征,此段程序只运行一遍即可
185    with h5py.File('cell_features.h5','w') as f:
186        x_train = f.create_dataset("x_train", data = x_train) #训练集特征
187        y_train = f.create_dataset("y_train", data = Y_train) #训练集标签
188        x_val = f.create_dataset("x_val", data = x_val) #验证集特征
189        y_val = f.create_dataset("y_val", data = Y_val) #验证集标签
190        classes = f.create_dataset('classes',data = classes) #数据集类名称
```

程序段 P3.27、P3.28 和 P3.29,这三段程序运行结束后,可以注释起来,因为提取的特征已经保存到外部的 HDF5 数据集文件中,后面根据需要直接从特征文件读取即可。

至此,细胞图像数据的预处理工作全部完成,保存当前程序文档。

3.27　基于 VGG-16 的迁移学习

关闭此前建立的程序文档,重启 NoteBook 后台服务器,释放前述特征提取占用的内存资源。新建一个程序文档 Cell_Model.ipynb,执行程序段 P3.30,读取特征集。

```
P3.30  #加载 VGG16 提取的特征数据集
191    import h5py
192    def load_VGG16_dataset():
193        with h5py.File('cell_features.h5','r') as f:
194            x_train = f['x_train'][:] #读取训练集特征
195            y_train = f['y_train'][:] #读取训练集标签
196            x_val = f['x_val'][:]     #读取验证集特征
197            y_val = f['y_val'][:]     #读取验证集标签
198            classes = f['classes'][:] #类名称
```

```
199             return x_train, y_train, x_val, y_val, classes
200     X_train, Y_train, X_val, Y_val, classes = load_VGG16_dataset()
```

自行定义一个基于 VGG-16 的细胞图像分类模型，不妨将新模型命名为 VGG16-Cell-Transfer。模型结构如图 3.65 所示。模型包含两部分，一部分称作特征提取，一部分是自主学习。

左边虚线框表示模型的输入部分，相当于用 VGG-16 模型的前 13 层的 ImageNet 参数做细胞图像的特征提取，这部分工作称作迁移学习。

右边的虚线框表示自主学习部分。自主学习定义了一个 1×1 卷积进行降维，卷积核数量可以调整，程序测试时采用了 32。1×1 卷积后面跟着三个全连接层，类似 VGG-16 后面的三层结构，不同的是神经元的数量做了调整，为了降低计算量，由 4096 调整为 2048，如果计算力允许，建议还是采用 4096 更好一些。Softmax 层的单元数量为 1108，与 sirna 的标签数保持一致。

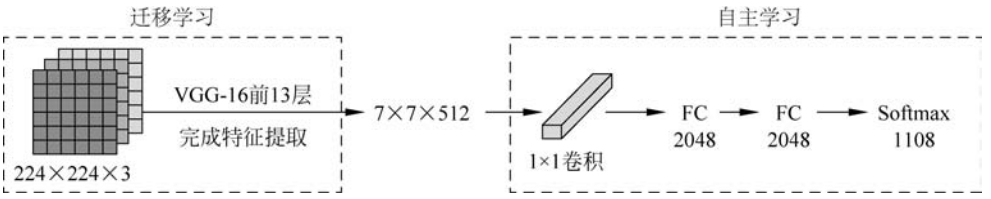


图 3.65　自定义基于 VGG-16 的迁移学习模型

根据图 3.65 的结构，执行程序段 P3.31，自定义基于 VGG-16 的迁移学习模型。

```
P3.31   #基于 VGG-16 的 ImageNet 预训练模型，定义迁移学习模型
201     from keras import Sequential
202     from keras.layers import Dense, Conv2D, Flatten, Dropout
203     model = Sequential(name = 'VGG16-Cell-Transfer')
204     model.add(Conv2D(filters = 64, kernel_size = (1, 1), activation = 'relu',
                padding = 'same', input_shape = (7, 7, 512)))
205     model.add(Flatten())
206     model.add(Dense(2048, activation = 'relu', name = 'fc1'))
207     model.add(Dropout(0.5))
208     model.add(Dense(2048, activation = 'relu', name = 'fc2'))
209     model.add(Dropout(0.5))
210     model.add(Dense(1108, activation = 'softmax', name = 'prediction'))
211     model.summary()
```

模型摘要显示参数量为 12 924 052 个，不到 VGG-16 参数数量的十分之一。

执行程序段 P3.32，指定优化算法、损失函数和模型评价指标，编译模型。

```
P3.32   #编译模型
212     model.compile(optimizer = 'RMSProp',
                      loss = 'categorical_crossentropy',
                      metrics = ['accuracy'])
```

执行程序段 P3.33，指定训练代数、批处理大小、训练集和验证集参数，开始模型训练。

```
P3.33  #训练模型
213    epochs = 10
214    batch_size = 32
215    history = model.fit(X_train, Y_train, epochs = epochs, batch_size = batch_size,
                validation_data = (X_val,Y_val))
```

由于采用了迁移学习，节省了大量的计算，所以模型训练速度很快。

执行程序段 P3.34，观察模型在训练集与验证集上准确率与损失值对比。

```
P3.34  #训练集和验证集上的准确率与损失值对比
216    import matplotlib.pyplot as plt
217    %matplotlib inline
218    x = range(1, len(history.history['accuracy']) + 1)
219    plt.plot(x, history.history['accuracy'])
220    plt.plot(x, history.history['val_accuracy'])
221    plt.title('Model accuracy')
222    plt.ylabel('Accuracy')
223    plt.xlabel('Epoch')
224    plt.xticks(x)
225    plt.legend(['Train', 'Val'], loc = 'upper left')
226    plt.show()
227    plt.plot(x, history.history['loss'])
228    plt.plot(x, history.history['val_loss'])
229    plt.title('Model loss')
230    plt.ylabel('Loss')
231    plt.xlabel('Epoch')
232    plt.xticks(x)
233    plt.legend(['Train', 'Val'], loc = 'lower left')
234    plt.show()
```

模型准确率对比如图 3.66 所示，损失值对比如图 3.67 所示。

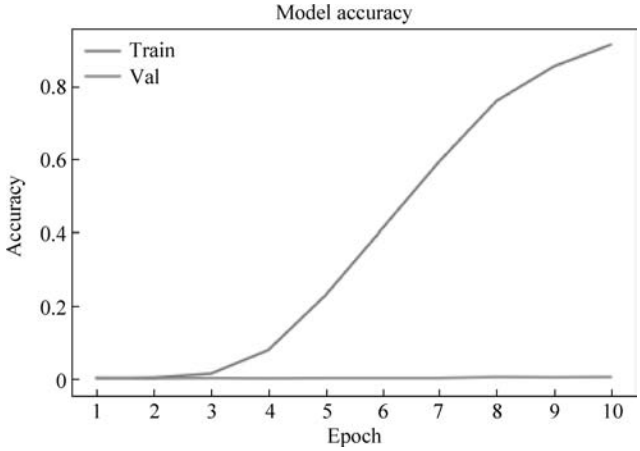


图 3.66　模型在训练集和验证集上的准确率对比

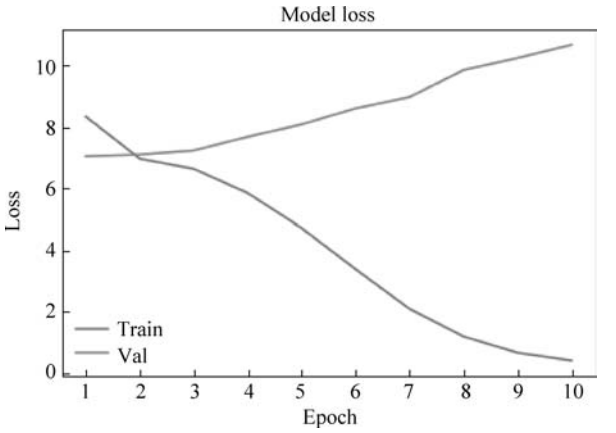


图 3.67　模型在训练集和验证集上的损失值对比

显然，这个结果令人失望！虽然模型在训练集上表现出了良好的学习能力，但是在验证集上的学习效果几乎为 0，确实令人沮丧，模型的方差很高，泛化能力极低。

如何解决上述问题？是模型算法不够好吗？如果此时回顾 3.13 节介绍的偏差与方差优化方法，则不难发现，问题出在数据集本身，归纳如下。

(1) 数据量不够充分。本章案例没有让全部数据参与学习训练，因为多数情况下，读者并不具备所需要的计算能力。选取的 U2OS 四个批次的实验数据远远不够。

(2) 迁移学习在细胞图像特征提取上的意义不大，虽然 ImageNet 数据集庞大，但是 ImageNet 包含的细胞图像的数据可能并不充分。

(3) 实验本身的不确定性，导致即使同一批次实验，在同一个实验板的同一微孔上的 s1 位置和 s2 位置的成像，也会呈现显著的不同，读者可以自行修改程序段 P3.19 中的第 106 行语句，将其中的位置参数 1 修改为 2，进行对比观察。至于不同批次或不同实验板，差别更大。RxRx1 项目网站(https://www.rxrx.ai/)对此有专门的描述与图像展示。

可以想见，模型之所以在验证集上表现糟糕，是因为验证集的图像与训练集不是同类分布，训练集上学习到的参数无法有效应用到验证集上去。

业界流行一种说法：数据和特征决定学习的上限，算法和模型只是逼近这个上限。此处再次得到印证。如果缺乏足够的算力支持，不能充分合理使用 RxRx1 项目数据集，则无法期待更好的模型效果。

3.28　训练 ResNet50 模型

既然直接基于 ImageNet 的迁移学习对细胞图像的分类效果不够理想，本节采用 ResNet50 模型，从头开始训练。

前面的程序段 P3.29 已经完成了数据集的 HDF5 格式存储，执行程序段 P3.35 直接读取数据集，用于 ResNet50 模型的训练。

```
P3.35  #加载细胞图像数据集
235    from cell_utils import load_dataset
236    X_train, Y_train, X_val, Y_val, classes = load_dataset()
```

执行程序段 P3.36，加载 ResNet50 模型用于细胞图像的分类训练。

```
P3.36  #基于 ResNet50 模型的建模训练
237    from keras.applications.resnet50 import ResNet50
       #加载 ResNet50 模型，不带 ImageNet 训练权重，分类数为 1108
238    ResNet50_model = ResNet50(include_top = True, weights = None, classes = 1108)
       #模型编译
239    ResNet50_model.compile(optimizer = 'RMSProp',
                              loss = 'categorical_crossentropy',
                              metrics = ['accuracy'])
       #开始训练
240    epochs = 5
241    batch_size = 32
242    history = ResNet50_model.fit(X_train, Y_train, epochs = epochs, batch_size = batch_size,
                  validation_data = (X_val,Y_val))
```

程序段 P3.36 只定义了 ResNet50 模型的 5 代训练，在普通台式计算机(8 核 i7 处理器)上大约需要 80min，训练完成后，模型准确率对比如图 3.68 所示，损失值对比如图 3.69 所示。

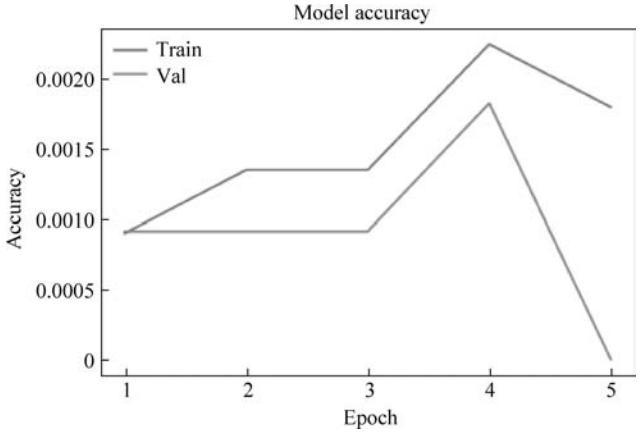


图 3.68　ResNet50 准确率对比

上述训练结果仍然很糟糕，甚至图 3.68 与图 3.69 的曲线趋势是矛盾和背离的，这不是 ResNet50 模型的错，选用的数据量不够充分，训练集与验证集数据分布极其不平衡，迭代次数也很少，因此图 3.68 和图 3.69 的观察结果不具参考价值。

获得本次 RxRx1 项目比赛第一名的作者团队，已经将项目代码开源在 Github 上，感兴趣的读者可以继续观摩学习，项目网址：https://github.com/maciej-sypetkowski/kaggle-rcic-1st。

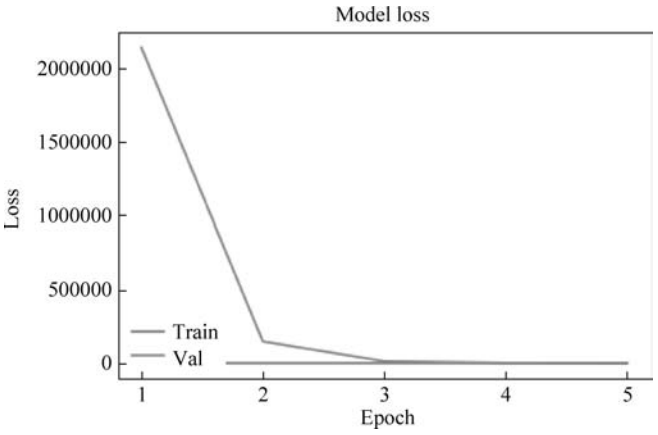


图 3.69 ResNet50 损失值对比

执行程序段 P3.37,保存 3.27 节完成的 VGG-16 迁移学习模型和本节完成的 ResNet50 的学习模型。

```
P3.37 #保存模型
model.save('VGG16_Transfer_model.h5')      #保存 VGG-16 迁移学习的结构和参数
ResNet50_model.save('ResNet50_model.h5')   #保存 ResNet50 模型结构和参数
```

3.29 ResNet50 模型预测

执行程序段 P3.38,加载 3.28 节完成的 ResNet50 模型,随机指定一幅细胞图像,用模型对其分类预测,显示其真实标签与模型预测的标签。

```
P3.38 #读取模型,检验预测效果
from keras.models import load_model
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input
import pandas as pd
import numpy as np
ResNet50_model = load_model('ResNet50_model.h5')    #加载模型
img_path = './cell/U2OS-01_2_B12.jpg'                #随机指定一幅测试图像
img = image.load_img(img_path, target_size=(224, 224))
plt.imshow(img)                                      #显示原图
train = pd.read_csv("./dataset/train.csv")
print('真实的标签为:{0}'.format(
     train[train.id_code == 'U2OS-01_2_B12']['sirna'].values))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
pred = ResNet50_model.predict(x)                     #模型预测
#将预测得到的概率值进行 One-Hot 编码
```

```
pred = pred.ravel()
index = np.argmax(pred)
pred = np.zeros(1108)
pred[index] = 1
print('模型预测值为: {0}'.format(pred))
for i, y in enumerate(Y_train):
    if (y == pred).all():
        print('模型预测的标签为: {0}'.format(classes[i].decode('utf-8')))
        break
```

本次模型预测，真实的图像如图 3.70 所示，真实的标签为 sirna_1018。模型预测的标签为 sirna_988。

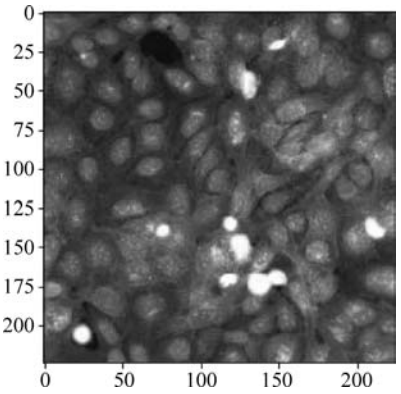


图 3.70　真实标签对应的图像

显然不必对预测结果抱有过高期望，受限于计算能力不够充分和项目的数据集过于庞大，本章在项目处理上以学习和演示为主。本章完成的项目结构，可能比爱因斯坦的小板凳糟糕很多，但是仍然值得庆贺的是，你基于真实的项目背景完成了一次历练。

需要着重指出的是，前面将 6 通道细胞图像合并为彩色图像，作为训练输入并不是最优方案，因为合成的彩色图像可能会丢失很多细节，如果算力允许，更好的方案是直接输入 6×512×512 的灰度图像。

对本项目全部程序（程序段 P3.1～P3.38，268 行编码）的全部编码做整体测试，测试主机的 CPU 配置为 Intel® Core™ i7-6700 CPU@3.40Hz，内存配置为 16GB，项目整体运行时间在 100 分钟左右。

小结

本章以细胞图像分类为背景，夯实了神经网络的理论基础，系统梳理了神经网络的基本理论与方法，包括神经网络的基本结构、符号化表示方法、激励函数、损失函数、梯度下降、正向传播、反向传播、偏差与方差、正则化、Mini-Batch 梯度下降、优化算法、参数与超参数、Softmax 回归函数等基础内容，深入学习了 VGG-16 卷积网络、ResNet 卷积网络、1×1 卷积网络和 Inception 卷积网络的方法与原理。

基于 VGG-16 和 ResNet50 网络的迁移学习方法，完成了细胞图像分类学习模型的构建、训练与评估，在数据集预处理与数据集 HDF5 文件制作等实践探索层面前进了一大步。

习题

一、单选题

1. 神经元的计算逻辑是什么？（　　）

A. 激励函数后面跟着线性计算，形如 $z=wx+b$

B. 神经元计算所有输入特征的平均值，然后传递给激励函数处理

C. 神经元先做线性计算，形如 $z=wx+b$，然后传递给激励函数

D. 神经元相当于一个函数 g，对输入的值 x 进行放缩，例如 $wx+b$

2. 有程序段：

```
a = np.random.randn(4,3)
b = np.random.randn(3,2)
c = a * b
```

根据数组 a、b、c 的定义，推断 c 的维度为（　　）。

A. c.shape=(3,3)

B. c.shape=(4,2)

C. c.shape=(4,3)

D. c 的计算会出错，因为 a 和 b 的维度不匹配

3. 假设每个样本的特征数量为 n，用 $\boldsymbol{X}$ 表示整个样本矩阵：

$$\boldsymbol{X}=[x^{(1)},x^{(2)},x^{(3)},\cdots,x^{(m)}]$$

则 $\boldsymbol{X}$ 的维度为（　　）。

A. (1,m)　　B. (m,1)　　C. (n,m)　　D. (m,n)

4. 哪一个是正确的正向传播表示方法？（　　）

A. $Z^{[l]}=w^{[l]}A^{[l-1]}+b^{[l]}\quad A^{[l]}=g^{[l]}(Z^{[l]})$

B. $Z^{[l]}=w^{[l]}A^{[l]}+b^{[l]}\quad A^{[l+1]}=g^{[l]}(Z^{[l]})$

C. $Z^{[l]}=w^{[l]}A^{[l-1]}+b^{[l-1]}\quad A^{[l]}=g^{[l]}(Z^{[l]})$

D. $Z^{[l]}=w^{[l]}A^{[l]}+b^{[l]}\quad A^{[l+1]}=g^{[l+1]}(Z^{[l]})$

5. 假定你正在构建一个神经网络，用于区分西瓜（$y=1$）与西红柿（$y=0$）。下列哪个函数可用于输出层的激励函数？（　　）

A. tanh　　B. ReLU　　C. Leaky ReLU　　D. sigmoid

6. 正向传播过程中，需要缓存一些变量，目的是（　　）。

A. 用于跟踪超参数的变化，加快参数搜索

B. 将缓存变量用于同一层的反向传播计算过程

C. 用于将反向传播块计算的值传回对应的正向传播块

D. 缓存变量包含用于计算损失函数的值

7. 如果你有 10 000 000 个样本，将如何划分数据集？（　　）

A. 98% train，1% dev，1% test

B. 33% train,33% dev,33% test

C. 60% train,20% dev,20% test

D. 50% train,20% dev,30% test

8. 验证集和测试集,应该(　　)。

A. 样本来自同一分布　　B. 样本来自不同分布

C. 样本之间有一一对应关系　　D. 拥有相同数量的样本

9. 什么是权重衰减?(　　)

A. 一种为了避免梯度消失增加权重参数值的技术

B. 一种 L2 正则化技术,目的是每次迭代都降低权重参数值

C. 一种训练过程中降低学习率的技术

D. 如果训练集有噪声数据,权重会逐渐下降

10. 如果增加正则化超参数 lambda 的值,则(　　)。

A. 权重参数趋向变得越来越小(更接近于 0)

B. 权重参数趋向变得越来越大(更远离 0)

C. Lambda 参数值翻倍,将导致权重参数值翻倍

D. 梯度下降将因 Lambda 值的增加,梯度变得更大

11. 如果需要表示第 8 组 mini-batch 的第 7 个样本在网络第 3 层的输出值,以下正确的是(　　)。

A. $a^{[3]\{7\}(8)}$　　B. $a^{[8]\{7\}(3)}$　　C. $a^{[8]\{3\}(7)}$　　D. $a^{[3]\{8\}(7)}$

12. 关于 mini-batch 梯度下降,描述正确的是(　　)。

A. 基于 mini-batch 的一次梯度下降,比基于 batch gradient descent 的一次梯度下降计算速度快

B. 基于 mini-batch 完成一个 epoch 训练,比基于 batch gradient descent 完成一代训练计算速度快

C. 为了节省时间,所有的 mini-batch 可以一次输入网络完成训练

D. mini-batch 梯度下降与随机梯度下降相比,需要更多的迭代步数

13. 假定训练模型的损失函数下降曲线如图 3.71 所示。则以下描述正确的是(　　)。

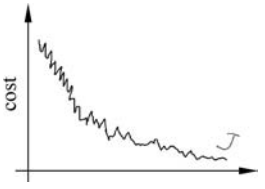


图 3.71　损失函数下降曲线

A. 不管使用的是 batch gradient descent 还是 mini-batch gradient descent,肯定是有严重的问题存在

B. 不管使用的是 batch gradient descent 还是 mini-batch gradient descent,这条曲线看起来是可以接受的正常情况

C. 如果使用的是 mini-batch gradient descent,说明有问题存在,但是如果使用的是 batch gradient descent,则看起来是可以接受的正常情况

D. 如果使用的是 mini-batch gradient descent,这看起来是可以接受的正常情况,但是如果使用的是 batch gradient descent,则说明某些环节存在严重的问题

14. 以下方法，哪种不是一个好的学习率衰减模式？这里 epochNum 表示 epoch 的编号(　　)。

A. $\alpha = e^{\text{epochNum}}\alpha_0$

B. $\alpha = 0.95^{\text{epochNum}}\alpha_0$

C. $\alpha = \dfrac{k}{\sqrt{\text{epochNum}}}\alpha_0$

D. $\alpha = \dfrac{1}{1+\text{decayRate}\times\text{epochNum}}\alpha_0$

二、多选题

1. 以下描述正确的是(　　)。

A. $w^{[2]}$，$b^{[2]}$表示网络第 2 层的训练参数

B. $\boldsymbol{a}^{[2]}$表示网络第 2 层激励函数的输出向量

C. $\boldsymbol{z}^{[2]}$表示网络第 2 层线性计算的输出向量

D. $\boldsymbol{a}^{[2](12)}$表示第 12 个样本在网络第 2 层的输出向量

2. 对于如图 3.72 所示的神经网络：

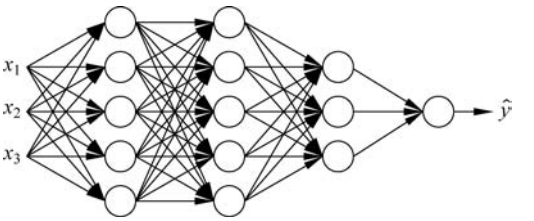


图 3.72　神经网络

以下描述正确的是(　　)。

A. $w^{[1]}$的维度为(5,3)　　B. $b^{[1]}$的维度为(1,1)

C. $w^{[1]}$的维度为(3,5)　　D. $w^{[3]}$的维度为(3,5)

3. 对于如图 3.72 所示的神经网络，以下描述正确的是(　　)。

A. $z^{[1]}$的维度为(5,3)　　B. $a^{[1]}$的维度为(5,1)

C. $z^{[1]}$的维度为(5,1)　　D. $z^{[2]}$的维度为(5,1)

4. 以下属于超参数的是(　　)。

A. 学习率 α

B. 网络的层数 L

C. 各层的参数矩阵 $\boldsymbol{w}$ 和偏差向量 $\boldsymbol{b}$

D. 各层激励函数的输出向量 $\boldsymbol{a}$

5. 如果网络模型具有高方差，以下哪些措施可以尝试？(　　)

A. 采用正则化方法

B. 增加网络的层数或者增加神经元的数量

C. 获取更多的测试数据

D. 获取更多的训练数据

6. 如果给一个超市构建了一个水果分类器，能够识别香蕉、苹果、西瓜、猕猴桃，假定训练集的误差为0.5%，验证集的误差为7%。

以下哪些措施有可能改善模型性能？（　　）

A. 增加正则化参数 lambda 的值

B. 降低正则化参数 lambda 的值

C. 获取更多的训练数据

D. 增加网络规模，例如层数和神经元数量

7. 增加 dropout 正则化的 keep_prob 参数值，例如从 0.5 修改为 0.6，将导致（　　）。

A. 增加正则化效果

B. 降低正则化效果

C. 导致神经网络在训练集上的误差增加

D. 导致神经网络在训练集上的误差降低

8. 为什么 mini-batch 的大小不能为1，也不能为样本总数 m？（　　）

A. 如果 mini-batch 大小为1，将不得不每次处理整个样本集才能完成一次梯度计算

B. 如果 mini-batch 大小为 m，将不得不采用随机梯度下降，其计算量比 mini-batch 大很多

C. 如果 mini-batch 大小为1，将不能获得向量化计算的优势，难以收敛到最优值

D. 如果 mini-batch 大小为 m，每次迭代都需要计算整个训练集，对计算力要求高

9. 关于 Adam 算法，以下正确的是（　　）。

A. Adam 算法的学习率需要调整

B. Adam 算法综合了 RMSprop 和 Momentum 的优点

C. Adam 算法应该用于 Batch Gradient Descent，不能用于 mini-batch Gradient Descent

D. Adam 算法的计算量比 RMSprop 和 Momentum 大

三、判断题

1. 用 tanh 作隐藏层的激励函数，一般情况下要好于 sigmoid 函数，因为 tanh 函数的均值更接近为0，有利于数据中心化和规范化。

2. 网络的深层结构比浅层的结构更易于计算和提取复杂的特征。

3. m 个样本，可以整体完成正向传播的一次迭代。

4. 正向传播经过第 L 层时，需要知道该层的激励函数，反向传播经过第 L 层时，不需要知道该层的激励函数。

5. 如果要在多个参数中协同寻找最优参数值，应该使用均匀网格取值而不是随机取值。

6. 超参数如果选择不当，对模型将有较大的负面影响，所以在参数调整策略方面，所有超参数都同等重要。

7. 寻找最优超参数费时费力，所以，应该在模型训练之前就指定最优参数。

8. 机器学习算法在图像识别领域的性能表现可能超过人类的能力。

9. 四块 384 微孔板(16 列,24 行),最外层的行和列空置,则共有 308×4=1232 个微孔可用。

10. RxRx1 数据集项目中,从同一个微孔的 site1 和 site2 两个位置各采集 6 幅图像。

11. 人工神经网络的基本构成单位是神经元,又称计算单元。

12. $a_j^{[l](i)}$ 表示第 i 个样本在第 ℓ 层的第 j 个神经元的输出值。

13. $\boldsymbol{w}_{jk}^{[l]}$ 表示第 ℓ 层的第 j 个神经元对应第 k 个特征的权重参数。

14. 激励函数不但表征了单个神经元的输出逻辑,而且作为下一层的输入,也在建构着层与层之间神经元的关系,对整个神经网络的功能逻辑有重要影响。

15. Sigmoid 函数能够把输入的连续实值变换为 0～1 的输出。

16. Sigmoid 函数的导数是以它本身为因变量的函数。

17. 通过求解损失函数的最小值,可以实现求解模型参数、优化模型参数和评价模型学习效果的目的。

18. 梯度下降,就是沿着函数的梯度(导数)方向更新自变量,使得函数的取值越来越小,直至达到全局最小或者局部最小。

19. 从正向传播开始,经过损失函数计算、反向传播到参数更新结束,称为一次梯度下降或一次迭代。

20. 训练集中的所有样本均完成一次梯度下降迭代,则称模型完成了一代训练。

21. 方差是指模型预测结果的离散程度,方差可用于度量模型在不同数据集上的稳定性,即模型在不同数据集上的泛化能力。

22. 正则化是为了防止模型过拟合而引入额外信息,对模型原有逻辑进行外部干预和修正,从而提高模型的泛化能力。

23. L2 正则化和 Dropout 正则化,是神经网络模型经常采用的正则化方法。

24. 训练集共有 64 000 个样本,mini-batch 大小为 64,模型经过了 5 个 epoch 训练,意味着模型迭代了 5000 次。

25. 残差块的跳连模式,可以一次跳连 5 个卷积层。

26. 1×1 卷积不但可以实现通道数量放缩效果,而且可以大大降低计算量。

27. VGG-19 比 VGG-16 多了三个卷积层。

28. Inception V1 网络包含 9 个 Inception 卷积块,每个 Inception 卷积块包含四个并行的计算通路,每个通路都包含 1×1 卷积。

29. HDF5 数据集可优化数据访问效率和存储效率,支持大规模并行计算。

30. 迁移学习是将某个领域或任务上学习到的知识或模式应用到其他相关领域的学习方法。

31. ResNet152 一定比 ResNet34 预测的准确率高。

四、编程题

请根据如图 3.73 所示的模型结构与参数,定义一个卷积模型,对 10 种植物花朵做出分类识别和预测。绘制模型在训练集和验证集上的准确率曲线。

1. 数据集描述。

包含 10 种开花植物的 210 幅图像(128×128×3),图像文件为.png 格式。flower-

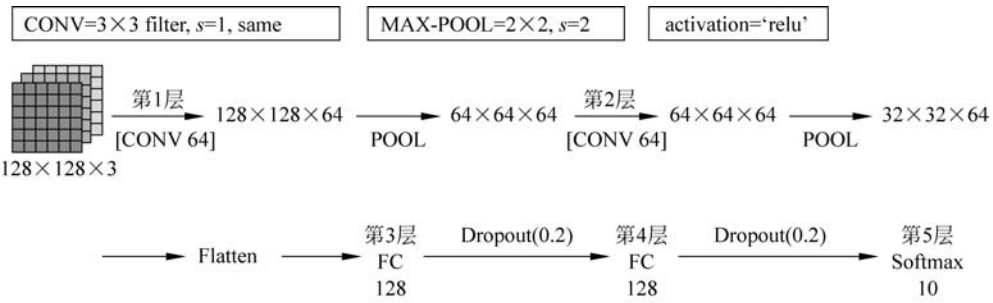


图 3.73　卷积网络的基本结构及其参数

labels.csv 为标签文件，标签用数字 0～9 表示。

2. 10 种植物花朵的标签意义如下。

0⇒phlox；1⇒rose；2⇒calendula；3⇒iris；

4⇒leucanthemum maximum；5⇒bellflower；6⇒viola；

7⇒rudbeckia laciniata (Goldquelle)；8⇒peony；9⇒aquilegia

3. 需要完成的编程工作。

本章习题课件中包含数据集和程序文档 flower_classification_begin.ipynb，程序文档已经完成了数据集制作和划分工作。请在此基础上完成模型的定义和训练工作。

4. 具体要求包括：

(1) 特征集归一化。

(2) 对标签进行 One-Hot 编码。

(3) 定义训练模型。

(4) 模型编译。

(5) 模型训练。

(6) 完成 20 代训练，绘制准确率与损失函数曲线。