武汉大学计算机学院

本科生课程设计报告

基于 numpy 的 DNN 神经网络新冠肺炎高效 CT 图像识别分类模型

专 业 名 称 ：计算机学院 软件工程

课 程 名 称 ：商务智能

指 导 教 师 ：朱卫平

学 生 学 号 ：2019302080117

学 生 姓 名 ：钟孝云

二〇二一年十二月

郑 重 声 明

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

本人签名：钟孝云　　日期：2021.12.28

摘　要

自 2019 年新冠肺炎在中国武汉爆发以来，截止至 2021 年 12 月，全球超过 200 个国家及地区累计确诊新冠肺炎人数已超 2 亿 8 千万人，迫切需要制定有效措施控制这一流行病。本课程设计提出了一种基于 numpy 库实现的 DNN 模型，采用线性全连接层进行 CT 图片数据的拟合并以此进行图像的分类预测，使用简单的模型参数获得不错的效果，在模型大小不超过 5MB 的同时，训练集和测试集的 accuracy 分别超过了 99%和 97.8%，同时不调用 API 实现模型的拟合过程，达到模型设置可控化、模型训练快速化、模型大小轻便化、模型使用简单化的效果。使用商务智能的预测性分析，通过 CT 图像预测患者感染新冠肺炎的概率。

关键词：DNN；深度学习；新冠肺炎；CT 图像；商务智能；预测性分析

目录

1. 请说明分类算法衡量时准确率(accuracy)的局限性并举例说明。

1.1 准确率(accuracy)的局限性：

准确率是样本分类问题中最简单也是最直观的评价指标。但存在明显的缺陷：比如负样本占99%时，分类器把所以样本都预测为负样本也可以获得99%的准确率。所以，当不同类别的样本比例非常不均衡时，占比大的类别往往成为影响准确率的最主要因素，此时准确率指标并不足以说明分类器的好坏。

1.2 准确率(accuracy)的局限性举例说明：

举例说明：google 抓取了 argcv 100 个页面，而它索引中共有 10,000,000 个页面,随机抽一个页面，要求分类是否是 argcv 的页面。如果以 accuracy 来判断分类效率，一个算法是直接将所有的页面都判断为"不是 argcv 的页面",因为这样效率非常高,而 accuracy 已经到了 99.999%(9,999,900/10,000,000),领先绝大多数分类器，但事实上这个算法显然不是需求期待的。

2. 请说明卷积神经网络中参数共享和稀疏连接的含义。

2.1 参数共享

由于卷积神经网络要处理的输入图片数据较大，如果统一采用全连接层进行图像特征提取时，网络中要训练的参数数量太多，比如如果图片数据为 100*100，而第一层隐藏层神经元个数为 10000，则需要一共10^8个参数记录连接的权值，太大的参数数量不利于网络的训练和拟合。

卷积神经网络的参数共享就是为了解决这个问题：对一副图像采用统一的卷积核进行卷积操作，对原图像中每个大小为 5*5 的窗口使用统一的卷积模板进行卷积，从而提取出卷积后的特征图；假设我们使用 100 个不一样的卷积核，分别执行上述操作，那么我们总共使用的参数只需要 5*5*100 + 100（b 偏置量数目），其结果远小于全连接层的参数数目，能极大地减小网络参数数量。

2.2 稀疏连接

在使用全连接层进行模型训练时，每第 m+1 层的神经元会接收所有第 m 层的神经元的输出数据，并将这些数据一同执行计算。

而在卷积神经网络中采用了稀疏连接的方法，即：每个隐藏层 m 中的每一个神经元具有固定的 receptive field 接收范围，也称为感受野的大小，即每个神经元只会固定接收一定数目的上一层神经元的数据。表现在空间域中，也就是卷积核的大小；比如一个 3*3 大小的卷积核去执行卷积操作，得到的该像素点数据只跟原窗口的 9 个值相关，而在全连接层中下一层的每个输入都与该层的所有输出相关，故称为稀疏连接。

3. 请针对下述数据画出 ROC 曲线图。

3.1 第一步：按照属于‘正样本’的概率将所有样本排序

元组编号	类	概率
1	P	0.97
3	N	0.70
6	N	0.63
4	P	0.60
5	P	0.55
7	N	0.53
8	N	0.51
2	N	0.50
10	N	0.40
9	P	0.30

3.2 对每个样本计算坐标

样本 1：

	预测结果	预测结果
真实情况	正例	反例
正例	1	3
反例	0	6

X 轴坐标（false positive rate）= 0/（0+6）= 0

Y 轴坐标（true positive rate）= 1/（1+3）= 0.25

样本 3：

	预测结果	预测结果
真实情况	正例	反例

正例	1	3
反例	1	5

X 轴坐标（false positive rate）= 1/（1+5）= 1/6

Y 轴坐标（true positive rate）= 1/（1+3）= 0.25

样本 6：

	预测结果	预测结果
真实情况	正例	反例
正例	1	3
反例	2	4

X 轴坐标（false positive rate）= 2/（2+4）= 2/6

Y 轴坐标（true positive rate）= 1/（1+3）= 0.25

样本 4：

	预测结果	预测结果
真实情况	正例	反例
正例	2	2
反例	2	4

X 轴坐标（false positive rate）= 2/（2+4）= 2/6

Y 轴坐标（true positive rate）= 2/（2+2）= 0.5

样本 5：

	预测结果	预测结果
真实情况	正例	反例
正例	3	1
反例	2	4

X 轴坐标（false positive rate）= 2/（2+4）= 2/6

Y 轴坐标（true positive rate）= 3/（3+1）= 0.75

样本 7：

	预测结果	预测结果
真实情况	正例	反例
正例	3	1
反例	3	3

X 轴坐标（false positive rate）= 3/6= 0.5

Y 轴坐标（true positive rate）= 3/（3+1）= 0.75

样本 8：

	预测结果	预测结果
真实情况	正例	反例
正例	3	1
反例	4	2

X 轴坐标（false positive rate）= 4/6= 2/3

Y 轴坐标（true positive rate）= 3/（3+1）= 0.75

样本 2：

	预测结果	预测结果
真实情况	正例	反例
正例	3	1
反例	5	1

X 轴坐标（false positive rate）= 5/6

Y 轴坐标（true positive rate）= 3/（3+1）= 0.75

样本 10：

	预测结果	预测结果
真实情况	正例	反例

正例	3	1
反例	6	0

X 轴坐标（false positive rate）= 1

Y 轴坐标（true positive rate）= 3/（3+1）= 0.75

样本 9：

	预测结果	预测结果
真实情况	正例	反例
正例	4	0
反例	6	0

X 轴坐标（false positive rate）= 1

Y 轴坐标（true positive rate）= 1

3.3 坐标结果与 ROC 曲线

FALSE	TRUE
0	0
0	0.25
0.167	0.25
0.33	0.25
0.33	0.5
0.33	0.75
0.5	0.75
0.667	0.75
0.83	0.75
1	0.75
1	1

图 3.1 坐标结果

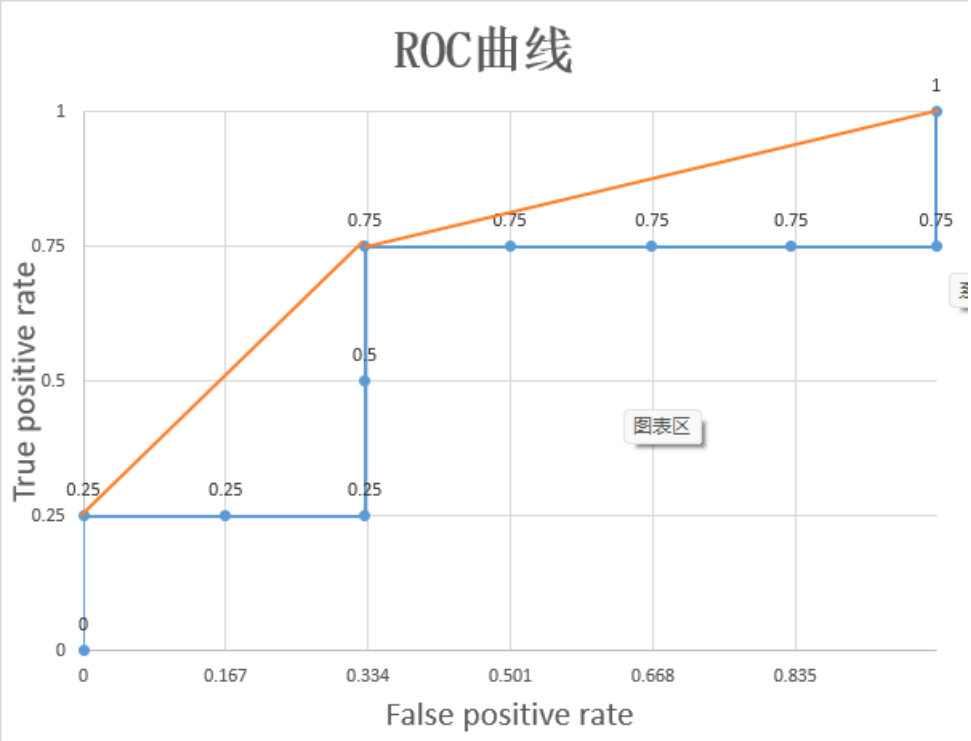
ROC曲线
True positive rate
False positive rate
图表区
1
0.75
0.5
0.25
0
0.167
0.334
0.501
0.668
0.835

图 3.2 ROC 曲线

4. 选题背景

2019 年 12 月，一场未知的病毒性肺炎疫情严重影响了中国武汉。该病毒很快被识别并命名为 SARS-CoV-2。世界卫生组织随后提出将其称为 2019 年冠状病毒肺炎（COVID-19）肺炎。截至 2021 年 12 月底，已有超过 200 个国家和地区受到影响，累计确诊病例超过 2 亿 8 千万例，病例数还在不断增加。这一严峻形势突出表明，迫切需要制定有效措施，控制这一流行病。

对 COVID-19 肺炎患者进行早期诊断以便及时治疗对于控制疫情至关重要。但受 COVID-19 影响的许多地区的医疗资源有限和患者人数众多，通常会导致诊断和医疗决策（如隔离或住院）的等待时间很长，这可能会增加交叉感染的机会并导致预后不良。

因此，寻找一种能快速有效识别新冠疫情感染情况的方法就成了一个研究方向。虽然核酸检测能有效检测新冠感染情况，但基于 CT 图像扫描更为迅速，为快速临床诊断提供条件。

在 COVID-19 大流行期间，第一手 CT 图像数据和临床数据集的可用性对于帮助指导临床决策，提供信息以加深对这种病毒感染的理解以及为系统建模提供基础（可能有助于早期诊断以进行及时的医疗干预）将是至关重要和重要的。实现这一目标的一个方法是创建一个开放获取的综合资源，其中包含个体患者的胸部 CT 图像，通过对 CT 图像建模高效区分是否感染新冠疫情，能够促进国际共同努力抗击 COVID-19 肺炎。

5. 需求分析

5.1 编写目的

为能够在规定时间内开发出符合预期标准的基于 numpy 的 DNN 神经网络新冠肺炎高效 CT 图像识别分类模型，编写此需求说明书。

模型要求能在保证模型微量化快速化的同时，对输入 CT 图像有高于 95%的新冠疫情预测准确率。完成这些目标需要一定的计划与明确的解决方案。

本需求分析说明书适用于该项目的客户方管理人员、需求分析员，用户文档编写者，项目管理员，项目产品开发人员，产品测试人员以及技术支持人员。

5.2 目标

设计一个基于 numpy 的 DNN 神经网络新冠肺炎高效 CT 图像识别分类模型，其要保证模型轻量化，训练简单化的同时保证高准确率。对输入肺部 CT 图像进行识别分析，给出其具体所属的类别，是新冠肺炎阳性、阴性或无信息。

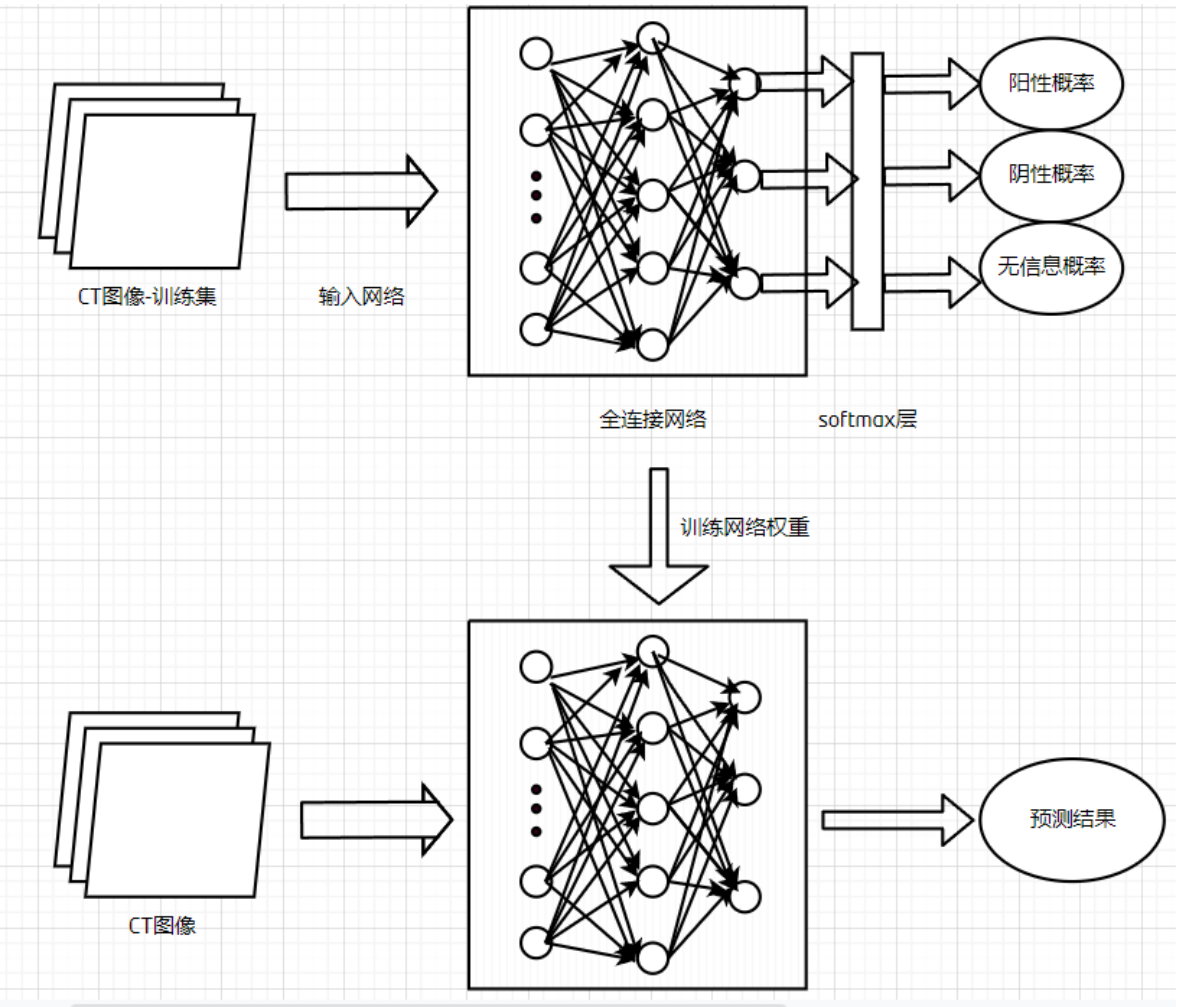

图 5.1 对模型的需求流程图

5.3 用户的特点

用户在使用该模型时，首先考虑到医疗需求需要尽可能准确分类，所以在模型测试时要保证高精度识别出结果，最好能将准确率控制在 95%以上。同时由于医疗设备特殊，需要模型尽可能轻量化便于实装；同时考虑到多种应用途径，一个合适的神经网络模型能拟合多种数据分类，所以还要求网络结构尽可能简单易用，同时易于修改和训练。

5.4 对性能的规定

5.4.1 精度

模型为了满足医疗诊断和疫情防控对时间和精度的高要求，要求在训练集上准确率 accuracy 不小于 98%、在测试集上准确率 accuracy 不小于 97%。

5.4.2 时间特性要求

处理时间：实装搭载模型后要求一张图片的网络处理时间不超过 0.1s（不包括读取图片时间），并能在总时间 0.5s 内完成全部读取、处理和输出结果流程。

5.4.3 灵活性

为保证模型使用灵活性，模型总大小应不超过 5MB，并能灵活更改模型隐藏层大小、层数与激活函数，便于依据同一网络模型针对不同数据进行拟合。

5.4.4 输入输出要求

输入：图片分辨率大小为 64*64，通道数为 3 的图片数据

输出：经 softmax 分类器输出的最大可能性的标签值，共 3 类信息，包含“Ni”（无信息）、“N”（阴性）、“P”（阳性）。

5.5 运行环境规定

运行配置要求：

（1）系统　Windows Vista 64 Bit Service Pack 2 以上 。

（2）CPU：Intel Core 2 Quad CPU Q6600（4 核,2.40GHz）以上

（3）内存需要 4GB。

（4）硬盘需要 2GB。

（5）Python 版本不低于 2.7.0

6. 解决方案概述

6.1 方案概述

基于上述背景及需求分析，我设计并实现了一种基于 numpy 的 DNN 神经网络新冠肺炎高效 CT 图像识别分类模型。该模型主体框架由纯 numpy 实现的神经网络各个步骤组成，包括定义网络的 w 与 b 的维度大小与初始化方式、前向传播、激活函数 Relu 的计算与求导、反向传播与梯度下降进行参数的更新。在实现全连接网络的基础上，还额外实现了数据的预处理、数据读取、accuracy 预测和最终 test 范例的脚本代码。

使用该神经网络模型进行 CT 图像新冠肺炎诊断，有如下 4 处优点与创新点：

6.2 优点&创新点 1-选题新颖贴合时事

在新冠疫情爆发近两年来，无数人内心被疫情牵动。宝贵的新冠病毒数据集被收录，背后是一个又一个患者与他身后的故事。本课程设计运用商务智能的预测性分析知识，结合深度学习与医疗生物专业，在与人们生命安全息息相关的医疗健康领域，想要通过机器学习的知识为驱散疫情贡献出自己的一份力量。

6.3 优点&创新点 2-模型精确度高

经网络训练测试，在训练集和测试集的 accuracy 分别超过了 99%和 97.8%，实现了网络对数据集较为精准的拟合，精确度已达到能够有效帮助诊断的大小，能够在一定程度上给予医疗人员诊断帮助。

6.4 优点&创新点 3-未借助深度学习框架，numpy 底层实现

该神经网络实现方式为 numpy 库进行底层函数编写，从最基本的神经网络各

个步骤组成，包括定义网络的 w 与 b 的维度大小与初始化方式、前向传播、激活函数 Relu 的计算与求导、反向传播与梯度下降进行参数的更新，不借助深度学习框架辅助网络训练。

6.5 优点&创新点 4-网络结构简单轻量化

该神经网络模型包含三个隐藏层，总参数大小不超过 5MB，部署方便轻量快捷。

7. 数据集介绍

7.1 数据集来源

数据集来源：http://ictcf.biocuckoo.cn[1]

7.2 数据集介绍

该数据集由华中科技大学医院（HUST-UH/ HUST-LH）收集，将单个 CT 切片分为三种类型：（1）非信息性 CT（NiCT）图像，其中肺实质未被捕获以供任何判断;（2）CT（pCT）阳性图像，其中可以明确识别与 COVID-19 肺炎相关的成像特征;和（3）阴性 CT（nCT）图像，其中两个肺部的成像特征与 COVID-19 肺炎无关。

其中，nCT 共包含 9979 张图片、pCT 共包含 4001 张图片、NiCT 共包含 5705 张图片。图片分辨率大小均为 512*512，位深度均为 8

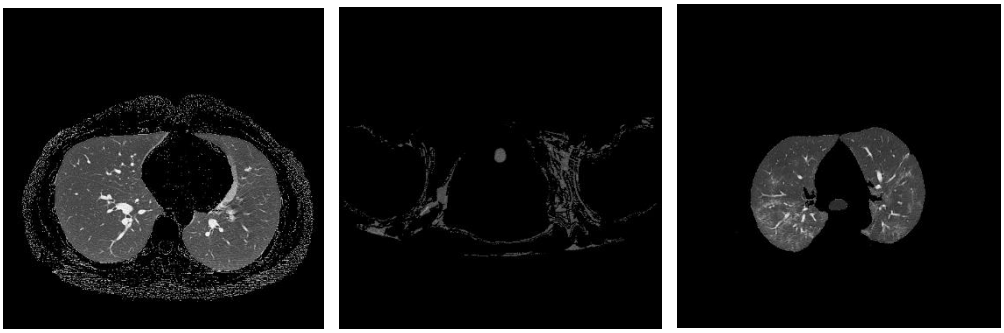

图 7.1 从左往右依次是 nCT0006、NiCT0015、pCT0040

7.3 数据集预处理

由于原图像大小过大，不适合全连接网络直接进行训练，于是采用图像下采样的方法，对图像进行了下采样，具体大小设置为 64*64 大小，保持通道数和位深度不变，处理后的图片可以直接全部读入内存中进行训练。

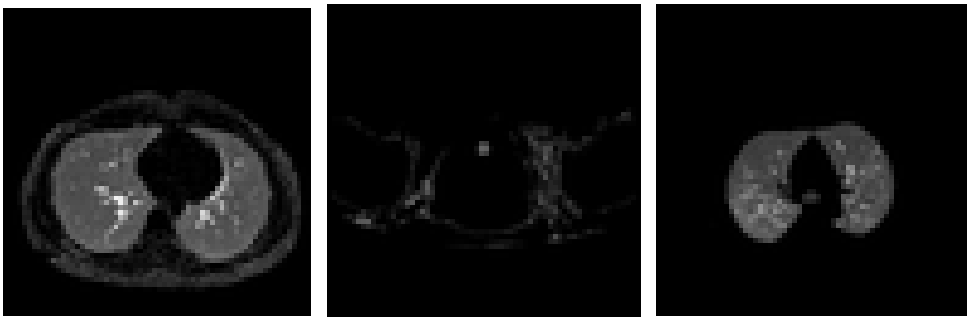

图 7.1 从左往右依次是下采样后的 nCT0006、NiCT0015、pCT0040

7.4 训练集与测试集分割

本次实验首先将训练集和测试集按 9：1 比例划分，然后在保存图片路径和 label 的 txt 文件中打乱顺序，形成三种 label 均匀分布的数据集合。得到 train 图片 17714 张，test 图片 1969 张。

8. 模型设置与关键代码实现

8.1 模型配置说明

该神经网络模型采用三个隐藏层的全连接网络架构，其网络神经元数目分别为：[64*64*64,50,30,20,10,config.numClass] （其中 config.numClass = 3）

```
layers_dims = [n_x,50,30,20,10,config.numClass]  #4个大小为 50、30、20、10的隐藏层
```

图 8.1

激活函数选用 Relu,输出层计算概率选用 softmax 层进行分类，与之对应选择 CrossEntropy 交叉熵损失函数作为优化的根本依据。

参数初始化方法为随机初始化，优化策略为梯度下降法，将训练集的 17714 张图片拼接为一个多维向量投入网络进行训练。

学习率采用 0.075、0.05、0.01 三个梯度，训练总 epoch 数量为 3000，具体训练策略在第 9 节会详细给出具体数据。

8.2 模型使用说明

程序入口在'code/model.py'文件中，可以手动选择是否加载预训练参数以及预训练参数的读取路径；在 layers_dims 中定义模型的隐藏层大小数量；在 L_layer_model 函数中定义训练的 epoch 数量和 learning rate，在训练完成后会自动保存参数并输出在测试集上验证结果。

```
#进行一次三十迭代的训练
parameters, costs = L_layer_model(train_X, train_Y, layers_dims, num_iterations = 400, print_cost = True)
```

图 8.2

8.3 参数初始化、前向传播与激活函数、反向传播

参数初始化中，传入一个表示网络层数与大小的 dim 向量，根据 dim 对每

一层的参数进行初始化，权重设置为接近 0 的随机数，bias 偏置设置为 0

```python
def initialize_parameters_deep(layer_dims): # for a deep network

    np.random.seed(3)
    parameters = {}
    L = len(layer_dims) # number of layers in the network

    for l in range(1, L):

        parameters['W' + str(l)] = np.random.randn(layer_dims[l],layer_dims[l-1]) * 0.01
        parameters['b' + str(l)] = np.zeros((layer_dims[l],1))

        assert(parameters['W' + str(l)].shape == (layer_dims[l], layer_dims[l - 1]))
        assert(parameters['b' + str(l)].shape == (layer_dims[l], 1))

    return parameters
```

图 8.3

线性前向传播，输入上一层的计算结果和权重以及偏置，输出计算的线性结果和保存的缓存便于反向传播。激活函数选择 Relu 实现。

```python
def linear_forward(A, W, b):

    Z = np.dot(W, A) + b
    cache = (A, W, b)
    return Z, cache
```

```python
def relu(Z):
    a = np.maximum(0,Z)
    cache = Z

    return a,cache
```

图 8.4

前向传播通过输入样本 X 和参数 parameters,输入一次前向计算的结果 AL 与计算过程中存储的缓存 cache；反向传播输入计算结构 AL，样本真实值 Y 和运算中存储的 cache，先对将交叉熵损失对输出层的激活函数 softmax 求偏导得到:

$$\frac{\mathrm{d}(\mathrm{cost})}{\mathrm{d}(\mathrm{z})}=\frac{\partial(\mathrm{cost})}{\partial(\mathrm{AL})}*\frac{\partial(\mathrm{AL})}{\partial(\mathrm{z})}=AL\ -\ Y$$

将这个结果带入到接下来的梯度计算中，得到每一层的梯度，输出梯度。

```python
def L_model_forward(X, parameters):

    caches = []
    A = X
    L = len(parameters) // 2

    for l in range(1, L):
        A_prev = A

        A, cache = linear_activation_forward(A_prev, parameters["W"+ str(l)], parameters["b"+ str(l)],
        caches.append(cache)

    AL, cache = linear_activation_forward(A, parameters["W"+ str(L)], parameters["b"+ str(L)], "softma
    caches.append(cache)

    return AL, caches
```

图 8.5 前向传播

```python
def L_model_backward(AL, Y, caches):

    grads = {}
    L = len(caches) # the number of layers
    Y = Y.reshape(AL.shape)

    current_cache = caches[L-1]

    dZ = AL - Y # softmax 对交叉熵损失函数的求导得出
    dA_prev_temp, dW_temp, db_temp = linear_backward(dZ, current_cache[0])

    grads["dA" + str(L-1)] = dA_prev_temp
    grads["dW" + str(L)] = dW_temp
    grads["db" + str(L)] = db_temp

    # Loop from l=L-2 to l=0
    for l in reversed(range(L-1)):

        current_cache = caches[l]
        dA_prev_temp, dW_temp, db_temp = linear_activation_backward(grads["dA" + str(l+1)], current_ca
        grads["dA" + str(l)] = dA_prev_temp
        grads["dW" + str(l+1)] = dW_temp
        grads["db" + str(l+1)] = db_temp

    return grads
```

$$\frac{d(z)}{}$$

图 8.6 反向传播

8.4 梯度下降与结果预测

梯度下降方法输入模型原参数、梯度和学习率，对每一层的参数进行更新。结果预测方法输入计算结果和真实标签，通过计算得到成功预测的准确率。

```
def update_parameters(params, grads, learning_rate):

    parameters = params.copy()
    L = len(parameters) // 2
    for l in range(L):

        parameters["W" + str(l+1)] = params["W" + str(l+1)] - learning_rate * grads["dW" + str(l+1)]
        parameters["b" + str(l+1)] = params["b" + str(l+1)] - learning_rate * grads["db" + str(l+1)]

    return parameters

def predict(AL,Y):
    AL -= np.max(AL, axis = 0, keepdims = True) #每一列减去最大值
    r = AL + Y #相机，计算出现的1的数量即为命中数
    result =(r == 1).sum()
    result /= Y.shape[1]
    return result
```

图 8.7

8.5 数据预处理、加载数据、开始训练

数据预处理脚本首先获取数据所在的目录，根据图片所在的目录名记录为它的 label 标签值，将其预处理后的图片保存在 miniCTscans 文件目录中，并将新路径与其 label 标签值拼接作为一行数据写入 txt 文件中，目的是方便后续读取数据集同时也读入 label。代码如下：

```
def Process():
    subfolders =  os.listdir('./dataset/CTscans')
    ftrain = open(config.trainPath,'a')
    fval = open(config.valPath,'a')
    for subfolder in
        os.mkdir('./d
        images = os.l
        threshold = (len(images) * 9) / 10
        count = 0

        for image in images:
            strcontent = "" # the text ready to write in a line
            strcontent += ("./dataset/miniCTscans/"+subfolder+"/"+image)
            count += 1

            img = Image.open("./dataset/CTscans/"+subfolder+"/"+image)
            out = img.resize((config.downSize, config.downSize),Image.ANTIALIAS) #resize image with high-quality
            out.save(strcontent,'png')

            if count > threshold:
                fval.write(strcontent+'_'+subfolder+'\n')
            else:
                ftrain.write(strcontent+'_'+subfolder+'\n')

        print('finish_' + subfolder)
```

```
(function) len: (__obj: Sized) -> int
Return the number of items in a container.
```

图 8.8 数据预处理

加载数据分图像数据与标签数据两部分加载。首先在按行读取 txt 文件后获得 string 格式的标签值与图片地址，根据图片地址读取图片数据后将其转为一维长向量并拼接到总训练数据的末端；标签数据则根据 config 中定义的 dictionary 转换成一维向量，同时也拼接到总 label 数据的末端。代码如下：

```
def load_data():
    trainImg = np.empty((config.trainNum,config.downSize,config.downSize,3))
    trainResult = np.empty((config.trainNum,config.numClass))

    testImg = np.empty((config.valNum,config.downSize,config.downSize,3))
    testResult = np.empty((config.valNum,config.numClass))

    countTrain = 0
    for line in open(config.trainPath):
        line = line.split()[0]
        if line != '':
            imgPath,label = line.split("_",1)
            img = cv2.imread(imgPath)[:,:,::-1]
            trainImg[countTrain] = img
            label = vectorized_result(config.classes[label])
            trainResult[countTrain,:] = label.T
            countTrain = countTrain + 1

    countTest = 0
    for line in open(config.valPath):
        line = line.split()[0]
        if line != '':
            imgPath,label = line.split("_",1)
            img = cv2.imread(imgPath)[:,:,::-1]
            testImg[countTest] = img
            label = vectorized_result(config.classes[label])
            testResult[countTest,:] = label.T
            countTest = countTest + 1
```

图 8.9 加载数据

```python
def L_layer_model(X, Y, layers_dims, learning_rate = 0.01, num_iterations = 3000, print_cost=False):

    np.random.seed(1)
    costs = []
    #parameters = initialize_parameters_deep(layers_dims) #随机初始化参数
    parameters = np.load('./model/DNN3000-1.npy', allow_pickle=True).item()
    start = time.time()
    thisTime = start

    for i in range(0, num_iterations):                                  #进行迭代

        AL,caches = L_model_forward(X, parameters)                      #一次向前传播
        cost = computer_cost_softmax(AL, Y)                                     #计算cost
        grads = L_model_backward(AL, Y, caches)                         #一次向后传播，计算梯度
        parameters = update_parameters(parameters, grads, learning_rate)    #根据梯度更新一次系数

        if print_cost and i % 20 == 0 or i == num_iterations - 1:

            elapsed = (time.time() - thisTime)
            thisTime = time.time()
            elapsed_all = (time.time() - start)

            print("Cost after iteration {}: {}".format(i, np.squeeze(cost)))
            print("this time: " + str(elapsed) + "  /  all time: " + str(elapsed_all))
            print("train accuracy = " + str(predict(AL,train_Y)))

        if i % 100 == 0 or i == num_iterations:
            costs.append(cost)

    return parameters, costs
```

图 8.10 开始训练

9. 模型训练与测试结果

9.1 多隐藏层训练结果

训练策略如下：

首先以 0.075 的学习率训练 200 个 epoch

再以 0.05 的学习率训练了 80 个 epoch

最后以 0.01 的学习率训练了 2720 个 epoch

训练过程日志文件记录如下（部分）：

	epoch	cost	train acc	test acc	lr
2	0	-0.33333	0.307836		0.075
3	20	-0.35329	0.507		0.075
4	40	-0.36533	0.507		0.075
5	60				0.075
6	80				0.075
7	100	-0.37893	0.507		0.075
8	120				0.075
9	140				0.075
10	160	-0.53309	0.507		0.075
11	180	-0.55537	0.741222		0.075
12	200	-0.51804	0.765778	0.762316	0.075
13	220	-0.58001	0.770012		0.05

图 9.1

1~200 lr=0.075

Cost after iteration 0: -0.3333346004435127
this time: 1.0496768951416016 / all time: 1.0496799945831299
train accuracy = 0.30783561025177825
Cost after iteration 20: -0.35329413437792206
this time: 18.602697134017944 / all time: 19.652414321899414
train accuracy = 0.5070001129050469
Cost after iteration 40: -0.365327088463533
this time: 16.76011562347412 / all time: 36.41253209114075
train accuracy = 0.5070001129050469
Cost after iteration 60: -0.3723441120139584
this time: 17.188909769058228 / all time: 53.6014461517334
train accuracy = 0.5070001129050469

图 9.2

2600~3000 lr = 0.01
Cost after iteration 0: -0.9723685131369504
this time: 1.1088085174560547 / all time: 1.1088109016418457
train accuracy = 0.9901772609235633
Cost after iteration 20: -0.9727379278292986
this time: 18.137423992156982 / all time: 19.24623727798462
train accuracy = 0.9902901659704189
Cost after iteration 40: -0.9730851169456489
this time: 17.369596242904663 / all time: 36.61583590507507
train accuracy = 0.9903466184938466
Cost after iteration 60: -0.9734156691099074
this time: 17.113216638565063 / all time: 53.72905468940735
train accuracy = 0.9903466184938466
Cost after iteration 80: -0.9737320222603727
this time: 14.0113365650177 / all time: 67.740394115448
train accuracy = 0.9904030710172744

图 9.3

```
this time: 17.01545548439026  /  all time: 244.70829510688782
train accuracy = 0.9908546912046968
Cost after iteration 320: -0.9767063589453303
this time: 17.23559856414795  /  all time: 261.94389629364014
train accuracy = 0.9908546912046968
Cost after iteration 340: -0.9769013819202448
this time: 16.142119884490967  /  all time: 278.0860188007355
train accuracy = 0.9908546912046968
Cost after iteration 360: -0.9770897167642181
this time: 18.62874746322632  /  all time: 296.7147681713104
train accuracy = 0.9908546912046968
Cost after iteration 380: -0.9772715101580419
this time: 15.675370931625366  /  all time: 312.3901422023773
train accuracy = 0.9908546912046968
Cost after iteration 399: -0.9774290877834428
this time: 17.30771780014038  /  all time: 329.6978621482849
train accuracy = 0.9908546912046968
train accuracy = 0.9908546912046968      2600~3000epoch , lr = 0.01
test accuracy = 0.97866937531742
```

图 9.4

Cost 变化如下：

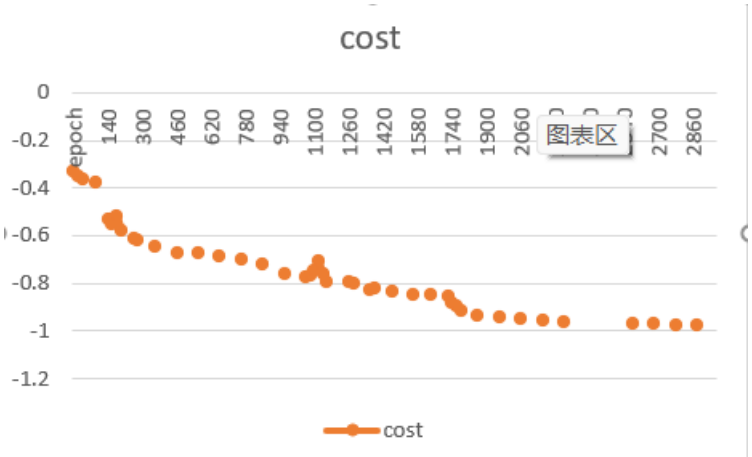

图 9.5

Accuracy 变化如下：

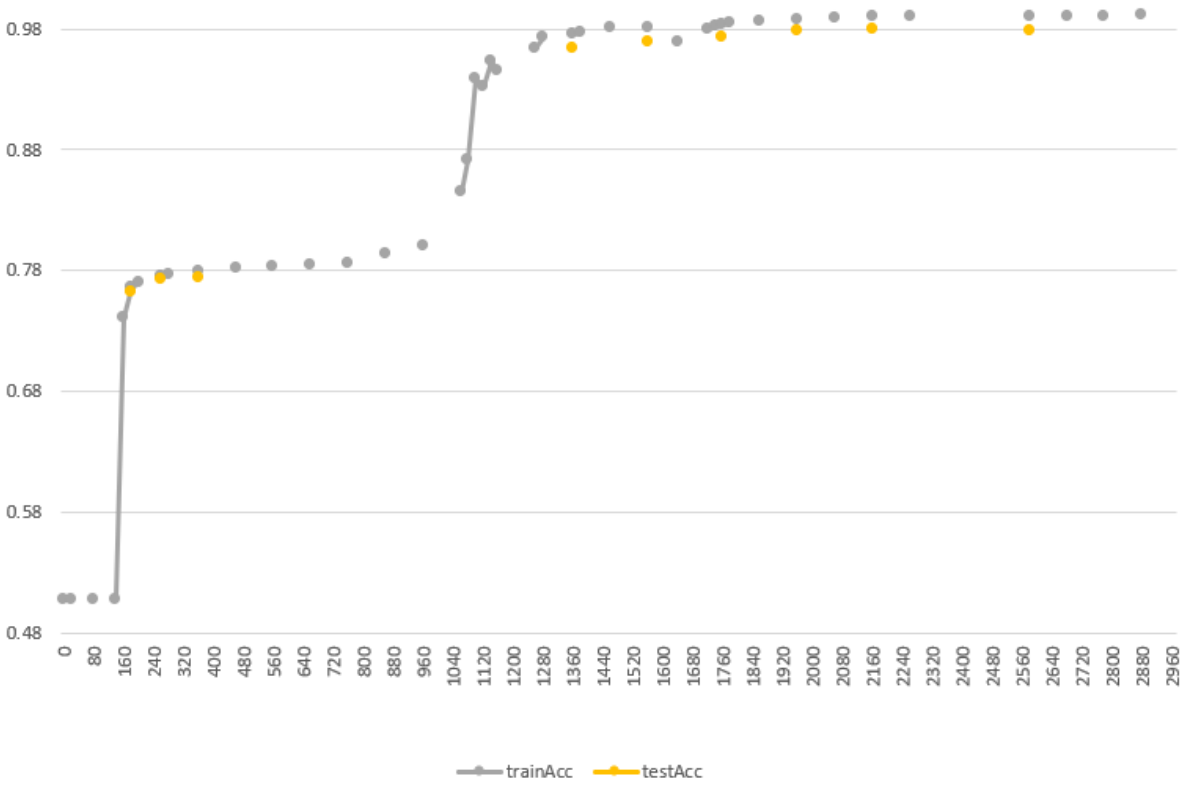

图 9.6

不难看出，在训练末期 trainAcc 达到了 0.99 以上，testAcc 也超过了 0.978。

9.2 单隐藏层训练结果

为保证实验完整性，另外选用大小为 30 的单个隐藏层神经网络进行训练，选用 0.0075 学习率进行 3000epoch 的训练，最终结果如下：

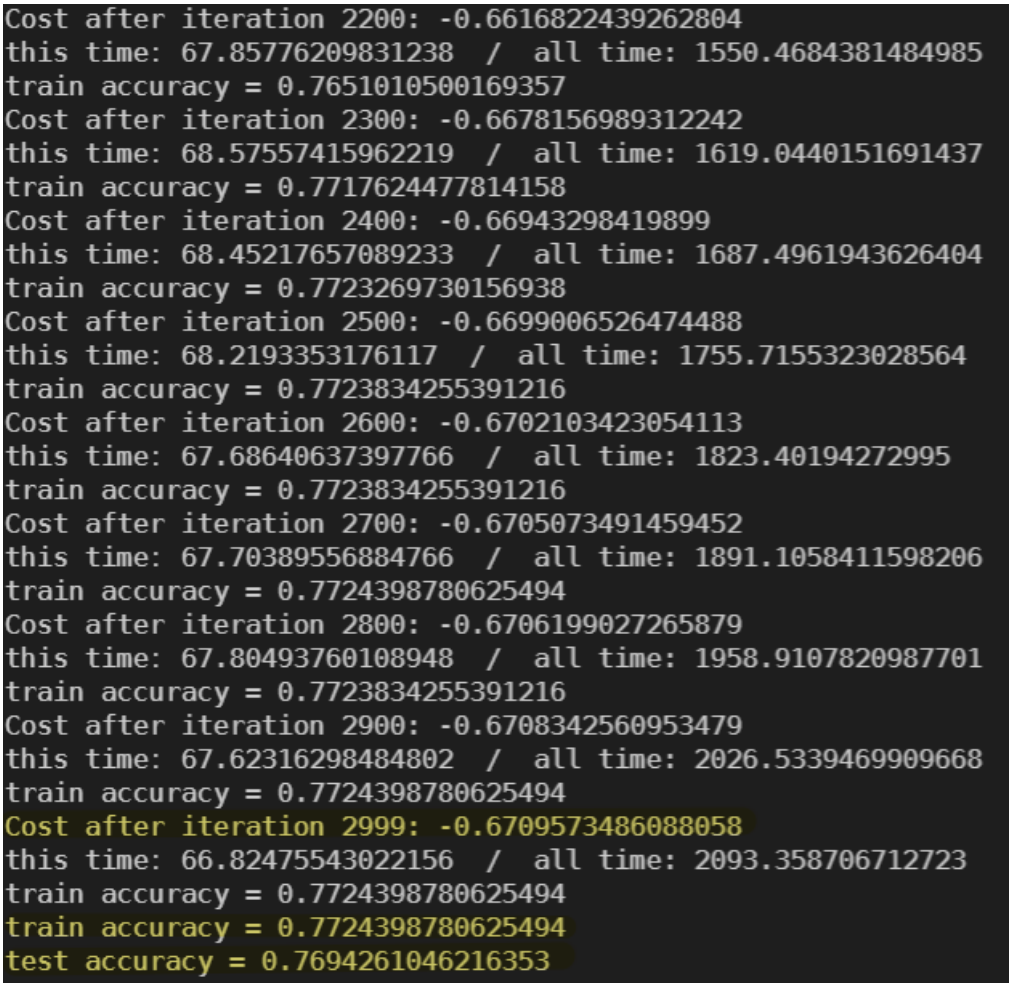

图 9.7 单隐藏层训练结果

可以发现单隐藏层得到结果远低于多隐藏层。

10. 总结与讨论

10.1 实验总结与验证

经上述实验证明，本次课程设计实现的 CT 图像分类模型，在经过适当训练后得到了较高的精度、较简单的模型结构，较小的模型参数大小与快速的收敛速度。在实验的最后我构建了一个应用该网络模型参数的脚本，利用参数对随机选择的三类图像进行了分类，毫无疑问都取得了正确结果和较高的概率。

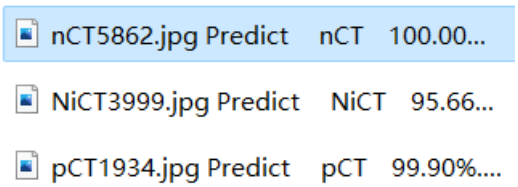

图 10.1 从左到右依次是图片名称、预测得到的标签值、预测概率

课程设计实验到此圆满结束，需求所需模型也成功构建。

10.2 改进与展望

这次多隐藏层的效果其实远远超出我的预期，我没有想到会有如此高的模型拟合度，毕竟第一次尝试单隐藏层只能训练到 0.78 不到的 acc 值。

能改进的地方还有很多：尝试使用 cnn 卷积网络、尝试更多层数、尝试更优质的梯度下降方法和参数初始化方法、使用正则化防止过拟合等等……

也许在不久的将来，这个模型能被不断完善，焕发光彩。

参考文献

[1] Ning, W., Lei, S., Yang, J. et al. Open resource of clinical data from patients with pneumonia for the prediction of COVID-19 outcomes via deep learning. Nat Biomed Eng 4, 1197–1207 (2020). https://doi.org/10.1038/s41551-020-00633-5

小记

感谢老师一学期如一日的辛勤付出。老师诙谐幽默的课堂氛围和透彻易懂的讲解方式令我陶醉于课堂之中。我知道自己在很多方面还有不足和没有领悟的地方，通过这次课设也渐渐让我学到更多，不只是模型的架构、训练与测试，更多的是对精益求精的思考和完善。通过这段时间的学习，我也更激发了对机器学习和大数据分析的学习热情。

再次感谢！

附录

该课程设计 github 仓库地址为我个人仓库：https://github.com/ssd777/CT-imageClassification-Covid19

预处理后的 64*64 大小图片数据集也过大，不便呈现。

训练日志文件与 excel 文件、模型文件已包含于压缩包中。