数字图像处理

作业设计报告

Final Report

报告题目：组织病理学肿瘤检测

学生姓名：廖本成 黄小虎 郭浩
指导教师：王兴刚
所在班级：种子 1601 班
成　　绩：

基于深度学习的组织病理学肿瘤检测报告

目录

第一章 绪论

1.1 问题重述

该问题需要我们对细胞组织图片中是否含有癌症细胞进行判定，癌症细胞只会出现在图片中央 32*32 区域，其中含有癌症细胞的为阳性（训练集取值为 1），不含癌症细胞的为阴性（训练集取值为 0）。

1.1.1 问题背景

组织病理学图像是用苏木精和曙红（H&E）染色的淋巴结的玻璃载玻片显微镜图像。这种染色方法是医学诊断中最广泛使用的方法之一，它产生蓝色，紫色和红色。深蓝苏木精与带负电荷的物质（如核酸和粉红素）结合到带正电的物质如氨基酸侧链（大多数蛋白质）。通常，细胞核染成蓝色，而细胞质和细胞外部分染成各种粉红色。

1.1.2 问题分析

该问题是一道二分类问题，我们通过设计的神经网络输出一维的数值对结果进行预测。这里，我们引入 ROC 曲线作为模型好坏的判定标准。ROC 曲线指受试者工作特征曲线 / 接收器操作特性曲线(receiver operating characteristic curve), 是反映敏感性和特异性连续变量的综合指标,是用构图法揭示敏感性和特异性的相互关系，它通过将连续变量设定出多个不同的临界值，从而计算出一系列敏感性和特异性，再以敏感性为纵坐标、（1-特异性）为横坐标绘制成曲线，曲线下面积越大，诊断准确性越高。

除 ROC 曲线外，我们还引入了 confusion matri 作为模型预测结果的其他呈现形式，confusion matrix 是真实值与测试值相比较得到的矩阵值，也是主流的体现分类性能的度量手段。

同时，由于癌症细胞只会出现在图片中央 32*32 区域，因此我们可以考虑对图片进行部分截取来训练模型。同时考虑癌症细胞与正常细胞在外形或颜色上会略有不同，我们可以尝试采用图片增强的方式来对图片进行预处理，获取更多特征信息。

1.2 方法介绍

在上面的分析中，我们发现对于目前主流图像二分类问题，主要采用深度学习，使用卷积神经网络来进行[1]，基于此我们调研了针对图片二分类识别问题相关的指标以及方法，解决此类问题应该有的完整的流程。

1.2.1 基本流程

我们发现对于此类问题有着成熟且完善的流程[2]，总结如下图所示：

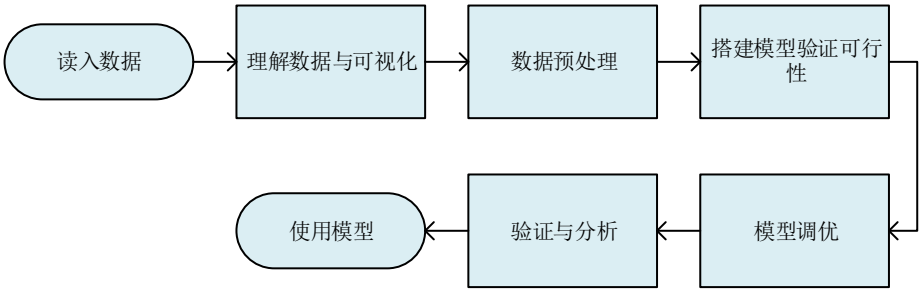


图 1-1 完整的机器学习的流程

1.2.2 实验使用主要模型介绍

(1) CNN[3]

CNN 是一种前馈网络，对大型图像处理有着出色的表现，是一个十分基础经典也很有效的方法

(2) finetuning[3]

Finetuning 指留用模型之前训练好的大多数参数，主要重构全连接层与 softmax 层来适应对应问题的分类，从而达到快速收敛

1) VGG16[4]

结构简洁，网络较深，在 ImageNet 上预训练好的网络有较好的特征提取的能力，能够较好的提升性能。

2) ResNet50[5]

通过引入恒等快捷连接（identity shortcut connection）达到了更深的深度，更低的复杂度。从而提升网络的性能。

1.2.3 工作简介[1]

我们根据 keras 提供的官方文档，自己搭建了框架，首先搭建了三层神经网络，验证了代码可以跑通后，输入输出没有问题，训练的模型的 loss 最终可以收敛后，抽取了 10000 张图片得到准确率为 79.3%，我们进一步实现了九层神经，并对其进行调参优化，最终结果为 86.3%，我们还采用了迁移学习使用 keras 自带的在 ImageNet 上预训练好的 VGG16 和 ResNet50，并对其进行微调，分别得到结果为 91.1%，60.2%。最终我们在全数据集上跑了进行调参过后的九层卷积神经网络以及 VGG16，九层卷积神经网络结果提交得分为 0.93。

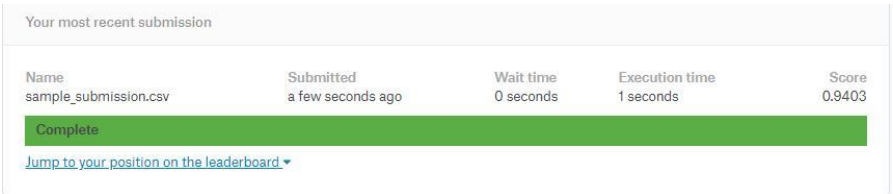


图 1.2-1 九层卷积神经网络结果

[1] 本次实验的 kernel 已设为 public，九层卷积神经网络网址 https://www.kaggle.com/liaobencheng/course-design-final

VGG 因为超时，故减少 VGG 模型可训练的层数，并提交，结果还在等待中。

1.3 人员分工

1.3.1 课程设计分工

廖本成
搭建好代码框架，实现三层卷积神经网络，九层卷积神经网络，ResNet50 以及 VGG16 的迁移学习。
黄小虎
九层卷积神经网络的实现，卷积神经网络参数的调优，图片截取大小性能优化探究。
郭浩
使用不同的优化器，同时调整学习速率，研究不同的优化器对结果性能的影响。

1.3.2 实验报告分工

廖本成
1.2，2.1，2.4
黄小虎
1.1，2.2
郭浩
2.3

第二章 设计实验及结果分析

2.1 基本框架搭建

在这里我们一开始参照 keras 官方提供的教程[6]以及网上相应的博客[7]很快的搭建起了整套的流程，但通过不断阅读 keras 的文档，以及对工程，为了能让每次实验的结果能够说明问题，也对代码不断进行优化，下面主要围绕代码基本流程以及除深度学习调参优化以外的代码优化展开。

2.1.1 表格读取

利用 pd.read_csv 函数读取 csv 文件生成 dataframe，发现训练集含有 130908 张 label 为 0 中间 36*36px 不含肿瘤的图片，89117 张 label 为 1 中间 36*36px 含肿瘤的图片。

```
0    130908
1     89117
Name: label, dtype: int64
                                     id  ...          class
0  f38a6374c348f90b587e046aac6079959adf3835  ...      a_no tumor
1  c18f2d887b7ae4f6742ee445113fa1aef383ed77  ...     b_has tumor
2  755db6279dae599ebb4d39a9123cce439965282d  ...      a_no tumor
3  bc3f0c64fb968ff4a8bd33af6971ecae77c75e08  ...      a_no tumor
4  068aba587a4950175d04c680d38943fd488d6a9d  ...      a_no tumor

[5 rows x 4 columns]
```

图 2.1-1 读入 dataframe01 构成以及改造后的展示

图上读入的直接读入 dataframe 只含有 id 和 label 这两列，导致我们后面只能使用 keras 的 flow_from_directory 的函数，从而需要建立目录，将图片 copy 到硬盘空间，然而 kaggle 只提供 4.9G 的 Disk 支持，无法支持全数据集的训练。通过查阅 keras 官方文档，还可以使用 flow_from_dataframe，而 kaggle 提供 13GB 的 RAM 是完全能够存下本次实验全数据集 6G 多大小的图片的，于是我们给读入的 dataframe 增加 filename 和 class 两列。

在这里因为 kaggle 官网提供的 kernel，经常会跑着跑着崩掉，导致实验得重头开始跑，而我们为了能够快速确定模型调参后的有效性，对总数据集进行采取 10000 个样本进行对比，一开始是随机采样，发现同一个模型重新跑一次，稳定的结果不一样，考虑到随机采样会导致每次重新跑随机抽取的 10000 个样本会改变，于是我们直接抽取读入的 dataframe 中前 10000 个从而保证每次样本是一样的，保证对比是能够说明问题

2.1.2 数据清洗

考虑到数据集很大，里面会有没有用的图片，比如全白以及全黑的图片，我们在这里设置了阈值 245 与 10，采用了图像处理的最大值滤波和最小值滤波滤除椒盐噪声的思想，最后得到结果如下：

2.1.3 载入数据集

先使用 keras 提供的 train_test_split 得到训练集与验证集，然后利用 keras 提供的函数 ImageDataGenerator 和 flow_from_directory，得到载入图片的给模型训练使用的训练集与测试集。后来因为想对全数据集进行训练，发现硬盘存储空间不够，故使用 flow_from_directory，改变了读入的 dataframe，一开始是起名字为 has tumor，no tumor，结果发现

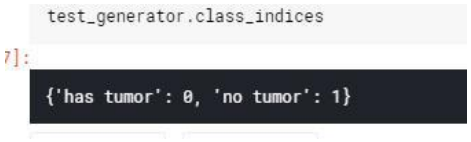


图 2.1-2 dataframe 的类别的索引

针对 class 的索引和图片标签刚好相反，推测 n 的字母序比 h 大故在后面，后改在标签前加上 a 和 b 的前缀，验证发现为正确，刚好与 label 对应。

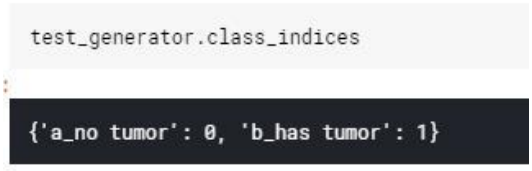


图 2.1-3 修改后 dataframe 的类别的索引

2.1.4 搭建模型与训练模型

搭建的三层模型结构如下图所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 94, 94, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 47, 47, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 47, 47, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 10, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0
_________________________________________________________________
dense (Dense)                (None, 256)               3277056
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 3,370,561
Trainable params: 3,370,561
Non-trainable params: 0
```

图 2.1-4 三层卷积神经网络结构

由 10000 张训练集的图片以 9：1 的比例再划分训练集与验证集得到的结果为 0.78，如下图所示

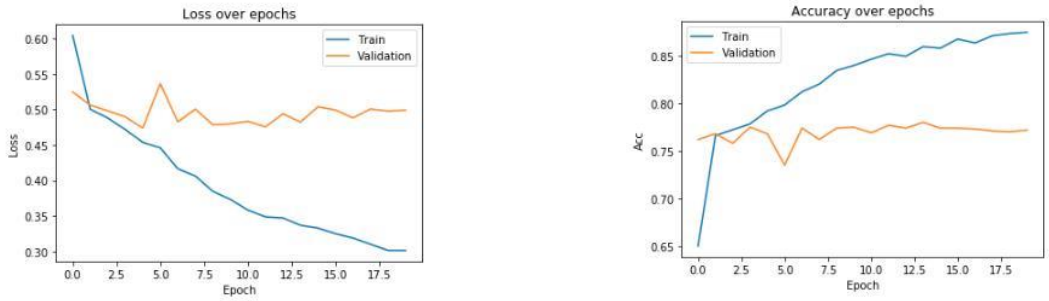


图 2.1-5 三层卷积神经网络模型结果

发现 20 个 epoch 后训练集上的 loss 并没有并没有收敛，于是我们加大 epoch，在原来的基础上继续跑：

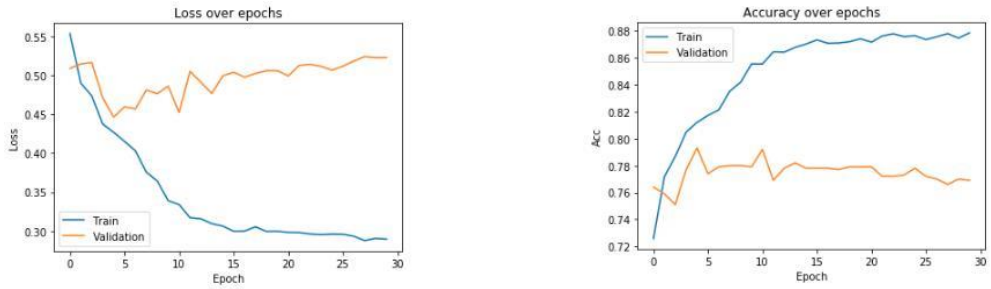


图 2.1-6 三层卷积神经网络模型继续训练 30epochs 结果

额外跑了 30epoch 后发现 loss 值收敛，故认为此时模型稳定下来得到了在训练集上得到了 loss 函数的极小值解，此时最终的结果为 0.793。

在验证了三层卷积神经网络的可行性后，我们尝试加深网络，搭建了九层神经网络，网络结构如下

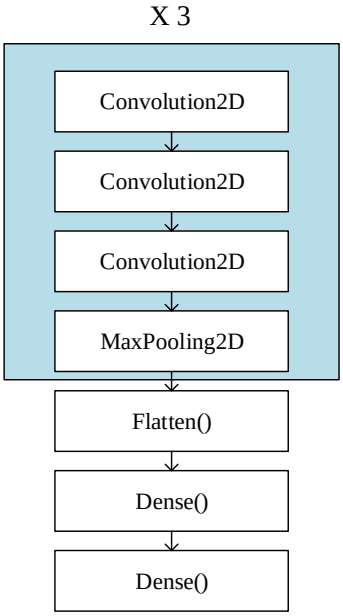


图 2.1-7 九层卷积神经网络模型

在与三层神经网络一样的情况下，得到准确率为 0.602，结果如下；

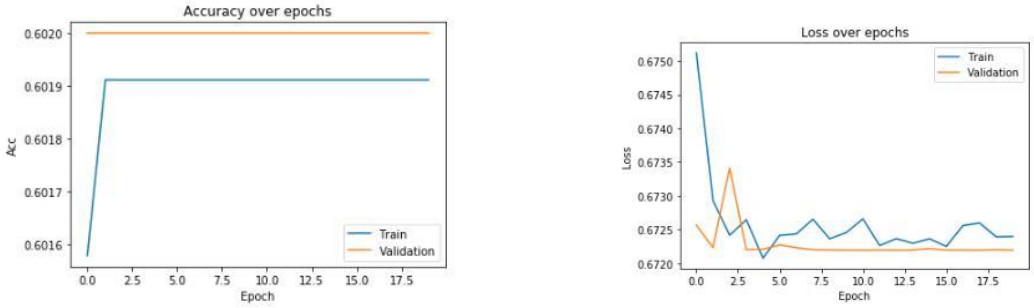


图 2.1-8 九层卷积神经网络模型结果(参数与三层卷积神经网络同)

发现准确率没有改变，但 loss 值也很快趋于稳定，故考虑到学习率太大导致无法找到最优点，考虑减小学习率从 0.001 减到 0.0001，结果如下：

准确率：0.816>0.602

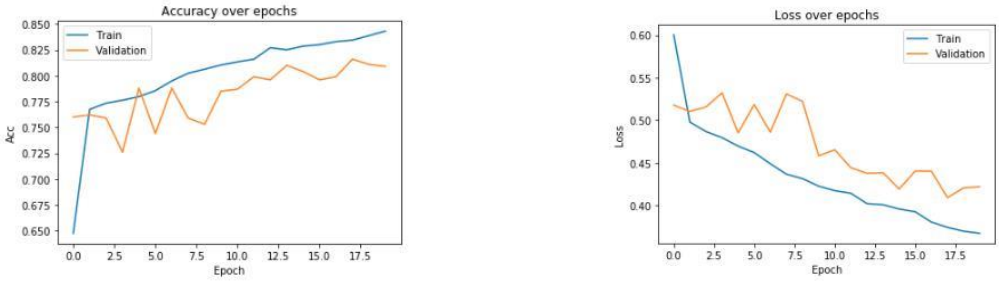


图 2.1-9 九层卷积神经网络模型结果(降低学习率后)

Loss 没有收敛，故考虑加大 epoch,最后 loss 收敛，准确率为 0.85

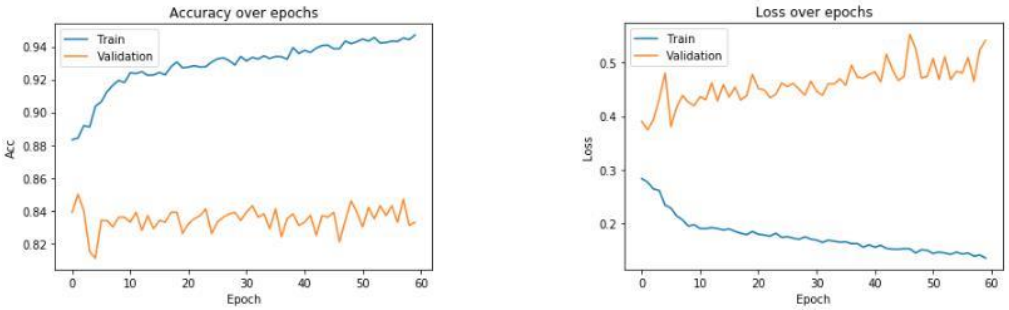


图 2.1-10 九层卷积神经网络模型收敛结果(降低学习率后)

由上图可知，最后训练集与验证集的准确率有较大区别，故考虑到有过拟合的现象，故从这里，我们决定单独研究如何防止过拟合。后来我们改变了优化器从 Adam(lr = 0.001)改到了 SGD(lr = 0.001)，在同样的训练集与验证集上，发现最后结果不同，于是我们又针对优化器与学习率的选择去设计实验寻找最优的方式，SGD 优化器准确率为 0.817 结果如下：

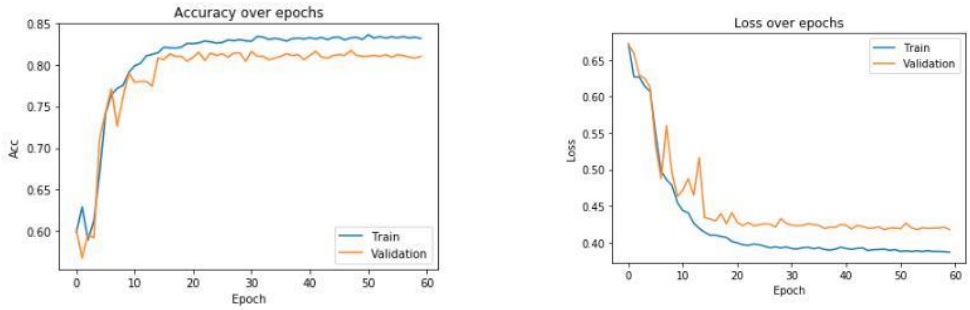


图 2.1-11 九层卷积神经网络模型收敛结果(改变优化器后)

2.1.5 评估与分析

对于二分类问题，我们采用 AUC，混淆矩阵来评价模型的性能。

2.2 防止过拟合及模型调优

2.2.1 增加 dropout

在基本的 9 层模型上我们训练 9000 个样本，其中设置 1000 个为验证集，epoch 设置为 20，可以得到准确率图像为：

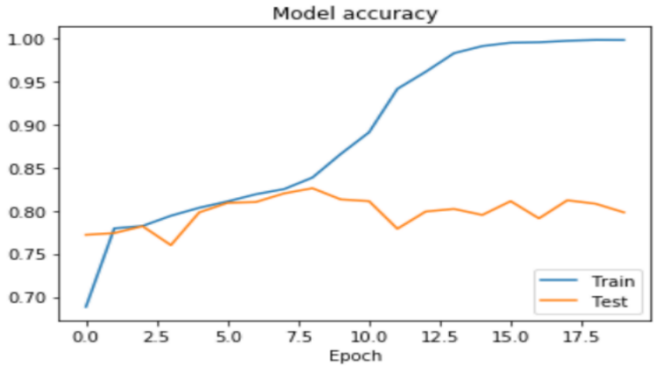


图 2.2-1 基础 9 层模型准确率

不难看出在训练集上模型的准确率在第 15 个 epoch 后逼近了 1，而验证集上模型的表现与训练集差的很远，出现了过拟合的情况。通过阅读[8]，我尝试在模型中采用 dropout 的方法去解决这个问题，得到的准确率图像如下图所示：

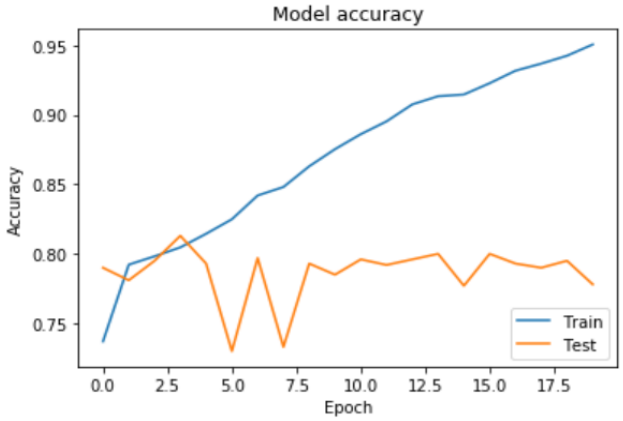


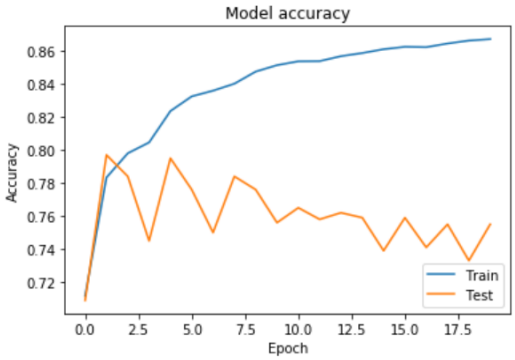
图 2.2-2 增加 dropout 后的准确率

我们可以看到，在增加了 dropout 策略之后，经过 20 个 epoch 训练集上的准确率与之前相比稍有下降，训练集和测试集的准确率差值略有降低，过拟合问题稍有改善。

2.2.2 正则化方法

根据[8]我又在模型上尝试通过正则化的方式继续改善过拟合的问题，对 L1 正则化分别选取正则化强度为 0.0001，0.001，0.01 的情况，得到的准确率结果如下图所示

图 2.2-4 α=0.001



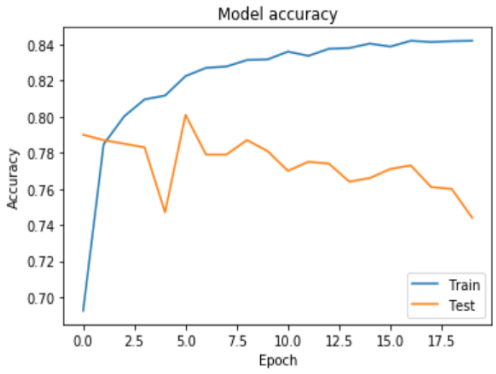


图 2.2-3 α =0.0001

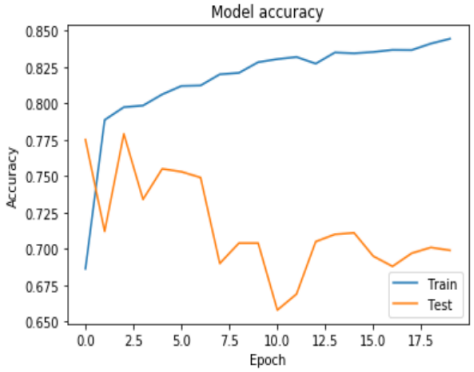


图 2.2-5 α =0.01

随着正则化强度的增加，训练集上的准确率在逐渐地降低，同时测试集上的准确率也在发生变化。当正则化强度增加到 0.01 时，我们可以看到测试集上的准确率开始有明显的下降，出现了欠拟合的情形。相比之下，当 α=0.001 时，训练集的准确率曲线相对比较平稳，与测试集的准确率之差也最小，故选择 0.001 作为正则化强度的参数。

2.2.3 调整 batch_size 改善测试集表现

在前两步实验基础之上为了进一步改善测试集的表现，通过阅读[9]，我们尝试改变了模型的 batch_size，将 batch_size 从原来的 10 分别改为 32，64，128，获得的准确率结果如下图所示：

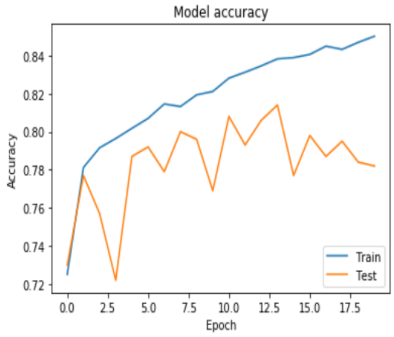


图 2.2-6 batch_size=32

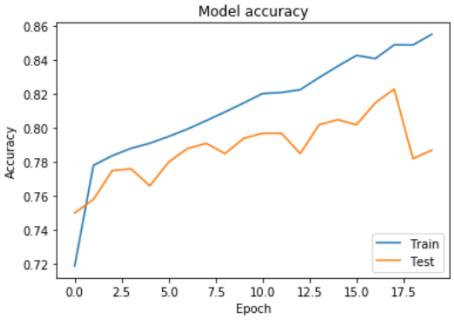


图 2.2-7 batch_size=64

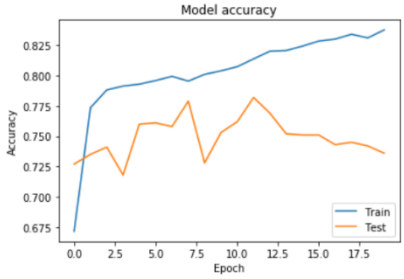


图 2.2-7 batch_size=128

通过对准确率曲线的对比，我们可以发现，当 batch_size 取 64 时训练集与测试集的曲线更为接近，相较 32 与 128 而言是更好的选择。

2.2.4 调整权值初始值改善测试集表现

通过以上三个步骤的优化，我们已经得到相对较好的训练模型，在此之上，我们阅读了[10]，尝试通过对神经网络权值的不同初始化方式得到更好的训练结果，除了默认的权值初始化 glorot_normal 方式外，我们还尝试了 lecun_normal, lecun_uniformal 权值初始化方式，得到的准确率图像如下图所示：

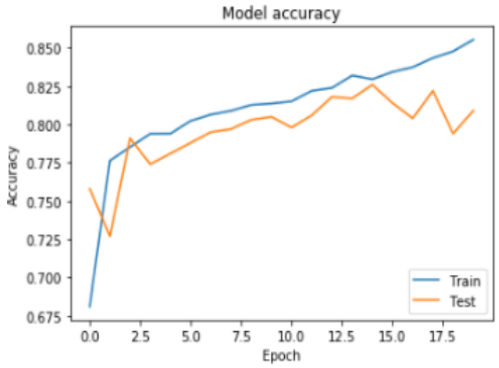


图 2.2-8 lecun_normal

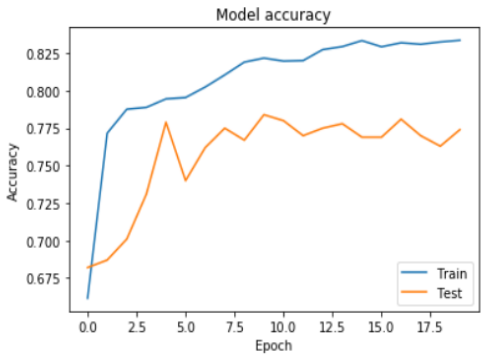


图 2.2-9 lecun_uniformal

从上图我们不难看出，当采用 lecun_normal 的权值初始化方式时，相较默认参数与 lecun_uniformal 的初始化方式，我们可以发现，训练集与测试集的准确率差距相较最低，并且测试集的准确率更高，因此我们选择使用 lecun_normal.

2.2.5 图片截取优化模型性能

由于癌症细胞只会出现在图片的中央 32*32 区域，而原图为 96*96，因此我们考虑对图片进行截取，我们分别将图片截取到 96*96，84*84，72*72，60*60，48*48，32*32，进行训练，这里我们使用 ROC 曲线对模型训练的结果进行评价，曲线下面积越大代表模型越好。

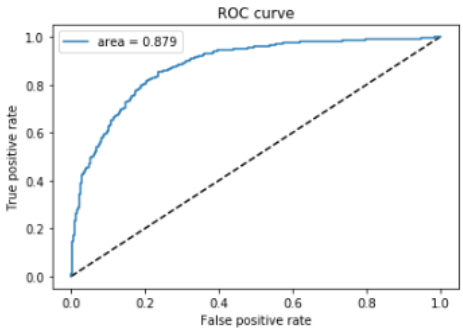


图 2.2-10 96*96

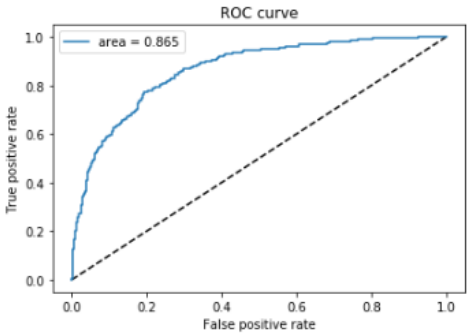


图 2.2-11 84*84

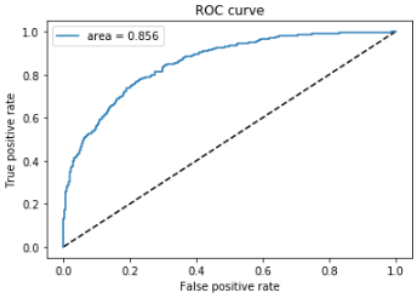


图 2.2-12 72*72

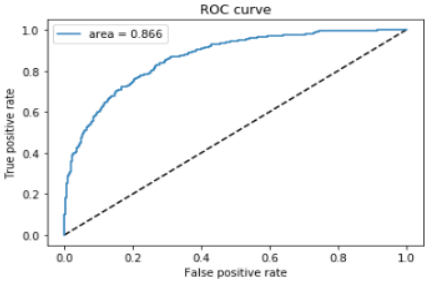


图 2.2-13 60*60

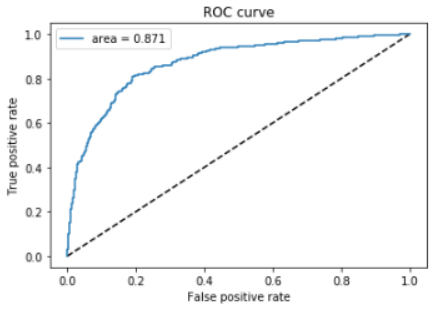


图 2.2-14 48*48

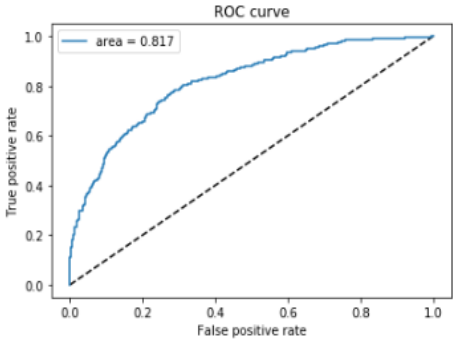


图 2.2-15 32*32

以上数据汇集于下图：

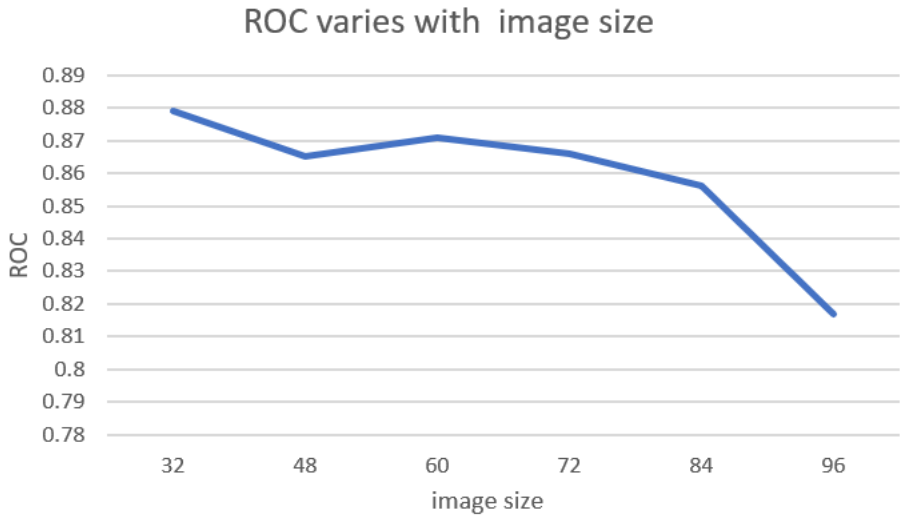


图 2.2-16 ROC 取值于图片大小关系图

通过上图，我们可以看出在被训练图片截小之后，ROC 的值也在跟着变小。因此我们可以得到合理的结论：癌症细胞虽然只存在于中央 32*32 区域，但是周边的组织具备很多关于癌症细胞信息，因此在逐渐截取图片后并没有让我们的结果变得更好。因此最终我们选择的图片大小是 96*96。2.3 优化器及学习率调优

2.3 优化器及学习率调优

2.3.1 代价函数

J(θ)称为代价函数（也叫目标函数），其中θ是模型的参数，其中$\theta \in \mathrm{R}_d$，J(θ)越小，代表模型的输出越接近真实值，因此我们需要求解 J（θ）的最小值。

2.3.2 梯度下降优化算法

梯度下降算法的核心是为了最小化代价函数 J(θ)，它的思路是在每次迭代中，对每个变量，按照 J（θ）在该变量梯度的相反方向来更新参数值，也即沿着 J(θ)的超平面的斜率下降的方向前进，直到遇到谷底。

2.3.3 几种梯度下降优化算法对比[11]

2.3.3.1 批量梯度下降（Batch gradient descent）

$$\theta=\theta-\eta*\nabla_{\theta}\mathrm{J}(\theta)$$

其每次使用全部的训练集样本来更新模型参数，使用减法即梯度的相反方向。

2.3.3.2 随机梯度下降（Stochastic gradient descent）

$$\theta=\theta-\eta*\nabla_{\theta}\mathrm{J}(\theta,xi,yi)$$

和 BGD 每次使用全部训练样本相比，SGD 每次取一个样本数据进行参数更新，大大减少的计算的冗余度，因此学习的速度会非常的快，并且可以进行在线更新。

以上这两种都是学习率固定的优化算法，但是固定的学习率会带来不好的结果，因此有以下的改进算法

2.3.3.3 Momentum

为了去理解后面几种梯度下降算法的思路，我们首先需要知道动量算法的思想。

$$v_t=\gamma v_{t-1}+\eta\nabla_{\theta}\mathrm{J}(\theta)$$
$$\theta=\theta-v_t$$

它的思想在于通过累积前面的梯度计算值，使得在后面的更新过程中，在梯度指向方向相同的方向逐渐增大，在梯度指向改变的方向逐渐减小。这样使得收敛速度加快。

2.3.3.4 RMSprop

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

当前时间的梯度均值是基于过去梯度均值和当前梯度值平方的加权平均，其中 0.9 是经验值

2.3.3.5 Adam

类似 RMSprop 存储过去梯度平方均值外，Adam 也存储过去梯度的均值。

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

同时使用偏差纠正系数，来修正一阶矩和二阶矩的偏差：

$$\hat{m} = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v} = \frac{v_t}{1-\beta_2^t}$$

然后整体的更新规则为：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t$$

2.3.4 学习率调整

	学习率 大	学习率 小
学习速度	快	慢
使用时间点	刚开始训练时	一定轮数过后
副作用	1.易损失值爆炸；2.易振荡。	1.易过拟合；2.收敛速度慢。

2.3.5 优化器数据分析

在我们的模型设计中，我们固定模型，通过使用不同的优化器类型和不同的学习率，来观察不同的优化器和不同的学习率对模型预测准确率的影响。

下图中从左到右，学习速率从 0.01 变化到 0.001，上面为 loss 曲线，下面为对应的 accu

曲线，横轴为迭代次数 epochs，蓝色的为 train loss，棕色的为 val loss.

2.5.1 SGD

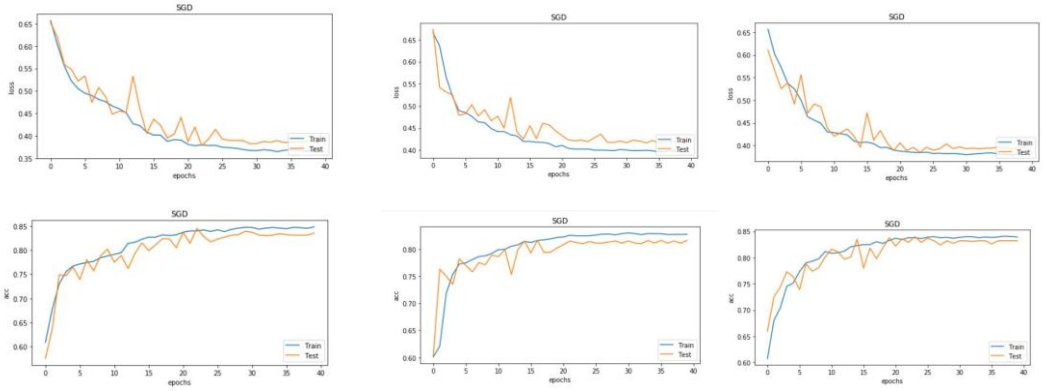


图 2. 3-1 SGD 优化器 loss 和 accu 的变化

从图中数据可以看出，随着 learning rate 的改变，曲线的波动程度也在发生变化。

1. 当 learning rate 比较大的时候，曲线的波动程度也比较大，learning rate 比较小时，线的波动程度会有所减小。

这是因为 SGD 的代价函数参数更新的方式为：$\theta=\theta-\eta*\nabla_{\theta}\mathrm{J}(\theta,xi,yi)$

其 learning rate 也就是η在每次的迭代过程中不会进行动态改变，而是固定的，因此当η比较大的时候，每次的步长都是很大的，导致在”谷底”来回的进行波动而不能收敛到最优点，而当η比较小时，由于每次的步长比较小，因此在谷底便可逐步的收敛到最优点，而减少了来回的震次数。

2.5.2 RMSprop

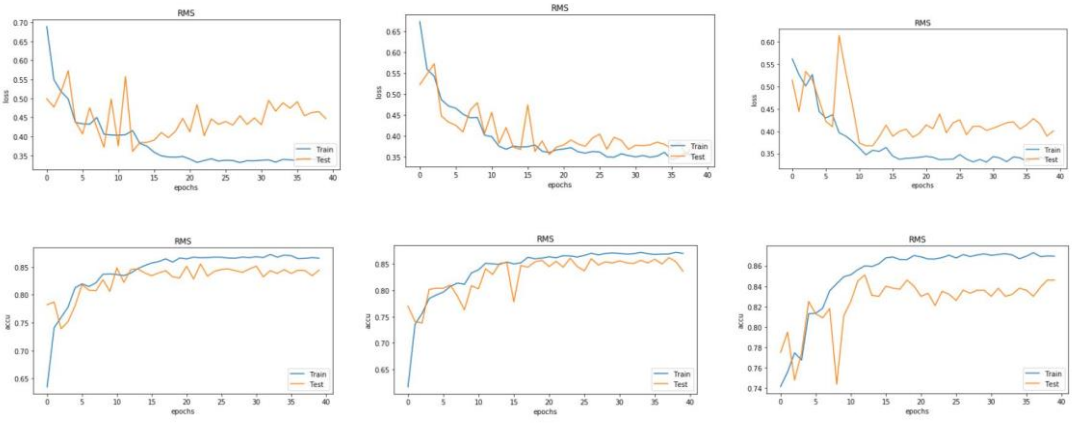


图 2. 3-2 RMSprop 优化器 loss 和 accu 的变化

2. 根据图中可以发现其收敛速度比 SGD 要快，loss 很快就收敛到了一个比较好的一个值。

由于 RMSprop 在每次迭代更新参数时为：$\theta_{t+1}=\theta_t-\frac{\eta}{\sqrt{E[g^2]_t+\varepsilon}}g_t$

全局学习率逐参数的除以经过衰减系数控制的历史梯度平方和的平方根，使得每个参数的学习率不同。其引起的效果就是在平缓的空间上会取得更大的步进，陡峭的方向上变化的平缓，从而加快训练速度，所以其收敛速度加快。

2.5.3 Adam

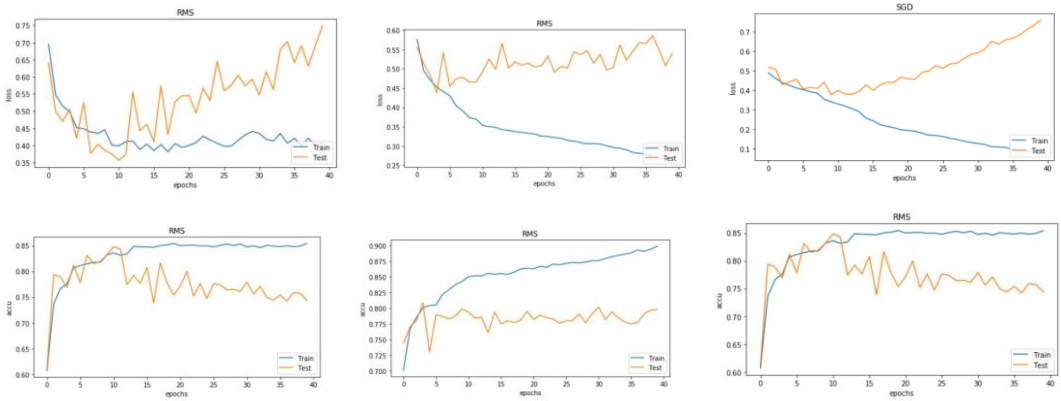


图 2. 3-3 Adam 优化器的 loss 和 accu 的变化

3. 从图中可以发现 Adam 相对于上面的两种优化器，其学习速率为 0.01 左右（也即第一张图）时，其收敛速度非常的快。

Adam 在每次的迭代之后更新参数时：$m_t=\beta_1 m_{t-1}+(1-\beta_1)g_t$

$v_t=\beta_2 v_{t-1}+(1-\beta_2)g_t^2$

在 Adam 的算法中不仅考虑了 RMSprop 算法中的梯度一阶矩均值，也考虑了梯度的二阶矩均值，通过这两个方面为不同的参数设计独立的、自适应性学习率，由于其学习速率有自适应的一个过程，因此其收敛速度会非常快。

4. 同时我们观测到第二张图明显其还没有收敛。

这是因为第二张图的学习速率为 0.0005，其学习速率过低，导致收敛速度过慢。第三张图则是学习速率过低导致了过拟合。

2.4 尝试其他预训练好的模型

在这里我们主要尝试了 VGG16 和 ResNet50 这两个 keras 库自带的在 ImageNet 上预训练好的模型。

因为我们想看看在其他数据集上预训练好的网络具有很强的提取特征的能力，而其提取特征的能力能否很好应用于针对本问题的肿瘤检测。

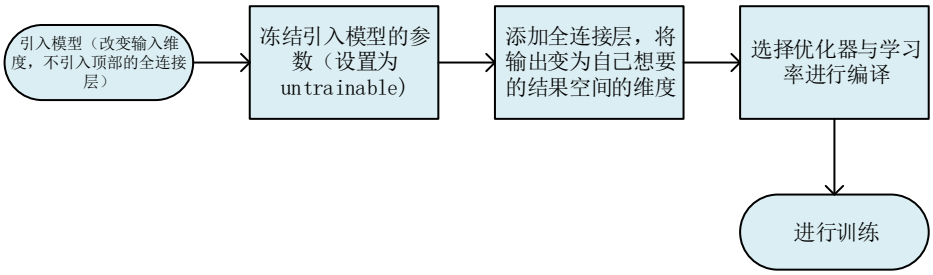


图 2.4-1 迁移学习的流程图

为了很好的比较两个模型的性能，我们将两个模型先后依次训练分别存到 VGG16_model.h5 和 resnet50_model.h5 中去，这样可以在同一个训练集和验证集上跑，从而使跑出的结果具有可比性。

数据处理与载入：

1. 取 10000 张图片
2. 数据清洗
3. 图片增强（像素归一化，水平翻转，垂直翻转，角度旋转 40°）
4. 优化器（优化器与学习率均设置为相同）

最终结果：

1. VGG

准确率：0.911

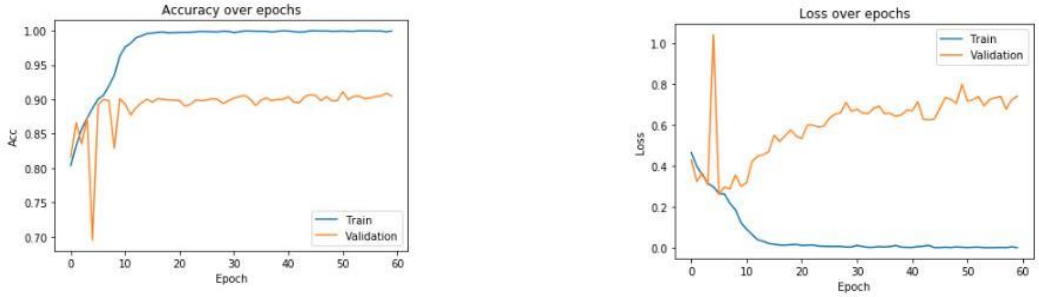


图 2.4-2 VGG16 迁移学习的结果

2. ResNet50

准确率：0.602

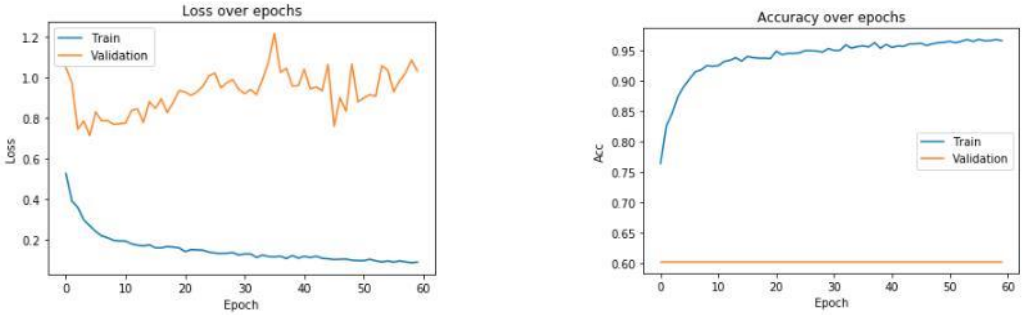


图 2.4-3 ResNet50 迁移学习的结果

我们发现在模型经过训练后 loss 收敛时，训练集上准确率十分高接近 1，但在验证集上一直不变。后面改变学习率，发现验证集上结果还是 0.602，考虑到 ResNet50 十分深，推测为因为数据量小导致过拟合，出现此种情况。

第三章 总结

廖本成：

通过这一次课设，我广泛的阅读博客与官方文档，了解到了机器学习面向工程问题的一套完整的流程，在调通模型，证明模型的有效性，提高模型的性能，经常需要调用官方函数，以及调参，让我对之前课堂上所讲的东西有了更为透彻的理解。让我很惊奇的一点就是用预训练好的 VGG16 时，在 10000 张图片上一开始准确率就很高，有 71%，而且最后在验证准确率也有 91%，这说明不同训练集训练出来的特征提取能力也能用于其他问题的分类，这是不是意味着，看似不相干的事物，可能本质上由某种并非直观上，肉眼能理解的特征所决定的，如果能够解释清楚这些，人类对自己对自然认识将超越肉眼凡胎，进入更高维度去感知与理解。还有一个很有意思的问题，就是换个优化器，调整学习率将会对模型收敛以及最终的结果有很重大的影响，如果能够在这种多维空间里找到，优化理论，将更多的数学分析，统计分析结合进来，可能将会带来更大更显著的进步！

黄小虎：

通过本次课程设计，我接触到了机器学习的一般建模过程，对卷积神经网络的设计及实现有了更清晰的认识，对实验过程中出现的问题例如过拟合，欠拟合等有了自己的应对措施。在实验过程中，出现的问题主要有以下几个：1、在利用验证集对模型进行验证时，对验证集也做了预处理，我们这样会影响验证集的数据分布，没有做到有效验证。2、在对模型权值进行初始化时曾经全都初始化为 0，导致模型没有被训练。3、在一开始做不同参数对模型影响比较的时候，每次都从数据集中重新随机获取数据，这样由于随机数据的影响，可能无法对性能进行正确地对比，应该采用同一批数据。

郭浩：

在这次实验中，我首先对整个代码的框架进行了熟悉，对 TensorFlow.keras 的框架有了一定的熟悉之后，我便锁定了一个方向—优化器的性能分析。首先我尝试了不同的优化器类型，发现不同的优化器对结果的影响不同，同时学习率的变化也会对优化器的性能产生影响。因此我便去找了讲解各种不同优化器原理的博客和论文。通过数学上公式的推导，理解优化器的工作原理，同时通过实验，观察分析学习率的变化对不同优化器带来的结果上的影响。通过这样的实验，使得我对损失函数的理解更加的加深，以及对学习率的含义理解的更加深刻，对学习速率变化的策略也有了自己的理解。

参考文献

[1] https://keras-cn-docs.readthedocs.io/zh_CN/latest/blog/image_classification_using_very_little_data/.

[2] https://www.kaggle.com/qitvision/a-complete-ml-pipeline-fast-ai.

[3] https://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C.

[4] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385.

[6] https://keras-cn-docs.readthedocs.io/zh_CN/latest/blog/image_classification_using_very_little_data/.

[7] https://www.kaggle.com/vbookshelf/cnn-how-to-use-160-000-images-without-crashing.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky. A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014) 1929-1958.

[9] Pavlo M. Radiuk. Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. Information Technology and Management Science. December 2017, vol. 20, pp. 20–24.

[10] Samuel L. Smith∗ , Pieter-Jan Kindermans∗ , Chris Ying & Quoc V. Le. DON'T DECAY THE LEARNING RATE, INCREASE THE BATCH SIZE. Published as a conference paper at ICLR 2018.

[11] Sebastian Ruder. 15 Jun (2017) An overview of gradient descent optimization algorithms∗.