

**HUAZHONG AGRICULTURAL UNIVERSITY**



课程项目报告

|  |  |  |  |
| --- | --- | --- | --- |
| 题 目 | **基于卷积神经网络的食物图像识别系统的设计与实现** | | |
| 姓 名 | 邢广杰 | 学 号 | 2017308210404 |
| 专业班级 |  |  |  |
| 计算机科学与技术1803 | | |
| 指导老师 | 彭辉 |  |  |

中国·武汉

2020年 12月

目录

[**第一章 项目系统分析与设计** **5**](#_Toc2134517762_WPSOffice_Level1)

[1.1 项目背景 5](#_Toc1096804109_WPSOffice_Level2)

[1.2 项目需求分析 6](#_Toc2134517762_WPSOffice_Level2)

[1.3 开发环境和工具 7](#_Toc1125702799_WPSOffice_Level2)

[1.4 系统设计思路 10](#_Toc356012723_WPSOffice_Level2)

[**第二章 相关算法和技术理论** **12**](#_Toc1125702799_WPSOffice_Level1)

[2.1 深度学习 12](#_Toc616394919_WPSOffice_Level2)

[2.2 人工神经网络 13](#_Toc288290505_WPSOffice_Level2)

[2.3 ReLU函数 14](#_Toc575409903_WPSOffice_Level2)

[2.4 反向传播和梯度下降 15](#_Toc795377280_WPSOffice_Level2)

[2.5 Adam算法 17](#_Toc1967726032_WPSOffice_Level2)

[2.6 小批量梯度下降 18](#_Toc323256024_WPSOffice_Level2)

[2.6 卷积神经网络 18](#_Toc1977852105_WPSOffice_Level2)

[2.7 残差网络 19](#_Toc860956822_WPSOffice_Level2)

[2.8 SqueezeNet 24](#_Toc356493868_WPSOffice_Level2)

[2.9 Softmax 回归 27](#_Toc113064346_WPSOffice_Level2)

[2.10 数据预处理 28](#_Toc1896919274_WPSOffice_Level2)

[2.11 数据增广 29](#_Toc2127498403_WPSOffice_Level2)

[2.12 Mixup 34](#_Toc1262936671_WPSOffice_Level2)

[2.13 迁移学习和预训练 35](#_Toc448262549_WPSOffice_Level2)

[2.14 SSIM图像结构相似性 37](#_Toc576027367_WPSOffice_Level2)

[**第三章 具体实现方式** **39**](#_Toc356012723_WPSOffice_Level1)

[3.1 数据预处理 39](#_Toc435676493_WPSOffice_Level2)

[3.2 模型训练 42](#_Toc1643065228_WPSOffice_Level2)

[3.3 自创建模型 46](#_Toc505070223_WPSOffice_Level2)

[3.4 微信小程序上线 51](#_Toc1859865017_WPSOffice_Level2)

[**第四章 结果分析** **55**](#_Toc616394919_WPSOffice_Level1)

[4.1 数据集在不同预训练模型上的表现 55](#_Toc2126858634_WPSOffice_Level2)

[4.2 Mixup对模型的影响 56](#_Toc1247757323_WPSOffice_Level2)

[4.2 尝试理解神经网络模型 56](#_Toc879514706_WPSOffice_Level2)

[**第五章 总结** **59**](#_Toc288290505_WPSOffice_Level1)

[5.1 心得体会 59](#_Toc873721441_WPSOffice_Level2)

[5.2 后续工作 60](#_Toc143080701_WPSOffice_Level2)

[5.3 致谢 61](#_Toc1723140714_WPSOffice_Level2)

[**参考文献 61**](#_Toc288290505_WPSOffice_Level1)

## 第一章 项目系统分析与设计

### 1.1 项目背景

2020年12月24日，国务院新闻办公室发布了《中国居民营养与慢性病状况报告（2020年）》，报告称，我国城乡居民膳食能量和蛋白质、脂肪、碳水化合物三大宏量营养素摄入充足，优质蛋白质摄入不断增加，成年人平均身高持续增长。但同时，我国居民超重肥胖问题不断凸显，慢性病患病/发病仍成上升趋势，18岁以上居民超重率和肥胖率分别为34.3%和16.4%。报告显示，我国居民不健康生活方式普遍存在，居民畜肉摄入较多，膳食脂肪供能占比持续上升，日常使用的食品添加剂超过推荐量较大。

同时，发布于著名医学杂志《柳叶刀》上的一篇题为《195个国家和地区饮食结构造成的死亡率和疾病负担》的文章指出，全球近20%的死亡与饮食有关，且在世界人口前20位的国家中，中国因饮食结构不合理造成的心血管疾病死亡率、癌症死亡率都是第一名。全球每年因肥胖问题所造成的经济损失已高达2万亿美元，相当于全球GDP的2.8¥%，且这一数字已经和战争所导致的经济损失相近。

中国BMI大于28以上的肥胖人群现已突破1亿，肥胖率已突破10%，其中城市成年人体重超重者已经突破40%，且肥胖问题正出现趋低龄化的特点，青少年肥胖率持续增长等特征。

分析上述产生的问题，不难发现随着我国国民经济的不断发展，人们的生活质量得到了质的提高，大部分中国人已经可以负担得起任何饮食消费了，但同时，由此引发的饮食结构不均衡、日均摄入热量过多、摄入量与产出量不对等等问题导致了肥胖或超重的大面积出现。

与此同时，通过调查发现，现代年轻人当中，大部分明确肥胖和不健康饮食带来的问题，也希望能够通过控制饮食来形成健康的体质条件。但在日常生活中控制热量摄入的难度较大，很难量化所摄入食物的热量。因此，本项目旨在利用数字图像处理技术与人工智能中的深度学习技术，利用现有的数据集训练出一张可以识别食物类别的卷积神经网络模型，开发一款部署有该模型的微信小程序，利用移动智能设备的摄像头捕捉食物图像，通过神经网络模型即可获悉所摄入食物的热量，便于控制日常饮食，以达到瘦身或健身的目的，减少肥胖带来的健康隐患。

本项目的设计与实现依据上述背景展开。

### 1.2 项目需求分析

针对项目背景，本项目小组制作了《食物热量识别小程序》调查问卷（附件1），对项目的需求程度和可行性进行了小范围的调查，共收集到312份调查结果，统计结果及分析如下：

* 19～25岁的接受调查者占93%，其余的全部为19岁以下，即该调查结果针对青少年群体
* 65份调查结果显示受访者会关注选择食物的热量，占到20.8%，剩下的则不会，说明大部分人其实并没有控制进食热量的意识
* 95.3%的受访者表示他们会经常使用微信小程序（一周两次以上），其余4%的受访者则表示他们一般使用（两周一次到一周一次），即微信小程序已经成了如今大多数人生活中必不可少的一部分，使用起来非常方便
* 会将食物热量作为参考，影响食物选择的受访者有161位，占比50.1%，而有93位受访者表示经常根据食物热量来选择合适的食物，剩下58位则是完全不会根据食物热量选择食物。此项说明大部分人对食物热量有一定的要求，对获悉食物热量也有一定需求
* 73%的受访者表示自己会有健身或瘦身的需求，而剩下的则表示不会。说明大部分人对自己的身体素质或者健康状况并不满意，食物热量识别有一定的市场。

综上所述，微信小程序作为如今智能移动设备上必备的功能，普及量非常广，有着庞大的用户基础，且鉴于其使用方便，已经成为了人们尤其是年轻人群体日常使用频率最高的应用。青少年群体中大多数人关心摄食热量，有着健身或者瘦身的健康需求，但由于食物热量的难于量化，大多数人没有渠道关心日常摄食的热量，对于本项目将产生的系统，已经有一定的需求量和用户基础，存在一定的市场，具备可行性。

对于上述的可行性分析，结合本项目作为《数字图像处理》课程的课程设计，对项目的整体需求分析如下：

1. 产生的系统需要具备一定的准确性
2. 基于该系统的应用使用要足够方便
3. 该系统的开发、维护要尽量简单
4. 实现该系统的过程中要尽量多运用数字图像处理的知识和技术
5. 要进行足够多的有说服力的对比实验，选择最合适的模型
6. 训练出的模型要足够精简

### 1.3 开发环境和工具

#### Colaboratory

由于GPU资源稀缺和项目成本有限，本项目中大部分代码的运行和模型训练均在Google公司的*Colaboratory*平台上展开

Colaboratory是Google的一个研究项目，旨在提供开发者一个云端训练神经网络的工具。它是Jupyter一个笔记本环境，不用做任何配置，完全运行在云端。 Colaboratory存储在Google Drive中，可以进行共享。Colaboratory向开发者提供了免费的Tesla K80 GPU使用。

#### PyTorch

PyTorch 是一个针对深度学习, 并且使用 GPU 和 CPU 来优化的 tensor library (张量库)

2017年1月，由[Facebook](https://baike.baidu.com/item/Facebook/7449587)[人工智能](https://baike.baidu.com/item/人工智能/9180)研究院（FAIR）基于Torch推出了PyTorch。它是一个基于Python的可续计算包，提供两个高级功能：

1. 具有强大的GPU加速的张量计算（如NumPy）。
2. 包含自动求导系统的深度神经网络。

其特点如下：

* PyTorch是相当简洁且高效快速的框架
* 设计追求最少的封装
* 设计符合人类思维，它让用户尽可能地专注于实现自己的想法
* 与google的Tensorflow类似，FAIR的支持足以确保PyTorch获得持续的开发更新
* PyTorch作者亲自维护的论坛 供用户交流和求教问题
* 入门简单

#### fastai

fastai是一个提供高度封装迅速产生可用于生产环境的深度学习库，其为研究人员提供可以混合匹配以构建模型的低级组件方法，归功于其精心分层的体系结构，在不影响易用性、灵活性或性能的前提下做到以解耦的抽象表达了许多深度学习和数据处理技术的通用底层模式。fastai为基于PyTorch库构建的生产力深度学习包，可以简洁明了地表达抽象模型。

fastai包括：

* 适用于Python的新型分派系统以及张量的语义类型层次结构
* GPU优化的计算机视觉库，可以在纯Python中扩展
* 一种将现代优化器的通用功能重构为两个基本部分的优化器，从而可以在4-5行代码中实现优化算法
* 一种新颖的双向回调系统，可以访问数据，模型或优化器的任何部分，并在训练过程中随时进行更改
* 新的数据块API

#### Tensorflow

谷歌大脑自2011年成立起开展了面向科学研究和谷歌产品开发的大规模深度学习应用研究，其早期工作即是TensorFlow的前身DistBelief。DistBelief的功能是构建各尺度下的神经网络分布式学习和交互系统，也被称为“第一代机器学习系统” 。DistBelief在谷歌和Alphabet旗下其它公司的产品开发中被改进和广泛使用。2015年11月，在DistBelief的基础上，谷歌大脑完成了对“第二代机器学习系统”TensorFlow的开发并对代码开源。相比于前作，TensorFlow在性能上有显著改进、构架灵活性和可移植性也得到增强 。此后TensorFlow快速发展，截至稳定API版本1.12，已拥有包含各类开发和研究项目的完整生态系统。在2018年4月的TensorFlow开发者峰会中，有21个TensorFlow有关主题得到展示

TensorFlow™是一个基于数据流编程（dataflow programming）的符号数学系统，被广泛应用于各类机器学习（machine learning）算法的编程实现，其前身是谷歌的神经网络算法库DistBelief [1] 。  
Tensorflow拥有多层级结构，可部署于各类服务器、PC终端和网页并支持GPU和TPU高性能数值计算，被广泛应用于谷歌内部的产品开发和各领域的科学研究。

TensorFlow由谷歌人工智能团队谷歌大脑（Google Brain）开发和维护，拥有包括TensorFlow Hub、TensorFlow Lite、TensorFlow Research Cloud在内的多个项目以及各类应用程序接口（Application Programming Interface, API）。自2015年11月9日起，TensorFlow依据阿帕奇授权协议（Apache 2.0 open source license）开放源代码

#### Keras

Keras是一个由Python编写的开源人工神经网络库，可以作为Tensorflow、Microsoft-CNTK和Theano的高阶应用程序接口，进行深度学习模型的设计、调试、评估、应用和可视化。

Keras在代码结构上由面向对象方法编写，完全模块化并具有可扩展性，其运行机制和说明文档有将用户体验和使用难度纳入考虑，并试图简化复杂算法的实现难度 [1] 。Keras支持现代人工智能领域的主流算法，包括前馈结构和递归结构的神经网络，也可以通过封装参与构建统计学习模型 [2] 。在硬件和开发环境方面，Keras支持多操作系统下的多GPU并行计算，可以根据后台设置转化为Tensorflow、Microsoft-CNTK等系统下的组件 。

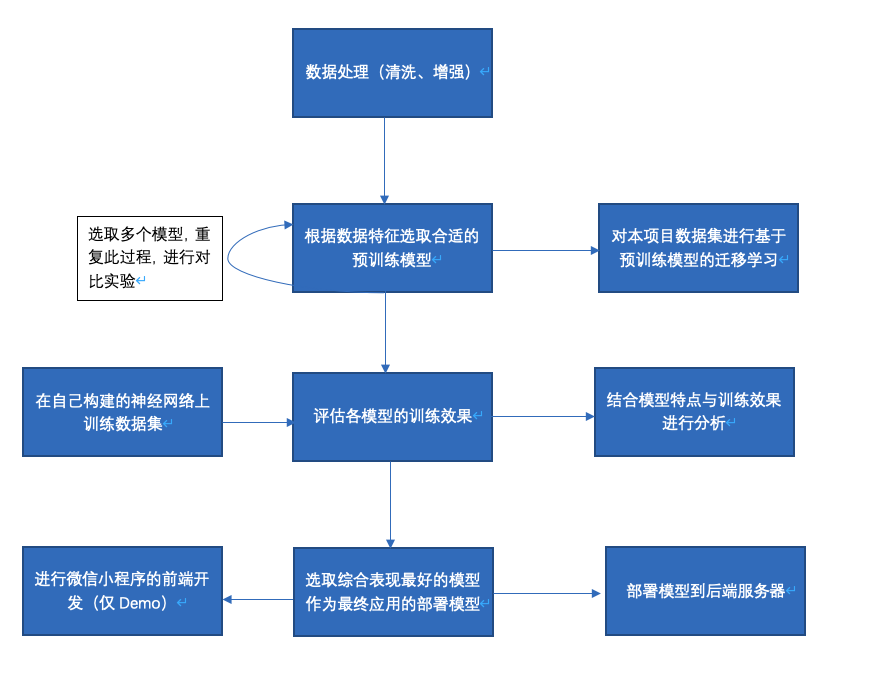
Keras的主要开发者是谷歌工程师François Chollet，此外其GitHub项目页面包含6名主要维护者和超过800名直接贡献者 。Keras在其正式版本公开后，除部分预编译模型外，按MIT许可证开放源代码

Keras的设计遵循以下的指导原则：

* **用户友好。** Keras 是为人类而不是为机器设计的 API。它把用户体验放在首要和中心位置。Keras 遵循减少认知困难的最佳实践：它提供一致且简单的 API，将常见用例所需的用户操作数量降至最低，并且在用户错误时提供清晰和可操作的反馈。
* **模块化。** 模型被理解为由独立的、完全可配置的模块构成的序列或图。这些模块可以以尽可能少的限制组装在一起。特别是神经网络层、损失函数、优化器、初始化方法、激活函数、正则化方法，它们都是可以结合起来构建新模型的模块。
* **易扩展性。** 新的模块是很容易添加的（作为新的类和函数），现有的模块已经提供了充足的示例。由于能够轻松地创建可以提高表现力的新模块，Keras 更加适合高级研究。
* **基于 Python 实现。** Keras 没有特定格式的单独配置文件。模型定义在 Python 代码中，这些代码紧凑，易于调试，并且易于扩展。

### 1.4 系统设计思路

根据项目背景和需求分析，结合本项目作为《数字图像处理》课程设计的性质及本项目小组成员能力，本系统的开发流程如下图所示：



其中，对于“根据特征选取合适的预训练模型”，本项目小组经过查阅资料、分析模型特点，结合本项目的需求及特点，选取了以下3个预训练过的神经网络模型：

* ResNet34（深度残差神经网络）
* ResNet101（更深层的深度残差神经网络）
* SqueezeNet1\_1（更适合云端和受限设备的小架构卷积神经网络）

之后，通过对数据集利用不同的扩增策略进行扩增，然后分别在选取的3个神经网络模型上进行小批次的训练，得到不同的模型参数，对以上得到的不同模型进行评估后，选取了一个综合表现较好的模型进行后续的研究。

mixup是一种非常规的数据增强方法，一个和数据无关的简单数据增强原则，其以线性插值的方式来构建新的训练样本和标签。最终对标签的处理如下公式所示，这很简单但对于增强策略来说又很不一般

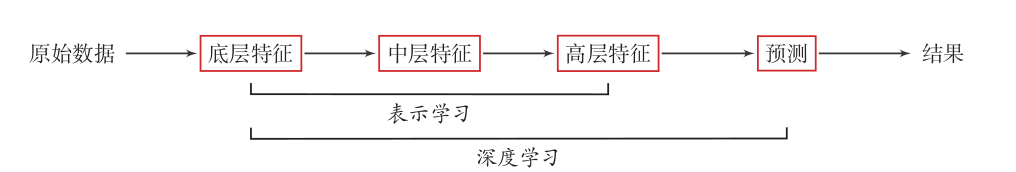
## 第二章 相关算法和技术理论

### 2.1 深度学习

为了学习一种好的表示, 需要构建具有一定 “深度” 的模型, 并通过学习算法  
来让模型自动学习出好的特征表示(从底层特征, 到中层特征, 再到高层特征),  
从而最终提升预测模型的准确率. 所谓 “深度” 是指原始数据进行非线性特征转  
换的次数. 如果把一个表示学习系统看作一个有向图结构, 深度也可以看作从输入节点到输出节点所经过的最长路径的长度

这样我们就需要一种学习方法可以从数据中学习一个“深度模型”,这就  
是深度学习( Deep Learning , DL ). 深度学习是机器学习的一个子问题, 其主要目的是从数据中自动学习到有效的特征表示

下图给出了深度学习的数据处理流程. 通过多层的特征转换, 把原始数据  
变成更高层次、 更抽象的表示. 这些学习到的表示可以替代人工设计的特征, 从而避免 “特征工程”



深度学习是将原始的数据特征通过多步的特征转换得到一种特征表示, 并  
进一步输入到预测函数得到最终结果. 和 “浅层学习” 不同, 深度学习需要解决  
的关键问题是贡献度分配问题( Credit Assignment Problem , CAP ) [Minsky,  
1961] , 即一个系统中不同的组件( component )或其参数对最终系统输出结果  
的贡献或影响. 以下围棋为例, 每当下完一盘棋, 最后的结果要么赢要么输. 我们会思考哪几步棋导致了最后的胜利, 或者又是哪几步棋导致了最后的败局. 如何判断每一步棋的贡献就是贡献度分配问题, 这是一个非常困难的问题. 从某种意义上讲, 深度学习可以看作一种强化学习( Reinforcement Learning , RL ), 每个内部组件并不能直接得到监督信息, 需要通过整个模型的最终监督信息(奖励)得到,并且有一定的延时性.

目前, 深度学习采用的模型主要是神经网络模型, 其主要原因是神经网络模  
型可以使用误差反向传播算法, 从而可以比较好地解决贡献度分配问题. 只要是超过一层的神经网络都会存在贡献度分配问题, 因此可以将超过一层的神经网络都看作深度学习模型. 随着深度学习的快速发展, 模型深度也从早期的 5 ∼ 10层增加到目前的数百层. 随着模型深度的不断增加, 其特征表示的能力也越来越强,从而使后续的预测更加容易.

### 2.2 人工神经网络

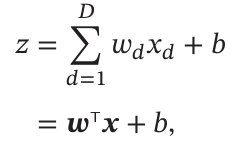
人工神经网络是为模拟人脑神经网络而设计的一种计算模型, 它从结构、 实  
现机理和功能上模拟人脑神经网络. 人工神经网络与生物神经元类似, 由多个节点(人工神经元)互相连接而成, 可以用来对数据之间的复杂关系进行建模. 不  
同节点之间的连接被赋予了不同的权重, 每个权重代表了一个节点对另一个节  
点的影响大小. 每个节点代表一种特定函数, 来自其他节点的信息经过其相应的权重综合计算, 输入到一个激活函数中并得到一个新的活性值(兴奋或抑制).  
从系统观点看, 人工神经元网络是由大量神经元通过极其丰富和完善的连接而  
构成的自适应非线性动态系统

虽然我们可以比较容易地构造一个人工神经网络, 但是如何让人工神经网  
络具有学习能力并不是一件容易的事情. 早期的神经网络模型并不具备学习能  
力. 首个可学习的人工神经网络是赫布网络, 采用一种基于赫布规则的无监督学习方法. 感知器是最早的具有机器学习思想的神经网络, 但其学习方法无法扩展到多层的神经网络上. 直到 1980 年左右, 反向传播算法才有效地解决了多层神经网络的学习问题,并成为最为流行的神经网络学习算法.

人工神经网络诞生之初并不是用来解决机器学习问题. 由于人工神经网络可以用作一个通用的函数逼近器(一个两层的神经网络可以逼近任意的函数),因此我们可以将人工神经网络看作一个可学习的函数, 并将其应用到机器学习中.理论上,只要有足够的训练数据和神经元数量,人工神经网络就可以学到很多复杂的函数. 我们可以把一个人工神经网络塑造复杂函数的能力称为网络容量( Network Capacity ), 这与可以被储存在网络中的信息的复杂度以及数量相关

人工神经元( Artificial Neuron ), 简称神经元( Neuron ), 是构成神经网络的基本单元, 其主要是模拟生物神经元的结构和特性, 接收一组输入信号并产生输出.

假设一个神经元接收 D 个输入 x 1 , x 2 , ⋯ , x D ,令向量 x = [x 1 ; x 2 ; ⋯ ; x D ] 来表示这组输入, 并用净输入( Net Input ) z ∈ R 表示一个神经元所获得的输入信号 x 的加权和

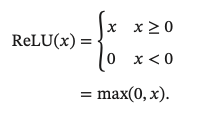


净输入 z 在经过一个非线性函数 f(⋅) 后, 得到神经元的活性值( Activation )a

该非线性函数称为激活函数

### 2.3 ReLU函数

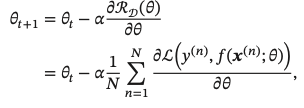
ReLU(Rectified Linear Unit，修正线性单元)[Nair et al., 2010]，也叫 Rectifier 函数 [Glorot et al., 2011]，是目前深度神经网络中经常使用的激活函数. ReLU 实际上是一个斜坡(ramp)函数，定义为



### 2.4 反向传播和梯度下降

为了充分利用凸优化中一些高效、成熟的优化方法，比如共轭梯度、拟牛顿法等，很多机器学习方法都倾向于选择合适的模型和损失函数，以构造一个凸函 数作为优化目标.但也有很多模型(比如神经网络)的优化目标是非凸的，只能退而求其次找到局部最优解.

在机器学习中，最简单、常用的优化算法就是梯度下降法，即首先初始化参 数 𝜃0 ，然后按下面的迭代公式来计算训练集 𝒟 上风险函数的最小值:



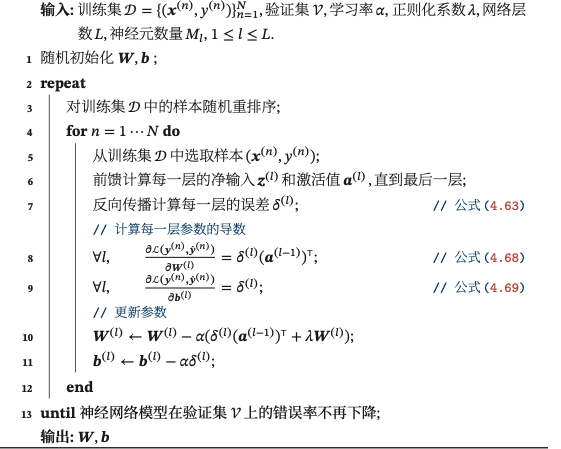
其中 𝜃𝑡 为第 𝑡 次迭代时的参数值，𝛼 为搜索步长.在机器学习中，𝛼 一般称为学习率(Learning Rate).

假设采用随机梯度下降进行神经网络参数学习，给定一个样本 (𝒙, 𝒚)，将其 输入到神经网络模型中，得到网络输出为 𝒚̂.假设损失函数为 L(𝒚, 𝒚̂)，要进行参数学习就需要计算损失函数关于每个参数的导数.

第 𝑙 层的误差项可以通过第 𝑙 + 1 层的误差项计算得到，这就是误差的反向传播(BackPropagation，BP).反向传播算法的含义是: 第 𝑙 层的一个神经元的误差项(或敏感性)是所有与该神经元相连的第 𝑙 + 1 层 的神经元的误差项的权重和.然后，再乘上该神经元激活函数的梯度.

在计算出每一层的误差项之后，我们就可以得到每一层参数的梯度.因此， 使用误差反向传播算法的前馈神经网络训练过程可以分为以下三步:

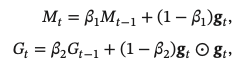
1. 前馈计算每一层的净输入 𝒛
2. 反向传播计算每一层的误差项 𝛿(𝑙)
3. 计算每一层参数的偏导数，并更新参数.和激活值 𝒂 ，直到最后一层

* 

### 2.5 Adam算法

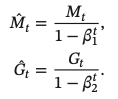
Adam 算法(Adaptive Moment Estimation Algorithm)[Kingma et al., 2015]可以看作动量法和 RMSprop 算法的结合，不但使用动量作为参数更新方向，而 且可以自适应调整学习率.

Adam 算法一方面计算梯度平方 **g**𝑡2 的指数加权平均(和 RMSprop 算法类似)，另一方面计算梯度 **g**𝑡 的指数加权平均(和动量法类似).

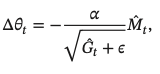


其中𝛽1和𝛽2分别为两个移动平均的衰减率，通常取值为𝛽1 =0.9,𝛽2 =0.99.我 们可以把 𝑀𝑡 和 𝐺𝑡 分别看作梯度的均值(一阶矩)和未减去均值的方差(二阶 矩).

假设 𝑀0 = 0, 𝐺0 = 0，那么在迭代初期 𝑀𝑡 和 𝐺𝑡 的值会比真实的均值和方差 要小.特别是当 𝛽1 和 𝛽2 都接近于 1时，偏差会很大.因此，需要对偏差进行修正.



Adam 算法的参数更新差值为：



### 2.6 小批量梯度下降

在训练深度神经网络时，训练数据的规模通常都比较大.如果在梯度下降 时，每次迭代都要计算整个训练数据上的梯度，这就需要比较多的计算资源.另 外大规模训练集中的数据通常会非常冗余，也没有必要在整个训练集上计算梯 度.因此，在训练深度神经网络时，经常使用小批量梯度下降法(Mini-Batch Gradient Descent).

影响小批量梯度下降法的主要因素有:1)批量大 小𝐾 、2)学习率𝛼、3)梯度估计.为了更有效地训练深度神经网络，在标准的小批 量梯度下降法的基础上，也经常使用一些改进方法以加快优化速度，比如如何选 择批量大小、如何调整学习率以及如何修正梯度估计.我们分别从这三个方面来 介绍在神经网络优化中常用的算法.这些改进的优化算法也同样可以应用在批量或随机梯度下降法上.

### 2.6 卷积神经网络

在全连接前馈神经网络中，如果第 𝑙 层有 𝑀𝑙 个神经元，第 𝑙 − 1 层有 𝑀𝑙−1 个 神经元，连接边有 𝑀𝑙 × 𝑀𝑙−1 个，也就是权重矩阵有 𝑀𝑙 × 𝑀𝑙−1 个参数.当 𝑀𝑙 和 𝑀𝑙−1 都很大时，权重矩阵的参数非常多，训练的效率会非常低.如果采用卷积来代替全连接，第 𝑙 层的净输入 𝒛(𝑙) 为第 𝑙 − 1 层活性值 𝒂(𝑙−1) 和卷积核 𝒘(𝑙) ∈ R𝐾 的卷积，即

𝒛(𝑙) = 𝒘(𝑙) ⊗ 𝒂(𝑙−1) + 𝑏(𝑙),

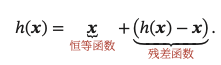
其中卷积核 𝒘(𝑙) ∈ R𝐾 为可学习的权重向量，𝑏(𝑙) ∈ R 为可学习的偏置.

* 局部连接
* 在卷积层(假设是第 𝑙 层)中的每一个神经元都只和下一层(第 𝑙 − 1 层)中某个局部窗口内的神经元相连，构成一个局部连接网络.如图5.5b所示，卷 积层和下一层之间的连接数大大减少，由原来的 𝑀𝑙 × 𝑀𝑙−1 个连接变为 𝑀𝑙 × 𝐾 个连接，𝐾 为卷积核大小.
* 权重共享
* 作为参数的卷积核 𝒘(𝑙) 对于第 𝑙 层的所有的 神经元都是相同的.如图5.5b中，所有的同颜色连接上的权重是相同的.权重共享可以理解为一个卷积核只捕捉输入数据中的一种特定的局部特征.因此，如果 要提取多种特征就需要使用多个不同的卷积核.

### 2.7 残差网络

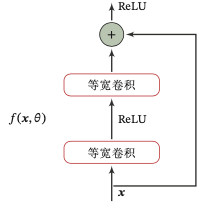
残差网络(Residual Network，ResNet)通过给非线性的卷积层增加直连边 (Shortcut Connection)(也称为残差连接(Residual Connection))的方式来提高信息的传播效率.

假设在一个深度网络中，我们期望一个非线性单元(可以为一层或多层的卷积层)𝑓(𝒙; 𝜃) 去逼近一个目标函数为 h(𝒙).如果将目标函数拆分成两部分:恒 等函数(Identity Function)𝒙 和残差函数(Residue Function)h(𝒙) − 𝒙.



根据通用近似定理，一个由神经网络构成的非线性单元有足够的能力来近似逼 近原始目标函数或残差函数，但实际中后者更容易学习 [He et al., 2016].因此， 原来的优化问题可以转换为:让非线性单元 𝑓(𝒙; 𝜃) 去近似残差函数 h(𝒙) − 𝒙，并 用 𝑓(𝒙; 𝜃) + 𝒙 去逼近 h(𝒙).

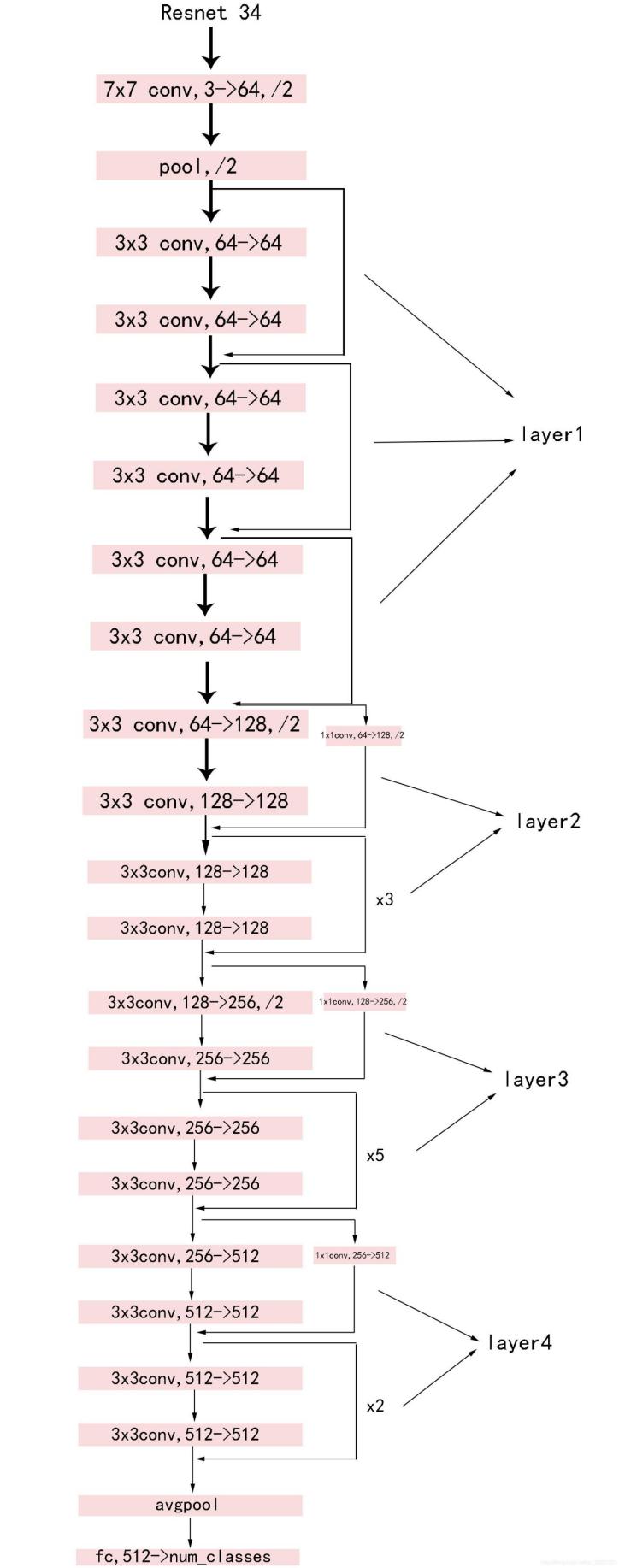
下图给出了一个典型的残差单元示例.残差单元由多个级联的(等宽)卷 积层和一个跨层的直连边组成，再经过 ReLU 激活后得到输出.



残差网络就是将很多个残差单元串联起来构成的一个非常深的网络

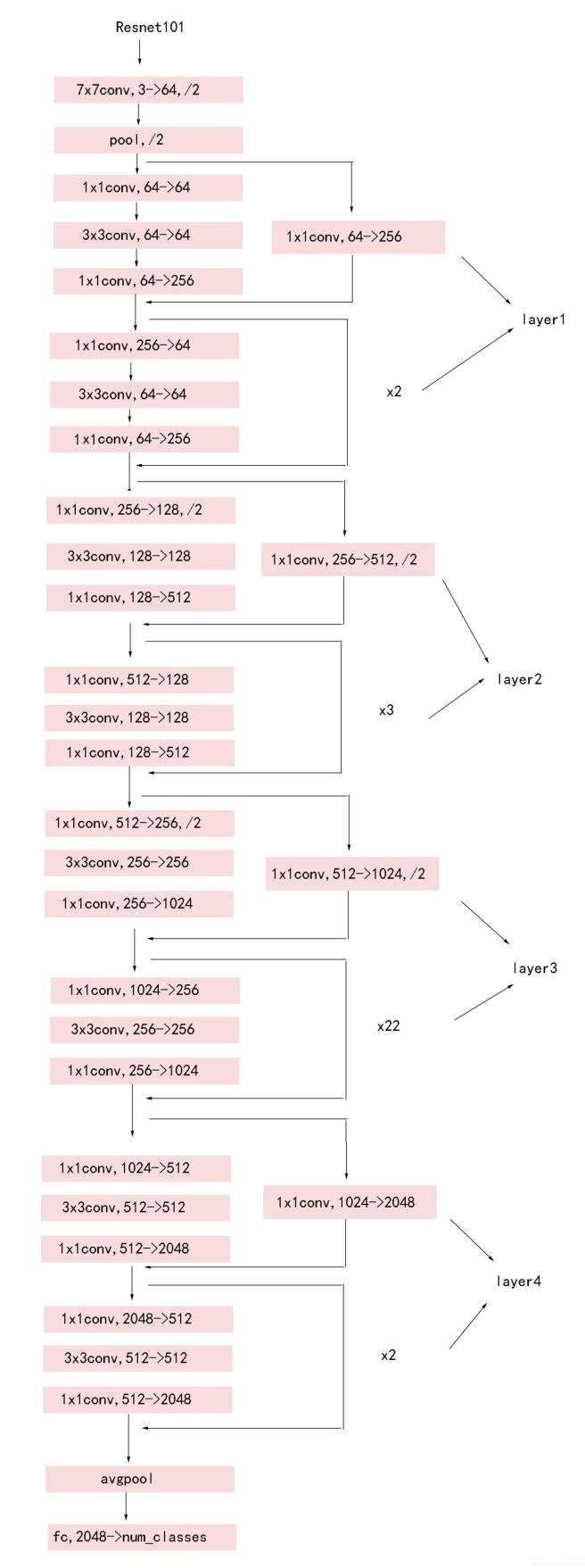
#### 2.7.1 ResNet34

即34层的残差网络



#### 2.7.2 ResNet101

即101层的残差网络



### 2.8 SqueezeNet

近来深层卷积网络的主要研究方向集中在提高正确率。对于相同的正确率水平，更小的CNN架构可以提供如下的优势：

* 在分布式训练中，与服务器通信需求更小
* 参数更少，从云端下载模型的数据量小
* 更适合在FPGA等内存受限的设备上部署。

基于这些优点，本文提出SqeezeNet。它在ImageNet上实现了和AlexNet相同的正确率，但是只使用了1/50的参数。更进一步，使用模型压缩技术，可以将SqueezeNet压缩到0.5MB，这是AlexNet的1/510。

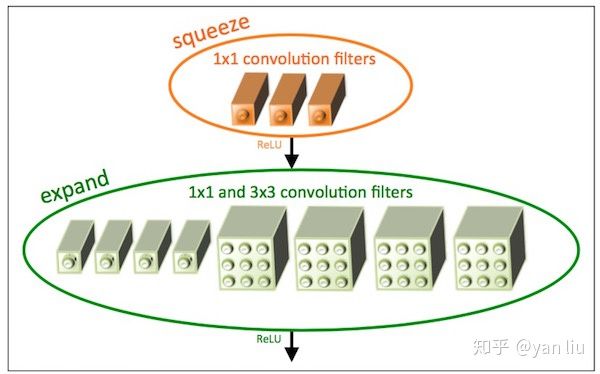
使用以下三个策略来减少SqueezeNet设计参数

* 使用1∗1卷积代替3∗3 卷积：参数减少为原来的1/9
* 减少输入通道数量：这一部分使用squeeze layers来实现
* 将欠采样操作延后，可以给卷积层提供更大的激活图：更大的激活图保留了更多的信息，可以提供更高的分类准确率

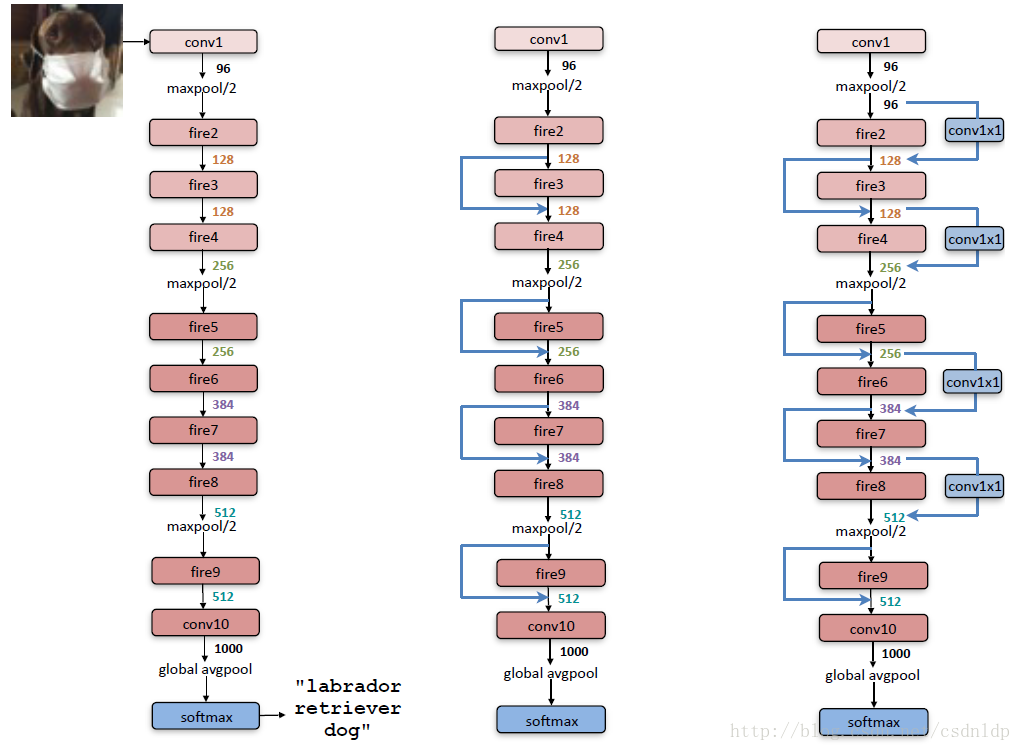
其中，（1）和（2）可以显著减少参数数量，（3）可以在参数数量受限的情况下提高准确率。

Fire Module是SqueezeNet中的基础构建模块，如下定义 **Fire Module** :

一个Fire模块由Squeeze部分和Expand部分组成（注意区分和Momenta的[SENet](https://zhuanlan.zhihu.com/p/47494490)[4]的区别）。Squeeze部分是一组连续的1\*1卷积组成，Expand部分则是由一组连续的1\*1卷积和一组连续的3\*3卷积cancatnate组成，因此3\*3卷积需要使用same卷积



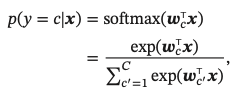
如下图，左边为原始的SqueezeNet，中间为包含simple bypass的改进版本，最右侧为使用complex bypass的改进版本。



### 2.9 Softmax 回归

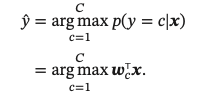
Softmax 回归(Softmax Regression)，也称为多项(Multinomial)或多类 (Multi-Class)的 Logistic 回归，是 Logistic 回归在多分类问题上的推广.

对于多类问题，类别标签 𝑦 ∈ {1, 2, ⋯ , 𝐶} 可以有 𝐶 个取值.给定一个样本 𝒙， Softmax 回归预测的属于类别 𝑐 的条件概率为



其中 𝒘𝑐 是第 𝑐 类的权重向量.

Softmax 回归的决策函数可以表示为

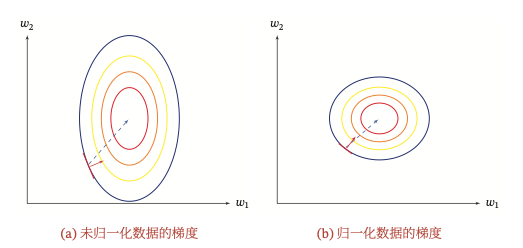


### 2.10 数据预处理

一般而言，样本特征由于来源以及度量单位不同，它们的尺度(Scale)(即 取值范围)往往差异很大.以描述长度的特征为例，当用“米”作单位时令其值为 𝑥，那么当用“厘米”作单位时其值为 100𝑥.不同机器学习模型对数据特征尺度的 敏感程度不一样.如果一个机器学习算法在缩放全部或部分特征后不影响它的 学习和预测，我们就称该算法具有尺度不变性(Scale Invariance).比如线性分 类器是尺度不变的，而最近邻分类器就是尺度敏感的.当我们计算不同样本之间 的欧氏距离时，尺度大的特征会起到主导作用.因此，对于尺度敏感的模型，必须 先对样本进行预处理，将各个维度的特征转换到相同的取值区间，并且消除不同 特征之间的相关性，才能获得比较理想的结果.

从理论上，神经网络应该具有尺度不变性，可以通过参数的调整来适应不 同特征的尺度.但尺度不同的输入特征会增加训练难度.假设一个只有一层的 网络 𝑦 = tanh(𝑤1𝑥1 + 𝑤2𝑥2 + 𝑏)，其中 𝑥1 ∈ [0, 10]，𝑥2 ∈ [0, 1].之前我们提 到 tanh 函数的导数在区间 [−2, 2] 上是敏感的，其余的导数接近于 0.因此，如果 𝑤1𝑥1 +𝑤2𝑥2 +𝑏过大或过小，都会导致梯度过小，难以训练.为了提高训练效率， 我们需要使 𝑤1 𝑥1 + 𝑤2 𝑥2 + 𝑏 在 [−2, 2] 区间，因此需要将 𝑤1 设得小一点，比如在 [−0.1, 0.1] 之间.可以想象，如果数据维数很多时，我们很难这样精心去选择每一 个参数.因此，如果每一个特征的尺度相似，比如 [0, 1] 或者 [−1, 1]，我们就不太 需要区别对待每一个参数，从而减少人工干预.

除了参数初始化比较困难之外， 不同输入特征的尺度差异比较大时， 梯度下降法的效率也会受到影响.图7.9给出了数据归一化对梯度的影响.其中， 图7.9a为未归一化数据的等高线图.尺度不同会造成在大多数位置上的梯度方向 并不是最优的搜索方向.当使用梯度下降法寻求最优解时，会导致需要很多次迭 代才能收敛.如果我们把数据归一化为相同尺度，如图7.9b所示，大部分位置的 梯度方向近似于最优搜索方向.这样，在梯度下降求解时，每一步梯度的方向都 基本指向最小值，训练效率会大大提高.



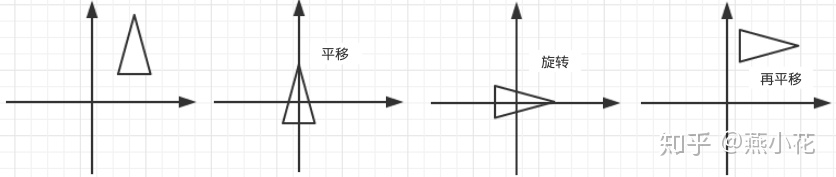
* 归一化(Normalization)方法泛指把数据特征转换为相同尺度的方法，比 如把数据特征映射到 [0, 1] 或 [−1, 1] 区间内，或者映射为服从均值为 0、方差为 1 的标准正态分布.归一化的方法有很多种，比如之前我们介绍的 Sigmoid 型函数 等都可以将不同尺度的特征挤压到一个比较受限的区间.这里，我们介绍几种在 神经网络中经常使用的归一化方法.
* 标准化(Standardization)也叫Z 值归一化(Z-Score Normalization)，来源于统计上的标准分数.将每一个维特征都调整为均值为 0，方差为 1.

### 2.11 数据增广

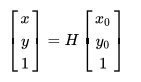
数据增广是深度学习中常用的技巧之一，主要用于增加训练数据集，让数据集尽可能的多样化，使得训练的模型具有更强的泛化能力．现有的各大深度学习框架都已经自带了数据增广，但是平时在用的使用只是直接调用了对应的接口函数，而没有进行详细的分析．在实际应用中，并非所有的增广方式都适用当前的训练数据，你需要根据自己的数据集特征来确定应该使用哪几种数据增广方式

目前数据增广主要包括：水平/垂直翻转，旋转，缩放，裁剪，剪切，平移，对比度，色彩抖动，噪声等

所有的数据增广在操作的时候默认是以图像中心点进行的．从数学角度来看，任何操作都可以分成以下几个步骤：1). 首先将旋转点移动到原点处 ；2). 执行如2所描述的绕原点的旋转；3). 再将旋转点移回到原来的位置

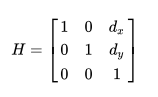


假设图像的原始坐标为(x0, y0) ，平移后的坐标为(x, y) ，则平移前和平移后的坐标关系为：



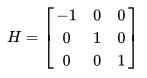
#### 图像平移

平移是指所有的像素在x和y方向各平移和，平移变换对应的数学矩阵为

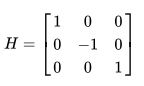


#### 图像翻转(图像镜像)

图像翻转包括水平翻转和垂直翻转．水平翻转的变换矩阵为：

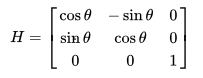


垂直翻转的变换矩阵为：



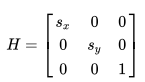
#### 图像旋转

图像旋转是指以某个点(默认为图像中心点)为中心进行任意角度的旋转，其变换矩阵为：



#### 图像缩放

图像缩放是指对当前图像进行任意尺度的缩放，其变换矩阵为：



#### 图像裁剪

深度学习的裁剪的常用做法是将图片缩放到原图的1.1倍，然后在缩放后的图像上进行裁剪操作

#### 组合变换

在深度学习中的数据增广一般会采用多种增广方式的组合，这里就会涉及到矩阵乘法运算，根据其运算的规则，可以知道**不同的组合顺序结果是不一样的**，即线性代数中的 AB != BA，当然特例除外．

#### fastai框架中的get\_transform方法

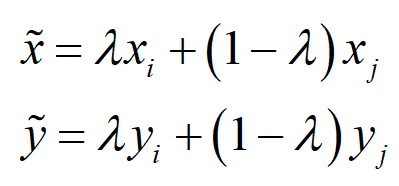
该方法用于指定训练用的数据集的图像增广策略

get\_transforms(  
 do\_flip:bool=True, # 是否进行水平翻转  
 flip\_vert:bool=False, #是否进行垂直翻转   
 max\_rotate:float=10.0, #最大旋转角  
 max\_zoom:float=1.1, #最大放缩比  
 max\_lighting:float=0.2, #最大亮度调整  
 max\_warp:float=0.2, #最大拉伸比  
 p\_affine:float=0.75, #仿射变换比  
 p\_lighting:float=0.75, #噪声比  
 xtra\_tfms:Optional[Collection[Transform]]=None) → Collection[Transform]

### 2.12 Mixup

mixup是一种运用在计算机视觉中的对图像进行混类增强的算法，它可以将不同类之间的图像进行混合，从而扩充训练数据集。

两个数据对是原始数据集中的训练样本对（训练样本和其对应的标签）。其中是一个服从B分布的参数,。Beta分布的概率密度函数如下图所示，其中



### 2.13 迁移学习和预训练

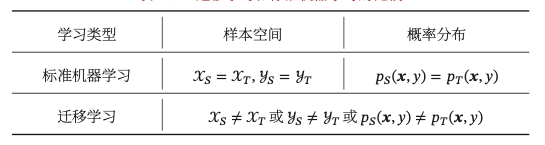
标准机器学习的前提假设是训练数据和测试数据的分布是相同的.如果不 满足这个假设，在训练集上学习到的模型在测试集上的表现会比较差.而在很多 实际场景中，经常碰到的问题是标注数据的成本十分高，无法为一个目标任务准 备足够多相同分布的训练数据.因此，如果有一个相关任务已经有了大量的训练 数据，虽然这些训练数据的分布和目标任务不同，但是由于训练数据的规模比较 大，我们假设可以从中学习某些可以泛化的知识，那么这些知识对目标任务会有 一定的帮助.如何将相关任务的训练数据中的可泛化知识迁移到目标任务上，就 是迁移学习(Transfer Learning)要解决的问题.

具体而言，假设一个机器学习任务 𝒯 的样本空间为 𝒳 × 𝒴，其中 𝒳 为输入空 间，𝒴 为输出空间，其概率密度函数为 𝑝(𝒙, 𝑦).为简单起见，这里设 𝒳 为 𝐷 维实数 空间的一个子集，𝒴 为一个离散的集合.

一个样本空间及其分布可以称为一个领域(Domain):𝒟 = (𝒳, 𝒴, 𝑝(𝒙, 𝑦)). 给定两个领域，如果它们的输入空间、输出空间或概率分布中至少一个不同，那 么这两个领域就被认为是不同的.从统计学习的观点来看，一个机器学习任务 𝒯 定义为在一个领域 𝒟 上的条件概率 𝑝(𝑦|𝒙) 的建模问题.

迁移学习是指两个不同领域的知识迁移过程，利用源领域(Source Domain) 𝒟𝑆 中学到的知识来帮助目标领域(Target Domain)𝒟𝑇 上的学习任务.源领域 的训练样本数量一般远大于目标领域.

下表给出了迁移学习和标准机器学习的比较



2014年Bengio等人在NIPS上发表论文 How transferable are features in deep neural networks，研究深度学习中各个layer特征的可迁移性（或者说通用性）文章中进行了如下图所示的实验，有四种模型

* Domain A上的基本模型BaseA
* Domain B上的基本模型BaseB
* Domain B上前n层使用BaseB的参数初始化（后续有frozen和fine-tuning两种方式）
* Domain B上前n层使用BaseA的参数初始化（后续有frozen和fine-tuning两种方式）

将深度学习应用在图像处理领域中时，会观察到第一层（first-layer）中提取的features基本上是类似于Gabor滤波器(Gabor filters)和色彩斑点(color blobs)之类的。  
通常情况下第一层与具体的图像数据集关系不是特别大，而网络的最后一层则是与选定的数据集及其任务目标紧密相关的；文章中将第一层feature称之为一般(general)特征，最后一层称之为特定(specific)特征

* 特征迁移使得模型的泛化性能有所提升，即使目标数据集非常大的时候也是如此。
* 随着参数被固定的层数n的增长，两个相似度小的任务之间的transferability gap的增长速度比两个相似度大的两个任务之间的transferability gap增长更快 两个数据集越不相似特征迁移的效果就越差
* 即使从不是特别相似的任务中进行迁移也比使用随机filters（或者说随机的参数）要好
* 使用迁移参数初始化网络能够提升泛化性能，即使目标task经过了大量的调整依然如此。

### 2.14 SSIM图像结构相似性

SSIM(S**tructural SIMilarity**)，结构相似性，是一种衡量两幅图像相似度的指标。该指标首先由德州大学奥斯丁分校的图像和视频工程实验室(Laboratory for Image and Video Engineering)提出。SSIM使用的两张图像中，一张为未经压缩的无失真图像，另一张为失真后的图像。

给定两个图像x和y, 两张图像的结构相似性可按照以下方式求出



结构相似性的范围为-1到1。当两张图像一模一样时，SSIM的值等于1。

### TTA

TTA(Test-Time Augmentation) ,即**测试时的数据增强**

实现步骤如下:

1. 将1个batch的数据通过flips, rotation, scale, etc.等操作生成batches
2. 将各个batch分别输入网络
3. 每个batch的masks/labels反向转换
4. 通过mean, max, gmean, etc.合并各个batch预测的结果
5. 最后输出最终的masks/labels

## 第三章 具体实现方式

### 3.1 数据预处理

数据预处理的具体实施方式均在*data\_handle.ipynb*中

由于计算资源的匮乏，本项目对原始数据集进行了缩小调整，将1000张/种随机删减到约300张/种，具体处理方式为

def decrease(path):  
 num = 0  
 if len(os.listdir(path)) <= 300:  
 return  
 for file in os.listdir(path):  
 num += 1  
 if num <= 300:  
 continue  
 os.remove(os.path.join(path, file))

接着，为了保证数据的尺寸的统一，通过调整图片的尺寸大小，将所有图片统一到

即数据集中仅有（512，512）或（512， 384）的图片数据，具体方法如下：

def Handle(file):  
 try:  
 img = Image.open(file)  
 size = img.size  
 print(size)  
 if min(size) < 512:  
 flag = 0  
 output\_size = ()  
 ret = 0  
 if size[0] == size[1]:  
 output\_size = (512, 512)  
 ret = 1  
 elif min(size)/max(size) >= 0.74 and min(size) / max(size) <= 0.76:  
 if size[0] == max(size):  
 output\_size = (512, 384)  
 else:  
 output\_size = (384, 512)  
 ret = 1  
 if ret == 1:  
 print(file + "transfoming to ", output\_size)  
 out = img.resize(output\_size, Image.ANTIALIAS)  
 out.save(file)  
 else:  
 os.remove(file)  
 except Exception as e:  
 print(e)  
 os.remove(file)  
 ret = 0  
 return ret

之后，通过python的爬虫技术从being.cn上爬取了9种中式食物，分别为：

* barbecue 烧烤
* bean curd 豆花
* fired rice 炒饭
* fried chicken 炸鸡
* gluteus rice balls 汤圆
* hand pulled noodle 拉面
* hot pot 火锅
* youtiao 油条
* zongzi 粽子

具体实现代码见*crawler.py*

对从互联网上爬取的图片进行去重，利用到了结构相似性的计算方式进行去重，具体见*ssim.py*

def find\_sim\_images(dir\_path):  
 imgs\_n = []  
 img\_files = [os.path.join(rootdir, file) for rootdir, \_, files in os.walk(dir\_path) for file in files if  
 (os.path.splitext(file)[-1] in EXT)]  
 for currIndex, filename in enumerate(img\_files):  
 if filename in imgs\_n:  
 continue  
 if currIndex >= len(img\_files) - 1:  
 break  
 for filename2 in img\_files[currIndex + 1:]:  
 if filename2 in imgs\_n:  
 continue  
 img = cv.imread(filename)  
 img1 = cv.imread(filename2)  
 try:  
 ssim = compare\_ssim(img, img1, multichannel=True)  
 if ssim > 0.9:  
 imgs\_n.append(filename2)  
 print(filename, filename2, ssim)  
 except ValueError:  
 pass  
 print(imgs\_n)  
 return imgs\_n

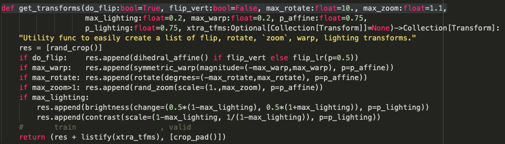
### 3.2 模型训练

#### 3.2.1 Part 1 选取综合表现最优的预训练神经网络模型

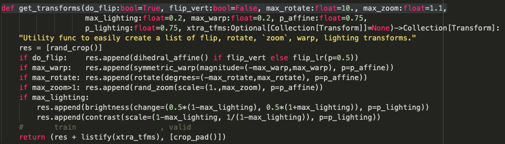
##### 模型训练

对于预处理完的数据，通过对其实施两种不同的增广策略：

1. fastai中自带的增广参数

* 

1. 本项目自设的增广参数

* 

在预先选定的3种预训练神经网络上进行训练，供训练出6个模型参数，得到6份不同的评估结果，见*\*model.ipynb*

以下仅对其中一次训练的过程进行描述：

1. 创建*ImageDataBunch*，其中batchsize设为64，验证集比例为0.2

* data = ImageDataBunch.from\_folder(path, valid\_pct=0.2,  
   ds\_tfms=get\_transforms(), size=224, num\_workers=8, bs=64).normalize(imagenet\_stats)

1. 创建预训练好的卷积神经网络模型

* learn = create\_cnn(data, models.resnet34, metrics=error\_rate,pretrained=True)  
    
  learn.loss\_func  
  output: FlattenedLoss of CrossEntropyLoss() #可知预训练模型采用的是交叉熵损失

1. 利用fastai中高度封装的方法寻找最优学习率

* learn.lr\_find()  
  learn.recorder.plot()

1. 先训练8个epoch

* learn.fit\_one\_cycle(8 , lr)

1. 将预训练模型的其他参数解冻

* learn.unfreeze()

1. 再训练5个epoch

* learn.fit\_one\_cycle(5, max\_lr=slice(1e-8,1e-4))

##### 结果评估

经过6次训练，得到6个不同的模型，对其进行的评估如下：

|  |
| --- |
| **Model** |
| ResNet34 |
| ResNet101 |
| SqueezeNet1.1 |

最后综合各方面表现选取了ResNet34作为Part 2研究的对象

#### 3.2.2 Part 2 考虑Mixup技术对训练效果的影响

##### 不同的划分策略

本部分数据集采取的划分策略为先将整体数据划分为80%训练集与20%的测试集，再从训练集中划分出来7%验证集

验证集用于评估每一轮的训练效果

测试集用于评估整个模型的拟合能力

##### TTL测试

本部分测试集评估采用了测试集增强技术

##### 模型训练

本部分控制的变量为是否应用Mixup技术，在原有的处理策略下对加入和不加入Mixup技术进行对比实验，共进行了两次实验，详情请见*mixup.ipynb*，以下仅对与上部分过程不同的步骤进行叙述

* 对数据集的划分
* train\_root\_dir = "train"  
  test\_root\_dir = "test"  
  if not os.path.exists(train\_root\_dir):  
   os.makedirs(train\_root\_dir)  
  if not os.path.exists(test\_root\_dir):  
   os.makedirs(test\_root\_dir)  
    
  for item in os.listdir("images"):  
   train\_path = os.path.join(train\_root\_dir, item)  
   test\_path = os.path.join(test\_root\_dir, item)  
   if not os.path.exists(train\_path):  
   os.makedirs(train\_path)  
   if not os.path.exists(test\_path):  
   os.makedirs(test\_path)  
    
  print(len(os.listdir(train\_root\_dir)))  
    
  valid\_pct = 0.2  
  for item in os.listdir("images"):  
   src = os.path.join("images", item)  
   train\_dft = os.path.join(train\_root\_dir, item)  
   test\_dft = os.path.join(test\_root\_dir, item)  
   files = os.listdir(src)  
   random.shuffle(files)  
   valid\_size = math.floor(valid\_pct\*len(files))  
   test\_files = files[0:valid\_size]  
   train\_files = files[valid\_size:]  
     
   for train\_file in train\_files:  
   shutil.copyfile(os.path.join(src, train\_file), os.path.join(train\_dft, train\_file))  
   for test\_file in test\_files:  
   shutil.copyfile(os.path.join(src, test\_file), os.path.join(test\_dft, test\_file))
* 启用Mixup
* learn = learn.mixup()
* 测试时运用TTA技术获取准确率
* preds\_tta, y\_tta = learn.TTA(beta=0.4, scale=1.05)  
  accuracy(preds\_tta, y\_tta)

##### 结果评估

|  |
| --- |
| **Model** |
| ResNet34 without **Mixup** |
| ResNet34 with **Mixup** |

### 3.3 自创建模型

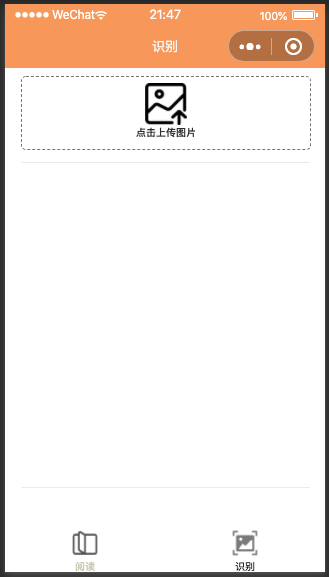
本项目实施时也尝试利用Keras框架构建了一个类似于AlexNet的复杂参数高深度的神经网络模型对其直接进行训练，但由于时间紧迫与计算资源匮乏，最终效果不太理想，训练过程可见*my\_model.ipynb*

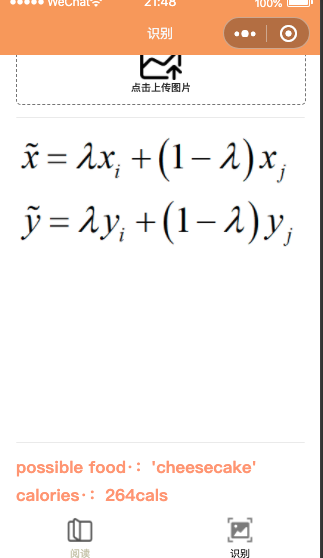
def MyNet(input\_shape=(224, 224, 3), num\_classes=110, l2\_reg=0.0, weights=None):  
 """  
 :param input\_shape: input shape  
 :param num\_classes: the number of classes  
 :param l2\_reg:  
 :param weights:  
 :return: model  
 """  
 input\_layer = Input(shape=input\_shape)  
  
 # Layer 1  
 # In order to get the same size of the paper mentioned, add padding layer first  
 x = ZeroPadding2D(padding=(2, 2))(input\_layer)  
 x = conv\_block(x, filters=96, kernel\_size=(11, 11),  
 strides=(4, 4), padding="valid", l2\_reg=l2\_reg, name='Conv\_1\_96\_11x11\_4')  
 x = MaxPool2D(pool\_size=(3, 3), strides=(2, 2), padding="valid", name="maxpool\_1\_3x3\_2")(x)  
  
 # Layer 2  
 x = conv\_block(x, filters=256, kernel\_size=(5, 5),  
 strides=(1, 1), padding="same", l2\_reg=l2\_reg, name="Conv\_2\_256\_5x5\_1")  
 x = MaxPool2D(pool\_size=(3, 3), strides=(2, 2), padding="valid", name="maxpool\_2\_3x3\_2")(x)  
  
 # Layer 3  
 x = conv\_block(x, filters=384, kernel\_size=(3, 3),  
 strides=(1, 1), padding="same", l2\_reg=l2\_reg, name="Conv\_3\_384\_3x3\_1")  
  
 # Layer 4  
 x = conv\_block(x, filters=384, kernel\_size=(3, 3),  
 strides=(1, 1), padding="same", l2\_reg=l2\_reg, name="Conv\_4\_384\_3x3\_1")  
  
 # Layer 5  
 x = conv\_block(x, filters=256, kernel\_size=(3, 3),  
 strides=(1, 1), padding="same", l2\_reg=l2\_reg, name="Conv\_5\_256\_3x3\_1")  
 x = MaxPool2D(pool\_size=(3, 3), strides=(2, 2), padding="valid", name="maxpool\_3\_3x3\_2")(x)  
  
 # Layer 6  
 x = Flatten()(x)  
 x = Dense(units=4096)(x)  
 x = BatchNormalization()(x)  
 x = Activation('relu')(x)  
  
 x = Dropout(0.5)(x)  
   
 # Layer 7  
 x = Dense(units=4096)(x)  
 x = BatchNormalization()(x)  
 x = Activation('relu')(x)  
  
 x = Dropout(0.5)(x)  
  
 # Layer 8  
 x = Dense(units=1000)(x)  
 x = BatchNormalization()(x)  
 x = Activation('relu')(x)  
  
 x = Dropout(0.5)(x)  
  
 # Layer 9  
 x = Dense(units=num\_classes)(x)  
 x = BatchNormalization()(x)  
 x = Activation("softmax")(x)  
  
 if weights is not None:  
 x.load\_weights(weights)  
 model = Model(input\_layer, x, name="MyNet")  
 return model

Model: "MyNet"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
input\_15 (InputLayer) [(None, 224, 224, 3)] 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
zero\_padding2d\_1 (ZeroPaddin (None, 228, 228, 3) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Conv\_1\_96\_11x11\_4 (Conv2D) (None, 55, 55, 96) 34944   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_22 (Batc (None, 55, 55, 96) 384   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_1 (Activation) (None, 55, 55, 96) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
maxpool\_1\_3x3\_2 (MaxPooling2 (None, 27, 27, 96) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Conv\_2\_256\_5x5\_1 (Conv2D) (None, 27, 27, 256) 614656   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_23 (Batc (None, 27, 27, 256) 1024   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_2 (Activation) (None, 27, 27, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
maxpool\_2\_3x3\_2 (MaxPooling2 (None, 13, 13, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Conv\_3\_384\_3x3\_1 (Conv2D) (None, 13, 13, 384) 885120   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_24 (Batc (None, 13, 13, 384) 1536   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_3 (Activation) (None, 13, 13, 384) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Conv\_4\_384\_3x3\_1 (Conv2D) (None, 13, 13, 384) 1327488   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_25 (Batc (None, 13, 13, 384) 1536   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_4 (Activation) (None, 13, 13, 384) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Conv\_5\_256\_3x3\_1 (Conv2D) (None, 13, 13, 256) 884992   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_26 (Batc (None, 13, 13, 256) 1024   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_5 (Activation) (None, 13, 13, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
maxpool\_3\_3x3\_2 (MaxPooling2 (None, 6, 6, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
flatten\_4 (Flatten) (None, 9216) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_12 (Dense) (None, 4096) 37752832   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_27 (Batc (None, 4096) 16384   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_6 (Activation) (None, 4096) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_8 (Dropout) (None, 4096) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_13 (Dense) (None, 4096) 16781312   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_28 (Batc (None, 4096) 16384   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_7 (Activation) (None, 4096) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_9 (Dropout) (None, 4096) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_14 (Dense) (None, 1000) 4097000   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_29 (Batc (None, 1000) 4000   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_8 (Activation) (None, 1000) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_10 (Dropout) (None, 1000) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_15 (Dense) (None, 110) 110110   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_30 (Batc (None, 110) 440   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
activation\_9 (Activation) (None, 110) 0   
=================================================================  
Total params: 62,531,166  
Trainable params: 62,509,810  
Non-trainable params: 21,356  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### 3.4 微信小程序上线

根据训练的结果，选取表现最优的模型，将其部署到服务器上，使得开发的微信小程序能够发送一张图片到云端通过神经网络计算得到该图片所属食物的类别





## 第四章 结果分析

### 4.1 数据集在不同预训练模型上的表现

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Default\_transforms** **Data\_size=224** **Freezed** **Epoches=8** | **Default\_transforms** **Data\_size=224** **After unfreeze** **Epoches=5** | **my\_transforms** **data\_size=256** **Freezed** **Epoches=8** | **my\_transforms** **data\_size=256** **After unfreeze** **Epoches\*=5** |
| ResNet34 | 0.658474 | **0.734183** | 0.675163 | 0.677135 |
| ResNet101 | **0.675163** | 0.677135 | **0.720831** | **0.724776** |
| SqueezeNet1.1 | 0.543924 | 0.722349 | 0.554999 | 0.58504 |

由上表可以看出：

* fastai自带的数据增广策略在三种模型上的表现比较平均，没有特别大的差异
* 在自设参数的增广策略下，三种模型的表现差异较大，其中参数量比较小的SqueezeNet的表现非常差，很有可能是SqueezeNet的小网络结构比较容易出现过拟合的现象，再加上训练批次较少，优势不明显，导致它没有获得一个很好的结果
* 三种模型在自设的增广策略下都表现为解冻后训练的批次对准确率影响不大，这很有可能是在小批次训练下陷入了局部最优
* ResNet101在自设增广策略下的表现比较好，但是在默认的增广策略下解冻前后的表现差异不大，很有可能是因为残差网络太深导致模型过拟合比较严重
* ResNet34在默认的增广策略下获得了最高的准确率，综合其在自设参数的增广策略上的表现以及解冻前后的差异化，选取ResNet34作为后续研究及上线的对象

### 4.2 Mixup对模型的影响

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Performance** **on validation dataset** | **Performance on test dataset** | **Performance on test dataset with** **TTA** |
| ResNet34 without **Mixup** | 0.730623 | 0.5973 | 0.5777 |
| ResNet34 with **Mixup** | **0.733333** | 0.5824 | 0.5978 |

由上表可以看出：

* 未应用Mixup的模型在raw Test dataset上表现较好
* 应用了Mixup的模型在Test dataset with TTA上的表现较好
* with Mixup的模型在validation dataset上的准确率较高，说明其由于训练批次较小，可能还没有充分收敛
* 这说明Mixup确实增加了模型的泛化能力
* 综合应用需求等，选定ResNet34 with Mixup作为最终的模型

### 4.2 尝试理解神经网络模型

#### 4.2.1 混淆矩阵

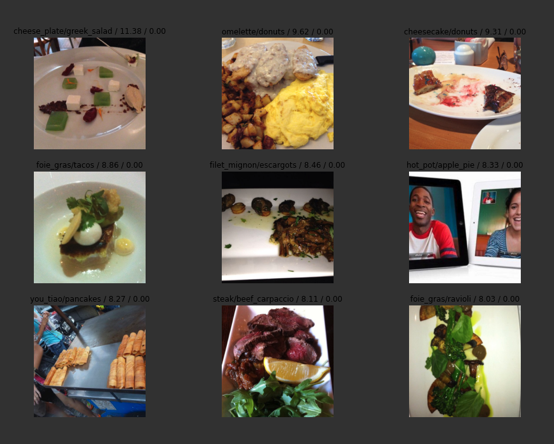
通过以下语句

interp = ClassificationInterpretation.from\_learner(learn.to\_fp32())  
interp.most\_confused(min\_val=4)

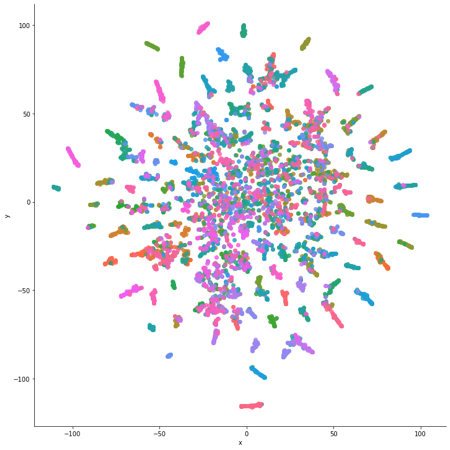
可以获取数据集中最容易混淆的一些类别：



可见这些食物类别由于食材相似、颜色相近等原因，确实不太好分辨，原始数据集也被证明想获取到更高准确率是一项非常困难的工作，因为人类对食物种类识别的能力也有限度



#### 4.2.2 高维特征在二维上的分布



由此分布图可以看出来，本项目的模型分辨能力确实不太优秀，但还是能够成功分辨出一些特征非常明显的类别

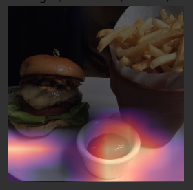
## 第五章 总结

### 5.1 心得体会

* 困难
* 由于本项目所用到的方法、技术都是比较先进、以前从来没有接触过的，因此在项目的实现上遇到了许多麻烦，其中比较突出的麻烦一是由于知识储备不足，遇到问题没有很好的办法解决，只能不断的尝试，同时在做出结果来的时候，由于缺乏理论知识的支撑，对结果的合理分析也变得十分困难。当然还有资源上的困难，没有GPU供我们使用导致我们训练模型的epoch不能太大，否则训练的时间会过于长。
* 数据至上
* 本项目让我真正意识到了数据决定训练效果的上限，很多时候训练的效果不好就是因为数据集不够干净，不够规整或者不够丰富，因此，对数据集的前期处理就显得非常重要了，我想这也就是我们为什么要学习《数字图像处理》这门课程，课程中所讲述的很多传统过的方法有助于帮助我们解释我们训练的模型的表现，或者是帮助我们通过增广增强自己的数据集。本项目也是用到了大量的数字图像处理的技术，甚至是基于对数字图像的不同处理去定义我们的对比实验，也算是很符合课程的要求了。
* 收获
* 收获肯定是非常丰富的，首先这个项目的体量是非常大的，跑了若干次模型，也自己设计了模型，对结果或者训练过程做了非常多的可视化，这其中遇到了不少麻烦，但好在都通过克重方法解决了。同时，这个课题也是项目组成员非常感兴趣的课题，完成了自己喜欢的课题，比较有成就感。当然，收获最大的还应属掌握了很多数字图像处理和深度学习的分析和实现方法，对数字图像处理技术的理解更加深刻了。

### 5.2 后续工作

如果有可能，日后还想继续完善这个项目，主要围绕以下三个方面展开

* 扩充数据集
* 目前有的数据集还太小了，日常食物的覆盖率较低，需要获得更多某种类食物的图像数据以及更多食物种类
* 图像分割
* 利用目前已有的图像分割技术，做到从照片中分割出食物实体，对其进行分析得到食物的量，进一步量化热量
* 类似下图的方式：
* 
* 同步训练
* 再度精简目前有的模型，或提出新的模型，在计算资源有限的情况下，能够做到部署服务的同时，收集用户提供的数据供神经网络模型同步训练

### 5.3 致谢

本项目能够顺利展开，首先要感谢课程教师彭辉老师的指导，在项目开展之前曾多次与老师展开交流，老师给我们提供了很有帮助的建议，为我们指定了研究的方向，提供了一些参考，让我们得以后续的顺利开展

同时，要向同小组的同学致谢，本项目作为课程设计的项目体量可能过于庞大，给同小组的同学增添了许多工作，最后还是顺利完成了这个项目，非常感谢。

最后，向给本项目提供理论或者技术上支持的其他同学致谢，本项目的顺利开展离不开大家宝贵的意见和热诚的帮助。

**参考文献**

[mixup: BEYOND EMPIRICAL RISK MINIMIZATION](https://arxiv.org/pdf/1710.09412.pdf) [*Hongyi Zhang Moustapha Cisse, Yann N. Dauphin, David Lopez-Paz, ICLR 2018*]