



机器学习纳米学位

基于迁移学习的猫狗识别 的设计与实现

作者姓名 陈锰杰

指导老师 杨培文

2018年05月17日

目 录

I.	问题的定义	1
1.1	项目背景	1
1.2	输入数据	1
1.3	问题描述	1
1.4	评价指标	3
II.	分析	4
2.1	数据的探索	4
2.2	算法和技术	7
2.2.1	分类算法	8
2.2.2	神经网络	8
2.2.3	卷积神经网络	9
2.2.4	技术	10
2.3	基准模型	11
2.4	设计大纲	12
III.	方法	15
3.1	数据预处理	15
3.1.1	剔除异常数据	15
3.1.2	调整文件目录	17
3.1.2	生成图片及归一化处理	17
3.2	执行过程	18
3.2.1	构建模型	18
3.2.2	使用 Inception v3 预训练模型	18
3.2.3	保存特征的权重	19
3.2.4	导入训练好的特征向量	19
3.2.5	添加全连接层和 dropout 层	19
3.2.6	编译模型	19
3.2.7	训练模型	20
3.2.8	模型参数调优	20
3.3	完善	22
IV.	结果	22

4.1 模型的评价和验证	22
4.2 合理性分析	24
V. 项目结论	25
5.1 对项目的思考	25
5.2 需要作出的改进	25
VI. 参考文献	25

I. 问题的定义

1.1 项目背景

本项目来源于 kaggle 竞赛项目，最终目的是训练一个机器学习模型，输入一张图片来分辨图像中的猫和狗，也可是猫或狗的面部坐标甚至是身体的一部分。是一个典型的图像二分类问题。

本项目使用的卷积神经网络（CNN），相比于其他算法，卷积神经网络在图像识别领域具有更低的错误率，通过卷积神经网络达到的错误率非常接近人工标注的错误率，甚至机器比人更低的错误率。

本项目最终需要训练基于 CNN 的机器学习模型，对测试样本进行分类，并将最终结果上传 kaggle 进行最终评判。

1.2 输入数据

项目的数据集来源于 kaggle 竞赛的数据，该数据集搜集了 25000 张猫和狗的图片，训练集中包含 12500 张被标记为猫和 12500 张被标记成狗的图片，测试集包含 12500 张未标记的图片。

经过训练后模型需要预测出猫或狗的概率(1=狗, 0=猫)，如果是狗概率越接近 1，否则概率接近 0。

判断依据如果概率大于 0.5 判定为狗，概率小于 0.5 判定为猫，是个典型的二分类问题。

1.3 问题描述

1. 训练数据量比较大的问题

由于训练集的图片数量很庞大，如果将图片一次加载到内存中，一方面会导致加载时间过长，另一方面容易导致 OOM 内存溢出。因此采用 batch 进行小批量的加载数据解决该问题，这也是对训练精度与训练时间、资源的折中考虑。

2. 对模型的泛化的问题

虽然训练集的图片数量很庞大，但是仍然收集到所有猫狗的图片，因此需要提高模型的泛化性。否则，很有可能在训练集的效果很理想，但是在测试集效果一般的現象。

这里可以利用 Keras 的图像生成器(ImageDataGenerator)对图像进行翻转变换、镜像、拉伸等处理，提高训练集的数量进一步提高训练模型的复杂度。另一方面，也可以利用用户上传带有标记的图片，进一步扩充训练集的数量。从而达到提高模型泛化能力的目的。

3. 识别猫狗脸部的坐标或者身体的 mask

利用 OpenCV 或者 `tf.image.draw_bounding_boxes` 函数勾画出猫狗脸部的坐标以及身体的 mask, 这里比较难的是识别出身体的 mask，比如有的猫藏在某个位置，只露出很小一部分，或者和背景颜色很接近，或者图片模糊，甚至混进狗的图像掩盖了猫。那么想要识别出猫是非常困难的，且容易误判。



图 1 辨别困难的图像样本

4. 光线强弱等环境对识别效果的影响

和图像翻转类似，调整图像的亮度、对比图、饱和度和色相在很多图像识别的应用中都不会影响识别的结果。所以在训练神经网络模型，可以随机调整图像的这些属性，从而使训练得到的模型尽可能地受无关因素的影响。



图 2 高亮度和对比度图像

5. 图像编码处理

一张 RGB 色彩模式的图像可以看成一个三维矩阵，矩阵中的每个数代表图像的不同位置，不同颜色的亮度。但是图像在存储时并不是直接记录这些矩阵中的数字，而是记录经过压缩编码后的结果。所以在做识别时候需要将一张图像还原成一个三维矩阵，需要解码的过程。TensorFlow 提供了对 jpeg 和 png 格式图像编码/解码的方式。对猫狗 jpg 格式图像进行编码/解码。

6. 图像大小调整

数据集中的图像大小是不固定的，但是神经网络输入节点的个数是固定的。所以在将图像的像素作为输入之前，需要将图像的大小统计。有两种方式解决这个问题，一种方式是通过算法得到新的图像尽量保存原始图像上的所有信息，可以利用 TensorFlow 的 `tf.image.resize_image` 函数。另一种方式是对原始图像进行裁剪或者填充，利用 `tf.image.crop_to_bounding_box` 和 `tf.image.pad_to_bounding_box` 函数。当图像大于目标图像时进行裁剪，当图像小于目标函数时，对图像进行填充。从而达到统一图像的目的。

7. 异常数据的处理

训练集中绝大部分数据都是准确的，但是仍然混入了不少异常的图片，包括图片异和标签异常，对于训练集来说这些异常数据是需要剔除的。但是由于图片的数量太大，光依靠人工将异常图片筛选出来既费时也费力，因此我们采用预训练模型对训练集进行筛选过滤，剔除异常值。可以达到提高训练效果的目的。



图 3 异常图像

1.4 评价指标

对数损失 (Log loss) 亦被称为逻辑回归损失 (Logistic regression loss) 或交叉熵损失 (Cross-entropy loss)。交叉熵是常用的评价方式之一，它实际上刻画的是两个概率分布之间的距离，是分类问题中使用广泛的一种损失函数。

本文实际上是二分类问题，因此可以采用 logloss 损失函数作为评价指标，计算公式如下：

$$\text{LogLoss} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

采用交叉熵作为损失函数可以有效的解决梯度消失和梯度爆炸的问题。

为了进一步的对模型进行优化，可以采用交叉熵和 L2 正则化损失和，作为损失函数。

L2 正则化公式如下所示：

$$R(w) = \|w\|_2^2 = \sum_i |w_i^2|$$

完整的损失函数公式为：

$$\text{loss} = \text{LogLoss} + R(w)$$

II. 分析

2.1 数据的探索

猫狗的数据集分成 3 个文件 train.zip、test.zip 和 sample_submission.csv。

train 训练集包含了 25000 张猫狗的图片，每张图片包含图片本身和图片名。命名规则根据“类别.编号.jpg”，例如 dog.93.jpg。

test 测试集包含了 12500 张猫狗的图片，每张图片命名规则根据“编号.jpg”，需要注意的是测试集编号从 1 开始，而训练集的编号从 0 开始。

sample_submission.csv 需要将最终测试集的测试结果写入.csv 文件中，上传至 kaggle 进行打分。

由于训练集混合了猫狗的图片，因此我们首先需要对图片进行分类，分为 cat 和 dog 两个文件夹。

文件目录结构如下所示：

```
|── train2
|   └── dog
|       ├── dog.0.jpg
|       ├── ...
|       └── dog.12499.jpg
|   └── cat
|       ├── cat.0.jpg
|       ├── ...
|       └── cat.12499.jpg
└── test2
    └── test
        ├── cat.1.jpg
        ├── ...
        └── cat.12500.jpg
```

图 4 目录结构

训练集图片数量如下图所示：

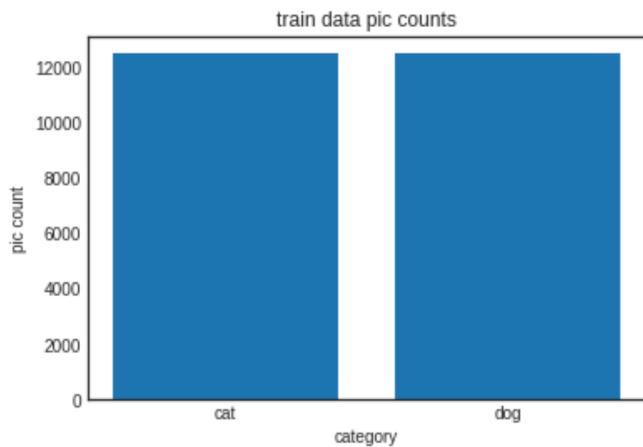


图 5 训练集猫狗数量比较

随机获取 16 张训练集图片：

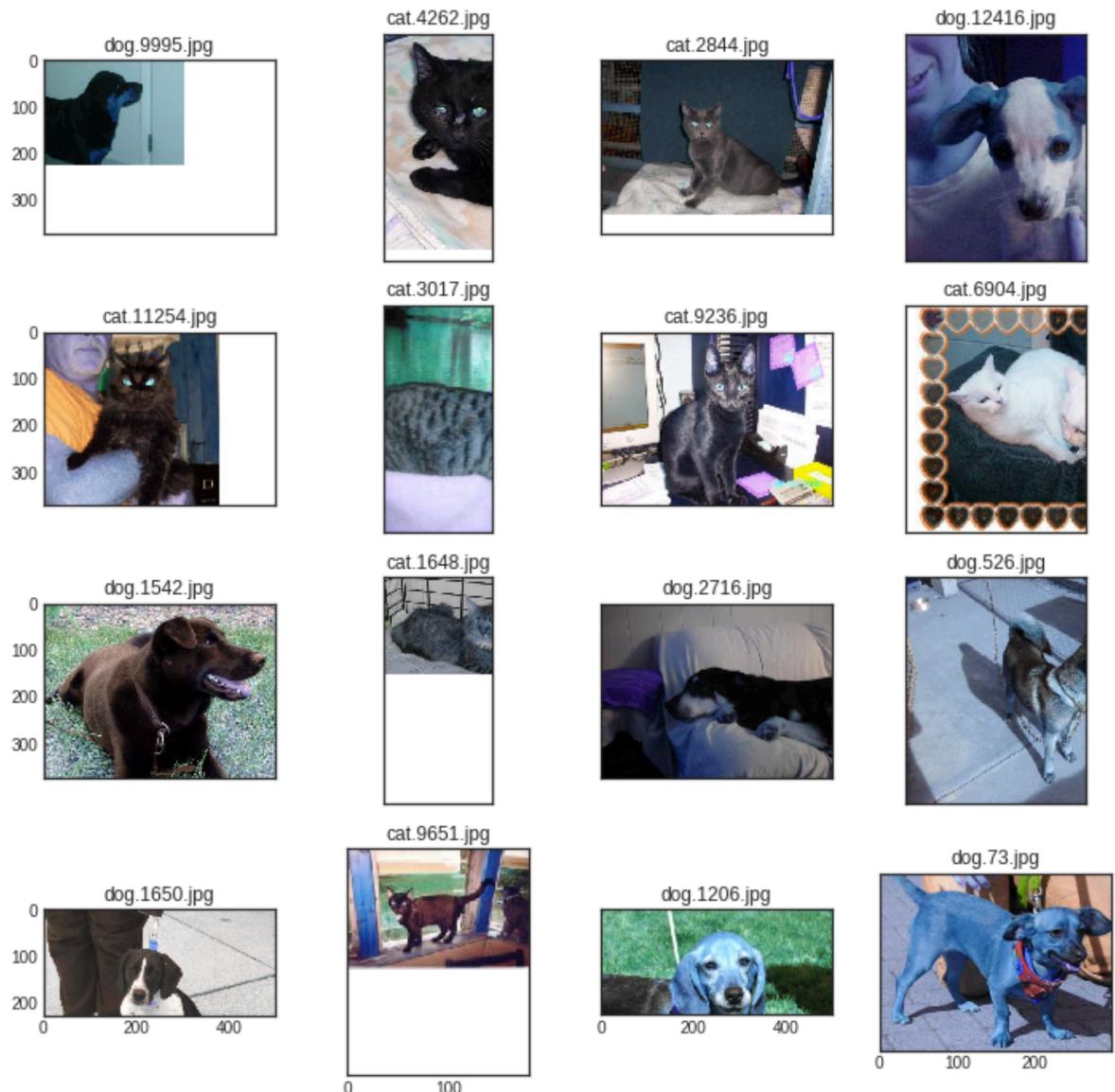


图 6 随机获取 16 张图片

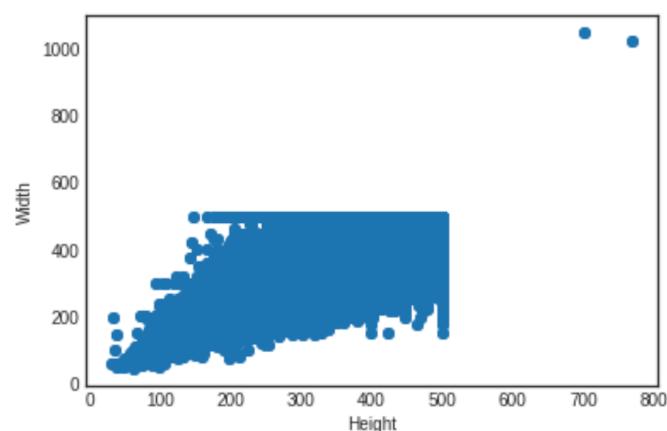


图 7 图片尺寸分布散点图

我们从上图中可以初步看出：

- 各个图片的尺寸不一致，尺寸范围高、宽 10-500 之间，有个别图片尺寸 700-1000.
- cat.3017.jpg 看到只是猫身体的一部分
- dog.12416.jpg 和衣服的颜色很相近
- dog.2716.jpg 只能识别狗的侧面

大致如 1.3 章节所描述。

本文采用预处理模型检测异常值结合人工筛选的方式进行异常值的排除，提高图片训练效果。

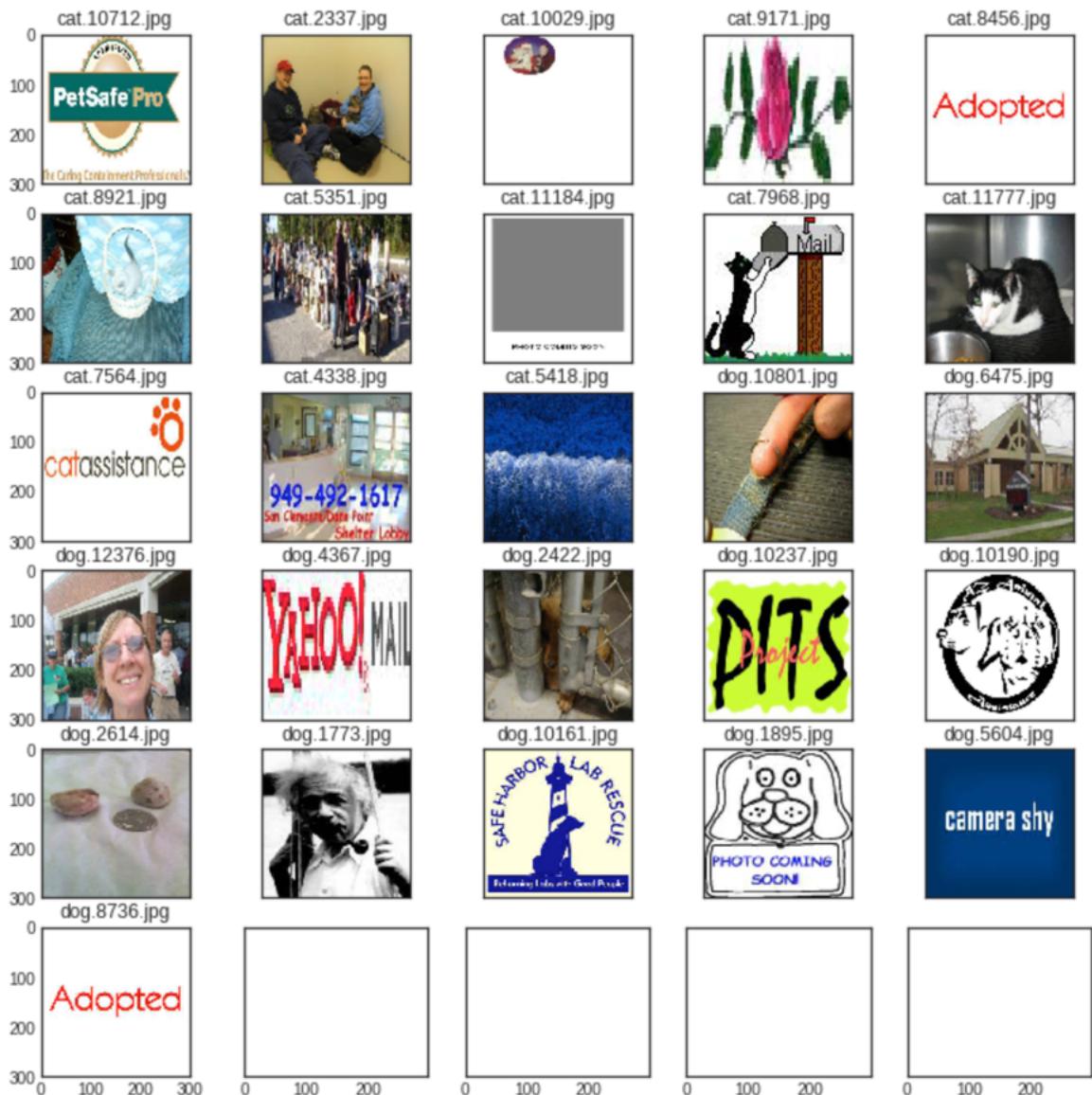


图 8 利用预处理模型检测异常值结果

2.2 算法和技术

2.2.1 分类算法

根据前面的分析可知，识别猫狗的本质上是一个分类问题。具体的算法包括感知机、支持向量机、决策树、逻辑回归、贝叶斯、神经网络等等方法。这些方法统称为机器学习算法。本文采用的算法的输入是一张张图片，每张图片由像素点组成，这些像素点所组成特征向量，而输出是一个概率。越接近 0 代表是猫，越接近 1 代表是狗。是一个典型的二分类问题。输入图片是一个特征空间，像素的个数是空间的维度。

支持向量机 SVM 是一个非常强大的分类模型，这种技术有很坚实的统计学理论基础，并在许多实际应用中（如手写数字识别、文本分类等）展现了很好的效果。此外 SVM 可以很好的应用于高维数据，避免维度灾难。它使用训练实例的一个子集表示决策边界，该子集被称为支持向量。¹SVM 的基本原理就是寻找具有最大边缘的超平面，使分类超平面离边界附近的样本距离尽可能的远。对于数据本身线性不可分的问题也能通过软边缘的方法克服，容许决策边缘的宽度和线性决策边界的训练错误数量之间的这种，但是因为误分样本的数量上没有限制。学习算法可能会找到边缘很宽的决策边界，且误分了许多训练实例。为了避免这个问题，必须要惩罚那些松弛变量值很大的决策边界。总的来说 SVM 还是很强大的模型。

决策树是一种简单但却广泛使用的分类技术。它的建模思路是尽量模拟人做决策的过程。因此，决策树与其他大多数机器学习模型不同，它几乎没有任何数学抽象，完全通过生成决策规则来解决了和回归问题。它会将原有特征空间分割成多个区域，尽可能的让每个区域的样本属于同一类，但是随着维度的增加，分割区域呈指数的上升，极易产生维度灾难。如果一张输入图片 $299*299*3$ 的彩色照片有 268203 个像素，即 268203 维。如果使用决策树简直是个灾难，因此决策树对于高维的数据效果不是很好。

2.2.2 神经网络

人工神经网络(ANN)的研究是由试图模拟生物神经系统而激发的。人类大脑主要是由称为神经元(neuron)的神经细胞组成，神经元通过叫做轴突(axon)的纤维丝连在一

¹ 《数据挖掘导论》 p156

起，当神经元受到刺激的时，神经脉冲通过轴突从一个神经元到另一个神经元。神经科学家发现，人的大脑通过同一个脉冲反复刺激下改变神经元之间的神经键连接强度进行训练。类似于人脑的结构，ANN 由一组相互连接的节点和有向链构成。

每个神经元由最简单的感知器构成，感知器包含两种结点：几个输入结点，用来表示输入属性；一个输出结点，用来表示模型输出。感知器对输入加权求和，再减去偏置因子，然后考察结果的符号，得到输出值。在数学上这是一个输入和权重的点积操作。符号函数，作为输出神经元的激活函数，常用的激活函数包括 sigmoid、softmax 和整流线性单元(ReLU)等，得到输出。

神经网络根据信息流的流向大概分为前馈神经网络(feedforward neural network)和循环神经网络(recurrent neural network)。前馈神经网络罗在模型的输出和模型本身之间没有反馈连接。当前馈神经网络被扩展成包含反馈连接时，被称为循环神经网络。

采用神经网络作为图像识别算法，需要建立图像和分类的映射关系，神经网络通过前向传播计算权重，通过反向传播算法更新权重。对于分类问题，可以使用训练数据和模型预测间的交叉熵作为代价函数。通过梯度下降算法更新权重的值，神经网络的非线性导致多数我们感兴趣的代价函数都变的非凸。这意味着通过梯度优化，使得代价函数达到非常小的值，而不是像训练线性回归模型的线性方程求解器，或者用于训练逻辑回归或 SVM 的凸优化那样保证全局收敛。

2.2.3 卷积神经网络

卷积网络也叫做卷积神经网络(convolutional neural network,CNN)，是一种专门处理具有类似网格结构的数据的神经网络，例如时间序列数据和图像数据。

传统的神经网络使用矩阵乘法来建立输入和输出的连接关系。其中，参数矩阵中每一个单独的参数都描述了一个输入单元与一个输出单元间的交互。这意味着每个输出单元与每个输入单元都产生交互。然而，卷积网络具有稀疏连接的特征。卷积神经网络，相邻的两层之间只有部分节点相连，为了展示每一层神经元的维度，一般会将每层卷积层的节点组织成一个三维矩阵。卷积神经网络和全连接神经网络的唯一区别就在于神经网络中相邻两层的连接方式。

卷积神经网络通过局部连接与权重共享大大减少了模型的参数个数，这意味着使用神经网络进行预测时的内存占用也会减少。内存占用的减少使得计算机能够训练层数更多、每一层神经元数目更多的神经网络。

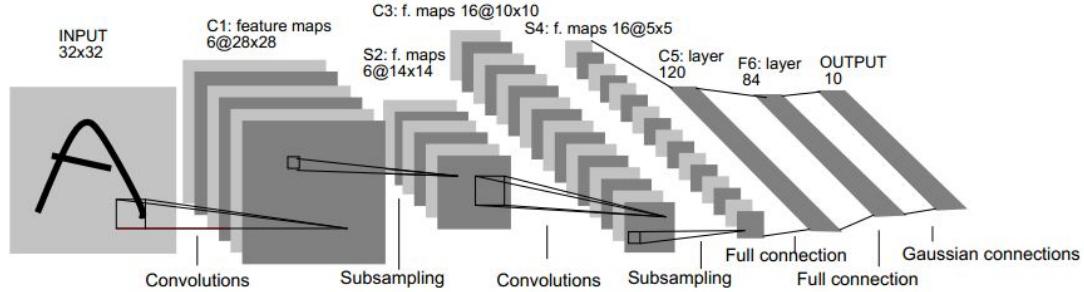


图 9 卷积神经网络 LeNet5 的基本结构

卷积神经网络主要由以下 5 个结构组成：

1. 输入层。输入层是整个神经网络的输入，在处理图像的卷积神经网络中，它代表了一张图片的像素矩阵。从输入层开始，卷积神经网络通过不同的神经网络结构将上一层的三维矩阵转化为下一层的三维矩阵，知道最后的全连接层。上图中，输入层为一张 32x32 的图片，深度代表颜色通道，黑白图片深度为 1，RGB 图片深度为 3。
2. 卷积层。卷积层中每个结点的输入只是上一层神经网络的一小块，常用的卷积核大小为 3x3 或者 5x5。卷积层试图将神经网络上的每一块进行更深入的提取从而得到抽象程度更高的特征。卷积神经网络深度越深一般效果越好。
3. 池化层。池化层不会改变三维矩阵的深度，但是它可以缩小矩阵的大小。实际上池化操作相当于将一张高分辨率的图片转化成低分辨率的图片。通过缩小最后全连接层中节点的个数，从而达到减少整个神经网络中参数的目的。
4. 全连接层，如上层所示，经过多次卷积层和池化层处理后，在卷积神经网络的最后一般会由 1 到 2 个全连接层来给出最后的分类结果。虽然前面已经对特征进行了提取，但是仍然需要全连接层来完成分类任务。
5. Softmax 层。Softmax 层主要用于分类问题，可以得到当前样本属于不同种类的概率分布情况。根据问题的不同也可以用其他层替代。

2.2.4 技术

项目使用 Keras 开源框架作为主要工具。Keras 是一个用 Python 编写的高级神经网络 API，它能够以 TensorFlow, CNTK, 或者 Theano 作为后端运行。Keras 的开发重点是支持快速的实验。能够以最小的时延把你的想法转换为实验结果，是做好研究的关键。

由于 Keras 底层可以通过 json 文件配置后端的运行模式，在不需要修改代码的情况下，在不同的后端环境下运行。具有以下几大优点：

- **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。

本项目的后端采用 Google 的 TensorFlow，它会将神经网络的计算流程转换成计算图。底层采用 C++ 编写的函数库，所以效率非常高。同时对 NVIDIA 显卡具有很好的支持，能过充分的利用多核 CPU 和 GPU 的优势。而且具有可视化的 TensorBoard 工具，能过一目了然的看到模型搭建效果。

2.3 基准模型

基准模型选择 VGG16，是 ImageNet2014 年非常流行的模型。

VGG 是一种由 K. Simonyan 和 A. Zisserman 提出的卷积神经网络模型，出自牛津大学的论文“非常深度的卷积网络用于大规模图像识别”¹。模型在 ImageNet 图像分类测试集中包含 1000 个分类 1400 多万张图片，其中每张图片属于且只属于一个分类。该模型达到了 92.7% 的 top-5 正确率。

VGG16 结构如下所示：

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

图 10 VGG16 结构图解

本文基准阈值参考 kaggle 排行榜 10%，也就是 logloss 要低于 0.06114。

2.4 设计大纲

本文采用迁移学习，所谓的迁移学习，就是讲一个问题上训练好的模型通过简单的调整使其适用于一个新的问题。目标模型采用 Inception-v3 模型。

Inception (GoogLeNet)是 Google 2014 年发布的 Deep Convolutional Neural Network。Inception-v3 是 2015 年提出的模型，跟前者比卷积神经网络模型的层数和复杂度都发生了巨大的变化。

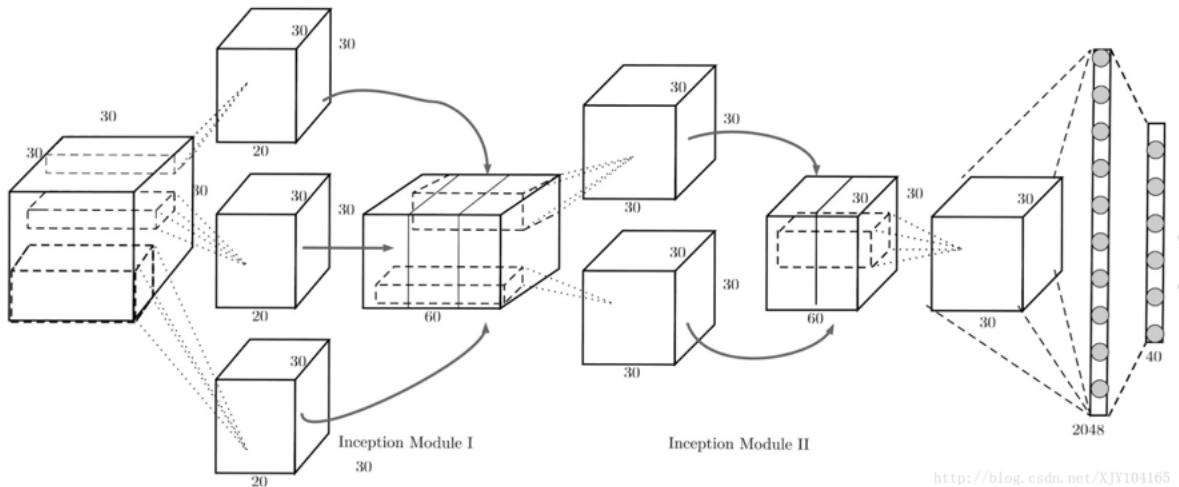


图 11 Inception 模块示意图

上图就是 2 个 Inception 模块，这种结构是一种和 LeNext-5 结构完全不同的卷积神经网络。在 LeNext-5 模型中，不同卷积层通过串联的方式连接在一起，而 Inception-v3 模型的 Inception 结构将不同的卷积层通过并联的方式结合起来。

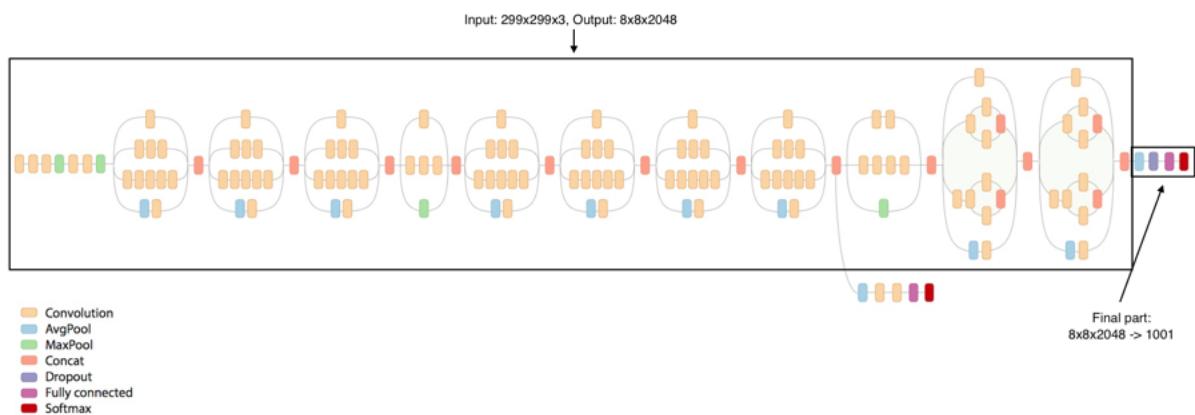


图 12 Inception-v3 模型架构图

Inception-v3 模型总共有 46 层，由 11 个 Inception 模块组成。在 Inception-v3 模型中有 96 个卷积层²。在 Inception-v3 模型中将 3x3 的卷积层替换成 3x1 和 1x3 卷积，相当于以 3x3 卷积相同的感受野滑动两层网络。如果输入和输出滤波器数量相同，那么对于相同数量的输出滤波器，两层解决方案会减少 33% 的计算量。如果采用 3x3 卷积的方案总共需要 9 次乘法和 9+1 次加法运算。如果采用 3x1 接 1x3 方案，那么总共需要 6 次乘法和 6+2 次加法运算。单从乘法角度节省了 33% 左右。如果采用 2 个 2x2 的方案表示仅节省了 11% 的计算量。因此 inception v3 采用 3x1 和 1x3 的方案是很高效的。

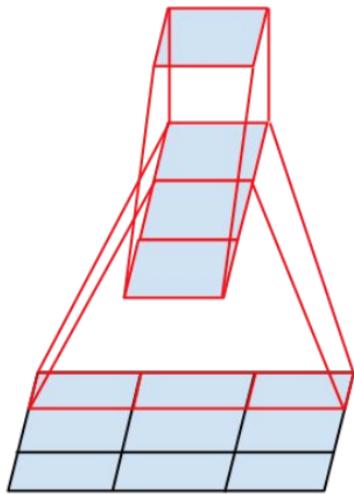


图 13 替换 3x3 卷积的 mini 网络

在 Inception-v3 的实际测试结果中，Top-1 的错误率为 4.2%，Top-5 的错误率为 18.77%，与其他的几种卷积神经网络来说错误率低了不少。

Network	Models Evaluated	Crops Evaluated	Top-1 Error	Top-5 Error
VGGNet [18]	2	-	23.7%	6.8%
GoogLeNet [20]	7	144	-	6.67%
PReLU [6]	-	-	-	4.94%
BN-Inception [7]	6	144	20.1%	4.9%
Inception-v3	4	144	17.2%	3.58%*

图 14 2012-2015 年 ILSVRC 综合比较结果

考虑到 Inception-v3 模型非常庞大，需要耗费大量时间和精力。如果自己重新去训练没有足够的分类数据，因此我们采用迁移学习，直接使用 ImageNet 图形训练后的 Inception-v3 作为模型，作为本文项目的识别算法。这样可以节约对复杂的卷积神经网络训练的时间，这里可能是几天甚至几周。同时也解决了收集大量数据集和标注数据集所耗费的时间。

根据论文 DeCAF:A Deep Convolutional Activation Feature for Generic Visual Recognition 中的结论，可以保存训练好的 Inception-v3 模型中所有卷积层的参数，只是替换最后一层全连接层。在最后一层全连接层之前的网络层称之为瓶颈层。

将新的图形通过训练好的卷积神经网络直到瓶颈层的过程可以看成是对图像进行特征抽取的过程。在训练好的 Inception-v3 模型中，因为将瓶颈层的输出在通过一个

单层全连接层神经网络可以很好的区分 1000 种类别的图像。由于本项目只需要识别猫和狗实际上是一个二分类问题，因此最后再加上一个 softmax 层或者 sigmoid 层。于是，在新的数据集上，可以直接利用训练好的神经网络对图像进行特征提取，然后再讲提取得到的特征向量作为输入来训练一个新的单层全连接神经网络处理新的分类问题。

一般情况下，在数据量足够的情况下，迁移学习的效果不如完全重新训练。但是迁移学习所需要的训练时间和训练样本数远远小于训练完整的模型。

III. 方法

3.1 数据预处理

3.1.1 剔除异常数据

本文采用预处理模型检测异常值，利用已有的模型和权重文件进行分类。ImageNet 对于狗进行了 118 个细分类，猫 7 个分类，本文分别测试了 Top-10,Top-20,Top-30,Top-50。在 top10-30 之间大概能得到 10 张左右的异常值，而在 Top-50 的时候检测出 26 张异常值图片，再结合人工筛选的少部分结果进行补充。

```

1 img_size = (299, 299)
2 ## 载入预先下载好的ImageNet分类csv文件
3 import csv
4 from keras.preprocessing import image
5 from keras.applications.inception_v3 import preprocess_input
6
7 def get_imageNet_class(file_path:str)->tuple:
8     category_class = []
9     with open(file_path,'r') as f:
10         reader = csv.reader(f)
11         for line in reader:
12             if(line[1] == '猫') or (line[1] == '狗'):
13                 category_class.append(line[0])
14
15     return (category_class)
16
17 def conv_img_path(img_arr:np.ndarray)->tuple:
18     dir_path = []
19     for i in train_total:
20         dir_path.append('imgs/train/%s' % i)
21     return (dir_path)
22
23 def read_image(img_path:str)->tuple:
24     img = image.load_img('%s' % img_path,target_size=img_size)
25     x = image.img_to_array(img)
26     x = np.expand_dims(x, axis=0)
27     x = preprocess_input(x)
28
29 def read_image_dir(dir_list:list)->tuple:
30     preprocess_imgs = [read_image(fp) for fp in tqdm(dir_arr)]
31     return (preprocess_imgs)
32
33 imageNet_class = get_imageNet_class('ImageNetClasses.csv')
34
35 #转换路径
36 dir_arr = conv_img_path(train_total)
37 #获取归一化后图片
38 preprocess_imgs = read_image_dir(dir_arr)
39
20%|██████████| 4881/25000 [00:09<00:40, 491.56it/s]/home/mj/.conda/envs/ten/lib/python3.6/site-packages/tqdm/_monitor.py:89: TqdmSynchronisationWarning: Set changed size during iteration (see https://github.com/tqdm/tqdm/issues/481)
TqdmSynchronisationWarning)
100%|██████████| 25000/25000 [00:51<00:00, 488.15it/s]

```

图 15 加载需要检测的图片并进行归一化处理

```

1 from keras.applications.inception_v3 import decode_predictions
2
3 def predict_model(model, imgs, top = 10):
4     pred = model.predict(imgs)
5     return (decode_predictions(pred,top=top)[0])
6
7 def get_predict(model, imgs, top=10):
8     result = [predict_model(model, x, top) for x in tqdm(imgs) ]
9     return(result)
10
11 model= InceptionV3(weights='imagenet')
12
13 ##预测模型
14 pred = get_predict(model, preprocess_imgs, 50)
2%||          | 462/25000 [00:09<08:45, 46.69it/s]/home/mj/.conda/envs/ten/lib/python3.6/site-packages/tqdm/_monitor.py:89: TqdmSynchronisationWarning: Set changed size during iteration (see https://github.com/tqdm/tqdm/issues/481)
TqdmSynchronisationWarning)
100%|██████████| 25000/25000 [07:43<00:00, 53.89it/s]

```

图 16 预测 Top-50 的结果

```

1 #删除异常值
2 exp_arr = np.array(exp_flist)
3 nor_arr = np.array(['cat.8921.jpg','cat.11777.jpg','cat.2337.jpg'])
4 manual_arr = np.array(['cat.92.jpg','cat.7377.jpg','cat.4085.jpg'])
5
6 need_delete_arr = []
7 for file_name in exp_arr:
8     if file_name not in nor_arr:
9         need_delete_arr.append(file_name)
10
11 need_delete_arr = np.concatenate((np.array(need_delete_arr),np.array(manual_arr)))
12 need_delete_arr

array(['cat.10712.jpg', 'cat.10029.jpg', 'cat.9171.jpg', 'cat.8456.jpg',
       'cat.5351.jpg', 'cat.11184.jpg', 'cat.7968.jpg', 'cat.7564.jpg',
       'cat.4338.jpg', 'cat.5418.jpg', 'dog.10801.jpg', 'dog.6475.jpg',
       'dog.12376.jpg', 'dog.4367.jpg', 'dog.2422.jpg', 'dog.10237.jpg',
       'dog.10190.jpg', 'dog.2614.jpg', 'dog.1773.jpg', 'dog.10161.jpg',
       'dog.1895.jpg', 'dog.5604.jpg', 'dog.8736.jpg', 'cat.92.jpg',
       'cat.7377.jpg', 'cat.4085.jpg'], dtype='<U13')

1 #删除操作
2 for file_name in need_delete_arr:
3     path = 'imgs/train/%s' %file_name
4     if os.path.exists(path):
5         os.remove(path)
6     else:
7         print ('no such file:%s'% file_name)

```

图 17 删除异常值图片

3.1.2 调整文件目录

根据数据探索部分内容，使用 Keras 的图像生成器进行图像生成需要对文件进行分类，按照探索结果对文件夹进行创建，将分类后的图片拷贝到新的文件目录下。

```

1 after_delete_train_cat = [img for img in filter(lambda x:x[:3] == 'cat', train_filenames)]
2 after_delete_train_dog = [img for img in filter(lambda x:x[:3] == 'dog', train_filenames)]
3
4 def build_mkdir(dirname):
5     if os.path.exists(dirname):
6         #delete old direct
7         shutil.rmtree(dirname)
8         os.mkdir(dirname)
9
10    build_mkdir('imgs/train2')
11    os.mkdir('imgs/train2/cat')
12    os.mkdir('imgs/train2/dog')
13
14    #copy files to new path
15    def copy_to_newPath(oldPath:str,newPath:str,imgName:list)->None:
16        for img in imgName:
17            pathA = oldPath + img
18            pathB = newPath + img
19            shutil.copyfile(pathA,pathB)
20
21    copy_to_newPath('imgs/train/','imgs/train2/cat/',after_delete_train_cat)
22    copy_to_newPath('imgs/train/','imgs/train2/dog/',after_delete_train_dog)
23

```

图 18 预处理-生成新的文件目录

3.1.2 生成图片及归一化处理

```

1 train_path = "imgs/train2"
2 test_path = "imgs/test2"
3
4 img_size = (299, 299)
5
6 gen = ImageDataGenerator()
7 X_train_gen = gen.flow_from_directory(train_path, img_size, shuffle = False,
8                                         batch_size = 16)
9 X_test_gen = gen.flow_from_directory(test_path, img_size, shuffle = False,
10                                       batch_size = 16, classes = None)

Found 24974 images belonging to 2 classes.
Found 12500 images belonging to 1 classes.

```

图 19 预处理-生成图片和归一化处理

由于原始图片的尺寸不一致，需要将载入的图片调整成相同的大小 299x299。输入图片的长宽 299 根据论文测试结果，因为在 299x299 的输入下预测出来的结果最佳。上图中分别对训练集和测试集进行归一化处理，`flow_from_directory` 底层实现会将图片的经过归一化处理后锁定在(-1,1)之间。具体的操作会对像素除以 127.5 后减去 1 达到归一化处理，测试集并不需要将图片打乱处理，预测集需要打乱次序，但是本文中将这部操作放在训练阶段，可以节约一次打乱次序所花费的时间。

3.2 执行过程

3.2.1 构建模型

```

#基础模型
input_tensor = Input((img_size[0], img_size[1], 3))
input_tensor = Lambda(inception_v3.preprocess_input)(input_tensor)
base_model = InceptionV3(input_tensor = input_tensor,
                           weights = 'imagenet', include_top = False)
base_model = Model(base_model.input, GlobalAveragePooling2D()(base_model.output))
base_model.summary()

```

图 20 构建 Inception v3 基础模型

构建 Inception 权重导出的模型，并在后面追加一层平均池化。因为我们模型后面会使用全连接层，因此在这里添加一层 AveragePooling，因为在多分类任务的时候，假设途中出现了一只狗和一个篮球。狗很大，篮球很小。我们应该输出狗而不是篮球。

假设狗对应的特征最大值为 2，但是激活图上的平均值为 1.3，篮球的特征最大值是 3，但是平均值为 0.5。如果用 AveragePooling 可以输出狗，但是用 MaxPooling 则会输出篮球。因此在多分类输出的时候通常会使用 AveragePooling。

3.2.2 使用 Inception v3 预训练模型

使用 Inception v3 预训练模型对图像进行预测。

```
1 X_train = base_model.predict_generator(X_train_gen, verbose=1)
2 X_test = base_model.predict_generator(X_test_gen, verbose=1)

1561/1561 [=====] - 125s 80ms/step
782/782 [=====] - 62s 79ms/step
```

图 21 训练特征向量

3.2.3 保存特征的权重

保存训练好的权重，按照类别进行区分。这一步作用是将权重固定下来，不用每次执行的时候重新做预训练，节约宝贵的时间。

```
with h5py.File('saved_models/weights.inv3.hdf5') as fp:
    fp.create_dataset('train', data = X_train)
    fp.create_dataset('test', data = X_test)
    fp.create_dataset('label', data = X_train_gen.classes)
```

图 22 导出特征权重

3.2.4 导入训练好的特征向量

```
X_train = []
X_test = []
with h5py.File('saved_models/weights.inv3.hdf5', 'r') as fp:
    X_train.append(np.array(fp['train']))
    X_test.append(np.array(fp['test']))
    y_train = np.array(fp['label'])

X_train = np.concatenate(X_train, axis=1)
X_test = np.concatenate(X_test, axis=1)
X_train, y_train = shuffle(X_train, y_train)
```

图 23 执行-导入特征向量

3.2.5 添加全连接层和 dropout 层

```
1 input_tensor = Input(X_train.shape[1:])
2 inception_v3_model = Model(input_tensor, Dropout(0.2)(input_tensor))
3 inception_v3_model = Model(inception_v3_model.input, Dense(1, activation = 'sigmoid')(inception_v3_model.output))
4 inception_v3_model.summary()
```

图 24 添加 dropout 层和全连接层

模型的输出需要添加一层 dropout 和全连接层，激活函数采用 sigmoid 因为是二分类问题，当然也能用 softmax，但是最后的优化器也需要相应的调整。Dropout 智力选择 0.2 是通过多次尝试得到的最优结果。

3.2.6 编译模型

```
1 inception_v3_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

图 25 编译模型

优化器选择 adam 算法，是由 Kingma 和 Lei Ba 两位学者在 2014 年提出的，它和传统的随机梯度下降不同。结合了 AdaGrad 和 RMSProp 两种优化算法的优点，传统的随机梯度下降单一的学习率更新所有的权重，学习率在训练过程中并不会改变。而 adam 通过计算梯度的一阶矩阵和二阶矩阵不同的参数设计独立自适应的学习率，计算出更新步长，从而能够更快的收敛。

主要包括以下几个显著的优点：

1. 实现简单，计算高效，对内存需求
2. 参数的更新不受梯度的伸缩变换影响
3. 超参数具有很好的解释性，且通常无需调整或仅需很少的微调
4. 更新的步长能够被限制在大致的范围内，即很快的收敛
5. 能自然地实现步长退火过程，即自适应的学习率
6. 很适合应用于大规模的数据及参数的场景
7. 适用于不稳定目标函数
8. 适用于梯度稀疏或梯度存在很大噪声的问题

损失函数采用 brinary_crossentropy 是因为我们采用了 sigmoid 作为二分类的激活函数，因此损失函数也需要对应选择。

3.2.7 训练模型

```
1 epochs = 10
2 batch_size = 128
3 checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.inv3.hdf5',
4                                 verbose=1, save_best_only=True)
5 history_tarin = inception_v3_model.fit(X_train, y_train, validation_split = 0.2,
6                                         epochs = epochs, batch_size = batch_size, verbose=1,
7                                         callbacks=[checkpointer])
```

图 26 训练模型

训练模型同时对训练集进行分割，20%作为验证集，80%作为训练集。训练结果通过回调 callback 保存成 hdf5。

3.2.8 模型参数调优

通过尝试不同的 dropout 保留率对单一参数进行优化。

```

1 epochs = 10
2 batch_size = 128
3
4 dropout_info = []
5
6 def modify_dropout_rate(rate):
7     input_tensor = Input(X_train.shape[1:])
8     inception_v3_model = Model(input_tensor, Dropout(rate)(input_tensor))
9     inception_v3_model = Model(inception_v3_model.input, Dense(1, activation = 'sigmoid')(inception_v3_model.output)
10    inception_v3_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
11    history_tarin = inception_v3_model.fit(X_train, y_train, validation_split = 0.2,
12                                          epochs = epochs, batch_size = batch_size, verbose=1)
13
14    return (rate,history_tarin.history['val_loss'],history_tarin.history['val_acc'])
15
16 dropout_result= []
17 for rate in np.arange(0.1,1,0.1):
18     rate_data= modify_dropout_rate(round(rate,1))
19     dropout_result.append(rate_data)

```

图 27dropout 0.1~0.9 调参

	Avg	Max	Min
dropout rate			
0.1	0.023771	0.038840	0.020481
0.2	0.023003	0.035460	0.019824
0.3	0.023591	0.038748	0.020342
0.4	0.025762	0.045979	0.020318
0.5	0.024320	0.040121	0.020563
0.6	0.024787	0.039904	0.021356
0.7	0.024945	0.041661	0.020169
0.8	0.026067	0.045350	0.021988
0.9	0.030635	0.052847	0.024387

图 28 不同 dropout 的 logloss 值

从上图中我们可以看出当 dropout 值为 0.2 的时候，Logloss 误差最小。因此我们利用测试结果调整之前模型的 dropout 值，作为最终预测的模型。

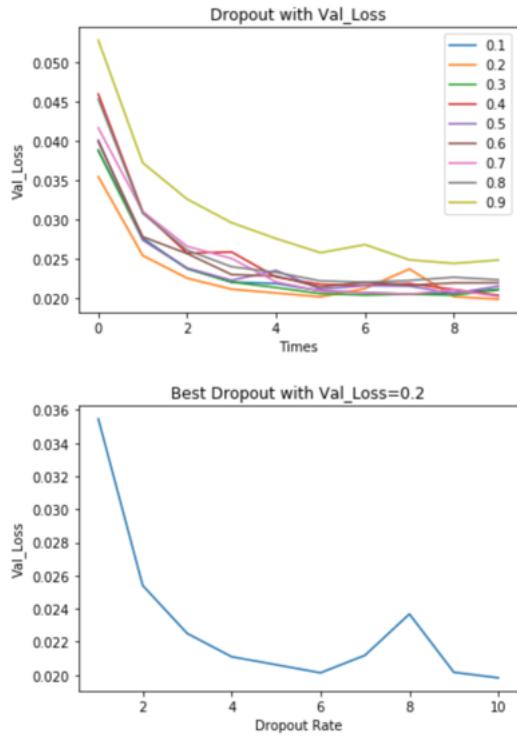


图 29 不同 dropout 下的 logloss 曲线图

3.3 完善

- 本文采用了单一的预处理模型进行异常值检测，但是仅仅依靠一个模型检测出所有的异常值有点困难，后期考虑采用多模型共同检测异常值的方式进行改进。
- 还有对模型参数的优化，通过不同的优化器的尝试得到更好的优化结果。
- 单一的模型根据相关论文测试结果大约保持 96% 左右的准确率，因此考虑和其他模型相结合的方式来降低 LogLoss。

IV. 结果

4.1 模型的评价和验证

```

Train on 19979 samples, validate on 4995 samples
Epoch 1/10
19979/19979 [=====] - 1s 41us/step - loss: 0.1336 - acc: 0.9603 - val_loss: 0.0412 - val_ac
c: 0.9914

Epoch 00001: val_loss improved from inf to 0.04118, saving model to saved_models/weights.best.inv3.hdf5
Epoch 2/10
19979/19979 [=====] - 0s 15us/step - loss: 0.0343 - acc: 0.9919 - val_loss: 0.0271 - val_ac
c: 0.9938

Epoch 00002: val_loss improved from 0.04118 to 0.02712, saving model to saved_models/weights.best.inv3.hdf5
Epoch 3/10
19979/19979 [=====] - 0s 18us/step - loss: 0.0260 - acc: 0.9931 - val_loss: 0.0221 - val_ac
c: 0.9942

Epoch 00003: val_loss improved from 0.02712 to 0.02212, saving model to saved_models/weights.best.inv3.hdf5
Epoch 4/10
19979/19979 [=====] - 0s 22us/step - loss: 0.0226 - acc: 0.9936 - val_loss: 0.0202 - val_ac
c: 0.9944

Epoch 00004: val_loss improved from 0.02212 to 0.02018, saving model to saved_models/weights.best.inv3.hdf5
Epoch 5/10
19979/19979 [=====] - 0s 24us/step - loss: 0.0197 - acc: 0.9936 - val_loss: 0.0186 - val_ac
c: 0.9942

Epoch 00005: val_loss improved from 0.02018 to 0.01863, saving model to saved_models/weights.best.inv3.hdf5
Epoch 6/10
19979/19979 [=====] - 0s 25us/step - loss: 0.0189 - acc: 0.9943 - val_loss: 0.0179 - val_ac
c: 0.9944

Epoch 00006: val_loss improved from 0.01863 to 0.01792, saving model to saved_models/weights.best.inv3.hdf5
Epoch 7/10
19979/19979 [=====] - 1s 28us/step - loss: 0.0183 - acc: 0.9941 - val_loss: 0.0175 - val_ac
c: 0.9942

Epoch 00007: val_loss improved from 0.01792 to 0.01750, saving model to saved_models/weights.best.inv3.hdf5
Epoch 8/10
19979/19979 [=====] - 1s 28us/step - loss: 0.0169 - acc: 0.9950 - val_loss: 0.0172 - val_ac
c: 0.9942

Epoch 00008: val_loss improved from 0.01750 to 0.01721, saving model to saved_models/weights.best.inv3.hdf5
Epoch 9/10
19979/19979 [=====] - 1s 29us/step - loss: 0.0158 - acc: 0.9946 - val_loss: 0.0171 - val_ac
c: 0.9946

Epoch 00009: val_loss improved from 0.01721 to 0.01707, saving model to saved_models/weights.best.inv3.hdf5
Epoch 10/10
19979/19979 [=====] - 1s 31us/step - loss: 0.0153 - acc: 0.9946 - val_loss: 0.0170 - val_ac
c: 0.9942

Epoch 00010: val_loss improved from 0.01707 to 0.01698, saving model to saved_models/weights.best.inv3.hdf5

```

图 30 Inception v3 epoch10 次训练结果

从上图可以看出在 10 次训练中准确率不断提升，logloss 不断下降。训练集和验证集误差较接近，说明没有过拟合，保证了模型的健壮性。同时通过预训练，大大降低了训练的时间，因为只更新了最后全连接层的权重，大大加快了训练速度。

```

1 plt.plot(history_tarin.history['val_loss'])
2 plt.xlabel('time')
3 plt.ylabel('val_loss')
4 plt.show()
5
6 plt.plot(history_tarin.history['val_acc'])
7 plt.xlabel('times')
8 plt.ylabel('val_acc')
9 plt.show()

```

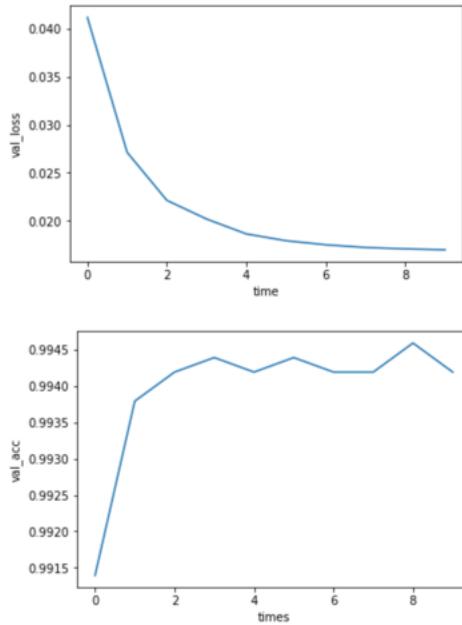


图 31 训练集的学习曲线及准确率

最后通过预测集提交至 kaggle 进行最终的验证。

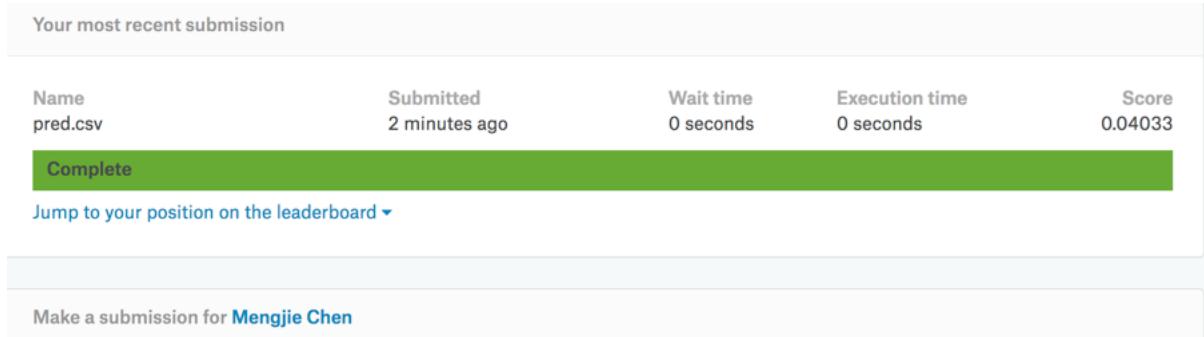


图 32kaggle 提交结果

测试集 Kaggle 得分 0.04033， 小于 0.06127 最终进入 top10%。

4.2 合理性分析

从训练模型的过程中能够很好的收敛，同时对预测的结果得到很不错的 LogLoss，最终达到 top 10。说明方案是合理的。

V. 项目结论

5.1 对项目的思考

通过本文当项目对迁移学习有了一定的了解，采用迁移学习的方式能够节省用户大量的时间和精力去搜集数据，训练数据。通过预训练的方式，能够快速的得到一个理想的模型。

但是仍然需要尝试很多可能的情况，在不同的场景对迁移学习也要有对应的改变，比如说预测图片与迁移学习模型相差较远，那么可能效果不是很理想，这时候需要采用其他的模型或者开放更多的卷积层重新进行训练。

总的来说迁移学习在图像处理领域还是大有可为的，但是也由于模型已经预先固化，想要进行改进也是困难重重。

5.2 需要作出的改进

- 尝试更多的模型融合，得到更好的效果。
- 增大训练集，比如旋转图像、镜像等降低过拟合现象。
- 采用更好的优化算法，提升预测效果。
- 采用多模型的异常值检测方法。

VI. 参考文献

[1] Karen Simonyan and Andrew Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. At ICLR,2015.

[2] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng and Trevor Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In ICML,2014.

[3] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In arXiv,2015.

1 Karen Simonyan and Andrew Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. At ICLR,2015.

2 Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In arXiv,2015.