基于Tensorflow的Android端相册分类App设计与实现

摘要：图像的识别和分类在计算机视觉领域是一个久经不衰的研究课题。大量的App应用对相册分类进行了尝试。也通过而利用性能相对羸弱的移动端设备，达到类似的分类效果是亟待解决的问题。而TensorflowLite框架能够在Android端良好运行的神经网络训练框架。通过构建一个App在Android端运行该框架，移植经典图像处理的卷积神经网络在Android端进行相册图像识别并分类可以较好得解决问题。

英文版Abstract：

XXXXXX

引言：

如果手机中相片数量非常庞大，那么用户寻找到一张特定的图像或者一类图像就会非常困难。自动相册分类是解决这个问题的重要途径，也是很多相册类应用的基本功能。目前，由于移动端的算力限制，相册分类实现都是将相册共享到云端，由高算力服务器进行图像分类并对照片打上标签，重新下发到移动端，再由移动端根据标签完成图像分类。这样的做法缺点非常明显，没有连接互联网就会无法进行分类，并且无法对新添加的照片立即识别，实效性差。解决上述缺陷，需要在移动端本地对图像进行图像分类。其最大的困难在于，如何利用移动端有限的算力，达到和服务器类似的分类效果。2017年，谷歌针对移动端推出了名为TensorflowLite的神经网络框架。可以通过该框架，在移动端运行InceptionV3图像识别神经网络，并达到较高地准确率。

本文目标基于TensorflowLite框架和相关图像识别算法，开发一款可以由移动设备独立进行图像分类，并且兼顾时效性和有一定准确度的移动端应用软件。接下来将围绕相册分类软件App的结构设计和实现，图像识别神经网络的训练和移植，图像分类功能的验证三方面来论述。

1 相册分类APP的主要结构

相册分类App（以下简称App）的主要遵循软件工程中经典的Model-View-Presenter（MVP）模式。视图（View）层，逻辑（Presenter）层，数据（Model）层三层彼此独立解耦，从而让软件获得较好的拓展性和鲁棒性。

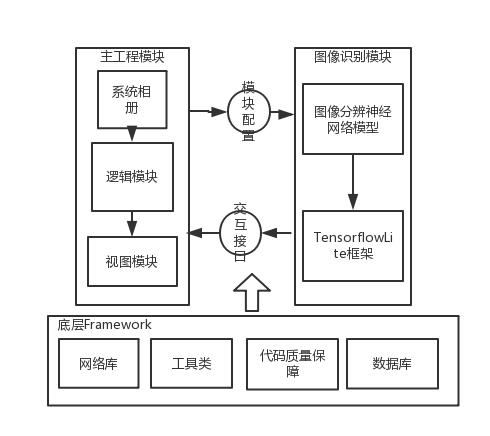


图1 App的主要架构

App主要由图1所示的三大模块组成：

分别是主工程模块、图像识别模块以及基础层Framework模块。

其中，主工程模块负责承载app的界面显示和逻辑，并初始化图像识别模块和基础层模块。通过读取系统相册，将所有的系统内所有的图片路径管理在一个字典（dict）中，通过图像识别模块给所有图片打上分类标签。并按标签并保存在数据库。最后通过分类标签，进行图片归类并通过视图界面呈现

基础层Framework模块向所有模块提供通用基础功能，比如数据库访问，代码编写中的工具类，网络库等。其中，工具类包括，字符工具类、图像工具类、url拼接工具类等。另外，为了保证代码质量，通常会编写单元测试进行代码逻辑检查或者一些Lint静态脚本来保证代码的书写规范。

图像识别模块是本文的核心功能模块，是真正进行图片识别和分类的模块。主工程会在App初始化阶段向图像识别模块注入配置参数并初始化。而图像识别模块仅仅向主工程暴露自己的功能接口，在App具体表现识别某一张图片并返回该图片类别的概率数据。

图像识别模块通过在Android上执行TensorflowLite框架，并在该框架上运行图像分类神经网络InceptionV3来实现图像分类的主要功能。本文接下来将通过InceptionV3图像分类神经网络的迁移训练，以及如何在Android端运行TensorflowLite框架两部分来论述图像识别功能的实现。

2分类模型的选取

图像分类是本文的核心功能，其中起到主要图像分类作用的是图像分类神经网络。表1展示了近年经典图像分类模型模型的错误率。可以看出自从2012年，ImageNet的ILSVRC图像识别挑战赛创办以来，图像识别神经网络的准确率有了长足的进步。

InceptionV3功耗适中，图像识别的准确率很高。神经网络的最后一层一般为一个全连接层，而在这一层全连接层之前的神经网络，被称作瓶颈层(BottleNeck)。已经通过训练的Inception-V3模型性能优秀，其瓶颈层仅仅通过一个单层的全连接网络就可以区分1000种类别的图像。这说明InceptionV3的瓶颈层输出可以作为图像简练且表义明确的特征向量，并可以利用该特征向量进行进一步的分类。

|  |  |  |
| --- | --- | --- |
| 年份 | 模型名称 | Top5错误率 |
| 2012 | AlexNet | 15.3% |
| 2013 | ZF Net | 14.8% |
| 2014 | GoogleNet(InceptionV1) | 6.67% |
| 2015 | InceptionV3 | 5.6% |
| 2016 | MobileNet(InceptionV4) | 5.0% |

表1 历年图像分类模型Top5错误率

3模型的迁移训练

训练InceptionV3图像分类的数据集有120万被标注的图片，收集图片数据集，并且利用数据集重新训练一个模型会非常费时间，可能需要几天甚至几周。为了解决训练一个全新的神经网络而产生的效率问题，迁移学习应运而生。

迁移学习就是将已经在某个问题上训练好的模型，通过相对简单的调整，使其能够适应新的问题。在保留已经训练好神经网络瓶颈层输出基础上，通过替换最后一层全连接层，可以让该模型解决新的图像分类问题。从上文可知，IncetionV3的瓶颈层具有良好的良好性能。通过迁移训练，可以让InceptionV3瓶颈层输出和全连接层组成的神经网络获得识别特定类目的图像的能力。

全连接层的训练网络如图2所示，利用不断迭代前向传播，后向传播更新参数，最后让交叉熵收敛是全连接网络的基本思路。综合神经网络训练以及其前后期工作，整个迁移训练的关键步骤归纳如下：

1、图像预处理和数据集获取

2、获取瓶颈层输出并缓存

3、定义全连接层

4、训练全连接层

5、模型性能数据的输出

接下来将对这几个步骤进行详细阐述。

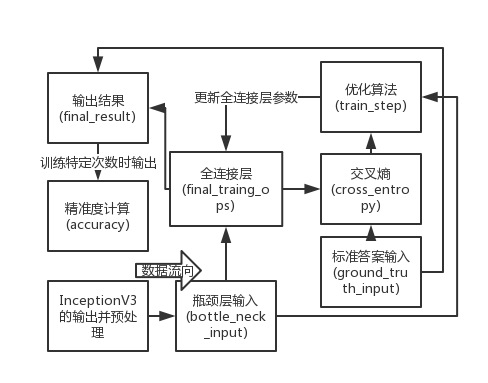


图2 全连接层训练网络

3.2图像预处理和数据集获取

虽然迁移学习对于数据集的数量要求相较于训练一个全新的模型，对图像数据集数量的要求大大降低，但是对于一个单一的类目识别，同样需要数百个样本图像。

通过基于python语言编写爬虫脚本是获取图片数据集的重要途径。图2展示了爬虫脚本的主要流程。整个爬虫脚本由config.py，spider.py，deal\_image.py三个文件组成。其中config.py是配置文件，spider.py是爬取脚本，deal\_image是图像处理脚本。

其主要原理是通过Baidu搜图等图像搜索引擎发起搜索请求，解析其搜索结果，并逐个下载图片，并对图片进行规范化裁切和格式化，最后进行部分人工筛选以保证数据的可用性。

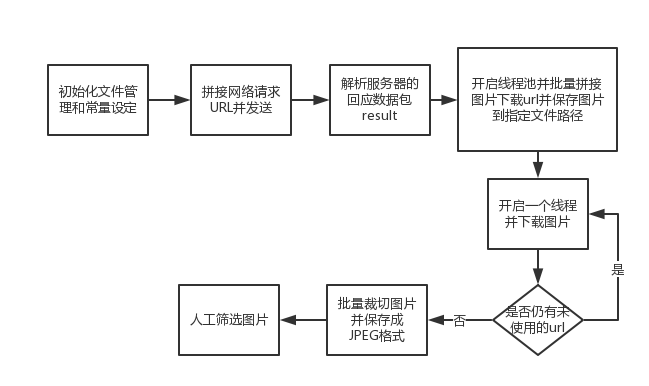


图2 图像爬虫脚本的主要流程

3.3获取瓶颈层的输出并缓存

瓶颈层的输出是利用InceptionV3模型对图像数据集进行处理，可以获得每张图片的特征向量，来作为全连接层的输入。为了运行效率的提升，需要配合缓存机制对图片进行存取。数据的流向如图3所示。

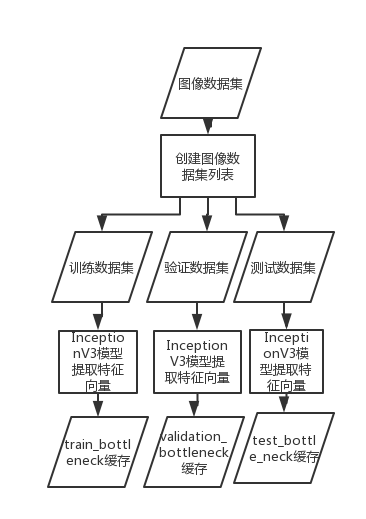


图3 瓶颈层数据的获取

通过create\_image\_lists方法创建数据集列表，将已有的数据集按照参数设定的比例，随机分为训练集、测试集和验证集合。并通过get\_image\_path()方法来获取对应的某一张图像的文件绝对路径。随后加载已经训练好的InceptionV3模型，对训练数据集和进行特征向量提取并缓存。其中train\_bottle\_neck缓存将作为图2中瓶颈层输入，validate\_bottle\_neck作为标准答案输入，test\_bottle\_neck作为最后计算准确率的输入

3.3 全联接层的定义。

根据图2，利用Tensorflow框架可以很方便地定义整个神经网络。

首先定义全联接层的两个输入，分别为输入数据集和标准答案。定义一个输入数据集，命名为bottoleneck\_input，顾名思义为将瓶颈层输出的特征向量作为输入。定义标准答案用于反向传递和计算交叉熵，命名为groud\_truth\_input。由上文可知，随随这两个输入的实质是特征向量，即浮点数组。代码表示为：

group\_truth\_input=tf.placeholder(tf.float32,[None,class\_count,],name=“GroundTruthInput”)

全连接层的定义需要设定一系列的参数。其初始的weights(权重分布)由tf.trucated\_normal(shape,mean,stddev)输出一个正态分布的矩阵产生，大小（shape）由瓶颈层输出的矩阵大小(bottleneck\_tensor\_size)决定，均值(mean)默认为0，标准差(stddev)为0.1。bias(偏置)默认为一个长度为训练类目个数的全零向量。

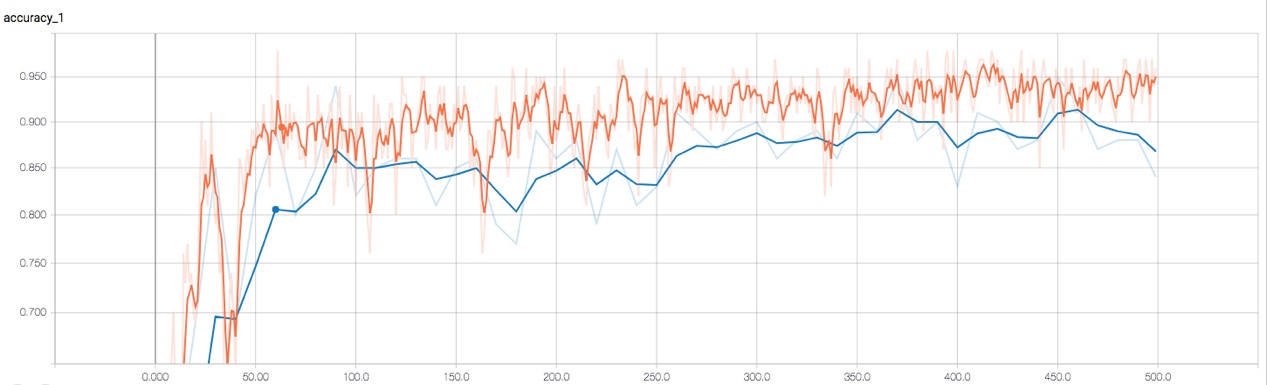
另外，需要定义交叉熵损失函数，命名为cross\_reduce\_mean。交叉熵损失函数用于描述预测的概率分布和正确概率分布的熵，用于反向传播。神经网络的通过递归，最终最小化损失函数来达到收敛。每次递归的优化策略，我们采用梯度下降优化器(GradientDescentOptimizer)，学习率默认为0.01。

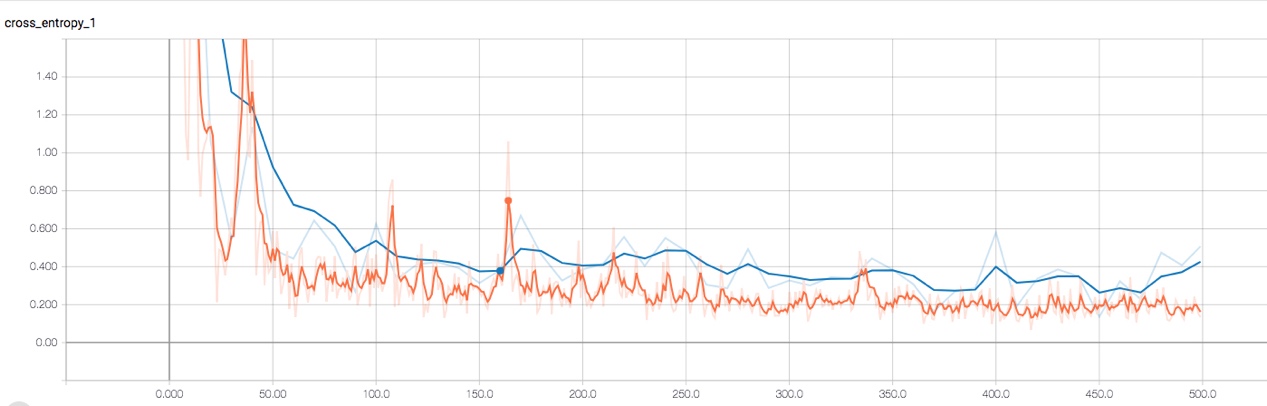
接着，定义准确率，命名为accuracy。通过比较最终输出和验证数据集相比较是否正确，得到准确率，用于描述模型的识别能力。

反向传递的最后需要定义优化算法。在训练一个简单的全联接层时，一般用梯度下降优化器，以收敛交叉熵为目标进行来进行模型数据优化，学习率默认设置为0.01。

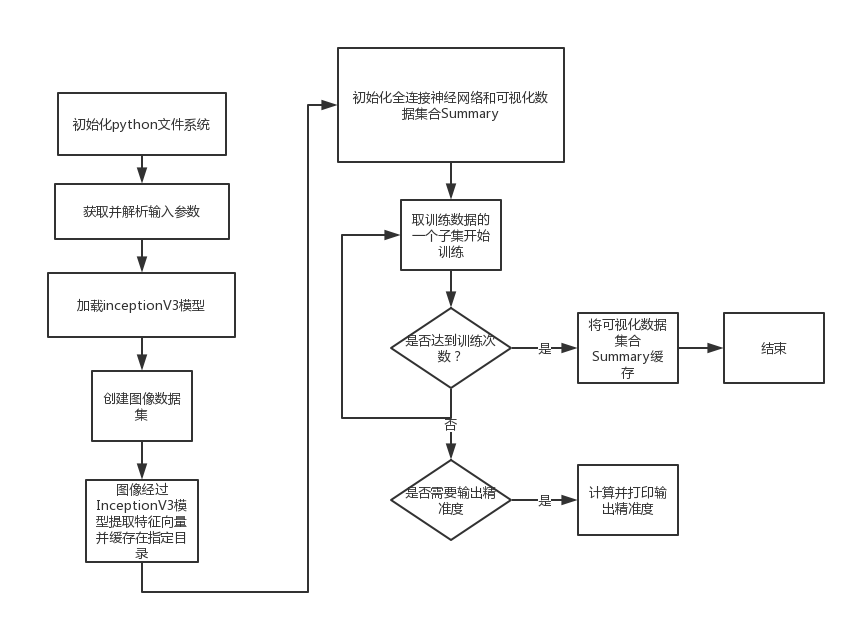
如果需要在TensorBoad（Tensorflow的训练可视化工具）当中可视化训练过程，还需要将这些定义的数据节点全都关联到Tensorflow的summary数据集中，summary数据集会自动缓存节点数据。

(5) 全联接层的训练。每次训练，输入数据集为训练数据集随机获取的一个子集（batch）。每次训练都将数据集数据填入用于可视化的Summary数据集，每100次训练，输出一次准确率，通过实验发现，1000次左右的训练，就可以将识别准确率提升到92%左右。





最终，通过tf.savemodel在指定的文件目录下，保存整个神经网络模型的节点数据文件（pb文件）和分类标签（label）文件。



4、分类模型在Android端口的移植

在第三节当中训练的模型，已经可以在PC端进行图像分类了。但是，如果需要将该模型运行在Android端，需要解决两个问题。

1、Tensorflow框架在移动端的运行。

2、分类模型针对移动端Tensorflow的优化。

接下来，将对这两个问题进行详细阐述。

4.1 在移动端运行Tensorflow

TensorflowLite通过Android8.1发布的神经网络相关API，可以实现在不同架构设备上运行相同的模型。从而设备的处理器架构不再是一个掣肘开发的障碍。

通过在Android编译配置文件build.gradle当中添加TensorflowLite的Android Archive（即AAR，Android系统中用于取代传统Java第三方库Jar格式压缩包的数据包格式），就可以通过谷歌Jcenter的Maven仓库引入TensorflowLite的Android版，进而可以在Android工程中被调用。

4.2 分类模型在移动端的优化

移动端在GPU带宽以及算力，CPU的指令集完备度上和PC端相比有明显的缺陷，所以，必须对分类模型进行必要的精简。由于PC端使用Tensorflow框架而移动端为TensorflowLite框架，所以需要将普通Tensorflow计算图转换成TensorflowLite计算图。

4.2.1 模型的精简

对于模型的精简，主要通过两个方面：1、去除在移动端无效的节点操作。谷歌在Tensorflow项目中提供了tf\_op\_files.txt文件，其中中枚举了所有支持的操作。2、去除在PC端为了精确性而进行的数据冗余，比如过大的神经网络深度depth，每个卷基层的channel数以及每个卷积的kernel宽度。

通过Tensorflow中的optimize\_for\_inference脚本可以去除掉模型当中在移动端不必要的节点，并且去除上文提及的冗余。

4.2.2 模型的转换

对于模型的转换，谷歌同样提供了解决方案。

与TensorflowLite同时发布的还有TensorflowLiteConverter模块(简称TOCO)。该模块用于解决普通的Tensorflow计算图向TensorflowLite计算图的转换。

TOCO目前支持三种形式的普通计算图，分别为使用Tensorflow的Saver类保存的SavedModel，用frozen\_gragh.py脚本保存的freeze\_graph以及tf.Keras模型文件。

通过该模块，可以将Tensorflow模型转换压缩成TensorflowLite模型。TensorflowLite模型为一个\*.pb文件和一个名为label的标签文件。将其加入Android工程的资源文件包Assets当中，就可以被加载在Android工程中的TensorflowLite框架使用。