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。随着分类算法的日趋复杂，对硬件的要求也越来越高，随之而来的是硬件设备的体积越来越庞大。这就催生了一个需求，利用性能相对羸弱的移动端设备，凭借比较小的算力，达到和大型设备类似的图像分类效果。而2015年谷歌开源的Tensorflow深度框架，让各种深度学习网络运行在移动设备上得以实现。而对Tensorflow框架的具体实践也逐渐火热了起来。本文主要介绍，如何在android设备上运行Tensorflow框架，并基于框架，利用经典的用于图像处理的卷积神经网络inceptionV3进行Android端的相册图像识别。具体包括：

1. 需求背景以及Tensorflow框架简介
2. InceptionV3模型的介绍
3. InceptionV3的迁移训练
4. 训练后的在移动端的移植。
5. Android端对数据库和AAR库的调用。
6. 后续改进的方案和设想

本文的App是对 Tensorflow框架在移动端上的具体应用的一次实践，为成熟项目的开发提供了借鉴和经验。

近年来，基于深度学习的各类预训练模型，对许多传统领域，诸如计算机视觉，语音识别技术产生了极大的促进作用。其中首当其冲的便是计算机视觉。而计算机视觉领域，图像识别和分类作为基础技术得到了最广泛的关注，发展也日新月异。

从识别的对象上看：发展经历了从文字识别，数字图像识别，发展到如今的自然物体识别的过程。从发展的指标上看：识别的准确率不断上升，受限的条件逐渐被突破，图像的复杂度也不断加强。从识别的技术上看：传统的SVM已经非常成熟，而基于数据驱动的机器学习更是打开了人工智能的大门。

深度学习的训练方法，相对于传统的前向反馈网络，需要大量的训练层数以及迭代次数，对于算力的要求可以说是空前的。基于此，通常需要应用深度学习方法的模型会部署在大型服务器上，借助其强大的算力迅速得到结果。而作为大前端，包括web页和移动端，只扮演了一个数据传递和结果展示的角色，并没有真正参与到运算当中。但是，移动端对于实时性要求非常高，需要通过网络才能完成其功能的方式，让应用出现了限制和使用上的缺陷。所以，将深度学习框架移植到移动端，并且借助其比较小的算力，依然能够较快较好地完成相应运算的需求就应运而生了。

1、Tensorflow介绍

Tensorflow是谷歌大脑团队在第一代深度学习系统DistBelief的基础上，改进而来的通用计算框架。DistBelief本身在谷歌内部已经获得了巨大的成功，基于该框架的图像分类系统ImageNet，用Inception模型获得了ImageNet2014图像识别比赛的第一名。本文将使用Inception模型的改进版本InceptionV3，其介绍将在下一节中阐述。Tensorflow在此基础上，进一步提升了系统的通用性以及计算速度，他可以在更多的平台上更快更稳定地进行模型计算。

Tensorflow是基于数据流图（graph）的计算系统。数据流图是一个抽象概念。在节点（session）上进行数据运算，用入边和出边来表示数据的输入和输出。整个Tensorflow的计算过程，是数据从一段流到另一段的过程。

Tensorflow可以很方便地支持大部分神经网络，包括CNN，RNN等。

2、InceptionV3模型简介

2012年，AlexNet作出历史性突破之后，主流的网络结构都是在让神经网络的深度更深，神经元数更多来让宽度更大，从而提升神经网络的能力。这样单纯地增大网络，带来了一系列的缺点。

1. 参数太多，在训练集中数据量不够多的情况下，容易过拟合。
2. 网络越大，计算复杂度越高，对算力要求过高，从而实际应用非常困难。
3. 网络深度过深，容易出现梯度弥散，难以优化模型。

2.1 incetion的卷积堆叠

Inception的一个主要的特性就是对卷积的堆叠。传统的卷积神经网络需

传统的卷积神经网络需要在1、3或者5这样的卷积边长当中选取一个合适的大小。而Inception给出的解决方案是，同时使用1、3、5边长的卷积滤波器，并把结果堆叠在一起，一方面增加了网络的宽度，另一方面增加了网络对尺度的适应性。这也是IncetionV1的主要特点。

2.2 Batch-N算法的加入

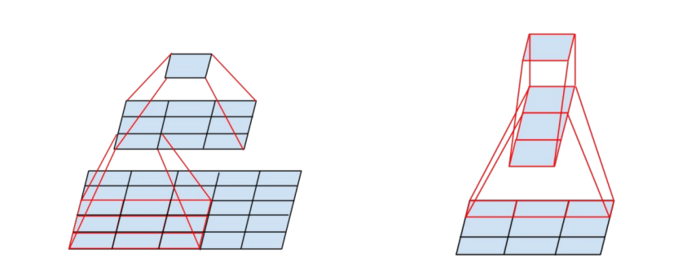
在一般的神经网网络中，除了输入层之外，其他神经网络层在训练时发生的参数变化，会引起后续神经网络数据分布的变化，并且随着层数的加深，变化会越来越剧烈。这种参数的剧烈变化，称之为Internal Convariate Shift。

Internal Convariate Shift会导致一个现象，深层的神经网络学习快，参数变化快，而浅层的神经网络几乎不学习，参数变化小，产生梯度消失现象。最终的结果结果就是最后几层神经网络几乎可以代表整个神经网络，而神经网络的深度就失去了意义。

BN算法的的目的是降低Internal Convariate Shift。它的主要原理在于网络的每一个输入之前，插入一个可学习，有参数的归一化层。在不影响前一个网络特征分布的同时，预处理（归一化）下一个网络的输入，从而降低上一层参数变化对一下层的影响，进而降低nternal Convariate Shift。

2.3 inception的降维

Inception模型的后续发展的精髓在于“降维”，用多个低维度的卷积块，来替换高维度的卷积块。

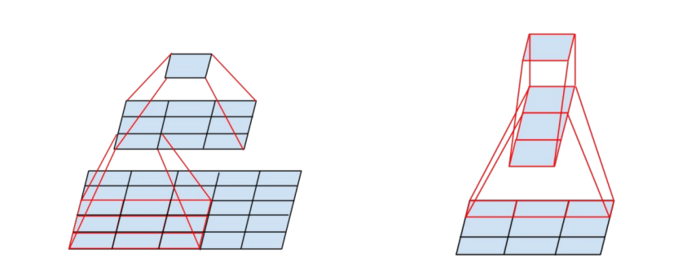


2.3.1分解大卷积核

大尺寸的滤波器的卷积（如5\*5，7\*7）引入的计算量非常大。一个5\*5卷积滤波器比一个3\*3卷积滤波器的计算量多25/9=2.78倍。大维度的卷积滤波器可以获得更多的信息，但可以通过多层的低纬度滤波器来代替这个大维度滤波器。5\*5的卷积，可以看作用一个3\*3的卷积和他的全连接层（相当于两个3\*3的卷积）来代替。这种方式可以理解成在卷积核大小上的因式分解。

2.3.2将大卷积核转化为非对称卷积核

在卷积神经网络当中，一般选用1\*1，3\*3，5\*5甚至7\*7这样的对称的卷积核。当使用较大的对称卷积核时，计算量就会激增。而Inception结构利用n\*1、1\*n的卷积核，进行串联来代替n\*n的对称卷积核。两种卷积的结果是等价的，但是计算量却完全不同。n\*n的卷积核心相当于计算了n\*n次乘法，而非对称网络，将计算次数降低到了2\*n次。当n越大时，计算量的差别就越大。

但是这一方法并不适用于所有的图像特征提取，当图像的为一个中等大小（特征图大小在12到20之间）时比较好

Inception在ImageNet挑战赛的成绩是，错误率低达5.1%。而本文当中将使用InceptionV3模型，利用它来对照片进行分类。

3、模型的迁移训练

3.1 迁移训练的简介

Inception图像分类的数据集有120万被标注的图片。如此巨大的数据大小，才能将神经网络训练到理想的准确率。而对于一般的研究人员来说，收集如此庞大的数据集并且标注其内容，其工作量非常庞大，几乎是不可能的。另外，用如此庞大的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。为了解决数据标注和训练时间过长的问题，迁移学习便应运而生。

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。根据《A Deep Convolutional Activation Feature for Generic Visual Recognition》的结论，在保留已经训练好的Inception-V3的卷基层的基础上，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。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，仅仅通过一个单层的全连接网络就可以区分1000个类别的图像。可以认为，InceptionV3模型已经可以对输入数据提取出足够的特征向量。所以，可以直接利用InceptionV3模型的特征向量作为输入，训练一个符合我们需求的全联接层。

3.2 迁移训练的关键步骤

迁移训练的实质，是通过前向传递训练一层用于分类的全联接层。是一个经典的基础分类问题。其关键步骤有：

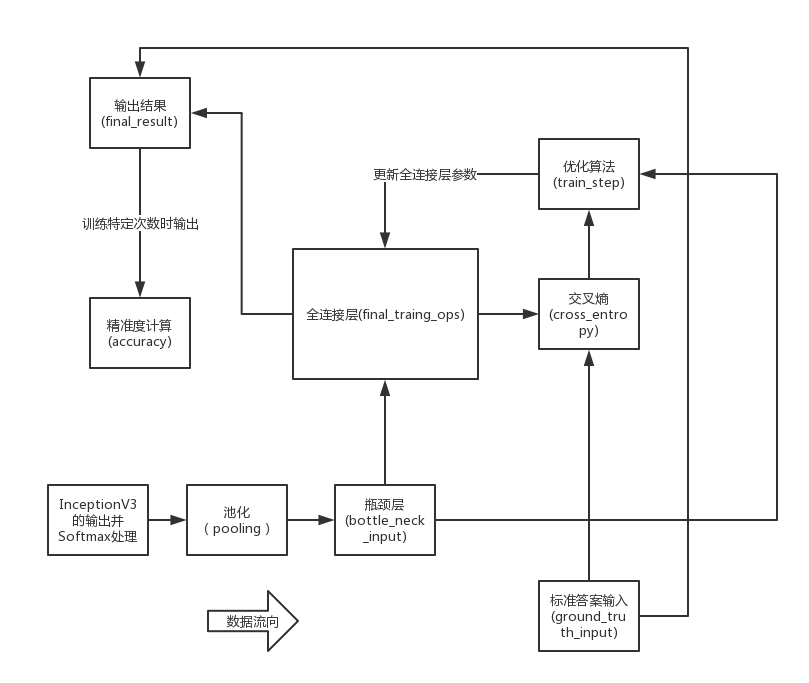
1、图像预处理和数据集获取

2、获取瓶颈层输出并缓存

3、定义全联接层

4、训练全联接层

5、模型性能数据的输出



接下来将对这几个模块进行详细阐述。

3.2.1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。

ImageNet项目是一个用于视觉对象识别研究的大型可视化数据库。基于WordNet。在ImageNet中有超过1400万图片和大约20000个名词同义词集进行了关联。ImageNet的图片都是从互联网上爬取下来，并进行人工标注的。本文在一部分数据集上，也进行了网络爬取，并进行了标注。

网络爬取的过程可以通过python3.6中的site-package包中的SpiderImagePlugin.py的python脚本进行。由于该脚本的拓展性不是很好，本文当中对网络爬取的进行封装。

3.1.1在config.py文件对需要进行爬取的图像目录的中英文进行了映射，对url当中的常见的字符转换进行了映射处理。

3.1.2在spider.py中，对所需要爬取的图像进行了爬取。本文的图像爬取工作主要通过Baidu的搜图功能。脚本的主要步骤如下：

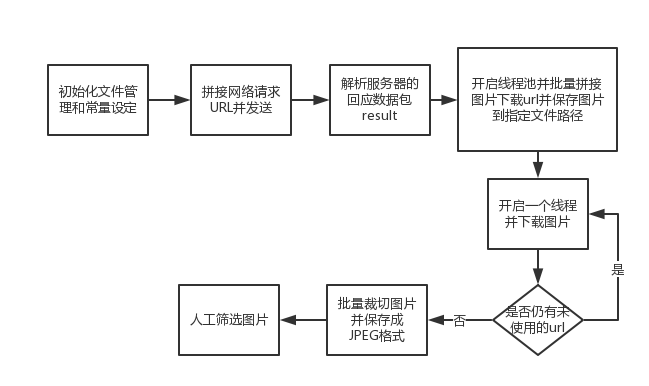
1、根据config.py文件对url进行拼装并向baidu的搜图服务器进行请求

2、解析Baidu服务器的请求回应的Json格式的result数据包，得到各个图像对应的请求连接（url），并根据图像类目缓存在字典（dict）中。

3、根据缓存的url，批量对对应的图像进行网络请求。并将下载下来的图像分别存储在他相应的目录。

4、在图像爬取完成后，用deal\_pic.py脚本对所有图像进行简单的预处理。包括格式转换，统一转换为JPEG格式；图像压缩，每张图像不超过300k；图像裁切，在没有裁切掉图像主体的前提下，裁切为299\*299像素。

5、人工检查图像是否有问题。剔除主体不明确，分类明显错误的图像。



3.2.2获取瓶颈层的输出并缓存

在训练图像获得之后，我们需要对这些图像经过InceptionV3模型对特征向量的提取来作为全联接层的输出。其主要步骤为：

(1)、创建图像数据集列表。在python脚本中通过create\_image\_lists方法进行。总共有3个参数，保存所有训练数据的文件路径，测试集比列和验证集比例。这个方法的主要作用，是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。

(2)、加载已经处理好的InceptionV3模型。InceptionV3作为谷歌已经训练好的数据模型，其神经网络节点可以通过\*.pb文件进行持久化。Tensorflow可以通过加载pb文件，来复用已经训练好的神经网络。在create\_model\_info()方法中，将解析传入的神经网络名称，在进行合法性检查后，下载相应的模型\*.pb文件到指定目录，最终输出模型的相关参数。

(3)、命中缓存。由于图像识别神经网络的训练会频繁的读取图片，为了训练的效率，必须采取缓存的方式，将图像和图像处理后的中间产物（本文中主要是瓶颈层输出的）保存在内存中。每次读取该文件，优先从缓存中读取，如果缓存没有命中，那么重新获取文件并缓存。重新获取文件的方式：原始图像通过寻找绝对路径从磁盘当中获取，瓶颈层的输出通过原始文件通过InceptionV3模型获取特征向量。

(4)全联接层的定义。这个步骤的主要工作是定义一个全联接层以及他的训练策略。

首先要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。

接下来要定义一个全联接层，命名为final\_training\_ops，并设置其初始参数。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。

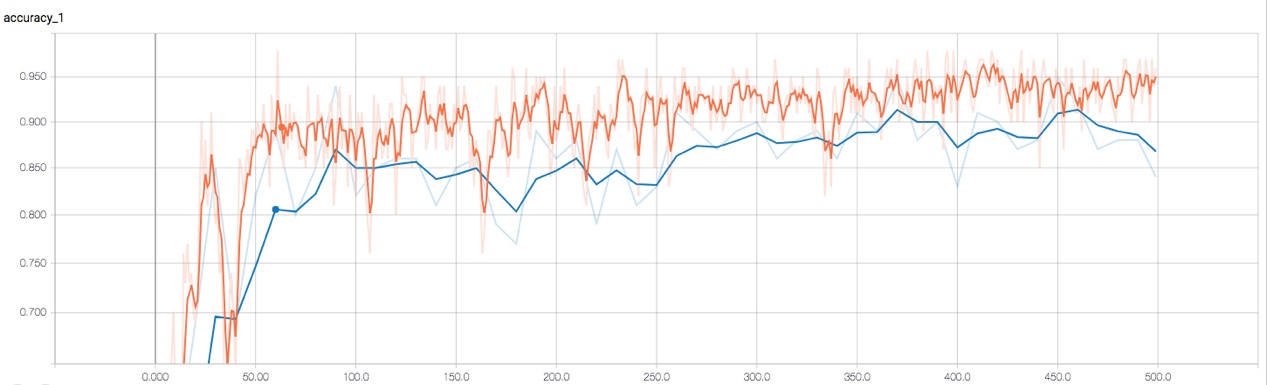
另外，需要定义交叉熵损失函数，命名为cross\_reduce\_mean。交叉熵损失函数用于描述预测的概率分布和正确概率分布的熵，用于反向传播。神经网络的通过递归，最终最小化损失函数来达到收敛。每次递归的优化策略，我们采用梯度下降优化器(GradientDescentOptimizer)，学习率默认为0.01。

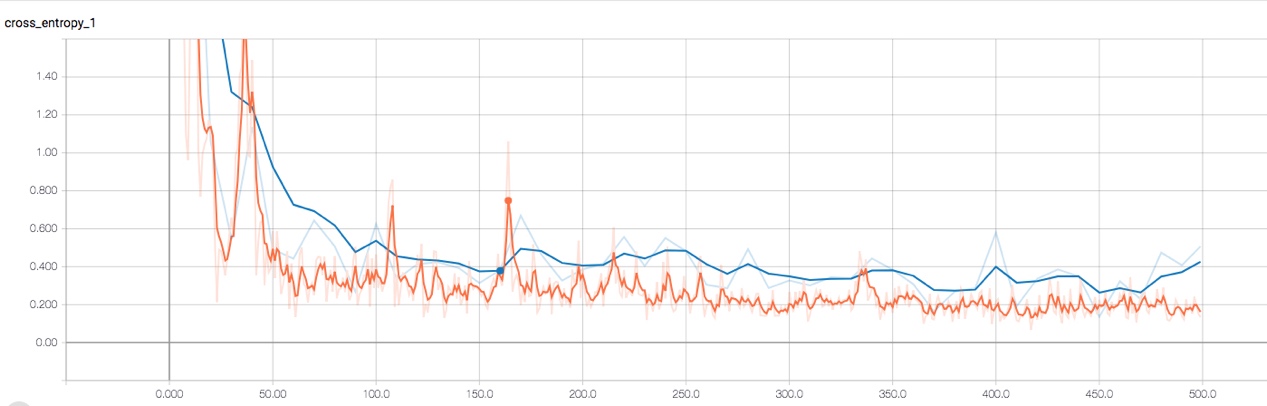
接着，定义准确率，命名为accuracy。通过比较最终输出和验证数据集相比较是否正确，得到准确率，用于描述模型的识别能力。

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。

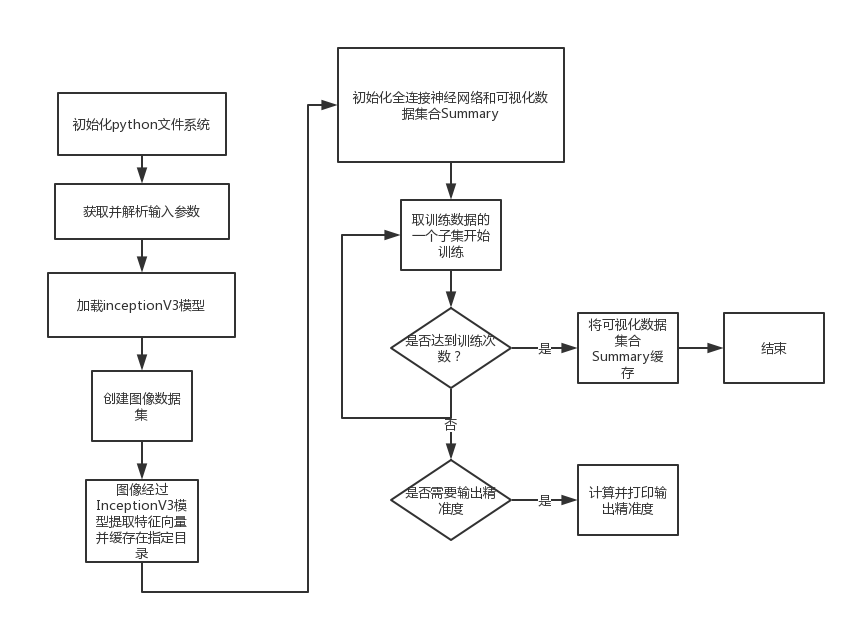
如果需要在TensorBoad（Tensorflow的训练可视化工具）当中可视化训练过程，还需要将这些定义的数据节点全都关联到Tensorflow的summary数据集中，summary数据集会自动缓存节点数据。

(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

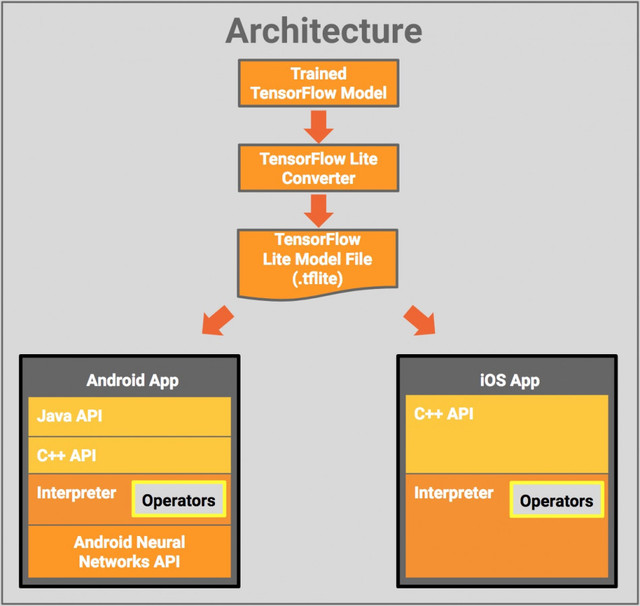
Android端作为最普及的移动端平台，在海量的设备上装载。Android平台的处理器架构通常为ARM。但如果使用PC端来运行Android模拟器，因为PC大多采用Intel公司的芯片，所以通常为X86架构，会因为不同架构对指令集的支持问题，而发生效率乃至无法运行的问题。

在Tensorflow的早期版本(0.10)，针对不同的架构，需要编译相应的动态链接库。利用Tensorflow文档中名为build\_all\_iso.sh的Shell脚本，通过配置指定的参数，生成对应架构的Tensorflow动态链接库，在Android端的动态链接库是一个底层为C语言，可以为Java调用的Jni库。最终，通过在Android编译器中引用该Jni库来完成调用。

在2017年，TensorflowLite推出之后，在移动端运行Tensorflow变得非常简便。TensorflowLite是谷歌在先前针对移动端和嵌入式设备的TensorflowMobile框架的后续版本，通过该框架，应用开发者可以在移动设备上轻易部署人工智能。

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。

5、Android端对TensorFlow接口以及分类模型的调用

由于移动端的限制，很多准备工作需要在PC端完成，而不应该在移动端进行。通常来说，移动端的Tensorflow只需要做以下这些工作：

1、从\*.pb文件中加载计算图与权重值。

2、利用计算图创建相应会话(Session)。

3、创建输入张量（Tensor）。

4、通过会话运行计算图。

5、输出归一化后的结果矢量V。

本文的移动端图像分类软件，主要包含这些模块：

1、系统相册操作模块。

2、数据库模块

3、视图组件模块

4、逻辑处理模块

5、Tensorflow操作模块

6、工具类

7、代码静态检查以及单元测试模块。

他们的

//TODO 移动端的这段文字还需要重新写一写，因为软件目前还存在一些BUG。