

移动应用编程 2018-课程项目

食物识别 APP

计科 152 班

1507300311 郭芷柔

2018/6/25

目录

一、课程项目目标.....	2
二、总体设计.....	2
三、技术选项.....	2
四、实验基本原理.....	2
五、技术实现方案.....	3
六、实验步骤.....	5
七、核心代码.....	7
八、实验心得和体会.....	10

一、课程项目目标

利用课程所学的安卓开发基本工具/方法等基础知识，结合深度学习技术，做一款有“AI”的应用，可以是娱乐应用，可以是游戏更可以是解决痛点的产品。

二、总体设计

设计一款基于深度学习 tensorflow 的 APP，利用谷歌现有的 Google Inception-V3 模型进行迁移训练模型，对面条、雪碧、饼干、蛋糕等十种食物的样本数据进行训练，来完成一个可以识别十种食物的模型，并将新训练的模型迁移到 Android 端平台，通过使用拍照和调用相机的功能，识别这十种食物，并给出相应的卡路里估计和健康饮食指南。

三、技术选项

- 1 深度学习框架选择 Tensorflow
- 2 使用安卓开发工具 Android Studio

四、实验基本原理

从代码上看，整个深度网络的结构体系是这样子的：从输入端开始，将自己的训练集中的每张图像输入网络，先有 3 个卷积层，然后是 1 个 pool 层。然后又是 2 个卷积层，一个 pool 层。会生成一定维度的特征向量，将这个特征保存在一个 txt 文件中，再用这个特征来训练 softmax 分类器。

Inception 模块：

Inception 结构将不同的卷积层通过并联的方式结合在一起。同时使用所有不同尺寸的过滤器，然后再将得到的矩阵拼接起来。不同的矩阵代表了 Inception 模型中的一条计算路径。虽然过滤器的大小不同，但如果所有的过滤器都使用全 0 填充且步长为 1，那么前向传播得到的结果矩阵的长和宽都与输入矩阵一致。这样经过不同过滤器处理的结果矩阵可以拼接成一个更深的矩阵。

Inception-v3 模型：

真正在 Inception-v3 模型中使用的 Inception 模块要更加复杂且多样：Inception-v3 模型总共有 46 层，由 11 个 Inception 模块组成，一共 96 个卷积层。为了更好地实现类似 Inception-v3 模型这样的复杂卷积神经网络，google 训练脚本中使用 Tensorflow-Slim 工具来更加简洁地实现一个卷积层。

TF-Slim：

slim 的 conv2d 构造的是一个激活函数为 Relu 的卷积神经网络。

MaxPool:

Pool 是一个将卷积参数进行减少的过程，主要进行特征图的压缩。

Dropout 层:

这个层的作用是随机除去一些神经元，使得整个模型不至于过拟合。

FullConnect:

全连接层，在整个过程的最后，才使用全连接，训练出权重。

Softmax:

这个神经网络的最后是 softmax 层。softmax 层也就是分类专用的层，使用一个概率来表示待分类对象有多大概率属于某个类。

五、技术实现方案

Github 连接: https://github.com/shiwena/guozhirou_classify

1、 实现图片识别的功能

- 在 Android 系统中执行 TensorFlow Inference 操作，需要调用 libandroid_tensorflow_inference_java.jar 中的 JNI 接口，构建 TensorFlow Inference 对象，构建该对象时候会加载 TensorFlow 动态链接库 libtensorflow_inference.so 到系统中；参数 assetManager 为 android asset 管理器；参数 modelFilename 为 TensorFlow 模型文件在 android_asset 中的路径：

```
TensorFlowInferenceInterface inferenceInterface = new  
TensorFlowInferenceInterface(assetManager, modelFilename);
```

- 向 TensorFlow 图中加载输入数据，本 App 中输入数据为摄像头截取到的图片；参数 inputName 为 TensorFlow Inference 中的输入数据 Tensor 的名称；参数 floatValues 为输入图片的像素数据，进行预处理：

```
inferenceInterface.feed(inputName, floatValues, 1, inputSize, inputSize, 3);
```

- 执行模型推理；outputNames 为 TensorFlow Inference 模型中要运算 Tensor 的名称，在本 APP 中为分类的 label 值：

```
inferenceInterface.run(outputNames);
```

- 获取模型 Inference 的运算结果，其中 outputName 为 Tensor 名称，参数 outputs 存储 Tensor 的运算结果。本 APP 中，outputs 为计算得到的相应的 label 值：

```
inferenceInterface.fetch(outputName, outputs);
```

- 调用 classifier 类的 recognizeImage 方法，将识别的结果保存在 result 队列里，/保存识别可能性最高的结果，将识别结果队列的第 1 个字符，与标签 label 对应。

```
Final List<Classifier.Recognition> results = classifier.recognizeImage(croppedBitmap);
```

```
final String mm=results.get(0).toString();
final char mm1=mm.charAt(1);
```

2、 实现相机的调用

拍照和从图库选择图片都先创建了一个 File 对象，用于存储摄像头拍下的图片 File
 outputImage = new
 File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES),
 mName + ".jpg");

然后将它放在手机的根目录下，然后调用 Uri 的 fromFile()方法将 File 对象转换成 Uri 对象。
 这个 Uri 对象标识着图片的唯一地址

```
private Uri mImageUri;    mImageUri = Uri.fromFile(outputImage);
```

接着构建出一个 Intent 对象，并将这个 Intent 的 action 指定为：MediaStore.ACTION_PICK，
 再调用 Intent 的 putExtra () 方法指定图片的输出地址，这里就填入刚刚得到的 Uri 对象，
 最后调用 startActivityForResult () 来启动活动

用摄像头拍照成功后会回调 onActivityResult () 方法，这时候会继续构建 Intent 对象，把它
 的 action 指定为：“com.android.camera.action.CROP”，并再调用 startActivityForResult ()
 来启动裁剪程序。裁剪后的图片同样会输出到手机根目录下的图片文件中。

```
Intent intent = new  

  Intent("com.android.camera.action.CROP");  

      intent.setDataAndType(mImageUri, "image/*");  

      intent.putExtra("scale", true);  

      intent.putExtra(MediaStore.EXTRA_OUTPUT, mImageUri);  

      startActivityForResult(intent, CROP_PHOTO);
```

3、 实现相册的调用

当实现的是打开相册的功能，那么与拍照时基本的操作没有什么太多的差别。
 都是先创建一个 File 对象，保存从图库选择的文件。然后构建出一个 Intent 对象，并将它的
 action 指定为：“android.intent.action.GET_CONTENT”：接着给这个 Intent 对象设置一些必要
 的参数，包括是否允许缩放和裁剪、图片的输出位置等。最后调用 startActivityForResult ()
 方法，就可以打开相册程序选择照片了。

```
fromI Intent intent = new Intent("com.android.camera.action.CROP");  

      intent.setDataAndType(fromImageUri, "image/*");  

      intent.putExtra("scale", true);  

      intent.putExtra(MediaStore.EXTRA_OUTPUT, mImageUri);  

      startActivityForResult(intent, CROP_PHOTO); mImageUri = data.getData();
```

裁剪操作完成后，程序又会回调到 onActivityResult 方法中，这个时候就可以利用
 BitmapFactory 的 decodeStream () 方法将存储在手机根目录下的图片文件解析成 Bitmap 对
 象，然后把它设置到 ImageView 控件上显示出来。

六、实验步骤

0、准备步骤：生成在 Android 平台上调用 tensorflow 模型需要的 jar 包和 so 文件：

在需要调用模型的.java 文件中，导入 jar 包：`import org.tensorflow.contrib.android.TensorFlowInferenceInterface`

在该 java 类定义的首行，导入 so 文件，载入动态链接库：`System.loadLibrary("tensorflow_inference")` 创建接口，实现调用

1、开始训练

新建训练样本文件夹，里面存放十种食物的照片，分别是：饺子、热狗、面条、蛋糕、雪碧、饼干、粥、巧克力、炒饭和冰激凌，每种食物我下载了至少两百张照片作为训练样本。





名称	修改日期	类型	大小
chocolate	2018/6/23 1:12	文件夹	
cookie	2018/6/23 1:08	文件夹	
cup_cake	2018/6/23 1:04	文件夹	
dumplings	2018/6/22 17:47	文件夹	
fried_rice	2018/6/22 20:22	文件夹	
hot_dog	2018/6/22 20:39	文件夹	
ice_cream	2018/6/22 20:49	文件夹	
noodle	2018/6/23 0:54	文件夹	
porridge	2018/6/23 0:55	文件夹	
xue_bi	2018/6/23 0:17	文件夹	

下载 Google 提供的迁移训练脚本 `retrain.py`，在训练文件夹下启动 tensorflow，运行该脚本。在运行 `retrain.py` 脚本时，需要配置一些运行命令参数，指定模型输入输出相关名称和其他训练要求的配置：

```
python retrain.py \
--bottleneck_dir=bottlenecks \
--how_many_training_steps=500 \
--model_dir=inception \
--summaries_dir=training_summaries/basic \
--output_graph=retrained_graph.pb \
--output_labels=retrained_labels.txt \
--image_dir=flower_photos
```

```
C:\Windows\System32\cmd.exe
2018-06-24 10:42:07.265091: Step 420: Cross entropy = 0.083726
2018-06-24 10:42:07.873672: Step 420: Validation accuracy = 99.0% (N=100)
2018-06-24 10:42:14.613371: Step 430: Train accuracy = 98.0%
2018-06-24 10:42:14.617381: Step 430: Cross entropy = 0.090839
2018-06-24 10:42:15.325702: Step 430: Validation accuracy = 97.0% (N=100)
2018-06-24 10:42:22.177758: Step 440: Train accuracy = 99.0%
2018-06-24 10:42:22.181269: Step 440: Cross entropy = 0.102240
2018-06-24 10:42:23.076717: Step 440: Validation accuracy = 100.0% (N=100)
2018-06-24 10:42:29.759266: Step 450: Train accuracy = 99.0%
2018-06-24 10:42:29.761309: Step 450: Cross entropy = 0.074342
2018-06-24 10:42:30.423032: Step 450: Validation accuracy = 97.0% (N=100)
2018-06-24 10:42:37.204006: Step 460: Train accuracy = 98.0%
2018-06-24 10:42:37.207015: Step 460: Cross entropy = 0.116560
2018-06-24 10:42:37.983073: Step 460: Validation accuracy = 96.0% (N=100)
2018-06-24 10:42:45.154466: Step 470: Train accuracy = 99.0%
2018-06-24 10:42:45.158482: Step 470: Cross entropy = 0.083751
2018-06-24 10:42:45.927195: Step 470: Validation accuracy = 98.0% (N=100)
2018-06-24 10:42:52.446777: Step 480: Train accuracy = 99.0%
2018-06-24 10:42:52.447779: Step 480: Cross entropy = 0.069010
2018-06-24 10:42:53.326130: Step 480: Validation accuracy = 97.0% (N=100)
2018-06-24 10:43:00.071150: Step 490: Train accuracy = 98.0%
2018-06-24 10:43:00.074159: Step 490: Cross entropy = 0.104799
2018-06-24 10:43:00.911593: Step 490: Validation accuracy = 96.0% (N=100)
2018-06-24 10:43:06.993303: Step 499: Train accuracy = 100.0%
2018-06-24 10:43:06.995309: Step 499: Cross entropy = 0.070943
2018-06-24 10:43:07.702187: Step 499: Validation accuracy = 99.0% (N=100)
Final test accuracy = 90.9% (N=242)
Converted 2 variables to const ops.
```

其中设定训练步骤为 500，从样本读入到结束训练共花费八十多分钟，不过具体时间也由电脑性能而定。等到训练完成后，将得到新生成的 `retrained_labels.txt` 和 `retrained_graph.pb` 这两个模型相关文件：

	optimize_for_inference	2018/6/18 10:19	PY 文件	5 KB
	retrain	2018/6/21 21:20	PY 文件	44 KB
	retrained_graph.pb	2018/6/24 10:43	PB 文件	85,393 KB
	retrained_labels	2018/6/24 10:43	文本文档	1 KB






上图retrained_for_inference.pb是完成迁移训练后的新模型文件,新生成的pb文件很大。考虑到要将这个模型移植到 Android 端去加载,这不仅会对应用的运行内存造成巨大压力,而且会导致安装包增大很多,因此,要考虑对模型文件进行优化,压缩它的体积。

2、优化模型文件:

调用 optimize_for_inference.py 脚本进行优化,调用脚本时,同样设置几个命令参数,输入的 PB 文件路径,输出的 PB 文件路径,输入节点名以及输出节点名等:

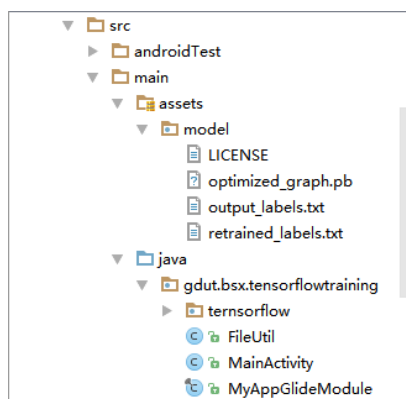
```
python optimize_for_inference.py \
--input=retrained_graph.pb \
--output=optimized_graph.pb \
--input_names="Cast" \
--output_names="final_result"
```

经过 potimize_for_inference 优化后的模型相对前者小了一点,将它放入手机端导入工程中。

	optimize_for_inference	2018/6/18 10:19	PY 文件	5 KB
	optimized_graph.pb	2018/6/24 11:15	PB 文件	85,119 KB
	retrain	2018/6/21 21:20	PY 文件	44 KB
	retrained_graph.pb	2018/6/24 10:43	PB 文件	85,393 KB
	retrained_labels	2018/6/24 10:43	文本文档	1 KB

3、将新训的 tensorflow 模型移植到 Android 中:

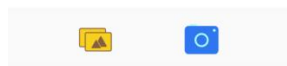
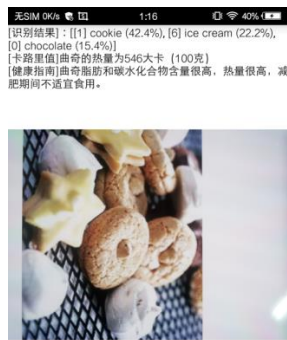
把把新训练的 pb 文件和 labels 文件复制到 assets 文件夹下,并直接使用 Google demo 项目中提供的 Classifier.java 和 TensorFlowImageClassifier.java 这两个类来实现。classifier 类中的 recognizeImage 方法将图片识别,返回一个结果队列。



4.结果判断

采用“照相”和“相册”的功能导入图片,生成 bitmap 传入模型进行识别,将返回的 list 从高到低排序,并选取可能性最高的结果取出,与我们之前生成的 label 对比,结果显示在界面中,同时根据识别的结果类别打印对应的卡路里和信息提示。

例如在饼干识别的过程中,可以得到 app 识别该图像为饼干自信程度为 42.4%,为冰激凌自信程度为 22.2%,巧克力自信程度为 15.4% 此时,判定该图片为饼干。同时,APP 给出饼干曲奇的卡路里值 以及健康指南。



七、核心代码

MainActivity.java 类中识别图片的核心代码

// 创建 Classifier 对象，使用训练模型 pb 文件，其中 MODEL_FILE 为模型 pb 文件路径，LABEL_FILE 为标签路径，INPUT_SIZE 保存图片处理后的大小，IMAGE_MEAN 保存图片均值，IMAGE_STD 保存标准图片大小值为 128f，INPUT_NAME 和 OUTPUT_NAME 表示输入输出名。

```
classifier = TensorFlowImageClassifier.create(MainActivity.this.getAssets(),MODEL_FILE,
LABEL_FILE, INPUT_SIZE, IMAGE_MEAN, IMAGE_STD, INPUT_NAME, OUTPUT_NAME);
```

```
/**
```

```
* MainActivity.java 中的图片识别匹配方法
```

```
*/
```

```
private void startImageClassifier(final Bitmap bitmap) {
    executor.execute(new Runnable() {
        @Override
```



```

        public void run() {
            try {
                Bitmap croppedBitmap = getScaleBitmap(bitmap, INPUT_SIZE);
                Final List<Classifier.Recognition> results = classifier.recognizeImage(croppedBitmap);
                //调用 classifier 类的 recognizeImage 方法，将识别的结果保存在 result 队
                列里
                final String mm=results.get(0).toString();//保存识别可能性最高的结果
                final char mm1=mm.charAt(1);//识别结果队列的第 1 个字符，与标签
                label 对应，例如 0 表示可乐。
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        String mm2="[识别结果]: "+results;//根据可能性最高的结果
                        给出相应的卡路里值和健康建议
                        if(mm1=='0')mm2+="\n[卡路里值][健康指南];
                        else if(mm1=='1')mm2+="\n[卡路里值][健康指南];
                        else if(mm1=='2')mm2+="\n[卡路里值][健康指南];
                        else if(mm1=='3')mm2+="\n[卡路里值][健康指南]";
                        else if(mm1=='4')mm2+="\n[卡路里值][健康指南]";
                        else if(mm1=='5')mm2+="\n[卡路里值][健康指南]";
                        else if(mm1=='6')mm2+="\n[卡路里值][健康指南]";
                        result.setText(""+mm2);
                    }
                });
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

TensorFlowImageClassifier.java 类中实现 Classifier 接口

/**构建 TensorFlow Inference 对象，构建该对象时候会加载 TensorFlow 动态链接库 libtensorflow_inference.so 到系统中；

参数 assetManager 为 android asset 管理器；参数 modelFilename 为 TensorFlow 模型文件在 android_asset 中的路径。

```

    */
    c.inferenceInterface = new TensorFlowInferenceInterface(assetManager, modelFilename);

    public List<Recognition> recognizeImage(final Bitmap bitmap) {
        Trace.beginSection("recognizeImage");
        Trace.beginSection("preprocessBitmap");
        // 将图片文件构建成输入数组，输入数组是一维 float 数组，所以在 Tensorflow 中
        的特征要转变为一维
    }

```

```

        bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0, bitmap.getWidth(),
        bitmap.getHeight());
        for (int i = 0; i < intValues.length; ++i) {
            final int val = intValues[i];
            floatValues[i * 3 + 0] = (((val >> 16) & 0xFF) - imageMean) / imageStd;
            floatValues[i * 3 + 1] = (((val >> 8) & 0xFF) - imageMean) / imageStd;
            floatValues[i * 3 + 2] = ((val & 0xFF) - imageMean) / imageStd;
        }
        Trace.endSection();
        Trace.beginSection("feed");
        //将模型输入放入 InferenceInterface
        /**向 TensorFlow 图中加载输入数据，本 App 中输入数据为摄像头截取到的图片；
        参数 inputName 为 TensorFlow Inference 中的输入数据 Tensor 的名称；
        参数 floatValues 为输入图片的像素数据，进行预处理后的浮点值；
        [1,inputSize,inputSize,3]为裁剪后图片的大小，比如 1 张 224*224*3 的 RGB 图片。
        */
        inferenceInterface.feed(inputName, floatValues, 1, inputSize, inputSize, 3);
        Trace.endSection();
        // Run the inference call.
        //执行模型推理； outputNames 为 TensorFlow Inference 模型中要运算 Tensor 的名
        称，本 APP 中为分类的 Logist 值。
        Trace.beginSection("run");
        inferenceInterface.run(outputNames, logStats);
        Trace.endSection();
        //获取模型 Inference 的运算结果，其中 outputName 为 Tensor 名称，参数 outputs
        存储 Tensor 的运算结果。本 APP 中，outputs 为计算得到的 Logist 浮点数组。
        Trace.beginSection("fetch");
        inferenceInterface.fetch(outputName, outputs);
        Trace.endSection();
        //用 PriorityQueue 获取 top-3,这儿的 Recognition 来自于接口 Classifier,是一个 bean
        类
        PriorityQueue<Recognition> pq =
            new PriorityQueue<Recognition>(3, new Comparator<Recognition>() {
                @Override
                public int compare(Recognition lhs, Recognition rhs) {
                    return Float.compare(rhs.getConfidence(), lhs.getConfidence());
                }
            });
        for (int i = 0; i < outputs.length; ++i) {
            if (outputs[i] > THRESHOLD) {
                pq.add(
                    //构建 bean 类，参数是 label, label name, confidence
                    new Recognition(
                        "" + i, labels.size() > i ? labels.get(i) : "unknown", outputs[i], null));
            }
        }
    }
}

```

```

        }
    }
    final ArrayList<Recognition> recognitions = new ArrayList<Recognition>();
    int recognitionsSize = Math.min(pq.size(), MAX_RESULTS);
    for (int i = 0; i < recognitionsSize; ++i) {
        recognitions.add(pq.poll());
    }
    Trace.endSection();
    return recognitions;//返回识别结果队列
}

```

八、实验心得和体会

这是我第一次尝试用 **tensorflow** 训练一个模型,在一开始我就遇到了很多困难,从 **tensorflow** 的安装到配置,到模型的选取,对于刚接触深度学习的我来说是个不小的难题,最后我根据网络上的教程,渐渐熟悉了一个简单的模型从训练到移植至 **Androidstudio** 的流程,并利用 **Google** 一个简单的开源迁移训练脚本 **retrain.py** 和优化训练脚本 **optimize_for_inference.py** 生成了我的 **pb** 文件,结合网络信息和谷歌开源代码,制作了自己的一个简单的食物识别的 **APP**,虽然准确率和 **APP** 大小仍有待提高,但是对于我来说也算是一个小小的成就了。很开心通过这次课设让我初步接触了深度学习框架 **tensorflow**,尽管在这次课设中我了解的还不够深入,但是对于我以后进行有关深度学习方面的研究算是有了初步的了解,明确了进行深度学习研究的正确方向,激起了我想要继续了解学习的欲望,收获感触颇多。