图像分类

预测图像类别的任务被称为 **图像分类**

比如我们想知道下图出现的是哪种动物

在**训练**中，用**图像**和其对应的 **标签** 投喂一个图像分类模型。每个标签是一个概念或种类的名字。这个模型就要学会去识别这些标签。

给予足够多的训练数据（通常一个标签对应数以百计的图片），这个图像分类模型就能够学习去**预测**新的图片是否属于训练数据中的某些种类。这个预测的过程被称为 **推断** 。

当我们提供一张新的图片给模型时，它会输出这张图片含有这三种动物的概率。以下是一个输出示例：

动物种类	概率
兔子	0.07
仓鼠	0.02
狗	0.91

基于输出，我们能够看到分类模型预测出，这张图片有很大概率表示的是一条狗。

模型的输入与输出

为了执行推断，一张图片被输入进模型中。接着，模型将输出一串代表概率的数组，元素大小介于 0 和 1 之间。结合我们的示例模型，这个过程可能如下所示：

→ [0.07, 0.02, 0.91]

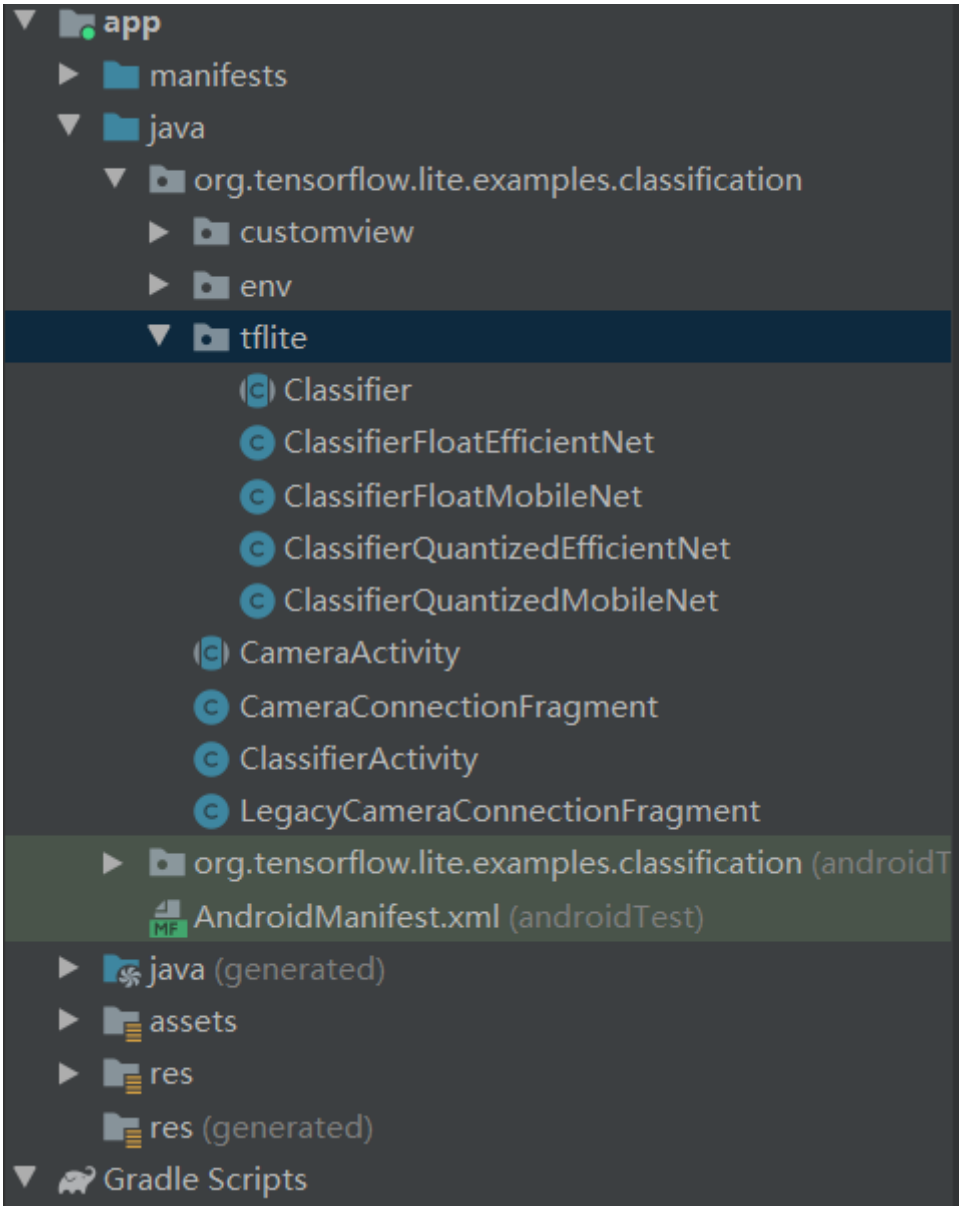
也就是说，我们得知了这个模型：

输入是一张**图片（bitmap）**

输出是一个**概率数组**

分析工程的目录结构（如果是自定义则可以跳过）

工程的目录结构如下：



可知：

- tflite文件夹下文件都是以Classifier开头的，翻译过来就是分类器，主要是模型的加载与初始化
- CameraActivity与CameraConnectionFragement是调用摄像头的逻辑
- ClassifierActivity是**调用模型**的地方，是接受输入，产生输出的地方

输入的流程

由于目录分析中我们了解到ClassifierActivity是接受输入、产生输出的文件。我们从这开始找

可以找到一个核心函数，也就是processImage也就是处理图片这个函数

```
protected void processImage() {
  rgbFrameBitmap.setPixels(getRgbBytes(), offset: 0, previewWidth, x: 0, y: 0, previewWidth, previewHeight);
  final int cropSize = Math.min(previewWidth, previewHeight);

  runInBackground(
      new Runnable() {
        @Override
        public void run() {
          if (classifier != null) {
            final long startTime = SystemClock.uptimeMillis();
            final List<Classifier.Recognition> results =
                classifier.recognizeImage(rgbFrameBitmap, sensorOrientation);
            lastProcessingTimeMs = SystemClock.uptimeMillis() - startTime;
            LOGGER.v("Detect: %s", results);

            runOnUiThread(
                new Runnable() {
                  @Override
                  public void run() {
                    showResultsInBottomSheet(results);
                    showFrameInfo(previewWidth + "x" + previewHeight);
                    showCropInfo(imageSizeX + "x" + imageSizeY);
                    showCameraResolution( cameraInfo: cropSize + "x" + cropSize);
                    showRotationInfo(String.valueOf(sensorOrientation));
                    showInference( inferenceTime: lastProcessingTimeMs + "ms");
                  }
                });
          }
          readyForNextImage();
        }
      });
}
```

这段代码大概在干什么呢?

- 创建了一个classifier对象，调用了classifier对象的recoginizeImage方法，得到了results
- 这个results经过分析，就是我们模型所输出的概率数组!
- 将results里的数据渲染到UI界面上

也就是说，这行代码就能得到调用模型的输出

```
final List<Classifier.Recognition> results =
    classifier.recognizeImage(rgbFrameBitmap, sensorOrientation);
```

我们可以看到，它只需要传入两个参数

rgbFrameBitmap：将相机数据传回来以获得bitmap

sensorOrientation：手机传感器的方向

这两个参数都可以从CameraActivity获得的数据中得到!

调用的流程

输入的两个参数获取的方式并没有很大的问题。那么我们接下来来分析调用

```
classifier.recognizeImage(rgbFrameBitmap, sensorOrientation);
```

不妨去看看Classifier.java文件以下是核心函数

```java
/** Runs inference and returns the classification results. */
public List<Recognition> recognizeImage(final Bitmap bitmap, int sensorOrientation) {
  // Logs this method so that it can be analyzed with systrace.
  Trace.beginSection( sectionName: "recognizeImage");

  Trace.beginSection( sectionName: "loadImage");
  long startTimeForLoadImage = SystemClock.uptimeMillis();
  inputImageBuffer = loadImage(bitmap, sensorOrientation);
  long endTimeForLoadImage = SystemClock.uptimeMillis();
  Trace.endSection();
  LOGGER.v("Timecost to load the image: " + (endTimeForLoadImage - startTimeForLoadImage));

  // Runs the inference call.
  Trace.beginSection( sectionName: "runInference");
  long startTimeForReference = SystemClock.uptimeMillis();
  tflite.run(inputImageBuffer.getBuffer(), outputProbabilityBuffer.getBuffer().rewind());
  long endTimeForReference = SystemClock.uptimeMillis();
  Trace.endSection();
  LOGGER.v("Timecost to run model inference: " + (endTimeForReference - startTimeForReference));

  // Gets the map of label and probability.
  Map<String, Float> labeledProbability =
      new TensorLabel(labels, probabilityProcessor.process(outputProbabilityBuffer))
          .getMapWithFloatValue();
  Trace.endSection();

  // Gets top-k results.
  return getTopKProbability(labeledProbability);
}
```

其中的核心代码无非是

```
tflite.run(inputImageBuffer.getBuffer(),
outputProbabilityBuffer.getBuffer().rewind());
```

简化来看

```
tflite.run(输入参数，装输出的容器)
```

继续分析：

- tflite是什么?

  ```
  tflite = new Interpreter(tfliteModel, tfliteOptions);
  ```

  tflite是一个Interpreter对象

  > tflite::Interpreter
  >
  > An interpreter for a graph of nodes that input and output from tensors.
  >
  > 张量输入和输出的节点图的解释器。
  >
  > 模型的解释器，解释tflite文件
  >
  > 也就是加载模型

- run方法是哪来的?

  就是Interpreter类里提供的一个方法/接口，Interpreter就是TensorFlow Lite库里提供的类!

综上，最**重要**的其实就是这个**run**方法

输出的流程

回到ClassifierActivity.java里，我们得到了results数组（其实是List）

因为results里存的是概率，也就是这幅图片像猫的概率是90%，像狗的概率是10%。

然后我们将这个概率（90%）和对应的label（猫）告诉UI组件，组件渲染

演示