> - 原文地址：Machine Learning for Android Developers with the Mobile Vision API— Part 1 —Face Detection

- 原文作者：Moyinoluwa Adeyemi
- 译文出自：掘金翻译计划
- 译者： Nicolas(Yifei) Li
- 校对者：jamweak, XHShirley

Android 开发者如何通过运动视觉 API 进行机器学习 - 第一部 - 人脸检测

在计算机科学中，机器学习是一个非常有意思的领域，它已经在我的最想学习的愿望清单中驻留已久。因为有太多来自于 `RxJava`， `Testing`， `Android N`， `Android Studio` 以及其他 `Android` 相关的技术更新，所以我都每能花时间来学习这个。甚至在 `Udacity` 还专门有一个有关机器学习的课程。😛。

让我非常激动的发现是，目前任意一个开发人员都能基于运动视觉(`Mobile Vision API`) 把机器学习运用在他们自己的应用程序中，这个技术来自于 Google，它让你甚至都不需要有机器学习领域的专业知识。你只需要关心怎么利用这些 `APIs` 。

在云服务和移动应用中，有很多运用于机器学习的 `APIs` ，但是在这些 `API` 中，我将只关注运动视觉 (`Mobile Vision`) `API` ，因为它是专门为 `Android` 开发者们创造的。目前运动视觉 (`Mobile Vision`) API 包含了三种功能: 人脸侦测 API，条形码侦测 API，文本侦测 API。在这篇文章中，我们将涉及人脸侦测的内容，并且会在之后的一系列文章里讨论剩下的两种功能。

人脸侦测 API

这个 API 被用于侦测和追踪在图片或视频中的人脸，但是它还不具备人脸识别的能力。它能在脸上进行侦测标定并提供人脸分类的功能。人脸标定是一系列在脸组成的点，例如眼睛，鼻子和嘴巴。人脸分类被用于检查那些标定的点是否符合某个特征，例如微笑的脸或者闭上了的眼睛，它们是目前仅支持的分类。这个 API 也能在不同的角度进行人脸侦测，并且记录欧式 (欧拉) 空间中的 Y坐标和 Z 的角度。

入门指南

我们准备创建一个具有两个过滤器的应用程序 `printf("%s Story", yourName)` 。请注意，本文的目的仅为了显示如何使用这个 API，所以这个初始版本的代码将不会进行测试或者遵循任何设计模式。也请注意，最好把所有的处理过程都从 UI 线程中分离。托管在 Github 上的源码 将会更新。

让我们开始吧...

- 在 `Android Studio` 中创建一个新的项目
- 将含有 Mobile Vision API 的 Google Play Services SDK 导入到你项目中 `app` 层级下的 `build.gradle` 文件内。在写这

篇文章的时候，最新版本是 `9.6.1\` 。请一定要小心这里，如果导入了整个 SDK 而不是仅导入你需要的那个 (play-services-vision)，那你一定会达到 65k 方法的限制。

```
compile 'com.google.android.gms:play-services-vision:9.6.1'
```

- 为了启用那些具有人脸侦测功能的依赖库，添加这个 `meta-data` 到你的 `manifest` 文件中。

```
<meta-data
    android:name="com.google.android.gms.vision.DEPENDENCIES"
    android:value="face"/>
```

- 下一步，你需要增加一个 *ImageView* 和 *Button* 到你的界面布局中。这个按钮通过选择一个图片开始，并处理这个图片，之后把它显示在 *ImageView*。这个图片能从摄像头或者照片库中获得和加载。为了节约时间，我保存并使用了一个在 `drawable` 文件夹内的图片。
- 在那个按钮的点击事件内，创建一个新的 *BitmapFactory.Options* 对象并且把 *inmutable* 属性设定为 `true` 。这确保了 `bitmap` 是可变的，以便我们对它动态地增加效果。

```
BitmapFactory.Options bitmapOptions = new BitmapFactory.Options();
bitmapOptions.inMutable = true;
```

- 下一步，从 `BitmapFactory` 类方法中用 *decodeResource* 方法创建一个新的 `Bitmap` 。你会使用来自你的 `drawable` 文件夹内相同的一个图片并且把 *BitmapOptions* 这个对象通过和之前一样的参数进行创建。

```
Bitmap defaultBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.image, bitmapOptions);
```

- 创建一个 *Paint* 对象并把它的 `style` 属性设定成 `stroke` 。这确保了图形不会被完全填充，因为我们需要确保头部在长方形内。

注意: 如果你正创建一个名为 *!(Recognize Me)* 的游戏，你需要在这个游戏内用一个图片挡住你的脸，所以你的对手不得不猜测你是谁，你想要把填充的形式设定成 *Paint.Style.FILL*

```
Paint rectPaint = new Paint();
rectPaint.setStrokeWidth(5);
rectPaint.setColor(Color.CYAN);
rectPaint.setStyle(Paint.Style.STROKE);
```

- 我们需要一个展现这个 `bitmap` 的画布。我们先创建一个有临时 `bitmap` 的画布。这个临时的 `bitmap` 会和之前的有着一样的尺寸，但是仅仅是这个一样的。我们之后要把原始的 `bitmap` 画在同一个画布上。

```
Bitmap temporaryBitmap = Bitmap.createBitmap(defaultBitmap.getWidth(), defaultBitmap
        .getHeight(), Bitmap.Config.RGB_565);

Canvas canvas = new Canvas(temporaryBitmap);
canvas.drawBitmap(defaultBitmap, 0, 0, null);
```

- 最后，让我们言归正传，说说看怎么使用 *FaceDectector* API 。因为我们在使用一个静态的图片，所以追踪功能被禁止了。它应该在视频上被启用。

```
FaceDetector faceDetector = new FaceDetector.Builder(this)
        .setTrackingEnabled(false)
        .setLandmarkType(FaceDetector.ALL_LANDMARKS)
        .build();
```

- 检查是否人脸侦测正常运作了。有可能第一次它不能正常工作，因为有一个依赖库需要被下载到设备上，而当你需要使用

它的时候还没有完全下载完毕。

```
if (!faceDetector.isOperational()) {
            new AlertDialog.Builder(this)
            .setMessage("Face Detector could not be set up on your device :(")
            .show();

    return;
}
```

- 下一步，我们用默认的 `bitmap` 创建一帧，然后调用人脸侦测功能获取人脸对象。

```
Frame frame = new Frame.Builder().setBitmap(defaultBitmap).build();
SparseArray sparseArray = faceDetector.detect(frame);
```

- 在这一步中矩形框画在这个人脸上。我们能获取每个人脸左边和上部的位置，但是我们还需要右边和底部的尺寸才能画矩形。为了解决这个问题，我们分别为左边和上部增加宽度和高度。

```
for (int i = 0; i < sparseArray.size(); i++) {
    Face face = sparseArray.valueAt(i);

    float left = face.getPosition().x;
    float top = face.getPosition().y;
    float right = left + face.getWidth();
    float bottom = right + face.getHeight();
    float cornerRadius = 2.0f;

    RectF rectF = new RectF(left, top, right, bottom);

    canvas.drawRoundRect(rectF, cornerRadius, cornerRadius, rectPaint);
}
```

- 我们之后创建一个新的 *BitmapDrawable*，它有一个临时的 `bitmap` 并且把它设定在界面布局中的 `ImageView` 中，之后这个人脸侦测的实例就能被释放了。

```
imageView.setImageDrawable(new BitmapDrawable(getResources(), temporaryBitmap));

faceDetector.release();
```

通过这些步骤，已经可以在每一张人脸上画出一个矩形框了。如果你想在每一张人脸上突出那些标定点，你只需要修改最后两步中的循环内容。你将为没一张脸依次加上标定点，获取标定点的 `x` 和 `y` 坐标，并且在每一个标定点处画上一个圆圈。

```
for (int i = 0; i < sparseArray.size(); i++) {
    Face face = sparseArray.valueAt(i);

    float left = face.getPosition().x;
    float top = face.getPosition().y;
    float right = left + face.getWidth();
    float bottom = right + face.getHeight();
    float cornerRadius = 2.0f;

    RectF rectF = new RectF(left, top, right, bottom);
    canvas.drawRoundRect(rectF, cornerRadius, cornerRadius, rectPaint);

    for (Landmark landmark : face.getLandmarks()) {
        int x = (int) (landmark.getPosition().x);
        int y = (int) (landmark.getPosition().y);
        float radius = 10.0f;

        canvas.drawCircle(x, y, radius, rectPaint);
```

```
        }
    }
```

有标定点的人脸图片

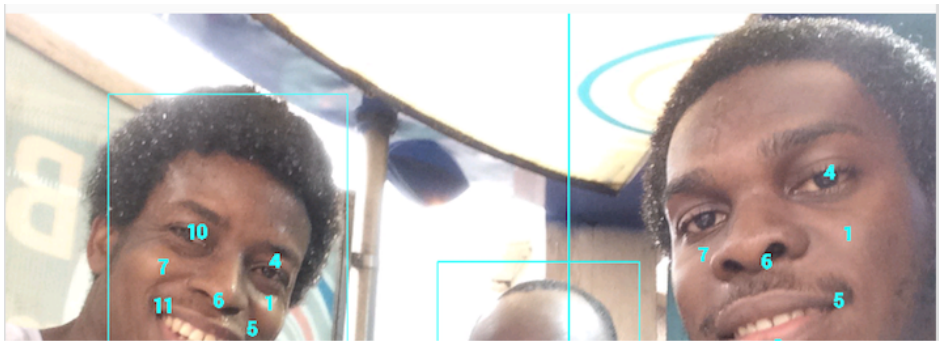
我很好奇这些标定点是什么展现出来的，所以我使用 *landmark.getType()* 来查明原由。原来每一个标定点都附带了特别的数字。

```
    for (Landmark landmark : face.getLandmarks()) {

        int cx = (int) (landmark.getPosition().x);
        int cy = (int) (landmark.getPosition().y);

        // canvas.drawCircle(cx, cy, 10, rectPaint);

        String type = String.valueOf(landmark.getType());
        rectPaint.setTextSize(50);
        canvas.drawText(type, cx, cy, rectPaint);
    }
```



当我们想在屏幕上定位跟某个人脸标定点相关的对象的时候，这就非常有用了。如果我们想创建自己的 `printf("%s Story", yourName)` 应用程序，我们要做的就是把一个图像放置到和其中一个标定点有关的位置上，因为我们现在知道了那些数字代表了什么。让我们开始如下的操作...

假设我们现在是一群海盗，并且我们想通过这个非常棒的 `printf("%s Story", yourName)` 滤镜来展现左眼上的眼罩。所以 `eyePatchBitmap` 会被画在左眼的位置。

```
    for (Landmark landmark : face.getLandmarks()) {

        int cx = (int) (landmark.getPosition().x);
        int cy = (int) (landmark.getPosition().y);

        // canvas.drawCircle(cx, cy, 10, rectPaint);
```

```
            // String type = String.valueOf(landmark.getType());
            // rectPaint.setTextSize(50);
            // canvas.drawText(type, cx, cy, rectPaint);

            // the left eye is represented by 4
            if (landmark.getType() == 4) {
                canvas.drawBitmap(eyePatchBitmap, cx - 270, cy - 250, null);
            }
        }
```

这里有更多 `printf("%s Story", yourName)` 应用程序的内容...

关于这个 `API` 还有很多内容。我们能更新这个应用程序，它可以用来在视频内追踪人脸并且允许过滤器跟随头部移动。文中提到的工程源码已经提交到了我们的 GitHub 仓库。