如何用TensorFlow在安卓设备上实现深度学习推断

对于个人和公司来说，存在许多状况是更希望在本地设备上做深度学习推断的：想象一下当你在旅行途中没有可靠的互联网链接时，或是要处理传输数据到云服务的隐私问题和延迟问题时。边缘计算（Edge computing）是一种在物理上靠近数据生成的位置从而对数据进行处理和分析的方法，为解决这些问题提供了方案。

以「Ok Google」这个功能为例：用一名用户的声音来训练「Ok Google」，他的手机在接收到这个关键词的时候就会被唤醒。这种小型关键词检测（small-footprint keyword-spotting，KWS）推断通常在本地设备上运行，所以你不必担心服务提供商随时监听你的声音。而云服务只在你发出指令后才启动。类似的概念可以扩展到智能家用电器或其他物联网设备上的应用，在这些应用中我们需要不依靠互联网进行免提语音控制。

更重要的是，边缘计算不仅为物联网世界带来了人工智能，还提供了许多其他的可能性和好处。例如，我们可以在本地设备上将图像或语音数据预处理为压缩表示，然后将其发送到云。这种方法解决了隐私和延迟问题。

在 Insight 任职期间，我用 TensorFlow 在安卓上部署了一个预训练的 WaveNet 模型。我的目标是探索将深度学习模型部署到设备上并使之工作的工程挑战！这篇文章简要介绍了如何用 TensorFlow 在安卓上构建一个通用的语音到文本识别应用程序。

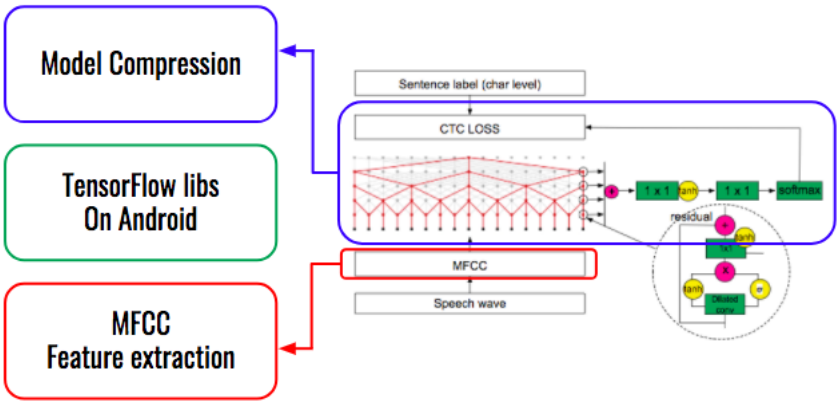

图 1. 流程概述。将 WaveNet 安装到安卓的三个步骤。

开发环境信息：

- Pixel, cpu type: ARM64
- Android 7.1.1
- Android NDK 15.2
- Android gradle plugin 2.3.0
- TensorFlow 1.3.0
- bazel 0.5.4-homebrew

详细教程和实现：https://github.com/chiachunfu/speech

第一步：模型压缩

为了将深度学习模型部署到移动/嵌入式设备上，我们应该致力于减少模型的内存占用，缩短推断时间，减少耗电。有几种方法可以实现这些要求，如量化、权重剪枝或将大模型提炼成小模型。

在这个项目中，我使用了 TensorFlow 中的量化工具来进行模型压缩。目前我只使用权重量化来减小模型大小，因为根据 Mac 上的测试结果，完整 8 位转换没有提供额外的好处，比如缩短推断时间。（由于 requant_range 中的错误，无法在 Pixel 上运行完整的 8 位模型）。由于 8 位量化工具不适合 CPU，时间甚至翻了一倍。如果你有兴趣了解更多关于量化的实用建议，可以阅读 Pete Warden 这篇很棒的文章（https://petewarden.com/2017/06/22/what-ive-learned-about-neural-network-quantization/）。

对模型进行权重量化：

1. 将模型写入协议缓冲区文件。

2. 从源安装和配置 TensorFlow（https://www.tensorflow.org/install/install_sources）。

3. 在 TensorFlow 目录下运行下列命令行：

```
bazel build tensorflow/tools/graph_transforms:transform_graphbazel-bin/tensorflow/tools/graph_transforms/transform_graph \  --in_graph=/your/.pb/file \  --outputs="
```

以我的项目为例，在量化权重后，预训练的 WaveNet 模型的大小从 15.5Mb 下降到了 4.0Mb。现在可以将这个模型文件移动到安卓项目中的「assets」文件夹。

第二步：适用于安卓的 TensorFlow 库

要用 TensorFlow 构建安卓应用程序，我推荐从 TensorFlow Android Demo开始。在我的项目中，我把 TF speech example 作为模板。这个示例中的 gradle 文件帮助我们构建和编译安卓的 TF 库。但是，这个预构建的 TF 库可能不包括模型所有必要的 ops。我们需要想清楚 WaveNet 中需要的全部 ops，并将它们编译成适合安卓 apk 的.so 文件。为了找到 ops 的完整列表，我首先使用 tf.train.write_graph 输出图的详细信息。然后在终端中运行下列命令：

```
grep "op: " PATH/TO/mygraph.txt | sort | uniq | sed -E 's/^.+"(.+)".?$/\1/g'
```

接着，编辑/tensorflow/tensorflow/core/kernels/里的 BUILD 文件，在 Android libraries section 中的「android_extended_ops_group1」或「android_extended_ops_group2」里添加缺失的 ops。我们也可以删除不必要的 ops，使 .so 文件变得更小。现在，运行下列命令：

```
bazel build -c opt //tensorflow/contrib/android:libtensorflow_inference.so \--crosstool_top=//external:android/crosstool \--host_crosstool_top=@bazel_tools//tools/c
```

你将在这里找到 libtensorflow_inference.so 文件：

```
bazel-bin/tensorflow/contrib/android/libtensorflow_inference.so
```

除了 .so 文件之外，我们还需要一个 JAR 文件。运行：

```
bazel build //tensorflow/contrib/android:android_tensorflow_inference_java
```

你将在这里找到该文件：

```
bazel-bin/tensorflow/contrib/android/libandroid_tensorflow_inference_java.jar
```

现在，可以将 .so 和 .jar 文件一起移到你的安卓项目中的「libs」文件夹。

第三步：在安卓上的数据预处理

最后，让我们将输入数据处理成模型训练所需格式。对于音频系统来说，原始的语音波被转换成梅尔频率倒谱系数（MFCC）来模拟人耳感知声音的方式。TensorFlow 有一个音频 op，可以执行该特征提取。然而，事实证明，实现这种转换存在一些变体。如图 2 所示，来自 TensorFlow audio op 的 MFCC 不同于 librosa 提供的 MFCC。librosa 是一个被预训练的 WaveNet 作者们用来转换训练数据的 Python 库。

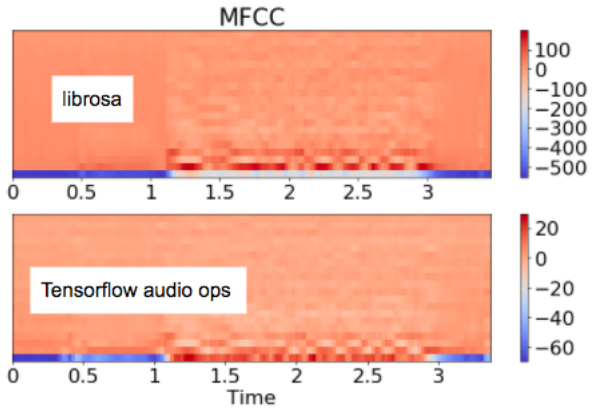

图 2. 来自 librosa 和 TensorFlow audio ops 的 MFCC 处在不同的刻度范围。

如果您正在训练自己的模型或重训练一个预先训练好的模型，那么在处理训练数据时，一定要考虑设备上的数据通道。最终，我在 Java 中重写了 librosa MFCC 来处理转换问题。

结果

图 3 展示了 app 的截图和示例。由于模型中没有语言模型，而且识别仅在字符级，因此句子中出现了一些拼写错误。虽然没有经过严格的测试，但在量化之后，我确实发现准确率略有下降，以及整个系统对周围的噪声很敏感。

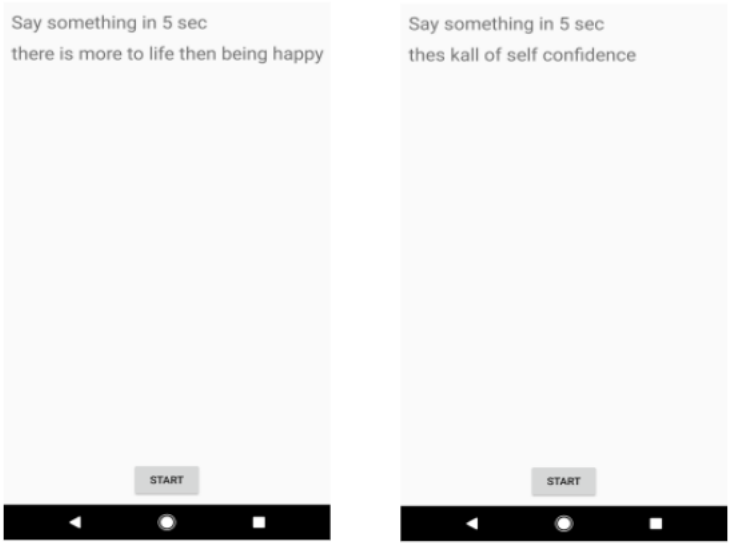

图 3. App 中两个示例的截图。

下表所示推断时间是对 5 秒音频的 10 次测试的平均值。推断时间在两个平台上都略有增加，而不是减少，因为权重量化有助于缩小文件大小，但不太能优化推断时间或耗电情况。

	Before Quantization	After Quantization
Mobile (ARM)	1.32	1.41
Mac (Intel Core i5)	2.27	2.46

(Seconds per 5-sec audio)

表 1. 权重量化前后的推断时间。测试环境是我的 Pixel 手机和 Macbook air。

接下来做些什么?

有两件重要的事情可以让这个项目更进一步，也可以为社区提供额外的教程和演练，以便在边缘设备上部署一个现实语音识别系统。

- 提高语音识别性能：添加拼写校正的语言模型和噪声下采样模型，以降低周围噪声的影响。
- 改善推断时间和耗电情况：用 NEON 或其他架构进行低层次优化，用 gemmlowp 进行低精度矩阵计算。

GitHub 地址：https://github.com/chiachunfu/speech

原文链接：https://blog.insightdatascience.com/ok-google-how-do-you-run-deep-learning-inference-on-android-using-tensorflow-c39fd00c427b