文件格式：h5/hdf5, pb, tflite,

# TensorFlow Lite（Google）

[2017年5月](https://zh.wikipedia.org/wiki/2017年5月)，Google宣布从[Android Oreo](https://zh.wikipedia.org/wiki/Android_Oreo)开始，提供一个专用于Android开发的软件栈TensorFlow Lite。

官网：<https://www.tensorflow.org/lite>

TensorFlow Lite主要由两个组件构成，解释器（Interpreter）和转换器（Converter）。

## 基本工作流程

1. **选择模型**：选择一个自定义的TensorFlow网络模型，也可以是TF Lite里自带的预训练网络（<https://www.tensorflow.org/lite/models>）
2. **转换模型**：如果是自定义的网络，使用转换器转换成TF Lite的格式（.tflite）
3. **部署到设备**：通过TF Lite Interpreter在设备上运行这个模型，可以通过多种语言的API实现。
4. **优化模型**：通过模型优化工具对模型进行优化：在尽量不影响准确率的情况下缩减尺寸、提高效率（<https://www.tensorflow.org/lite/performance/model_optimization>）

## Converter

转换器（Converter）可以将一些框架下的网络模型转换成解释器可以使用的格式（.tflite文件），转换器可以通过命令行（toco，tflite\_convert）或者Python接口进行调用。

Converter可以转换几种格式的数据为tflite文件：

* TensorFlow GraphDef（.pb）文件
* TensorFlow SavedModel
* Keras模型（.h5）文件
* TensorFlow 2.0 Concrete Function

简单命令：

tflite\_convert --keras\_model\_file=MODEL.h5 --output\_file=MODEL.tflite

参考：

<https://www.tensorflow.org/lite/convert>

<https://www.tensorflow.org/lite/r2/convert>

## Interpreter

解释器（Interpreter）通过以下步骤来运行模型：

1. 加载模型：将描述网络模型的 .tflite文件加载到内存中
2. 数据转换：将输入数据转换为网络模型认可的格式，比如图像的缩放
3. 运行模型：
4. 解释输出：将模型输出的张量转换为应用程序可以利用的格式

解释器可以以库的形式运行在安卓、iOS和Linux上。

参考：

<https://www.tensorflow.org/lite/guide/inference>

## 性能优化

## 预训练模型

TF Lite官网自带了五种功能的预训练网络模型：

* **图像分类（Image Classification）**：给图像分类，包括人、动物、植物、活动、地点等。模型会输出一组0到1之间的数字（和为1），表示各种类别的可能性。图像分类的缺省模型是MobileNetV1。

（<https://www.tensorflow.org/lite/models/image_classification/overview>）

* **物体检测（Object Detection）**：检测图像中的若干物体，用矩形框标示出位置。模型的输出为一组数据，其中每个数据由类别、得分（置信度）、位置三部分构成，位置则包括上、下、左、右四个数据。物体检测的缺省模型是用coco数据集训练的MobileNetV1。

（<https://www.tensorflow.org/lite/models/object_detection/overview>）

* **智能问答（Smart Reply）**：智能问答可以根据聊天内容自动生成回复，而且是上下文相关的。（<https://www.tensorflow.org/lite/models/smart_reply/overview>）
* **姿势评估（Pose Estimation）**：评估图像/视频中人的姿势，或者说是定位其中人的各个身体节点（包括鼻、眼、耳、肩、肘、腕、臀、膝、踝）的位置。姿势评估用的模型是PoseNet。

**（**<https://www.tensorflow.org/lite/models/pose_estimation/overview>**）**

* **图像分割（Segmentation）**：图像分割有点类似物体检测，其区别在于标示的方式不是矩形框，而是物体的准确轮廓。图像分割用的缺省模型是DeepLabV3。

**（**<https://www.tensorflow.org/lite/models/segmentation/overview>**）**

其他已经在TF Lite中可以工作的预训练好的模型：

<https://www.tensorflow.org/lite/guide/hosted_models>

或者可以从TensorFlow Hub上自行下载需要的模型（<https://www.tensorflow.org/hub>）进行转换。

## Transfer Learning

**迁移学习（Transfer Learning）**，指的是在已经训练好的网络上做适度训练，以便解决一个类似的问题，比如用于辨识小轿车的模型可以通过少量训练用于辨识卡车。迁移学习的好处是不用从头对网络进行训练，大大减少了训练时间和训练数据。

参考：

[https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android/#1](https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android/" \l "1)

## 交叉编译 label\_image

label\_image是TF lite里自带的一个对tflite的C++使用样例，位于：

tensorflow/tensorflow/lite/examples/label\_image

对其在x86 android上进行交叉编译的步骤如下：

* 修改tensorflow/tensorflow/WORKSPACE文件，增加：

android\_ndk\_repository(

name = "androidndk", # Required. Name \*must\* be "androidndk".

path = "/home/xxx/ndk", # Optional. Can be omitted if `ANDROID\_NDK\_HOME` environment variable is set.

)

其中path指向ndk的所在目录

* 在tensorflow/中执行编译：

bazel build //tensorflow/lite/examples/label\_image:label\_image --crosstool\_top=//external:android/crosstool --cpu=x86 --host\_crosstool\_top=@bazel\_tools//tools/cpp:toolchain --cxxopt="-std=c++11"

可进行在x86 android上的交叉编译，如果编译中出现std::round相关的错误，去掉std即可。

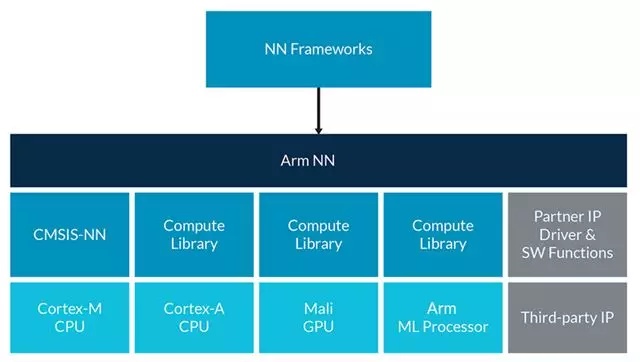
* 将程序和mobilenet网络模型上传到设备上的同一个目录下：
  + tensorflow/tensorflow/bazel-bin/tensorflow/lite/examples/label\_image/label\_image
  + examples/lite/examples/image\_classification/android/app/src/main/assets/下的tflite网络模型文件和label文件

# ARM NN（ARM）

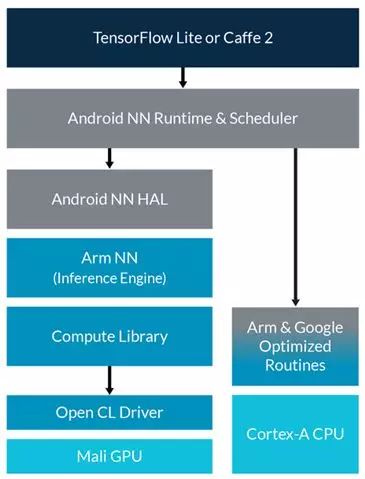
**ARM NN（ARM Neural Network）**是ARM推出的一套开源软件/工具，使得神经网络可以高效的运行在ARM设备上，它桥接了上层的神经网络框架（TensorFlow，Caffe，TensorFlow Lite，ONNX）和下层的ARM系列硬件（Cortex CPU、Mali GPU、ARM ML处理器）。这样，在其它主机上训练的网络，可以通过ARM NN在ARM硬件上无缝运行。



ARM NN通过**Compute Library**使用Cortex-A，Mali和ML处理器，通过**CMSIS-NN**使用Cortex-M处理器：



ARM NN也提供对谷歌的Android NNAPI的支持：



## ARM Compute Library

**ARM Compute Library**是一个专为ARM的 CPU和GPU优化过的底层函数的集合，这个函数集基于Cortex-A、Mali和ARM的ML处理器，向外提供了图像处理、计算机视觉和机器学习的功能。

## CMSIS-NN

**CMSIS（Cortex Microcontroller Software Interface Standard）**是ARM的Cortex-M系列处理器提供的具体硬件无关的HAL层接口，其基本组件是CMSIS-Core，除此之外还有CMSIS-SVD、CMSIS-RTOS、CMSIS-DSP、CMSIS-Driver、CMSIS-Packs、CMSIS-DAP。

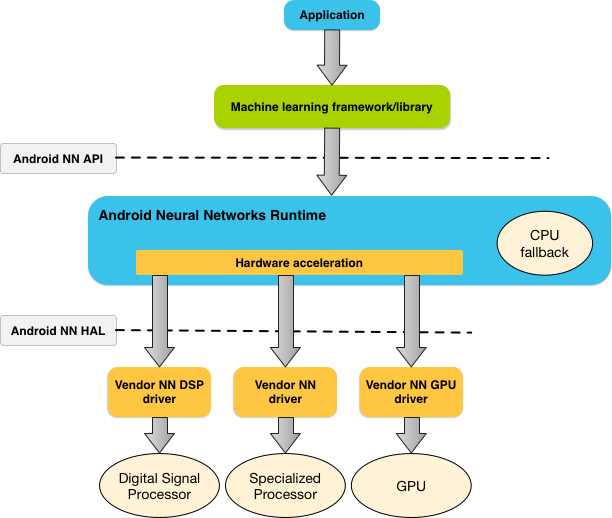
CMSIS-NN则是其中专为神经网络的应用而提供的统一抽象接口。

# NNAPI（Google）

**NNAPI（Neural Network API）**是Google推出的Anroid的C语言API，用于提高神经网络在安卓设备上的计算效率。NNAPI位于上层的机器学习库（TensorFlow Lite、Caffe2等）的下层，这些上层机器学习库在设备外搭建并训练神经网络模型，之后通过NNAPI在安卓设备上运行。

通常App不会直接调用NNAPI，而是直接使用上层的机器学习库。而这些机器学习库会通过调用NNAPI来实现加速。

基于App的需求和硬件的条件，NNAPI runtime会机动的分配计算载荷到设备的计算部件上，这些计算部件可以是专用的NN硬件、GPU以及DSP，对于没有这些计算部件（或者驱动）的设备，NNAPI runtime会优化代码并在CPU上执行。



NNAPI和ARMNN的相同之处在于，都是为了将上层的神经网络框架和下层的嵌入式硬件桥接起来，区别在于NNAPI仅为Android提供，下层可以由多种硬件实现。而ARMNN仅封装下层的ARM硬件，而可以供给不同的上层使用。

参考：<https://developer.android.com/ndk/guides/neuralnetworks>

# CNNDroid

CNNdroid是一个用于在Android设备上执行已经训练好的CNN的开源库。