快速部署 Tensorflow 和 TFLITE 模型在 Jacinto7 Soc

Fredy Zhang *EP FAE*

摘要

Jacinto™ 7 TDA4xx 是目前 TI 最新一代的汽车处理器，面向 ADAS 和自动驾驶车辆 (AV) 应用， 并且建立在 TI 在 ADAS 处理器市场十余年领先地位中积累的广泛市场知识基础之上。TDA4VM 以业界领先的功耗/性能比为传统和深度学习算法提供高性能计算，并具有很高的系统集成度，从而使支持集中式 ECU 或多种传感器的高级汽车平台实现可扩展性和更低的成本。

TI 最新一代的汽车处理器 TDA4VM 集成了高性能计算单元 C7x DSP（Digital Signal Processor）和 Deep-learning Matrix Multiply Accelerator（MMA），TIDL 是 TI 的 Deep Learning 加速库，用于加速 TI 嵌入式设备上的深度神经网络 Deep Neural Networks (DNN)。 上一代产品 TDA2/3 系列处理器，集成了计算单元 DSP（Digital Signal Processor）和 EVE（Embedded Vision/Vector Engine），用于加速深度神经网络。相比于上一代 TDA2/TDA3 系列处理器，最新一代的 TDA4 处理器在算例上得到了大幅提高的同时，在软件方面提供了更好地支持。

基于深度神经网络 (DNN) 的机器学习算法应用于许多行业，例如机器人、工业和汽车。越来越多的基于 DNN 的机器学习算法被应用于 ADAS 产品中，这些神经网络通常需要大量的计算，这些计算都可以在 TI TDA4 系列处理器中的 C7x DSP 和 MMA 中高效地运行。RTOS SDK 中集成了众多的 Demo，Demo 展示 TIDL 在 TDA4 处理器上对实时的语义分割和 SSD 目标检测的能力。TIDL 当前支持的训练框架有 Tensorflow、Pytorch、Caffe 等，用户可以根据需要选择合适的训练框架进行模型训练。

Tensorflow Lite(简称 TF Lite) 是 Tensorflow 用于移动设备和嵌入式设备的轻量级解决方案。具有轻量级、跨平台、快速的特点。TF 在 2.x 版本中已经不再计划支持 Frozon Graph。TF Lite 模型格式可同时支持 TF 1.x.x 和 TF 2.x.x。 因此，针对 TF 的解决方案， TIDL 里面推荐使用 TF Lite 模型格式。

TIDL 涉及到深度学习领域和嵌入式设计领域，同时，TIDL 给大部分算法工程师的使用带来了大量困难。本文旨在通过中文的方式，快速、直接地基于 TIDL 讲解如何快速部署 Tensorflow 和 TFLITE 模型，并验证模型的正确性。

修改记录

Version	Date	Author	Notes
1.0	Feb 2022	Fredy Zhang	First release

目录

1. 基本介绍 4
1.1. TIDL 简介 4
1.2. Tensorflow 简介 5
1.3. TF Lite 简介 5
2. 导入模型到 TIDL 6
2.1. Tensorflow 6
2.1.1.TIDL 对 Tensorflow 的支持 6
2.1.2.Tensorflow 模型导入 TIDL 8
2.2. TF Lite 9
2.2.1.TIDL 对 TF Lite 的支持 9
2.2.2.TF Lite 模型转换 10
2.2.3.TF Lite 模型导入 TIDL 11
3. 模型优化与导入 12
3.1. Tensorflow 模型优化 12
3.2. 模型导入 13
4. 模型 PC 推理 15
4.1. PC 模型推理 16
4.2. PC 模型部署 17
5. 模型 EVM 推理 20
5.1. EVM 模型推理 21
5.2. EVM 模型部署 22
6. FAQ 25
6.1. SDK 中使用的 Tensorflow 版本及推荐的 Tensorflow 版本？ 25
6.2. 如何得到 Frozen Graph 模型？ 25
6.3. 如何优化 Tensorflow 模型？ 25
6.4. 有哪些 Tensorflow 模型在 TIDL 验证过？ 25
6.5. TIDL Importer 工具导入 Tensorflow 模型需要注意哪些方面？ 25
6.6. TIDL 如何打印 log 信息？ 26
7. 参考 26

图

图 1. TIDL SW Framework 4
图 2. TF Lite Architecture 6
图 3. Tensorflow 模型导入 TIDL 流程 8
图 4. TIDL Importer 工具处理流程 9
图 5. TFLITE 模型导入 TIDL 流程 11
图 6. TIDL Importer 工具处理流程 12
图 7. 模型推理 15
图 8. 图片输入 16
图 9. Image Classification Application 17

图 10. PC Image Classification Application 20
图 11. EVM Image Classification Application 24

表

表 1. Tensorflow 支持的模型 7
表 2. Demo 环境 8
表 3. TFLite 支持的模型 10
表 4. Demo 环境 11
表 3. Input Image 15

1. 基本介绍

1.1. TIDL 简介

TIDL 用于加速 TI 嵌入式设备上的深度神经网络 (DNN)。 它支持 TI 的最新一代处理器 TI Jacinto7 TDA4 处理器。 TDA4 处理器属于 TI Jacinto7 家族的处理器，基于异构、可扩展的架构开发，此架构包含了 TI 数字信号处理 C7x DSP 和 C66x DSP、Cortex A72、Cortex-R5F、图形处理器 GPU 等核，拥有 MMA 深度学习加速器，属于多核异构的架构。Cortex A72 可用于通用计算、图形处理器 GPU 用于 3D 图像的加速、DSP 可用于算法的加速、C7x&MMA 可支持深度学习的处理、Cortex-R5F 可用于外设的控制和图像的前后处理等。多核异构的优点是采用适合的核做擅长的事，再加上专用硬件加速器也可处理特定任务，从而在性能、功耗和成本上达到最佳平衡。

TDA4 处理器集成了 TI 最新一代 C7x DSP 和 TI 的 DNN 加速器 (MMA) 用于执行 DNN。 TIDL 可用于德州仪器 (TI) 的各种嵌入式设备，是作为 TI 软件开发套件 (SDK) 的一部分发布的，同时还有其他计算机视觉功能和优化的库，包括 OpenCV。

基于深度神经网络 (DNN) 的机器学习算法用于许多行业，例如机器人、工业和汽车。越来越多的基于 DNN 的机器学习算法被应用于 ADAS 产品中，这些神经网络通常需要大量的计算，这些计算都可以在 TI TDA4 系列处理器中的 C7x DSP 和 MMA 中高效地运行。RTOS SDK 中集成了众多的 Demo 展示 TIDL 在 TDA4 处理器上对实时的语义分割和 SSD 目标检测的能力。TIDL 当前支持的训练框架有 Tensorflow、Pytorch、Caffe 等，用户可以根据需要选择合适的训练框架进行模型训练。

如图 1 所示，是 TIDL 的软件框架。在 TIDL 上，深度学习网络应用开发主要分为三个大的步骤（以 TI Jacinto7 TDA4VM 处理器为例）：

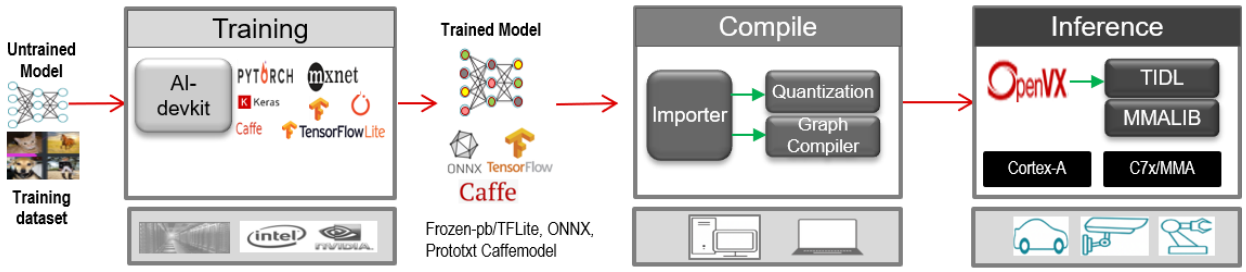

图 1. TIDL SW Framework

1. 基于 Tensorflow、Pytorch、Caffe 等训练框架，训练模型：选择一个训练框架，然后定义模型，最后使用训练的数据集训练出满足需求的模型。
2. 基于 TI Jacinto7 TDA4VM 处理器导入模型： 训练好的模型，需要使用 TIDL Importer 工具导入成可在 TIDL 上运行的模型。导入的主要目的是对输入的模型进行量化、优化并保存为 TIDL 能够识别的网络模型和网络参数文件。

3. 基于 TI Jacinto7 SDK 验证模型，并在应用里面部署模型：
    a. PC 上验证并部署
        i. 在 PC 上使用 TIDL 推理引擎进行模型测试。
        ii. 在 PC 上使用 OpenVX 框架开发程序，在应用上进行验证。
    b. EVM 上验证并部署
        i. 在 EVM 上使用 TIDL 推理引擎进行模型测试。
        ii. 在 EVM 上使用 OpenVX 框架开发程序，在应用上进行验证。

1.2. Tensorflow 简介

TensorFlow 最初是由 Google Brain 团队创建的。TensorFlow 是一个端到端开源机器学习平台。它拥有一个全面而灵活的生态系统，其中包含各种工具、库和社区资源，可助力研究人员推动先进机器学习技术的发展，并使开发者能够轻松地构建和部署由机器学习提供支持的应用。

TensorFlow 提供多个抽象级别，因此您可以根据自己的需求选择合适的级别。您可以使用高阶 Keras API 构建和训练模型，该 API 让您能够轻松地开始使用 TensorFlow 和机器学习。TensorFlow 社区是一个由开发者、研究人员、创想家、生手和问题解决者组成的活跃群组。其体系结构使其可以轻松部署在任何计算设备上。例如，可以将其部署到具有 CPU，GPU 或 TPU 的服务器群集，也可以部署到移动设备和边缘设备。Tensorflow 灵活部署及活跃的社区使其成为深度神经网络环境的最爱。2017 年 2 月 16 日，Google 正式对外发布 Google Tensorflow 1.0 版本，其 2.0 版本发布于 2019 年，2.0 版本引入了 Keras。Keras 是用于构建神经网络的高级 API，大大加强了集成度，减少了使用难度，到现在为止， 最新版本是 2.7。

1.3. TF Lite 简介

TensorFlow 最初是由 Google Brain 团队创建的。TensorFlow 是一个端到端开源机器学习平台。它拥有一个全面而灵活的生态系统，其中包含各种工具、库和社区资源，可助力研究人员推动先进机器学习技术的发展，并使开发者能够轻松地构建和部署由机器学习提供支持的应用。

TensorFlow 提供多个抽象级别，因此您可以根据自己的需求选择合适的级别。您可以使用高阶 Keras API 构建和训练模型，该 API 让您能够轻松地开始使用 TensorFlow 和机器学习。TensorFlow 社区是一个由开发者、研究人员、创想家、生手和问题解决者组成的活跃群组。其体系结构使其可以轻松部署在任何计算设备上。例如，可以将其部署到具有 CPU，GPU 或 TPU 的服务器群集，也可以部署到移动设备和边缘设备。Tensorflow 灵活部署及活跃的社区使其成为深度神经网络环境的最爱。2017 年 2 月 16 日，Google 正式对外发布 Google Tensorflow 1.0 版本，其 2.0 版本发布于 2019 年，2.0 版本引入了 Keras。Keras 是用于构建神经网络的高级 API，大大加强了集成度，减少了使用难度，到现在为止， 最新版本是 2.7。

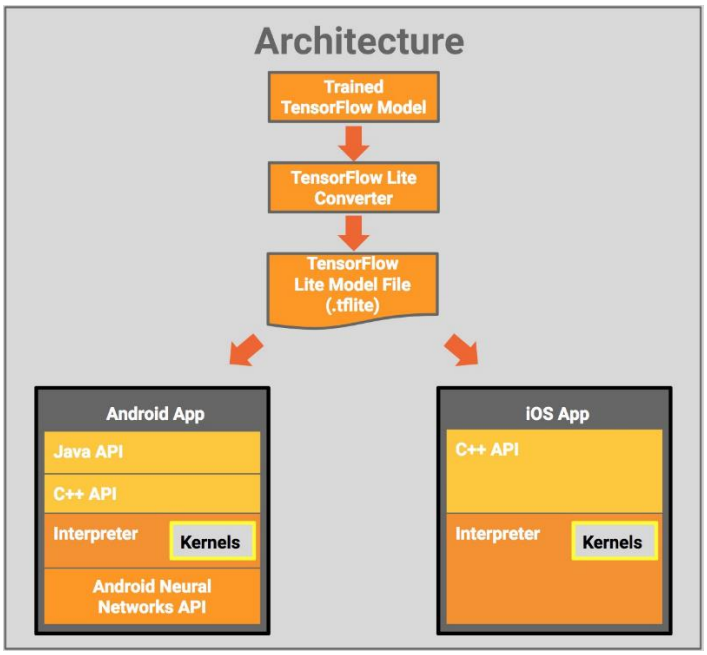

图 **2.** TF Lite Architecture

TF Lite 是 Tensorflow 针对移动和嵌入式设备的轻量级解决方案（深度学习框架）。 TensorFlow Lite 提供了转换 TensorFlow 模型，并在移动端（mobile）、嵌入式（embeded）和物联网（IoT）设备上运行 TensorFlow 模型所需的所有工具。

如下图 2 TF Lite 框架，TF Lite 组件报考： 保存在磁盘中的 TF 模型（Trained Tensorflow Model）；TF Lite 模型转换器（Tensorflow Lite Converter）和 TF Lite 模型文件（Tensorflow Lite Model File）。其使用流程是：从训练好的 TF 模型开始； 然后，使用 TF Lite 模型转换器转将正常训练的模型转换为 TF Lite 文件格式；然后我们就可以在移动应用程序中使用该转换后的文件。

2. 导入模型到 TIDL

2.1. Tensorflow

2.1.1. TIDL 对 Tensorflow 的支持

TIDL 支持 Tensorflow 模型的导入，预训练的模型使用的 Tensorflow 的版本是 Tensorflow - 1.12, 当前发布的版本 PSDKRA(PROCESSOR-SDK-RTOS-J721E_08.01.00.11), 其 TIDL 的版本是 08_01_00_05。 当前 SDK 中已经验证并且支持的模型如表 1 所示：

表 1. Tensorflow 支持的模型

Num	Network Architecture	Source
1	MobileNet-1.0 V1	Frozen Graph Link
2	InceptionNet v1	Checkpoint Link
3	MobileNet-1.0 V2	Frozen Graph Link
4	Resnet 50 V1-TF	Checkpoint Link
5	Resnet 50 V2-TF	Checkpoint Link
6	ssd_mobilenet_v1_0.75 SSD	Link
7	ssd_mobilenet_v1 1.0 SSD	Link
8	ssd_mobilenet_v2 SSD	Link

由于 Tensorflow 2.x 不支持 Frozen Graph，且 Tensorflow 2.x 开始使用 Keras 模型，导出是 savedmodel 格式或者 h5 格式。网上也有一些方法可以使用 Tensorflow 2.x 生成 Frozen Graph 模型。在导入的过程中也可能遇到以一些问题，比如，算子版本过高、算子新增参数不支持、算子本身不支持等情况。

注意

Tensorflow 2.x 不支持 Frozen Graph， 因此，推荐使用 Tensorflow 1.x 版本。

当前 TIDL 仅支持 Tensorflow Frozen Graph 模型导入，要想在 TIDL 里面运行 Tensorflow 模型，首先需要转换为 Frozen Graph 模型，得到 TensorFlow 上的 Frozen Graph 模型。Frozen Graph 模型进行优化之前无法导入。 因此需要先执行 optimize_for_inference.py 以创建优化模型文件。只有经过优化过的模型才能被 Importer 工具导入。

注意

Savedmodel 格式的模型文件不能直接导入。要想在 TIDL 里面运行 Tensorflow 模型：

1. Savedmodel 格式需要转换为 Frozen Graph 模型。
2. Frozen Graph 模型需要优化才能够使用 Importer 工具导入。

2.1.2. *Tensorflow 模型导入 TIDL*

PSDKRA 中集成了众多的 Demo 应用，这些 TIDL 应用展示了 TIDL 在 TDA4 处理器上对实时的语义分割和 SSD 目标检测的能力。TIDL 涉及到深度学习领域和嵌入式设计领域，同时，TIDL 的给大部分算法工程师的使用带来了大量困难。开发者、研究人员在广泛使用 Tensorflow，Tensorflow 的模型不能直接在 TIDL Runtime 上运行，需要经过导入才能够被 TIDL 所支持。因此，对其流程进行了整理。本文后续 Demo 所使用的开发环境如表 1 所示：

表 2. Demo 环境

名称	内容
PSDKRA SDK	PROCESSOR-SDK-RTOS-J721E_08.01.00.11
TIDL 版本	tidl_j7_08_01_00_05
PC 开发环境	Ubuntu18.04
Python 版本	3.6
SOC	TDA4VM

如图 2 所示，Tensorflow 模型导入 TIDL 的流程（本文将以 MobileNet V2 为例介绍 Tensorflow 模型的导入和推理步骤）：

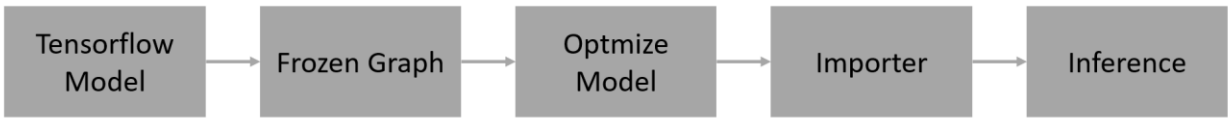

图 **3**. Tensorflow 模型导入 TIDL 流程

1. Tensorflow Model: 基于 Tensorflow 训练框架，使用数据集进行训练，然后得到 Tensorflow 模型。TIDL 仅支持.pb 格式的模型，在 Tensorflow 2.x 上，可以使用 SavedModel 格式。
2. Frozen Graph: 得到 SavedModel（.pb）格式的模型，需要先转换为 Frozen Model。Tensorflow 1.x 转换 Frozen Graph 模型请参考这里。Tensorflow 2.x 转换 Frozen Graph 模型请参考这里。
3. Optimize Model：Frozen Graph 模型进行优化之前无法导入，因此需要先进行优化。优化采用 optimize_for_inference.py 脚本。
4. Importer：TIDL Runtime 提供了 Importer 工具, Importer 工具可以支持 Tensorflow 后缀为 pb 格式的模型导入，Importer 在内部执行各种处理，最后提供要在 TIDL Runtime 推理引擎上推理的模型。Importer 具体处理过程如图 3 所示：

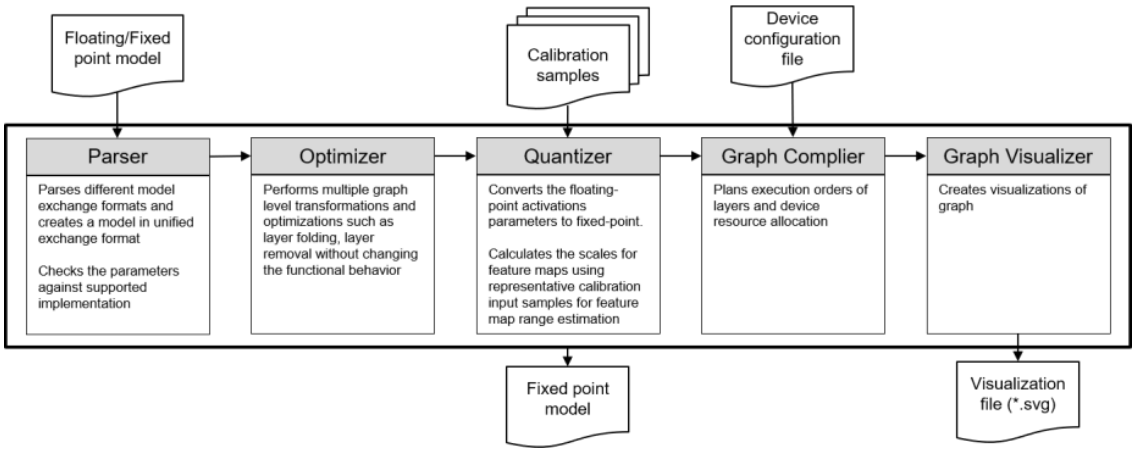

图 **4**. TIDL Importer 工具处理流程

5. Inference：TIDL 里面提供了应用程序可以对导入后的模型进行推理。该应用程序可以支持 PC（Emulation）和 Target（TDA4VM EVM）。PC_dsp_test_dl_algo.out 运行在 Ubuntu 上。TI_DEVICE_a72_test_dl_algo_host_rt.out 运行在 EVM 上。

接下来的两章节，将会快速、直接地基于 SDK 讲解 TIDL PC 和 EVM 的运行环境，及如何快速部署 Tensorflow 模型，并验证模型的正确性。 第一步和第二步这里不做具体介绍。第 3 章节将介绍第 3 步模型优化和第 4 步模型导入。第四章节将具体介绍在 TIDL 上面的推理部分。

2.2. TF Lite

2.2.1. TIDL 对 TF Lite 的支持

TIDL 支持 TF Lite 模型的导入，并可以同时支持 TF 1.x.x 和 TF 2.x.x 模型。TIDL 仅支持 TF Lite 定点模型，仍然需要校准图像作为输入。 因此可以使用 TF 1.x.x 或 2.x.x 来训练模型。然后，通过 TF Lite Converter 转换成 TF Lite 模型。当前发布的版本 PSDKRA(PROCESSOR-SDK-RTOS-J721E_08.01.00.11), 其 TIDL 的版本是 08_01_00_05。 当前 SDK 中已经验证并且支持的模型如表 1 所示：

表 3. TFLite 支持的模型

Num	Network Architecture	Source
1	MobileNet-1.0 V1	Link
2	MobileNet-1.0 V2	Link
3	InceptionNet v1	Link
4	InceptionNet V3	Link
5	Efficientnet-Lite 0	Link
6	deeplabv3_mnv2	Link
7	deeplabv3_mnv2_dm05	Link
8	mobileNetv1_ssd	Link
9	mobileNetv2_ssd	Link
10	Efficientnet-Lite 0	Link
11	Efficientnet-Lite 4	Link

2.2.2. TF Lite 模型转换

TF Lite 的设计旨在在各种设备上高效执行模型。这种高效部分源于在存储模型时，采用了一种特殊的格式。TF 模型在能被 TensorFlow Lite 使用前，必须转换成这种格式。

有多种方式可以获得 TF 模型，从使用预训练模型（pre-trained models）到训练自己的模型。为了在 TF Lite 中使用模型，模型必须转换成一种特殊格式。TF Lite 团队提供了一系列预训练模型（pre-trained models），用于解决各种机器学习问题。这些模型已经转换为能与 TF Lite 一起使用，且可以在您的应用程序中使用的模型。许多其他地方得到预训练的 TensorFlow 模型，包括 TensorFlow Hub。在大多数情况下，这些模型不会以 TF Lite 格式提供，您必须在使用前转换（convert）这些模型。

注意

TF Lite 仅支持部分 TF 运算符，所以并非所有模型都能转换。参看 参考 2 链接获取更多信息。

如果您设计并训练了您自己的 TF 模型，或者您训练了从其他来源得到的模型，在使用前，您需要将此模型转换成 TF Lite 的格式。

TF Lite 转换器（converter）是一个将训练好的 TF 模型转换成 TF Lite 格式的工具。TF Lite 解释器（interpreter）是一个库（library），它接收一个模型文件（model file），执行模型文件在输入数据（input data）上定义的运算符（operations），并提供对输出（output）的访问。

TF Lite 转换器以 Python API 的形式提供。下面的例子说明了将一个 TF `SavedModel` 转换成 TF Lite 格式的过程，推荐用 Python API 进行转换：

```
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_model = converter.convert()
open("converted_model.tflite", "wb").write(tflite_model)
```

TF Lite 支持将值的精度从全浮点降低到半精度浮点 (float16) 或 8 位整数。每种选择都要在模型大小和准确度上进行权衡取舍，而且有些运算有针对这些降低了精度的类型的优化实现。如需对 TF Lite 模型进行优化，请参考第六章节链接 3 进行优化。

2.2.3. TF Lite 模型导入 TIDL

PSDKRA 中集成了众多的 Demo 应用，这些 TIDL 应用展示了 TIDL 在 TDA4 处理器上对实时的语义分割和 SSD 目标检测的能力。TIDL 涉及到深度学习领域和嵌入式设计领域，同时，TIDL 的给大部分算法工程师的使用带来了大量困难。开发者、研究人员在广泛使用 TF Lite，TF Lite 的模型不能直接在 TIDL Runtime 上运行，需要经过导入才能够被 TIDL 所支持。因此，对其流程进行了整理。本文后续 Demo 所使用的开发环境如表 1 所示：

表 4. Demo 环境

名称	内容
PSDKRA SDK	PROCESSOR-SDK-RTOS-J721E_08.01.00.11
TIDL 版本	tidl_j7_08_01_00_05
PC 开发环境	Ubuntu18.04
Python 版本	3.6
SOC	TDA4VM

如图 2 所示，TF Lite 模型生成及导入 TIDL 的流程（本文将以 MobileNet V2 为例介绍 TF Lite 模型的导入和推理步骤）：

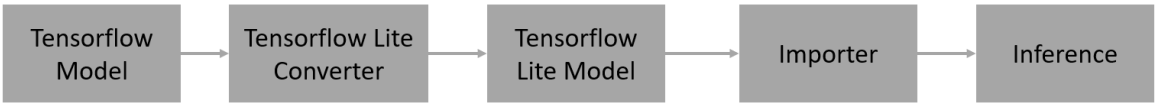

图 5. TFLITE 模型导入 TIDL 流程

1. Tensorflow Model: 基于 Tensorflow 训练框架，使用数据集进行训练，然后得到 Tensorflow 模型或者使用预训练的模型。
2. Tensorflow Lite Converter: TF 模型通常需要使用 Tensorflow Lite Converter 工具进行转换得到 TF Lite 模型。
3. Tensorflow Lite Model：TF Lite 模型的通常.tflite 结尾。
4. Importer：TIDL Runtime 提供了 Importer 工具, Importer 工具可以支持 TF Lite 后缀为 tflite 格式的模型导入，Importer 在内部执行各种处理，最后提供要在 TIDL Runtime 推理引擎上推理的模型。Importer 具体处理过程如图 3 所示：

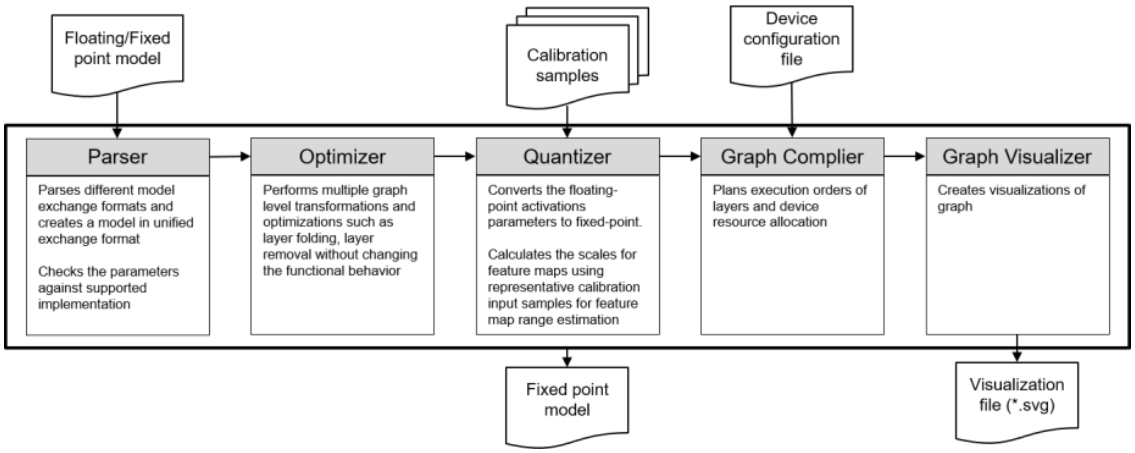

图 **6**. TIDL Importer 工具处理流程

5. Inference：TIDL 里面提供了应用程序可以对导入后的模型进行推理。该应用程序可以支持 PC（Emulation）和 Target（TDA4VM EVM）。PC_dsp_test_dl_algo.out 运行在 Ubuntu 上。TI_DEVICE_a72_test_dl_algo_host_rt.out 运行在 EVM 上。

3. 模型优化与导入

3.1. Tensorflow 模型优化

在 Ubuntu 上执行下面的脚本可以优化 Frozen Graph 模型。Frozen Graph 模型进行优化之前无法导入，因此需要先进行优化。优化采用 optimize_for_inference.py 脚本（该脚本在 tensorflow 的安装包里面，其路径：`~/.local/lib/python3.6/site-packages/tensorflow/python/tools/optimize_for_inference.py` ）。

注意

在执行下面的脚本之前，需要先安装 Tensorflow 在 Ubuntu，参考这里进行安装。

MobileNet V2 的 Frozen Model 可以从这里下载。optimize_for_inference.py 脚本主要根据输入的模型处理得到优化后的模型。“--input”是输入模型，输入 Frozen Graph 模型的路径。 “--output”是输出模型，输出优化后模型的路径。推荐使用下面命令里面的路径存放模型，方便模型的统一管理。这里我

们输入 mobilenet_v2_1.0_224_frozen.pb 模型，输出 mobilenet_v2_1.0_224_optimized.pb 模型 。mobilenet_v2_1.0_224_optimized.pb 模型将用于 TIDL 的导入模型的输入。

```
user@ubuntu-pc$ cd ~/.local/lib/python3.6/site-packages/tensorflow/python/too
ls/ && python3 optimize_for_inference.py \
    --input==${TIDL_INSTALL_PATH}/ti_dl/test/testvecs/models/public/tensorflo
w/mobilenetv2/ mobilenet_v2_1.0_224_frozen.pb \
    --output==${TIDL_INSTALL_PATH}/ti_dl/test/testvecs/models/public/tensorfl
ow/mobilenetv2/ mobilenet_v2_1.0_224_optimized.pb \
    --input_names="input" \
    --output_names="MobilenetV2/Predictions/Softmax"
```

3.2. 模型导入

Tensorflow 的模型和 TFLITE 模型的导入步骤相似。这里以 Tensorflow 模型的导入介绍模型导入的过程，对于 TFLITE、ONNX、Caffe 模型注意文件路径和配置也要做相应的适配。模型导入使用 tidl_model_import.out 工具，该工具导入输入的模型（TF Model （Proto）File），输出能够为 TIDL 所使用的网络模型和参数文件（TIDL Network File 和 TIDL IO Info File）,输入输出文件如下：

```
TF Model (Proto) File  : ../../test/testvecs/models/public/tensorflow/mobile
netv2/mobilenetv2.pb
TIDL Network File      : ../../test/testvecs/config/tidl_models/tensorflow/t
idl_net_mobilenetv2.bin
TIDL IO Info File      : ../../test/testvecs/config/tidl_models/tensorflow/t
idl_io_mobilenetv2
```

模型的导入需要模型导入配置文件，tidl_import_mobilenetv2.txt ，需要将其放在 ${TIDL_INSTALL_PATH}/ti_dl/test/testvecs/config/import/public/tensorflow/路径下。导入的配置文件里面指定了模型配置参数，如果对参数有不清楚的地方参考链接 TIDL-RT Import Configuration Parameters：

```
# modelType : 1 - TensorFlow (.pb files)
modelType         = 1
numParamBits      = 8
quantizationStyle  = 2
# inputNetfile: neural network file
inputNetFile       = "../../test/testvecs/models/public/tensorflow/mobilenetv2/mobil
enetv2.pb"
# outputNetFile: neural network file produced from TIDL-RT importer
outputNetFile       = "../../test/testvecs/config/tidl_models/tensorflow/tidl_net_mo
bilenetv2.bin"
# outputParamsFile: neural network input and output buffer descriptor file produced
from TIDL-RT importer
outputParamsFile       = "../../test/testvecs/config/tidl_models/tensorflow/tidl_io_
mobilenetv2"
```

```
inDataNorm  = 1
inMean = 128 128 128
#inScale =  0.01041667 0.01041667 0.01041667
inScale =  0.0078125 0.0078125 0.0078125
resizeWidth = 256
resizeHeight = 256
inWidth  = 224
inHeight = 224
inNumChannels = 3
inData = ../../test/testvecs/config/imageNet_sample_val_bg.txt
postProcType = 1
```

在 Ubuntu 命令行执行下面命令导入模型：

```
user@ubuntu-pc$ cd /home/fredy/startJacinto/sdks/ti-processor-sdk-rtos-j721e-evm-08_
01_00_11/tidl_j7_08_01_00_05/ti_dl/utils/tidlModelImport && \
./out/tidl_model_import.out /home/fredy/startJacinto/sdks/ti-processor-sdk-rtos-j721
e-evm-08_01_00_11/ tidl_j7_08_01_00_05/ti_dl/test/testvecs/config/import/public/tens
orflow/tidl_import_mobilenetv2.txt --numParamBits 15
```

Ubuntu 命令行执行上述命令输出的 log 如下，结尾输出 ALL MODEL CHECK PASSED 说明模型导入成功：

```
TF Model (Proto) File  : ../../test/testvecs/models/public/tensorflow/mobilenetv2/mo
bilenetv2.pb
TIDL Network File      : ../../test/testvecs/config/tidl_models/tensorflow/tidl_net_
mobilenetv2.bin
TIDL IO Info File      : ../../test/testvecs/config/tidl_models/tensorflow/tidl_io_m
obilenetv2

~~~~~Running TIDL in PC emulation mode to collect Activations range for each layer~~
~~~

Processing config file #0 : /home/fredy/startJacinto/sdks/ti-processor-sdk-rtos-j721
e-evm-08_01_00_11/tidl_j7_08_01_00_05/ti_dl/test/testvecs/config/tidl_models/tensorf
low/tidl_import_mobilenetv2.txt.qunat_stats_config.txt
 ---------------------- TIDL Process with REF_ONLY FLOW -----------------------

#    0 . .. T     315.50  .... ..... ... A :   896, 1.0000, 1.0000,   896 .... .....
#    1 . .. T     302.75  .... ..... ... A :   558, 0.5000, 0.5000,   875 .... .....
#    2 . .. T     303.01  .... ..... ... A :   443, 0.3333, 0.3333,   830 .... .....
#    3 . .. T     305.62  .... ..... ... A :   499, 0.2500, 0.2500,   752 .... .....
~~~~~Running TIDL in PC emulation mode to collect Activations range for each layer~~
~~~

Processing config file #0 : /home/fredy/startJacinto/sdks/ti-processor-sdk-rtos-j721
e-evm-08_01_00_11/tidl_j7_08_01_00_05/ti_dl/test/testvecs/config/tidl_models/tensorf
low/tidl_import_mobilenetv2.txt.qunat_stats_config.txt
```

```
 ----------------------- TIDL Process with REF_ONLY FLOW -----------------------

#    0 . .. T     615.74  .... ..... ... A :   896, 1.0000, 1.0000,   896 .... .....
#    1 . .. T     606.38  .... ..... ... A :   558, 0.5000, 0.5000,   875 .... .....
#    2 . .. T     605.94  .... ..... ... A :   443, 0.3333, 0.3333,   830 .... .....
#    3 . .. T     608.53  .... ..... ... A :   499, 0.2500, 0.2500,   752 .... .....


 *****************   Calibration iteration number 0 completed **********************
**
------------------ Network Compiler Traces -----------------------------
successful Memory allocation
**************************************************
**                 ALL MODEL CHECK PASSED          **
**************************************************
```

4. 模型 PC 推理

Tensorflow 的模型和 TFLITE 模型的推理步骤相似。这里以 Tensorflow 模型的导入介绍模型推理的过程，对于 TFLITE、ONNX、Caffe 模型注意文件路径和配置也要做相应的适配。完成模型的导入后，就可以使用 TIDL tidl_model_import.out 工具生成的模型进行推理。如图 3 所示，在这一章节中，我们将使用上一章节生成的模型文件（TIDL Network File 和 TIDL Network File ）验证模型的正确性，并将其部署在实际的应用中。

```
TIDL Network File     : ../../test/testvecs/config/tidl_models/tensorflow/tidl_net_mobilenetv2.bin
TIDL IO Info File     : ../../test/testvecs/config/tidl_models/tensorflow/tidl_io_mobilenetv2
```

图 7. 模型推理

部署的模型 MobileNet V2 用来进行图像分类，其训练数据集来自于 ImageNet, 其对应的标签可以在这里查询，如表 3 所示，从左到右是输入图像、推理结果、结果对应的标签：

表 5. Input Image

Input Image	Inference Result	Lable
testvecs/input/airshow.jpg	896	'warplane, military plane'
testvecs/input/ti_lindau_I00000.jpg	558	'flagpole, flagstaff'
testvecs/input/ti_lindau_000020.jpg	443	'bell cote, bell cot'
testvecs/input/0000000271.png	499	'cinema, movie theater, movie theatre, movie house, picture palace'

4.1. PC 模型推理

经过 TIDL tidl_model_import.out 工具导入 tensorflow 模型，我们可以快速利用 Ubuntu 的工具 PC_dsp_test_dl_algo.out 进行推理验证结果。 PC_dsp_test_dl_algo.out 需要配置文件 testvecs/config/infer/public/tensorflow/tidl_infer_mobilenetv2.txt（相关参数的解释，请参考这里）

```
inFileFormat    = 2 # Tensorflow
postProcType = 1
numFrames   = 1
netBinFile      = "testvecs/config/tidl_models/tensorflow/tidl_net_mobilenetv2.bin"
ioConfigFile    = "testvecs/config/tidl_models/tensorflow/tidl_io_mobilenetv21.bin"
inData  =   testvecs/config/classification_list_1.txt
outData =   testvecs/output/airshow_mobilenetv2_tf.bin
writeTraceLevel = 0
numFrames   = 1
```

如图 5 所示，本节使用 SDK TIDL 路径 testvecs/input/airshow.jpg 的图片作为感知的输入，模型采用 Tensorflow 经 TIDL 工具导入的 Tensorflow 模型，经过模型的推理，结果输出 896， 表示的是“warplane, military plane”，也就是识别结果是军用飞机。

图 **8**. 图片输入

在 Ubuntu 上，运行如下命令进行 TIDL 推理：

```
user@ubuntu-pc$  cd ti-processor-sdk-rtos-j721e-evm-08_01_00_11/tidl_j7_08_01_00_0
5/ti_dl/test/
user@ubuntu-pc$  ./PC_dsp_test_dl_algo.out
```

其 log 如下，显示的是推理的结果：

```
Processing config file #0 : testvecs/config/infer/public/tensorflow/tidl_infer_mob
ilenetv2.txt
 ----------------------- TIDL Process with REF_ONLY FLOW -----------------------

#    0 . .. T     623.55  .... ..... ... A :   896, 1.0000, 1.0000,   896 ....
 .....
```

上述结果显示了在 PC 推理所需的周期数，单位 Mega Cycles，假设 C7x 以 1 GHz 运行，则每个测试的每帧耗时和 FPS 可以计算为：

```
Time-taken-per-frame = (1000 / C7x CPU clock in MHz) x Number of mega cycles
FPS = 1 / Time-taken-per-frame
```

4.2. PC 模型部署

TDA4x 使用了 OpenVX 框架，可支持 PC 仿真。因此，应用可以基于 PC 验证。本节我们通过 SDK 中自带的 Image Classification Application 应用部署我们导入的 MobileNet V2 模型。

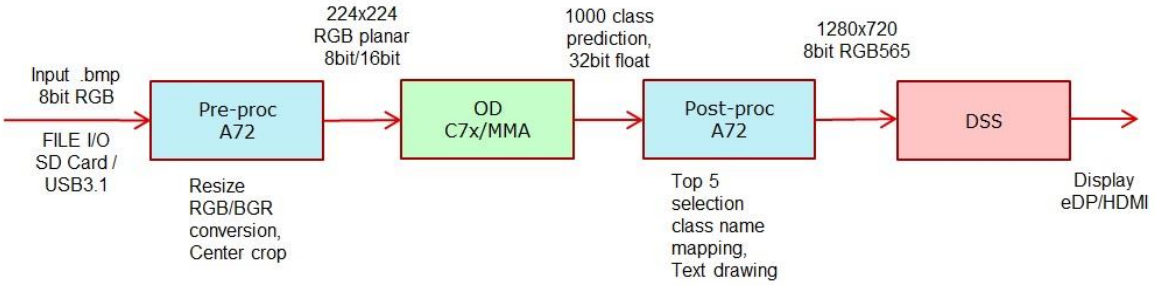

图 **9**. Image Classification Application

如图 6 所示，该应用程序使用了 TI 的深度学习库 TIDL 的 node 和 TI 的 MobileNet V1 网络以支持图像分类。应用程序接受指定文件中指定的分辨率小于 640x512 的 .bmp 文件列表。 Pre-proc 节点运行在 A72 上用于图像的预处理，功能是将图像大小调整为 224x224 ，经过预处理的图像，然后将其提供给在 C7x DSP 上运行的 TIDL。TIDL 的输出感知结果，经 Post-proc 节点（Post-proc 节点功能是将感知结果所对应的标签文字，图像合成一张大图）处理后，进行显示。PC Emulation Mode 下，不会直接显示在屏幕上，而是会存储成图片。

参考链接配置 PC Emulation Mode 环境。配置好环境后，检查配置文件 ti-processor-sdk-rtos-j721e-evm-08_01_00_11/vision_apps/apps/dl_demos/app_tidl/config/app_oc.cfg，该应用程序使用 app_oc.cfg 配置文件如下，主要配置了应用程序使用的模型、tidl_network_file_path、output_file_path 等：

```
# location of config
tidl_config_file_path   /ti/j7/workarea/tiovx/conformance_tests/test_data/tivx/tid
l_models/tidl_io_mobilenet_v1_1.bin

# location of network
tidl_network_file_path  /ti/j7/workarea/tiovx/conformance_tests/test_data/tivx/tid
l_models/tidl_net_mobilenet_v1.bin

# location of input files
# input_file_path         /ti/j7/workarea/tiovx/conformance_tests/test_data/psdkra
/app_tidl
```

```
# input_file_path       /ti/j7/workarea/tiovx/conformance_tests/test_data/psdkra/t
idl_demo_images
input_file_path     /ti/j7/workarea/tiovx/conformance_tests/test_data/psdkra/app_t
idl/
# list of the files to be executed. Input file names are provided explicitly in th
is file
# it should list all the file names present at 'input_file_path'. Each line in fil
e should just list only one file.
input_file_list         /ti/j7/workarea/tiovx/conformance_tests/test_data/psdkra/a
pp_tidl/names.txt

# path where ti_logo.bmp file is kept
ti_logo_file_path       /ti/j7/workarea/tiovx/conformance_tests/test_data/tivx/tid
l_models

# location of output
output_file_path        ./app_tidl_out

# display option, 0 - console output, 1 - display on screen (DSS)
display_option    0

# delay between 2 images in msecs
delay 0

# number of iterations to run
num_iterations 1

# 1, interactive mode, 0 no user interaction mode
is_interactive 0
```

在 Ubuntu 执行如下命令运行该应用程序：

注意

在执行下面的命令之前，需要先进行设置 PC Emulation Mode，并编译：

- 编辑 SDK/tiovx/build_flags.mak
  - 设置 BUILD_EMULATION_MODE=yes
  - 设置 BUILD_TARGET_MODE=no
- 编译 SDK： cd vision_apps && make sdk

```
user@ubuntu-pc$ cd ti-processor-sdk-rtos-j721e-evm-08_01_00_11/vision_apps/out/PC/
x86_64/LINUX/release/
user@ubuntu-pc$ ./vx_app_tidl --cfg /ti-processor-sdk-rtos-j721e-evm-08_01_00_11/v
ision_apps/apps/dl_demos/app_tidl/config/app_oc.cfg
```

部分 log 如下：

```
0.0s:  VX_ZONE_INIT:Enabled
 0.15s:  VX_ZONE_ERROR:Enabled
 0.16s:  VX_ZONE_WARNING:Enabled
 0.364s:  VX_ZONE_INIT:[tivxInit:178] Initialization Done !!!
Turning display option off ...
app_tidl: Init ...
app_tidl: Reading config file /ti/j7/workarea/tiovx/conformance_tests/test_data/ti
vx/tidl_models/tidl_io_mobilenet_v1_1.bin ...
app_tidl: Reading config file /ti/j7/workarea/tiovx/conformance_tests/test_data/ti
vx/tidl_models/tidl_io_mobilenet_v1_1.bin ... Done. 37256 bytes
app_tidl: Tensors, input = 1, output = 1
app_tidl: Reading network file /ti/j7/workarea/tiovx/conformance_tests/test_data/t
ivx/tidl_models/tidl_net_mobilenet_v1.bin ...
app_tidl: Reading network file /ti/j7/workarea/tiovx/conformance_tests/test_data/t
ivx/tidl_models/tidl_net_mobilenet_v1.bin ... Done. 5067792 bytes
app_tidl: Init ... Done.
app_tidl: Creating graph ...
app_tidl: Creating graph ... Done.
app_tidl: Verifying graph ...
app_tidl: Verifying graph ... Done.
network file: /ti/j7/workarea/tiovx/conformance_tests/test_data/tivx/tidl_models/t
idl_net_mobilenet_v1.bin
config  file: /ti/j7/workarea/tiovx/conformance_tests/test_data/tivx/tidl_models/t
idl_io_mobilenet_v1_1.bin
Iteration 0 of 1 ...
Classifying input 0001.bmp ...
app_tidl: Reading input file /ti/j7/workarea/tiovx/conformance_tests/test_data/psd
kra/app_tidl//0001.bmp ...  input_sizes[0] = 225, dim = 224 padL = 1 padR = 0
 input_sizes[1] = 227, dim = 224 padT = 1 padB = 2
 input_sizes[2] = 3, dim = 3
app_tidl: Reading bmp file ...
app_tidl: Reading bmp file ... Done.
app_tidl: Image Pre processing for image of size 620 x 462 (pitch = 1860 bytes)...
app_tidl: Deinterleaving data ...
app_tidl: Resizing image ...
app_tidl: Rearranging data ...
app_tidl: Image Pre processing ... Done.
Done!
app_tidl: Running Graph ... Done!
app_tidl: Finding top-5 ...
app_tidl: Image classification Top-5 results:
app_tidl:  tusker, class-id: 101, score: 0.333128
app_tidl:  Indian elephant, Elephas maximus, class-id: 385, score: 0.333128
app_tidl:  African elephant, Loxodonta africana, class-id: 386, score: 0.333128
app_tidl:  triceratops, class-id: 51, score: 0.000474
```

```
app_tidl:  water buffalo, water ox, Asiatic buffalo, Bubalus bubalis, class-id: 34
6, score: 0.000053
app_tidl: Finding top-5 ... Done
app_tidl: Showing output ... Done.
Classifying input 0001.bmp ...Done!
Classifying input 0002.bmp ...
app_tidl: Reading input file /ti/j7/workarea/tiovx/conformance_tests/test_data/psd
kra/app_tidl//0002.bmp ...  input_sizes[0] = 225, dim = 224 padL = 1 padR = 0
……
```

运行结果保存在路径： ti-processor-sdk-rtos-j721e-evm-08_01_00_11/vision_apps/out/PC/x86_64/LINUX/release/app_tidl_out： 如下图 7 所示，选取了前面四张图片，每张图片左边是感知的图片，图片右边显示了感知的结果。根据图片显示的内容我们可以判断模型的正确性。

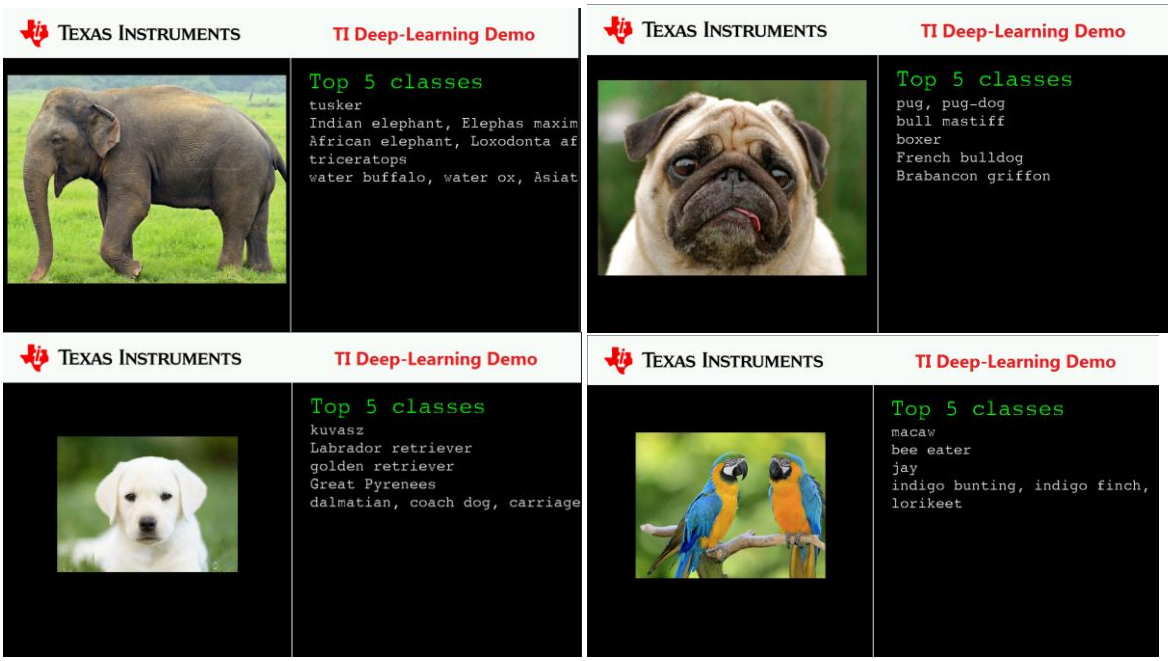

图 **10**. PC Image Classification Application

5. 模型 EVM 推理

Tensorflow 的模型和 TFLITE 模型的 EVM 推理步骤相似。这里以 Tensorflow 模型的导入介绍模型 EVM 模型的推理过程，对于 TFLITE、ONNX、Caffe 模型注意文件路径和配置也要做相应的适配。模型在 PC 验证后，最终可以部署在 EVM 上进行性能和结果测试。如果觉得 PC 验证会浪费时间，可以省略第四章的内容，完成模型的导入后，就可以直接在 EVM 上验证。部分内容已经在第四章节做了介绍，本章节不做具体介绍，对于有疑问的地方，可以先阅读第四章节的内容。本章节也分为两个部分，EVM 模型推理和 EVM 模型部署。

5.1. EVM 模型推理

经过 TIDL tidl_model_import.out 工具导入的 tensorflow 模型，在 EVM 上，使用 TI_DEVICE_a72_test_dl_algo_host_rt.out 工具进行推理测试。该工具不仅可以测试使用模型的正确性，还可以测试真实的帧率。 按如下步骤进行测试：

1. 参考链接制作 SD 卡，执行如下命令将 TIDL 的应用拷贝到 SD Card

```
user@ubuntu-pc$ cd ${PSDKRA_PATH}/vision_apps
user@ubuntu-pc$ make linux_fs_install_sd
```

2. 将 SD 卡插入 EVM，然后启动 EVM 板到命令行，执行如下命令：

```
root@ j7-evm:~# cd /opt/vision_apps
root@ j7-evm:~# source ./vision_apps_init.sh
root@ j7-evm:~# cd /opt/tidl_test
root@ j7-evm:/opt/tidl_test# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib
root@j7-evm: /opt/tidl_test# ./TI_DEVICE_a72_test_dl_algo_host_rt.out
```

3. Log 输出如下：

```
root@j7-evm:/opt/tidl_test# ./TI_DEVICE_a72_test_dl_algo_host_rt.out
Processing config file #0 :
testvecs/config/infer/public/tensorflow/tidl_infer_mobilenetv2.txt
APP: Init ... !!!
MEM: Init ... !!!
MEM: Initialized DMA HEAP (fd=6) !!!
MEM: Init ... Done !!!
IPC: Init ... !!!
IPC: Init ... Done !!!
REMOTE_SERVICE: Init ... !!!
REMOTE_SERVICE: Init ... Done !!!
3261.007805 s: GTC Frequency = 200 MHz
APP: Init ... Done !!!
3261.007880 s: VX_ZONE_INIT:Enabled
3261.007888 s: VX_ZONE_ERROR:Enabled
3261.007893 s: VX_ZONE_WARNING:Enabled
3261.010166 s: VX_ZONE_INIT:[tivxInitLocal:130] Initialization Done !!!
3261.010425 s: VX_ZONE_INIT:[tivxHostInitLocal:86] Initialization Done for
HOST !!!
---------------------- TIDL Process with TARGET DATA FLOW -------------------
-----
# NETWORK_EXECUTION_TIME = 3.18 (in ms, c7x @1GHz) with DDR_BANDWIDTH (Read +
Write) = 0.00, 0.00, 0.00 (in Mega Bytes/frame) ... A : 896, 1.0000, 1.0000,
896 .... ..... 3261.07537!
```

```
3261.076723 s: VX_ZONE_INIT:[tivxDeInitLocal:193] De-Initialization Done !!!
APP: Deinit ... !!!
REMOTE_SERVICE: Deinit ... !!!
REMOTE_SERVICE: Deinit ... Done !!!
IPC: Deinit ... !!!
IPC: DeInit ... Done !!!
MEM: Deinit ... !!!
MEM: Alloc's: 7 alloc's of 8511362 bytes
MEM: Free's : 7 free's of 8511362 bytes
MEM: Open's : 0 allocs of 0 bytes
MEM: Deinit ... Done !!!
APP: Deinit ... Done !!!
```

Log 上述结果显示了推理的结果是 896， 由表 3 我们知道识别结果为军用飞机以及在 EVM 上推理一帧数据所花费的时间，假设 C7x 以 1 GHz 运行，则每个测试的每帧耗时和 FPS 可以计算为：

```
FPS = 1 / NETWORK_EXECUTION_TIME
```

经 TI_DEVICE_a72_test_dl_algo_host_rt.out 处理后，输出图像存储在/opt/tidl_test/testvecs/output,可插入 SD 卡到 PC 确认其结果。

5.2. EVM 模型部署

本节我们基于图 5 所示的 Image Classification Application 应用部署我们导入的 MobileNet V2 模型。SDK 默认的应用使用了 mobilenet_v1：

```
Network file  : /opt/vision_apps/test/testvecs/config/tidl_models/tensorflow/tidl_
net_mobilenet_v.bin
Config  file  : /opt/vision_apps/test/testvecs/config/tidl_models/tensorflow/tidl_
io_mobilenet_v1_1
```

这里我们使用 TIDL tidl_model_import.out 导入的 mobilenetv2 模型(注意将模型文件拷贝到 SD 卡/home/root 目录)：

```
Network file  : /home/root/tidl_net_mobilenetv2.bin
Config  file  : /home/root/tidl_io_mobilenetv21.bin
```

程序运行之前检查 app_oc.cfg（/opt/vision_apps）配置，配置应用程序的相关配置：

```
# location of config
tidl_config_file_path  /home/root/tidl_io_mobilenetv21.bin
# location of network
tidl_network_file_path /home/root/tidl_net_mobilenetv2.bin
# location of input files
input_file_path /opt/vision_apps/test_data/psdkra/app_tidl
# list of the files to be executed. Input file names are provided explicitly in th
is file
```

```
# it should list all the file names present at 'input_file_path'. Each line in fil
e should just list only one file.
input_file_list /opt/vision_apps/test_data/psdkra/app_tidl/names.txt
# path where ti_logo.bmp file is kept
ti_logo_file_path /opt/vision_apps/test_data/tivx/tidl_models
# location of output
output_file_path ./app_tidl_out
# display option, 0 - console output, 1 - display on screen (DSS)
display_option 1
# delay between 2 images in msecs
delay 1000
# number of iterations to run
num_iterations 1000000
# 1, interactive mode, 0 no user interaction mode
is_interactive 1
```

注意

在执行 run_app_tidl.sh 脚本之前，请注意配置 app_oc.cfg 里面的模型。

运行应用程序验证结果，步骤如下：

1. 将 SD 卡插入 EVM，然后启动 EVM 板到命令行，执行如下命令启动应用程序：

```
root@j7-evm:~# cd /opt/vision_apps
root@j7-evm:~# source ./vision_apps_init.sh
root@j7-evm:~# ./run_app_tidl.sh
```

2. Log 如下：log 显示我们使用的模型文件

```
root@j7-evm:/opt/vision_apps# ./run_app_tidl.sh
APP: Init ... !!!
MEM: Init ... !!!
MEM: Initialized DMA HEAP (fd=4) !!!
MEM: Init ... Done !!!
IPC: Init ... !!!
IPC: Init ... Done !!!
REMOTE_SERVICE: Init ... !!!
REMOTE_SERVICE: Init ... Done !!!
7537.376927 s: GTC Frequency = 200 MHz
APP: Init ... Done !!!
7537.383500 s: VX_ZONE_INIT:Enabled
7537.383538 s: VX_ZONE_ERROR:Enabled
7537.383544 s: VX_ZONE_WARNING:Enabled
7537.385919 s: VX_ZONE_INIT:[tivxInitLocal:130] Initialization Done !!!
7537.387831 s: VX_ZONE_INIT:[tivxHostInitLocal:86] Initialization Done for
HOST !!!
network file: /home/root/tidl_net_mobilenetv2.bin
```

```
config file: /home/root/tidl_io_mobilenetv21.bin
Iteration 0 of 1000000 ...
================================
Demo : TIDL Object Classification
================================
p: Print performance statistics
x: Exit
```

3. 在 EVM 连接的显示器上，我们看到实时的显示结果。屏幕上 1s 切换一张图片，前面 4 张如下，应用程序实时处理输入的图片，进行推理并将结果显示在屏幕上。

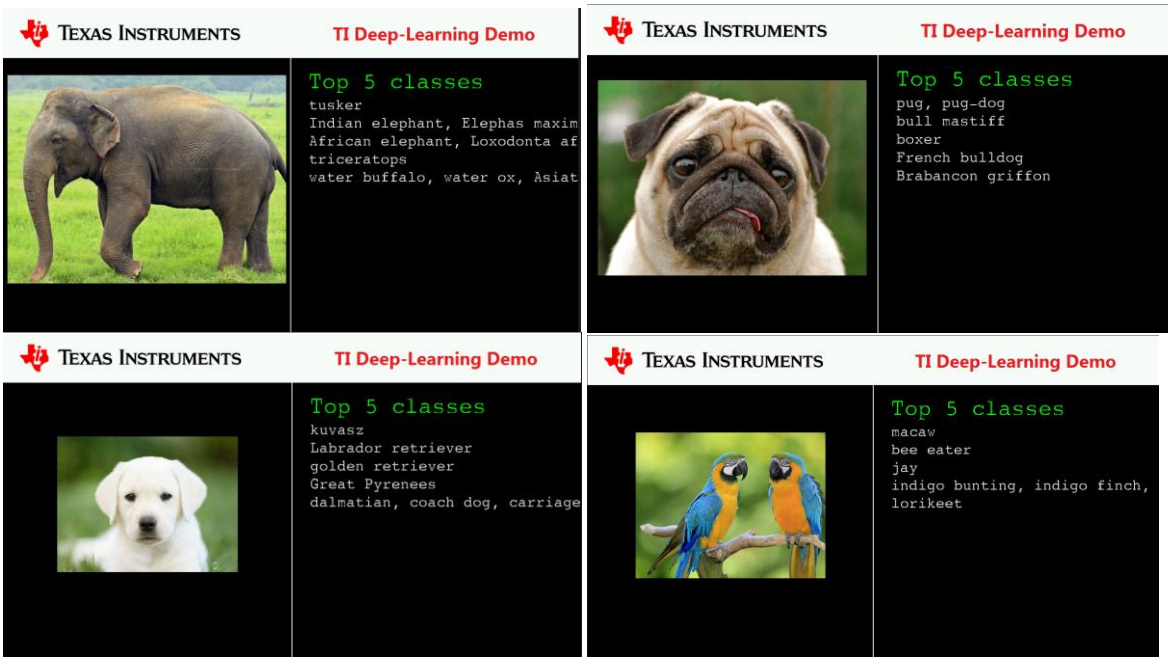

图 **11**. EVM Image Classification Application

6. FAQ

6.1. SDK 中使用的 Tensorflow 版本及推荐的 Tensorflow 版本？

SDK 里面验证使用的 Tensorflow 版本参考链接说明，目前最新的 SDK 中，使用的 Tensorflow - 1.12 验证模型，新版本的模型也可以工作，因为卷积、池化等基本操作不会改变。

由于 Tensorflow 2.x 不支持 Frozen Graph，且 Tensorflow 2.x 开始使用 Keras 模型，导出是 savedmodel 格式或者 h5 格式。网上也有一些方法可以使用 Tensorflow 2.x 生成 Frozen Graph 模型。在导入的过程中也可能遇到以一些问题，比如，算子版本过高、算子新增参数不支持、算子本身不支持等情况。因此，推荐使用 Tensorflow 1.x，不建议使用 Tensorflow 2.x 的版本。如果要使用 Tensorflow 2.x 推荐使用 Tensorflow lite 模型。

6.2. 如何得到 Frozen Graph 模型？

得到 Tensorflow（.pb）格式的模型，需要先转换为 Frozen Model。 Tensorflow 1.x 转换 Frozen Graph 模型请参考这里。Tensorflow 2.x 转换 Frozen Graph 模型请参考这里。参考脚本 Example script to Reference output from TensorFlow for Frozen Graph。

6.3. 如何优化 Tensorflow 模型？

在 Ubuntu 上执行下面的脚本 optimize_for_inference.py 可以优化 Frozen Graph 模型。Frozen Graph 模型进行优化之前无法导入，因此需要先进行优化。优化采用 optimize_for_inference.py 脚本（该脚本在 tensorflow 的安装包里面，其路径：~/.local/lib/python3.6/site-packages/tensorflow/python/tools/ optimize_for_inference.py ）。

6.4. 有哪些 Tensorflow 模型在 TIDL 验证过？

SDK Release 的包里面已经提供了如下模型的 import 和 Inference 的配置文件。这些模型可以在 SDK 里面进行验证，也可以作为参考。

6.5. TIDL Importer 工具导入 Tensorflow 模型需要注意哪些方面？

如图 2 所示，Tensorflow 模型导入 TIDL 需要先转换再优化，得到优化后的模型才可以进行导入。TIDL Importer 工具输入配置文件路径在：ti-processor-sdk-rtos-j721e-evm-08_01_00_11/tidl_j7_08_01_00_05/ti_dl/test/testvecs/config/import/public/tensorflow/tidl_import_xxx.txt。对于 import 配置文件注意检查如下内容：

1. 导入的 modelType： Tensorflow 模型配置为 1.
2. inputNetfile/outputNetFile: 检查输入/输出模型文件的路径。
3. inData： 输入数据的配置，通常自己的模型，需要调整输入图片。
4. inWidth/inHeight/resizeWidth/resezeHeight ：配置图片输入的 size 及调整后的 size。
5. inNumChannels ： 输入图片通道数。

6.6. *TIDL 如何打印 log 信息？*

不论是 Import 模型的时候，还是 inference 的时候，难免会遇到问题，当遇到问题的时候，怎么样才能输出更多的调试信息呢？ TIDL 提供了两个标志变量 writeTraceLevel 和 debugTraceLevel 用来获取更多的信息。

debugTraceLevel 可以打印更多的输出信息，便于追踪 import 和 inference 的过程。默认配置是 0，可支持 1、2 配置。数字越大，表明输出的信息越多。

writeTraceLevel 可以输出每一层的信息到文件，便于模型逐层比较。默认配置是 0，没有输出，可支持 1、2、3 配置：1- Fixed Point , 2- Padded Fixed Point, 3 - Floating point。

7. 参考

1. https://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-jacinto7/08_01_00_11/exports/docs/tidl_j7_08_01_00_05/ti_dl/docs/user_guide_html/usergroup0.html
2. https://www.tensorflow.org/
3. https://leimao.github.io/blog/Save-Load-Inference-From-TF2-Frozen-Graph/
4. https://leimao.github.io/blog/Save-Load-Inference-From-TF-Frozen-Graph/

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265