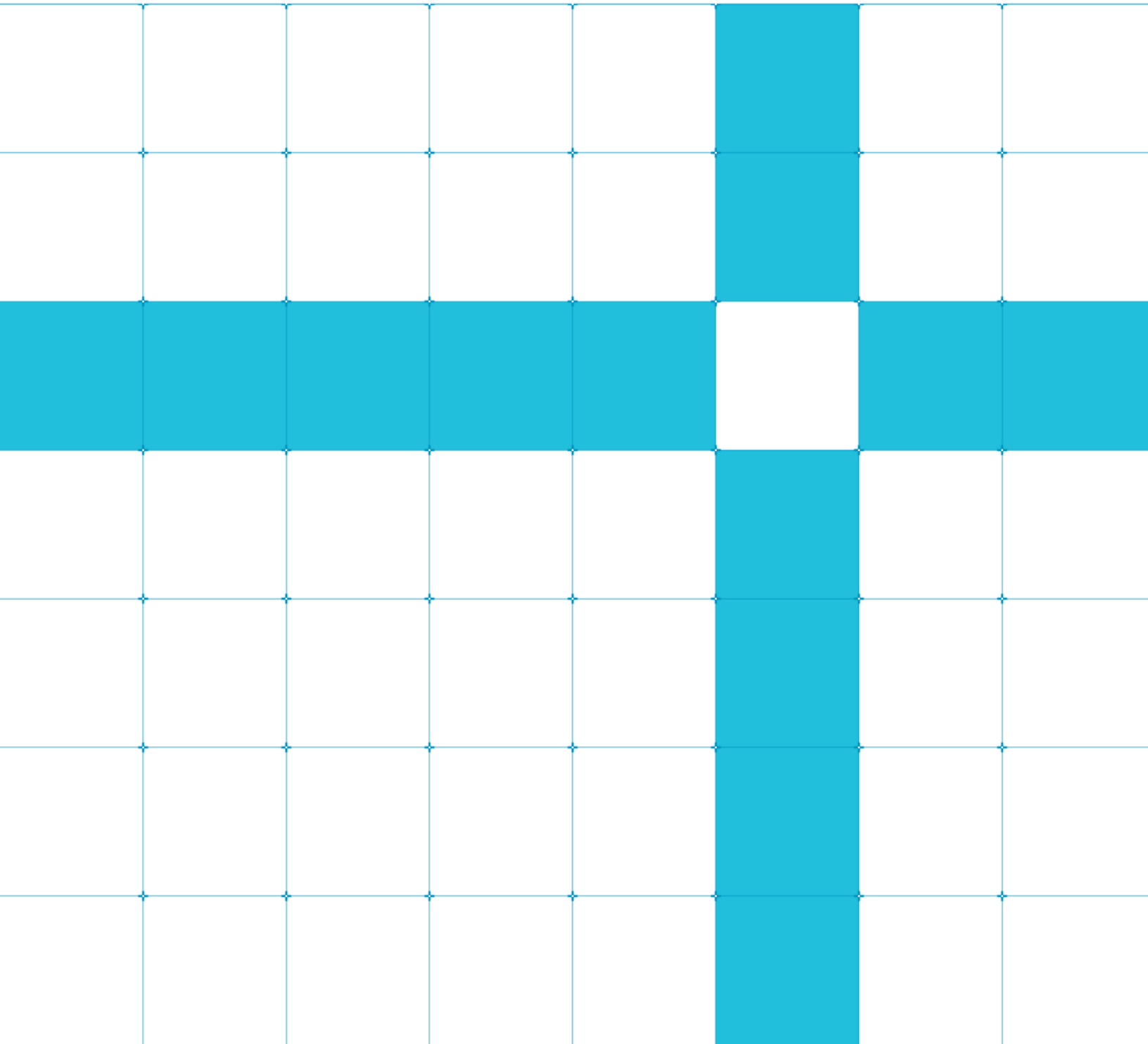




Deploying a quantized TensorFlow Lite MobileNet V1 model using the Arm NN SDK

Version 1.0



Deploying a quantized TensorFlow Lite MobileNet V1 model using the Arm NN SDK

Copyright © 2019 Arm Limited (or its affiliates). All rights reserved.

Release Information

Document History

Version	Date	Confidentiality	Change
1.0	12/08/2019	Non-Confidential	First release.

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2019 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

www.arm.com

Contents

1 Overview	5
2 Before you begin	6
3 Write your application	7
3.1. Load the model output labels.....	7
3.2. Load and pre-process an input image for the quantized model.....	7
3.3. Prepare the output tensor.....	8
3.4. Import a graph.....	8
3.5. Optimize the model and load it onto a compute device	9
3.6. Run a graph on a compute device	9
3.7. Interpret and report the output	10
4 Build your application	11
5 Run your application	12
5.1. Transfer the application, Arm NN SDK, data files, and model file to an Android device	12
5.2. Run the application.....	12
6 Next steps	13

1 Overview

Using an example, this guide shows how we develop an application that classifies images using a TensorFlow Lite quantized Mobilenet V1 model. The guide also covers how we deploy the model using the open-source [Arm NN SDK](#). We will be able to use the knowledge from this guide to run our own models in our own applications on [Arm Cortex CPUs](#) and [Mali GPUs](#).

2 Before you begin

This guide assumes:

- You have built version 19.08 or newer of the Arm NN SDK for Android and installed the Boost library. For more information on how to build the Arm NN SDK see, [Configuring the Arm NN SDK build environment for TensorFlow Lite](#).
- You have access to the complete [example application](#) with source code, data, and model.
- You have a quantized Mobilenet V1 model and its output labels. This model is a `.tflite` file.
- You have images that are in the RGB format with channel values between 0 and 255.

3 Write your application

There are specific steps that you carry out to deploy and use a TensorFlow Lite quantized model with the Arm NN SDK.

You must do the following:

- Load the model output labels.
- Load and pre-process an input image for the quantized model.
- Prepare the output tensor.
- Import a graph.
- Optimize the model and load it onto a compute device.
- Run a graph on a device.
- Interpret and report the output.

Using the example code, this guide walks you through each step.

3.1. Load the model output labels

You must use the model output labels to interpret the outputs of the model. These labels are usually in a text file the model creator or distributor provides. In this file, each line contains the label or labels corresponding to each output node. You can use the utility function `LoadModelOutputLabels` that the `model_output_labels_loader.hpp` file defines to load the labels. The following example code loads the labels using the `LoadModelOutputLabels` function:

```
const std::vector<CategoryNames> modelOutputLabels =  
    LoadModelOutputLabels(programOptions.modelOutputLabelsPath);
```

3.2. Load and pre-process an input image for the quantized model

You must pre-process images before the model can use them as inputs. The pre-processing method that you use depends on the framework, model, or model data type you use.

For the purposes of this guide, you must do the following to pre-process the image:

1. Resize the input images to match the dimensions of the input tensor of the model. In this example, the MobileNet V1 model accepts 224x224 input images.
2. For floating-point models, you must scale the input image values to a range of -1 to 1. For example, if the values of the input image are between 0 to 255, you must divide the image values by 127.5 and subtract 1. For integer quantized models, the image values must be within the 0 to 255 range. Given that these image values are already within the correct range, you do not need to scale the input images of integer quantized models.
3. Use the C++ operation `static_cast` to convert the input image values from floating point to 8-bit unsigned integer type.

You can pre-process images offline with your own tools. However, Arm NN comes with the `PrepareImageTensor` utility function that handles pre-processing.

Note: You can also use the `ImageTensorGenerator` as an offline tool to use `static_cast` to convert images to input tensors. Refer to the README in the folder of the tool for more information.

The following example code loads and pre-processes an image the command-line option `imagePath` specifies:

```
// Load and preprocess input image  
const std::vector<TContainer> inputDataContainers =  
{ PrepareImageTensor<uint8_t>(programOptions.imagePath,  
    inputTensorWidth, inputTensorHeight,  
    normParams,
```

```
inputTensorBatchSize,  
inputTensorDataLayout) } ;
```

As they are specific to the MobileNet V1 model, you must specify the following in your code:

- inputTensorWidth
- inputTensorHeight
- inputTensorBatchSize
- inputTensorDataLayout
- inputName
- outputName

Note: The `inputName` and `outputName` of your specific model can differ from the names in the example code. Ensure that you specify the correct `inputName` and `outputName`.

The following is the example code:

```
const std::string inputName = "input";  
const std::string outputName = "MobilenetV1/Predictions/Reshape_1";  
const unsigned int inputTensorWidth = 224;  
const unsigned int inputTensorHeight = 224;  
const unsigned int inputTensorBatchSize = 1;  
const armnn::DataLayout inputTensorDataLayout = armnn::DataLayout::NHWC;
```

The `normParams` variable determines how the input image is normalized. The following pseudocode shows how the image pre-processor within the `PrepareImageTensor` utility function calculates normalized image values:

```
out = ((in / scale) - mean) / stddev
```

Therefore, you specify the `normParams` variable as the following example code shows:

```
// Prepare image normalization parameters  
normParams.scale = 1.0;  
normParams.mean = { 0.0, 0.0, 0.0 };  
normParams.stddev = { 1.0, 1.0, 1.0 };
```

3.3. Prepare the output tensor

You must prepare a container to receive the output of the model.

The following example code prepares a container to receive the output of the model:

```
// Output tensor size is equal to the number of model output labels  
const unsigned int outputNumElements = modelOutputLabels.size();  
std::vector<TContainer> outputDataContainers = { std::vector<uint8_t>(outputNumElements) };
```

3.4. Import a graph

You must import the TensorFlow Lite graph that you use. The Arm NN SDK provides parsers for reading graphs from TensorFlow Lite.

The SDK supports TensorFlow Lite graphs in text and binary ProtoBuf formats. To import the graph, you must:

1. Load the model.

2. Bind the input and output points of its graph.

The following example code imports the graph:

```
// Import the TensorFlowLite model.
using IParser = armnnTfLiteParser::ITfLiteParser;
auto armnnparser(IParser::Create());
armnn::INetworkPtr network = armnnparser->CreateNetworkFrom
BinaryFile(programOptions.modelPath.c_str());
```

After this step, the code is common regardless of the framework that you started with.

The following example code binds the input and output tensors to the data and selects the loaded network identifier:

```
// Find the binding points for the input and output nodes
using BindingPointInfo = armnnTfLiteParser::BindingPointInfo;
const std::vector<BindingPointInfo> inputBindings = { armnnparser->GetNetworkInputBindingInfo(0,
inputName) };
const std::vector<BindingPointInfo> outputBindings = { armnnparser->GetNetworkOutputBindingInfo(0,
outputName) };
```

Note: You define the `inputName` and `outputName` strings at the beginning of the code.

3.5. Optimize the model and load it onto a compute device

You must optimize your network and load it onto a compute device. The Arm NN SDK supports optimized execution on multiple CPU and GPU devices. Before you start executing a graph, you must select the appropriate device context and optimize the graph for that device.

To select an Arm Mali GPU for use, you must specify `-c GpuAcc` in the command line.

The following example code optimizes and loads your network onto a compute device:

```
// Create a runtime and optimize the network for a specific compute device,
// e.g. CpuAcc, GpuAcc
armnn::IRuntime::CreationOptions options;
armnn::IRuntimePtr runtime(armnn::IRuntime::Create(options));
armnn::IOptimizedNetworkPtr optimizedNet = armnn::Optimize(*network, programOptions.computeDevice,
runtime->GetDeviceSpec());

// Load the optimized network onto the device
armnn::NetworkId networkId;
runtime->LoadNetwork(networkId, std::move(optimizedNet));
```

3.6. Run a graph on a compute device

A compute device performs inference using the `EnqueueWorkload()` function of the context.

The following example code runs a single inference on the test image:

```
runtime->EnqueueWorkload(networkId,
    armnnUtils::MakeInputTensors(inputBindings, inputDataContainers),
    armnnUtils::MakeOutputTensors(outputBindings, outputDataContainers));
```

3.7. Interpret and report the output

The output of the model is a tensor of the same size as the number of output labels. The size of the ImageNet tensor is 1001. You must interpret each value as the probability of the input image being classified as the corresponding label. To find the label that the model predicts most confidently, you must find the label of the output node with the highest output value.

The `std::distance()` function in the following example code is used to find the index of the largest element in the output. This function is equivalent to the `argmax()` function from the NumPy library.

```
std::vector<uint8_t> output = boost::get<std::vector<uint8_t>>(outputDataContainers[0]);

size_t labelInd = std::distance(output.begin(), std::max_element(output.begin(), output.end()));
std::cout << "Prediction: ";
for (const auto& label : modelOutputLabels[labelInd])
{
    std::cout << label << ", ";
}
std::cout << std::endl;
```

4 Build your application

You must link your application with the Arm NN SDK library, the TensorFlow Lite parsing library, the inference test library that is part of Arm NN SDK, and the Boost library.

The following command compiles and links your application with the necessary libraries:

```
$(CXX) -DARMNN_TF_LITE_PARSER \  
-I$(ARMNN_ROOT)/include \  
-I$(ARMNN_ROOT)/src/backends \  
-I$(ARMNN_ROOT)/src/armnnUtils \  
-I$(ARMNN_ROOT)/tests \  
-I$(BOOST_ROOT)/include \  
-Wall -O3 -std=c++14 -fPIE mobilenetv1_quant_tflite.cpp -o mobilenetv1_quant_tflite -pie \  
-L$(ARMNN_BUILD)-L$(ARMNN_BUILD)/tests \  
-L$(BOOST_ROOT)/lib \  
-larmnn -larmnnTfLiteParser -linferenceTest \  
-lboost_system -lboost_filesystem -lboost_program_options
```

The commands are specified in the Makefile of the example code. Alternatively, you can build the example code with the `make` command. Refer to the README document from the example code for more information.

A successful build of your application produces the `mobilenetv1_quant_tflite` executable file.

Note: The example code runs on the Android platform. Therefore if you are on an X86 host, `$(CXX)` must specify the same cross-platform compiler that you use to build ArmNN SDK.

5 Run your application

5.1. Transfer the application, Arm NN SDK, data files, and model file to an Android device

Before you run the application, you must transfer specific files to your Android device.

The following example code transfers the application, Arm NN SDK, data files, and model file to an Android device:

```
adb push libarmnn.so /data/temp/app
adb push libarmnnTfLiteParser.so /data/temp/app
adb push mobilenetv1_quant_tflite /data/temp/app
adb push mobilenet_v1_1.0_224_quant.tflite /data/temp/app/model
adb push model_output_labels.txt /data/temp/app/model
```

Note: The example code makes the following assumptions:

- You put the application and ArmNN SDK in the `/data/temp/app` directory.
- You put the model and model output labels files in the `/data/temp/app/model` directory.
- The image data resides in the `/data/temp/app/images` directory.

5.2. Run the application

To run the application, run the following commands on an Android device:

```
cd /data/temp/app
LD_LIBRARY_PATH=. ./mobilenetv1_quant_tflite -m models/mobilenet_v1_1.0_224_quant.tflite -d
images/{image_name} -p model/model_output_labels.txt
```

The following is example output:

```
ArmNN v20190500

Running network...
Prediction: sea snake,
Ran successfully!
```

6 Next steps

This guide covers the steps to develop an image classification application using a quantized TensorFlow Lite Mobilenet V1 model and the Arm NN SDK. You can use the example code this guide provides as a starting point to develop your own application using a quantized TensorFlow Lite MobileNet V1 model.