Image Classification with MobilenetV2, Arm NN, and TensorFlow Lite Delegate pre-built binaries

Version 21.11

Tutorial

Non-Confidential

Issue 01

102561_2111_01_en

Image Classification with MobilenetV2, Arm NN, and TensorFlow Lite Delegate pre-built binaries

Tutorial

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	21 July 2021	Non-Confidential	First release for version 1.01
2108-01	17 August 2021	Non-Confidential	First release for version 21.08
2108-02	18 October 2021	Non-Confidential	Second release for version 21.08
2111-01	22 November 2021	Non-Confidential	First release for version 21.11

Proprietary Notice

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

developer.arm.com

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1 Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm® Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace italic	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
monospace bold	Denotes language keywords when used outside example code.
monospace underline	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
Caution	This represents a recommendation which, if not followed, might lead to system failure or damage.
Warning	This represents a requirement for the system that, if not followed, might result in system failure or damage.
Danger	This represents a requirement for the system that, if not followed, will result in system failure or damage.

Convention	Use
Note	This represents an important piece of information that needs your attention.
Tip	This represents a useful tip that might make it easier, better or faster to perform a task.
Remember	This is a reminder of something important that relates to the information you are reading.

1.2 Additional reading

This document contains information that is specific to this product. See the following documents for other relevant information:

Table 1-2: Arm publications

Document Name	Document ID	Licensee only
None	-	-

1.3 Feedback

Arm welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title Image Classification with MobilenetV2, Arm NN, and TensorFlow Lite Delegate pre-built binaries Tutorial.
- The number 102561_2111_01_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

1.4 Other information

See the Arm website for other relevant information.

- Arm® Developer.
- Arm® Documentation.
- Technical Support.
- Arm® Glossary.

2 Overview

This guide reviews a sample application for image classification using the Arm NN TensorFlow Lite Delegate (Arm NN TfLite Delegate). The guide explains how to build the application and deploy it onto your device and goes into a code deep dive.

This guide is suitable for both beginners and experienced developers. We can use the early sections of the guide to get an app up and running in minutes. For more experienced developers, we also go further into detail later in the guide in the Code deep dive section.

2.1 Before you begin

This guide requires installing the Arm NN TfLite Delegate on your device. In this guide, we show steps on how to use our pre-built binaries available on GitHub. To use these binaries, you require a 64-bit Arm device.

We provide installation steps for different devices in the Device-specific installation section.

2.2 Arm NN TensorFlow Lite Delegate

The Arm NN TfLite Delegate is the latest addition to the Arm NN library. It is a library for accelerating certain TensorFlow Lite (TfLite) operators on Arm hardware. The Arm NN TfLite Delegate uses the delegate system of TfLite delegate system. As a result, TfLite delegates any operations Arm NN supports to Arm NN during execution.

There are two advantages of using the Arm NN TfLite Delegate over the Arm NN TfLite Parser.

The first advantage is that the number of supported operations is far greater. The Arm NN TfLite Parser only supports a model if Arm NN supports all operations in the model. Conversely when using the Arm NN TfLite Delegate, any operations not supported by Arm NN are delegated to TensorFlow Lite. This delegation means Arm NN TfLite Delegate can execute all TfLite models, and accelerates any operations that Arm NN supports.

The second advantage is that the Arm NN TfLite Delegate can be easily integrated into your Python TfLite projects. Arm NN TfLite Delegate only requires an extra line to load the delegate into the interpreter.

2.3 Image classification

Image classification is one of the most popular deep learning problems. Because of the vast number of datasets available, neural networks have been able to excel in this field.

There have also been many advancements for edge devices. One key model is the MobileNet architecture. This architecture uses depth-wise separable convolution layers to create a lightweight image classification model that performs efficiently on mobile and embedded devices.

This guide shows how to use one of the latest versions of MobileNet, MobileNetv2, with the Arm NN TfLite Delegate.

3 Device-specific installation

The easiest way to get started with the Arm NN TfLite Delegate on your device is using the pre-built binaries in the Arm NN repository. We provide pre-built binaries for 64-bit Arm devices and Android devices.

3.1 Install on Raspberry Pi

The steps in this section cover installing the Arm NN TfLite Delegate on the Raspberry Pi.

Procedure

1. Install Ubuntu. See installation instructions for the Raspberry Pi4 on the Ubuntu website. This guide has been tested on 64-bit versions of Ubuntu 20.04 and Ubuntu 21.04.
2. Run the `apt update` and `apt upgrade` commands. When on a fresh installation, it is best practice to run these commands. This step ensures all packages are the latest versions. Enter the following commands:

```
sudo apt update
sudo apt upgrade
```

3. Install the required packages that build and install the TensorFlow Lite runtime with Arm NN support. To install the packages, enter the following commands:

```
sudo apt-get install git wget unzip zip python cmake scons openjdk-11-jdk python3
 python3-pip
```

4. Set a base directory variable to refer to. To set the base directory, enter the following command:

```
export BASEDIR=$(pwd)
```

5. Install the TensorFlow Lite runtime package from source. To install the package, enter the following commands:

```
pip3 install --extra-index-url https://google-coral.github.io/py-repo/
 tflite_runtime
```

6. Download the pre-built binaries for Arm NN 21.11 for 64-bit Arm systems. The following commands download version 21.11 of the Arm NN binaries and place them into a directory called ArmNN-aarch64:

```
wget -O ArmNN-aarch64.tgz https://github.com/ARM-software/armnn/releases/down\
load/v21.11/ArmNN-linux-aarch64.tar.gz
mkdir ArmNN-aarch64
tar -xvf ArmNN-aarch64.tgz -C ArmNN-aarch64 --strip-components 1
```

3.2 Install on Odroid N2 Plus

The steps in this section cover installing the Arm NN TfLite Delegate on the Odroid N2 Plus.

About this task

Odroid lists many official Linux releases available on the Odroid wiki. This guide covers testing the Odroid N2 Plus on the Ubuntu MATE 20.04 image. In addition to an Arm Cortex-A CPU, the Odroid N2 Plus has an Arm Mali GPU. This installation includes installing the required packages to allow the Arm NN Delegate to take advantage of the GPU.

Note

If you struggle with memory errors on the Odroid N2 Plus in step nine, edit `build_pip_package_with_bazel.sh` to limit Bazel to use only four jobs using the option `--jobs=4`. Making this change can help the build system work more efficiently.

Procedure

1. Install the Ubuntu MATE 20.04 image.
2. Run the `apt update` and `apt upgrade` commands. When on a fresh installation, it is best practice to run these commands. This step ensures all packages are the latest versions. Enter the following commands:

```
sudo apt update
sudo apt -y upgrade
```

3. Install the OpenCL package for GPU support. To install the package, enter the following command:

```
sudo apt-get install -y ocl-icd-opencl-dev
sudo mkdir -p /etc/OpenCL/vendors/
sudo bash -c 'echo "libmali.so" > /etc/OpenCL/vendors/mali.icd'
```

4. Install the required packages to build and install TensorFlow Lite runtime with Arm NN support. To install the packages, enter the following commands:

```
sudo apt-get install git wget unzip zip python git-lfs cmake scons openjdk-11-jdk
 python3 python3-pip
```

5. Set a base directory variable to refer to. To set a base directory variable, enter the following command:

```
export BASEDIR=$(pwd)
```

6. Install the TensorFlow Lite runtime package from source. To install the package, enter the following commands:

```
pip3 install --extra-index-url https://google-coral.github.io/py-repo/
 tflite_runtime
```

7. Download the pre-built binaries for Arm NN 21.11 for 64-bit Arm systems.
The following commands download version 21.11 of the Arm NN binaries and place them into a directory called `ArmNN-aarch64`:

```
wget -O ArmNN-aarch64.tgz https://github.com/ARM-software/armnn/releases/down\
load/v21.11/ArmNN-linux-aarch64.tar.gz
```

```
mkdir ArmNN-aarch64
tar -xvf ArmNN-aarch64.tgz -C ArmNN-aarch64 --strip-components 1
```

4 Overview of running the application

This section includes the steps to run the application on your device.

This section covers the following:

- Downloading the sample application.
- Copying your `libtensorflow_lite_all.so` and `libarmnn.so` libraries into the example folder.
- Downloading a model and the corresponding label-mapping file.
- Downloading a sample image to use in the model.
- Installing the required Python packages.

4.1 Run the application

The steps in this section cover running the application on your device.

Before you begin

Before following the instructions in this section, ensure you follow the installation steps and have built and downloaded both `libarmnnDelegate.so` and `libtensorflow_lite_all.so`. Also, ensure you have installed the `tflite_runtime` Python package from source.

Procedure

1. Download the sample application. The application is inside the Arm NN repository in the samples folder. Perform a `git clone` on the Arm NN repository and change into the example folder. To perform a `git clone` and change into the example folder, enter the following commands:

```
git clone https://github.com/arm-software/armnn
cd armnn/samples/ImageClassification
```

2. Copy over the `libarmnn.so` and `libarmnnDelegate.so` libraries into the example folder. To copy these libraries, enter the following commands:

```
cp $BASEDIR/ArmNN-aarch64/libarmnn.so.27 .
cp $BASEDIR/ArmNN-aarch64/libarmnnDelegate.so.25 .
```

3. Download a model and the corresponding label-mapping file. In this application, we use the MobilenetV2 model. An optimized version of this model can be found in the Arm Model Zoo on GitHub.

```
export EXAMPLEDIR=$(pwd)
git clone https://github.com/arm-software/ml-zoo.git
# generate the label mapping
cd ml-zoo/models/image_classification/mobilenet_v2_1.0_224/tflite_uint8
./get_class_labels.sh
cd $EXAMPLEDIR
cp ml-zoo/models/image_classification/mobilenet_v2_1.0_224/tflite_uint8/mo\
bilenet_v2_1.0_224_quantized_1_default_1.tflite .
```

```
cp ml-zoo/models/image_classification/mobilenet_v2_1.0_224/tflite_uint8/labelmap\
pings.txt .
```

You can also use your own model if it satisfies the following requirements:

- It has been converted to the `.tflite` format.
- It expects an image input of either greyscale or RGB.
- It outputs a single vector containing probabilistic predictions for each class.
- You have the label-mapping file which converts the output vector index to a class label.

4. Download a sample image to use in the model. To download a sample image, enter the following command:

```
wget -O cat.png "https://github.com/dmlc/mxnet.js/blob/main/data/cat.p\
ng?raw=true"
```

5. Install the required Python packages. You must have followed the steps in the Device-specific installation section and installed `tflite_runtime` to perform this step. To install the Python packages you require, enter the following command:

```
pip3 install -r requirements.txt
```

6. Run the `run_classifier.py application`. You are ready to run this application. To run the application, we provide a command-line interface which allows you to pass in your model, your label mapping, the Arm NN backends, your input image, and the location of your delegate. The following is example usage for a device with an Arm CPU and GPU:

```
python3 run_classifier.py \
--input_image cat.png \
--model_file mobilenet_v2_1.0_224_quantized_1_default_1.tflite \
--label_file labelmappings.txt \
--delegate_path $BASEDIR/ArmNN-aarch64/libarmnnDelegate.so.24 \
--preferred_backends GpuAcc CpuAcc CpuRef
```

To see all available options with information about their use, you can use the command `python3 run_classifier.py -h`.

5 Code deep dive

In this section of the guide, we take a deep dive into the code we use in the application by building the application ourselves from scratch.

The Arm NN TfLite Delegate integrates with the TFLite package. If you are familiar with TFLite, using the delegate only adds an extra line of code to your projects. If you are not familiar with TFLite, there are many TensorFlow tutorials, guides, and sample applications online you can use to learn more.

An image classification application performs the following steps:

- Load the model.
- Load the input image.
- Load the label-mapping files.
- Resize the image to fit with the models expected input size.
- Run the inference engine.
- Calculate which output neuron has the highest output value. The highest output value represents the highest probability.
- Convert the index of the neuron to a label-mapping file.

5.1 Build the application

The following steps allow you to follow along with the code deep dive and build the application.

Procedure

1. Import the dependencies and create a command-line interface with `argparse`. We rely on the following external dependencies: Pillow, a package to help with image processing; Numpy, a package used to store and modify arrays; and the TensorFlow Lite runtime package.
   For the command-line interface, we add options that allow you to input the following:
   - An input image.
   - A model.
   - A label mapping.
   - The location of the delegate library.
   - The backends Arm NN uses.

   ```
   import argparse
   from pathlib import Path
   from typing import Union
   import tflite_runtime.interpreter as tflite
   from PIL import Image
   import numpy as np

   parser = argparse.ArgumentParser(
   formatter_class=argparse.ArgumentDefaultsHelpFormatter
   ```

```
)
parser.add_argument(
'--input_image', help='File path of image file', type=Path,
required=True
)
parser.add_argument(
'--model_file', help='File path of the model tflite file', type=Path,
required=True
)
parser.add_argument(
'--label_file', help='File path of model labelmapping file', type=Path,
required=True
)
parser.add_argument(
'--delegate_path', help='File path of ArmNN delegate file', type=Path,
required=True
)
parser.add_argument(
'--preferred_backends', help='list of backends in order of preference',
type=str, nargs='+', required=False, default=["CpuAcc", "CpuRef"]
)
args = parser.parse_args()
```

2. Load the `tflite` model file. Compared to a normal TensorFlow Lite application, we add one or two extra lines to import the Arm NN delegate.
   Use the `load_delegate()` function from `tflite`. This function takes as input the path of the delegate and the options for the delegate, given in a Python dictionary format. For the Arm NN Delegate, the main option is the backends option. On this option, you input the backends available on your device or Arm NN installation.

   The backend option is in string format. For example: `GpuAcc,CpuAcc,CpuRef`.

   Use the `tflite` Interpreter class. For this class, pass the model file path as an input and add a section option of `experimental_delegates`. In this input, we pass the Arm NN delegate. This input tells the interpreter to offload calculations to the delegate.

   The following example code loads the `tflite` model file:

```
delegate_path = args.delegate_path
model_path = args.model_file
backends = args.preferred_backends
backends = ",".join(backends)
#load the delegate
armnn_delegate = tflite.load_delegate(delegate_path,
options={
    "backends": backends,
    "logging-severity": "info"
}
)
#load the model and delegate to the interpreter
interpreter = tflite.Interpreter(
model_path=model_path.as_posix(),
experimental_delegates=[armnn_delegate]
)
```

3. Load the label-mapping files. You must convert the output into an understandable format. When a neural network model is being trained, we assign each class an output neuron. Label-mapping files take the output from a neural network and convert it back to a class. This output usually comes in the form of a text file where each row has a class name. Usually, the row

number refers to the index of an output neuron. Also, both the row number and the index of the output neuron usually start at zero.
For this reason, when loading the label-mapping files it is a good idea for it to be in a Python dictionary. Loading the label-mapping files this way makes it easy to query the class label from the output of a neural network.

The following example code loads the label-mapping files:

```
label_mapping_p = args.label_file
idx = 0
label_mapping = {}
with open(label_mapping_p) as label_mapping_raw:
for line in label_mapping_raw:
    label_mapping[idx] = line
    idx += 1
```

4. Load the input image. The input image must be of a specific size and format. A common format is 224 x 224 x 3. This format means a height of 224 pixels, a width of 224 pixels, and three channels of red, green, and blue. You can find out the expected input format for the model from the interpreter you created previously. Using this input format, you can process your image.
The following example code loads the input image:

```
input_details = interpreter.get_input_details()
input_shape = input_details[0]["shape"]
#batch size = input_shape[0], this can be ignored here. We will only be using 1
 img.
height, width, channels = input_shape[1], input_shape[2], input_shape[3]
#if channels == 1 then greyscale
greyscale = channels == 1
#load the input image
image_path = args.input_image
image = Image.open(image_path).resize((width, height))
if grayscale:
image = image.convert("LA")
#extend dims so model can expect it.
image = np.expand_dims(image, axis=0)
```

5. Run the inference engine. You must allocate the tensors, assign the input image to the input tensor, and run the inference. The following example code runs the inference engine:

```
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
interpreter.set_tensor(input_details[0]['index'], image)
interpreter.invoke()
output_data = interpreter.get_tensor(output_details[0]['index'])
```

6. Process the output data. To process the output data, you calculate the largest output value. The largest output represents the highest probability, so by taking the largest value you get the value the model believes is most likely. The following example code calculates the largest output value:

```
output_index = np.argmax(output_data[0])
```

7. Calculate the class label. You can query the dictionary you made earlier for class labels with the output index you calculated. The following example code calculates the class label:

```
output_class = label_mapping[output_index]
```

```
print("Predicted class: ", output_class)
```

6 Related information

Here are some resources related to the material in this guide:

- Odroid N2+
- Odroid Wiki
- Raspberry Pi 4
- TensorFlow Lite

Other Arm resources:

- Arm Community - ask development questions and find articles and blogs on specific topics from Arm experts.
- Arm NN Github - raise queries or issues associated with the Arm NN how-to guides.
- Arm NN Product Documentation - find out more about the latest Arm NN features.

7 Next steps

This guide covers the steps to develop an image classification application using the quantized TensorFlow Lite MobileNet V2 model. This model is available in the Arm ML-Zoo and the Arm NN TensorFlow Lite Delegate. You can use the example code in this application as a starting point for your own image classification applications.

You can also have a look at how the MobileNet V2 model was trained and train your own using the TensorFlow Slim GitHub repository.

For application inspiration, why not look at our fire detection use case which uses image classification to detect fires?

Appendix A Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1: First release for version 1.01

Change	Location
First release	—

Table A-2: First release for version 21.08

Change	Location
Updates steps	Install on Odroid N2 Plus
Updates steps	Run the application
Adds additional resources	Related information

Table A-3: Second release for version 21.08

Change	Location
Adds additional resources	Related information
Fixes typo	Install on Odroid N2 Plus

Table A-4: First release for version 21.11

Change	Location
Updates the Odroid N2 Plus installation instructions	Install on Odroid N2 Plus
Updates the Raspberry Pi installation instructions	Install on Raspberry Pi
Updates library version numbers and removes non-inclusive language	Run the application