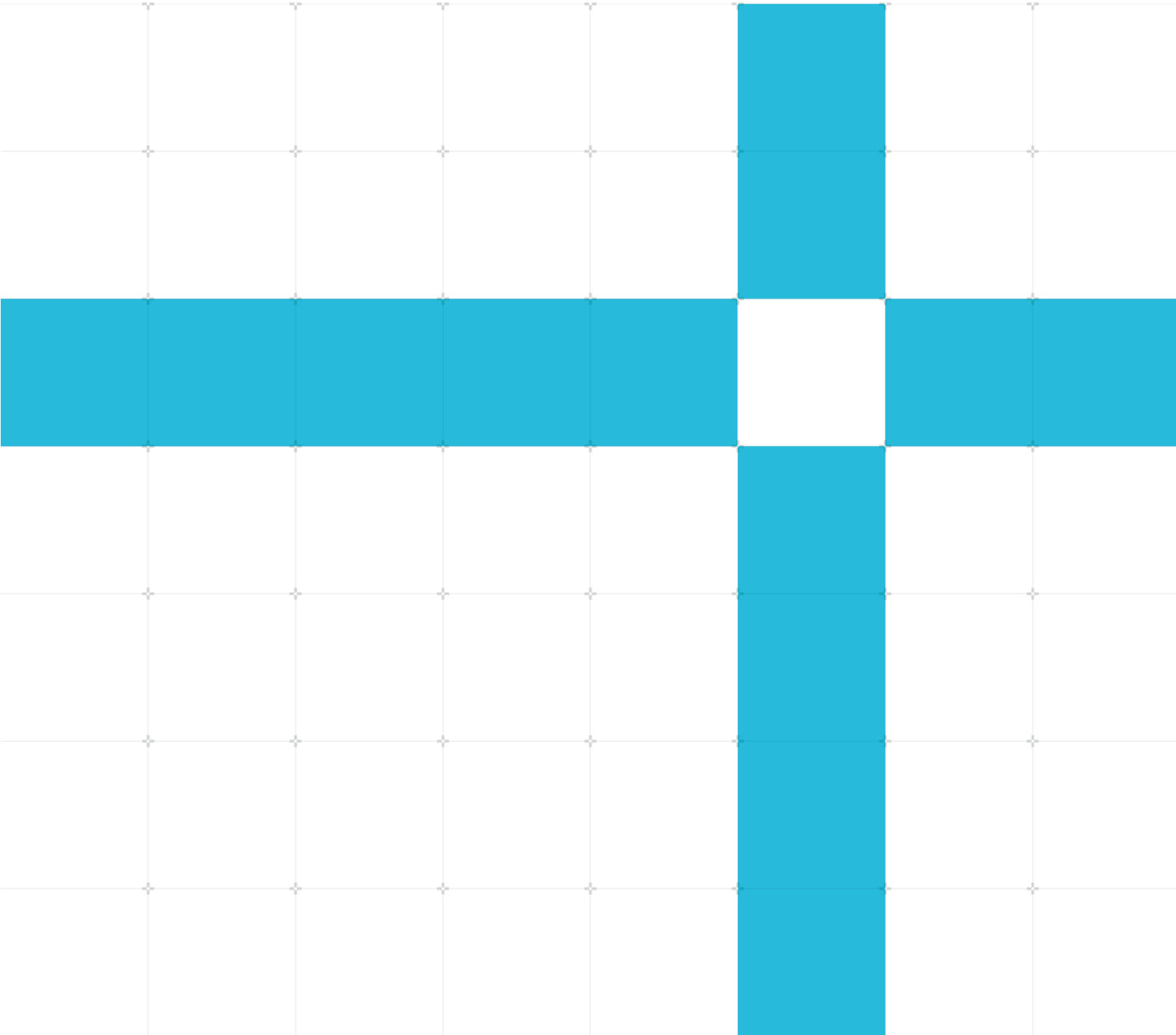
Deploying a TensorFlow MNIST model using the Arm NN SDK

Non-Confidential

Issue 1.2
ARM-ECM-0744361



Deploying a TensorFlow MNIST model using the Arm NN SDK

Release information

Document history

Issue	Date	Confidentiality	Change
1.1	July 07 2018	Non-Confidential	First release
1.2	May 27 2020	Non-Confidential	General editorial updates

Non-Confidential Proprietary Notice

(LES-PRE-20349)

Confidentiality Status

Web Address

www.arm.com

Contents

1 Introduction

1.1 Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rm

Identifies the major revision of the product, for example, r1.

pn

Identifies the minor revision or modification status of the product, for example, p2.

1.2 Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rm

Identifies the major revision of the product, for example, r1.

pn

Identifies the minor revision or modification status of the product, for example, p2.

1.3 Intended audience

This guide is for software developers who want to deploy an application that runs a TensorFlow model to an Arm-based device.

1.4 Conventions

The following subsections describe conventions used in Arm documents.

1.4.1 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: **https://developer.arm.com/glossary**.

1.4.2 Typographical conventions

Convention	Use
italic	Introduces 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` **`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.
	This represents a recommendation which, if not followed, might lead to system failure or damage.
	This represents a requirement for the system that, if not followed, might result in system failure or damage.
	This represents a requirement for the system that, if not followed, will result in system failure or damage.
	This represents an important piece of information that needs your attention.
	This represents a useful tip that might make it easier, better, or faster to perform a task.
	This is a reminder of something important that relates to the information you are reading.

1.5 Feedback

Arm welcomes feedback on this product and its documentation.

1.5.1 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.

1.5.2 Feedback on content

If you have comments on content, send an email to errata@arm.com and give:

- The title Deploying a TensorFlow MNIST model using the Arm NN SDK.
- The number ARM-ECM-0744361.
- 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.

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 Overview

This guide shows you how to run a TensorFlow model using the open-source **Arm NN SDK** using an example application. You can use the knowledge that you gain from this guide to run your own models on Arm Cortex CPUs and Mali GPUs.

This guide uses Arm NN to run a model, following these steps:

1. Load and parse the MNIST test set.
2. Import graph.
3. Optimize and load onto a compute device.
4. Run graph on device.

The guide explains each step of the example code to help you understand each stage of the process.

2 Before you begin

We assume that you are familiar with neural networks and the MNIST dataset. If you are new to either of these concepts, read the TensorFlow tutorials and the MNIST database of handwritten digits paper.

The complete example with source code, data, and model is available on **GitHub**.

3 Building you TensorFlow model using the Arm NN SDK

3.1 Load and parse the MNIST test set

To begin building your own TensorFlow model, load and parse the MNIST test set.

The following sample code loads and parses the MNIST test set:

```
// Load a test image and its correct label
std::string dataDir = "data/";
int testImageIndex = 0;
std::unique_ptr<MnistImage> input = loadMnistImage(dataDir, testImageIndex);
```

The `loadMnistImage` helper function is not covered here. In simple terms, it parses the MNIST data files and returns an `MnistImage` struct for the requested image with a label and a 28*28=784 element array containing the data:

```
//Helper struct for loading MNIST data
struct MnistImage
{
	unsigned int label;
	float image[g_kMnistImageByteSize];
};
```

3.2 Import graph

Arm NN provides parsers for reading model files from neural network frameworks. There are typically two steps to do this:

1. Load the model.
2. Bind the input and output points of its graph.

The following sample code imports the graph:

```
// Import the TensorFlow model. Note: use CreateNetworkFromBinaryFile for .pb files.
armnnTfParser::ITfParserPtr parser = armnnTfParser::ITfParser::Create();
armnn::TensorInfo inputTensorInfo({1, 784, 1, 1}, armnn::DataType::Float32);
armnn::INetworkPtr network = parser-
>CreateNetworkFromTextFile("model/simple_mnist_tf.prototxt",
                                                                { {"Placeholder",
{1, 784, 1, 1}} },
                                                                { "Softmax" });
```

TensorFlow graphs in both text and binary ProtoBuf formats are supported. For more details on freezing TensorFlow graphs to include their weights, see the Customization basics: tensors and operations guide in the Related information section.


Note

After this step, the code is common regardless of the framework that you started with. This is because the `INetwork` and two `BindingPointInfo` objects provide everything that is needed.

3.3 Optimize and load onto a compute device

Arm NN supports optimized execution on multiple devices, including CPU and GPU. Before you start executing a graph, you must select the appropriate device context and optimize the graph for that device.

Using an Arm Mali GPU is as simple as specifying `Compute::GpuAcc` when creating the context. No other changes are required.

For a Raspian base installation, the only dependency that you must add is TensorFlow from Google's binaries. Install some TensorFlow prerequisites by entering the following code in the command line:

```
// Create a context and optimize the network for one or more compute devices in order
of preference

// e.g. GpuAcc, CpuAcc = if available run on Arm Mali GPU, else try to run on Arm v7 or
v8 CPU

armnn::IRuntime::CreationOptions options;

armnn::IRuntimePtr context = armnn::IRuntime::Create(options);

armnn::IOptimizedNetworkPtr optNet = armnn::Optimize(*net, {armnn::Compute::GpuAcc,
armnn::Compute::CpuAcc}, runtime->GetDeviceSpec());

//Load the optimized network onto the device

armnn::NetworkID networkIdentifier;

context->LoadNetwork(networkIdentifier, std::move(optNet));
```

3.4 Run graph on device

Inference on a compute device is performed using the `EnqueueWorkload()` function of the context.

This example code runs a single inference on the test image:

```
// Run a single inference on the test image

std::array<float, 10> output;

armnn::Status ret = context->EnqueueWorkload(networkIdentifier,

                                             MakeInputTensors(inputBindingInfo, &input-
>image[0]),
```

```
                                                MakeOutputTensors(outputBindingInfo,
&output[0]));
```

Here the input and output tensors are bound to data and the loaded network identifier is selected. The result of the inference can be read directly from the output array and compared to the MnistImage label that we read from the data file:

```
// Convert 1-hot output to an integer label and print
int label = std::distance(output.begin(), std::max_element(output.begin(),
output.end()));
std::cout << "Predicted: " << label << std::endl;
std::cout << "   Actual: " << input->label << std::endl;
```

In this case, the `std::distance` function is used to find the index of the largest element in the output. This function is the equivalent to NumPy's `argmax()` function.

4 Deploy an application using Arm NN

You must link your application with the Arm NN library and the TensorFlow parsing library. The following code links your application:

```
g++ -std=C++11 -I$(ARMNN_INC) mnist.cpp -o mnist -L$(ARMNN_LIB) -larmnn -larmnnTfParser
```

For convenience, Arm NN can run with a reference implementation on x86 for development and testing, but the optimized kernels are only available for Arm CPUs and GPUs. This means that you must run your profiling and optimization steps on a real Arm-powered device, because x86 performance is not representative.

5 Related information

Here are some resources related to material in this guide:

Arm Community - Ask development questions and find articles and blogs on specific topics from Arm experts at the Arm community.

Arm documentation - Arm architecture and reference manuals

Arm NN - The Arm Developer Arm NN page.

Customization basics: tensors and operations - An introductory TensorFlow tutorial

TensorFlow Tutorials - Beginner TensorFlow guides

The MNIST database of handwritten digits - Overview of MNIST database

6 Next steps

This guide describes the steps that are required to run a TensorFlow model using the Arm NN SDK. Using the information in this guide, you can to build and run your own models using Arm NN.

Explore **this complete example in our GitHub repository**.

Visit **the Arm NN site** to see what else is possible.

Appendix A Revisions

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

Table A-1 Issue 1.1

Change	Location	Affects
First release	-	-

Table A-2 Differences between issue 1.0 and issue 1.2

Change	Location	Affects
General content tweaks for uniformity across how-to guides.	All	All
Adds Related Information section	Related Information	All
Updates Next steps section	Next steps	All