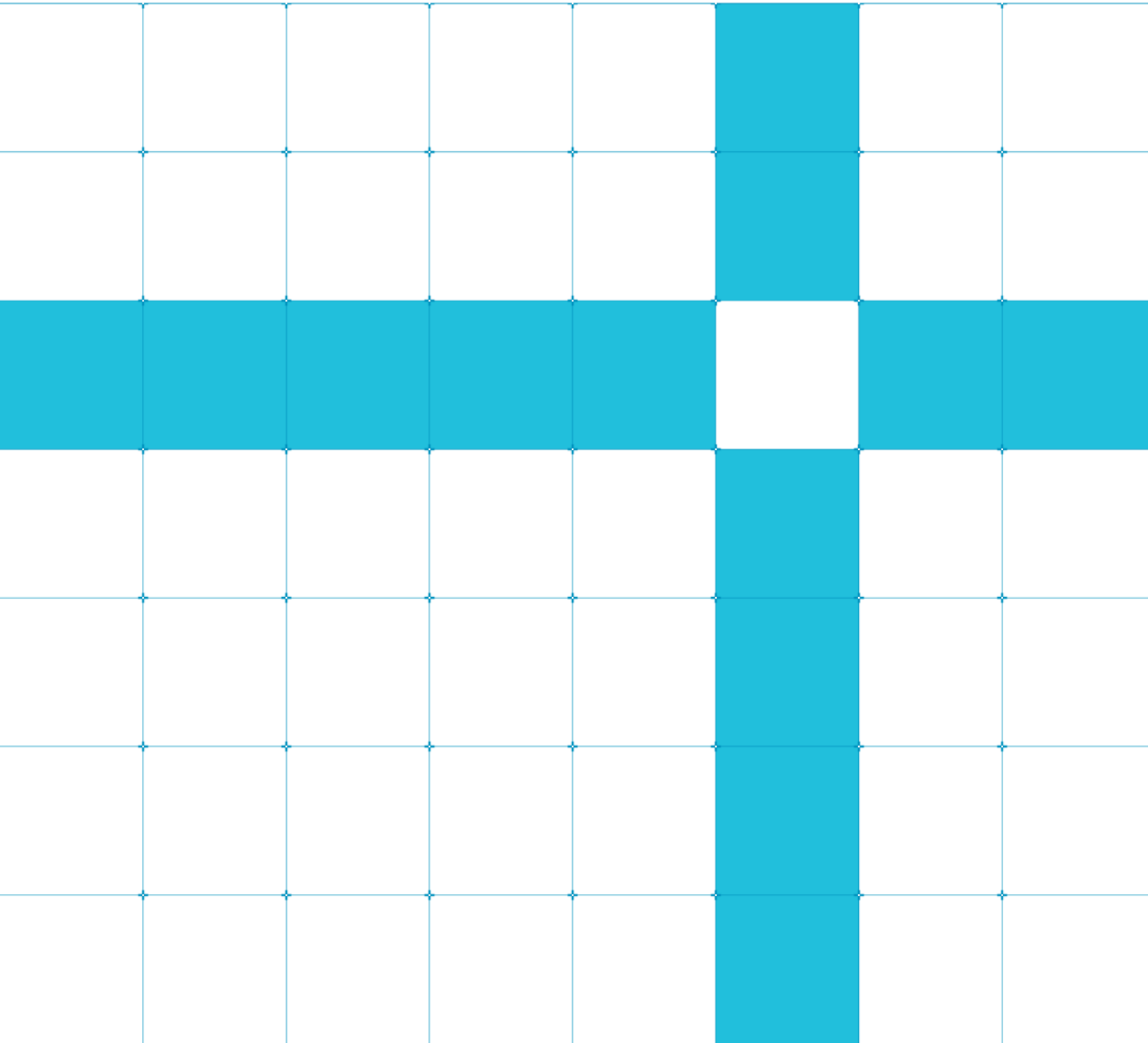
Improving your machine learning workflow using the Arm NN SDK

Version 1.0



Improving your machine learning workflow using the Arm NN SDK

Release Information

Document History

Version	Date	Confidentiality	Change
1.0	28 August 2018	Non-Confidential	First release

Non-Confidential 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

http://www.arm.com

Contents

1 Overview

This guide takes you through a typical machine learning workflow with a 32-bit floating-point *Convolutional Neural Network* (CNN). The guide covers training your neural network to running inference. You will also learn how to diagnose problems with your models and tune your models to improve performance when using a GPU backend.

2 Prerequisites

This guide assumes:

- You use the TensorFlow or Caffe neural network framework.
- Your machine learning workflow is carried out using 32-bit floating-point data types.
- The Arm NN SDK build environment is configured for the required parser. For more information on how to do this configuration, see **Configuring the Arm NN SDK build environment for TensorFlow** and **Configuring the Arm NN SDK build environment for Caffe**.

3 Training your neural network model

To start building your neural network application, you select a CNN model to solve a problem. You can download pre-trained models to build your neural network applications or train your own.

CNNs are commonly trained with 32-bit floating-point data and there is documentation widely available on training your own on different frameworks. For more information about training a CNN model using the MNIST test set, see the **TensorFlow** or **Caffe** documentation.

4 Removing training-only nodes

4.1. TensorFlow

The Arm NN SDK is an inference-only engine. For performance and compatibility, you must remove training-only nodes when using the TensorFlow neural network framework. These nodes include operations that you do not require for inference.

TensorFlow provides **documentation on how to remove these training-only nodes**.

4.2. Caffe

You do not need to remove training-only nodes when you use the Caffe neural network framework as the Arm NN SDK handles the removal.

5 Loading the model into the Arm NN SDK runtime

If everything has been done correctly, your network is now ready for use with the Arm NN SDK. For a detailed example of how to load your model into the Arm NN SDK runtime, see **Deploying a Caffe MNIST model using the Arm NN SDK** or **Deploying a TensorFlow MNIST model on Arm NN**. This documentation includes example code.

Briefly, to load your model into the Arm NN runtime, you must complete the following steps:

1. Link to the appropriate parser for your framework of choice and the core Arm NN runtime.
2. Instantiate the parser.
3. Load the model file using the parser.
4. Create a `RunTime` object using the `lRuntime::Create()` function.
5. Pass the `DeviceSpec` object and the input network object to the `Optimize()` function. The parser produces the input network object and you can query the `DeviceSpec` object using the `Runtime::GetDeviceSpec()` function.
6. Load the `IOptimizedNetwork` object that the `Optimize()` function produces into the runtime using the `IRuntime::LoadNetwork()` function.

6 Running inference

After loading your model into the Arm NN SDK, you are now ready to run inference. You must ensure that any inputs you run inference on have had the same pre-processing that you used for training. All models have different pre-processing requirements. The following are examples of some of the actions you perform during pre-processing:

- Format conversion.
- Resizing.
- Mean subtraction.

You can optionally do batching for performance improvements.

Finally, you call the function to run inference. For an example of how to load your model into the Arm NN SDK runtime and run inference see, **Deploying a Caffe MNIST model using the Arm NN SDK** or **Deploying a TensorFlow MNIST model on Arm NN**. This documentation includes example code.

7 Cleaning up

To free the run-time memory that a model uses on your device, you must unload the model from the runtime. To unload your model, you must use the `IRuntime::UnloadNetwork` method.

8 Testing

You can set the Arm NN SDK logging level to diagnose problems when loading or executing models. Set the Arm NN SDK logging level using the `armnn::ConfigureLogging()` and `armnnUtils:: ConfigureLogging ()` functions.

The following example code sets the Arm NN SDK logging level when you insert it at the beginning of your application:

```
armnn:: LogSeverity level =
armnn:: LogSeverity::Debug;
armnn:: ConfigureLogging (true, true, level);
armnnUtils:: ConfigureLogging (boost::log::core::get().get(), true, true, level);
```

9 Tuning

When you use the GPU backend, you can improve performance by tuning the application. To improve performance, you:

1. Set the Arm NN SDK in tuning mode by doing the following:
    a. Create an `ICLTunedParameters` object using the `ICLTunedParameters::Create` method. Pass `Mode::UpdateTunedParameters` as the argument.
    b. Before you create the `RunTime` object, set the `m_ClTunedParameters` member on the `CreationOptions` struct using the tuned parameters object created in step 1.
2. Optionally load any existing tuning data that you want to use to extend new models with using the `IClTunedParameters::Load` method.
3. Run the model with the Arm NN SDK set in tuning mode by:
    a. Loading the model that you want to tune the performance of into the `RunTime` object. For information on how to load the model, see **Loading the model into the Arm NN SDK**.
    b. Running inference at least once to tune the performance of the model. For information on how to run inference, see **Running inference**.
    c. Unloading the model.
    d. For every model you want to do performance tuning on, repeat steps 1-3.
    e. Saving the tuned parameters using the `ICLTunedParameters::Save` method.

10 Next steps

This guide covered the steps to building a neural network application with the Arm NN SDK using a 32-bit floating-point CNN. You can also use this guide as a starting point to handle other types of neural networks. You are now ready to build your own neural network application using the Arm NN SDK.