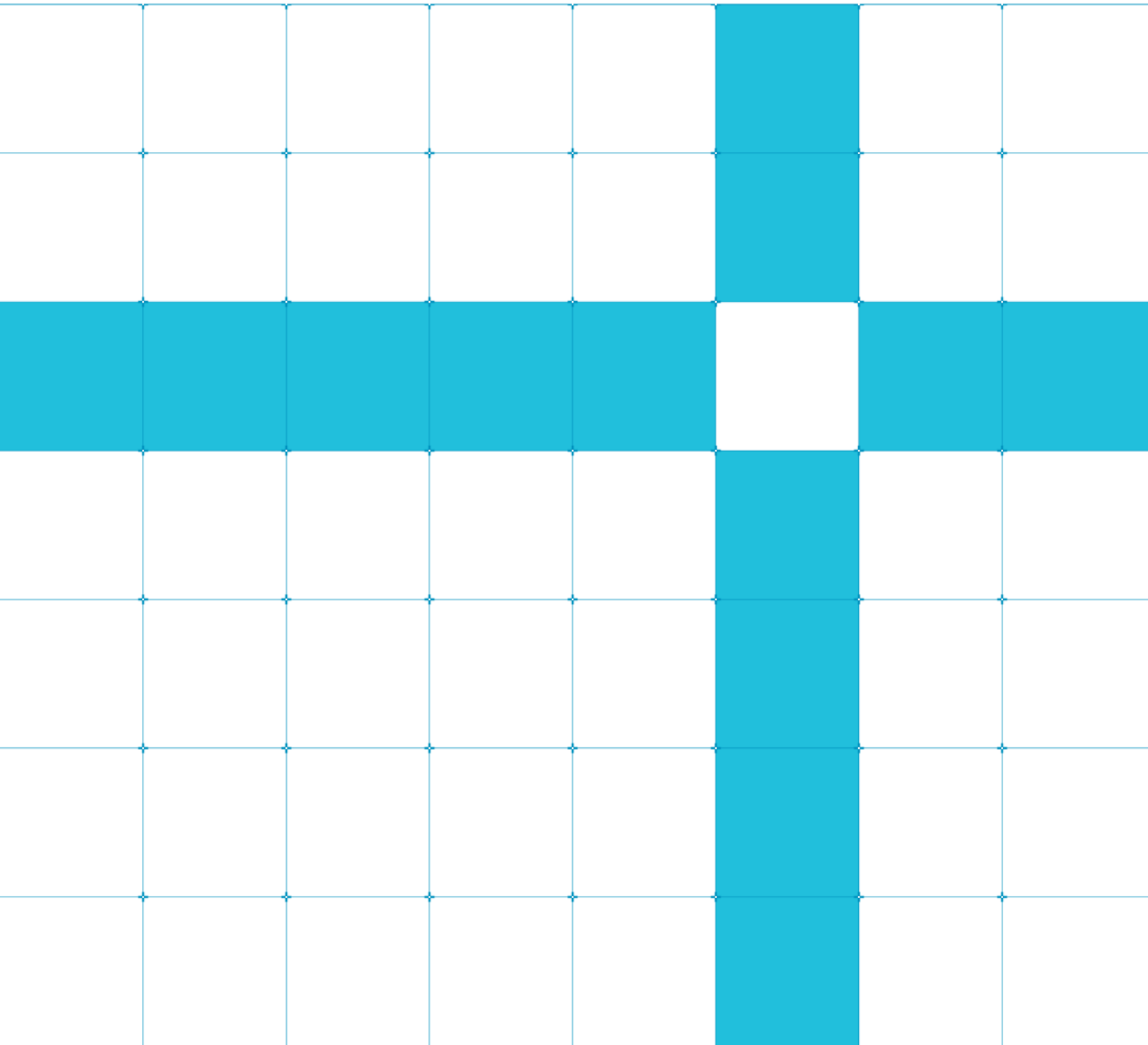
Quantizing Neural Networks to 8-bit Using TensorFlow

Version 1.2



Quantizing Neural Networks to 8-bit Using TensorFlow

Release Information

Document History

Version	Date	Confidentiality	Change
1.0	01 October 2018	Non-Confidential	First release
1.1	18 January 2019	Non-Confidential	Minor enhancements
1.2	20 February 2020	Non-Confidential	Changes the TensorFlow version in Before you begin to reflect latest version Arm tested.

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

www.arm.com

Contents

1 Overview

With the launch of TensorFlow Lite, TensorFlow has been updated with quantization techniques and tools that you can use to improve the performance of your network.

This guide shows you how to quantize a network so that it uses 8-bit data types during training, using features that are available from TensorFlow 1.9 or later.

Devices can execute 8-bit integer models faster than 32-bit floating-point models because there is less data to move and simpler integer arithmetic operations can be used for multiplication and accumulation.

If you are deploying TensorFlow models using CoreML, Arm recommend that you convert the 32-bit unquantized model to CoreML. To convert the model to CoreML, use **github.com/tf-coreml/tf-coreml** and then use the CoreML quantization tools to optimize the model for deployment. Check the **Apple Developer** website for more updates on this.

Note that it is not currently possible to deploy 8-bit quantized TensorFlow models via CoreML on iOS. However, you can use the same technique to reduce the compressed model size for distribution using the `round_weights` transform described in the **TensorFlow GitHub**, or to deploy 8-bit models using the TensorFlow C++ interface.

2 Before you begin

Before you can use the TensorFlow Lite quantization tools, you must:

1. **Install TensorFlow** 1.9 or later. Arm tested TensorFlow version 1.15.
2. To follow the CifarNet examples in this article, clone the tensorflow/models repository from GitHub using the command:
   ```
   git clone https://github.com/tensorflow/models.git
   ```
   Use the master branch. Arm tested the checkout, d4e1f97fd8b929deab5b65f8fd2d0523f89d5b44.
3. Prepare your network for quantization:
   1. Remove unsupported operations that the TensorFlow quantization toolchain doesn't support yet. Note that this support will change over time. See the **TensorFlow documentation** for details.

      To remove unsupported operations from CifarNet, lines 68 and 71 must be removed from `models/research/slim/nets/cifarnet.py`.
   2. Add fake quantization layers to the model graph, before you initialize the optimizer. Call this function on the finished graph to add these layers:
      ```
      tf.contrib.quantize.create_training_graph()
      ```
      For a CifarNet example, you modify `models/research/slim/train_image_classifier.py` and add `tf.contrib.quantize.create_training_graph(quant_delay=90000)` before the procedure to configure the optimization procedure on line 514. The `quant_delay` parameter specifies how many steps the operation allows the network to train in *32-bit floating-point* (FP32), before it introduces quantization effects. The value 90000 indicates that the model will be trained for 90000 steps with floating-point weights and activations, before the quantization process begins. You can also load the weights of an existing trained model and fine-tune it for quantization. In this case, add the code that loads the weights and set the `quant_delay` value to 0 so that quantization begins immediately.

3 Train or Fine-tune Your Model

Train your model using your training data and compare the accuracy with the original 32-bit network. The training process varies by model.

To fine-tune an existing model using quantization, load the weights from your trained model into a graph that you use the `create_training_graph()` function to prepare. The `create_training_graph()` function is described in the **Prerequisites** section. When you have loaded the weights, allow the model to continue training for a smaller number of steps.

When the training is complete, compare the accuracy with the original 32-bit network.

For CifarNet, you can use the following commands to download the training data to `/tmp/cifar10` and train the network for 100000 steps:

```
cd models/research/slim/
bash scripts/train_cifarnet_on_cifar10.sh
```

The fake quantization layers that `tf.contrib.quantize.create_training_graph()` adds become active after 90000 steps and ensure that the final network is fine-tuned to work with quantized weights. The training process creates a `/tmp/cifarnet-model` directory that contains the graph and checkpoint weights.

To view the training progress:

1. Run the following command into TensorBoard:
   ```
   tensorboard --logdir=/tmp/cifarnet-model/
   ```
2. Open port 6006 on the training server in a browser. If you're training on your laptop or desktop, this is **http://localhost:6006/**.

Training takes approximately 40 minutes on a p3.2xlarge EC2 instance.

The final accuracy is approximately 85% for CifarNet.

4 Prepare the Graph for Inference

To prepare the graph for inference with TensorFlow Lite or Arm NN, optimize the graph for inference, and freeze it:

1. Add fake quantization layers to the graph. This modifies the way the inference graph is exported, to make sure that it is exported with the quantization information in the right format. To add the fake quantization layers, call `tf.contrib.quantize.create_eval_graph()` on the inference-ready graph before saving it.

   For CifarNet, you can do this by modifying the file `models/research/slim/export_inference_graph.py` and adding `tf.contrib.quantize.create_eval_graph()` before `graph_def = graph.as_graph_def()` on line 118.
2. Export the inference graph to a protobuf file. For CifarNet this is done using:

   ```
   python export_inference_graph.py --model_name=cifarnet --dataset_name=cifar10 --
   output_file=/tmp/cifarnet_inf_graph.pb
   ```
   At this point, the graph does not contain your trained weights.
3. Freeze the graph using the `freeze_graph` tool. You can specify any checkpoint during training for this. The command to freeze the graph is:
   ```
   python -m tensorflow.python.tools.freeze_graph \
   --input_graph=<your_graph_location> \
   --input_checkpoint=<your_chosen_checkpoint> \
   ```

```
--input_binary=true \
--output_graph=<output_directory> \
--ouput_node_names=<output_nodes>
```

For CifarNet, using the last checkpoint, you can do this with the commands:

```
export LAST_CHECKPOINT=`head -n1 /tmp/cifarnet-model/checkpoint | cut -d'"' -f2`
python -m tensorflow.python.tools.freeze_graph \
--input_graph=/tmp/cifarnet_inf_graph.pb \
--input_checkpoint=${LAST_CHECKPOINT} \
--input_binary=true \
--output_graph=/tmp/frozen_cifarnet.pb \
--output_node_names=CifarNet/Predictions/Softmax
```

5 Quantize the Graph

Use the TensorFlow Lite Converter `tflite_convert` to optimize the TensorFlow graphs and convert them to the TensorFlow Lite format for 8-bit inference. This tool is installed as standard in your path with TensorFlow 1.9 or later.

To use the TensorFlow Lite Converter:

1. Use the `tflite_convert` command-line program using the command:

   ```
   tflite_convert --graph_def_file=<your_frozen_graph> \
   --output_file=<your_chosen_output_location> \
   --input_format=TENSORFLOW_GRAPHDEF \
   --output_format=TFLITE \
   --inference_type=QUANTIZED_UINT8 \
   --output_arrays=<your_output_arrays> \
   --input_arrays=<your_input_arrays> \
   --mean_values=<mean of input training data> \
   --std_dev_values=<standard deviation of input training data>
   ```

   For CifarNet, this command is:

   ```
   tflite_convert --graph_def_file=/tmp/frozen_cifarnet.pb \
   --output_file=/tmp/quantized_cifarnet.tflite \
   --input_format=TENSORFLOW_GRAPHDEF \
   --output_format=TFLITE \
   --inference_type=QUANTIZED_UINT8 \
   --output_arrays=CifarNet/Predictions/Softmax \
   --input_arrays=input \
   --mean_values 121 \
   --std_dev_values 64
   ```

   This command creates an output file that is one quarter of the size of the 32-bit frozen input file.

   For more information on using the TensorFlow Lite Converter, see the **TensorFlow GitHub**.

2. Check the accuracy of your result to ensure that it is the same as the original 32-bit graph.