



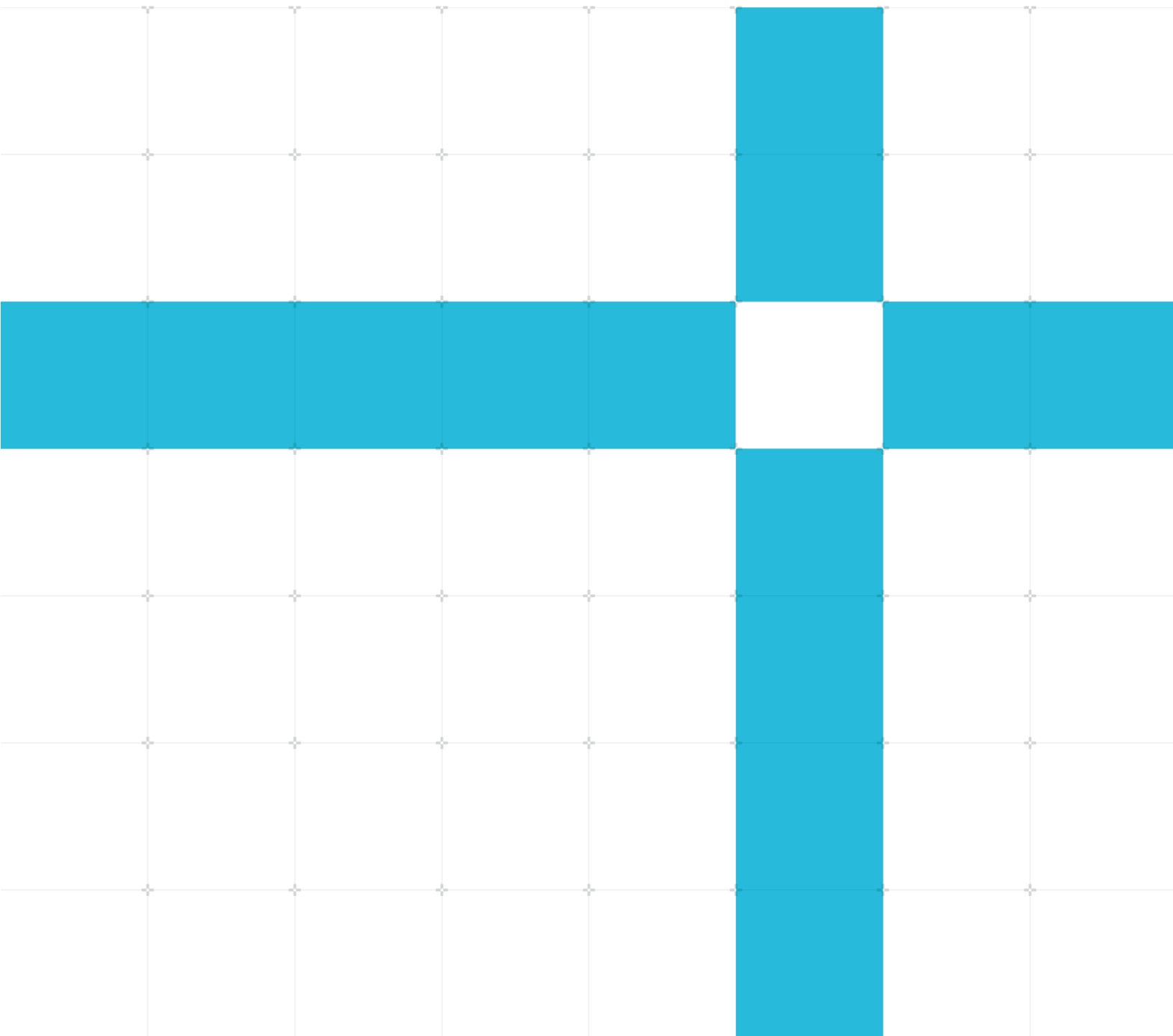
Optimizing neural networks for mobile and embedded devices with TensorFlow

Non-Confidential

Copyright © 2018, 2020 Arm Limited (or its affiliates).
All rights reserved.

Issue 1.2

ARM-ECM-0744361



Optimizing neural networks for mobile and embedded devices with TensorFlow

Copyright © 2018, 2020 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
1	20/04/2018	Non-Confidential	First release
1.2	6/8/2020	Non-Confidential	General editorial updates

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 affiliates) 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 © 2018, 2020 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.

Web Address

www.arm.com

Contents

1 Introduction.....	5
1.1 Product revision status	5
1.2 Intended audience.....	5
1.3 Conventions	5
1.3.1 Glossary.....	5
1.3.2 Typographical conventions.....	6
1.4 Additional reading.....	7
1.5 Feedback.....	8
1.5.1 Feedback on this product	8
1.5.2 Feedback on content.....	8
2 Overview	9
3 Before you begin.....	11
4 Determine the names of input and output nodes.....	12
5 Generate an optimized 32-bit model.....	15
6 Generate an optimized 8-bit model.....	16
7 Benchmark the optimized models	20
7.1 Improving model performance.....	21
8 Deploy the optimized models.....	20
9 Related information.....	21
10 Next steps.....	22

1 Introduction

1.1 Product revision status

The *rm**pn* 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 Intended audience

This guide is for software developers who want to learn how to optimize neural networks for embedded devices with TensorFlow.

1.3 Conventions







The following subsections describe conventions used in Arm documents.

1.3.1 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>.

1.3.2 Typographical conventions

Convention	Use
<i>italic</i>	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 <u>underline</u>	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: <code>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></code>
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.
 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.

1.4 Additional reading

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

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 Optimizing TensorFlow models for mobile and embedded devices.
- 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.



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.

2 Overview

There are many ways to deploy a trained neural network model to a mobile or embedded device. Different frameworks support Arm, including TensorFlow, PyTorch, Caffe2, MxNet, and CNTK on a various platforms, such as Android, iOS, and Linux. The deployment process for each is similar but every framework and operating system may use different tools. This walkthrough looks specifically at preparing TensorFlow models for deployment on Android, Linux, and iOS.

Optimization of a trained neural network model with TensorFlow follows these steps:

1. Determine the names of the input and output nodes in the graph and the dimensions of the input data.
2. Generate an optimized 32-bit model using TensorFlow's `transform_graph` tool.
3. Generate an optimized 8-bit model that is more efficient but less accurate using TensorFlow's `transform_graph` tool.
4. Benchmark the optimized models on-device and select the one that best meets your deployment needs.

This tutorial goes through each step in turn, using a pretrained ResNet-50 model (`resnetv1_50.pb`). The process is the same for other models, although input and output node names differ.

At the end of this tutorial, you will be ready to deploy your model on your chosen platform.

3 Before you begin

This tutorial assumes you already have a TensorFlow .pb model file using 32-bit floating point weights. If your model is in a different format

- Keras
- PyTorch
- Caffe
- MxNet
- CNTK

If you want to deploy it using TensorFlow then you will need to use a tool to convert it to the TensorFlow format first.

There are various projects and resources building up around converting model formats. Both, [MMdnn](#) and [Deep learning model converter](#) are useful resources, and [the ONNX format](#) has potential to vastly simplify this in the future.

The most important preparation that you can do is to ensure that the size and complexity of your trained model is suitable for the device that you intend to run it on. To ensure this:

- If you are using a pre-trained model for feature extraction or transfer learning, then you should consider using mobile-optimized versions such as MobileNet, TinyYolo, and so on.
- If you designed the architecture yourself, then you should consider adapting the architecture for faster execution and smaller size. An example of this is using depth-separable convolutions where possible, as in MobileNet. This provides better performance and accuracy improvements than by post-processing the model file after training.

Please follow [TensorFlow](#) documentation to install Bazel and other dependencies, and download the TensorFlow source code. At the root of your TensorFlow source tree, run `./configure` to configure the system build.

This tutorial uses TensorFlow's `graph_transforms` tool, which is built from the [TensorFlow source](#) with this command:

```
bazel build tensorflow/tools/graph_transforms:transform_graph
```

For more details on how to install and build TensorFlow, see the [TensorFlow documentation](#).

4 Determine the names of input and output nodes

Skip this step if you can determine the names of the input and output nodes from the provider of your model or the training code. However, this step also demonstrates how to visualize the computational graph in a neural network model. This will help you to understand what will be executed at runtime and how the various transform_graph operations affect the structure of the model in practice.

The simplest way to visualize graphs is to use TensorBoard. To install TensorBoard, enter the following on the command line:

```
pip3 install tensorboard
tensorboard --logdir=/tmp/tensorboard
```

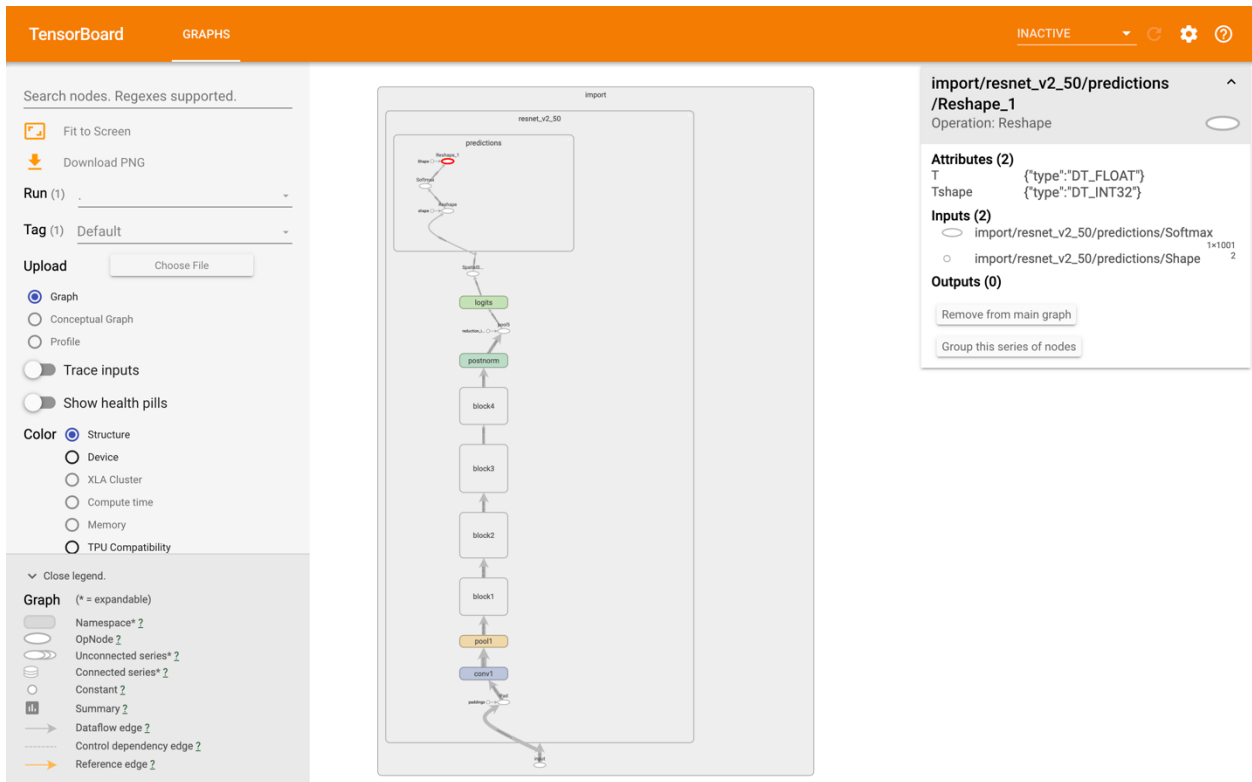
Download resnet_v2_50.pb from [here](#). Use the script provided in the TensorFlow source distribution to import model (.pb) files to TensorBoard by entering the following on the command line:

```
python3 tensorflow_core/python/tools/import_pb_to_tensorboard.py --model_dir
resnet_v2_50.pb --log_dir /tmp/tensorboard
tensorboard --logdir=/tmp/tensorboard
```

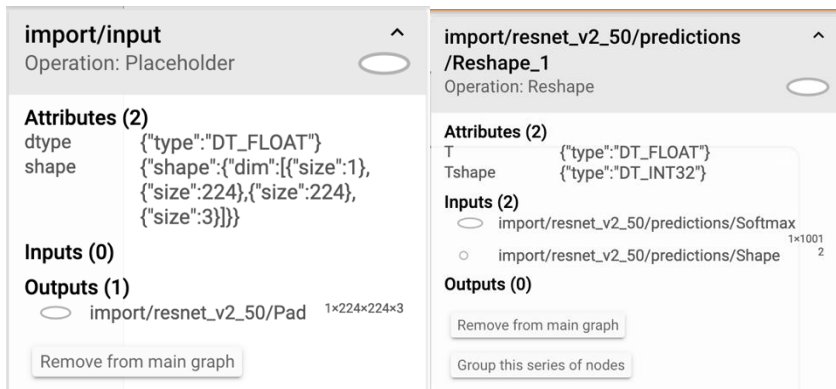
Do not let the name of the argument model_dir confuse you. A .pb file is an acceptable target.

Important: If you repeat this command for importing multiple models, empty the /tmp/tensorboard directory after each import to prevent confusion.

In a browser, navigate to <http://localhost:6006/> and select the GRAPHS tab to see the model's graph. If it is collapsed under a single node, as shown in this image, use the expand control until you get to the actual operations.



In this network, we can see that Placeholder is the only input node and Reshape_1 is the output node. To get their full names, select them and look at the details box on the right, as this image shows.



Here, the name of the input node is import/input/Placeholder and the output node is import/resnet_v2_50/predictions/Reshape_1.

The details box also shows that this model has been trained with the standard 224x224x3 input size which is typical for a ResNet architecture. The transform_graph tool assumes the import namespace, so these are simplified to:

- **Input name:** Placeholder.
- **Input dimensions:** 1x224x224x3.

- **Output name:** resnet_v2_50/predictions/Reshape_1.

We can also use the `summarize_graph` tool to inspect the model and provide guesses about likely input and output nodes, as well as other information that is useful for debugging. Here is an example of how to use it on the `resnet_v2_50` graph:

```
$ bazel build tensorflow/tools/graph_transforms:summarize_graph
$ bazel-bin/tensorflow/tools/graph_transforms/summarize_graph --
in_graph=resnet_v2_50.pb
Found 1 possible inputs: (name=input, type=float(1), shape=[1,224,224,3])
No variables spotted.
Found 1 possible outputs: (name=resnet_v2_50/predictions/Reshape_1, op=Reshape)
Found 25615936 (25.61M) const parameters, 0 (0) variable parameters, and 0
control_edges
Op types used: 328 Const, 272 Identity, 147 Mul, 114 Add, 54 Conv2D, 49 Relu, 49 Rsqrt,
49 Sub, 22 BiasAdd, 4 MaxPool, 4 Pad, 2 Reshape, 1 Mean, 1 Placeholder, 1 Softmax, 1
Squeeze
To use with:
tensorflow/tools/benchmark:benchmark_model
try these arguments:
bazel run tensorflow/tools/benchmark:benchmark_model -- --graph=resnet_v2_50.pb --
show_flops --input_layer=input --input_layer_type=float --input_layer_shape=1,224,224,3
--output_layer=resnet_v2_50/predictions/Reshape_1
```

5 Generate an optimized 32-bit model

This step primarily removes unnecessary nodes and ensures that the operations that are used are available in the TensorFlow distributions on mobile devices. One way it does this is by removing training-specific operations in the model's computational graph.

To generate your model, enter the following on the command line:

```
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \  
--in_graph=resnet_v2_50.pb \  
--out_graph=optimized_resnet_v2_50_fp32.pb \  
--inputs='Placeholder' \  
--outputs='resnet_v2_50/predictions/Reshape_1' \  
--transforms='strip_unused_nodes(type=float, shape="1,224,224,3")  
fold_constants(ignore_errors=true)  
fold_batch_norms  
fold_old_batch_norms'
```

The input and output names depend on your model. The shape is also included here as part of the `strip_unused_nodes` command.

If you encounter any problems, the [TensorFlow documentation](#) covers this step in more detail.

The transformed model does not differ greatly in size or speed from the training model. The difference is that it is capable of being loaded by the TensorFlow distribution for mobile devices, which may not implement all training operators.

It is a good idea to verify that this inference-ready model runs with the same accuracy as your trained one. How you do this depends on your training/test workflow and datasets.

You can now distribute this model and deploy it with TensorFlow on both mobile and embedded or low-power Linux devices.

6 Generate an optimized 8-bit model

Most trained models use 32-bit floating-point numbers to represent their weights. Research has shown that for many networks you can reduce the precision of these numbers and store them in 8-bit integers, reducing the model size by a factor of 4. This has benefits when distributing and loading models on mobile devices.

In theory, 8-bit integer models can also execute 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. However, not all commonly used layers have 8-bit implementations in TensorFlow 1.4. This means that a quantized model may spend more time converting data between 8-bit and 32-bit formats for different layers than it saves through faster execution.

The optimal balance between speed, size, and accuracy vary by model, application and hardware platform. So, it is best to quantize all deployed models and compare them to the unquantized version on the deployment hardware itself. You can quantize a neural network using TensorFlow's `graph_transforms` tool with the following command:

```
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \  
--in_graph=resnet_v2_50.pb \  
--out_graph=optimized_resnet_v2_50_int8.pb \  
--inputs='Placeholder' \  
--outputs='resnet_v2_50/predictions/Reshape_1' \  
--transforms='  
  add_default_attributes  
  strip_unused_nodes(type=float, shape="1,224,224,3")  
  remove_nodes(op=Identity, op=CheckNumerics)  
  fold_constants(ignore_errors=true)  
  fold_batch_norms  
  fold_old_batch_norms  
  quantize_weights  
  quantize_nodes  
  strip_unused_nodes  
  sort_by_execution_order'
```

The quantized network should be significantly smaller than the trained model. In this case, `optimized_resnet_v2_50_fp32.pb` is 102MB whereas `optimized_resnet_v2_50_int8.pb` is 27MB. This can also be important if the model is distributed as part of a mobile application, quite apart from any inference speed improvements. If size is the primary concern, read the [TensorFlow documentation](#) and try the `round_weights` transform that reduces the size of the compressed model for deployment without improving speed or affecting accuracy.

There are several variants of this step, which can include performing extra fine-tuning passes on the model or instrumented runs to determine quantization ranges. To learn more about this and the corresponding deployment workflow with TensorFlow Lite see the [TensorFlow Quantization Guide](#).

7 Benchmark the optimized models

It is important to benchmark these models on real hardware. TensorFlow contains optimized 8-bit routines for Arm CPUs but not for x86, so 8-bit models perform much slower on an x86-based laptop than a mobile Arm device. Benchmarking varies by platform; on Android you can build the TensorFlow benchmark application with this command:

```
bazel build -c opt --cxxopt='--std=c++11' --config=android_arm  
tensorflow/tools/benchmark:benchmark_model
```

With the Android deployment device connected (this example uses a HiKey 960), run:

```
adb shell "mkdir -p /data/local/tmp"  
adb push bazel-bin/tensorflow/tools/benchmark/benchmark_model /data/local/tmp  
adb push optimized_resnet_v2_50_fp32.pb /data/local/tmp  
adb push optimized_resnet_v2_50_int8.pb /data/local/tmp
```

The benchmarks are run on a single core (num_threads=1) or four cores (num_threads=4) with these commands:

```
adb shell '/data/local/tmp/benchmark_model \  
--num_threads=1 \  
--graph=/data/local/tmp/optimized_resnet_v2_50_fp32.pb \  
--input_layer="Placeholder" \  
--input_layer_shape="1,224,224,3" \  
--input_layer_type="float" \  
--output_layer="resnet_v2_50/predictions/Reshape_1"'  
adb shell '/data/local/tmp/benchmark_model \  
--num_threads=1 \  
--graph=/data/local/tmp/optimized_resnet_v2_50_int8.pb \  
--input_layer="Placeholder" \  
--input_layer_shape="1,224,224,3" \  
--input_layer_type="float" \  
--output_layer="resnet_v2_50/predictions/Reshape_1"'
```

Alternatively, deploy the models directly in your application, on iOS, Linux or Android, and use real-world performance measurements to compare the models.

Accuracy should always be evaluated using your own data, as the impact of quantization on accuracy can vary. In terms of compute performance on our HiKey 960 development platform, we see the following:

Model type	Processor	Model size	1-batch inference
32-bit floating point	Arm Cortex-A73	97MB	1794ms
8-bit integer	Arm Cortex-A73	25MB	935ms
32-bit floating point	4x Arm Cortex-A73	102MB	583ms
8-bit integer	4x Arm Cortex-A73	27MB	524ms

7.1 Improving model performance

ResNet-50 on a 224x224x3 image uses around 7 billion operations per inference. It is worth considering whether your application requires a high resolution for fine details in the input, as running ResNet-50 on a 160x160 image would almost halve the number of operations and double the speed. For even faster inference of image-processing workloads, investigate performance-optimized models such as the [MobileNet family of networks](#). The MobileNet family allows you to scale computation down by a factor of a thousand, enabling you to scale the model to meet a wide range of FPS targets on existing hardware for a modest accuracy penalty.

8 Deploy the optimized models

The exact deployment method depends on your platform. Use the links in the table to access resources for deploying on your platform:

Platform	Deployment method
Android	TensorFlow Java or C++ interface
iOS	CoreML converter (no 8-bit)
iOS	TensorFlow C++ interface
Linux	TensorFlow C++ interface

9 Related information

Here are some resources related to material in this guide:

- [Graph Transform Tool](#)
- [Machine Learning on Arm](#)
- [TensorFlow Model Optimization toolkit](#)

10 Next steps

This guide demonstrates how to use TensorFlow Graph Transform Tool to optimize a frozen TF graph before deploying it in production. There are different types of optimizations. One is to make the model smaller and faster in size without accuracy loss. And the other is to change the weights from higher precision to lower precisions, usually from FP32 to FP16 or INT8. For deploying your model to a phone or embedded device, you can optimize away batch normalization or other training-only features. The TensorFlow Graph Transform framework offers a suite of tools for modifying computational graphs, and a framework to make it easy to write your own modifications.

Error! No text of specified style in document.
Error! No text of specified style in document.