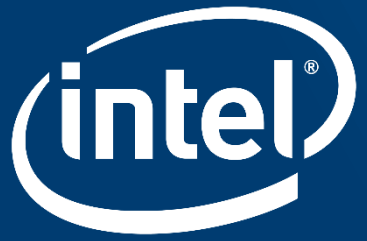


BUILDING IMAGE CLASSIFIER USING INTEL® NEURAL COMPUTE STICK 2 AND OPENVINO™ TOOLKIT

LEGAL DISCLAIMER

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

This sample source code is released under the Intel Sample Source Code License Agreement.

Intel, the Intel logo, and OpenVINO are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

AGENDA

Chapter Outcome: Deploy an Image Classifier Model on Intel® Neural Compute Stick 2

Part one

- What is Image Classifier?
- Popular image classification topologies
- OpenVINO™ toolkit support for image classification

Part Two

- Deep dive into OpenVINO toolkit
  - Learn how to create hardware-agnostic intermediate representation (IR) using Model Optimizer
  - Learn how to create FP16 quantization to work with Intel Neural Compute Stick 2 (Intel® NCS2) using pretrained Inception model
  - Learn how to deploy on NCS2 through the Inference Engine

PART 1: LEARN THE BASICS OF IMAGE CLASSIFICATION AND COMMON TOPOLOGIES

IMAGE CLASSIFICATION

WHAT IS IMAGE CLASSIFICATION?

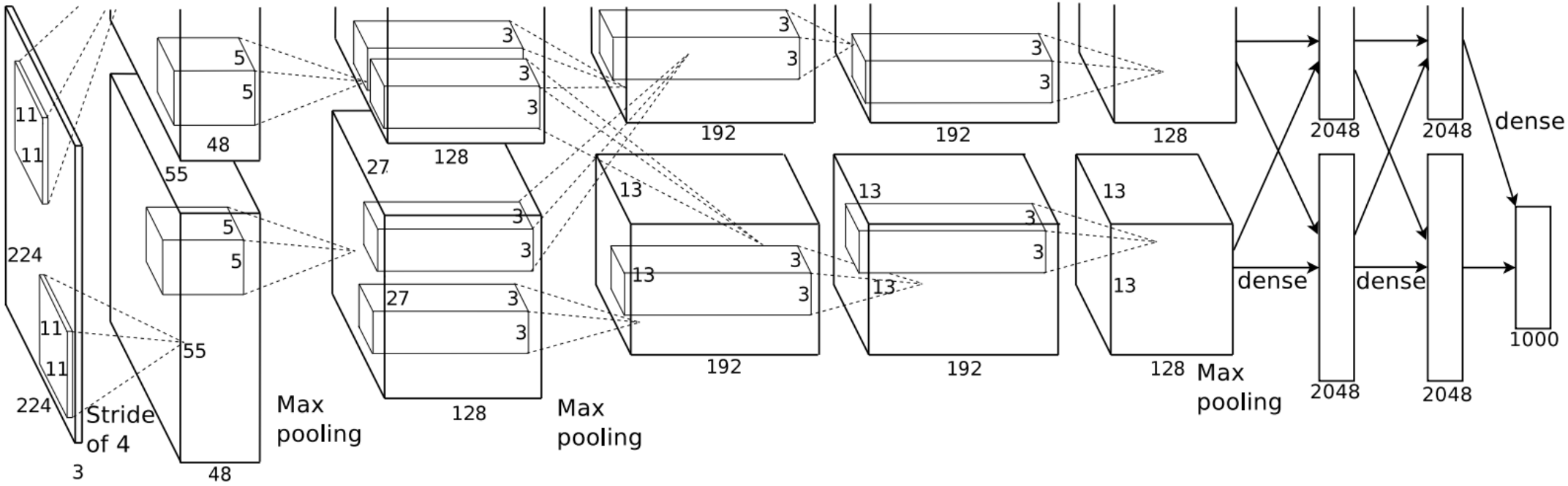
Image source:
https://commons.wikimedia.org/wiki/File:US_Navy_100310-N-4178C-005_Three-year-old_Joey_Adams_identifies_items_from_flash_cards_during_an_in-home_therapy_session._Adams.jpg

IMAGE CLASSIFICATION

- Image classification is a computer vision problem that aims to classify a subject or an object present in an image into predefined classes.
- Traditional approaches to providing such visual perception to machines have relied on complex computer algorithms that use feature descriptors, like edges, corners, colors, and so on, to identify or recognize objects in the image.
- Deep learning takes a rather interesting, and by far most efficient approach, to solving real-world imaging problems. It uses multiple layers of interconnected neurons, where each layer uses a specific computer algorithm to identify and classify a specific descriptor.

POPULAR IMAGE CLASSIFICATION TOPOLOGIES

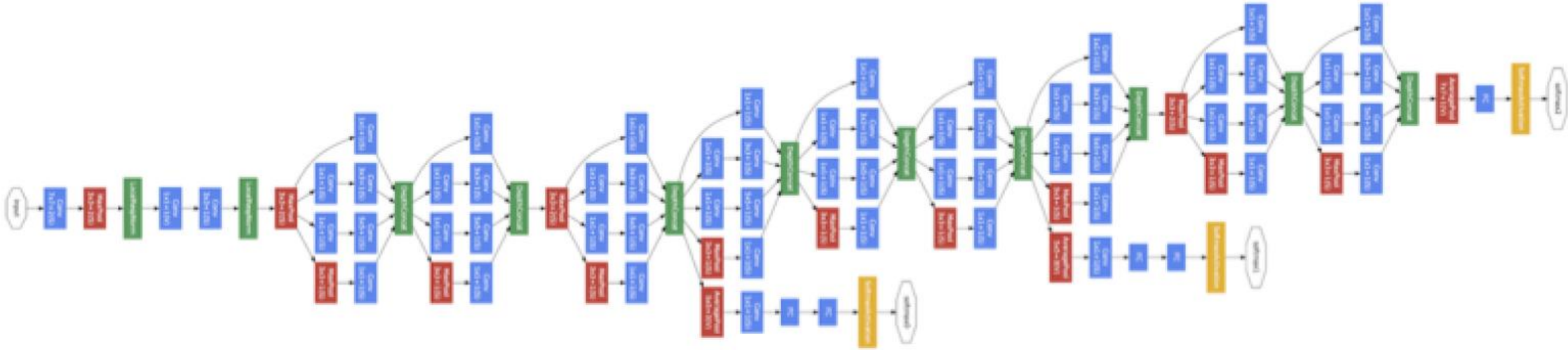
AlexNet



ALEXNET

- Created in 2012 for the ImageNet Large Scale Visual Recognition Challenge
- Task: Predict the correct label from among 1000 classes
- Dataset: Around 1.2 million images
- Considered the “flash point” for modern deep learning
- Demolished the competition
- Top five error rate of 15.4%
- Next best: 26.2%

GOOGLENET

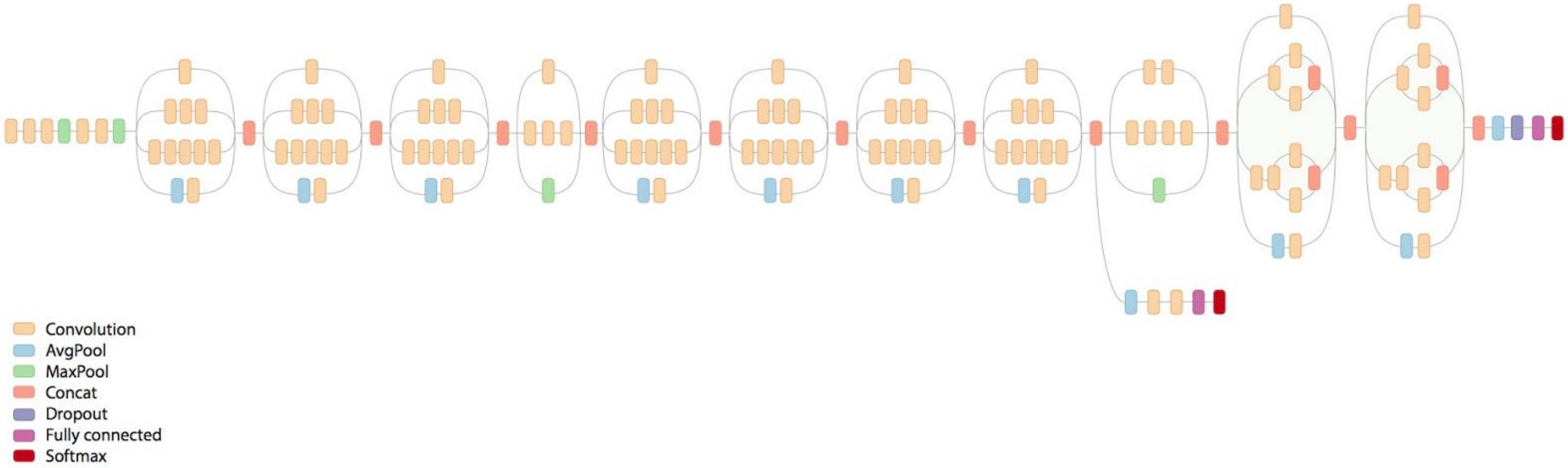


Convolution
Pooling
Softmax
Other

GOOGLENET

- The winner of the ImageNet Large Scale Visual Recognition Challenge 2014 competition was GoogLeNet (also known as Inception V1) from Google
- The module is based on several very small convolutions in order to drastically reduce the number of parameters****
- Their architecture consisted of a 22-layer deep convolutional neural network but reduced the number of parameters from 60 million (AlexNet) to 4 million
- It achieved a top-five error rate of 6.67%

INCEPTION V3 SCHEMATIC



INCEPTION

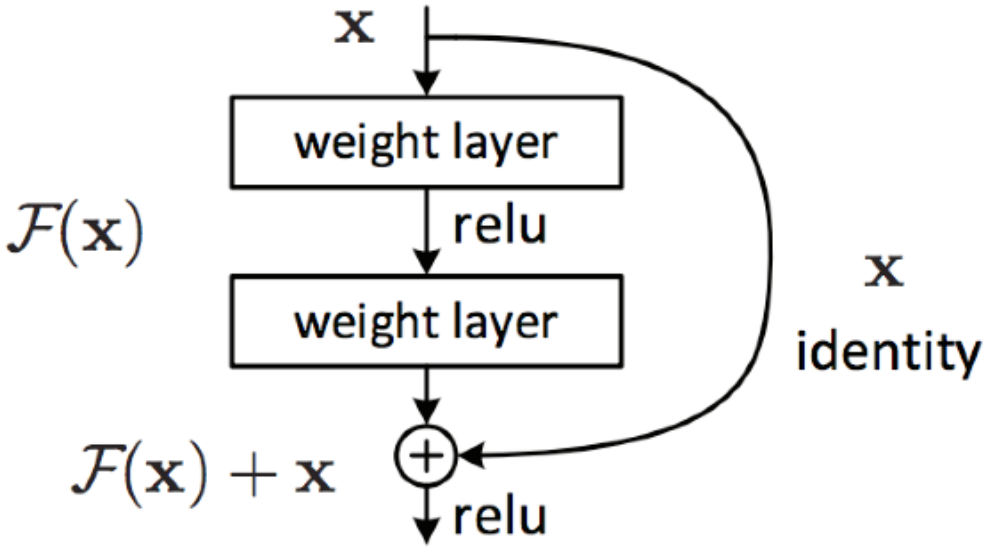
- Szegedy, et al. 2014
- Idea: network would want to use different receptive fields
- Want computational efficiency
- Also want to have sparse activations of groups of neurons
- Hebbian principle: “Fire together, wire together”
- Solution: Turn each layer into branches of convolutions
- Each branch handles smaller portion of workload
- Concatenate different branches at the end

RESNET-50

- ResNet* stands for Residual Network
- The number indicates the number of layers in the network. For example: ResNet-34, ResNet-50, ResNet-152, and so on
- Works by flowing information in earlier layers in the network to later layers—a concept known as skip connection
  - In deep neural networks, this mitigates the problem of vanishing gradients

RESNET* ARCHITECTURE

- Assumption: Best transformation over multiple layers is close to $\mathcal{F}(x)+x$
- x->input to series of layers
- $\mathcal{F}(x)$->function represented by several layers (such as convs)
- Enforce this by adding “shortcut connections”
- Add the inputs from an earlier layer to the output of current layer



SUPPORTED IMAGE CLASSIFICATION TOPOLOGIES

SUPPORTED IMAGE CLASSIFICATION TOPOLOGIES

Most common Image Classification topologies are supported through the OpenVINO™ toolkit.

- Inception v1- v4
- Inception ResNet* v2
- MobileNet* v1-128
- MobileNet v1-160
- MobileNet v1- 224
- VGG-16
- VGG-19
- ResidualNet-50 v1and v2
- ResidualNet-101 v1 and v2
- ResidualNet-152 v1 and v2

For a complete list of all supported topologies, refer to the OpenVINO™ Toolkit documentation

PART 2: LEARN HOW TO USE THE OPENVINO™ TOOLKIT TO DEPLOY IMAGE CLASSIFIERS

WORKFLOW—DEPLOY IMAGE CLASSIFIER USING OPENVINO™ TOOLKIT

Step 1: Freeze the TensorFlow* model

- If your model is not already frozen, convert it to a frozen graph format (.pb file)

Step 2: Create intermediate representation (IR) files through the Model Optimizer

- Based on the trained network topology, weights, and biases values, generate the .bin and .xml files

Step 3: Test the model with the intermediate representation files

- Use the Inference Engine in the target environment via provided sample applications
- **We will focus on deployment to the Intel® Neural Compute Stick 2 (Intel® NCS2)**

Step 4: Integrate the Inference Engine in your application

- Deploy the model in the target environment (CPU, processor graphics, Intel NCS2, field-programmable gate array (FPGA))
- We will cover this in Chapter 5 when we learn how to deploy custom models

STEP 1: FREEZE TENSORFLOW* MODEL

This step is required to export the inference graph—the protobuf (.pb) file, which contains the network architecture

For the sake of simplicity, we will start with a pretrained model example from the TensorFlow Slim Repo: inception_v1

(To create a frozen graph for a custom model, refer to chapter 5)

Instructions

- Download the TensorFlow-Slim models git repository

```
mkdir tf_models
cd tf_models
git clone https://github.com/tensorflow/models.git
```

- Download and extract the checkpoint file

```
wget -nc http://download.tensorflow.org/models/inception_v1_2016_08_28.tar.gz
tar -xvf inception_v1_2016_08_28.tar.gz
```

STEP 1: FREEZE TENSORFLOW* MODEL

Export the inference graph

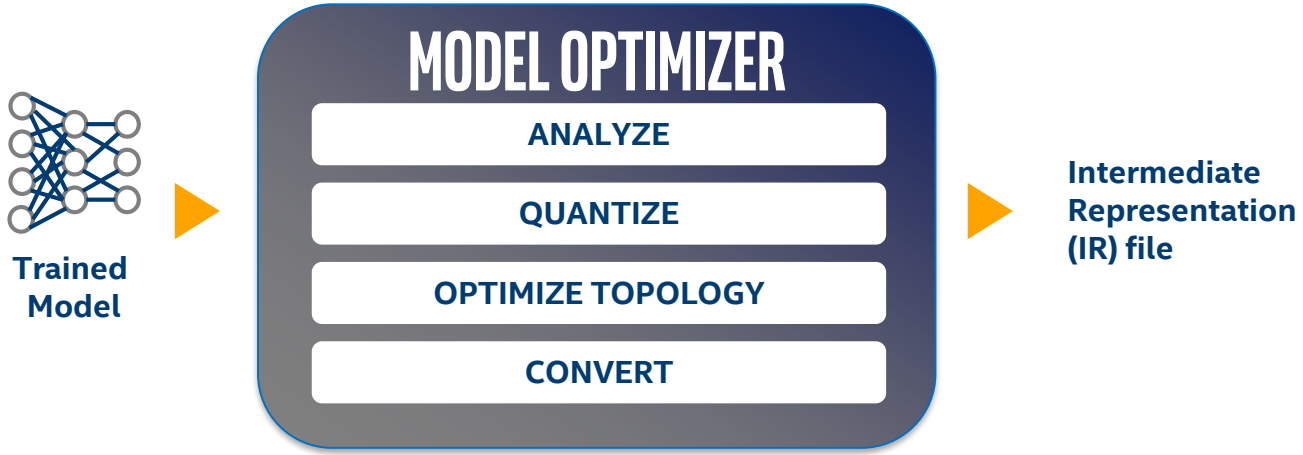
```
python3 models/research/slim/export_inference_graph.py \
        --alsologtostderr \
        --model_name=inception_v1 \
        --output_file=inception_v1.pb
```

The output is a frozen graph with the name specified in the –output_file parameter

STEP 2: CREATE INTERMEDIATE REPRESENTATION FILES USING MODEL OPTIMIZER

The Model Optimizer creates hardware-agnostic IR files

- Takes as input the inference graph, checkpoint file, and required output quantization and creates two output files
  - Bin file contains the weights
  - Xml file contains the network architecture



STEP 2: CREATE INTERMEDIATE REPRESENTATION FILES USING MODEL OPTIMIZER

Instructions

Use the Inference graph and checkpoint files from Step 1 for Inception V1 and create intermediate representation (IR) files in FP16 format

```
<MODEL_OPTIMIZER_INSTALL_DIR>/mo_tf.py --input_model ./inception_v1.pb --input_checkpoint ./inception_v1.ckpt -b 1 --mean_value [127.5,127.5,127.5] --scale 127.5 –data_type FP16 –model_name inception_v1_FP16
```

- --input_model: Frozen graph file in protobuf (.pb) format
- --input_checkpoint: Checkpoint file from Step 1
- --mean_value and --scale: Mean and scale values for the topology
  - If you are using other topologies, refer here for mean and scale values
- --data_type: Required precision for the IR. Since we intend to deploy on Intel® Neural Compute Stick 2 (Intel® NCS2), we set this to FP16
- --model_name: Name of the output files. Inception_v1_FP16.xml and Inception_v1_Fp16.bin can be found in the current directory

STEP 3: TEST THE IR FILES WITH SAMPLE APPS

The OpenVINO™ toolkit installation comes with many sample applications that integrate the Inference Engine capabilities for demonstration

Build these samples using the OpenVINO toolkit installation instructions (covered in Chapter 2)

To test the IR created in Steps 1–3, we will use the classification_sample_async

Instructions

- The executable samples will be found in the <INFERENCE_ENGINE_SAMPLES_BUILD_DIR>/intel64/Release directory
- Run the following command to execute the sample application. We set the device to MYRIAD (using the –d flag) to deploy to Intel® Neural Compute Stick 2 (Intel® NCS2)

```
<INFERENCE_ENGINE_SAMPLES_BUILD_DIR>/intel64/Release$./classification_sample -i
<path_to_image>/cat.bmp -m <path_to_model>/ inception_v1_FP16.xml –d MYRIAD
```

STEP 3: TEST THE IR FILES WITH SAMPLE APPS

- The sample application requires IR input for models that have a single output (Example: AlexNet, GoogLeNet, Inception V1)
- The sample verifies the results against the ImageNet dataset
- The sample application integrates the Inference Engine plugins to infer on the specified target
- Sample output lists the top 10 labels with the highest probability

```
:~/inference_engine_samples_build/intel64/Release$ ./classification_sample
_async -i ../../classification_sample_async/cat.bmp -m /home/meghana/tf_models/inception_v1_FP16.xml
-d MYRIAD
```

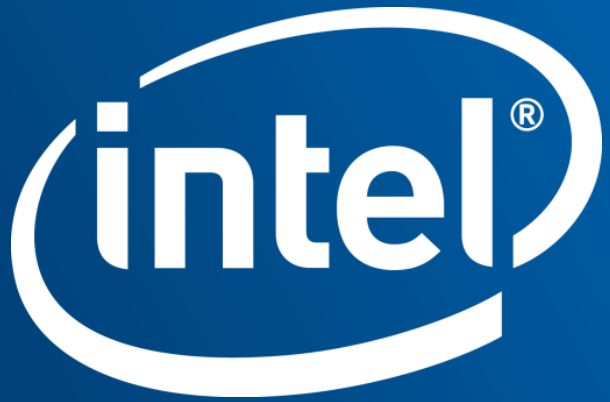
```
Top 10 results:

Image ../../classification_sample_async/cat.bmp

classid probability
------- -----------
286     0.9612547
288     0.0109824
282     0.0079575
283     0.0026535
165     0.0019332
211     0.0011239
213     0.0009525
966     0.0003393
290     0.0003125
216     0.0002165

[ INFO ] Execution successful
```

In Chapter 5, we will show you how to use a custom topology with the OpenVINO™ toolkit


intel