Intel Edge AI Foundations - Lesson 3

Model Optimizer:
The **Model Optimizer** helps convert models in multiple different frameworks to **an Intermediate Representation**, which is used with the **Inference Engine.** If a model is not one of the pre-converted models in the Pre-Trained Models OpenVINO™ provides, it is a required step to move onto the Inference Engine.

As part of the process, it can perform various optimizations that can help shrink the model size and help make it faster, although this will not give the model higher inference accuracy. In fact, there will be some loss of accuracy as a result of potential changes like lower precision. However, these losses in accuracy are minimized.

Optimization Techniques:
Quantization, Freezing and Fusion.

Quantization:
Quantization is related to the topic of precision I mentioned before, or how many bits are used to represent the weights and biases of the model. During training, having these very accurate numbers can be helpful, but it's often the case in inference that the precision can be reduced without substantial loss of accuracy. Quantization is the process of reducing the precision of a model.

With the OpenVINO™ Toolkit, models usually default to FP32, or 32-bit floating point values, while FP16 and INT8, for 16-bit floating point and 8-bit integer values, are also available (INT8 is only currently available in the Pre-Trained Models; the Model Optimizer does not currently support that level of precision). FP16 and INT8 will lose some accuracy, but the model will be smaller in memory and compute times faster. Therefore, quantization is a common method used for running models at the edge.

Freezing:
Freezing in this context is used for TensorFlow models. Freezing TensorFlow models will remove certain operations and metadata only needed for training, such as those related to backpropagation. Freezing a TensorFlow model is usually a good idea whether before performing direct inference or converting with the Model Optimizer.

Fusion:
Fusion relates to combining multiple layer operations into a single operation. For example, a batch normalization layer, activation layer, and convolutional layer could be combined into a single operation. This can be particularly useful for GPU inference, where the separate operations may occur on separate GPU kernels, while a fused operation occurs on one kernel, thereby incurring less overhead in switching from one kernel to the next.

The supported frameworks with the OpenVINO™ Toolkit are:
Caffe
TensorFlow
MXNet
ONNX (which can support PyTorch and Apple ML models through another conversion step)
Kaldi

Intermediate Representations:
Intermediate Representations (IRs) are the OpenVINO™ Toolkit's standard structure and naming for neural network architectures. A Conv2D layer in TensorFlow, Convolution layer in Caffe, or Conv layer in ONNX are all converted into a Convolution layer in an IR.

The IR is able to be loaded directly into the Inference Engine, and is actually made of two output files from the Model Optimizer: an **XML file and a binary file**. T**he XML file holds the model architecture and other important metadata**, **while the binary file holds weights and biases in a binary format.** You need both of these files in order to run inference Any desired optimizations will have occurred while this is generated by the Model Optimizer, such as changes to precision. You can generate certain precisions with the **--data_type** argument, which is usually FP32 by default.

Tensorflow:
TensorFlow model is to determine whether to use a **frozen** or **unfrozen model.**

Unfrozen models usually need the **--mean_values** and **--scale** parameters fed to the **Model Optimizer,** while the **frozen models from the Object Detection Model Zoo don't need those parameters.**

For Example, Object detection model zoo needs below parameter,
--tensorflow_use_custom_operations_config
--tensorflow_object_detection_api_pipeline_config
--reverse_input_channels : Is usually needed, as TF model zoo models are trained on RGB images, while OpenCV usually loads as BGR.
Certain models, like YOLO, DeepSpeech, and more, have their own separate pages.

Q: How do we decide the custom operation configuration based on a network? how to write a custom configuration for custom network, how do we decide those configuration parameters which operations comes under custom configurations?
--tensorflow_object_detection_api_pipeline_config <path_to_pipeline.config> — A special configuration file that describes the topology hyper-parameters and structure of the TensorFlow Object Detection API model. For the models downloaded from the TensorFlow* Object Detection API zoo, the configuration file is named pipeline.config has to pass to the --tensorflow_object_detection_api_pipeline_config parameter.**But, why we need this?**

The Tensorflow Object Detection API uses protobuf files to configure the training and evaluation process. The schema for the training/eval pipeline can be found in object_detection/protos/pipeline.proto. At a high level, the config file is split into 5 parts:The model configuration. This defines what type of model will be trained (ie. meta-architecture, feature extractor).
The train_config, which decides what parameters should be used to train model parameters (ie. SGD parameters, input preprocessing and feature extractor initialization values).
The eval_config, which determines what set of metrics will be reported for evaluation.
The train_input_config, which defines what dataset the model should be trained on.
The eval_input_config, which defines what dataset the model will be evaluated on. Typically this should be different from the training input dataset. We can find all the Tensorflow object detection config file in the below location,
https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs

Example:
Frozen - SSD:

```
wget http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz
cd ssd_mobilenet_v2_coco_2018_03_29/
python /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model frozen_inference_graph.pb --
tensorflow_object_detection_api_pipeline_config pipeline.config --reverse_input_channels --tensorflow_use_custom_operations_config
/opt/intel/openvino/deployment_tools/model_optimizer/extensions/front/tf/ssd_v2_support.json
```

Unfrozen - SSD:
In this case, a model has meta file,

```
python /opt/intel/openvino/deployment_tools/model_optimizer/mo_tf.py  --input_meta_graph model.ckpt.meta --
tensorflow_object_detection_api_pipeline_config pipeline.config --reverse_input_channels --tensorflow_use_custom_operations_config
/opt/intel/openvino/deployment_tools/model_optimizer/extensions/front/tf/ssd_v2_support.json
```

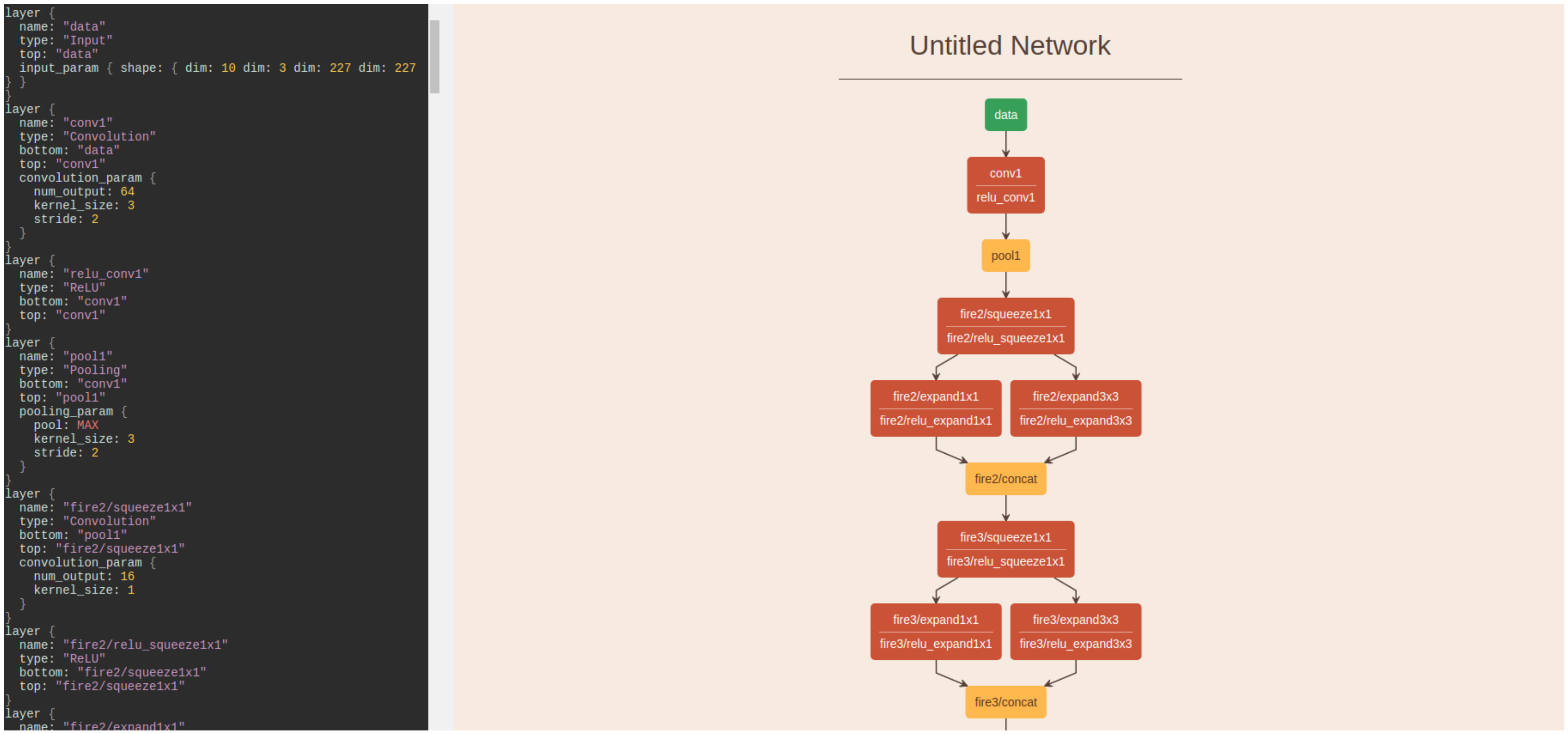
Caffe:
Caffe models need to feed both the **.caffemodel file**, as well as a **.prototxt fil**e, into the Model Optimizer.
Note:
If they have the same name, only the model needs to be directly input as an argument, while if the .prototxt file has a different name than the model, it should be fed in with --input_proto as well.

Example:

```
git clone https://github.com/DeepScale/SqueezeNet
```

```
python /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model squeezenet_v1.1.caffemodel --input_proto
deploy.prototxt
```

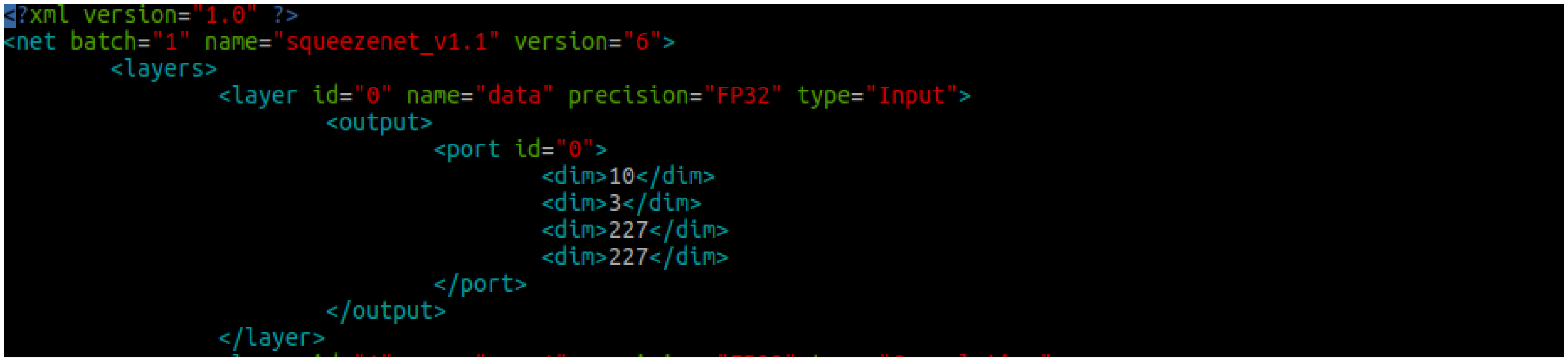
Model Optimizer advanced:
Caffe topology can be visualize by using netscope.
https://ethereon.github.io/



```
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 10 dim: 3 dim: 227 dim: 227
} }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu_conv1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "fire2/squeeze1x1"
  type: "Convolution"
  bottom: "pool1"
  top: "fire2/squeeze1x1"
  convolution_param {
    num_output: 16
    kernel_size: 1
  }
}
layer {
  name: "fire2/relu_squeeze1x1"
  type: "ReLU"
  bottom: "fire2/squeeze1x1"
  top: "fire2/squeeze1x1"
}
layer {
  name: "fire2/expand1x1"
```

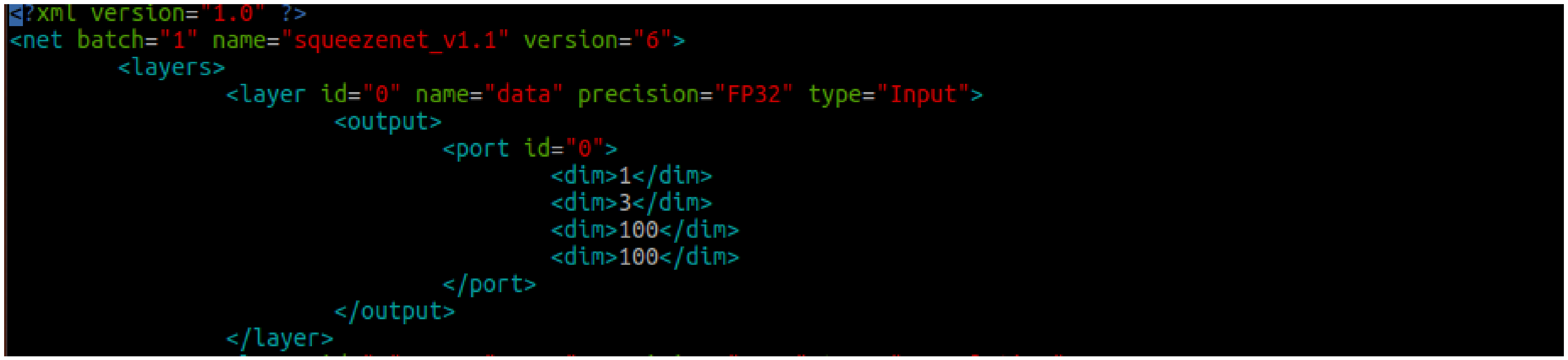
Reshape:
In SqueezeNet,first layer is an input layer and it expects the data to be size of 1 * 3 * 227 * 227. This data will passed to Convolution layer then RELU activation. In case I want to use input images with a different size I do not need to train a new model. The model-optimizer can manipulate the sizes of the input.

```
#Before Reshape
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model squeezenet_v1.1.caffemodel
```



```
<?xml version="1.0" ?>
<net batch="1" name="squeezenet_v1.1" version="6">
        <layers>
                <layer id="0" name="data" precision="FP32" type="Input">
                        <output>
                                <port id="0">
                                        <dim>10</dim>
                                        <dim>3</dim>
                                        <dim>227</dim>
                                        <dim>227</dim>
                                </port>
                        </output>
                </layer>
```

```
# After Reshape
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model squeezenet_v1.1.caffemodel --input_shape
[1,3,100,100]
```

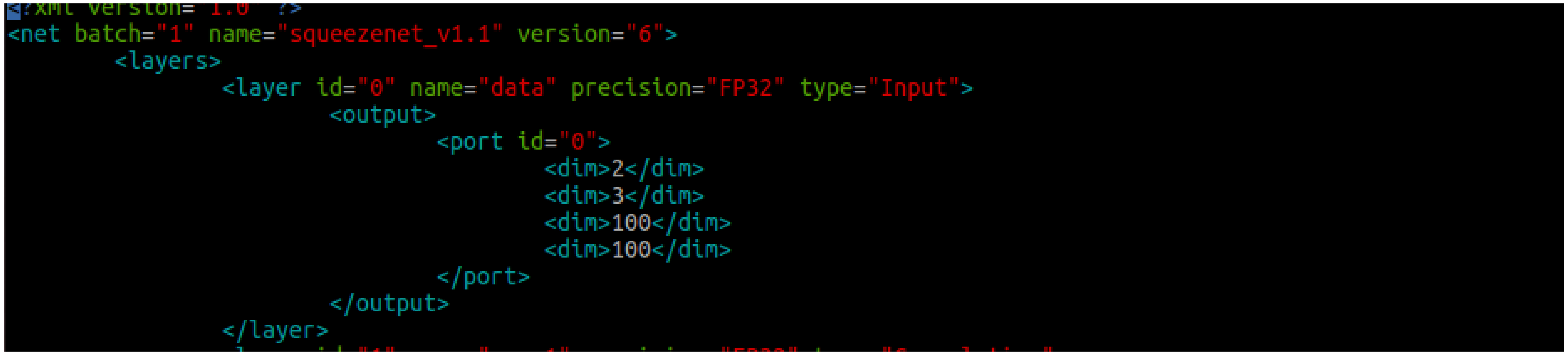


```
<?xml version="1.0" ?>
<net batch="1" name="squeezenet_v1.1" version="6">
        <layers>
                <layer id="0" name="data" precision="FP32" type="Input">
                        <output>
                                <port id="0">
                                        <dim>1</dim>
                                        <dim>3</dim>
                                        <dim>100</dim>
                                        <dim>100</dim>
                                </port>
                        </output>
                </layer>
```

Model Optimizer advanced:
Batch size:

```
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model squeezenet_v1.1.caffemodel --input_shape
[2,3,100,100]
or
```

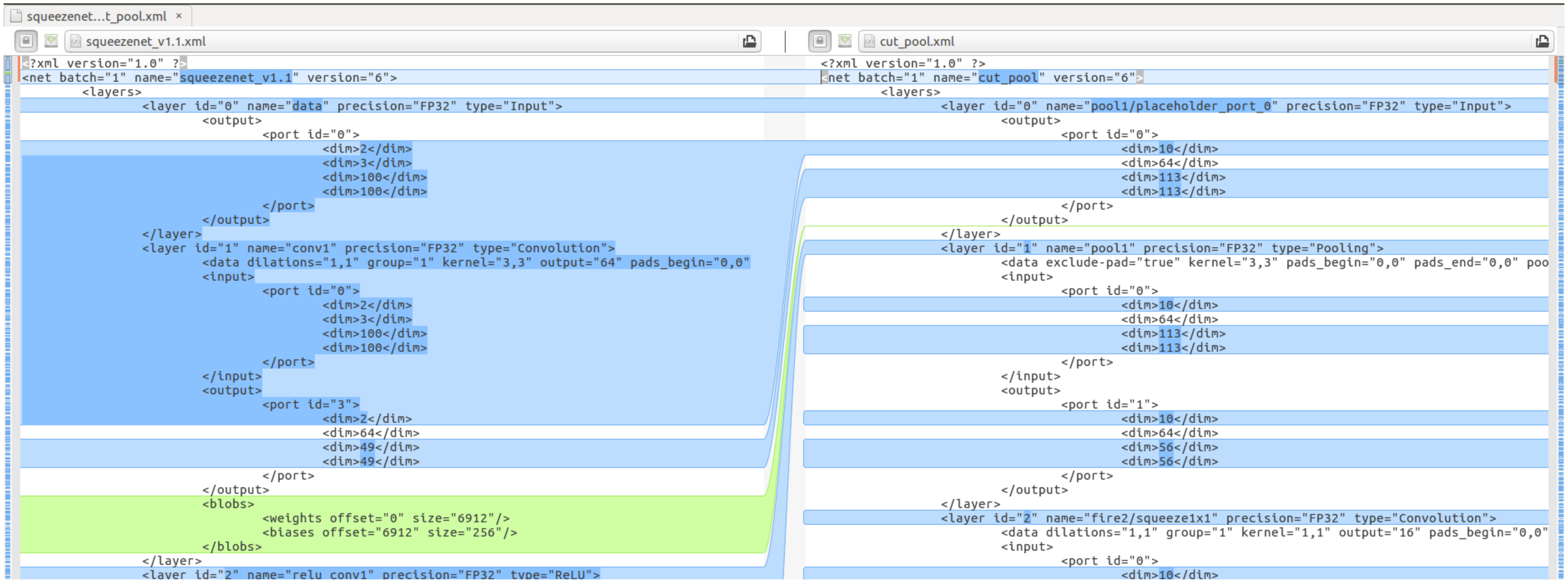
```
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model squeezenet_v1.1.caffemodel --batch 4
```



```
<?xml version="1.0" ?>
<net batch="1" name="squeezenet_v1.1" version="6">
        <layers>
                <layer id="0" name="data" precision="FP32" type="Input">
                        <output>
                                <port id="0">
                                        <dim>2</dim>
                                        <dim>3</dim>
                                        <dim>100</dim>
                                        <dim>100</dim>
                                </port>
                        </output>
                </layer>
```

Cut network:

```
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model squeezenet_v1.1.caffemodel --model_name cut_pool --
input pool1
```



```
<?xml version="1.0" ?>
<net batch="1" name="squeezenet_v1.1" version="6">
        <layers>
                <layer id="0" name="data" precision="FP32" type="Input">
                        <output>
                                <port id="0">
                                        <dim>2</dim>
                                        <dim>3</dim>
                                        <dim>100</dim>
                                        <dim>100</dim>
                                </port>
                        </output>
                </layer>
                <layer id="1" name="conv1" precision="FP32" type="Convolution">
                        <data dilations="1,1" group="1" kernel="3,3" output="64" pads_begin="0,0"
                        <input>
                                <port id="0">
                                        <dim>2</dim>
                                        <dim>3</dim>
                                        <dim>100</dim>
                                        <dim>100</dim>
                                </port>
                        </input>
                        <output>
                                <port id="3">
                                        <dim>2</dim>
                                        <dim>64</dim>
                                        <dim>49</dim>
                                        <dim>49</dim>
                                </port>
                        </output>
                        <blobs>
                                <weights offset="0" size="6912"/>
                                <biases offset="6912" size="256"/>
                        </blobs>
                </layer>
                <layer id="2" name="relu_conv1" precision="FP32" type="ReLU">
```

```
<?xml version="1.0" ?>
<net batch="1" name="cut_pool" version="6">
        <layers>
                <layer id="0" name="pool1/placeholder_port_0" precision="FP32" type="Input">
                        <output>
                                <port id="0">
                                        <dim>10</dim>
                                        <dim>64</dim>
                                        <dim>113</dim>
                                        <dim>113</dim>
                                </port>
                        </output>
                </layer>
                <layer id="1" name="pool1" precision="FP32" type="Pooling">
                        <data exclude-pad="true" kernel="3,3" pads_begin="0,0" pads_end="0,0" poo
                        <input>
                                <port id="0">
                                        <dim>10</dim>
                                        <dim>64</dim>
                                        <dim>113</dim>
                                        <dim>113</dim>
                                </port>
                        </input>
                        <output>
                                <port id="1">
                                        <dim>10</dim>
                                        <dim>64</dim>
                                        <dim>56</dim>
                                        <dim>56</dim>
                                </port>
                        </output>
                </layer>
                <layer id="2" name="fire2/squeeze1x1" precision="FP32" type="Convolution">
                        <data dilations="1,1" group="1" kernel="1,1" output="16" pads_begin="0,0"
                        <input>
                                <port id="0">
                                        <dim>10</dim>
```

```
<input>
        <port id="0">
                <dim>2</dim>
                <dim>64</dim>
                <dim>49</dim>
                <dim>49</dim>
        </port>
```

```
                <dim>64</dim>
                <dim>56</dim>
                <dim>56</dim>
        </port>
</input>
<output>
        <port id="3">
```

Mean:
A very common practice during training is to use not the images directly, but normalized image values instead. Often data scientists subtract the mean image value from each of the training images that are fed to the model. Subtracting the mean values is easy by the model-optimizer, but you need to know the values.

```
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model squeezenet_v1.1.caffemodel --model_name mean --mean_value data[104,117,123]
```

Scale/normalization:
Data normalization is an important step which ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network. Data normalization is done by subtracting the mean from each pixel and then dividing the result by the standard deviation. The distribution of such data would resemble a Gaussian curve centered at zero. For image inputs we need the pixel numbers to be positive, so we might choose to scale the normalized data in the range [0,1] or [0, 255]. So, If we use normalization during training it necessary to normalize the image during inference.

We use the "scale" option --scale 10, will multiply all the channels by 10 --scale_values can multiply each of the planes, R,G or B by a different factor.
You can also specify where to do the scaling, so you can --scale_values with the layer name.

```
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --input_model squeezenet_v1.1.caffemodel --model_name scale --scale 10
```

Supported Layers:
Layers supported for direct conversion from supported framework layers to intermediate representation layers through the Model Optimizer. While nearly every layer you will ever use is in the supported frameworks is supported, there is sometimes a need for handling Custom Layers.

Custom Layers:
Custom layers are those outside of the list of known, supported layers, and are typically a rare exception. Handling custom layers in a neural network for use with the Model Optimizer depends somewhat on the framework used; other than adding the custom layer as an

extension, you otherwise have to follow instructions specific to the framework.