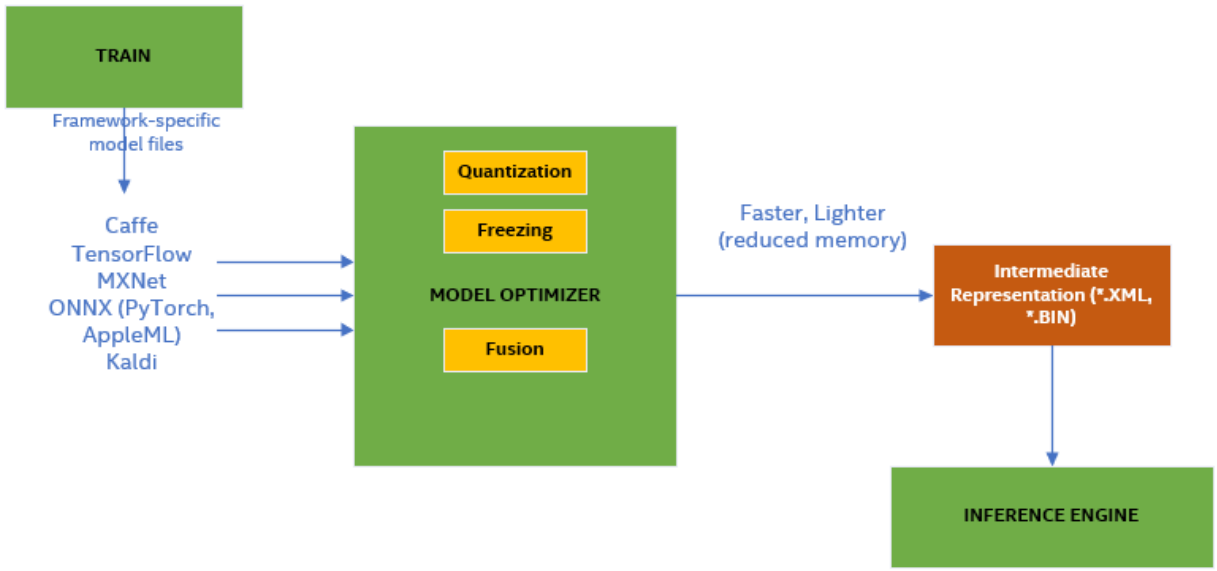
Lesson 3 Notes

Model Optimizer

Converts to Intermediate Representation for use in Inference Engine



Optimization Techniques

Optimization	**Benefit**
Quantization: (reduce precision of model weights and biases with minimal loss in accuracy)	Reduce compute time and size
Freezing: TF-specific removal of metadata needed for training but not for inference like backpropagation metadata	Reduce size, reduce compute time
Fusion: Combine a set of operations into one operation	Reduce computational overhead. Multiple operations running on separate GPU kernels can now run on one kernel, thus eliminating overhead in switching between kernels.

Exercise 7: Convert TF model to IR

To download from an internet link, use *wget*

wget <link>
tar -xvf

python3 mo_tf.py --reverse_input_channels --options from documentation specific for TF Object Model Zoo.

Exercise 10: Convert a Caffe Model

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

Now, we run the model optimizer. So, read the documentation to build the command:

python3 mo.py --input_model <INPUT_MODEL>.caffemodel

From <https://docs.openvinotoolkit.org/2018_R5/_docs_MO_DG_prepare_model_convert_model_Convert_Model_From_Caffe.html#Convert_From_Caffe>

python3 /opt/intel/openvino_2019.3.376/deployment_tools/model_optimizer/mo.py --input_model SqueezeNet/SqueezeNet_v1.1/squeezenet_v1.1.caffemodel --input_proto SqueezeNet/SqueezeNet_v1.1/deploy.prototxt

Exercise 13: Converting an ONNX Model

Much more straightforward, just provide path to model

PyTorch to ONNX: https://michhar.github.io/convert-pytorch-onnx/
Gotta try the above with fast.ai models

Custom Layers:

Sometimes, there are some layers that are in your model that are not supported. The list of supported layers:
https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_Supported_Frameworks_Layers.html

So, in such times, we use Custom Layers. Model Optimizer defines all unsupported layers as Custom Layers, automatically.

Final Exercise: Custom Layers

So, taking an example of a cosine function as a custom layer, the exercise takes us through:

- Building the Model
  Creating Extension Templates for Model Optimizer to convert and Inference Engine to execute the custom layer
- Edit the above templates
  - Model Optimizer extensions would not require modification if the model parameters are used without modification and the shape of input is the same as shape of output
  - Inference Engine extensions would require some modification mainly to define the functionality of the custom layer (exercise example cosine function is defined in a C++ file and a Cmake file)
- Compile with the edits completed
- Test by executing