tinyML® Foundation

Enabling Ultra-low Power Machine Learning at the Edge

tinyML Summit April 22 - 24, 2024

TINY ML

www.tinyML.org

Deploying ONNX models on embedded devices with TensorFlow Lite inference engine using conversion-based approach

Robert Kalmar, Lukas Sztefek, Martin Pavella, Pavel Macenauer

April 2024

Motivation

ONNX

- De facto a standard exchange format for DNN models.
- Rich ecosystem of ML frameworks for model development with export capabilities to ONNX format.

TensorFlow Lite

- Inference engine and model format.
- Mature and well-established support among embedded NPU vendors and solutions.
- Embedded GPU and microcontrollers support.

Design Objectives

ONNX to TFLite conversion

- Standalone converter
- Efficient and clear operator mapping
- Straightforward conversion flow
- Floating point and quantized model
- Mathematical equivalence
- Focus on NPU performance

Mapping the ONNX Operators into the TensorFlow Lite

ONNX Operators

187 ONNX operators (v1.14)

97 ONNX Runtime operators (v1.16.0)

Standard is to have the all the operators documented in markdowns, including changes between operator versions. For ONNX operators reference implementation is available.

TensorFlow Lite Operators

162 TFLite operators (v2.14)

753 Selectable TensorFlow operators (Flex Delegate)

Almost no documentation is available, details are retrieved directly from source code. Reference implementation is helpful here.

Performance Aware Operator mapping

80% of ONNX operators can be mapped into TensorFlow Lite.

85% when including the selectable operators from TensorFlow.

TensorFlow Lite enables Custom Operators, what can be used for the remaining ONNX operators.

ONNX operator	Naive TFLite	Optimal TFLite	Fallback
Group convolution	Conv2D, Conv3D	DepthwiseConv2/3D Split Conv2/3D	Conv2D, Conv3D
Clip	Min + Max	Relu	Min + Max
Gemm	BatchMatMul	FullyConnected	BatchMatmul

Shape Inference: Handling Model Dynamicity

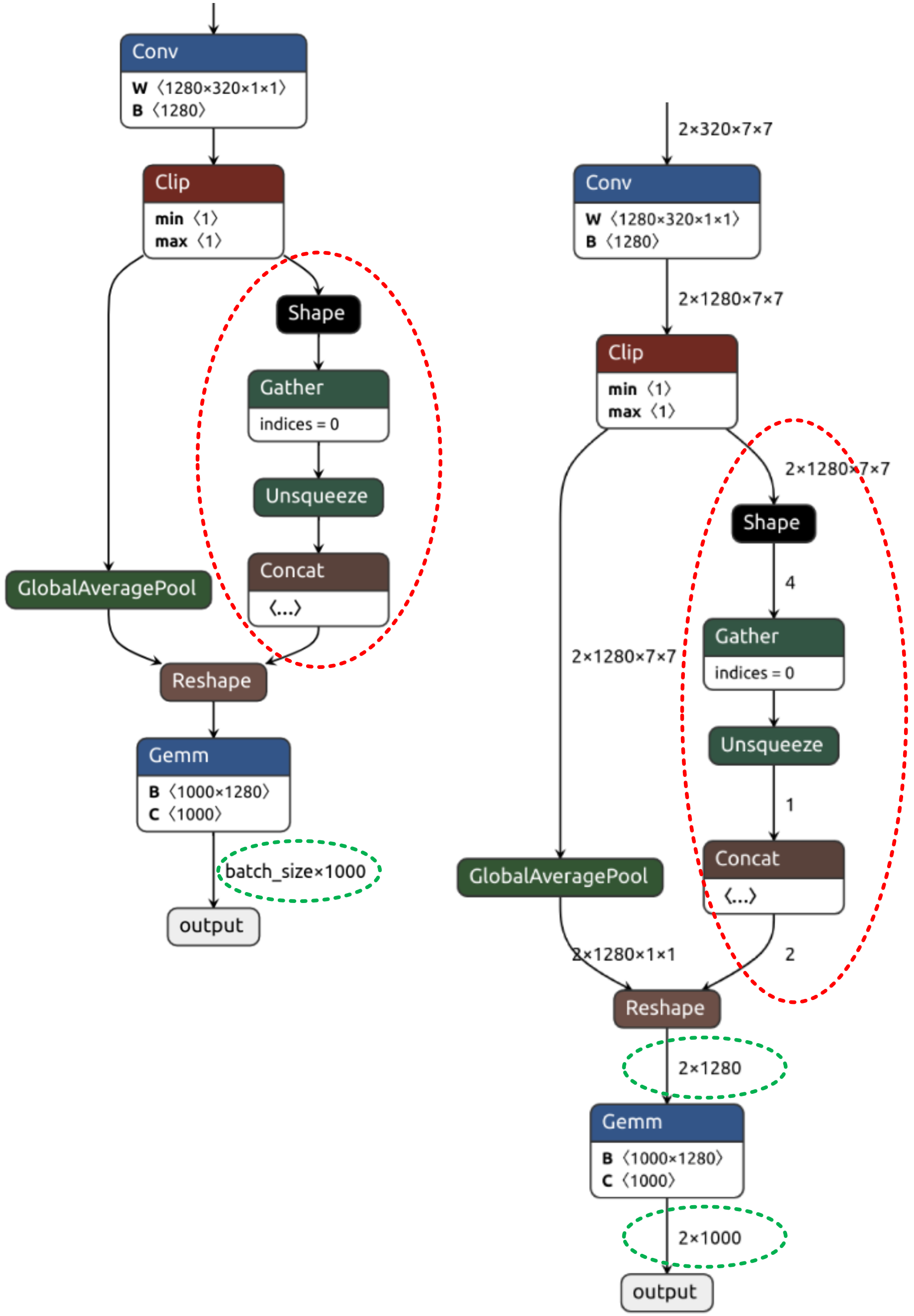
For ONNX models it is usual the shapes of internal tensors are not declared in the model.

NPU-like HW accelerators typically don't support dynamic tensor shapes and requires the shapes to be declared in the model.

(Symbolic) Shape inference:

- Base implementation in ONNX Runtime
- Extended by registering "custom" shape resolvers for required operators.

Tensor shape dynamism	Solution
No dynamism	Shape inference
Dynamism based on input tensor shape using a symbolic dimension	Shape inference + user input of desired values for symbolic dimensions Side effect: Unnecessary compute logic in the graph, what can be removed.
Dynamism based on input value	Cannot determine tensor shapes statically.



Format Inference: Converting Channels First to Channels Last

ONNX native data layout format is Channels First (NCHW), whereas for TensorFlow Lite it is Channels Last (NHWC).

Using Transpose operators to convert the layout has significant impact on final performance. Tensors must be transposed statically during the conversion.

Post Processing Approach	Preprocessing Approach
Initially, all the necessary ***Transpose*** operation are inserted to build the conversion logic from Channels First to Channels Last into the model. Consequently, in the graph optimization phase, these transpositions are removed, and the weight tensors are transposed accordingly.	Before the conversion, ***Tensor Format Inference*** is executed to determine the format of each tensor in the compute graph. The tensor format is consequently used during the operator conversion, and the weights are transposed accordingly in this step.

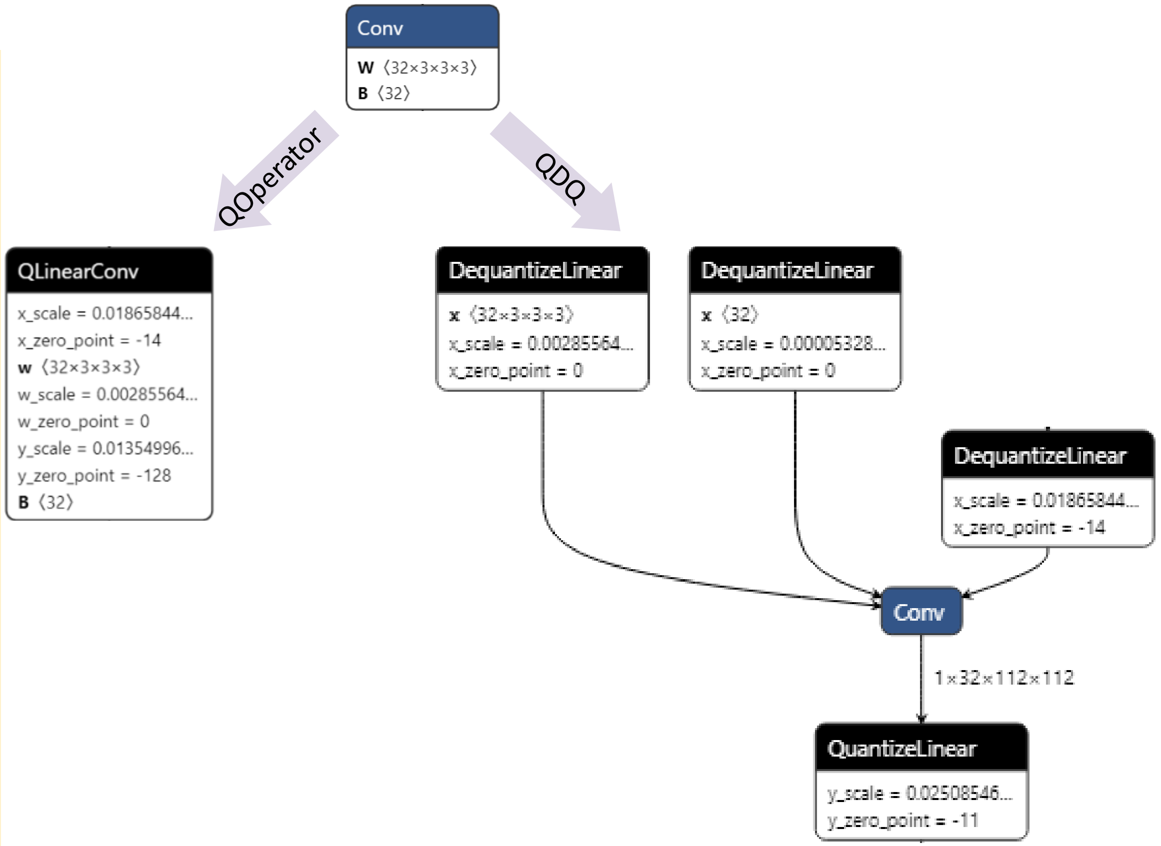
Mapping the ONNX Quantization into TensorFlow Lite

ONNX

- Quantized model representation:
  - **Operator Oriented** (QOperator),
  - **Tensor Oriented** (QDQ - Quantize-DeQuantize)
- Limited number of QOperators available
- Asymmetric and Symmetric 8-bit quantization
- Quantization parameter are input of the operators

TFLite

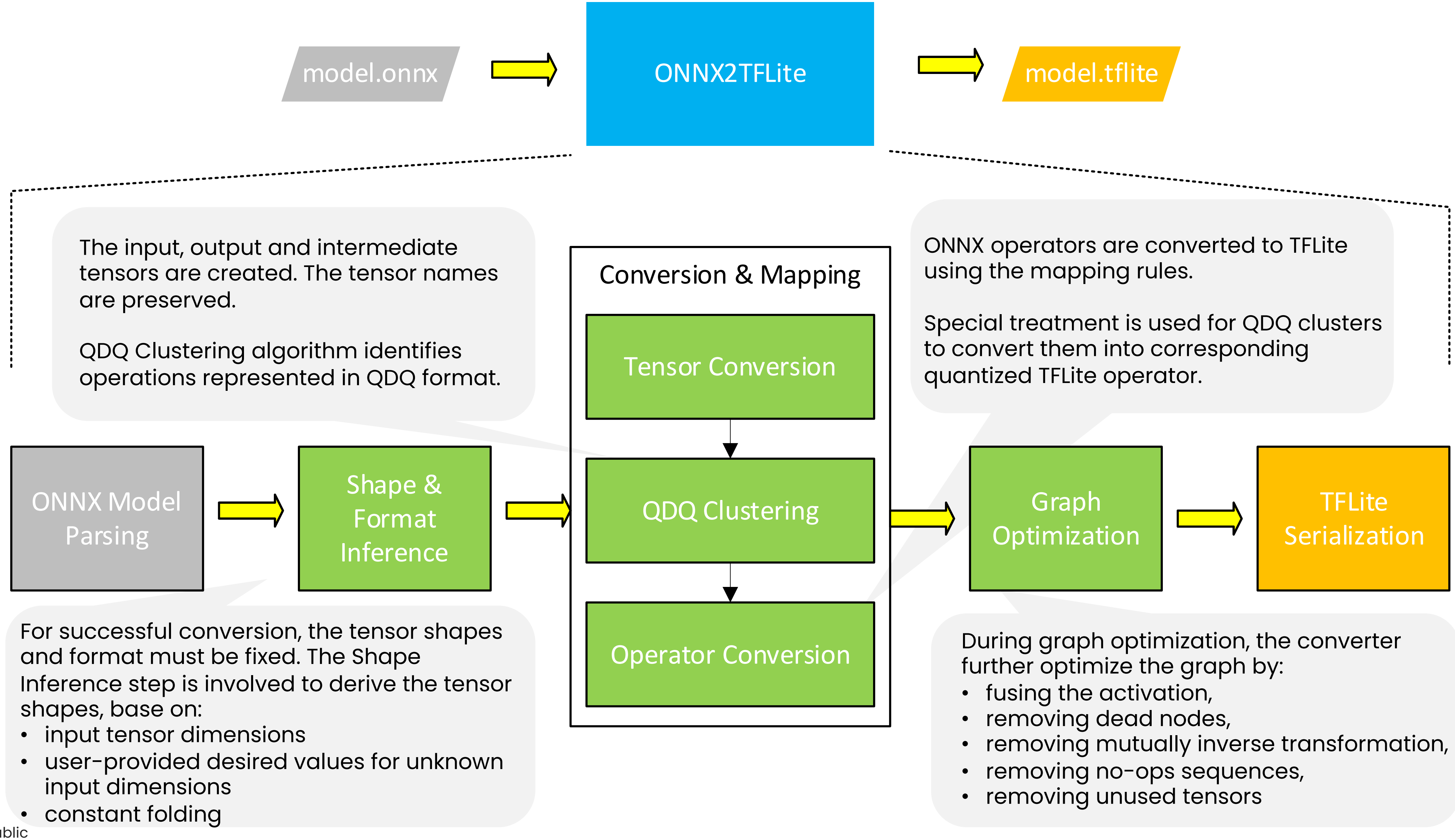
- Quantized model representation:
  - Only sort of **Operator Oriented** representation, no separate operators for quantized compute
- Decent number of operators available for quantized compute
- Asymmetric and Symmetric 8-bit and Hybrid 8/16-bit quantization
- Quantization parameters are part of the tensors



Both use same quantization formula and applies similar constraints for operators.

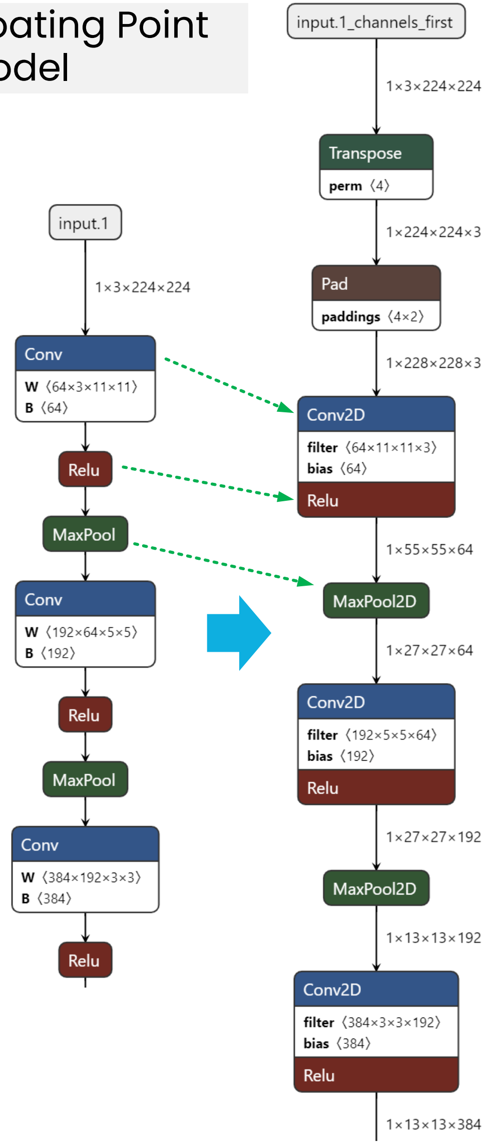
```
val_fp32 = scale * (value_quantized - zero_point)
```

Conversion Flow Details

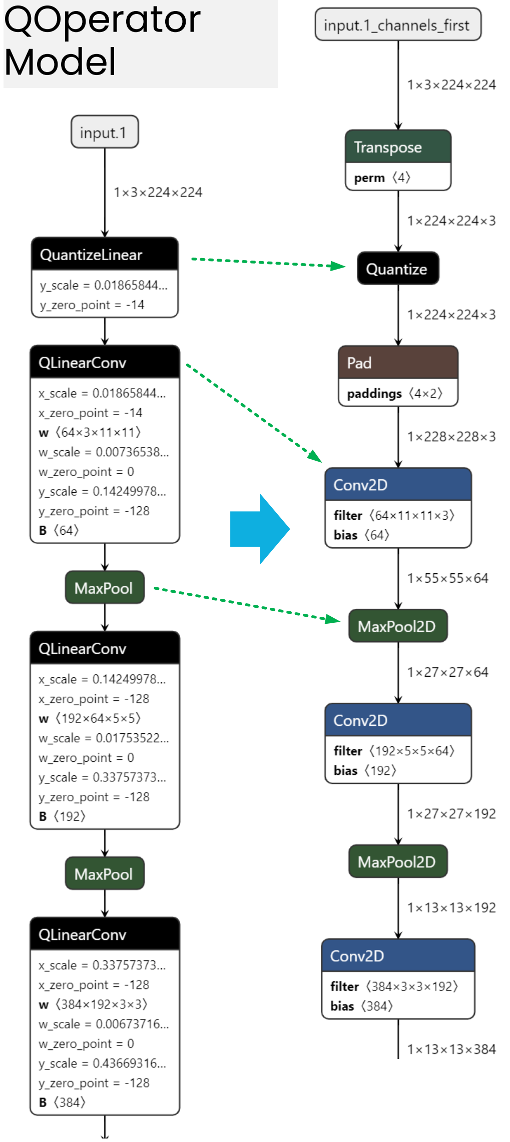


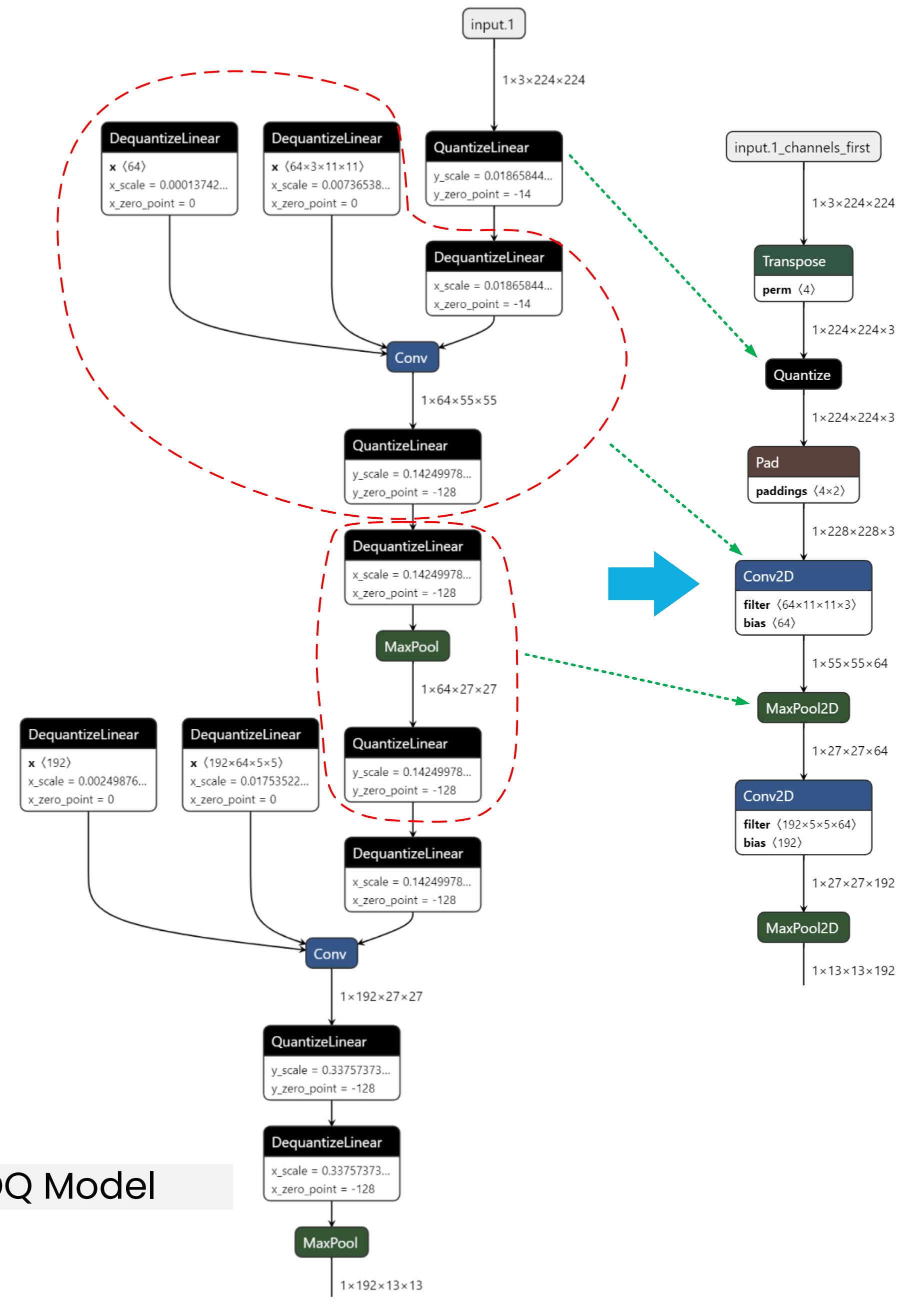
Example: Convert the AlexNet Model

Floating Point Model



QOperator Model



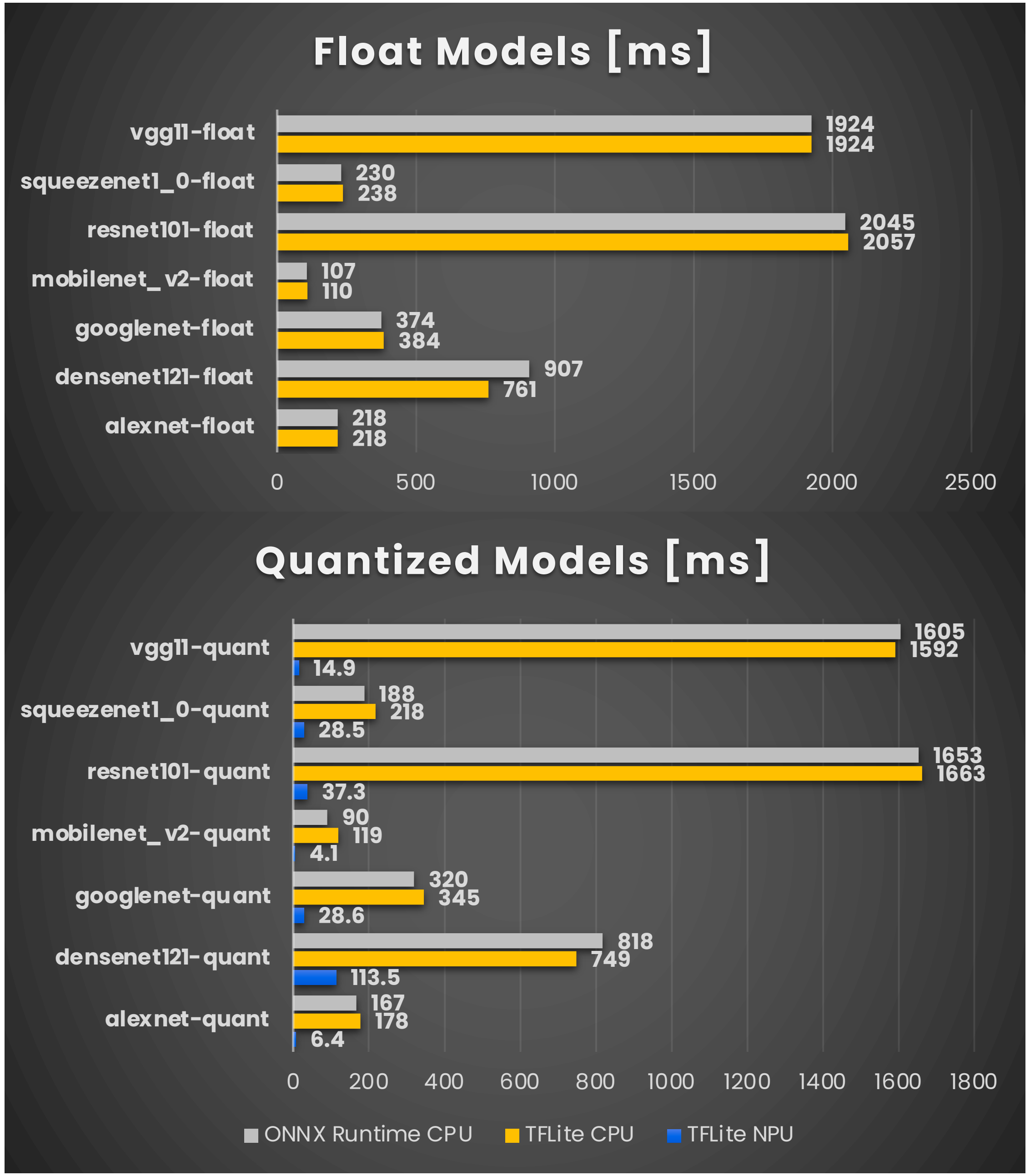


QDQ Model

Model Source: https://pytorch.org/vision/stable/models/alexnet.html

Performance Comparison

- PyTorch Vision Models [1]
- End-to-End flow:
  - PyTorch model -> *export* -> ONNX model -> *quantization* -> Quantized ONNX model -> *onnx2tflite* -> TFLite model
- Platform i.MX 8M Plus, Linux 6.6.3_1.0.0
  - 4x Cortex A53
  - 2.3 TOps NPU
  - ONNX Runtime 1.16.1; TensorFlow Lite 2.14.0
- Results:
  - Comparable performance btw. ONNX Runtime and TensorFlow Lite on CPU for both float and quantized model.
  - **Conversion to TFLite enables to run the model on existing TFLite runtime for the NPU. Achieved more than 20x acceleration leveraging the NPU.**



[1] https://pytorch.org/vision/stable/models.html

Integration into NXP eIQ® Toolkit

NXP ML Toolkit for End-to-End Model Development and Deployment

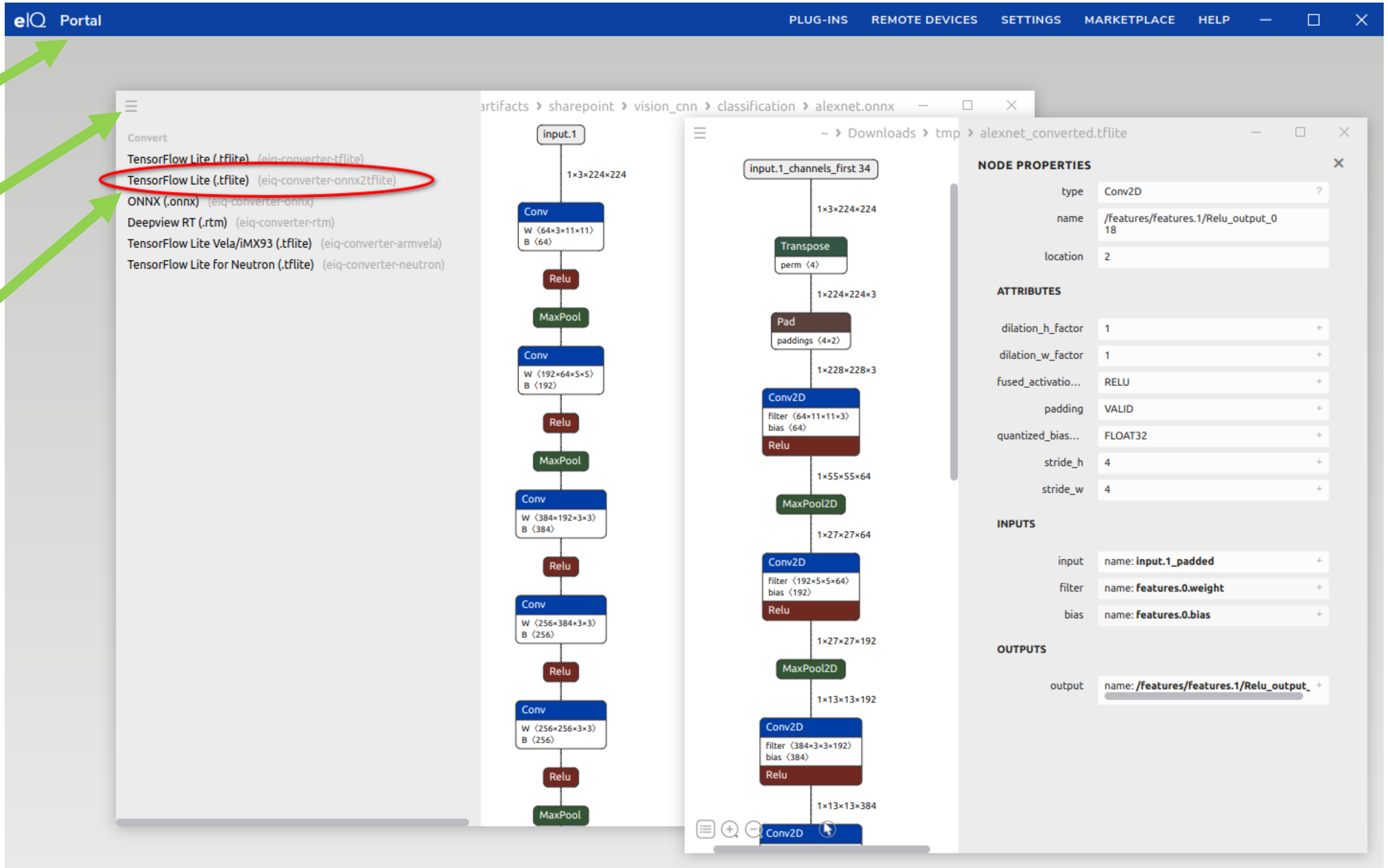
eIQ Portal ->

Model Tool ->

Options ->

Convert ->

eiq-converter-onnx2tflite



https://www.nxp.com/design/design-center/software/eiq-ml-development-environment/eiq-toolkit-for-end-to-end-model-development-and-deployment:EIQ-TOOLKIT

Get In Touch

Robert Kalmar

robert.Kalmar@nxp.com

nxp.com

Copyright Notice