Run an inference with the PyCoral API

A.UPDATE EXISTING TFLITE CODE FOR EDGE TPU :

Prerequisites:
Your model should be complied for Edge TPU.

Edge Tpu model includes a custom Edge TPU operator.
So we need to specify a delegate for the Edge TPU.Then, whenever the interpreter encounters the Edge TPU custom operator, it sends that operation to the Edge TPU. As such, inferencing on the Edge TPU requires only the TensorFlow Lite API.

So if you already have code that runs a TensorFlow Lite model, you can update it to run your model on the Edge TPU by following the steps below to update your TensorFlow Lite code.

STEPS:

To get started, all you need to do is the following:

1. On Linux, you can install it with a Debian package:
2. In your Python code, import the `pycoral` module (or just some specific sub-modules).
3. Initialize the TensorFlow Lite `Interpreter` for the Edge TPU by calling `make_interpreter()`.
4. Then use our "model adapter" functions to simplify your code—such as using `classify.set_input()` and `classify.get_output()` to process the input and output tensors—in combination with calls to the TensorFlow Lite `Interpreter`

If you already have code using the TensorFlow Lite API, then you essentially need to change just one line of code: When you instantiate the TensorFlow Lite Interpreter, also specify the Edge TPU runtime library as a delegate.

Just follow these steps convert your existing code for the Edge TPU:

1.Install the latest version of the TensorFlow Lite API by following the TensorFlow Lite Python quickstart.

2.In your Python code, import the tflite_runtime module.

```
import tflite_runtime.interpreter as tflite
```

3.Open the Python file where you'll run inference with the Interpreter API. (For an example, see the TensorFlow Lite code, label_image.py).

Instead of using import tensorflow as tf, load the tflite_runtime package like this:

4.Add the Edge TPU delegate when constructing the `Interpreter`.

For example, your TensorFlow Lite code will ordinarily have a line like this:

```
interpreter = tflite.Interpreter(model_path)
```

```
So change it to this:
```

```
interpreter = tflite.Interpreter(model_path,


experimental_delegates=[tflite.load_delegate('libedgetpu.so.1')]
)
```

5.The file passed to `load_delegate()` is the Edge TPU runtime library, and you should have installed it when you first set up your device. The filename you must use here depends on your host operating system, as follows:

- Linux: `libedgetpu.so.1`
- macOS: `libedgetpu.1.dylib`
- Windows: `edgetpu.dll`

Now when you run a model that's compiled for the Edge TPU, TensorFlow Lite delegates the compiled portions of the graph to the Edge TPU.

Edge TPU Compiler:

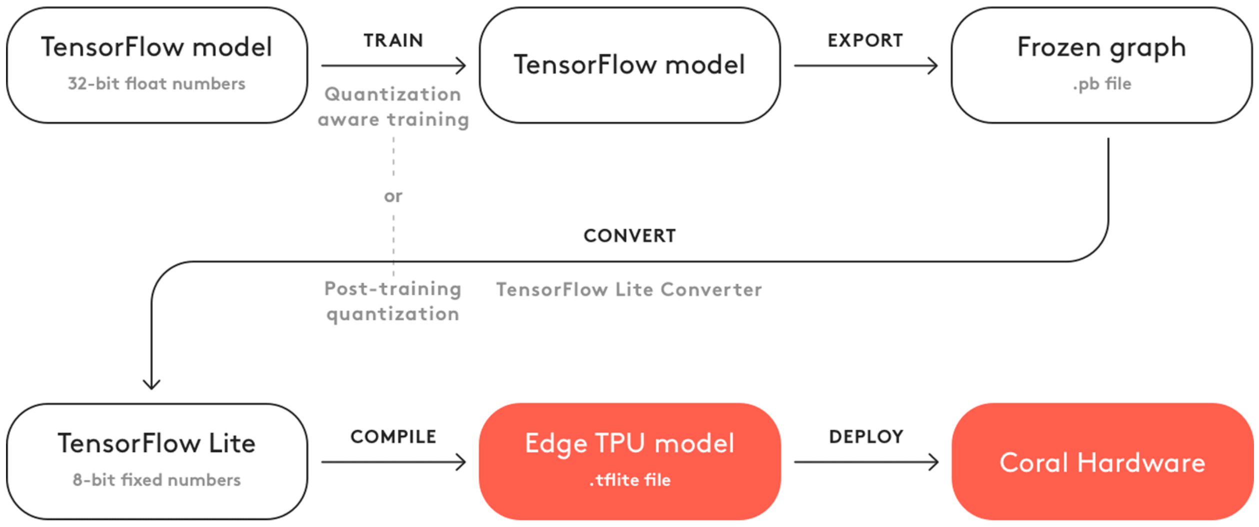
The Edge TPU Compiler (`edgetpu_compiler`) is a command line tool that compiles a TensorFlow Lite model (`.tflite` file) into a file that's compatible with the Edge TPU.

This page describes how to use the compiler and a bit about how it works.

Before using the compiler, be sure you have a model that's compatible with the Edge TPU.

COMPATIBILITY OVERVIEW:

The Edge TPU is capable of executing deep feed-forward neural networks such as convolutional neural networks (CNN). It supports only TensorFlow Lite models that are fully 8-bit quantized and then compiled specifically for the Edge TPU.TensorFlow Lite models can be made even smaller and more efficient through quantization, which converts 32-bit parameter data into 8-bit representations (which is required by the Edge TPU).
Figure 1 illustrates the basic process to create a model that's compatible with the Edge TPU. Most of the workflow uses standard TensorFlow tools. Once you have a TensorFlow Lite model, you then use our Edge TPU compiler to create a `.tflite` file that's compatible with the Edge TPU.



B.BUILDING OWN MODELS FOR EDGE TPU :

If you want to build your own TensorFlow model that takes full advantage of the Edge TPU at runtime, it must meet the following requirements:

- Tensor parameters are quantized (8-bit fixed-point numbers; int8 or uint8).
- Tensor sizes are constant at compile-time (no dynamic sizes).
- Model parameters (such as bias tensors) are constant at compile-time.
- Tensors are either 1-, 2-, or 3-dimensional. If a tensor has more than 3 dimensions, then only the 3 innermost dimensions may have a size greater than 1.
- The model uses only the operations supported by the Edge TPU (see table 1 below).

NOTE : Although the Edge TPU requires 8-bit quantized input tensors, if you pass a model to the Edge TPU Compiler that uses float inputs, the compiler leaves a quantize op at the beginning of your graph (which runs on the CPU). So as long as your tensor parameters are quantized, it's okay if the input and output tensors are float because they'll be converted on the CPU.

Supported operations

When building your own model architecture, be aware that only the operations in the following table are supported by the Edge TPU. If your architecture uses operations not listed here, then only a portion of the model will execute on the Edge TPU, as described in the section below about compiling.

When creating a new TensorFlow model, also refer to the list of operations compatible with TensorFlow Lite.

Operation name	**Runtime version***	**Known limitations**
Add	All	
AveragePool2d	All	No fused activation function.
Concatenation	All	No fused activation function. If any input is a compile-time constant tensor, there must be only 2 inputs, and this constant tensor must be all zeros (effectively, a zero-padding op).

Conv2d	All	Must use the same dilation in x and y dimensions.
DepthwiseConv2d	≤12	Dilated conv kernels are not supported.
	≥13	Must use the same dilation in x and y dimensions.
ExpandDims	≥13	
FullyConnected	All	Only the default format is supported for fully-connected weights. Output tensor is one-dimensional.
L2Normalization	All	
Logistic	All	
Maximum	All	
MaxPool2d	All	No fused activation function.
Mean	≤12	No reduction in batch dimension. Supports reduction along x- and/or y-dimensions only.
	≥13	No reduction in batch dimension. If a z-reduction, the z-dimension must be multiple of 4.

Minimum	All	
Mul	All	
Pack	≥13	No packing in batch dimension.
Pad	≤12	No padding in batch dimension. Supports padding along x- and/or y-dimensions only.
	≥13	No padding in batch dimension.
Quantize	≥13	
Relu	All	
Relu6	All	
ReluN1To1	All	
Reshape	All	Certain reshapes might not be mapped for large tensor sizes.
ResizeBilinear	All	Input/output is a 3-dimensional tensor. Depending on input/output size, this operation might not be mapped to the Edge TPU to avoid loss in precision.

ResizeNearestNeighbor	All	Input/output is a 3-dimensional tensor. Depending on input/output size, this operation might not be mapped to the Edge TPU to avoid loss in precision.
Slice	All	
Softmax	All	Supports only 1-D input tensor with a max of 16,000 elements.
SpaceToDepth	All	
Split	All	No splitting in batch dimension.
Squeeze	≤12	Supported only when input tensor dimensions that have leading 1s (that is, no relayout needed). For example input tensor with [y][x][z] = 1,1,10 or 1,5,10 is ok. But [y][x][z] = 5,1,10 is not supported.
	≥13	None.
StridedSlice	All	Supported only when all strides are equal to 1 (that is, effectively a Stride op), and with ellipsis-axis-mask == 0, and new-axis-max == 0.
Sub	All	

Sum	≥13	Reduction in z-dimension only (innermost dimension).
Tanh	All	
TransposeConv	≥13	

* *You must use a version of the Edge TPU Compiler that corresponds to the runtime version.*

2.QUANTIZATION:

Quantizing your model means converting all the 32-bit floating-point numbers (such as weights and activation outputs) to the nearest 8-bit fixed-point numbers. This makes the model smaller and faster. And although these 8-bit representations can be less precise, the inference accuracy of the neural network is not significantly affected.

For compatibility with the Edge TPU, you must use either quantization-aware training (recommended) or full integer post-training quantization.

Quantization-aware training (for TensorFlow 1) uses "fake" quantization nodes in the neural network graph to simulate the effect of 8-bit values during training. Thus, this technique requires modification to the network before initial training. This generally results in a higher accuracy model (compared to post-training quantization) because it makes the model more tolerant of lower precision values, due to the fact that the 8-bit weights are learned through training rather than being converted later. It's also currently compatible with more operations than post-training quantization.

Full integer post-training quantization doesn't require any modifications to the network, so you can use this technique to convert a previously-trained network into a quantized model. However, this conversion process requires that you supply a representative dataset. That is, you need a dataset that's formatted the same as the original training dataset (uses the same data range) and is of a similar style (it does not need to contain all the same classes, though you may use previous training/evaluation data). This representative dataset allows the quantization process to measure the dynamic range of activations and inputs, which is critical to finding an accurate 8-bit representation of each weight and activation value.

However, not all TensorFlow Lite operations are currently implemented with an integer-only specification (they cannot be quantized using post-training quantization). By default, the TensorFlow Lite converter leaves those operations in their float format, which is not compatible with the Edge TPU. As described below, the Edge TPU Compiler stops compiling when it encounters an incompatible operation (such as a non-quantized op), and the remainder of the model executes on the CPU. So to enforce integer-only quantization, you can instruct the converter to throw an error if it encounters a non-quantizable operation.

NOTE: As mentioned in the model requirements, the Edge TPU requires 8-bit quantized input tensors. However, if you pass the Edge TPU Compiler a model that's internally quantized but still uses float inputs, the compiler leaves a quantize op at the beginning of your graph (which runs on the CPU). Likewise, the output is dequantized at the end.

So it's okay if your TensorFlow Lite model uses float inputs/outputs. However, if you run an inference with the Edge TPU Python API, that API requires all input data be in uint8 format. You can instead use the TensorFlow Lite API, which provides full control of the input tensors, allowing you to pass your model float inputs—the on-CPU quantize op then converts the input to int8 for processing on the Edge TPU.

But beware that if your model uses float input and output, then there will be some amount of latency added due to the data format conversion, though it should be negligible for most models (the bigger the input tensor, the more latency you'll see). So to achieve the best performance possible, we recommend fully quantizing your model so the input and output use int8 or uint8 data.

COMPILING:

After you train and convert your model to TensorFlow Lite (with quantization), the final step is to compile it with the Edge TPU Compiler.

If your model does not meet all the requirements listed at the top of this section, it can still compile, but only a portion of the model will execute on the Edge TPU. At the first point in the model graph where an unsupported operation occurs, the compiler partitions the graph into two parts. The first part of the graph that contains only supported operations is compiled into a custom operation that executes on the Edge TPU, and everything else executes on the CPU.