EdgeTPU-optimized Vision Models

Image classification task

Introduction

We are presenting a family of computer vision models based on MobileNetEdgeTPUV2 that are optimized for the next generation Edge TPU ML accelerator in the Google Tensor SoC that powers the Pixel 6 phones. These models improve the latency-accuracy pareto-frontier compared to the existing SOTA on-device models including their predecessor MobileNetEdgeTPUs. MobileNetEdgeTPUV2 can be used as a standalone image classification model or as a backbone for other computer vision tasks such as object detection or semantic segmentation.

Search space design

During the design of MobileNetEdgeTPUV2 we crafted a neural network search space which includes building blocks that run efficiently on the Edge TPU accelerator while providing better algorithmic qualities and leveraged AutoML to find the optimal architectures. As one of the key optimizations, we introduce Group Convolution based Inverted Bottleneck (IBN) blocks that provide great flexibility in achieving a tradeoff between latency and accuracy.

Inverted Bottleneck (IBN) is a widely used building block in architecting a neural network for mobile vision tasks. A conventional IBN uses pointwise convolutions for expansion/projection before/after a depthwise convolution. Previously it has been shown that using a full convolution replacing the pointwise expansion and depthwise convolution can provide more trainable parameters while being faster. However, one big limitation is that using these full-convolution IBNs can get very expensive in terms of latency and memory requirements, especially for narrow/deep tensors that we see in later stages of vision models. This limits the use of “fused” full-convolution IBNs throughout the model and leaves depthwise IBN as the only alternative.

Inverted bottleneck block (IBN) variants: (a) Conventional with depthwise, (b) Fused-IBN, (c)GC-IBN with group convolutions in the expansion phase

In this work we utilize Group Convolution (GC) as part of the fused expansion in constructing IBNs (Figure 1). GC based IBN becomes a versatile block that opens up a large design space between conventional depthwise IBNs and fused full-convolution IBNs which can be controlled by the group size parameter. Figure 2 demonstrates the search space enabled by GC-based IBNs that allows a flexible tradeoff between latency and number of trainable parameters. GC-based IBNs allow increasing the number of trainable parameters gradually without requiring the latency cost of full-convolution based IBNs. Moreover, they can also be faster than conventional IBNs with depthwise convolutions while providing more trainable parameters.

Model performance on Edge TPU

Tradeoffs discussed above and exemplified in Figure 2 are highly dependent on the tensor shapes and cannot be generalized throughout the neural network. Hence, we use AutoML techniques as a rescue to find the optimal block decisions and craft a family of network architectures at different latency targets. Figure 3 demonstrates that the resulting MobileNetEdgeTPUV2 model-family improves the pareto-frontier compared to the existing on-device SOTA models when run on Edge TPU.

Comparison of Imagenet top-1 accuracy and Pixel 6 Edge TPU latency of MobileNetEdgeTPUV2 models with other on-device classification models

On-device benchmarking of classification models Results on on-device benchmarking of various int8 quantized image classification models for 224x224 input resolution:

Model (Checkpoint)	Accuracy (int8)	Pixel 6 Edge TPU Latency (ms)	tflite
MobileNetEdgeTPUv2-Tiny	74.66%	0.78	link
MobileNetEdgeTPUv2-XS	75.79%	0.82	link
MobileNetEdgeTPUv2-S	77.36%	1.03	link
MobileNetEdgeTPUv2-M	78.43%	1.35	link
MobileNetEdgeTPUv2-L	79.00%	1.64	link
MobileNetEdgeTPU dm1.0	75.6%	0.92	link
MobileNetEdgeTPU dm1.25	77.06%	1.20	link
MobileNetEdgeTPU dm1.5	75.9%	1.42	link
MobileNetEdgeTPU dm1.75	78.6%	1.93	link

Model performance on Pixel 6 CPU

Our primary optimization target is the Edge TPU accelerator however in our search space we include operations that also run well on Pixel 6 CPU to be able to reach a wide range of platforms. Moreover, we implement GC using functionally equivalent series of commonly used ML primitives (channelwise slice, full convolution, concatenation) as shown in Figure 2, since a native GC operation may not be supported for all target platforms. As a result, the performance of MobileNetEdgeTPUV2 is also superior to other on-device models when run on Pixel 6 CPU as shown in Figure 4.

Comparison of Imagenet top-1 accuracy and Pixel 6 latency of MobileNetEdgeTPUV2 models with other on-device classification models

Semantic segmentation task

Using classification models as backbone

We also present segmentation models based on MobileNetEdgeTPUV2 backbone and DeepLab v3 plus decoder and head (first used here). These models optimized for the next generation Edge TPU accelerators featured in Pixel 6 phones and improve the latency-accuracy pareto-frontier compared to the their predecessor based on MobileNetV2 and DeepLabV3+.

The segmentation model is built using the pretrained MobileNetEdgeTPUV2 as a feature encoder and ASPP decoder in conjunction with a Deeplab V3 Plus head. Separable convolutions used to reduce the size of the model.

Using architecture search to find high-quality, low-latency segmentation models To further improve the quality of on-device segmentation models, we invoke architecture search to jointly search for the model's feature extractor and the segmentation head. Autoseg-EdgeTPU is a set of searched segmentation models customized for the Edge TPU in Pixel 6. The feature extractor is derived from Edge TPU search space where a mixture of IBN and fused IBN are used. We automatically find the optimal kernel size, channel multiplier, expansion ratio, and groups on a per layer basis using a reinforcement learning algorithm. The segmentation head is an optimized version of Bi-FPN head, with customized number of repeats and feature selection.

Argmax fusion to improve segmentation model latency The last two levels of the model (bilinear resizing and Argmax) contribute significantly to latency on the device model. This is due to the large activation size between these layers (512 x 512 x Number of classes). These layers can be merged without significantly impacting quality scores by making Argmax smaller and scaling the classes to the desired size with nearest neighbor.

On-device benchmarking of segmentation models

Performance of AutosegEdgeTPU and MobileNetEdgeTPUV2+DeeplabV3+ models on the 32-class ADE20K semantic segmentation task.

Model Name (Checkpoint)	Backbone	Segmentation Head	#Parameters (million)	ADE20K 32-class mIOU	Pixel 6 EdgeTPU latency (ms)	Tflite
deeplabv3plus_mobilenet_edgetpuv2_baseline	MobileNet V2 (baseline)	DeeplabV3+	2.34	54.06%	7.5	link
deeplabv3plus_mobilenet_edgetpuv2_xs	MobileNet EdgeTPUV2-XS	DeeplabV3+	3.6	56.02%	5.2	link

Model Name (Checkpoint)	Backbone	Segmentation Head	#Parameters (million)	ADE20K 32-class mIOU	Pixel 6 EdgeTPU latency (ms)	Tflite
deeplabv3plus_mobilenet_edgetpuv2_s	MobileNet EdgeTPUV2-S	DeepLabV3+	2.2	59.43%	5.9	link
deeplabv3plus_mobilenet_edgetpuv2_m	MobileNet EdgeTPUV2-M	DeepLabV3+	2.7	59.81%	7.2	link
autoseg_edgetpu_xs	AutoSeg-EdgeTPU-XS	BiFPN	2.9	59.64%	5.4	link
autoseg_edgetpu_s	AutoSeg-EdgeTPU-S	BiFPN	3.1	61.31%	5.7	link

By fusing argmax with resize operator as shown above, it is possible to further improve the on-device latency of the segmentation models without significantly impacting the quality:

Note: Models with default argmax and fusing argmax are using the same checkpoint since there is no parameter change.

Model Name	ADE20K 32-class mIOU	Pixel 6 EdgeTPU latency (ms)	tflite
deeplabv3plus_mobilenet_edgetpuv2_xs	56%	3.4	link
deeplabv3plus_mobilenet_edgetpuv2_s	59.41%	4.2	link
deeplabv3plus_mobilenet_edgetpuv2_m	59.79%	5.5	link
autoseg_edgetpu_xs	59.62%	3.6	link
autoseg_edgetpu_s	61.28%	3.9	link

Object detection task

EdgeTPU-optimized models for object detection are hosted in the TensorFlow object detection API

Training the models

Note that the `EXPERIMENT_TYPE` has to be in one of the preregistered classification configs, such as `mobilenet_edgetpu_xs` for classification models. In case you train segmentation model `EXPERIMENT_TYPE` has to be in one of the preregistered segmentations configs], such as `seg_deeplabv3plus_mobilenet_edgetpuv2_s_ade20k`, `autoseg_edgetpu_xs`

```
EXPERIMENT_NAME=xxx  # Change this for your run, for example, 'mobilenet-edgetpu-test-run'
EXPERIMENT_TYPE=xxx  # Change this for your run, for example, 'mobilenet_edgetpu_v2_xs'
$ python3 train.py \
--experiment_name=${EXPERIMENT_NAME} \
--experiment_type=${EXPERIMENT_TYPE}  \
--mode=train_and_eval
```

From training to quantized inference deployment

To export quantized tflite models using tensorflow post-training quantization:

For classification models:

```
$ python3 serving/export_tflite.py
--model_name=${EXPERIMENT_TYPE} \
--ckpt_path=${CHECKPOINT} \
--dataset_dir=/path/to/calibration/dataset \
--output_dir=/tmp \
--quantize \
--image_size=224
```

Note that the `EXPERIMENT_TYPE` has to be in one of the preregistered classification configs, such as `mobilenet_edgetpu_xs`.

For segmentation models:

```
$ python3 serving/export_tflite.py \
--model_name=${EXPERIMENT_TYPE}
--ckpt_path=${CHECKPOINT} \
--dataset_dir=/path/to/calibration/dataset \
--output_dir=/tmp \
--quantize \
--quantize_less_restrictive \
--image_size=512 \
--finalize_method=${ARGMAX_FUSION}
```

`EXPERIMENT_TYPE` has to be in one of the preregistered segmentations configs, such as `deeplabv3plus_mobilenet_edgetpuv2_s_ade20k_32`.

`ARGMAX_FUSION` has to be in one of the following:

- `resize512,argmax`: Argmax applied after scaling the output to 512x512.
- `resize256,argmax,resize512,squeeze`: Scale the output to 256x256, apply argmax, scale to 512x512 using nearest neighbor upsampling
- `resize128,argmax,resize512,squeeze`: Scale the output to 128x128, apply argmax, scale to 512x512 using nearest neighbor upsampling

On-device benchmarking

The models in this repository are compatible with NNAPI and can be benchmarked on Pixel 6 devices using the tflite benchmark tool

While using the benchmark tool, enable the use of NNAPI by setting the `use_nnapi` command line argument to `true`, and specifying the `nnapi_accelerator` as `google-edgetpu`

```
$ bazel build -c opt --config=android_arm64 tensorflow/lite/tools/benchmark:benchmark_model
# Push binary to device
$ adb push bazel-bin/tensorflow/lite/tools/benchmark/benchmark_model /data/local/tmp
# Push model to device
$ adb push /path/to/model.tflite /data/local/tmp/
# Run on-device benchmarking
$ adb shell /data/local/tmp/benchmark_model --graph=/data/local/tmp/model.tflite --use_nnapi
nnapi_accelerator_name=google-edgetpu
```