REPORT PINEBOARD BRICOLA

Introduzione

Le EdgeTPU sono dispositivi embedded che utilizzano un acceleratore hardware chiamato Tensor Processing Unit per calcolare l'inferenza di una rete neurale. Le Edge TPU si possono usare non solo per l'inferenza ma anche per il fine-tuning di una rete neurale preaddestrata (pycoral - retraining) Le Edge TPU eseguono solo reti neurali quantizzate, ovvero reti neurali che contengono operazioni tra numeri in artimetica a 8 bit, a differenza delle reti neurali che su PyTorch e Tensorflow sono float32 per impostazione predefinita.

Tutti i pesi (parametri) della rete neurale sono salvati come variabili intere a 8 bit quindi una rete neurale quantizzata richiede meno memoria. Le TPU e le GPU NVIDIA di ultima generazione possiedono un hardware in cui le operazioni in aritmetica int8 sono più rapide delle operazioni float32 e float16, quindi il modello quantizzato int8 ha uno speedup potenziale di x4 nel calcolo dell'inferenza rispetto al modello float32.

Partendo da una rete neurale ordinaria con operazioni in precisione float32, si puo' ottenere una rete neurale quantizzata int8, con una riduzione di accuracy trascurabile

Si usa la libreria PyCoral (documentazione) per eseguire le reti neurali su Edge TPU. La libreria PyCoral esegue modelli di rete neurale in formato Tensorflow Lite. Anche l'ingresso e l'uscita del modello devono essere quantizzati a 8 bit.

In particolare, la guida sul sito PineBoards descrive l'installazione della repositori deprecata Edgetpu (documentazione), (repository)

Configurazione

La compagnia PineBoard distribuisce un Raspberry5 collegato ad una Edge TPU di Google. Si seguono i due tutorial ufficiali sul sito di PineBoard, con la diffenza che si installa Docker secondo la guida ufficiale di Docker

tutorial configurazione della PineBoard

tutorial installazione docker (tutorial ufficiale per linux debian)

Il seguente tutorial illustra come installare la repository edgetpu (repository) su un container, ma utilizza la versione deprecata 13.

tutorial installazione di pycoral su pineboard

Export di modello da Tensorflow a TFLite

Il modo più rapido per ottenere un modello quantizzato è la “post-training quantization”, dove si converte direttamente un modello float32 a int8 con una piccola riduzione di accuracy. Il modo che permette di conservare meglio l’accuracy del modello float32 ma più lento è il “quantization-aware training”.

Il metodo “post training integer quantization” (link google ai) permette di convertire un direttamente modello da float32 a int8, con ingresso ed uscita del modello quantizzati. La conversione richiede di fornire un “representative dataset” in modo che il modello converta adeguatamente le immagini d’ingresso da float a int. Questo metodo permette di creare un modello supportato dall’edge TPU

Esempio pratico “post training integer quantization” in colab

Si prende un modello dallo zoo di modelli keras.application, si fornisce un dataset rappresentativo,

```
import tensorflow as tf
import os
from PIL import Image
import cv2
import numpy as np
import tensorflow as tf
from google.colab import drive
drive.mount('/content/drive')
import matplotlib.pyplot as plt
folder_path = "drive/MyDrive/ricerca/representative_ds"
image_names = os.listdir(folder_path)
image_paths = [folder_path + "/" + image_name for image_name in image_names]
image_array = []
for image_path in image_paths:
image = cv2.imread(image_path)
image = cv2.resize(image, (224,224))
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_preprocessed = tf.keras.applications.resnet50.preprocess_input(image)
image_array.append(image_preprocessed)
image_array = np.array(image_array)

model = tf.keras.applications.resnet.ResNet152(weights='imagenet', input_shape=(224, 224, 3))
```

```
def representative_data_gen():
for input_value in tf.data.Dataset.from_tensor_slices(image_array).batch(1).take(100):
yield [input_value]

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen
# Ensure that if any ops can't be quantized, the converter throws an error
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Set the input and output tensors to uint8 (APIs added in r2.3)
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8

tflite_model_quant = converter.convert()
```

```
interpreter = tf.lite.Interpreter(model_content=tflite_model_quant)
input_details = interpreter.get_input_details()[0]
output_details = interpreter.get_output_details()[0]
for key in input_details.keys():
print(key, input_details[key])
print()
for key in output_details.keys():
print(key, output_details[key])
```

```
import pathlib

tflite_models_dir = pathlib.Path("drive/MyDrive/ricerca/modelli/")
tflite_models_dir.mkdir(exist_ok=True, parents=True)

# Save the quantized model:
tflite_model_quant_file = tflite_models_dir/f"resnet152.tflite"
tflite_model_quant_file.write_bytes(tflite_model_quant)
```

Una volta creato il modello Tensorflow Lite, si usa il compilatore “Edge TPU Compiler” per convertire il modello in formato edge tpu. Il seguente (provabile su Colab) installa il compilatore e converte un modello. Il parametro --min_runtime_version 13 rende il modello compatibile con la versione deprecata di Pycoral scelta dagli sviluppatori di Pineboard.

```
! curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
! echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" | sudo tee
/etc/apt/sources.list.d/coral-edgetpu.list
! sudo apt-get update
! sudo apt-get install edgetpu-compiler
! edgetpu_compiler --min_runtime_version 13 modeltflite
```

Esecuzione rete neurale su Edge TPU

Con il seguente comando si esegue il container su cui è stato installata la repository edgetpu. In particolare si esegue il binding affinché la cartella Documents del raspberry corrisponda alla cartella home del container.

rasperrypi:~ $ sudo docker run -it --mount type=bind,source=/home/pi/Documents/,destination=/home/ --device /dev/apex_0:/dev/apex_0 coral /bin/bash

Codice Benchmark

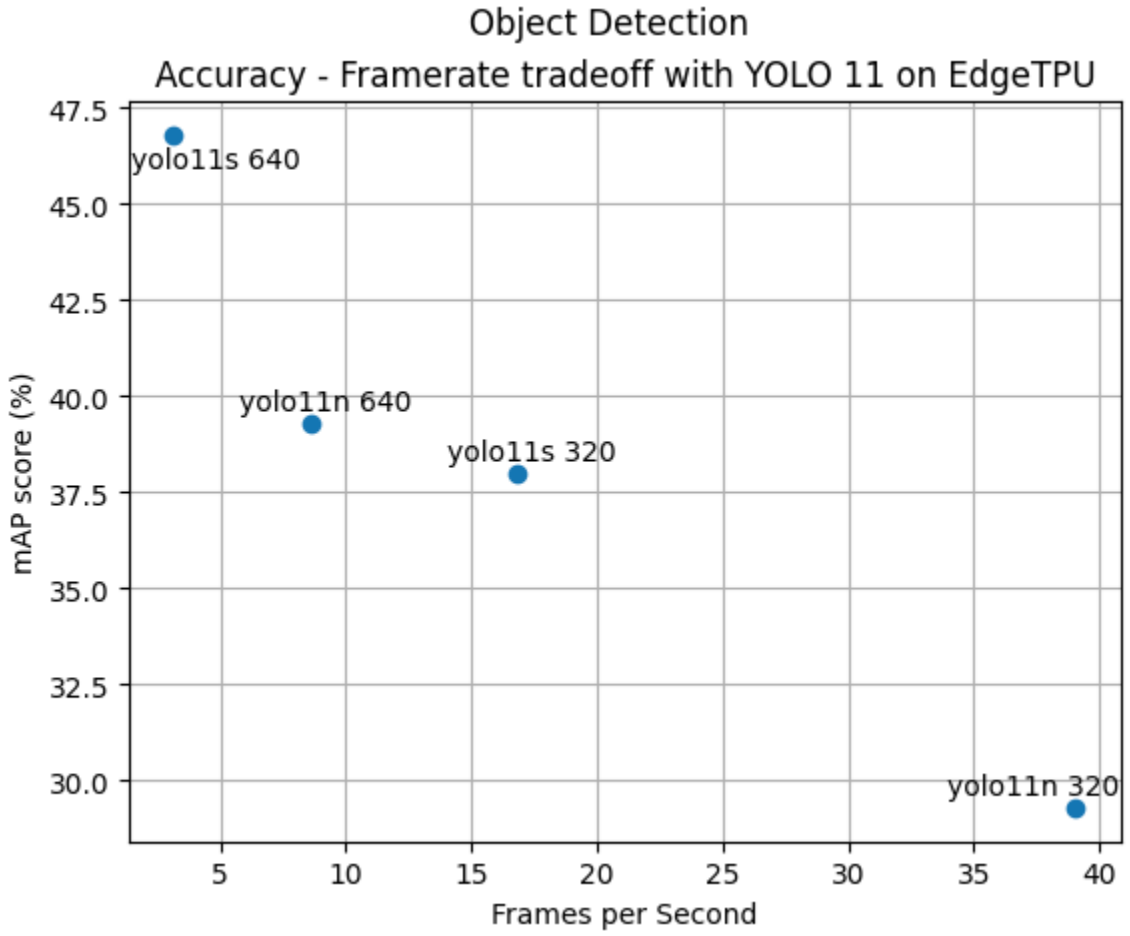
```
import cv2
from edgetpu.basic.basic_engine import BasicEngine
from PIL import Image
import numpy as np

BENCHMARK_SAMPLES_NO = 600
IMAGE_RES = 224

folder_path = "/home/models/keras/"

import os
print("models in folder", os.listdir(folder_path))



model_path = folder_path + "/" + "DenseNet201_edgetpu.tflite"
engine = BasicEngine(model_path)
input_size = engine.required_input_array_size()
```

```
inference_times = np.zeros((BENCHMARK_SAMPLES_NO,))
video_path = '/home/benchmark/benchmark_int_224.mp4'
cap = cv2.VideoCapture(video_path)

for i in range(BENCHMARK_SAMPLES_NO):
ret, frame = cap.read()
if not ret:
break
image_input = np.reshape(frame, IMAGE_RES*IMAGE_RES*3)
engine.run_inference(image_input)
inference_times[i] = engine.get_inference_time()

print("avg inference time", np.average(inference_times))
```

Risultati Benchmark

Modello	FLOPs	Latenza media(ms)
resnet50		47.56
resnet101		95.5
resnet152		135
densnet121		14
densenet169		29.6
densenet201		47.6

	FLOPs (billions)	mAP^{bo} score	Latenza
yolo11n 320	0.827	29.3%	25.6 ms
yolo11n 640	3.30	39.3%	115.6 ms
yolo11s 320	2.71	38%	59.4 ms
yolo11s 640	10.9 M	46.8%	316 ms



Segmentazione con Yolo 11 Segment

	FLOPs (billions)	mAP score	Latenza
yolo11n seg 320	1.31	28.8%	45.3 ms
yolo11n seg 640	5.26	38.7%	215 ms
yolo11s seg 320	4.47	37.5%	110 ms
yolo11s seg 640	17.9	46.5%	575 ms