

Bài 51 - Quantization trong deep learning

23 Nov 2020 - phamdinhkhanh

Menu

- 1. Tại sao cần quantization
 - 1.1. Khái niệm về quantization
 - 1.2. Quantization giảm kích thước bao nhiêu lần ?
 - 1.2.1. Biểu diễn nhị phân
 - 1.3. Các dạng quantization
- 2. Quantization trên tensorflow
 - 2.1. Convert một model tflite
 - 2.2. Quantization model
 - 2.3. Độ chính xác của mô hình quantization
 - 2.3.1. Convert model by commandline
 - 2.3.2. Quantization trên pytorch
 - 3. Kết luận
- 4. Tài liệu tham khảo

1. Tại sao cần quantization

Khi tôi viết bài này thì quantization đã khá phổ biến trong Deep Learning. Đây là một khái niệm không còn mới, nhưng lại rất quan trọng. Vậy nó quan trọng như thế nào và vì sao chúng ta lại cần quantization ?

Các mô hình deep learning ngày càng đạt độ chính xác cao hơn qua thời gian. Không khó để bạn tìm thấy những mô hình đạt độ chính xác SOTA trên bộ dữ liệu ImageNet. Nếu không tin bạn có thể theo dõi độ chính xác qua thời gian trên bảng leader board của Imagenet classification (<https://paperswithcode.com/sota/image-classification-on-imagenet>).

Nhưng hầu hết những mô hình có độ chính xác cao lại không có khả năng deploy trên các thiết bị IoT và mobile có phần cứng rất yếu vì nó quá lớn để triển khai.

Quantization là một kỹ thuật giúp bạn **giảm nhẹ kích thước** các mô hình deep learning nhiều lần, đồng thời **giảm độ trễ (latency)** và **tăng tốc độ inference**. Cùng tìm hiểu về kỹ thuật tuyệt vời này qua bài viết hôm nay.

1.1. Khái niệm về quantization

Có nhiều định nghĩa khác nhau về quantization. Xin được trích dẫn khái niệm từ wikipedia ([https://en.wikipedia.org/wiki/Quantization_\(signal_processing\)](https://en.wikipedia.org/wiki/Quantization_(signal_processing))):

`Quantization, in mathematics and digital signal processing, is the process of mapping input values from a large set (often a continuous set) to output values in a (countable) smaller set, often with a finite number of elements. Rounding and truncation are typical examples of quantization processes. Quantization is involved to some degree in nearly all digital signal processing, as the process of representing a signal in digital form ordinarily involves rounding. Quantization also forms the core of essentially all lossy compression algorithms.`

Top

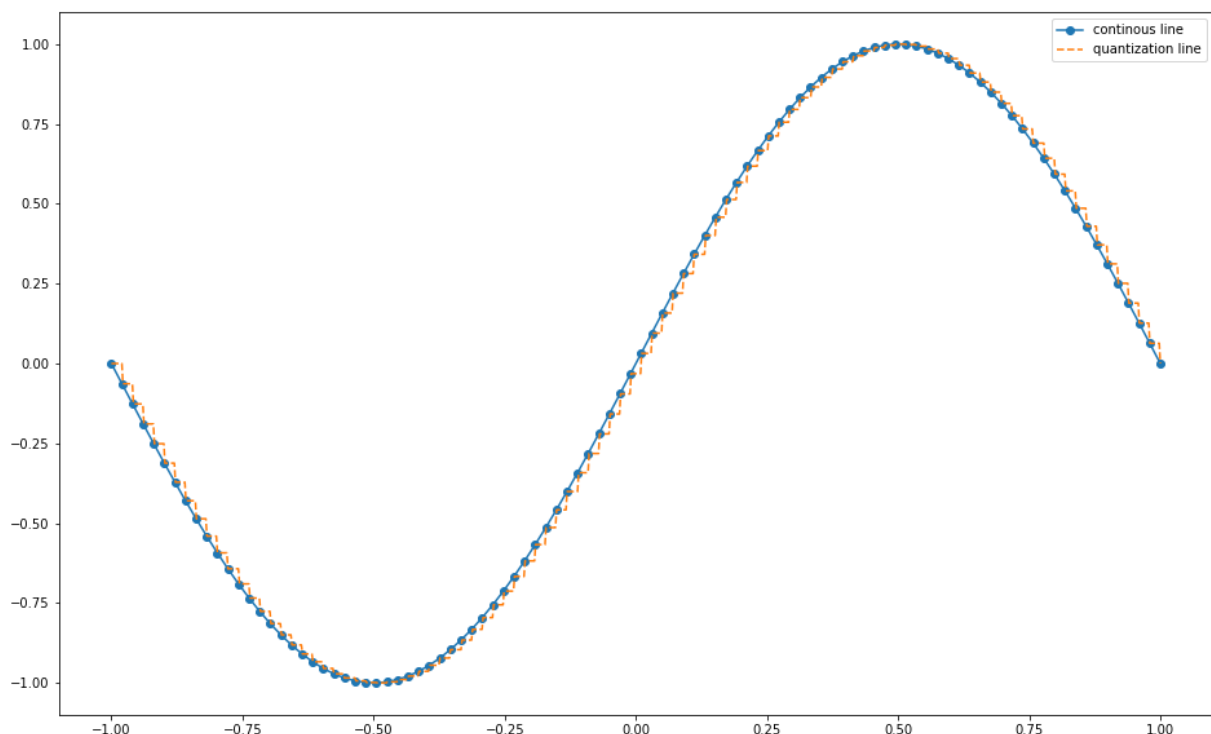
Như vậy quantization trong toán học và xử lý tín hiệu số là quá trình map các giá trị đầu vào từ một tập số lớn (thường là liên tục) sang các giá trị output là một tập nhỏ hơn có thể đếm được, hữu hạn các phần tử.

Ví dụ về quantization

Bạn có thể hình dung dễ dàng khái niệm trên thông qua ví dụ sau đây:

Giả sử một hàm $f(x) = \sin(x)$ là một hàm liên tục trong khoảng $[-\pi, \pi]$. Ta có thể tìm được một tập hợp $S = \frac{i\pi}{100} \forall i \in [-100, 100]$ là một tập hữu hạn rời rạc các điểm cách đều nhau một khoảng là $\frac{\pi}{100}$ sao cho tập các điểm (x, y) có miền xác định trên tập S có thể biểu diễn một cách gần đúng mọi điểm (x, y) trên miền liên tục $[-\pi, \pi]$.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import scipy.interpolate
4
5 x = np.linspace(-1, 1, num=100)
6 xnew = np.linspace(-1, 1, num=1000, endpoint=True)
7 y = np.sin(x*np.pi)
8 f = interpolate.interp1d(x, y, kind='previous')
9 intpol = f(xnew)
10 plt.figure(figsize=(16, 10))
11 plt.plot(x, y, marker='o', label = 'continous line')
12 plt.plot(xnew, intpol, '--', label = 'quantization line')
13 plt.legend()
14 plt.show()
```



Hình 1: Biểu diễn của hàm $\sin(x)$ và giá trị quantization của nó trên khoảng $[-\pi, \pi]$.

Trong quá trình inference model thì các xử lý được tính toán chủ yếu trên kiểu dữ liệu float32. Đơn vị khoảng (interval unit) của float32 rất nhỏ và dường như là liên tục. Do đó nó có thể biểu diễn mọi giá trị. Trong khi int8 và float16 là những tập hợp có kích thước nhỏ hơn và có đơn vị khoảng lớn hơn rất nhiều so với float32. Việc chuyển đổi định dạng từ float32 sang float16 hoặc int8 có thể coi là quantization theo định nghĩa nêu trên.

float32 sẽ có kích thước ghi nhớ lớn hơn so với các kiểu lưu trữ khác như int8, float16. Để lưu trữ float32 thì chúng ta cần 32 bits. Các bits này nhận một trong hai giá trị 0, 1 để máy có thể đọc được. Quantization hiểu đơn giản hơn là kỹ thuật chuyển đổi định dạng của dữ liệu từ những kiểu dữ liệu có độ chính xác cao sang những kiểu dữ liệu có độ chính xác thấp hơn, qua đó làm giảm memory khi lưu trữ. Do đó làm tăng tốc độ inference là giảm latency khi load model. Tất nhiên là khi bạn làm giảm độ chính xác xuống thì thường accuracy của mô hình sẽ giảm một chút (không quá nhiều) so với mô hình gốc.

Bạn đọc đã hiểu về quantization rồi chứ ? Đơn giản phải không nào ?

1.2. Quantization giảm kích thước bao nhiêu lần ?

1.2.1. Biểu diễn nhị phân

Trên máy tính, mọi giá trị đều được biểu diễn dưới dạng nhị phân gồm các bit 0, 1.

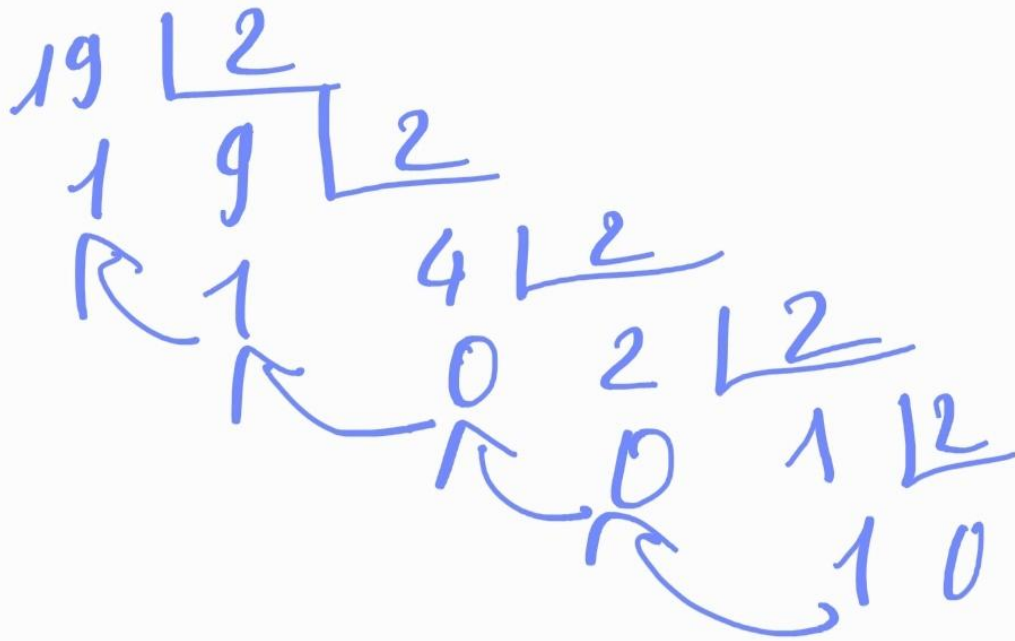
Cách convert một số nguyên từ hệ thập phân sang hệ nhị phân

Chúng ta có thể biểu diễn dễ dàng một số nguyên dưới dạng nhị phân bằng cách khai triển thành tổng các lũy thừa của 2.

$$x = \sum_{i=1}^n d_i(0, 1) \times 2^i$$

Trong đó $d_i(0, 1) \in \{0, 1\}$. Biểu diễn nhị phân của x sẽ là chuỗi $d_n d_{n-1} \dots d_0$ mà mỗi vị trí d_i nhận 2 giá trị $\{0, 1\}$. Ở cấp 2 chắc các bạn đã từng làm bài toán biến đổi số thập phân sang hệ nhị phân. Bạn còn nhớ chúng ta sẽ thực hiện liên tiếp các phép chia liên hoàn cho 2 cho đến khi không chia được nữa chứ ? Quá trình này lặp lại cho đến khi kết quả thu được sau cùng là 0. Khi đó phần dư ở các bước chia sẽ được lưu lại và cuối cùng nghịch đảo thứ tự của chuỗi phần dư ta thu được biểu diễn nhị phân của số thập phân. Cùng xem hình bên dưới.

Top



Hình 2: Phương pháp tìm biểu diễn nhị phân của 19 thông qua phép chia liên hoàn cho 2. Kết quả thu được là 10011.

Chứng minh công thức này không khó, mình xin dành cho bạn đọc như một bài tập bổ sung.

Dựa trên ý tưởng trên, chúng ta có thể code hàm biến đổi số nguyên dương sang nhị phân một cách khá dễ dàng:

```

1  import numpy as np
2
3  def _binary_integer(integer):
4      exps = []
5      odds = []
6      # Hàm lấy phần nguyên và phần dư khi chia cho 2
7      def _whole_odd(integer):
8          # phần nguyên là whole, phần dư là odd.
9          whole = integer // 2;
10         odd = integer % 2;
11         return whole, odd
12
13     (whole, odd) = _whole_odd(integer)
14     odds.append(odd)
15     while (whole > 0): # Khi phần nguyên vẫn lớn hơn 0 thì còn tiếp tục
16         (whole, odd) = _whole_odd(whole)
17         # Lưu lại phần dư sau mỗi lượt chia
18         odds.append(odd)
19     # Revert chuỗi số dư để thu được list nhị phân
20     odds = np.array(odds)[::-1]
21     return odds
22
23  _binary_integer(19)
```

Top

```
1 array([1, 0, 0, 1, 1])
```

Như vậy bạn đã nắm được ý tưởng biến đổi một số nguyên sang hệ nhị phân rồi chứ ? Tiếp theo chúng ta cùng phân tích biểu diễn của một số nguyên đối với định dạng float 32.

Giả sử số đó là 1993 . Khi biểu diễn dưới dạng float 32 thì chúng ta sẽ đưa thêm phần thập phân vào sau nó.

```
1 print("Integer Part: ", "".join([str(i) for i in _binary_integer(1993)])
2 print("Total Bits: ", len(_binary_integer(1993)))
```

```
1 Integer Part: 11111001001
2 Total Bits: 11
```

Phần nguyên chiếm 11 bits, như vậy 21 bits còn lại sẽ là các chữ số 0 ở phần dư và dấu phẩy. Tuy nhiên khi chuyển sang định dạng float 16 thì ta chỉ cần 16 bits để lưu cùng số nguyên ở trên, trong đó 11 bits cho phần nguyên và 5 bits còn lại cho phần dư và dấu phẩy. Số bits sử dụng sẽ giảm một nửa giúp cho mô hình giảm một nửa bộ nhớ lưu trữ.

1.3. Các dạng quantization

Bên dưới là bảng mức độ giảm kích thước (về bộ nhớ, số lượng tham số mô hình không đổi) khi thực hiện quantization theo các kiểu dữ liệu khác nhau.

Technique	Benefits	Hardware
Dynamic range quantization	4x smaller, 2x-3x speedup	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, Microcontrollers
Float16 quantization	2x smaller, GPU acceleration	CPU, GPU

Source: Post training quantization - tensorflow

(https://www.tensorflow.org/lite/performance/post_training_quantization)

Trong bảng trên là 3 kiểu quantization cơ bản với các tính chất khác nhau.

- **Dynamic range quantization:** Đây là kiểu quantization mặc định của tflite. Ý tưởng chính là dựa trên khoảng biến thiên của dữ liệu trên một batch để ước lượng các giá trị quantization. Kiểu quantization này giúp giảm 4 lần kích thước bộ nhớ, tăng tốc độ 2-3 lần trên CPU.
- **Full Integer quantization:** Toàn bộ các hệ số sẽ được chuyển về kiểu số nguyên. Kích thước bộ nhớ giảm 4 lần, tăng 3 lần tốc độ trên CPU, TPU, Microcontrollers.
- **Float16 quantization:** Tất nhiên là sẽ được chuyển về kiểu float và kích thước giảm 2 lần (từ 32 bits về 16 bits). Giúp tăng tốc tính toán trên GPU.

Hiện nay hầu hết các framework deep learning đều hỗ trợ quantization. Trong khuôn khổ của blog này mình chỉ giới thiệu quantization trên framework tensorflow và pytorch.

2. Quantization trên tensorflow

Top

Về Quantization trên tensorflow thì đã có hướng dẫn khá chi tiết cho 3 kiểu ở trên. Các bạn có thể xem tại Post-training quantization

(https://www.tensorflow.org/lite/performance/post_training_quantization).

Mình xin tổng hợp lại các ý chính bao gồm:

- Cách convert model trên tflite.
- Thực hiện quantization.
- Kiểm tra độ chính xác của mô hình sau khi quantization.

Trên tensorflow chúng ta thường quantization mô hình trên tensorflow lite trước khi deploy trên các thiết bị di động. Nếu bạn chưa biết về tensorflow lite thì đây là một định dạng thuộc kiểu FlatBuffer, một cross platform hỗ trợ nhiều ngôn ngữ khác nhau như C++, C#, C, Go, Java, Kotlin, JavaScript, Lobster, Lua, TypeScript, PHP, Python, Rust, Swift. FlatBuffer cho phép serialization data nhanh hơn so với các kiểu dữ liệu khác như Protocol Buffer vì nó bỏ qua quá trình data parsing. Do đó quá trình load model sẽ nhanh hơn đáng kể.

Tiếp theo chúng ta sẽ thực hành Quantization model trên định dạng tflite.

2.1. Convert một model tflite

Tensorflow cung cấp một module là `tf.lite.TFLiteConverter` để convert dữ liệu từ các định dạng như `.pb`, `.h5` sang `.tflite`.

Để đơn giản thì mình sẽ không huấn luyện model từ đầu mà load một pretrain model từ tensorflow hub. Tensorflow hub (<https://www.tensorflow.org/hub>) là một nơi khá tuyệt vời chia sẻ opensource chia sẻ các mô hình pretrain trên tensorflow. Từ hình ảnh, văn bản cho tới tiếng nói. Bạn có thể tìm được rất nhiều các pretrain model tại đây. Đồng thời nếu muốn chia sẻ weight từ các mô hình của mình, bạn cũng có thể trở thành một Tensorhub publisher (<https://www.tensorflow.org/hub/publish>).

Để load một pretrain layer trên tensorflow hub thực hiện như sau:

```
1 import tensorflow as tf
2 import tensorflow_hub as hub
3
4 mobilenet_v2 = tf.keras.Sequential([
5     tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),
6     hub.KerasLayer("https://tfhub.dev/tensorflow/tfgan/eval/mnist/logits.
7     tf.keras.layers.Dense(10, activation='softmax')
8 ])
```

```
1 mobilenet_v2.summary()
```

Top

```

1 Model: "sequential"
2
3 Layer (type)                Output Shape                Param #
4 =====
5 keras_layer (KerasLayer)    (None, 1792)                4363712
6
7 dense (Dense)               (None, 10)                  17930
8 =====
9 Total params: 4,381,642
10 Trainable params: 17,930
11 Non-trainable params: 4,363,712
12

```

Huấn luyện lại model trên tập dữ liệu mnist

```

1 # Load MNIST dataset
2 mnist = tf.keras.datasets.mnist
3 (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
4
5 # Normalize the input image so that each pixel value is between 0 to 1
6 train_images = train_images.astype(np.float32) / 255.0
7 test_images = test_images.astype(np.float32) / 255.0
8
9
10 mobilenet_v2.compile(optimizer='adam',
11                      loss=tf.keras.losses.SparseCategoricalCrossentropy(
12                          from_logits=True),
13                      metrics=['accuracy'])
14 mobilenet_v2.fit(
15     train_images,
16     train_labels,
17     epochs=5,
18     validation_data=(test_images, test_labels)
19 )

```

```

1 Epoch 1/5
2 1875/1875 [=====] - 58s 31ms/step - loss: 1.14
3 Epoch 2/5
4 1875/1875 [=====] - 58s 31ms/step - loss: 1.14
5 Epoch 3/5
6 1875/1875 [=====] - 57s 31ms/step - loss: 1.14
7 Epoch 4/5
8 1875/1875 [=====] - 57s 31ms/step - loss: 1.14
9 Epoch 5/5
10 1875/1875 [=====] - 57s 31ms/step - loss: 1.14
11
12
13
14
15
16 <tensorflow.python.keras.callbacks.History at 0x7fb6674dd8d0>

```

Top

Model của chúng ta được khởi tạo từ keras nên nó có định dạng h5 . Chúng ta có thể convert model sang .tflite .

```

1  import pathlib
2
3  converter = tf.lite.TFLiteConverter.from_keras_model(mobilenet_v2)
4  mobilenetv2_tflite = pathlib.Path("/tmp/mobilenet_v2.tflite")
5  mobilenetv2_tflite.write_bytes(converter.convert())
6  !ls -lh /tmp

1  INFO:tensorflow:Assets written to: /tmp/tmpua53ldhf/assets
2
3
4  INFO:tensorflow:Assets written to: /tmp/tmpua53ldhf/assets
5
6
7  total 13M
8  -rw-r--r-- 1 root root 13M Nov 21 14:24 mobilenet_v2.tflite
9  drwxr-xr-x 2 root root 4.0K Nov 21 13:57 __pycache__
10 -rw----- 1 root root 461 Nov 21 13:36 tmp2gra8nm1.py
11 -rw----- 1 root root 5.7K Nov 21 13:35 tmp84s7w721.py
12 -rw----- 1 root root 1.1K Nov 21 13:36 tmpeiaoz8w9.py
13 -rw----- 1 root root 2.6K Nov 21 13:35 tmpmqhsez24.py
14 -rw----- 1 root root 931 Nov 21 13:57 tmpndkkbpy7.py
15 -rw----- 1 root root 3.5K Nov 21 13:35 tmpx7jgybfh.py
16 -rw----- 1 root root 931 Nov 21 13:56 tmpzeqpyi47.py

```

Như vậy ở định dạng tflite chúng ta có mô hình mobilenet_v2.tflite có kích thước là 13 Mb.

2.2. Quantization model

Để quantization được các layer input, output và layer trung gian thì chúng ta phải thông qua RepresentativeDataset, RepresentativeDataset là một generator function có kích thước đủ lớn, có tác dụng ước lượng khoảng biến thiên cho toàn bộ các tham số của mô hình.


```

1 def _quantization_int8(model_path):
2     def representative_data_gen():
3         # prepare batch with batch_size = 100
4         for input_value in tf.data.Dataset.from_tensor_slices(train_image:
5             input_value = tf.reshape(input_value, (1, 28, 28, 1))
6             yield [input_value]
7         # Declare optimizations model
8         converter.optimizations = [tf.lite.Optimize.DEFAULT]
9         # representative dataset
10        converter.representative_dataset = representative_data_gen
11        # Declare target specification data type
12        converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS]
13        # Define input and output data type
14        converter.inference_input_type = tf.uint8
15        converter.inference_output_type = tf.uint8
16        tflite_model_quant = converter.convert()
17        mobilenetv2_quant_tflite = pathlib.Path(model_path)
18        mobilenetv2_quant_tflite.write_bytes(tflite_model_quant)
19        return tflite_model_quant
20
21    tflite_model_quant = _quantization_int8("/tmp/mobilenet_v2_quant.tflite")

```

```

1 INFO:tensorflow:Assets written to: /tmp/tmp7_vyq_ud/assets
2
3
4 INFO:tensorflow:Assets written to: /tmp/tmp7_vyq_ud/assets

```

Trong hàm `_quantization_int8` Chúng ta phải khai báo các tham số cho converter. Trong đó bao gồm các nội dung:

- `optimizations`: Phương pháp tối ưu được sử dụng để tính toán quantization sao cho sai số là nhỏ nhất so với định dạng gốc.
- `representative_dataset`: Generator trả ra một batch input **X** cho mô hình. Có kích thước đủ lớn để tính toán dynamic range cho các tham số của mô hình.
- `target_spec`: Định dạng tính toán của các operation trong mạng deep learning.
- `inference_input_type`: Định dạng input.
- `inference_output_type`: Định dạng output.

Định dạng input, output và `target_spec` phải chung một kiểu với định dạng dữ liệu quantization.

Trước khi quantization thì định dạng của input và output là float32, sau quantization thì chúng được chuyển về int8.

```

1 interpreter = tf.lite.Interpreter(model_content=tflite_model_quant)
2 input_type = interpreter.get_input_details()[0]['dtype']
3 print('input: ', input_type)
4 output_type = interpreter.get_output_details()[0]['dtype']
5 print('output: ', output_type)

1 input:  <class 'numpy.uint8'>
2 output:  <class 'numpy.uint8'>

```

Top

Kích thước mô hình sau khi chuyển đổi

```
1 !ls -lh /tmp
```

```
1 total 16M
2 -rw-r--r-- 1 root root 3.2M Nov 21 14:45 mobilenet_v2_quant.tflite
3 -rw-r--r-- 1 root root 13M Nov 21 14:24 mobilenet_v2.tflite
4 drwxr-xr-x 2 root root 4.0K Nov 21 13:57 __pycache__
5 -rw----- 1 root root 461 Nov 21 13:36 tmp2gra8nm1.py
6 -rw----- 1 root root 5.7K Nov 21 13:35 tmp84s7w721.py
7 -rw----- 1 root root 1.1K Nov 21 13:36 tmpeiaoz8w9.py
8 -rw----- 1 root root 2.6K Nov 21 13:35 tmpmqhsez24.py
9 -rw----- 1 root root 931 Nov 21 13:57 tmpndkkbpy7.py
10 -rw----- 1 root root 3.5K Nov 21 13:35 tmpx7jgybfh.py
11 -rw----- 1 root root 931 Nov 21 13:56 tmpzeqpyi47.py
```

Chúng ta có thể thấy quantization đã làm giảm kích thước file tflite so với ban đầu gấp 4 lần. Bạn đọc cũng có thể tự kiểm tra tốc độ inference của mô hình sau khi quantization.

2.3. Độ chính xác của mô hình quantization

Sau khi thực hiện quantization chúng ta luôn phải kiểm tra lại độ chính xác trên một tập validation độc lập. Quá trình này đảm bảo mô hình sau quantization có chất lượng không quá giảm so với mô hình gốc.

Kiểm tra độ chính xác mô hình trên tập validation

Top

```

1 def _run_tflite_model(model_file, test_images, test_image_indices):
2     # Khởi tạo model tf.lite
3     interpreter = tf.lite.Interpreter(model_path=str(model_file))
4     interpreter.allocate_tensors()
5
6     input_details = interpreter.get_input_details()[0]
7     output_details = interpreter.get_output_details()[0]
8
9     # Dự báo
10    predictions = np.zeros((len(test_image_indices),), dtype=int)
11    for i, test_image_index in enumerate(test_image_indices):
12        test_image = tf.reshape(test_images[test_image_index], (28, 28, 1))
13        test_label = test_labels[test_image_index]
14
15        # Kiểm tra xem input đã được quantized, sau đó rescale input data
16        if input_details['dtype'] == np.uint8:
17            input_scale, input_zero_point = input_details["quantization"]
18            test_image = test_image / input_scale + input_zero_point
19
20        test_image = np.expand_dims(test_image, axis=0).astype(input_details['dtype'])
21        interpreter.set_tensor(input_details["index"], test_image)
22        interpreter.invoke()
23        output = interpreter.get_tensor(output_details["index"])[0]
24        predictions[i] = output.argmax()
25    return predictions
26
27    # Helper function to evaluate a TFLite model on all images
28    def _evaluate_model(model_file, model_type):
29        global test_images
30        global test_labels
31
32        test_image_indices = range(test_images.shape[0])
33        predictions = _run_tflite_model(model_file, test_images, test_image_indices)
34
35        accuracy = (np.sum(test_labels == predictions) * 100) / len(test_images)
36
37        print('%s model accuracy is %.4f%% (Number of test samples=%d)' % (
38            model_type, accuracy, len(test_images)))
39
40    _evaluate_model("/tmp/mobilenet_v2_quant.tflite", model_type="unit8")

```

```

1 Float model accuracy is 98.8200% (Number of test samples=10000)

```

Ta thấy sau khi quantization thì độ chính xác của mô hình giảm khoảng 0.05%. Đây là một mức giảm không đáng kể so với lợi ích đạt được từ việc giảm kích thước mô hình và tốc độ inference.

2.3.1. Convert model by commandline

Ngoài sử dụng `tf.lite.TFLiteConverter` để convert model thì chúng ta có thể convert trên commandline. Để tránh các lỗi phát sinh thì kinh nghiệm của mình là các bạn nên convert từ save mode của mô hình (save model là định dạng lưu trữ được cả graph của mô hình và tham số trong các checkpoint, do đó hạn chế được lỗi so với phương pháp chỉ lưu tham số).

Top

Lưu mô hình dưới dạng savemode

```

1  mobilenet_v2.save("/tmp/mobilenet_v2")
2  !ls /tmp/mobilenet_v2/

1  INFO:tensorflow:Assets written to: /tmp/mobilenet_v2/assets
2
3
4  INFO:tensorflow:Assets written to: /tmp/mobilenet_v2/assets
5
6
7  assets  saved_model.pb  variables

1  !tflite_convert \
2  --saved_model_dir=/tmp/mobilenet_v2 \
3  --output_file=/tmp/mobilenet_v2_quant_cmd.tflite
4
5  !ls -lh /tmp

1  2020-11-21 15:37:17.750486: I tensorflow/stream_executor/platform/default/dso_loader.cc:44: Successfully opened dynamic library libcudart.so.10.1
2  2020-11-21 15:37:17.759721: E tensorflow/stream_executor/cuda/cuda_driver.cc:319: failed to initialize the driver API: cudaError_40_incompatible_driver_version
3  2020-11-21 15:37:17.759803: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156: kernel path is not defined
4  2020-11-21 15:37:17.786543: I tensorflow/core/platform/profile_utils/cpu_utils.cc:113: CPU features: 0x1000000000000000
5  2020-11-21 15:37:17.786937: I tensorflow/compiler/xla/service/service.cc:169: xla service 0
6  2020-11-21 15:37:17.786990: I tensorflow/compiler/xla/service/service.cc:169: xla service 0
7  2020-11-21 15:37:18.172092: I tensorflow/core/grappler/devices.cc:55]
8  2020-11-21 15:37:18.172260: I tensorflow/core/grappler/clusters/single_device.cc:92]
9  2020-11-21 15:37:18.298400: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:816]
10 2020-11-21 15:37:18.298458: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:816]
11 2020-11-21 15:37:18.298470: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:816]
12 2020-11-21 15:37:18.523775: I tensorflow/core/grappler/devices.cc:55]
13 2020-11-21 15:37:18.523968: I tensorflow/core/grappler/clusters/single_device.cc:92]
14 2020-11-21 15:37:18.711797: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:816]
15 2020-11-21 15:37:18.711856: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:816]
16 2020-11-21 15:37:18.711867: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:816]
17 total 29M
18 drwxr-xr-x 4 root root 4.0K Nov 21 15:36 mobilenet_v2
19 -rw-r--r-- 1 root root 13M Nov 21 15:37 mobilenet_v2_quant_cmd.tflite
20 -rw-r--r-- 1 root root 3.2M Nov 21 14:45 mobilenet_v2_quant.tflite
21 -rw-r--r-- 1 root root 13M Nov 21 14:24 mobilenet_v2.tflite
22 drwxr-xr-x 2 root root 4.0K Nov 21 13:57 __pycache__
23 -rw----- 1 root root 461 Nov 21 13:36 tmp2gra8nm1.py
24 -rw----- 1 root root 5.7K Nov 21 13:35 tmp84s7w721.py
25 -rw----- 1 root root 1.1K Nov 21 13:36 tmpeiaoz8w9.py
26 -rw----- 1 root root 2.6K Nov 21 13:35 tmpmqhsez24.py
27 -rw----- 1 root root 931 Nov 21 13:57 tmpndkkbpy7.py
28 -rw----- 1 root root 730 Nov 21 15:37 tmpPSeMCj
29 -rw----- 1 root root 3.5K Nov 21 13:35 tmpx7jgybfh.py
30 -rw----- 1 root root 931 Nov 21 13:56 tmpzeqpyi47.py

```

2.3.2. Quantization trên pytorch

Top

Trên pytorch cũng hỗ trợ ba định dạng quantization tương tự như tensorflow. Về hướng dẫn quantization cho các model trên pytorch thì các bạn có thể tham khảo tại hướng dẫn pytorch - quantization (<https://pytorch.org/docs/stable/quantization.html>), khá đầy đủ và chi tiết.

Mình sẽ hướng dẫn các bạn cách thực hiện quantization theo Dynamic Quantization .

Đầu tiên các bạn cần xây dựng một CNN network bao gồm các CNN layers và fully connected layers. Tiếp theo đó chúng ta sẽ sử dụng hàm `torch.quantization.quantize_dynamic()` cho tác vụ quantization.

```

1      import torch
2      from torch import nn
3      import torch.nn.functional as F
4
5      # Khởi tạo một network
6      class Net(nn.Module):
7          def __init__(self):
8              super(Net, self).__init__()
9              self.conv1 = nn.Conv2d(3, 6, 5)
10             self.pool = nn.MaxPool2d(2, 2)
11             self.conv2 = nn.Conv2d(6, 16, 5)
12             self.fc1 = nn.Linear(16 * 5 * 5, 120)
13             self.fc2 = nn.Linear(120, 84)
14             self.fc3 = nn.Linear(84, 10)
15
16         def forward(self, x):
17             x = self.pool(F.relu(self.conv1(x)))
18             x = self.pool(F.relu(self.conv2(x)))
19             x = x.view(-1, 16 * 5 * 5)
20             x = F.relu(self.fc1(x))
21             x = F.relu(self.fc2(x))
22             x = self.fc3(x)
23             return x
24
25     # khởi tạo model instance
26     model_fp32 = Net()
27     # khởi tạo quantize model instance
28     model_int8 = torch.quantization.quantize_dynamic(
29         model_fp32, # model gốc
30         {torch.nn.Linear,
31          torch.nn.Conv2d}, # một tập hợp các layers cho dynamically quan
32         dtype=torch.qint8) # target dtype cho các quantized weights
33
34     # run model
35     input_fp32 = torch.randn(1, 3, 32, 32)
36     res = model_int8(input_fp32)

```

```

1      model_int8.parameters

```

Top

```

1 <bound method Module.parameters of Net(
2 (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
3 (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil
4 (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
5 (fc1): DynamicQuantizedLinear(in_features=400, out_features=120, dtype:
6 (fc2): DynamicQuantizedLinear(in_features=120, out_features=84, dtype=
7 (fc3): DynamicQuantizedLinear(in_features=84, out_features=10, dtype=t
8 )>

```

Ta thấy định dạng của các layers Linear và Conv2d model đã được chuyển qua dạng int8. Về thực quan thì mình cảm thấy cách convert Dynamic Quantization trên pytorch đơn giản hơn so với tensorflow khá nhiều. Chúng ta không phải dựa vào RepresentativeDataset, chỉ cần truyền các thông số vào hàm `torch.quantization.quantize_dynamic()` bao gồm model gốc, các layers cần quantize và định dạng mục tiêu cần quantize.

3. Kết luận

Như vậy qua bài viết này mình đã hướng dẫn tới các bạn ý tưởng chính của quantization. Các thức để tiến hành quantization trên tensorflow lite và kiểm tra độ chính xác của mô hình sau quantization.

Đây là một trong những phương pháp được sử dụng phổ biến trong quá trình xây dựng và triển khai các model deep learning trên những thiết bị có cấu hình yếu. Bên cạnh phương pháp quantization thì còn có nhiều phương pháp khác như pruning, weight-sharing, Knowledge Distillation, binary network cũng giúp cho mô hình có kích thước nhẹ hơn và tăng tốc độ inference mà mình sẽ tiếp tục giới thiệu tới các bạn trong những bài sắp tới.

Cảm ơn các bạn đã theo dõi blog, đừng ngại thử thách và học hỏi.

4. Tài liệu tham khảo

1. Post training quantization - tensorflow (https://www.tensorflow.org/lite/performance/post_training_quantization)
2. Pytorch quantization (<https://pytorch.org/docs/stable/quantization.html>)
3. mxnet quantization (<https://medium.com/apache-mxnet/model-quantization-for-production-level-neural-network-inference-f54462ebba05>)
4. A Tale of Model Quantization in TF Lite (<https://towardsdatascience.com/a-tale-of-model-quantization-in-tf-lite-aeb09f255ca>)
5. How to accelerate and compress neural networks with quantization (<https://towardsdatascience.com/how-to-accelerate-and-compress-neural-networks-with-quantization-edfbbabb6af7>)
6. A Survey of Model Compression and Acceleration for Deep Neural Networks (<https://arxiv.org/pdf/1710.09282.pdf>)
7. Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding (<https://arxiv.org/pdf/1510.00149.pdf>)
8. Learning both Weights and Connections for Efficient Neural Networks (<https://arxiv.org/pdf/1506.02626.pdf>)
9. EIE: Efficient Inference Engine on Compressed Deep Neural Network (<https://arxiv.org/pdf/1602.01528.pdf>)

Top