УТВЕРЖДЕН
RU.РМДВ.04.06.003-01 97 01-ЛУ

КОМПЛЕКТ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РАБОТЫ С УСКОРИТЕЛЕМ ТЕНЗОРНЫХ ВЫЧИСЛЕНИЙ МИКРОПРОЦЕССОРА IVA H

Руководство пользователя.

RU.РМДВ.04.06.003-01 97 01

Листов 61

Инв. № по	Подп. и дата	Взам. инв.	Инв. №	Подп. и дата

2021

Литера

АННОТАЦИЯ

В настоящем программном документе приведено руководство пользователя по работе с комплектом программного обеспечения для работы с ускорителем тензорных вычислений микропроцессора IVA H (Фреймворк FW_TPU_H). Фреймворк FW_TPU_H предназначен для преобразования оригинальных графов нейронных сетей в набор инструкций ускорителя тензорных вычислений микропроцессора IVA H. Программа обеспечивает возможность генерации исполняемой программы для IVA H посредством последовательного выполнения процедур квантования и компиляции. Основными функциями программы является предоставление пользовательского доступа к процессу квантования и компиляции с помощью API языка Python 3.8 и интерфейса командной строки.

Фреймворк FW_TPU_H разработан для использования в отраслях, применяющих алгоритмы искусственных нейронных сетей, компьютерного зрения, распознавания по голосу, машинного обучения и других методов искусственного интеллекта.

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

1.1 Наименование и обозначение программы

1.1.1 Наименование программы

Наименование — «Комплект программного обеспечения для работы с ускорителем тензорных вычислений микропроцессора IVA H» (далее – Программа и/или «Фреймворк FW_TPU_H»).

В настоящем документе следующие используемые термины считаются равнозначными:

1) Комплект программного обеспечения для работы с ускорителем тензорных вычислений микропроцессора IVA H;
2) фреймворк;
3) фреймворк FW_TPU_H;
4) программное обеспечение фреймворка;
5) программа.

1.1.2. Обозначение программы

Обозначение программы – RU.РМДВ.04.06.003-01.

1.2. Назначение и области применения программы

Фреймворк FW_TPU_H предназначен для преобразования оригинальных графов нейронных сетей в набор инструкций ускорителя тензорных вычислений микропроцессора IVA H. Программа обеспечивает возможность генерации исполняемой программы для IVA H посредством последовательного выполнения процедур квантования и компиляции. Основными функциями программы является предоставление пользовательского доступа к процессу квантования и компиляции с помощью API языка Python 3.8 и интерфейса командной строки.

Фреймворк FW_TPU_H разработан для использования в отраслях, применяющих алгоритмы искусственных нейронных сетей, компьютерного

зрения, распознавания по голосу, машинного обучения и других методов искусственного интеллекта.

1.3. Требования к аппаратному и программному обеспечению, необходимым для функционирования программы

Для корректного функционирования фреймворка FW_TPU_H к аппаратному обеспечению и системным программным средствам, используемым программой, предъявляются следующие характеристики:

- Операционная система: Linux Ubuntu 18.04 LTS и выше
- Процессор: не хуже Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
- Оперативная память: не менее 32 Гб
- HDD/SSD: не менее 512 Гб
- TensorFlow: 2.4.1
- Видеокарта: не хуже nVidia 1080Ti
- Компилятор: GCC 7.3.1
- cuDNN: 8.0
- CUDA: 11.0
- Python 3.8.6

2. ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПО ФРЕЙМВОРКА FW_TPU_H

2.1. Краткое описание пакета поставляемых файлов

В комплект программного обеспечения для работы с ускорителем тензорных вычислений микропроцессора IVA H входят следующие компоненты:

– «SMP.zip» - архив, содержащий основные исполнительные файлы фреймворка FW_TPU_H;

– «installer.sh» - файл установщика фреймворка FW_TPU_H;

– «dnn_quant.rst» - файл, содержащий описание примера использования утилиты квантования, входящей в состав фреймворка FW_TPU_H.

2.2. Инструкция по установке ПО фреймворка FW_TPU_H

Для выполнения процедуры установки ПО фреймворка FW_TPU_H необходимо выполнить следующие действия:

1) Распаковать архив SMP.zip (распаковка архива выполняется в удобную для пользователя директорию);

2) Открыть консоль/терминал операционной системы Linux Ubuntu;

3) В открывшейся консоли набрать и выполнить команду:

```
cp installer.sh && cd SMP && sh installer.sh && cd ..
```

Данная команда выполняет копирование файлов установщика в директорию распакованного архива SMP.zip и выполняет процедуру его последующей установки с помощью пакетного менеджера Python. О корректности выполнения процедуры установки и настройки ПО фреймворка информируют соответствующие сообщения установщика. По итогам выполнения указанных шагов ПО фреймворка установлено и настроено для последующего использования.

Примечание 1.

Рекомендуется обновить до актуальной версии средства установки ПО в среде ОС Linux Ubuntu с помощью команды:

```
$ pip install -U pip setuptools
```

Примечание 2.

Для осуществления поддержки TensorFlow на графическом процессоре (GPU) необходимо использовать руководство, размещенное по адресу: https://www.tensorflow.org/install/gpu

3. РЕКОМЕНДАЦИИ ПО ЭКСПЛУАТАЦИИ ПО ФРЕЙМВОРКА FW_TPU_H

Ознакомиться с инструкцией по запуску и настройке процесса квантования возможно посредством набора в консоли операционной системы Linux Ubuntu и последующего выполнения команды:

```
quantize_graph --help
```

С примером запуска ПО из базового использования утилиты квантования возможно ознакомиться по средствам запуска файла с именем: «dnn_quant.rst».

Ознакомиться с инструкцией по запуску и настройке процесса компиляции возможно по средствам набора в консоли операционной системы Linux Ubuntu и последующего выполнения команды:

```
tcf --help
```

4. РУКОВОДСТВО ПО БЫСТРОМУ СТАРТУ

4.1 Инструкция по работе

Для осуществления инференса нейронной сети ResNet50 (в качестве примера) из модуля tf.keras.applications (https://www.tensorflow.org/api_docs/python/tf/keras/applications) необходимо выполнить следующие команды:

```
import os
from PIL import Image
import tensorflow as tf
from iva_applications import imagenet
from iva_applications.resnet50 import image_to_tensor
from iva_applications.utils import TPURunner
from iva_applications.utils import freeze_keras_model
from iva_applications.utils import pb_to_tensorboard_event
from iva_applications.resnet50 import
save_calibration_tensor
```

Определение предварительно обученной нейронной сети осуществляется с помощью команды:

```
resnet50 = tf.keras.applications.ResNet50()
```

Чтобы получить исходный граф нейронной сети необходимо выполнить следующую команду:

```
freeze_keras_model(resnet50, save_dir='work_dir')
```

Поиск входящих и исходящих узлов (нодов) исходного графа осуществляется путем экспорта TensorBoard Event с помощью функции pb_to_tensorboard_event из модуля iva_applications:

```
pb_to_tensorboard_event('work_dir/resnet50.pb',
'work_dir/event')
#!tensorboard --logdir=work_dir/event
```

После определения входных и выходных имен узлов исходного графа и выяснения необходимости включения некоторых узлов в «предварительную обработку» (preprocessing) калибровочный тензор сохраняется путем выполнения следующей команды:

```
save_calibration_tensor('path/to/images', 'work_dir')
```

Квантование исходного графика с помощью интерфейса командной строки (порядок калибровочных тензоров и имена входных узлов должны быть идентичными) происходит путем выполнения следующей команды:

```
!quantize_graph work_dir/resnet50.pb
work_dir/calibration_tensors_resnet.npy \
--input_node=input \
--output_node=resnet50/predictions/Softmax \
--save_dir=work_dir/quantized_data
```

Для получения программы TPU необходимо выполнить компиляцию квантованных данных с помощью команды:

```
!tcf work_dir/quantized_data/to_compile/tpu.conf.json \
work_dir/quantized_data/to_compile/quant_constants.pickle \
fpga_64x64 --dump-tpu-program-to work_dir/resnet50.tpu
```

Последним шагом является инициализация TPURunner программой TPU и ее инференс. После инициализации TPURunner IVA TPU готов к расчету нейронной сети. Когда вызывается TPURunner нейронная сеть запускается на IVA TPU:

```
program_path = 'work_dir/resnet50.tpu'
# read an image
with open(os.path.join('path/to/images', 'image.jpg'), 'rb')
as img_file:
image = Image.open(img_file)
# preprocess data
tensor = image_to_tensor(image)
# TPURunner initialization
runner = TPURunner(program_path)
# get input and output nodes names
input_nodes = runner.input_nodes
output_nodes = runner.output_nodes
# inference neural network using IVA TPU
output = runner(
{
input_nodes[0]:tensor
}
)
# Decode result to classes.
result = imagenet.tpu_tensor_to_classes(
output[output_nodes[0]])
```

Список нейронных сетей, поддерживаемых iva_applications, представлен в таблице 1:

Таблица 1 – Список нейронных сетей, поддерживаемых iva_applications

Название поддерживаемой нейронной сети	**Ссылка на оригинальный граф**
covidnet_cxr4_	https://drive.google.com/file/d/1yDZVzwMoj023w84qDOCjAlU43jp9Fyhv/view
efficientnet_b0	https://drive.google.com/file/d/1pxdz9mxcY9aMFEIo0-culv8Y6FEm8Liv/view

Название поддерживаемой нейронной сети	**Ссылка на оригинальный граф**
efficientnet_b1	https://drive.google.com/file/d/18VtAxfXeTRTVeE6wGwQqlI0v__Op2Z3s/view
efficientnet_b2	https://drive.google.com/file/d/1heEdzDeKkgW_dNyL63dw5FDwShA-1BEI/view
efficientnet_b3	https://drive.google.com/file/d/1e33WUHWzPwsjf1u2ZWWMl2TrxbJZFZg5/view
efficientnet_b4	https://drive.google.com/file/d/1Ap151ed9NqpdNGxq7ZpRJ1tTsfQh-VMA/view
efficientnet_b5	https://drive.google.com/file/d/1rzAmTw43oBPBflj4LDsJRqd-u0d_nuf-/view
inception_v1	https://drive.google.com/file/d/1jAIC6FAPtP_eNpfHi9nkfRDv9VKBUSek/view
inception_v2	https://drive.google.com/file/d/1wKxf8ASp0kqfGrtZCD8Br-MJFNj828gf/view
inception_v3	https://drive.google.com/file/d/12vbnpUceCJR795X3640hHkxPzXocFXAt/view
inception_v4	https://drive.google.com/file/d/1nlKiwBX8iyLBjpseXQX6iE0_2cvoeTam/view

Название поддерживаемой нейронной сети	**Ссылка на оригинальный граф**
lenet5	https://drive.google.com/file/d/1x_4Q5j9styS8zwm229V4g3FBBpRYskho/view
resnet34	https://drive.google.com/file/d/1W_t5dGvLXtoA0ANQsX5P2RfWhDik5bvE/view
resnet50	https://drive.google.com/file/d/1X8xhBZmYwcbnmAy8twjiDDgnNsZF-062/view
seresnet50	https://drive.google.com/file/d/1WSUj67PdMd_6MvGt9t4TcmZv-FZJZFM4/view
ssd_mobilenet_v1	https://drive.google.com/file/d/1fTrQH-BdYCmUWtT5RyxB59Stiv82y4Xp/view
ssd_mobilenet_v2	https://drive.google.com/file/d/16LkpBCZvVD1BmkTFtzF68wuyjlfUECVu/view
tiny_yolo2	https://drive.google.com/file/d/1QEULXH1YmRAsotn5eLekBa_Qi9b786Ia/view
tiny_yolo3	https://drive.google.com/file/d/1cAecqcwsg2xelvNDeKhz0iU9v9UPXEIl/view
yolo2	https://drive.google.com/file/d/1IpDgUn5feNj9r_LSalAGNveyHpGKlDY4/view

Название поддерживаемой нейронной сети	**Ссылка на оригинальный граф**
yolo3	https://drive.google.com/file/d/1eKXooXejjxWmLaKhKtP59zos-h7g745m/view
yolo4	https://drive.google.com/file/d/13CjRPEMpX2N7K_KxZdHgR5N3H0zmc7bb/view
vgg19	https://drive.google.com/file/d/1bNC_0jX2eTsw4AgLen53umB2Jg8vX3LZ/view

4.2 Список поддерживаемых операций

Список поддерживаемых операций представлен в таблице 2:

Таблица 2 – Список поддерживаемых операций

Операция TensorFlow	**Статус поддержки**	**Примечание**
tf.boolean_mask	Отложено	
tf.cast	Отложено	
tf.clip_by_value	Отложено	
tf.compat.v1.placeholder	Отложено	
tf.concat	Частичная	Объединяет тензоры только

Операция TensorFlow	**Статус поддержки**	**Примечание**
		по размеру каналов.
tf.expand_dims	Отложено	
tf.gather	Отложено	
tf.image.extract_patches	Отложено	
tf.image.resize	Полная	Только целочисленный ввод. Произведена декомпозиция с помощью ResizeReplacement
tf.keras.activations.elu	Отложено	
tf.keras.activations.gelu	Отложено	
tf.keras.activations.hard_sigmoid	Отложено	
tf.keras.activations.selu	Полная	Произведена декомпозиция с помощью SeluReplacement
tf.keras.activations.sigmoid	Полная	
tf.keras.activations.softmax	Частичная	Вычисляет только по осям HWC
tf.keras.activations.swish	Полная	Произведена декомпозиция с помощью SwishDecomposition
tf.keras.activations.tanh	Полная	
tf.keras.backend.expand_dims	Отложено	

Операция TensorFlow	**Статус поддержки**	**Примечание**
tf.keras.layers.Add	Полная	
tf.keras.layers.AveragePooling1D	Отложено	
tf.keras.layers.AveragePooling2D	Полная	
tf.keras.layers.AveragePooling3D	Отложено	
tf.keras.layers.Concatenate	Частичная	Объединяет тензоры только по размеру каналов
tf.keras.layers.Conv1D	Отложено	
tf.keras.layers.Conv2D	Частичная	Вычисляет двумерную свертку с расширениями [1, 1, 1, 1]
tf.keras.layers.Conv2DTranspose	Частичная	Произведена декомпозиция с помощью Conv2DTransposeReplacement
tf.keras.layers.Dense	Полная	
tf.keras.layers.DepthwiseConv2D	Частичная	Глубинная двумерная свертка с расширением (1, 1)
tf.keras.layers.Flatten	Частичная	Эквивалентно Flatten: преобразовывает только тензор [1, h, w, c] в [1, 1, 1, h * w * c]

Операция TensorFlow	**Статус поддержки**	**Примечание**
tf.keras.layers.GRU	Отложено	
tf.keras.layers.InputLayer	Отложено	
tf.keras.layers.LSTM	Отложено	
tf.keras.layers.LayerNormalization	Частичная	Нормализует входы по оси каналов
tf.keras.layers.LeakyReLU	Полная	
tf.keras.layers.MaxPool1D	Отложено	
tf.keras.layers.MaxPool3D	Отложено	
tf.keras.layers.Multiply	Полная	
tf.keras.layers.PReLU	Отложено	
tf.keras.layers.ReLU	Полная	
tf.keras.layers.Reshape	Частичная	Эквивалентно Flatten: преобразовывает только тензор [1, h, w, c] в [1, 1, 1, h * w * c]
tf.keras.layers.Subtract	Отложено	
tf.keras.layers.maximum	Отложено	
tf.linalg.matmul	Частичная	Только для матриц с размерами, кратными CWL

Операция TensorFlow	**Статус поддержки**	**Примечание**
tf.math.add	Полная	
tf.math.exp	Полная	
tf.math.greater	Отложено	
tf.math.log	Отложено	
tf.math.maximum	Отложено	
tf.math.multiply	Полная	
tf.math.multiply & tf.math.add	Полная	
tf.math.reduce_mean	Частичная	Вычисляет среднее значение элементов по указанным размерам кроме оси каналов
tf.math.reduce_sum	Частичная	Вычисляет сумму элементов по указанным размерам кроме оси каналов
tf.math.sigmoid	Полная	
tf.math.square	Отложено	
tf.math.subtract	Отложено	
tf.math.tanh	Полная	
tf.nn.avg_pool	Полная	
tf.nn.avg_pool1d	Отложено	

Операция TensorFlow	**Статус поддержки**	**Примечание**
tf.nn.avg_pool3d	Отложено	
tf.nn.conv1d	Отложено	
tf.nn.conv2d	Частичная	Вычисляет двумерную свертку с расширениями [1, 1, 1, 1]
tf.nn.conv2d_transpose	Частичная	Произведена декомпозиция с помощью Conv2DTransposeReplacement
tf.nn.depth_to_space	Полная	
tf.nn.depthwise_conv2d	Частичная	Глубинная двумерная свертка с расширением (1, 1)
tf.nn.elu	Отложено	
tf.nn.gelu	Отложено	
tf.nn.leaky_relu	Полная	
tf.nn.local_response_normalization	Отложено	
tf.nn.max_pool	Полная	
tf.nn.max_pool1d		
tf.nn.max_pool3d	Отложено	
tf.nn.relu	Полная	

Операция TensorFlow	**Статус поддержки**	**Примечание**
tf.nn.relu6	Отложено	
tf.nn.selu	Полная	Произведена декомпозиция с помощью SeluReplacement
tf.nn.softmax	Частичная	Вычисляет только по осям HWC
tf.nn.space_to_depth	Полная	
tf.nn.swish	Полная	Произведена декомпозиция с помощью SwishDecomposition
tf.pad	Отложено	
tf.raw_ops.Exp	Полная	
tf.raw_ops.Log	Отложено	
tf.raw_ops.MatMul	Частичная	Только для матриц с размерами кратными CWL
tf.raw_ops.MaxPool	Полная	
tf.raw_ops.Pack	Отложено	
tf.raw_ops.Rsqrt	Отложено	
tf.raw_ops.Select	Отложено	
tf.raw_ops.Square	Отложено	
tf.reshape	Частичная	Эквивалентно Flatten:

Операция TensorFlow	**Статус поддержки**	**Примечание**
		преобразовывает только тензор [1, h, w, c] в [1, 1, 1, h * w * c]
tf.reverse	Отложено	
tf.reverse_sequence	Отложено	
tf.shape	Отложено	
tf.sparse.to_dense	Отложено	
tf.split	Отложено	
tf.squeeze	Полная	
tf.strided_slice	Частичная	Извлекает полосатый фрагмент тензора с шагами (1, 1) и ellipsis_mask == new_axis
tf.tensor_scatter_nd_add	Частичная	Объединяет тензоры только по размеру каналов
tf.tensordot	Отложено	
tf.transpose	Полная	

Легенда:

Полная – полная поддержка и имплементация операции;

Частичная – частичная поддержка (см. примечания);

Отложено – на текущий момент приостановлено.

4.3 Список поддерживаемых нейронных сетей

Далее представлен список нейронных сетей, протестированных с использованием tpu_framework:

- bisenet;
- covidnet;
- densenet;
- efficientnet;
- inception (v1, v2, v3, v4);
- mobilenet (v1, v2);
- lenet5;
- pnasnet_large;
- pointnet;
- resnet (34, 50, 50v2, 152);
- rfdnet;
- selu;
- seresnet50;
- ssd_mobilenet (v1, v2);
- ssd;
- tiny_yolo (v2, v3);
- yolo (v2, v3, v4);
- vgg19;
- xception;

5. ДОКУМЕНТАЦИЯ НА МОДУЛЬ DNN_QUANT

Особенностью TPU является использование форматов данных с низкой точностью для экономии памяти и повышения производительности вычислений.

Как правило, нейронные сети обучаются в 32-битном типе данных с плавающей запятой, поэтому их сначала необходимо преобразовать в форматы данных TPU.

Эта процедура называется квантованием нейронных сетей.

Модуль dnn_quant помогает выполнить эту процедуру, тем самым подготовив данные для компиляции.

5.1 Установка модуля dnn_quant

5.1.1 Требования к среде функционирования модуля

Модуль dnn_quant функционирует в среде Python 3.8.6

Рекомендуется обновить утилиты pip и setuptools командой:

```
$ pip install -U pip setuptools
```

5.1.2 Установка модуля dnn_quant

Для завершения установки модуля dnn_quant используйте установщик Python:

```
$ pip install dnn_quant
```

После успешной установки пакета он будет доступен для импорта с именем dnn_quant.

5.2 Интерфейс командной строки

5.2.1 Команда quantize_graph

Команда quantize_graph выполняет квантование исходного графа в соответствии с калибровочными данными. Квантование графа происходит от заданных входных узлов к заданным выходным узлам. Если форма входных узлов не указана, то можно вручную указать их как параметр input_shape.

Данная функция создает:

1) **Events** – Каталог, содержащий события TensorBoard исходных и квантованных графов;
2) **quant_network_graph.pb** – Квантованный граф;
3) **to_compile** – Каталог, содержащий файлы, необходимые для компилятора TPU (conf.tpu.json, Quant_constants.pickle);
4) **features** – Каталог с файлами feature_#.npy и файлом сопоставления.

Примеры:

quantize_graph original_graph.pb calibration_tensor.npy

–input_nodes=input_1 –output_nodes=output_1,output_2,output_3 –save_dir=quantized_data

```
quantize_graph [OPTIONS] ORIGINAL_GRAPH_PATH
CALIBRATION_TENSORS_FILES
```

Опции:

--input_nodes <input_nodes>

Обязательный аргумент, имена входных узлов исходного графа, разделенные символом "," без пробела.

--output_nodes <output_nodes>

Обязательный аргумент, имена выходных узлов исходного графа, разделенные символом "," без пробела.

--save_dir <save_dir>

Обязательный аргумент, путь к каталогу сохранения для результата квантования.

--input_shape <input_shape>

Формы входных узлов исходного графа, разделенные символом «,» без пробела. Необходимо использовать эту опцию, если заполнитель не имеет фиксированной формы.

-cbs, --calibration_batch_size <calibration_batch_size>

Размер партии для процесса калибровки. Он должен быть меньше или равен размеру пакету инференса и значения пакета форма калибровочного тензора.

-oct, --optimize_calibration_tensors

Установите значение True, чтобы включить оптимизацию калибровочных тензоров (экспериментальная функция).

-fbs, --features_batch_size <features_batch_size>

Создает файлы numpy с аргументом features_batch_size на основе калибровочных тензоров.

Аргументы:

ORIGINAL_GRAPH_PATH

Обязательный аргумент.

CALIBRATION_TENSORS_FILES

Обязательный аргумент.

5.3 Python API

5.3.1 dnn_quant.api.auto_quantization.py

«Обертки» функциональностей низкоуровневого квантования.

Бэкэнд оболочки квантования.

calibrate(quant_cfg, regular_model_weights, calibration_dict, calibration_batch_size=1)

Вычисляет и выполняет пороги квантования для сетевых уровней.

Параметры:

quant_cfg (Dict[str, Any]) – Данные конфигурации с параметрами квантования.

regular_model_weights (Dict[str, Any]) – Веса регулярного графа, которые хранятся в словаре.

calibration_dict (Dict[str, ndarray]) – Словарь калибровочных данных.

calibration_batch_size (int) – Размер партии для процесса калибровки.

Тип возвращаемого значения Dict[str, Dict[str, Any]]

parse_original_graph(graph_path, save_dir, input_nodes, output_nodes)

Анализирует исходный граф, обрабатывает его, чтобы получить конфигурацию и веса обычной модели.

Параметры:

graph_path (str) – Каталог с оригинальным графом;

save_dir (str) – Путь к каталогу сохранения результата анализа;

input_nodes (List[Union[str, Dict[str, Any]]]) – Список спецификаций входных узлов, которые будут превращены в заполнители. Каждая спецификация узла может быть строкой (именем узла) или словарем:

```
{
'node_name': string node name,
'node_shape': user-specified shape (interpretable by tensor-
flow),
'force_reshape': boolean whether to force shape setting,
'node_anchor': string identifier for the node (will be used
as a␣
↪reference
to original node after parsing), Optional.
}
```

Эти узлы должны быть представлены в графе.

output_nodes (List[Union[str, Dict[str, Any]]])

Может быть строкой, представляющей имя узла графа или словарь:

```
{ ‘node_name’: str, ‘node_anchor’: str,
}
```

Тип возвращаемого значения Tuple[Dict[str, Any], Dict[str, Any]]

regular_to_quant_config(regular_model_config)

Произведение данных конфигурации с параметрами квантования.

Parameters regular_model_config (Dict[str, Any]) – Данные конфигурации регулярного графа, которые хранятся в словаре.

Тип возвращаемого значения Dict[str, Any]

5.3.2 dnn_quant.api.load_store_data.py

Функция загрузки и хранения данных.

Модуль загрузки и сохранения данных.

export_tpu_data(software_network, save_dir)

Экспорт конфигурации модели с параметрами и данными модели, необходимыми для компилятора TPU.

Параметры:

• **software_network (SoftwareNetwork) –** Программно-квантованная сеть;

• **save_dir (str) –** Путь к каталогу для сохранения результата квантования.

Тип возвращаемого значения Tuple[Dict[str, Any], Dict[str, Any]]

5.3.3 dnn_quant.api.optimize_calibrations.py

Оптимизация калибровочных тензоров.

Статистическая оптимизация калибровочного тензора.

improve_calibration_tensor(tensor)

Статистическая оптимизация калибровочного тензора. Может снизить потерю производительности после квантования.

Parameters tensor (ndarray) – Калибровочные тензоры.

Тип возвращаемого значения ndarray

5.3.4 dnn_quant.network.py

Классы для построения квантованных графов с использованием файлов конфигурации.

class SoftwareNetwork(cfg_or_cfg_path, weights, network_thresholds_data, add_training_options=False)

Bases: dnn_quant.network.regular_network.RegularNetwork

Программно-квантованная сеть. Используется мнимое квантование в расчетах.

Параметры:

cfg_or_cfg_path (Union[str, Path, dict]) – Квантованный конфигурационный словарь или путь к квантованному файл конфигурации;

weights (Dict[str, Dict[str, ndarray]]) – Веса сети;

network_thresholds_data (Dict[str, Any]) – словарь, данные порогов, используемые для построения NetworkThresholdsManager

add_training_options (bool) – Добавьте параметры обучения «все» для данных конфигурации слоев, отображающих, что пороги квантования должны быть обучаемыми.

static create_configuration_for_training(configuration)

Установка параметры для метаузлов, чтобы разрешить обучение всего, что запрашивает метаузел.

Parameters configuration (Union[str, Path, dict]) – Квантованная конфигурация сети.

Тип возвращаемого значения Dict[str, Dict[str, Any]]

Возвращает конфигурацию сети с вариантами обучения, позволяющими обучать все.

get_configuration_and_data_for_tpu()

Собирает параметры модели, необходимые для запуска квантованной модели на TPU.

Тип возвращаемого значения Tuple[Dict[str, Any], Dict[str, Any]]

Возвращает конфигурацию модели с параметрами, необходимыми для TPU и данных модели.

property trainable: bool

Флаг, отображающий содержит ли модель обучаемые переменные.

property variables_checkpoint_path: Optional[pathlib.Path]

Путь к контрольной точке, откуда модель берет данные для инициализации.

Тип возвращаемого значения Optional[Path]

class HardwareNetwork(cfg_or_cfg_path, weights)

Базовые сущности: dnn_quant.network.regular_network.RegularNetwork

Аппаратная квантованная сеть. Использование собственное квантование в TPU.

Параметры:

cfg_or_cfg_path (Union[str, Path, dict]) – Квантованный конфигурационнный словарь или путь к квантованному файлу конфигурации.

weights (Dict[str, Dict[str, ndarray]]) – Веса сети.

5.4 Пример Python API

5.4.1 Пошаговое квантование нейронной сети

Импорт необходимых библиотек и функций:

```
import numpy as np
from dnn_quant.network import SoftwareNetwork
from dnn_quant.network import HardwareNetwork
from dnn_quant.api.auto_quantization import parse_origi-
nal_graph
from dnn_quant.api.auto_quantization import regu-
lar_to_quant_config
from dnn_quant.api.auto_quantization import calibrate
from dnn_quant.api.load_store_data import export_tpu_data
graph_path = 'LENET5_original_frozen.pb'
calibration_tensor_path = 'calibration_tensors_mnist.npy'
input_nodes = ['input_input']
output_nodes = ['output/Softmax']
save_dir = 'quantized_data'
```

Чтобы исходный граф можно было интерпретировать с помощью модуля dnn_quant, он обрабатывается и транслируется в промежуточное представление. В результате синтаксического анализа генерируются два словаря, конфигурация и константы. Конфигурация описывает архитектуру обычного

графа. Константы содержат веса и смещения регулярного графа. С помощью полученных словарей строится конфигурация квантованного графа.

```
regular_model_config, regular_model_weights =
parse_original_graph(
graph_path,
save_dir,
input_nodes=input_nodes,
output_nodes=output_names
)
quant_cfg = regular_to_quant_config(regular_model_config)
```

Запуск процесса калибровки. На основе калибровочного тензора выбираются пороговые значения для квантования.

Примечание. Каждый входной узел имеет свой собственный калибровочный тензор с таким же типом данных, как входной узел dtype.

```
calibaration_tensor = np.load(calibration_tensor_path)
calibration_dict = {
input_nodes[0]: calibaration_tensor
}
network_thresholds = calibrate(quant_cfg,
regular_model_weights, calibration_dict)
```

После построения квантованной конфигурации и выбора пороговых значений для квантования создаются модель квантованной сети (SoftwareNetwork) и данные для компиляции.

```
software_network = SoftwareNetwork(
cfg_or_cfg_path=quant_cfg,
weights=regular_model_weights,
network_thresholds_data=network_thresholds
)
data_for_tpu_dict, cfg_for_fpga = export_tpu_data(
software_network,
save_dir
)
```

Построение квантованного графа и (при желании) его сохранение.

```
hardware_network = HardwareNetwork(
cfg_or_cfg_path=cfg_for_fpga,
weights=data_for_tpu_dict,
)
hardware_network.save_graph(os.path.join(save_dir,
"quant_network_graph.pb"))
```

Примечание. SoftwareNetwork и HardwareNetwork чрезвычайно близки по составу. Их принципиальное отличие состоит в том, что HardwareNetwork не содержит промежуточных операций квантования. В связи с этим модель становится как можно ближе к инференсу TPU.

Ссылки на файлы:

https://drive.google.com/file/d/1do9_gvOHR15qrCqTN80SxAba-2mmg1Lq/view – Оригинальный граф TensorFlow;

https://drive.google.com/file/d/1qYHNFOIaEr_fUKXAR-Gepp73GIsWyfCx/view – Калибровочный тензор.

6. TPU FRONTEND

tpu_compiler_frontend обеспечивает компиляцию CLI и функционал API для преобразования вывода утилиты квантования в Программу TLM / TPU.

Вывод утилиты квантования состоит из двух файлов: *JSON-файл с описанием сетевой модели (или топологии, или структуры), так называемый conf.tpu.json – файл Python pickle с квантованными константами (весами, смещениями и другими массиво-подобными коэффициентами) и так называемый Quant_constants.pickle.

Программа TPU состоит из одного файла, обычно с расширением .tpu.

Пользователь может получить программу TPU с помощью инструмента командной строки tcf (см. TPU_COMPILER_FRONTEND.CLI.CLI_COMPILE_ ()) или серией Python-вызовов.

Внутри Python должна быть вызвана функция tpu_compiler_frontend.compile_ () и необязательное ключевое слово output_tpu_path (полный путь к сериализации .tpu файла), необходимая для выполнения компиляции и сериализации за один вызов.

Для tpu_compiler.compiler.CompileParameters доступны различные параметры компиляции, которые приводят к созданию программы с тем же выходом, но, возможно, с лучшей производительностью. В настоящее время считается, что точная настройка этих параметров предназначена только для внутреннего использования. Пользователю предлагается выбрать одну из предустановок из: * DEFAULT - разумное значение по умолчанию, она была наиболее тщательно проверена; * STAGING - экстремальный режим для лучшей производительности; * DISABLED - возврат в режим для отключения почти всех оптимизаций для стабильности.

Альтернативный подход включает получение промежуточного объекта отладки. Этот объект используется для отладки программы, выполнения глубокого анализа и запуска программы в смоделированной среде программной модели TPU TLM. Объект отладки вызывается TLMProgram и состоит из tpu_tlm_is.base.Executable и словаря tpu_tlm_is.base.TensorDescriptions. Программа TLM получается путем вызова tpu_compiler_frontend.compile_ (). Инференс программы TLM может выполняться через функцию tpu_compiler_frontend.tlm (). В свою очередь программу TLM может быть сериализована в файл .tpu с помощью tpu_compiler_frontend.dump_to_tpu_program ().

6.1 API

Использование общедоступного API лучше всего можно представить в следующем тесте:

```
1 import os
2 import tempfile
3
4 import numpy as np
5
6 from tpu_compiler.compiler import DISABLED
7 from tpu_compiler_frontend import compile_
8 from tpu_compiler_frontend import dump_to_tpu_program
9 from tpu_compiler_frontend import tlm
from tpu_tlm_is import get_hw_params
11
12 PATH = '/auto/tests/quantized_data/master'
13
14
15 def test_complex_web_api():
16 # Compile into TLMProgram instance
17 tlm_program = compile_(
18 model_path=os.path.join(PATH, 'lenet5', 'to_compile',
'conf.tpu.json'),
19 quant_const_path=os.path.join(PATH, 'lenet5', 'to_com-
pile', 'quant_constants.
↪pickle'),
20 hardware_parameters=get_hw_params('fpga_64x64'),
21 batch=1,
22 parameters=DISABLED,
23 )
24
25 # Generate dummy input
26 input_dtype = tlm_program[1]['input_input'].user_dtype
27 dummy_input_data = {
28 'input_input': np.random.randint(-127, 127, size=1 * 28 *
28 * 1).astype(input_
↪dtype).reshape((1, 28, 28, 1)),
29 }
30
31 # Run on TLM, get output and execution time estimation
32 tlm_estimation_time, tlm_output_data = tlm(tlm_program,
dummy_input_data)
33
34 assert tlm_estimation_time > 0
35 assert 'output/Softmax' in tlm_output_data
36
37 # Dump to TPU program
38 with tempfile.NamedTemporaryFile() as temp_file:
39 dump_to_tpu_program(tlm_program, temp_file.name)
40 assert os.path.getsize(temp_file.name) > 0
```

compile_(model_path, quant_const_path, hardware_parameters, batch, parameters, output_tpu_path=None)

Компиляция программы TLM и (при желании) сохранение ее как программы TPU.

Параметры:

model_path (str) – Путь к файлу модели JSON, описывающий квантованную структуру сети (обычно вызывается conf.tpu.json).

Quant_const_path (str) - Путь к файлу pickle, содержащему квантованные сетевые константы (обычно вызывается Quant_constants.pickle).

hardware_parameters (HardwareParameters) - Экземпляр HardwareParameters, описывающий аппаратных параметров целевой платформы (TPU).

batch (int) - перезапись пакета (экземпляр Сети может допускать явное описание пакета).

parameters (CompileParameters) - различные параметры компиляции. Включение оптимизации обычно приводит к более быстрой программе, но увеличивает время компиляции. Может привести к исключению для произвольной сети.

output_tpu_path (Optional[str]) - Если указано, то программа TLM будет преобразована в программу TPU и сохранится по этому пути. Преобразование выполняется путем вызова dump_to_tpu_program ().

Тип возвращаемого значения Tuple[Executable, Dict[str, TensorDescription]]

Возвращает Tuple of Executable и словарь TensorDescription с ключами, соответствующими именам входных или выходных слоев.

Примечание. Обратите внимание на следующее правило, которое применяется к необязательному аргументу пакета.

Таблица 3 – Описание правила, которое применяется к необязательному аргументу пакета

Аргумент пакета compile_ function	Значение пакета из conf.tpu.json	Результат
Любая константа X	Нет или 1	X
Любая константа X	Любая константа Y!=X	Исключение
Нет	Нет	1
Нет	Любая константа Y	Y

dump_to_tpu_program(tlm_program, output_path)

Преобразование программы TLM в программу TPU и сохранение ее в файл.

Параметры:

tlm_program (Tuple[Executable, Dict[str, TensorDescription]]) – Исполняемый файл и словарь тензорных описаний TensorDescription.

output_path (str) – Путь к выходному файлу (обычно заканчивается на .tpu)

Тип возвращаемого значения: Нет.

tlm(tlm_program, input_data)

Инференс TLM модели.

Внимание. Эта функция предназначена только для внутреннего использования и требует установки дополнительных зависимостей.

Внимание. Этот метод будет переименован в TPU-4151.

Параметры:

tlm_program (Tuple [Executable, Dict [str, TensorDescription]]) - программа TLM в виде исполняемого файла Executable и словаря тензорных описаний TensorDescription;

input_data (Dict [str, ndarray]) - словарь входных данных с ключами, соответствующими именам входных слоев.

Тип возвращаемого значения Tuple [Optional [float], Dict [str, ndarray]]

Возвращает набор оценок времени инференса TLM и выходной словарь с ключами, соответствующими названиям выходных слоев.

6.2 CLI

6.2.1 tcf

TPU Compiler Frontend – приложение CLI для компиляции программ TPU и TLM из выходных файлов утилиты квантования.

MODEL_PATH – это путь к файлу модели JSON, описывающему топологию сети. Обычно вызывается conf.tpu.json.

QUANT_CONST_PATH - это путь к зафиксированным константам сети (весам, смещениям и другим квантованным константам). Обычно вызывается Quant_constants.pickle.

ПРИМЕР: tcf -v debug –dump-tlm-program-to = ”lenet5.tlm” –dump-tpu-program-to = ”lenet5.tpu” “conf.tpu.json” «Quant_constants.pickle» fpga_64x64 Обратите внимание, что параметры дампа являются необязательными и по умолчанию выходной файл не будет производиться.

```
tcf [OPTIONS] MODEL_PATH QUANT_CONST_PATH
{fpga_64x64|fpga_32x32_kcu|fpga_16x1
6_zynq|mobile|server|server_silicon|v2p1|experimental|mobile
_fpga|tiny_tpu
|130nm}
```

Параметры:

--dump-tlm-program-to <dump_tlm_program_to>

Сериализация программного объекта TLM с помощью pickle. Обратите внимание, что эта функция предназначена только для внутреннего использования и требует установки необязательных зависимостей.

--dump-tpu-program-to <dump_tpu_program_to>

Сериализация программы TPU

--dump-json-program-to <dump_json_program_to>

Сериализация программного объекта TLM через json. Обратите внимание, что эта функция предназначена только для внутреннего использования и требует установки необязательных зависимостей.

--batch <batch>

Пакет входных тензоров.

--parameters <parameters>

Предварительная установка параметров компиляции (обратите внимание, что разные значения этой опции могут привести к увеличению времени компиляции и может привести к выходу из строя для больших сетей)

Параметры disabled | default

-v, --log-level <log_level>

Уровень ведения журнала

Параметры critical | error | warning | info | debug | notset

Аргументы

MODEL_PATH

Обязательный аргумент

QUANT_CONST_PATH

Обязательный аргумент

PLATFORM

Обязательный аргумент

7. ФУНКЦИОНАЛЬНЫЕ ИНСТРУКЦИИ

7.1 Заморозка исходного графа нейронной сети

Заморозка исходного графа нейронной сети является отправной точкой при подготовке нейронной сети к выводу на IVA TPU. На этом этапе все переменные преобразуются в константы, промежуточные операции, специфичные для обучения (optimizers, assertions, prints и т. д.) и другие операции, не связанные с выполнением модели, удаляются с графа.

В библиотеке iva_applications, которая является частью tpu_framework, подготовлены функции для автоматизации замораживания исходного графа.

```
[1]: from iva_applications.utils import freeze_session
from iva_applications.utils import freeze_keras_model
2021-08-26 14:29:22.944864: W tensorflow/stream_execu-
tor/platform/default/dso_loader.cc:
↪60] Could not load dynamic library 'libcudart.so.11.0';
dlerror: libcudart.so.11.0: ␣
↪cannot open shared object file: No such file or directory
2021-08-26 14:29:22.944889: I tensorflow/stream_execu-
tor/cuda/cudart_stub.cc:29] Ignore␣
↪above cudart dlerror if you do not have a GPU set up on
your machine.
/home/azolotarev/.virtualenvs/tpu_framework_docs/lib/py-
thon3.8/site-packages/pandas/
↪compat/__init__.py:124: UserWarning: Could not import the
lzma module. Your installed␣
↪Python is incomplete. Attempting to use lzma compression
will result in a RuntimeError.
warnings.warn(msg)
Warning: No module named 'iva_tpu'
```

Посмотрите на этот процесс. Например ResNet50 из [tf.keras.applications]

```
[2]: import tensorflow as tf
```

Определение предварительно обученной нейронной сети (TensorFlow). Любой tf.Graph и tf.keras.Model, отвечающий требованиям исходного графа, подходит в качестве предварительно обученной нейронной сети.

```
[3]: resnet50 = tf.keras.applications.ResNet50()
2021-08-26 14:29:24.203770: I tensorflow/com-
piler/jit/xla_cpu_device.cc:41] Not creating␣
↪XLA devices, tf_xla_enable_xla_devices not set
2021-08-26 14:29:24.204357: W tensorflow/stream_execu-
tor/platform/default/dso_loader.cc:
↪60] Could not load dynamic library 'libcuda.so.1'; dler-
ror: libcuda.so.1: cannot open␣
↪shared object file: No such file or directory
2021-08-26 14:29:24.204369: W tensorflow/stream_execu-
tor/cuda/cuda_driver.cc:326] failed␣
↪call to cuInit: UNKNOWN ERROR (303)
2021-08-26 14:29:24.204383: I tensorflow/stream_execu-
tor/cuda/cuda_diagnostics.cc:156]␣
↪kernel driver does not appear to be running on this host
(MSK-NB-0317): /proc/driver/
↪nvidia/version does not exist
2021-08-26 14:29:24.204516: I tensorflow/core/plat-
form/cpu_feature_guard.cc:142] This␣
↪TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use␣
↪the following CPU instructions in performance-critical op-
erations: AVX2 FMA
To enable them in other operations, rebuild TensorFlow with
the appropriate compiler␣
↪flags.
2021-08-26 14:29:24.204926: I tensorflow/com-
piler/jit/xla_gpu_device.cc:99] Not creating␣
↪XLA devices, tf_xla_enable_xla_devices not set
```

Зафиксируем граф с помощью функции freeze_keras_model из iva_applications. Подробнее о реализации других функций заморзки см. документацию iva_applications. В результате функции замороженный граф появится в каталоге save_dir в виде файла с расширением .pb

```
[4]: freeze_keras_model(resnet50, save_dir='save_dir')
2021-08-26 14:29:25.750298: I tensorflow/core/grappler/de-
vices.cc:69] Number of eligible␣
↪GPUs (core count >= 8, compute capability >= 0.0): 0
2021-08-26 14:29:25.750420: I tensorflow/core/grappler/clus-
ters/single_machine.cc:356]␣
↪Starting new session
2021-08-26 14:29:25.750614: I tensorflow/com-
piler/jit/xla_gpu_device.cc:99] Not creating␣
↪XLA devices, tf_xla_enable_xla_devices not set
2021-08-26 14:29:25.767937: I tensorflow/core/platform/pro-
file_utils/cpu_utils.cc:112]␣
↪CPU Frequency: 2699905000 Hz
2021-08-26 14:29:25.785590: I tensorflow/core/grappler/opti-
mizers/meta_optimizer.cc:928]␣
↪Optimization results for grappler item: graph_to_optimize
function_optimizer: function_optimizer did nothing. time =
0.225ms.
function_optimizer: function_optimizer did nothing. time =
0.001ms.
```

7.2 Выбор входных и выходных узлов исходного графа

При запуске квантования исходного графа требуется явно установить входные и выходные узлы. Выбор узлов выполняется пользователем вручную. Следует соблюдать общее правило: «Узлы ввода и вывода во время квантования — это входные и выходные имена узлов архитектуры нейросети (обучающая часть) без специфической обработки, такой как предварительная обработка нейронной сети и постобработка нейронной сети.

Совет: Выбор имен для запуска квантования необходимо проводить таким образом, чтобы выбранный подграф принадлежал исключительно архитектурной части нейронной сети и не имел неподдерживаемых операций. Если исходный граф полностью поддерживается и в нем нет операций, не связанных с конкретной обработкой данных, то подграф представляет собой полный граф.

Может возникнуть ситуация, когда исходный граф содержит неподдерживаемые операции. В этом случае необходимо определить, какие узлы графа не могут быть обработаны, и выбрать их для предварительной обработки нейронной сети, постобработки нейронной сети или квантования по частям.

Рассмотрим примеры выбора входных и выходных узлов исходного графа.

7.2.1 Пример №1

Существует нейронная сеть, состоящая из сверточных и полносвязных слоев.

Сравнив операции на графе и поддерживаемых операций, можно заметить, что весь граф может быть обработанным, так как все операции в графе поддерживаются. Поэтому входной узел для квантования называется «Input_1», а выходной узел называется «Identity».

7.2.1 Пример №2

Есть нейросеть, как в примере 1, но с дополнительным слоем tf.keras.layers.Cropping2D.

Мы рассуждаем так: операция Cropping относится к предварительной обработке нейронной сети, эта операция не нуждается в ускорении, она стоит в начале сети, поэтому лучше удалить её из исходного графа. После удаления определяем узлы ввода и вывода, как в примере 1.

Иногда редактирование исходного графа может быть «неудобным» в данный момент, тогда вы можете попробовать квантовать сеть как есть, указав входной узел как input_1, а выходной узел как Identity. В некоторых случаях предварительная обработка все равно будет выполнена и квантование будет

успешно завершено. Однако, несмотря на такую гибкость для пользователя, все же лучше исключить узел и указать на входе «sequential/crop/strided_slice» и на выходе «Identity». В то же время, стоит помнить, что исключенную операцию нужно передать на предварительную обработку нейронной сети.

7.3 Подготовка калибровочных тензоров

Для выполнения квантования необходимо создать калибровочный тензор. В случае, когда имеется несколько входных узлов на графе необходимо создать калибровочный тензор для каждого. Подготовка калибровочного тензора практически ничем не отличается от предварительной обработки данных нейронной сети для прямого прохода нейронной сети, за исключением того, что обработанные данные необходимо сохранить в numpy-файл.

Для некоторых нейронных сетей с открытой архитектурой в модуле iva_applications есть вспомогательные функции, такие как предварительная обработка, постобработка, сохранение калибровочных тензоров и др.

Рассмотрим пример подготовки калибровочного тензора на примере сети ResNet50.

Она тренируется на наборе данных ImageNet.

Предварительная обработка: обрезка 224 × 224 случайным образом выбирается из изображения или его горизонтального отражения со средним значением для каждого вычтенного пикселя.

Визуализация событий tensorboard для графа показывает, что необходимо указать входной узел для квантования input_1. Это означает, что для предварительной обработки не требуется дополнительных операций для подготовки калибровочного тензора из графа.

Таким образом, код для подготовки тензора калибровки будет выглядеть так:

7.3.1 Пошаговая подготовка калибровочного тензора ResNet50

```
import os
import glob
import numpy as np
from PIL import Image
from iva_applications.imagenet.images_utils import
crop_and_resize
```

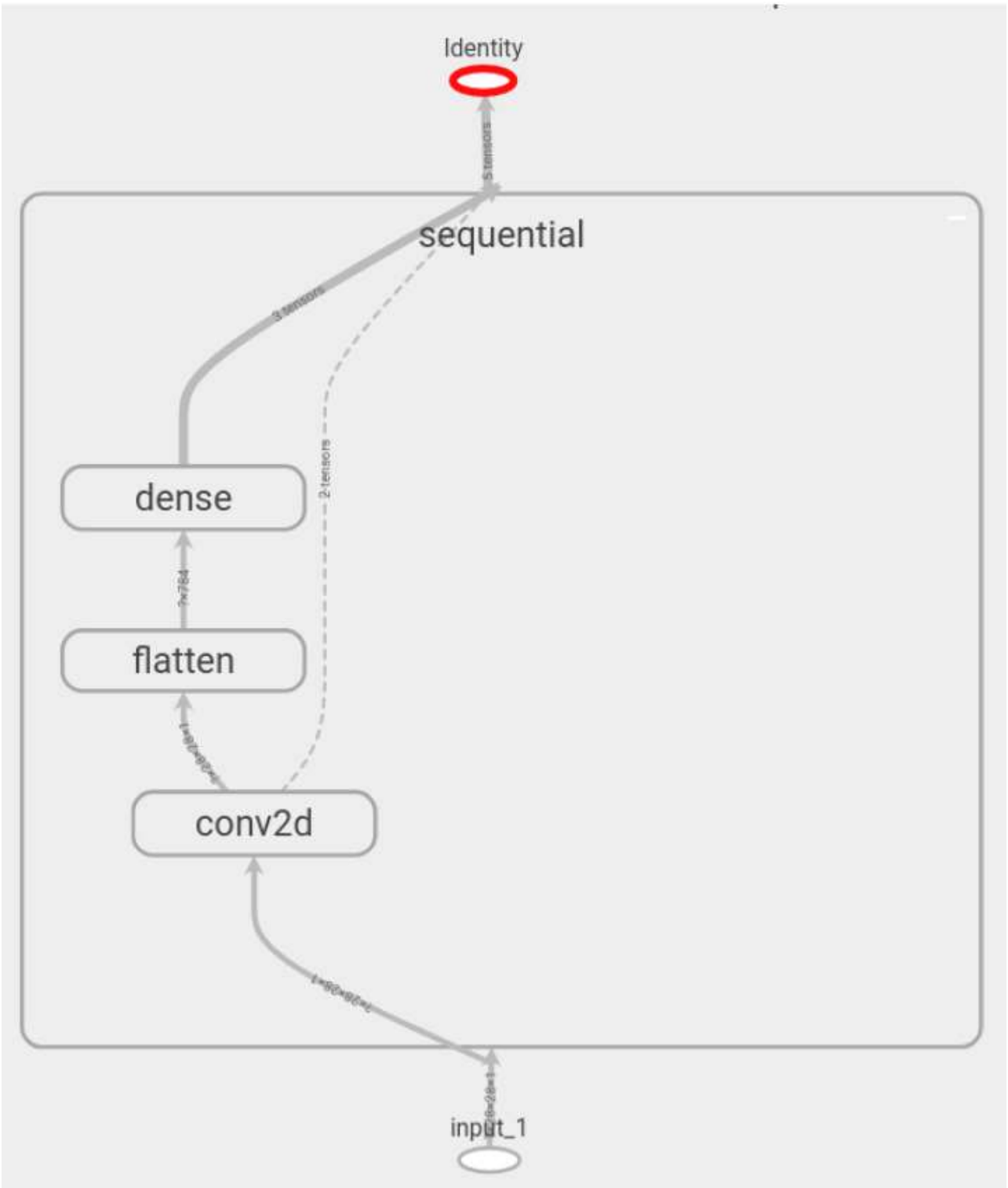

Рисунок 1 – Визуализация графа. Часть 1

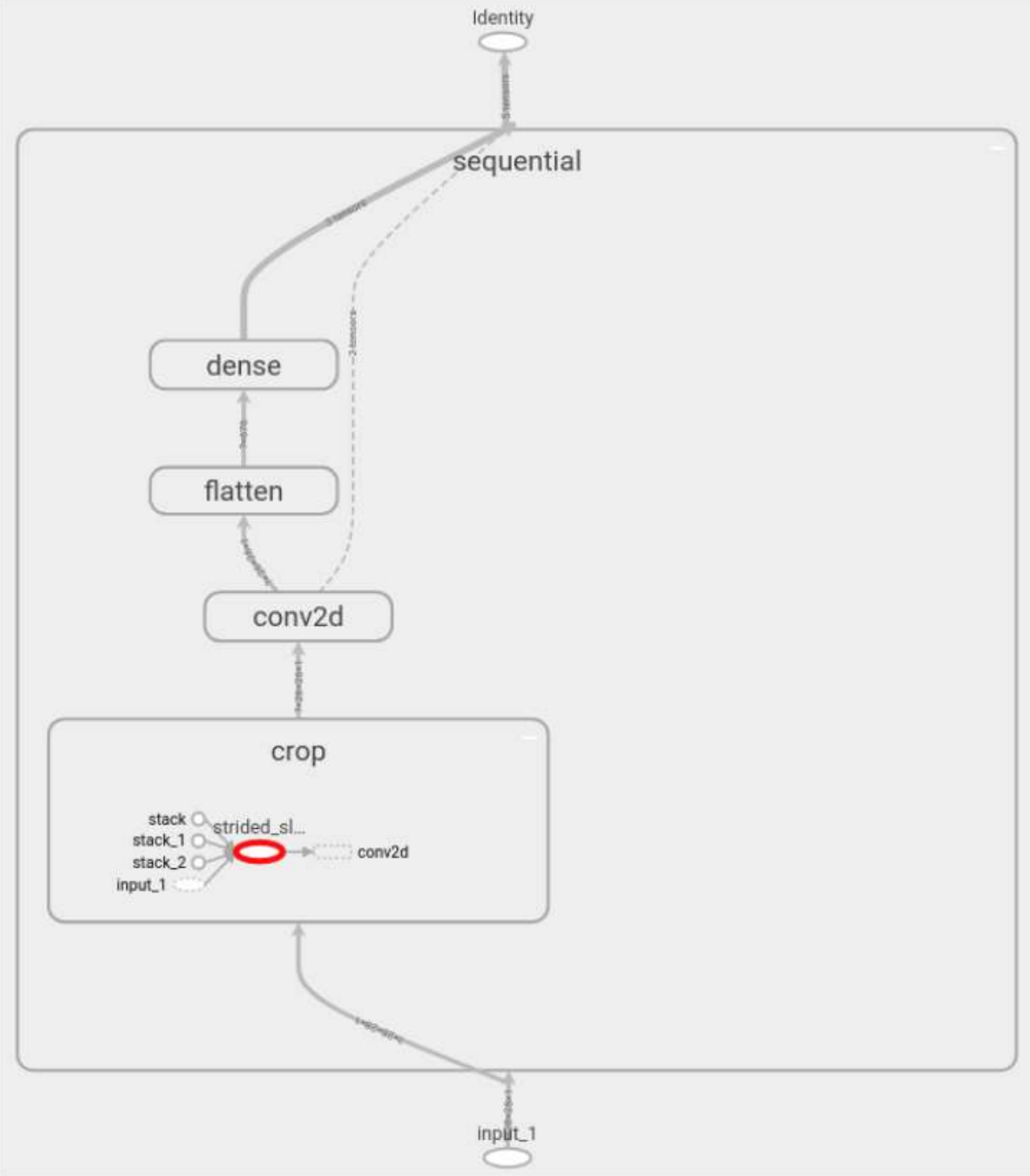

Рисунок 2 – Визуализация графа. Часть 2

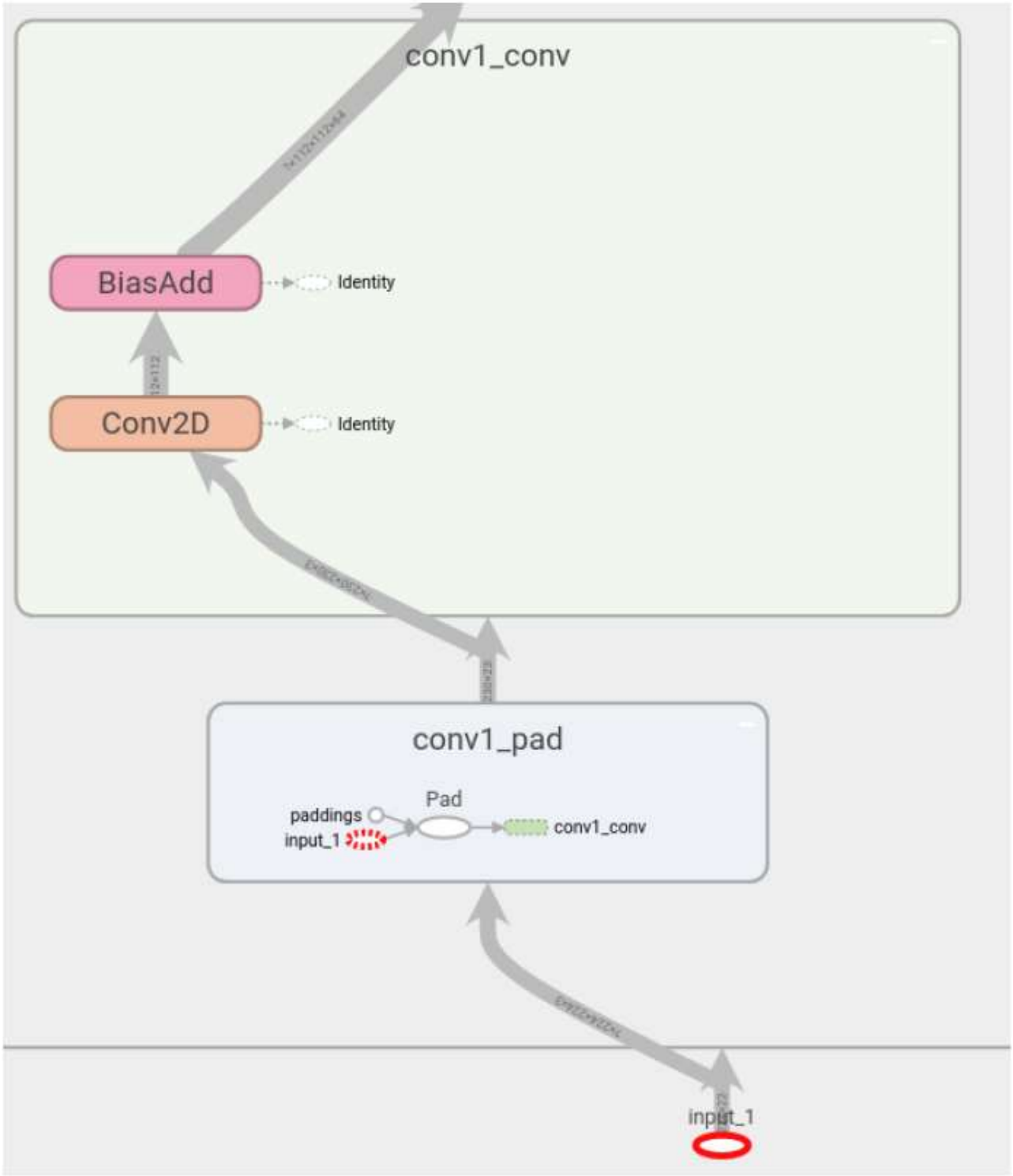

Рисунок 3 – Визуализация графа. Часть 3

```
def image_to_tensor(image: Image) -> np.ndarray:
"""
Preprocess an input pillow image and convert it to numpy ar-
ray for vanilla ResNet50
Parameters
----------
image
An input image to be processed.
Returns
-------
tensor
Preprocessed numpy array.
"""
mean = [123.68, 116.779, 103.939]
image = image.convert('RGB')
tensor = np.asarray(image)
tensor = crop_and_resize(tensor, 224)
tensor = tensor - np.array(mean)
tensor = tensor.astype(np.float32)
assert tensor.shape == (224, 224, 3)
return tensor
def save_calibration_tensor(path_to_dataset: str, save_dir:
str):
"""
Save JPEG images from dataset directory to calibration_ten-
sors numpy file.
Parameters
----------
path_to_dataset
Path to dataset directory.
save_dir
Path to saving directory.
-------
"""
tensors = []
images = glob.glob(os.path.join(path_to_dataset, "*.JPEG"))
for image in images:
image = Image.open(image)
tensor = image_to_tensor(image)
tensors.append(tensor)
tensors = np.array(tensors, dtype='float32')
np.save(os.path.join(save_dir,
'calibration_tensors_resnet'), tensors)
```

8.3.2 Подготовка калибровочного тензора ResNet50 с помощью iva_applications

```
from iva_applications.resnet50.calibration import save_cali-
bration_tensor
# path_to_dataset - path to ImageNet training dataset images
# save_dir - path to the directory where the .npy file with
calibration tensors will be␣
↪saved
save_calibration_tensor(path_to_dataset, save_dir)
```

8.4 Квантование по частям

Квантование по частям используется как метод обхода неподдерживаемых операций. Суть его в том, чтобы сгенерировать несколько программ IVA TPU для той же нейронной сети.

Например, есть граф нейронной сети. Допустим, операция sequential/crop/strided_slice не поддерживается. Чтобы такой граф можно было вывести с помощью IVA TPU, необходимо представить граф как последовательность TPU-> CPU-> TPU.

В этом случае квантование и компиляцию необходимо будет выполнить дважды, а операцию sequential/crop/strided_slice запустить на CPU.

Часть TPU. Первое квантование: input_nodes = input_1, output_nodes = serial / conv2d / BiasAdd, калибровочные тензоры для input_1.

Часть CPU: input_nodes = последовательный / обрезка / strided_slice, output_nodes = sequential/crop/strided_slice

Часть TPU. Второе квантование: input_nodes=sequential/crop/strided_slice, output_nodes=Identity. калибровочные тензоры для sequential/crop/strided_slice.

Более сложные случаи, когда граф содержит несколько неподдерживаемых операций, требуют аналогичных рассуждений.

8.5 Инференс на IVA TPU

Для нейронной сети с прямой связью необходимо инициализировать TPURunner с помощью программы TPU и выполнить инференс. После инициализации TPURunner IVA TPU готов к вычислению нейронной сети. После вызова TPURunner нейронная сеть выполняется на IVA TPU.

Примечание. Для начала инференса необходимо иметь программу для IVA TPU.

```
import os
from PIL import Image
from iva_applications.utils import TPURunner
from iva_applications import imagenet
from iva_applications.resnet50 import image_to_tensor
# read an image
with open(image_path, 'rb') as img_file:
image = Image.open(img_file)
# preprocess data
tensor = image_to_tensor(image)
# TPURunner initialization
runner = TPURunner(program_path)
# get input and output nodes names
input_nodes = runner.input_nodes
output_nodes = runner.output_nodes
```

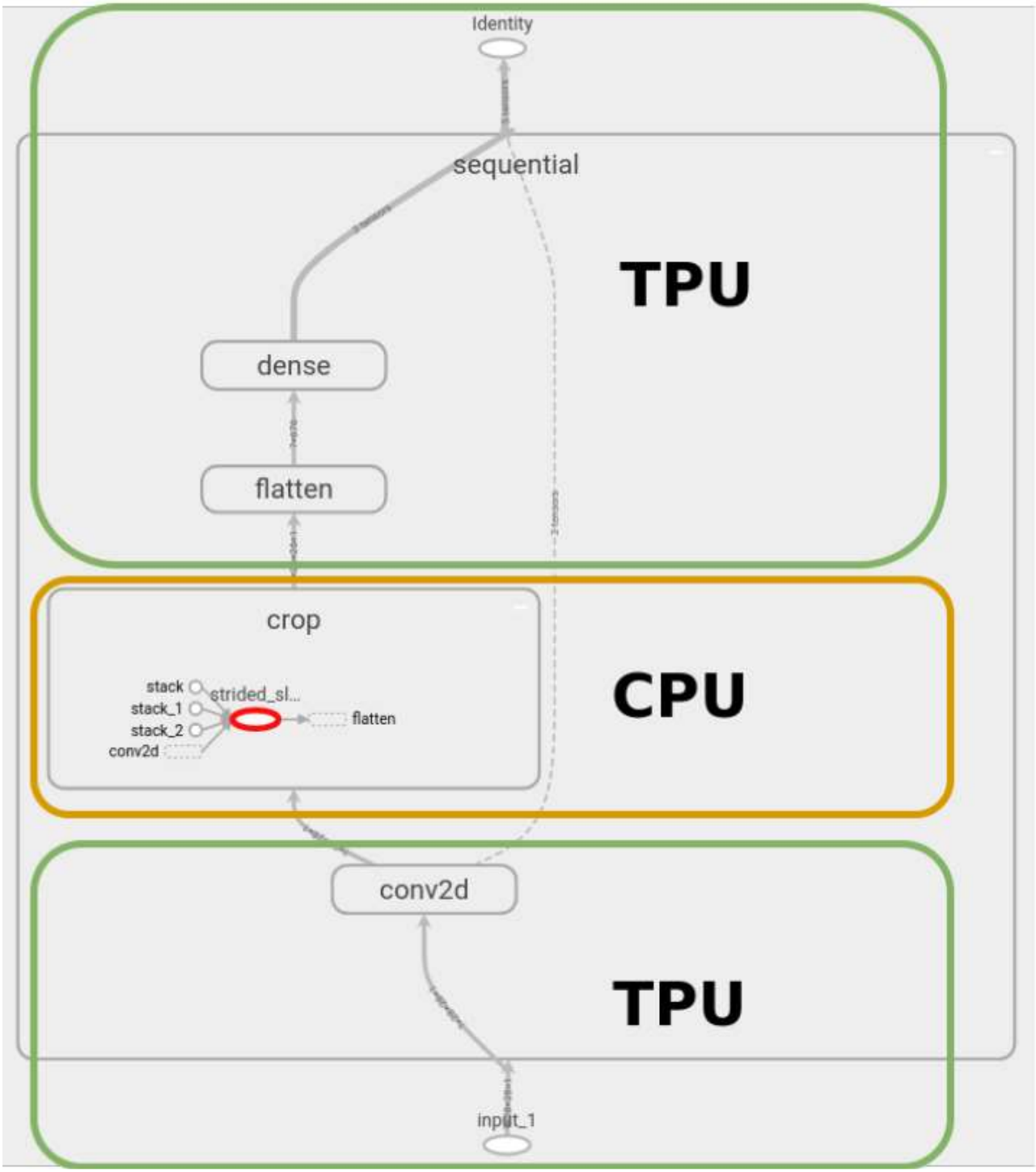

Рисунок 4 – Визуализация последовательности расчетов нейросети на TPU и CPU

ПЕРЕЧЕНЬ ТЕРМИНОВ

Инференс — процесс выполнения обученной модели нейронной сети.

«Заморозить» граф нейронной сети – Удалить из нейросети операции, не влияющие на результат ее выполнения, и преобразовать переменные в константы.

Исходный граф нейронной сети – Нейронная сеть с обученными весами, которую планируется запускать на TPU, замороженная и сохраненная в форматах protobuf или onnx.

Квантование – Последовательное масштабирование и округление данных, необходимое для представления действительных чисел через числа в формате целых чисел.

Конфигурация квантованной сети – Словарь, описывающий топологию квантованного графа нейронной сети, включая слои и их параметры. Он представлен в виде файла tpu.conf.json.

Квантованные сетевые константы – Словарь, в котором хранятся постоянные значения квантованного графа нейронной сети. Представлен в виде файла Quant_constants.pickle.

Квантованный граф нейронной сети – Исходный граф нейронной сети, прошедший этап квантования и сгенерированный с помощью dnn_quant. Обычно вызывается Quant_network_graph.pb.

Калибровочный тензор – Ndarray, содержащий N выборок из обучающего набора данных, предварительно обработанных так же, как и для обучения. Рекомендуемое количество образцов – 100.

Компиляция – Преобразование файла конфигурации и констант квантованной нейронной сети в файл программы IVA TPU.

Парсер графов – Модуль protobuf_evert, предоставляющий набор инструментов для предварительной обработки исходных графов нейронной сети и получения обобщенного описания модели по слоям.

Программный файл IVA TPU – Набор машинных инструкций для выполнения исходного графа нейронной сети на тензорном процессоре. Результат сетевой компиляции.

Сверточный слой — слой искусственной нейронной сети, обладающей специальной архитектурой, направленной на эффективное распознавание изображений.

Слой – Группа операций, которая представляет собой автономную структуру для вычисления высокоуровневых операций, таких как свертка, умножение матриц, объединение, транспонирование и т. д.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

IVA TPU – Семейство тензорных микропроцессоров, разрабатываемых IVA Technologies, которые предназначены для ускоренного расчета нейронных сетей. Архитектура данного вида процессоров оптимизирована для выполнения задач ускоренного расчета нейронных сетей. Основой процессоров является блок матричного умножения, который выполняет наиболее ресурсоемкие вычисления со скоростью десятки тысяч операций за такт. Специализированная архитектура процессора и вычислительных элементов позволяет получить более высокую производительность и энергоэффективность по сравнению с другими решениями, такими как многоядерные цифровые сигнальные процессоры и графические карты.

Семейство процессоров включает в себя:

- IVA TPU – настраиваемый модуль (IP-блок) тензорного ускорителя, пригодный для изготовления в ИС;
- IVA TPU for edge – универсальный процессор с тензорным ускорителем – для встраиваемых систем;
- IVA TPU for cloud – серверный тензорный процессор – для крупных систем обработки больших объемов данных;

- IVA TPU for FPGA – параметризированное тензорное ядро для интеграции в специализированные системы на базе FPGA.

TensorFlow – Открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия.

TPU – Tensor processor unit. Тензорный процессор, относящийся к классу нейронных процессоров, являющийся специализированной интегральной схемой.

Лист регистрации изменений									
Изм.	*Номера листов (страниц)*				*Всего листов (страниц) в документе*	*№ документа*	*Входящий номер сопроводи-тельного документа и дата*	*Подпись*	*Дата*
	изменен-них	заменен-ных	новых	аннули-рованных					