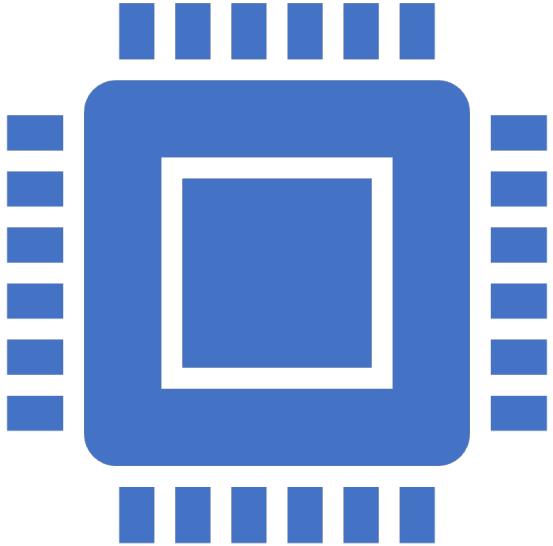
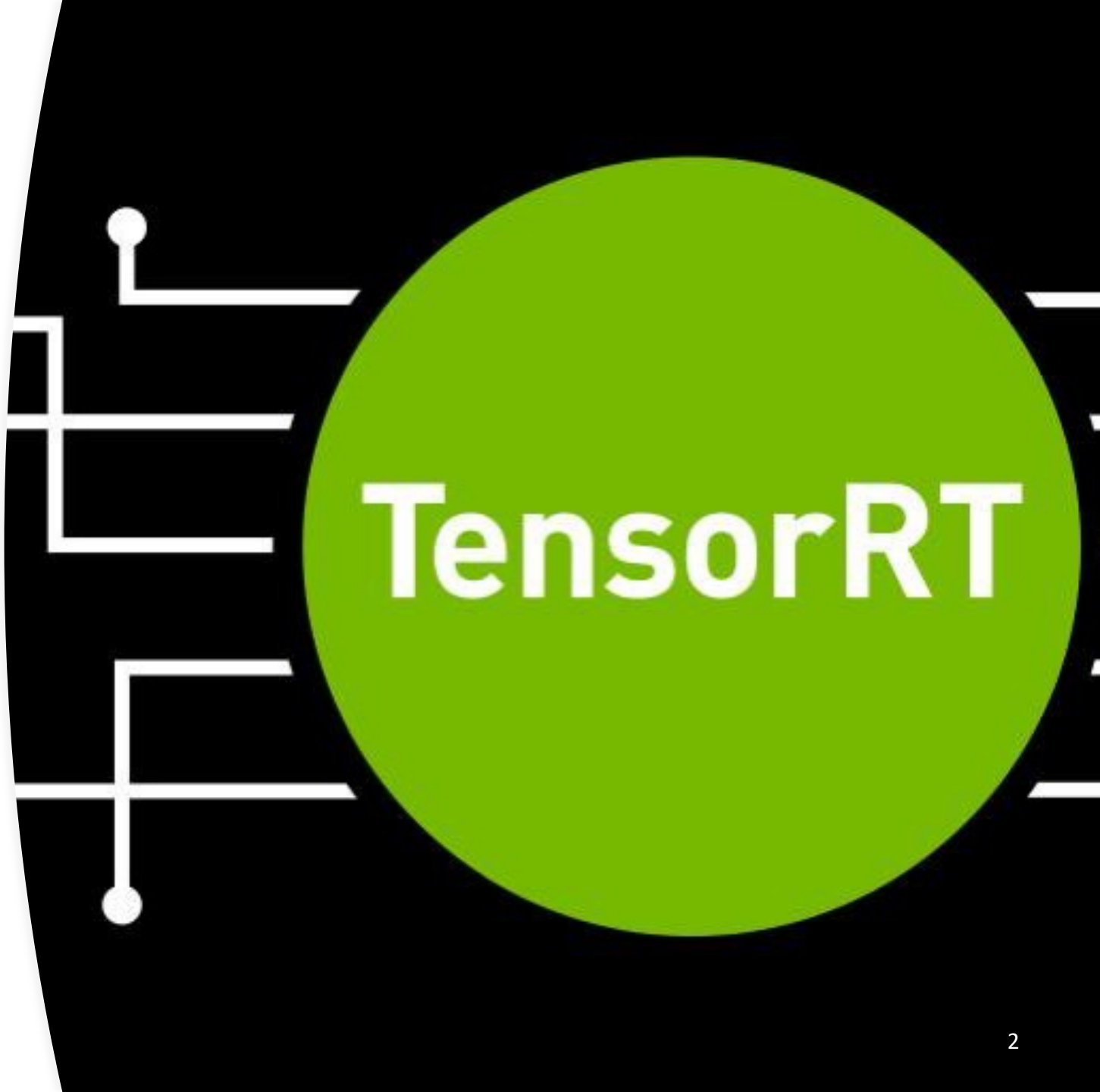


Оптимизация нейронных сетей

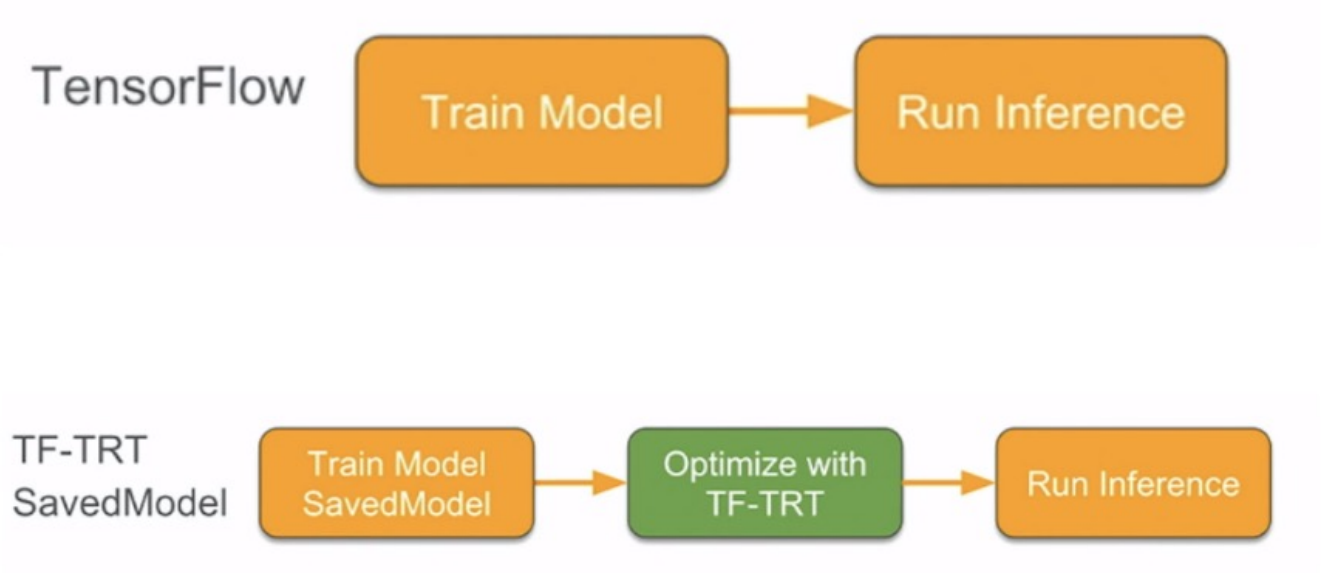
О чем поговорим

- Типы оптимизаций в TensorRT
- Использованием TF-TRT для оптимизации моделей tensorflow
- ONNX
- Как оптимизировать Pytorch



TensoRT

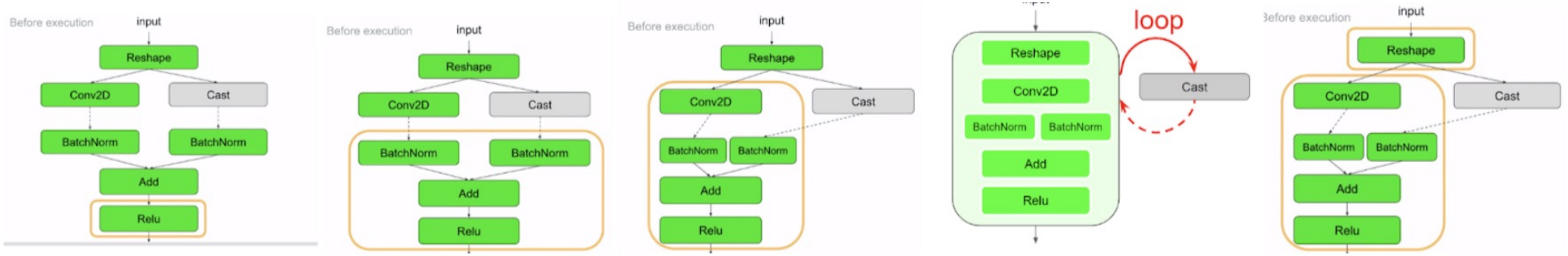
Что такое TensorRT? NVIDIA TensorRT это высокопроизводительный оптимизатор инференса моделей и среда выполнения, которые могут быть использованы для инференса с низкой точностью чисел (FP16, INT8) на GPU.
TensorRT сейчас интегрирован в Tensorflow и его использование заметно упрощается.
Помимо отпитизации также удаляется все лишнее в модели, что не пригодится для инференса.



https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html

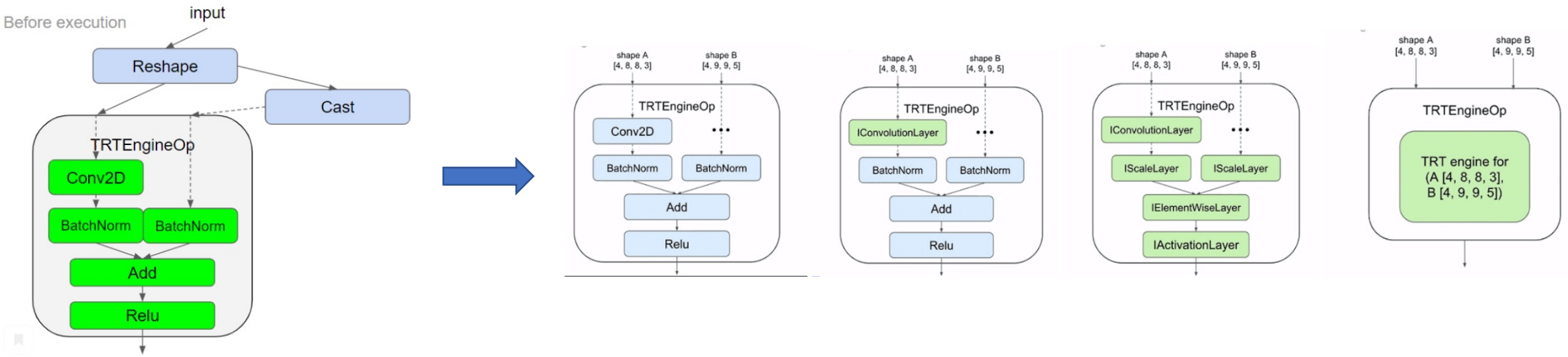
TensoRT оптимизация графа

1) TensoRT сканирует граф сети и определяет те места, которые он может оптимизировать
2) Преобразует те участки графа, которые ему известны (что не знает, оставляет на движке tensorflow)
3) Оптимизированные подграфы заменят собой сходные части графа модели



https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html

TensoRT оптимизация графа



https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html

TensoRT оптимизация графа

Оптимизация в TensorRT – это баланс между скоростью и расходами на создание подграфа и использованием движка TensorRT. Что нужно контролировать?

minimum_segment_size – минимальный размер подграфа для оптимизации. Рекомендуется брать стандартно значение равное 3. Меньше\больше не всегда приводит к улучшению результата, но можно попробовать.
is_dynamic_op – если все размеры входных параметров известны, то ставим False и оптимизация будет идти лучше. Если модель типа BERT, то тут ставим True.
max_batch_size – максимальный размер батча, под который также будет оптимизирована архитектура. Не рекомендуется брать большим.
max_workspace_size_bytes – операторам TF-TRT часто требуется временное рабочее пространство. Параметр ограничивает это пространство и если места будет слишком мало, то может быть не найдено хорошей оптимизации.
precision_mode – параметр устанавливает тип точности. О нем мы поговорим далее.

TensoRT precision mode

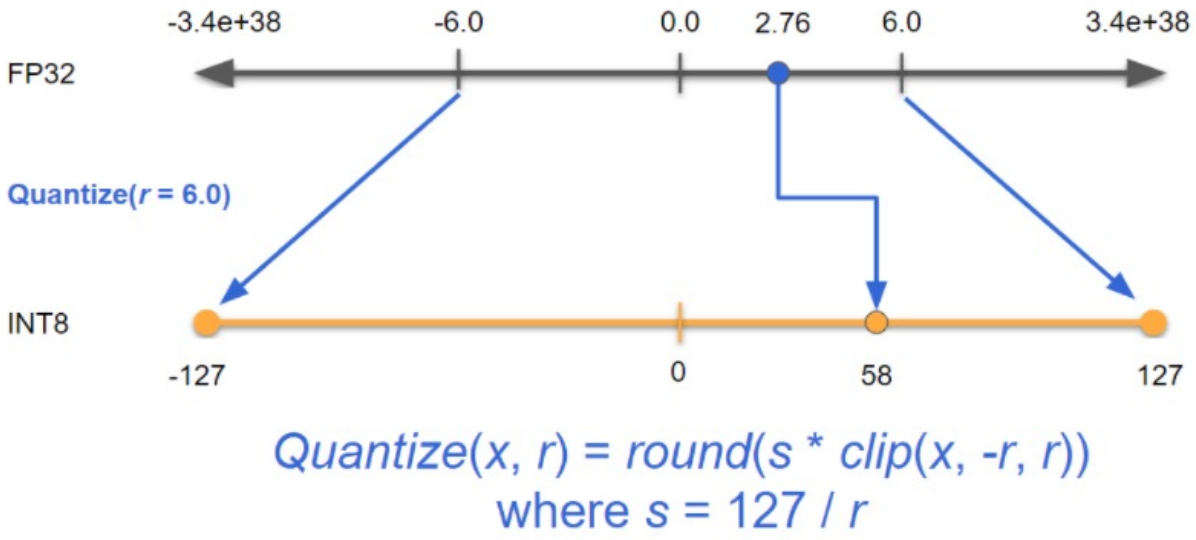
Базовый вариант это FP32, произойдет только оптимизация графа, описанная ранее.

Если вы счастливый обладатель видеокарт поколения Turing или Volta, то вам доступны также и точности FP16, INT8. На Pascal тоже будет работать, например, но в производительности ничего не поменяется.

FP16 позволяет снизить точность чисел и тем самым ускорить модель. Результат зависит от видеокарты.

TensoRT INT8

INT8 требует уже от нас калибровочного датасета, так как нам нужно откалиброваться на интервал [-127;127]. Основная идея в чем? Веса часто слабо отличаются друг от друга и их можно сгруппировать, используя квантизацию.



https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html

Но мы же можем потерять в точности?

Как показывают исследования, сильной потери в точности не будет, все кажется довольно приличным.

Table 1. Verified Models

	Native TensorFlow FP32	TF-TRT FP32	TF-TRT FP16	TF-TRT INT8	
	Volta and Turing	Volta and Turing	Volta and Turing	Volta	Turing
MobileNet v1	71.01	71.01	70.99	69.49	69.49
MobileNet v2	74.08	74.08	74.07	73.96	73.96
NASNet - Large	82.72	82.71	82.70	Work in progress	82.66
NASNet - Mobile	73.97	73.85	73.87	73.19	73.25
ResNet-50 v1.5[1]	76.51	76.51	76.48	76.23	76.23
ResNet-50 v2	76.43	76.37	76.4	76.3	76.3
VGG16	70.89	70.89	70.91	70.84	70.78
VGG19	71.01	71.01	71.01	70.82	70.90
Inception v3	77.99	77.99	77.97	77.92	77.93
Inception v4	80.19	80.19	80.19	80.14	80.08

https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html#performance

TF-TRT

Как мы уже ранее отмечали, TensorRT интегрирован в Tensorflow и его использование заметно упрощается.

1) Обучаем модель на tf или tf.keras
2) Сохраняем модель
3) Загружаем модель
4) Оптимизируем
5) Profit!

Оптимизация resnet50 для классификации imagenet (Tesla T4):

Base TF	TF-TRT FP32	TF-TRT FP16	TF-TRT INT8
202 img/sec	465 img/sec	467 img/sec	1939 img/sec

TF-TRT

Результаты в том числе сильно зависят от видеокарты, от ее производительность, типов ядер cuda, архитектуры.

Например, Tesla V100 16GB и оптимизация ResNet152V2:

Base TF	TF-TRT FP32	TF-TRT FP16	TF-TRT INT8
455 img/sec	565 img/sec	1693 img/sec	1746 img/sec

Tesla V100 16GB и оптимизация ResNet50

Base TF	TF-TRT FP32	TF-TRT FP16	TF-TRT INT8
973 img/sec	1389 img/sec	3874 img/sec	4031 img/sec

ONNX

- ONNX (Open Neural Network Exchange) – открытая библиотека для построения нейронных сетей. Но не только. С помощью ONNX можно обмениваться моделям из разных фреймворков.

Поддерживаемые фреймворки

- Caffe2
- Microsoft Cognitive Toolkit
- MXNet
- Tensorflow
- Pytorch
- XGBoost
- Catboost
- Etc..

https://onnx.ai/supported-tools.html

ONNX Runtime

- У ONNX есть свой движок для быстрого обучения и инференса моделей.
- Много различных стратегий оптимизации в части CPU, CUDA и многих других, но лично у меня не вышло обогнать ванильный Pytorch даже.
- В целом можно попробовать ускорить обучение моделей, например моделей Pytorch.
- https://github.com/Microsoft/onnxruntime#api-documentation

Оптимизация Pytorch

На самом деле вариантов вроде много, но ONNX не зашел.
Поэтому основной вариант следующий:

- Обучаем модель в Pytorch
- Конвертируем ее в ONNX
- Конвертируем из ONNX в Tensorflow
- Оптимизируем через TF-TRT

Конвертация Pytorch-ONNX

```
torch.onnx.export(model,
                  dummy_input,
                  "resnet_onnx/resnet.onnx",
                  input_names=['input’],
                  output_names=['output'],
                  dynamic_axes = {'input' : {0: 'batch', 1: 'sequence'},
                                  'output' : {0: 'batch'}
                  })
```

model – сам экземпляр класса модели

dummy_input – пример данных для прогона модели для ONNX

dynamic_axes – если не указать, то ONNX будет ожидать такой же по размеру батч, как и в dummy_input

Конвертация ONNX-Tensorflow

Для конвертации можно использовать библиотеку onnx_tf
https://github.com/onnx/onnx-tensorflow

```
import onnx
from onnx_tf.backend import prepare

model_onnx = onnx.load('resnet_onnx/resnet.onnx')
tf_rep = prepare(model_onnx, device='CUDA', auto_cast=True)
tf_rep.export_graph('resnet_onnx/resnet.pb')
```