Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

Компиляторы для глубоких нейросетевых моделей

Обзор инструментов глубокого обучения

При поддержке компании YADRO

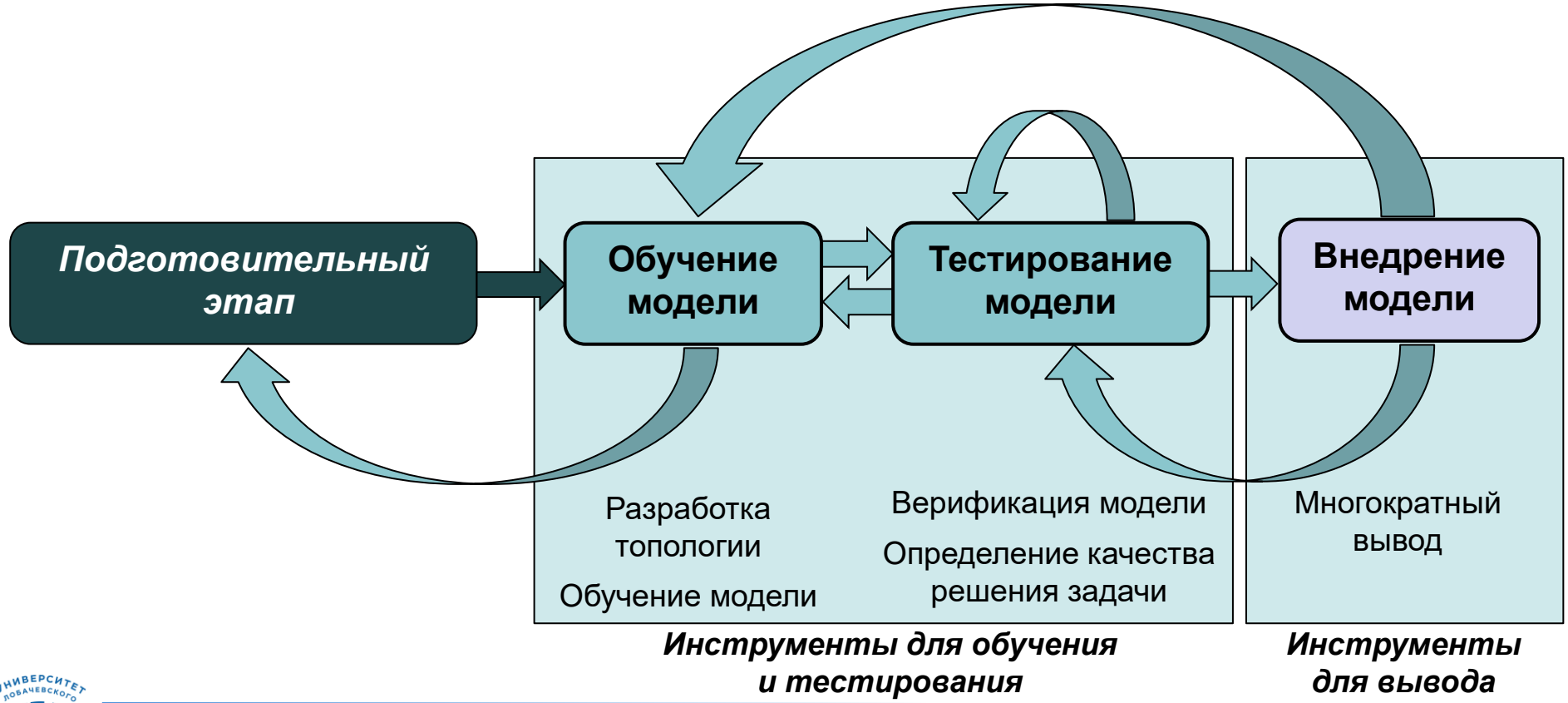
Кустикова В.Д.

Содержание…

Содержание

ВВЕДЕНИЕ

Общая схема решения задач с использованием глубокого обучения



Инструменты глубокого обучения

- ***Инструменты для обучения и тестирования глубоких моделей*** обеспечивают следующий функционал:
  - Описание архитектуры модели
  - Обучение построенной модели – реализация метода обратного распространения ошибки (backpropagation) или вызов соответствующей функции
  - Тестирование обученной модели
- ***Инструменты для вывода глубоких моделей*** обеспечивают следующий функционал:
  - Загрузка обученной модели
  - Многократный прямой проход по загруженной модели для вычисления выхода сети с целью решения задачи
  - *Примечание:* модель хранится в формате какого-либо фреймворка первой группы инструментов, поэтому требуется предварительная конвертация модели

ИНСТРУМЕНТЫ ДЛЯ ОБУЧЕНИЯ И ТЕСТИРОВАНИЯ ГЛУБОКИХ НЕЙРОННЫХ СЕТЕЙ

Обзор инструментов глубокого обучения для обучения и тестирования глубоких моделей…

№	Название	Интерфейсы	ОС	FCNN	CNN	RNN	AE	RBM	GAN
1	**TensorFlow**	Python, Java, Go	Linux, Windows, Mac OS, Android	+	+	+	+	+	+[1]
2	Caffe	C++, Python, MATLAB	Linux, Windows, OS X	+	+	+	+	–	–
3	Keras	Python	Linux, Vagrant	+	+	+	+	+	+[2]
4	Torch	C, Lua	Linux, iOS, Android	+	+	+	+	+	–
5	MXNet	C++, Python, R, Scala, Julia, Perl, Clojure, Java	Linux, Windows, Mac OS	+	+	+	+	+	+[3]
6	**PyTorch**	C++, Python, Java	Linux, Windows, Mac OS	+	+	+	+	+[4]	+[1]

[1] TensorFlow, PyTorch содержат библиотеки для реализации GAN [https://github.com/tensorflow/gan], [https://github.com/torchgan/torchgan].

[2] Keras содержит коллекцию широко известных моделей GAN [https://github.com/eriklindernoren/Keras-GAN].

[3] Стандартные средства MXNet позволяют разработать GAN [https://gluon.mxnet.io/chapter14_generative-adversarial-networks/gan-intro.html].

[4] Стандартные средства PyTorch позволяют разработать RBM.

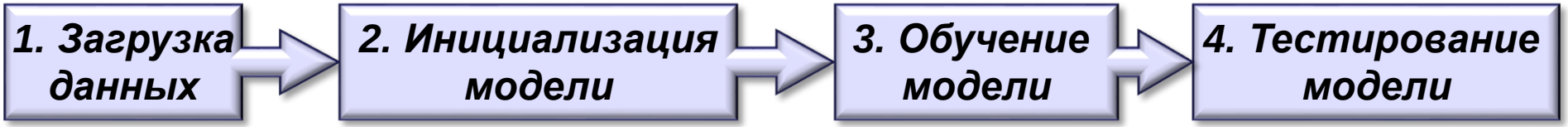
Обзор инструментов глубокого обучения для обучения и тестирования глубоких моделей…

- Поддерживаемые типы нейросетевых моделей:
  - FCNN (Fully-Connected Neural Networks) – полносвязные нейронные сети
  - CNN (Convolutional Neural Networks) – сверточные нейронные сети
  - RNN (Recurrent Neural Networks) – рекуррентные нейронные сети
  - AE (Autoencoders) – автокодировщики
  - RBM (Restricted Boltzmann Machines) – ограниченные машины Больцмана
  - GAN (Generative Adversarial Networks) – генеративные состязательные сети

Обзор инструментов глубокого обучения для обучения и тестирования глубоких моделей

- **Keras является надстройкой над TensorFlow**. Цель – упростить разработку глубоких моделей и использовать ее для быстрого старта
- **Torch** построен на базе скриптового языка программирования Lua, был популярен в течение долгого времени
- В библиотеке **MXNet** акцент сделан на переносимость и наличие широкого спектра программных интерфейсов
- **TensorFlow (Keras), MXNet, PyTorch** обеспечивают возможность распределенного обучения нейросетей
- Разработка **Caffe, Torch и MXNet** остановлена в связи с развитием библиотеки PyTorch, которая вобрала в себя положительный опыт разработки указанных инструментов
- Приведенный перечень инструментов можно расширять (ONNX Runtime, …)

Инструменты для обучения и тестирования глубоких моделей. Типовая схема использования



Использование функций, встроенных в инструментарий, или сторонних библиотек и пакетов

Формирование описания архитектуры сети

Вызов или высокоуровневая реализация метода обратного распространения ошибки

Проход по тестовым данным и вычисление ошибки на этих данных

Инструменты для обучения и тестирования глубоких моделей. План изучения

- **Библиотека PyTorch**
  - Общая информация
  - Ключевые особенности
  - Представление модели нейронной сети
  - Установка и программный интерфейс для языка Python
- **Постановка задачи распознавания рукописных цифр на наборе данных MNIST**
- **Архитектура глубоких моделей**
- **Пример разработки нейронной сети, обучение и тестирование сети с использованием PyTorch**

Общая информация о библиотеке PyTorch

- **PyTorch [https://pytorch.org]** – одна из самых широко используемых в настоящее время библиотек глубокого обучения
- PyTorch – библиотека машинного обучения с открытым исходным кодом, разработанная на базе библиотеки Torch
- Разрабатывается в основном в Facebook's AI Research lab (FAIR) с сентября 2016 года
- Распространяется под лицензией BSD (Modified BSD license)
- PyTorch вобрала положительный опыт разработки других инструментов глубокого обучения с точки зрения предоставляемого функционала, гибкости и возможностей расширения, поэтому является одной из наиболее популярных библиотек

Ключевые особенности библиотеки PyTorch

- Библиотека реализована на языках C, C++, Python с использованием технологии CUDA
- Программные интерфейсы: C++, Python, Java
- Поддерживаемые ОС: Linux, Windows и Mac OS
- PyTorch может работать в CPU- и GPU-режимах
- Реализована возможность распределенного обучения нейронных сетей
- Обширная экосистема инструментов и библиотек для расширения PyTorch и поддержки процесса разработки в разных областях
- PyTorch поддерживается на основных облачных платформах (Alibaba Cloud, Amazon Web Services, Google Cloud Platform, Microsoft Azure)

Представление модели нейронной сети в PyTorch

- Нейронная сеть – динамический граф вычислений
- Узлы этого графа соответствуют математическим операциям, входным и выходным данным
- Вычисление градиентов по переменным, соответствующим выходным данным, выполняется с помощью техники автоматического дифференцирования* по графу нейронной сети
- Автоматическое дифференцирование ≠ символьное дифференцирование
- Символьное дифференцирование дает формулу для вычисления производной
- Автоматическое дифференцирование описывает процедуру вычисления производной

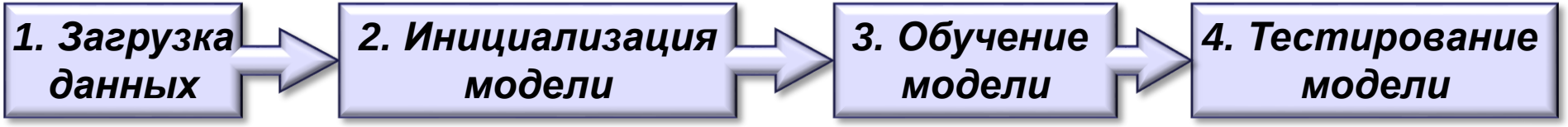
* Automatic differentiation [https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec10.pdf].

Установка библиотеки PyTorch и связанных пакетов

- Один из возможных вариантов установки:

```
# Создание и активация виртуальной среды
conda create -n pytorch_env
conda activate pytorch_env

# Установка некоторых пакетов для запуска на CPU
# Другие варианты установки: https://pytorch.org/get-started/locally
conda install pytorch torchvision torchmetrics cpuonly -c pytorch

# Деактивация виртуальной среды
conda deactivate

# Получение доступа к виртуальной среде из Jupiter notebook
python -m ipykernel install --user --name pytorch_env \
       --display-name "Python (pytorch_env)"
```

Схема построения глубокой модели в PyTorch



Создание итератора для множества загруженных примеров	Формирование архитектуры сети в виде последовательности слоев	Реализация метода обратного распространения ошибки (проход по числу эпох и пачек данных)	Проход по всем пачкам данных и промежуточное вычисление ошибки

Программные пакеты

`torchaudio`
`torchtext`
`torchvision`
`torch.utils.data`
	`torch.nn`	`torch.nn` `torch.optim`
		`torch.autograd`
		`torch.disributedtraining`
		`torchmetrics`
			`torch.utils.data`
			`torchmetrics`

`torch, torch.cuda`

Программный интерфейс PyTorch. Основные пакеты Python…

- **`torch`** – структуры данных для представления многомерных тензоров и определение математических операций над ними
- **`torch.nn`** – основные блоки для графа вычислений
- **`torch.autograd`** – классы и функции для автоматического дифференцирования
- **`torch.cuda`** – поддержка тензорных типов CUDA и операций на GPU
- **`torch.disributedtraining`** – поддержка распределенного обучения

Программный интерфейс PyTorch. Основные пакеты Python

- **`torch.optim`** – реализация методов оптимизации (Adadelta, Adagrad, Adam, SGD и др.)

- **`torch.utils.data`** – пакет, ядром которого является класс **`torch.utils.data.DataLoader`** (итератор над набором данных)

* PyTorch Documentation [https://pytorch.org/docs/stable/index.html].

Программный интерфейс PyTorch. Вспомогательные пакеты Python

- **`torchaudio`** – функции чтения/записи наборов данных с аудио и общие функции обработки аудио
- **`torchtext`** – утилиты для загрузки и обработки текстовых данных
- **`torchvision`** – функции загрузки наборов для задач компьютерного зрения, топологии известных глубоких моделей и преобразования изображений
- **`torchmetrics`** – реализация метрик качества решения задач

Программный интерфейс PyTorch. Пакет torch.nn…

- **Класс `torch.nn.Module`** – базовый класс для представления нейронной сети

- Разработка архитектуры нейронной сети – реализация наследника класса **`torch.nn.Module`**
  - Реализация конструктора – создание слоев сети и сохранение в полях класса
  - Переопределение метода **`forward()`** – реализация прямого прохода по нейронной сети для заданных входных данных (последовательное применение преобразований, соответствующих слоям сети, и возврат результата)
- Перегруженная операция `` `()` `` позволяет получить выходное значение сети для заданных входных данных
- Метод **`parameters()`** возвращает набор оптимизируемых параметров сети

Программный интерфейс PyTorch. Пакет torch.nn

- **Класс `torch.nn.CrossEntropyLoss`** – класс кросс-энтропийной функции ошибки

- Операция `()` позволяет вычислить значение функции ошибки между полученным на выходе сети значением и разметкой
- Метод **`backward()`** реализует обратный проход по нейронной сети (вычисление градиента функции ошибки)

Программный интерфейс PyTorch. Пакет torch.optim…

- **Класс `torch.optim.SGD`** реализует стохастический градиентный спуск

- Конструктор класса:

```
torch.optim.SGD(params, lr=<required>, momentum=0,
                dampening=0, weight_decay=0, nesterov=False)
```

  - **`params`** – набор настраиваемых параметров сети
  - Параметры обучения сети: **`lr`** – скорость обучения, **`momentum`** – импульс, **`weight_decay`** – коэффициент снижения вклада веса, **`dampening`** – коэффициент затухания импульса, **`nesterov`** – включение импульса Нестерова

Программный интерфейс PyTorch. Пакет torch.optim

- **Класс `torch.optim.SGD`** реализует стохастический градиентный спуск

- Метод **`zero_grad()`** устанавливает значения градиентов для всех оптимизируемых тензоров в ноль для их вычисления на следующем проходе
- Метод **`step()`** реализует один шаг метода оптимизации

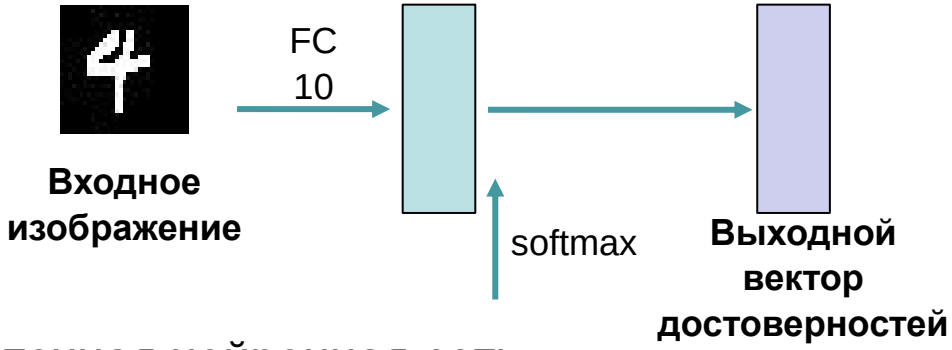
Постановка задачи распознавания рукописных цифр

- База изображений MNIST [http://yann.lecun.com/exdb/mnist]
- Одноканальные изображения
- Разрешение изображений – 28×28 пикселей
- Цифры отцентрированы на изображении
- Тренировочный набор – 60 000 изображений, тестовый – 10 000 изображений

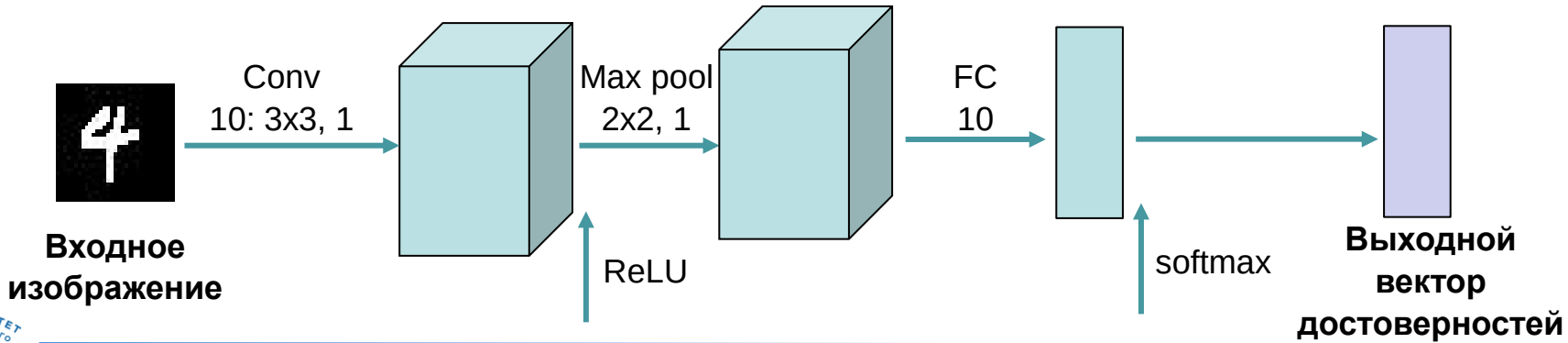
- ***Задача – определить, какая цифра приведена на изображении***

Архитектура глубоких моделей

- Логистическая регрессия:



- Однослойная сверточная нейронная сеть:



Пример разработки нейронной сети средствами библиотеки PyTorch

- Примеры обучения и тестирования логистической регрессии и однослойной сверточной сети с использованием библиотеки PyTorch: lectures/sources/03_PyTorch_Samples.ipynb

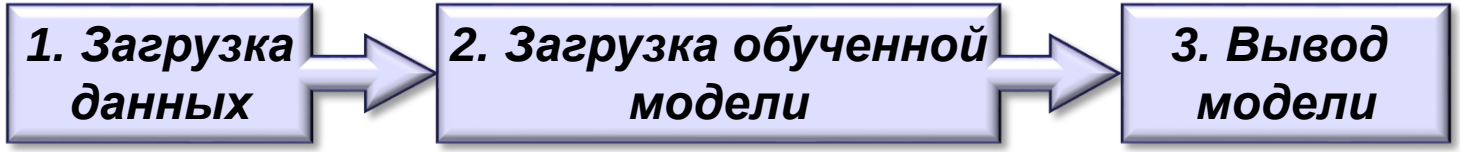
ИНСТРУМЕНТЫ ДЛЯ ВЫВОДА ГЛУБОКИХ НЕЙРОННЫХ СЕТЕЙ

Обзор инструментов глубокого обучения для вывода глубоких моделей

- Приведенные ранее фреймворки можно использовать как для обучения и тестирования глубоких нейросетевых моделей, так и для вывода

- Специализированные инструменты для вывода:
  - Модуль DNN библиотеки компьютерного зрения **OpenCV** [https://opencv.org]
  - **OpenVINO toolkit** [https://github.com/openvinotoolkit/openvino] – инструментарий для эффективного вывода на платформах компании Intel (CPUs, GPUs, NPUs), ARM
  - **TensorRT** [https://developer.nvidia.com/tensorrt] – SDK для эффективного вывода на графических ускорителях NVIDIA
  - **TensorFlow Lite** [https://www.tensorflow.org/lite] (переименован Lite RT) – мобильная библиотека для развертывания глубоких нейросетевых моделей на мобильных устройствах, микроконтроллерах и других периферийных устройствах (edge devices)

Инструменты для вывода глубоких моделей. Типовая схема использования



Использование функций, встроенных в инструментарий, или сторонних библиотек и пакетов

Форматы хранения модели:
- Формат исходного инструментария, с помощью которого модель была обучена
- Промежуточное представление модели, обрабатываемое фреймворком для вывода

Проход по обученной сети, получение выхода сети и последующая его обработка

Инструменты для последующего изучения

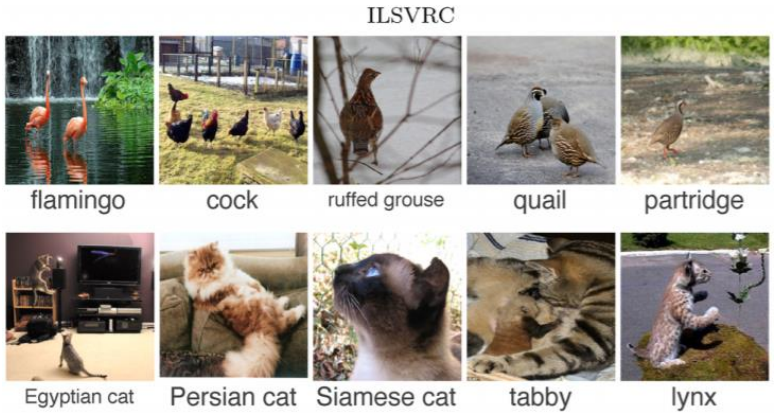
- **OpenVINO toolkit** [https://github.com/openvinotoolkit/openvino]
  - Компания Intel для эффективного вывода на CPU добавляет в систему команд x86 векторные и матричные расширения VNNI и AMX (RISC-V-устройства развиваются в аналогичном направлении)
  - Поддержка RISC-V находится в процессе разработки
  - Сообщество пользователей OpenVINO может «безболезненно» перенести приложения на устройства RISC-V
- **TensorFlow Lite** [https://www.tensorflow.org/lite] (переименован Lite RT)
  - Широко используется на мобильных платформах
  - Вывод оптимизирован для запуска RISC-V (библиотека примитивов XNNPACK)
  - На момент подготовки материалов TensorFlow Lite демонстрировал лучшие показатели производительности для ряда тестовых моделей

Инструменты для вывода глубоких моделей. План изучения

- **Постановка задачи классификации изображений с большим числом категорий**
- **Глубокие модели для вывода**
- **OpenVINO toolkit**
  - Ключевые особенности
  - Режимы вывода
  - Установка и программный интерфейс для языка Python
  - Пример реализации вывода средствами OpenVINO
- **TensorFlow Lite**
  - Ключевые особенности
  - Установка и программный интерфейс для языка Python
  - Пример реализации вывода средствами TensorFlow Lite

Постановка задачи классификации изображений

- Задача классификации изображений состоит в том, чтобы поставить в соответствие изображению класс объектов, содержащихся на этом изображении
- Примеры изображений и соответствующих им классов:



* Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M., Berg A.C., Fei-Fei L. ImageNet Large Scale Visual Recognition Challenge // International Journal of Computer Vision, 2015.

Инструменты для вывода глубоких моделей. План изучения

- Постановка задачи классификации изображений с большим числом категорий
- **Глубокие модели для вывода**
- OpenVINO toolkit
  - Ключевые особенности
  - Режимы вывода
  - Установка и программный интерфейс для языка Python
  - Пример реализации вывода средствами OpenVINO
- TensorFlow Lite
  - Ключевые особенности
  - Установка и программный интерфейс для языка Python
  - Пример реализации вывода средствами TensorFlow Lite

Глубокие модели

DenseNet-121

– Huang G., Liu Z., Maaten L., Weinberger K.Q. Densely Connected Convolutional Networks. – 2016. – [https://arxiv.org/pdf/1608.06993.pdf].

– Обученная модель загружена из Open Model Zoo – OpenVINO (в формате библиотеки TensorFlow)

```
pip install openvino
pip install openvino-dev[tensorflow2]
omz_downloader --name densenet-121-tf
omz_converter --name densenet-121-tf
```

MobileNet-v2

– Sandler M., Howard A., Zhu M., Zhmoginov A., Chen L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. – 2018. – [https://arxiv.org/pdf/1801.04381.pdf].

– Обученная модель загружена из Open Model Zoo – OpenVINO (в формате библиотеки TensorFlow)

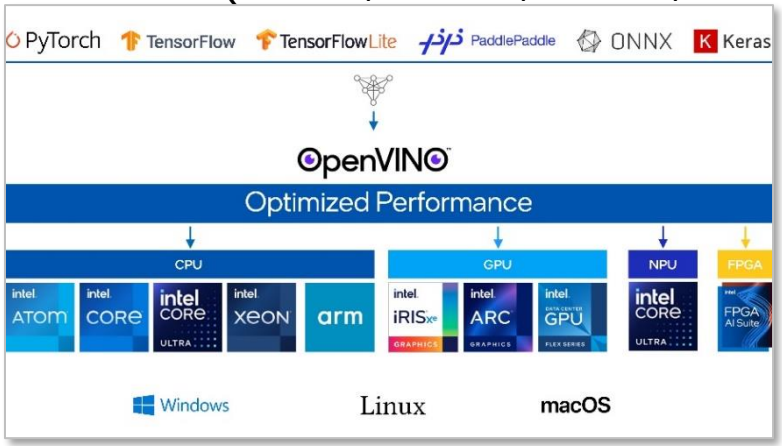
```
pip install openvino
pip install openvino-dev[tensorflow2]
omz_downloader --name mobilenet-v2-1.4-224
omz_converter --name mobilenet-v2-1.4-224
```

Инструменты для вывода глубоких моделей. План изучения

- Постановка задачи классификации изображений с большим числом категорий
- Глубокие модели для вывода
- **OpenVINO toolkit**
  - Ключевые особенности
  - Режимы вывода
  - Установка и программный интерфейс для языка Python
  - Пример реализации вывода средствами OpenVINO
- TensorFlow Lite
  - Ключевые особенности
  - Установка и программный интерфейс для языка Python
  - Пример реализации вывода средствами TensorFlow Lite

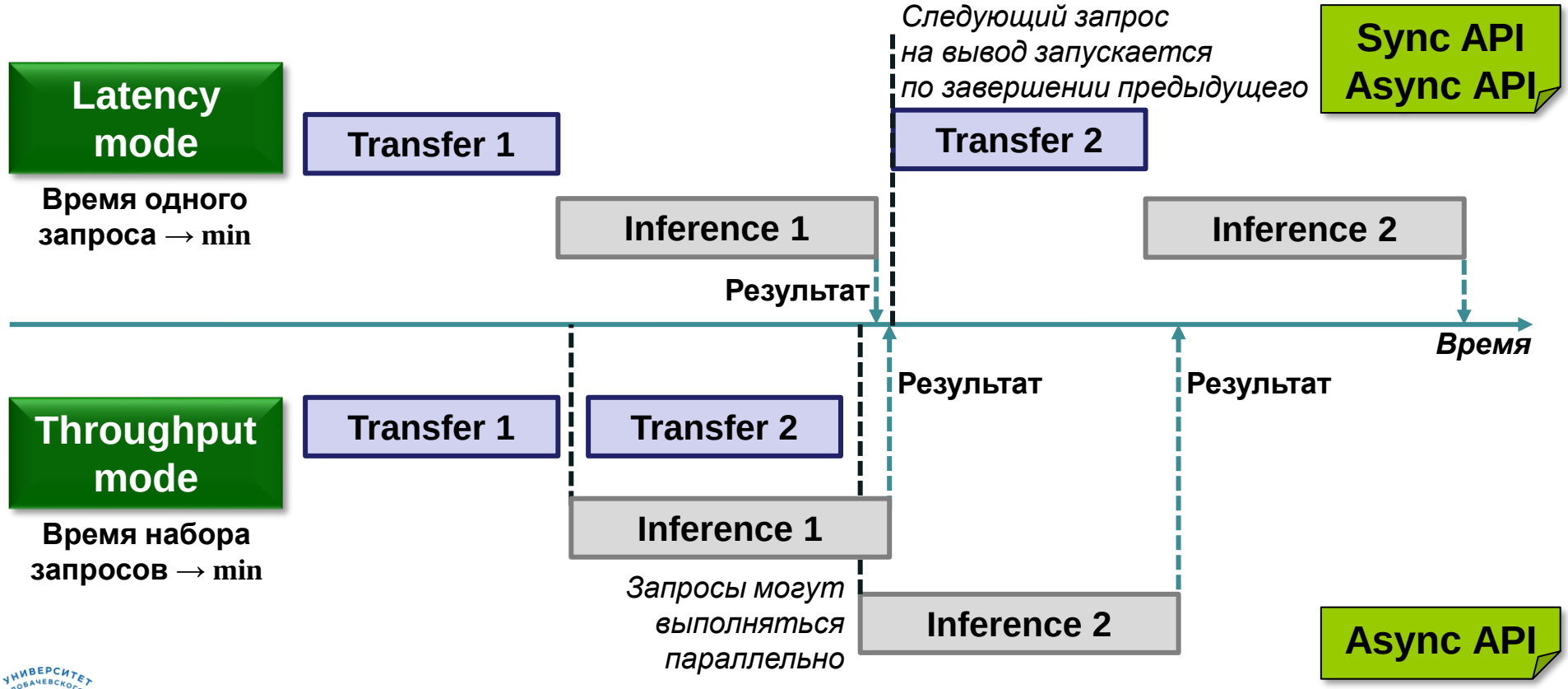
OpenVINO. Ключевые особенности

- Ускоряет производительность вывода при решении задач искусственного интеллекта, видео- и аудиоанализа, обработки естественного языка
- Поддерживает вывод моделей в формате TensorFlow, PyTorch и других
- Обеспечивает оптимизацию и развертывание глубоких моделей на аппаратной инфраструктуре компании Intel (CPUs, GPUs, NPUs, FPGAs)



* OpenVINO 2024.1 [https://docs.openvino.ai/2024/home.html].

OpenVINO. Режимы вывода



OpenVINO. Режимы вывода…

Запрос на вывод (inference request) – однократный прямой проход по сети

Режимы вывода:

- Режим минимизации времени выполнения одного запроса (latency mode)
  - Каждый следующий запрос на вывод запускается по завершении предыдущего
  - Реализуется с использованием Sync API или Async API
- Режим минимизации времени выполнения набора запросов (throughput mode)
  - Набор входных данных разбивается на пачки равного размера
  - Формируется очередь запросов на вывод для всего набора пачек
  - Несколько запросов могут выполняться параллельно
  - Очередной запрос начинает выполняться по завершении одного из параллельно запущенных
  - Реализуется с использованием Async API

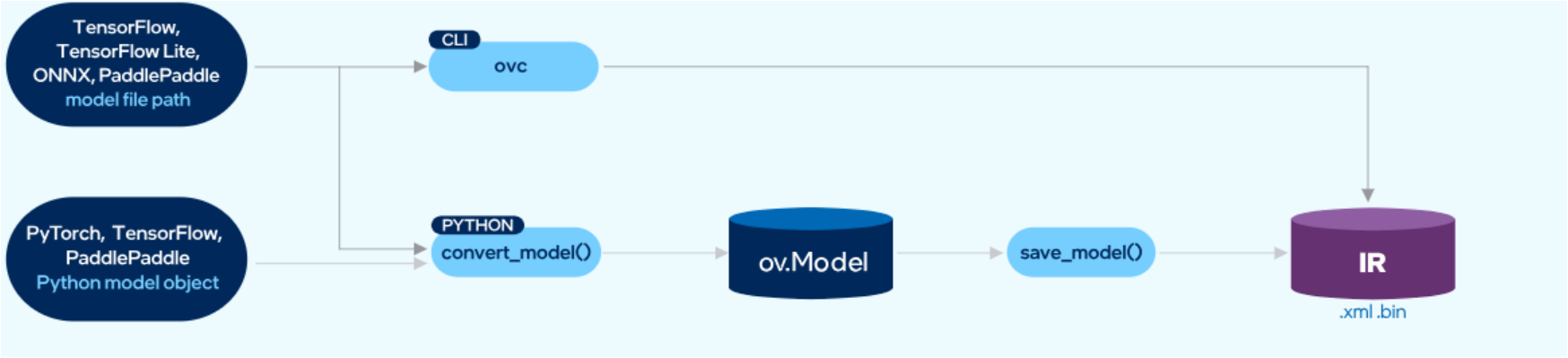
OpenVINO. Установка

- Один из возможных вариантов установки Python-пакета:

```
# Создание и активация виртуальной среды
conda create -n openvino_env
conda activate openvino_env

# Установка пакета OpenVINO для разработчика
pip install openvino-dev
# Проверка корректности установки
python -c "from openvino import Core; print(Core().available_devices)"

# Деактивация виртуальной среды
conda deactivate

# Получение доступа к виртуальной среде из Jupiter notebook
python -m ipykernel install --user --name openvino_dev \
       --display-name "Python (openvino_dev)"
```

OpenVINO. Подготовка глубокой модели для вывода…

Конвертация модели в промежуточное представление (Intermediate Representation, IR)



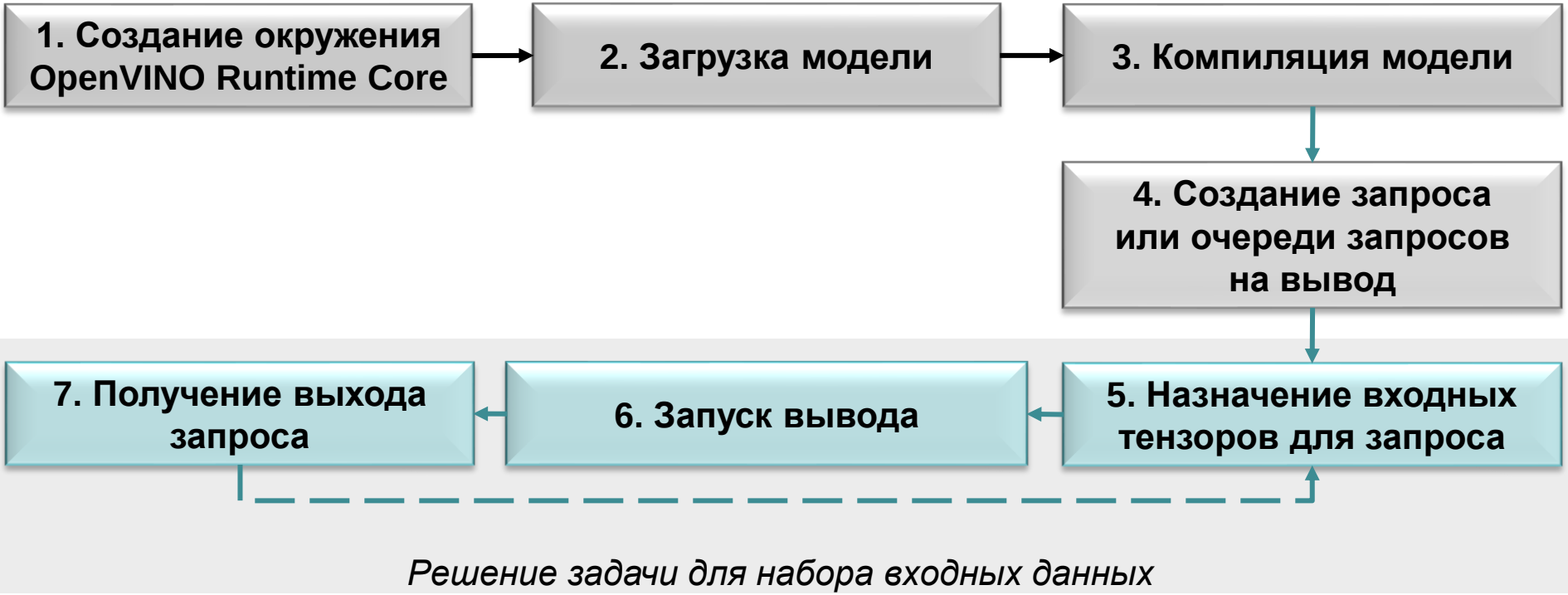
Чтение и компиляция сконвертированной модели

* OpenVINO Workflow [https://docs.openvino.ai/2024/openvino-workflow.html]

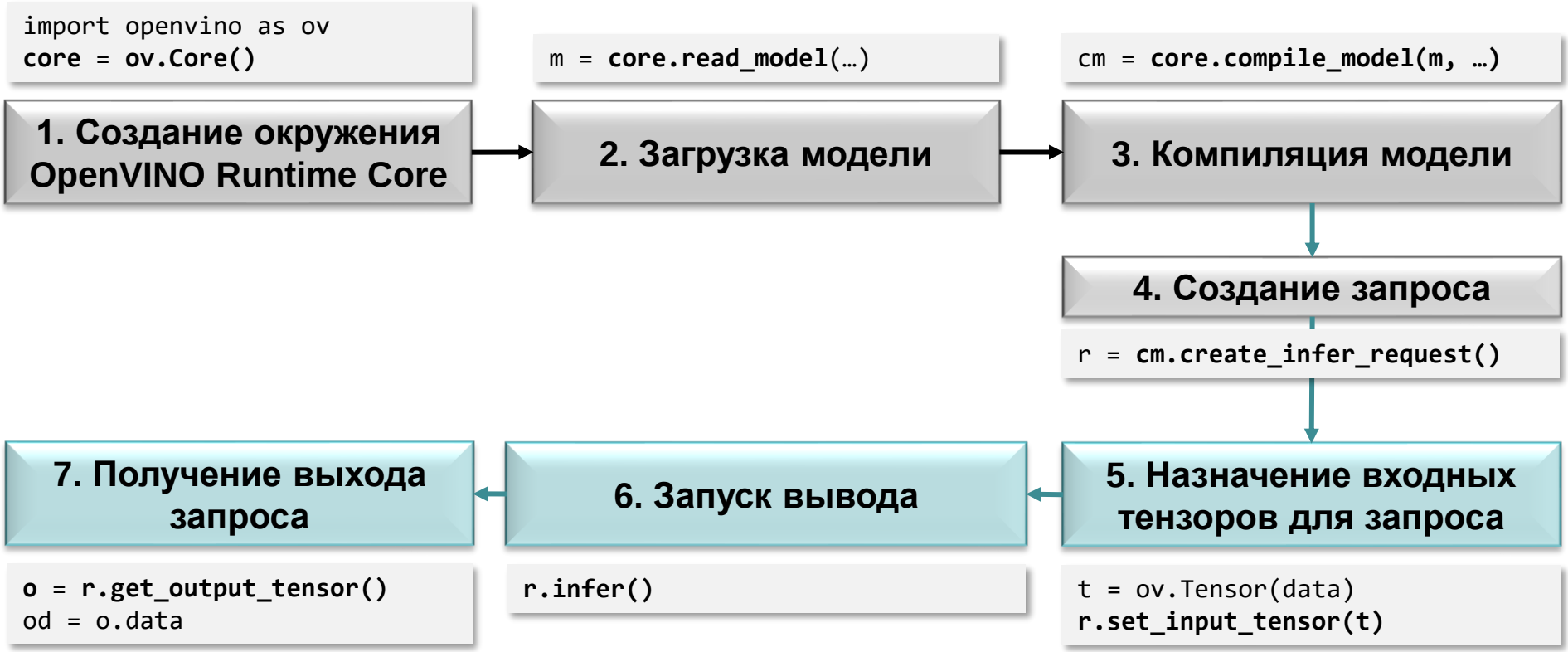
OpenVINO. Подготовка глубокой модели для вывода

- **Конвертация модели в промежуточное представление (Intermediate Representation, IR)** – предварительный этап, выполняется до вывода
  Интерфейсы:
  - Командный интерфейс (command language interface, CLI). Получает на вход модель в формате какого-либо фреймворка и конвертирует ее в IR (.xml + .bin)
  - ***Python API.*** Получает на вход объект загруженной модели, выполняет конвертацию (`convert_model(…)`) и сохранение (`save_model(…)`) модели в IR (.xml + .bin)
- **Чтение и компиляция сконвертированной модели** – этап, предшествующий вызову функций вывода. Интерфейсы:
  - C++ API. Получает сконвертированную в IR модель (.xml + .bin), читает (`read_model(…)`) и компилирует (`compile_model(…)`) ее для запуска вывода
  - ***Python API.*** Получает сконвертированную в IR модель (.xml + .bin), читает (`read_model(…)`) и компилирует (`compile_model(…)`) ее для запуска вывода

OpenVINO. Типовая схема работы приложения



OpenVINO. Типовая схема работы приложения. Sync API



OpenVINO. Типовая схема работы приложения. Async API

```
import openvino as ov
core = ov.Core()
```

1. Создание окружения OpenVINO Runtime Core

```
m = core.read_model(…)
```

2. Загрузка модели

```
cm = core.compile_model(m, …)
```

3. Компиляция модели

4. Создание очереди запросов на вывод

```
iq = ov.AsyncInferQueue(cm, mn)
```

5. Назначение входных тензоров для запроса

6. Запуск вывода

7. Получение выхода запроса

```
while iteration < num_batches:
    idle_id = iq.get_idle_request_id()
    if idle_id < 0:
        iq.wait(num_requests=1)
        idle_id = iq.get_idle_request_id()
    iq[idle_id].set_input_tensor(…)
    iq.start_async()
    iteration += 1
iq.wait_all()
```

OpenVINO. Программный интерфейс…

1. ***Создание и настройка окружения OpenVINO Runtime Core:***
   – Создание объекта окружения класса **`Core`** – вызов конструктора класса
   – Входные параметры: отсутствуют
   – Выходные параметры: объект типа **`Core`**

```
import openvino as ov

core = ov.Core()
```

   – Добавление расширений
     • **`core.add_extension(…)`** – регистрация расширений, содержащих реализацию нестандартных операций, в окружении
   – Настройка параметров
     • **`core.set_property(…)`** – настройка параметров исполнения вывода на устройстве

OpenVINO. Программный интерфейс…

2. ***Загрузка модели:***
   – Вызов метода **`read_model(…)`** для объекта окружения **`Core`** позволяет загрузить модель в формате OpenVINO IR, ONNX, PaddlePaddle, TensorFlow и TensorFlow Lite
   – Входные параметры: пути до файлов модели (описание в формате .xml и обученные веса в формате .bin для OpenVINO IR, .onnx для ONNX, .pdmodel для PaddlePaddle, .pb для TensorFlow и .tflite для TensorFlow Lite)
   – Выходные параметры: объект типа **`Model`**

   – Пример чтения модели в формате OpenVINO IR:

```
model = core.read_model(model='model.xml', weights='model.bin')
```

   Примечание: если параметр **`weights`** не задан, то выполняется поиск файла с весами, название которого совпадает с названием файла, переданного в параметр **`model`**. Если файл с весами не обнаружен, то загружается только архитектура

OpenVINO. Программный интерфейс…

3. ***Компиляция модели под конкретное устройство:***
   - Вызов метода **`compile_model(…)`** для объекта окружения **`Core`** позволяет получить модель, скомпилированную для вывода под конкретное устройство, параметры запуска описаны в окружении
   - Входные параметры: объект класса **`Model`**
   - Выходные параметры: объект класса **`CompiledModel`**
   - Пример компиляции модели для запуска на Intel CPU:

```
compiled_model = core.compile_model(model, 'CPU')
```

   - *Примечание 1:* метод **`compile_model(…)`** перегружен и может принимать название файла модели, при этом выполняет сразу загрузку и компиляцию. Такой вызов может быть более эффективен, чем **`read_model(…) + compile_model(…)`**
   - *Примечание 2:* перед компиляцией модели необходимо установить размер входа через **`model.reshape([batch_size,width,height,nchannels])`** (для моделей TensorFlow) или **`model.reshape([batch_size,nchannels,width,height])`**

OpenVINO. Программный интерфейс…

4. ***Создание запроса на вывод:***
   – **Sync API:** вызов метода **`create_infer_request()`** для объекта скомпилированной модели **`CompiledModel`**

```
request = compiled_model.create_infer_request()
```

   – **Async API:** создание очереди запросов **`AsyncInferQueue`**, содержащей **`max_num_requests`** запросов

```
infer_queue = ov.AsyncInferQueue(compiled_model, max_num_requests)
```

OpenVINO. Программный интерфейс…

5. ***Назначение входных тензоров:***
   – Создание объекта типа **`Tensor`** из загруженных данных **`data`** и установка входа

```
input_tensor = ov.Tensor(data)
request.set_input_tensor(input_tensor)
```

   – *Примечание 1:* при наличии нескольких входов у модели необходимо назначить входные тензора для каждого из них в соответствии с названием входа **`name`**

```
tensor = ov.Tensor(data)
request.set_tensor(name, tensor)
```

   – *Примечание 2:* размеры входного тензора **`tensor`** должны соответствовать размерам входа нейронной сети, которые были получены при загрузке модели или установлены через вызов **`model.reshape(…)`**
   – *Примечание 3:* загрузка данных выполняется средствами сторонних библиотек (для загрузки и предварительной обработки изображений используется OpenCV)

OpenVINO. Программный интерфейс…

6. ***Запуск вывода:***
   – **Sync API:** вызов метода **`infer()`** у объекта запроса на вывод

```
request.infer()
```

   – **Async API:** ожидание завершения работы какого-либо запроса, передача очередной пачки данных и запуск вывода

```
while iteration < num_batches:
    idle_id = infer_queue.get_idle_request_id() # id простаивающего запроса
    if idle_id < 0:
        infer_queue.wait(num_requests=1) # ожидание освобождения запроса
        idle_id = infer_queue.get_idle_request_id()
    infer_queue[idle_id].set_input_tensor(ov.Tensor(data[iteration]))
    infer_queue.start_async() # запуск
    iteration += 1
infer_queue.wait_all()
```

OpenVINO. Программный интерфейс…

7. ***Получение выхода запроса:***
   - Вызов метода **`get_output_tensor()`** для объекта запроса на вывод

```
output = request.get_output_tensor()
output_data = output.data
```

   - *Примечание 1:* при наличии нескольких выходов у модели результат вывода можно получить через поле **`results`** у объекта запроса на вывод. Пример формирования словаря из выходных тензоров

```
result = {}
for output_node, tensor in infer_request.results.items():
    result[output_node.get_any_name()] = copy(tensor)
```

   - *Примечание 2:* последующая обработка полученного выходного тензора **`output_data`** зависит от задачи и способа интерпретации выхода сети
   - *Примечание 3:* для Async API получение выходов реализуется посредством регистрации функции обратного вызова для объекта очереди запросов (пример далее)

OpenVINO. Пример реализации вывода

- Пример реализации вывода с использованием Sync API: lectures/sources/03_OpenVINO_Samples_sync.ipynb

- Пример реализации вывода с использованием Async API: lectures/sources/03_OpenVINO_Samples_async.ipynb

Инструменты для вывода глубоких моделей. План изучения

- Постановка задачи классификации изображений с большим числом категорий
- Глубокие модели для вывода
- OpenVINO toolkit
  - Ключевые особенности
  - Режимы вывода
  - Установка и программный интерфейс для языка Python
  - Пример реализации вывода средствами OpenVINO
- **TensorFlow Lite**
  - Ключевые особенности
  - Установка и программный интерфейс для языка Python
  - Пример реализации вывода средствами TensorFlow Lite

TensorFlow Lite. Ключевые особенности

- Оптимизированный запуск нейросетевых моделей на маломощных устройствах
- Кросс-платформенность: Android, iOS, embedded Linux, microcontrollers
- Разнообразие поддерживаемых языков программирования: Java, Swift, Objective-C, C++, Python
- Высокая производительность вывода
  - Оптимизация на уровне глубоких моделей (например, за счет квантизации)
  - Применение аппаратно-зависимых оптимизаций

TensorFlow Lite. Установка

- Один из возможных вариантов установки Python-пакета:

```
# Создание и активация виртуальной среды
conda create -n tflite_env
conda activate tflite_env

# Установка пакета tensorflow, включающего lite-версию
conda install conda-forge::tensorflow

# Деактивация виртуальной среды
conda deactivate

# Получение доступа к виртуальной среде из Jupiter notebook
python -m ipykernel install --user --name tflite_dev \
       --display-name "Python (tflite_dev)"
```

- *Примечание:* вместо пакета **`tensorflow`** можно установить **`tflite-runtime`**

TensorFlow Lite. Типовая схема работы приложения

1. Загрузка модели в формате .tflite

```
try:
    import tensorflow.lite as tflite
except ModuleNotFoundError:
    import tflite_runtime.interpreter as tflite
model = tflite.Interpreter(model_path=model_path)
```

2. Преобразование данных

```
model.allocate_tensors()
input_details = model.get_input_details()
model.set_tensor(input_details[0]['index'], input_data)
```

3. Запуск вывода

```
model.invoke()
```

4. Получение результатов

```
output_details = model.get_output_details()
output_data = model.get_tensor(output_details[0]['index'])
```

TensorFlow Lite. Программный интерфейс…

1. ***Загрузка модели:***
   – Создание объекта класса **`Interpreter`**
   – Параметры конструктора класса:
   - **`model_path`** – путь до модели в формате .tflite
   - **`num_threads`** – количество потоков, обеспечивающих параллелизм на общую память
   - **`experimental_delegates`** – набор загруженных делегатов (реализации ядер, обеспечивающие аппаратное ускорение моделей). Подробнее здесь
   – Пример загрузки модели:

```
try:
    import tensorflow.lite as tflite # установлен tensorflow с tflite
except ModuleNotFoundError:
    import tflite_runtime.interpreter as tflite # установлен tflite-runtime

model = tflite.Interpreter(model_path=model_path)
```

TensorFlow Lite. Программный интерфейс…

2. ***Преобразование данных:***
   – Выделение памяти для рабочих тензоров модели

```
model.allocate_tensors()
```

   – Получение информации о входных тензорах модели

```
input_details = model.get_input_details()
```

   – Подготовка входных данных (опционально)
     • Предполагает чтение, масштабирование, вычитание среднего и другие преобразования
     • Тензора имеют тип **`numpy.array`** с элементами типа **`numpy.float32`**
   – Установка данных в качестве входных тензоров

```
model.set_tensor(input_details[0]['index'], input_data)
```

   – *Примечание 1:* если модель имеет несколько входов, то инициализируются все
   – *Примечание 2:* для изменения размеров входа вызвать **`resize_tensor_input(…)`**

```
model.resize_tensor_input(input_details[0]['index'], input_shape)
```

TensorFlow Lite. Программный интерфейс…

3. ***Запуск вывода:***
   – Вызов метода **`invoke()`** для объекта загруженной модели типа **`Interpreter`**

```
model.invoke()
```

TensorFlow Lite. Программный интерфейс

4. ***Обработка результатов:***
   - Получение информации о выходных тензорах:

```
output_details = model.get_output_details()
```

   - Получение копии выходного тензора:

```
output_data = model.get_tensor(output_details[0]['index'])
```

   - *Примечание:* при наличии нескольких выходов и необходимости их последующей обработки реализуется обход по массиву **`output_details`**

TensorFlow Lite. Пример реализации вывода

- Пример реализации вывода с использованием TensorFlow Lite: lectures\sources\03_TFLite_samples.ipynb

Заключение

- Существует множество инструментов глубокого обучения, обеспечивающих высокую скорость работы нейросетевых моделей на разной аппаратуре
- Представленный материал демонстрирует базовые возможности инструментов PyTorch, OpenVINO и TensorFlow Lite, связанные с типовыми схемами их использования
- В рассмотренных инструментах предусмотрено множество дополнительных настроек, позволяющих повысить производительность вывода глубоких нейросетевых моделей
- Оптимальные параметры запуска вывода определяются внутренним устройством модели и параметрами вычислительной инфраструктуры
- Определение оптимальных параметров запуска вывода – тема отдельного исследования для каждой конкретной модели и решаемой задачи

Литература

1. PyTorch [https://pytorch.org]
2. OpenVINO toolkit [https://github.com/openvinotoolkit/openvino].
3. TensorFlow Lite [https://www.tensorflow.org/lite].
4. OpenCV [https://opencv.org].
5. TensorRT [https://developer.nvidia.com/tensorrt].

Примеры исходного кода

- Примеры обучения и тестирования логистической регрессии и однослойной сверточной сети с использованием библиотеки PyTorch: lectures/sources/03_PyTorch_Samples.ipynb
- Пример реализации вывода с использованием OpenVINO (Sync API): lectures/sources/03_OpenVINO_Samples_sync.ipynb
- Пример реализации вывода с использованием OpenVINO (Async API): lectures/sources/03_OpenVINO_Samples_async.ipynb
- Пример реализации вывода с использованием TensorFlow Lite: lectures\sources\03_TFLite_samples.ipynb

Авторский коллектив

- Мееров Иосиф Борисович, к.т.н., доцент, зав. каф. ВВиСП Института ИТММ ННГУ им. Н.И. Лобачевского
- Кустикова Валентина Дмитриевна, к.т.н., доцент каф. ВВиСП Института ИТММ ННГУ им. Н.И. Лобачевского
- Родимков Юрий Александрович, младший научный сотрудник каф. ВВиСП Института ИТММ ННГУ им. Н.И. Лобачевского
- Сысоев Александр Владимирович, к.т.н., доцент каф. ВВиСП Института ИТММ ННГУ им. Н.И. Лобачевского