Нейронные сети для мобильных приложений

Вводная лекция

Искусственный интеллект:
математические модели и прикладные решения
2022

Структура лекции

- Фреймворки для обучения нейронных сетей
- Мобильные устройства и их особенности
- Фреймворки для запуска нейронных сетей на мобильных устройствах
- Проблема переноса моделей между фреймворками
- Оптимизация моделей

Фреймворки для обучения нейронных сетей

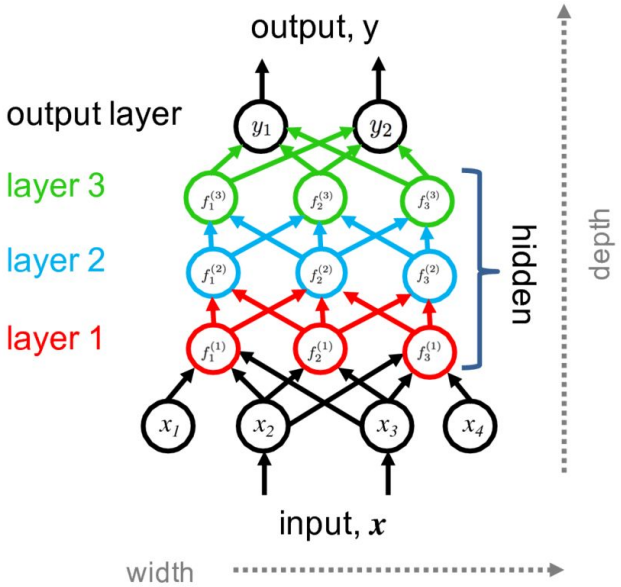
- PyTorch

- TensorFlow

- MXNet

Что такое модель?

$$y = f(x) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(x))))$$



NETRON

Мобильные устройства

1. Мобильные смартфоны

2. Встраиваемые устройства

Система на кристалле (SoC)

- Один или несколько микроконтроллеров, микропроцессоров или ядер цифровой обработки сигналов
- NPU (Neural Processor Unit)
- GPU (Graphic Processor Unit)
- TPU (Tensor Processor Unit)
- Дополнительные вспомогательные модули

Смартфоны: операционные системы и процессоры

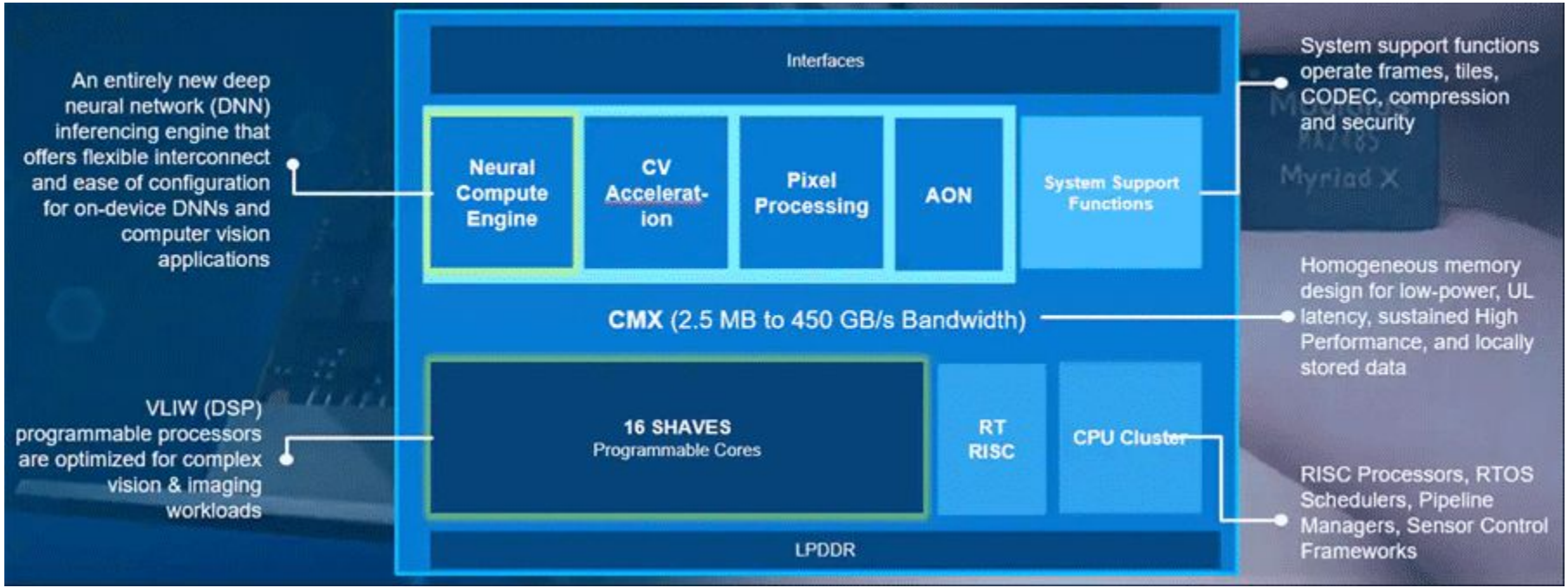
- iOS
  - Apple Ax Bionic
  - M1
- Harmony OS
  - HiSilicon K3 (Kirin)
- Android
  - Qualcomm (Snapdragon)
  - HiSilicon K3 (Kirin)
  - Samsung Exynos
  - Google Tensor

Встраиваемые устройства

- Intel Neural Compute Stick
- Intel NUC
- Raspberry Pi
- Nvidia Jetson Nano, Xavier NX, AGX Xavier
- Google Coral

Intel Neural Compute Stick

Intel Neural Compute Stick



Intel Neural Compute Stick

- Supported frame
  - TensorFlow
  - Caffe

Intel NUC

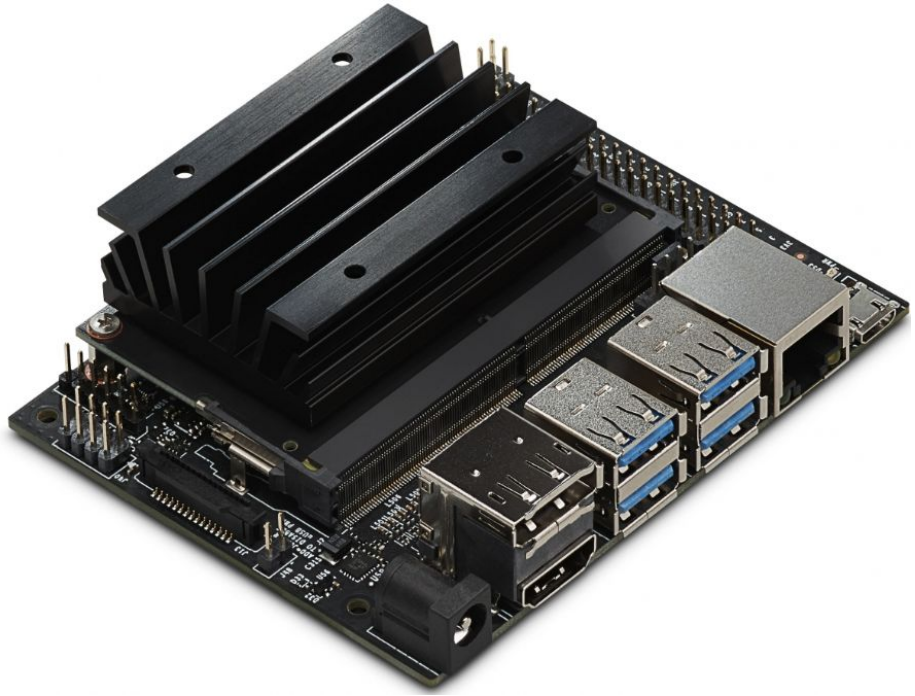
- CPU: Intel i5/i7
- RAM: Custom
  (~16 Gb)
- DL Accelerator:
  Из CPU
- Supported frameworks:
  Все на x86

Raspberry Pi 4

- CPU: quad-core ARM A72
- RAM: 2-8 Gb
- DL Accelerator: Отсутствует
- Supported frameworks:
  Все на ARMv8

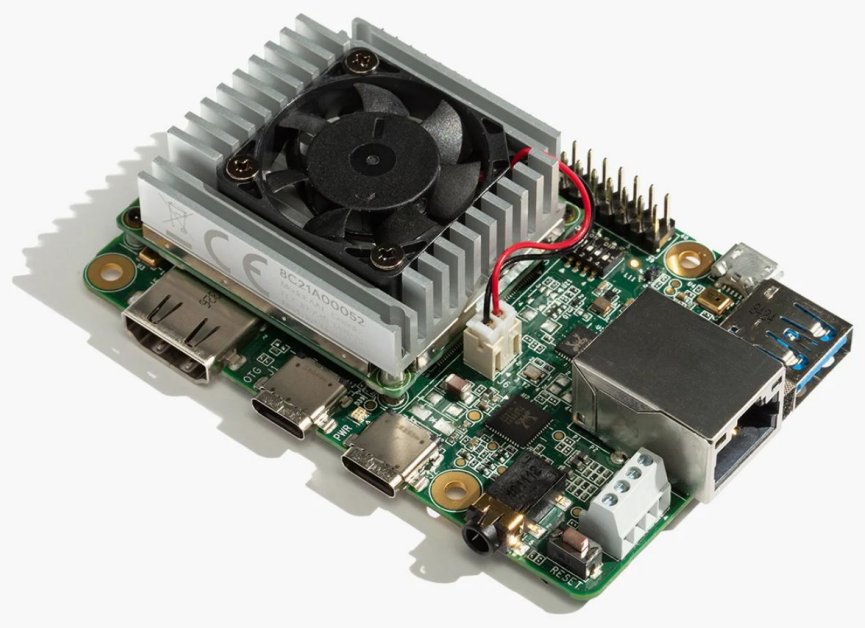
Nvidia Jetson Nano

- CPU: quad-core ARM A57
- RAM: 2-4 Gb
- DL Accelerator:
  128 Maxwell CUDA cores
- Supported frameworks:
  Проприетарные на ARMv8



Google Coral

- CPU: quad-core ARM A57
- RAM: 1 Gb
- DL Accelerator:
  Google TPU (4 TOPS)
- Supported frameworks:
  TensorFlow (ограниченный)

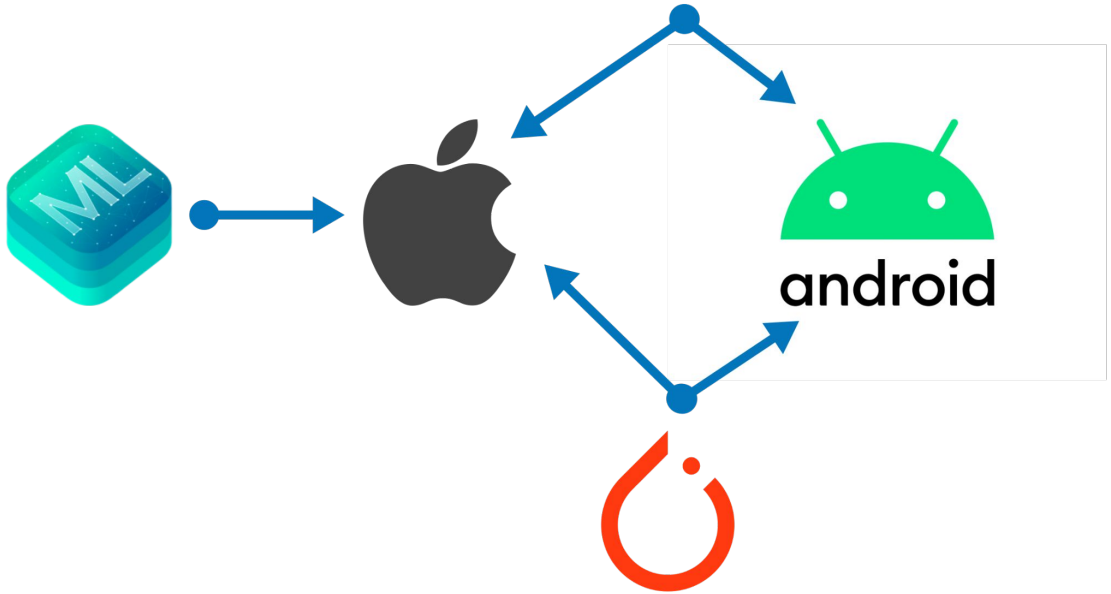


Фреймворки для запуска нейронных сетей на мобильных устройствах

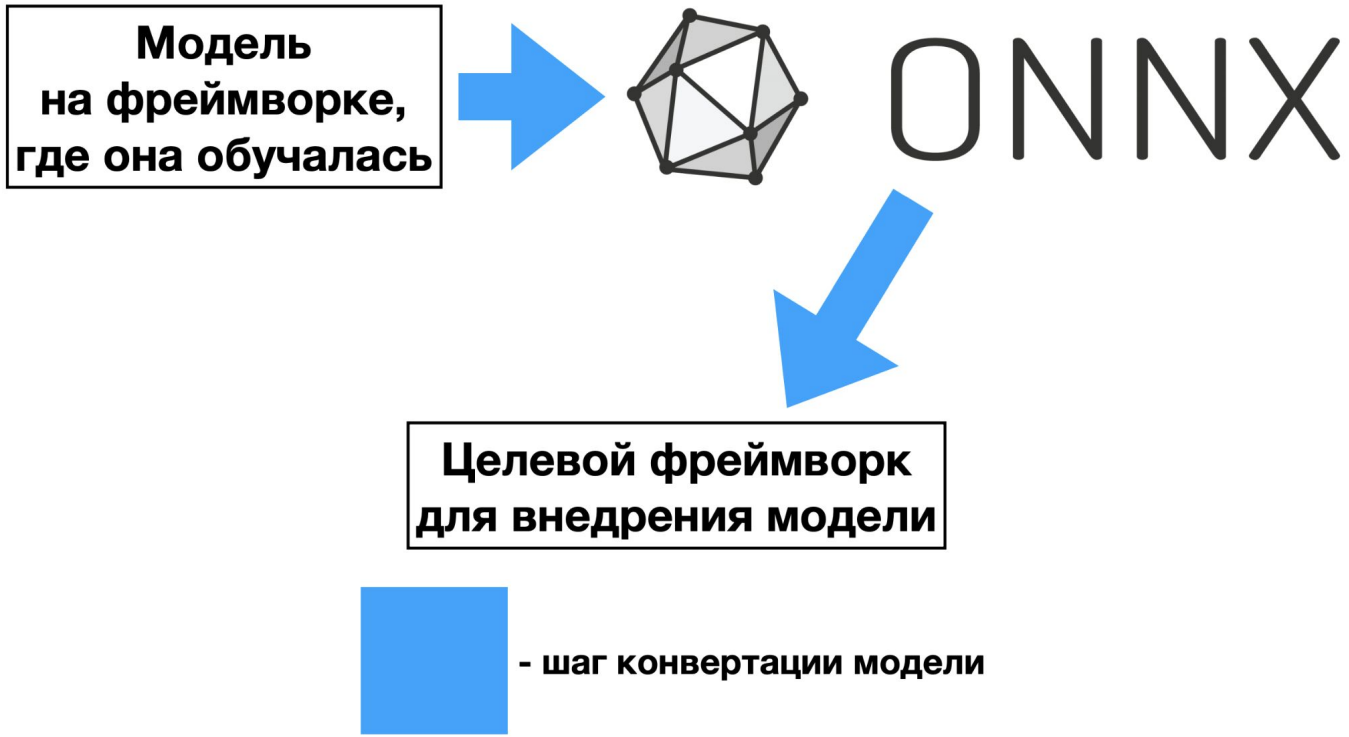
- TensorFlow, TensroFlow-Lite
- PyTorch, PyTorch Mobile
- CoreML
- ONNX
- OpenVINO
- TensorRT
- Caffe

Фреймворки для запуска нейронных сетей на смартфонах

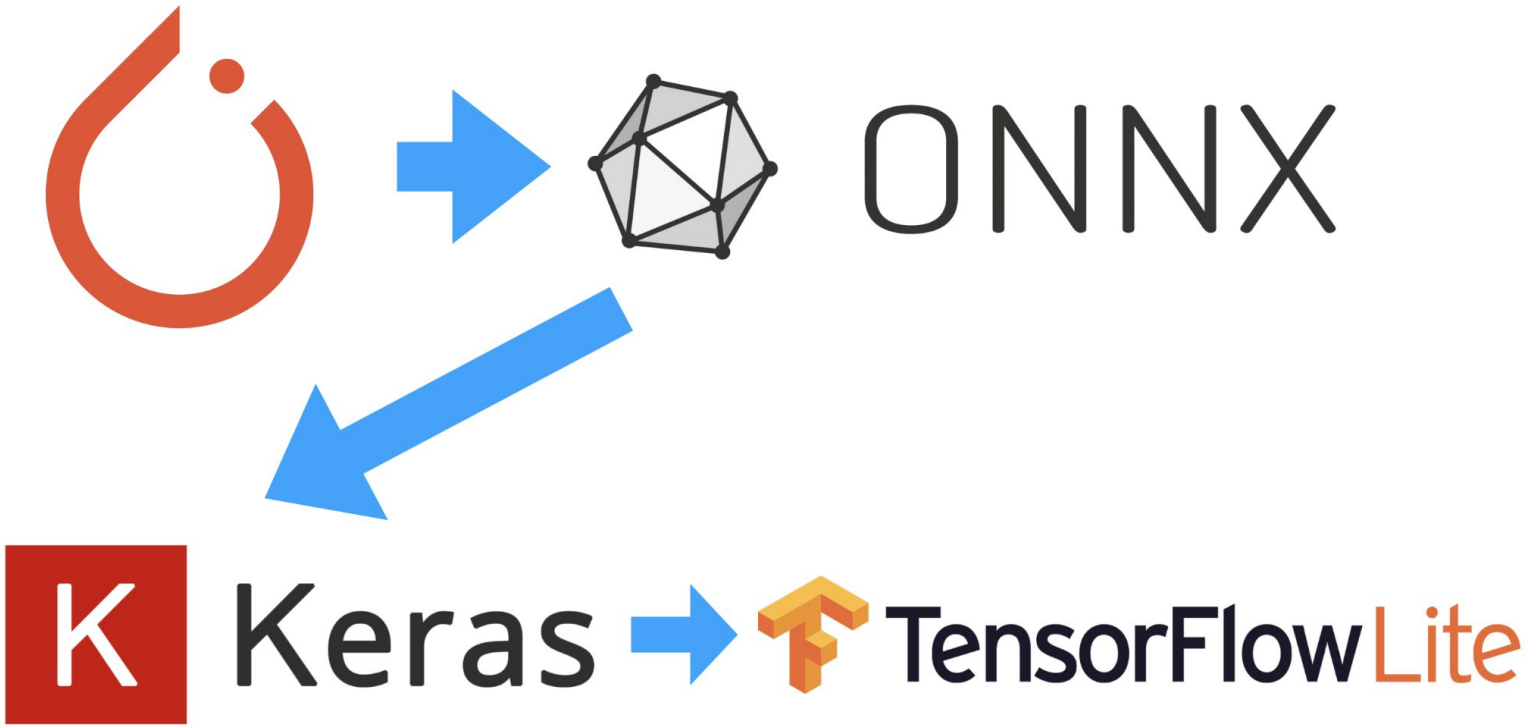
- TensorFlow Lite



Классическая схема переноса модели



Пример переноса с PyTorch на TensorFlow-Lite



Оптимизация моделей

- Квантизация обученной модели (к UINT8)
- Использование половинной точности вычислений чисел с плавающей запятой
- Pruning (сокращение количества параметров модели)
- Дистилляция модели (сокращение количества слоев)

Фреймворки для запуска моделей на мобильных устройствах

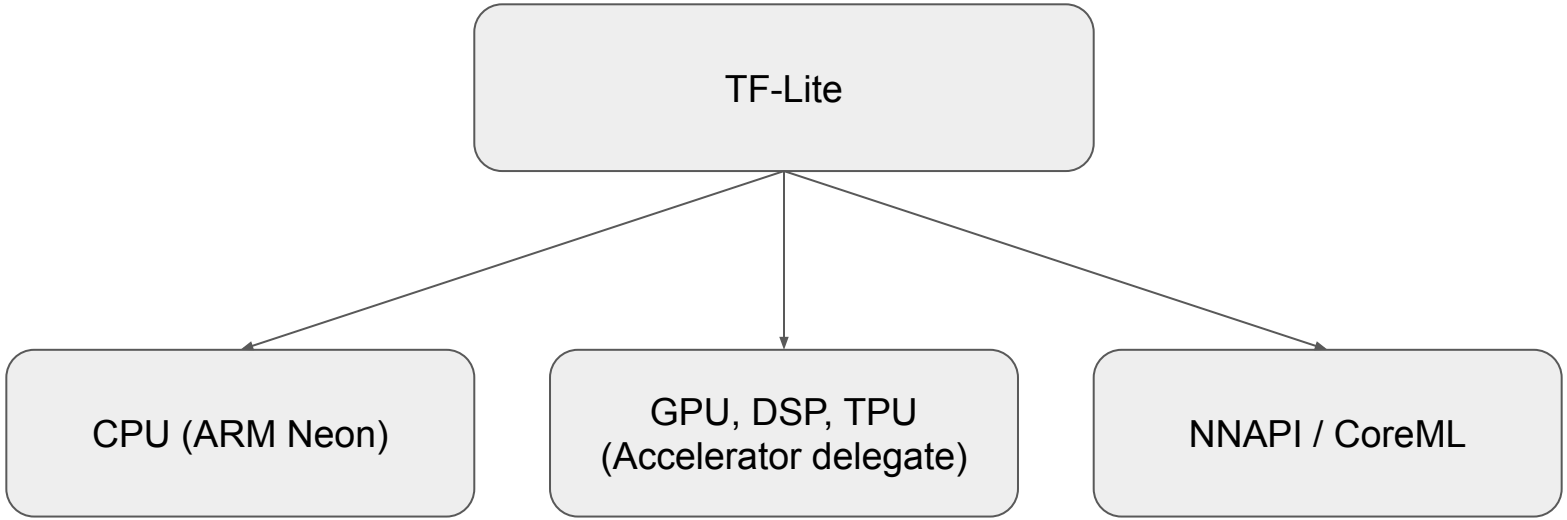
Лекция 2

Искусственный интеллект:
математические модели и прикладные решения
2022

Структура лекции

- TensorFlow-Lite
- PyTorch Mobile
- CoreML
- Tengine
- Сравнение производительности
- Проблема падения точности модели

TensorFlow-Lite



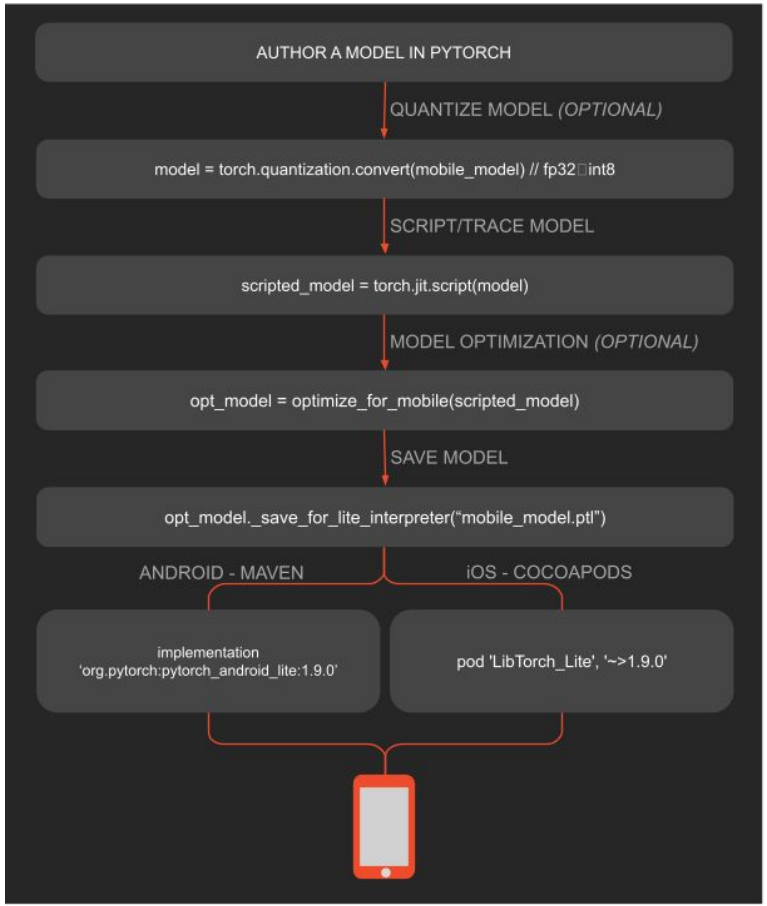
TensorFlow-Lite

Model Name	Device	CPU, 4 threads	GPU	NNAPI
Mobilenet_1.0_224(float)	Pixel 3	23.9 ms	6.45 ms	13.8 ms
	Pixel 4	14.0 ms	9.0 ms	14.8 ms
Mobilenet_1.0_224 (quant)	Pixel 3	13.4 ms	---	6.0 ms
	Pixel 4	5.0 ms	---	3.2 ms

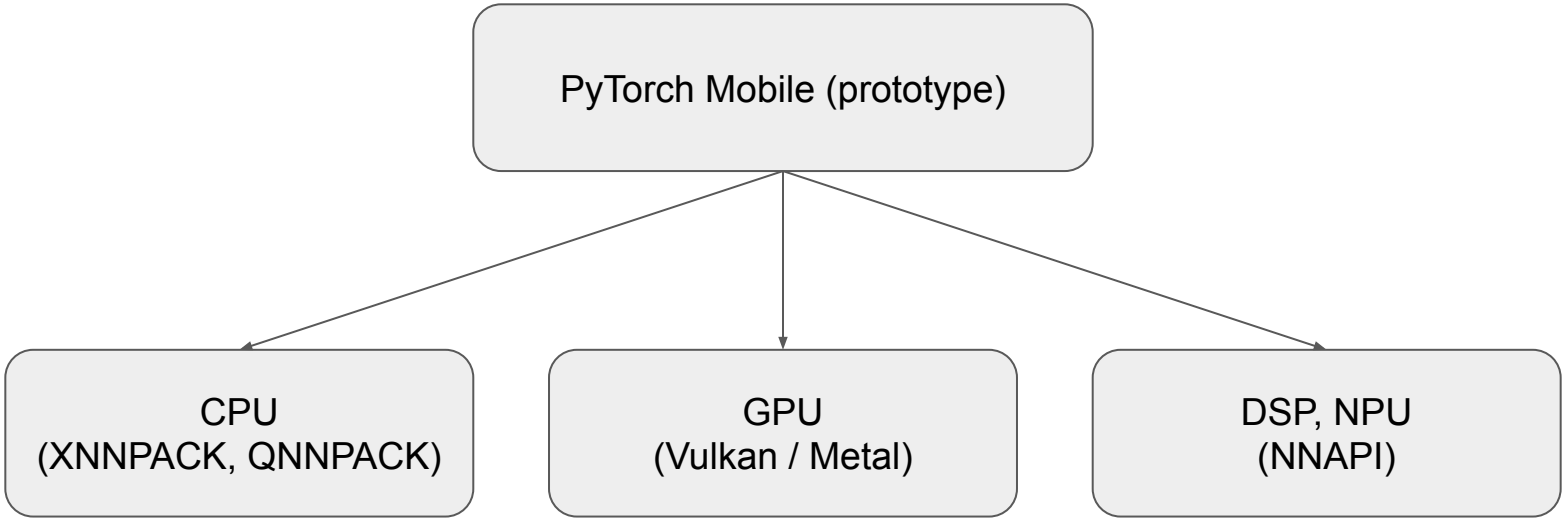
TensorFlow-Lite

Model Name	Device	CPU, 2 threads	GPU
Mobilenet_1.0_224(float)	iPhone XS	14.8 ms	3.4 ms
Mobilenet_1.0_224 (quant)	iPhone XS	11 ms	---

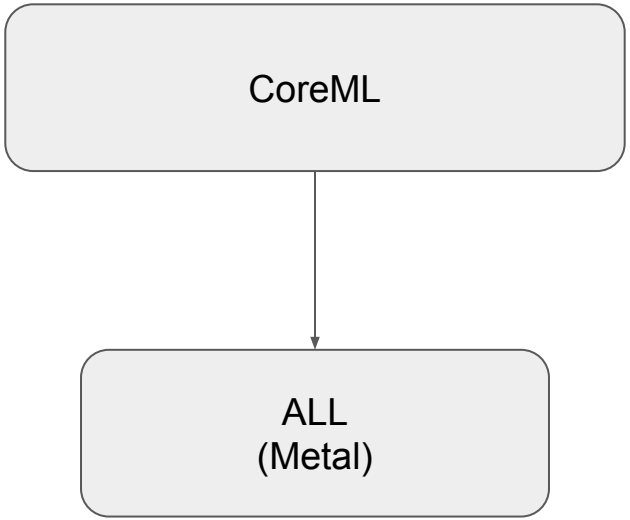
PyTorch Mobile



PyTorch Mobile



CoreML



CoreML vs TensorFlow-Lite

MobileNet V2

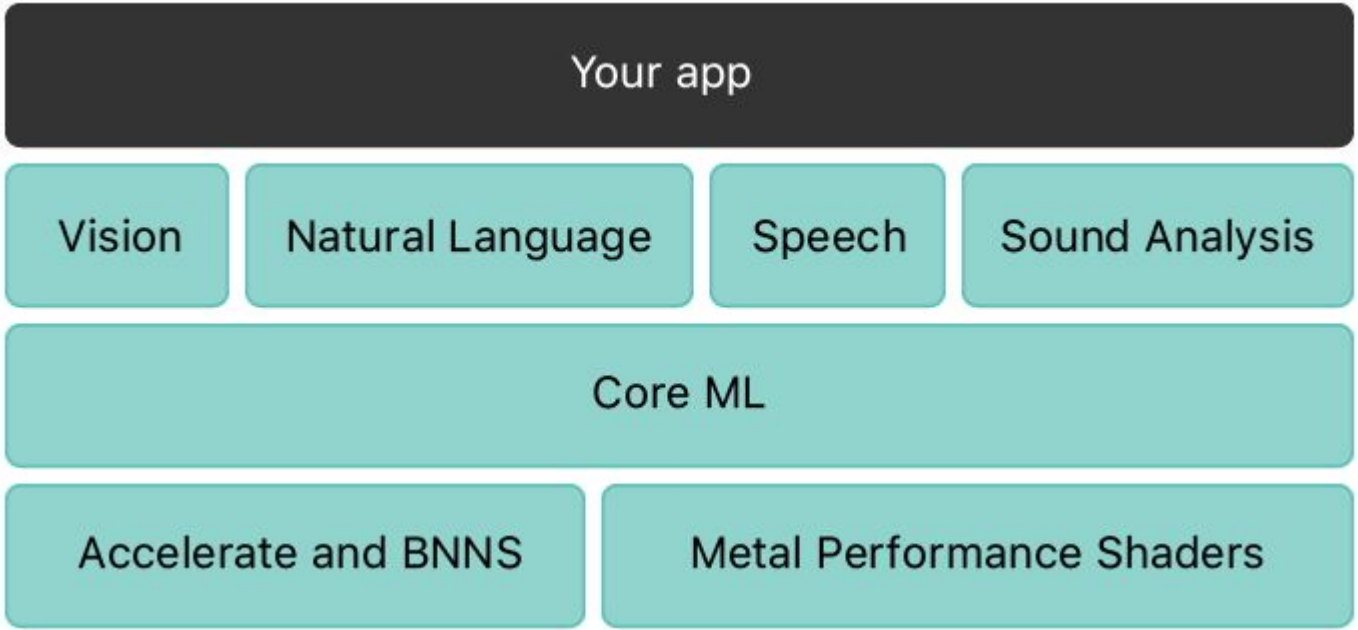
Phone	CPU (ms)	GPU (ms)	Core ML delegate (ms)
iPhone 8+	11.03	8.74	23.91 (GPU)
iPhone XS	9.97	4.55	7.26 (NPU)
iPhone 11 Pro	8.86	5.35	5.84 (NPU)

CoreML vs TensorFlow-Lite

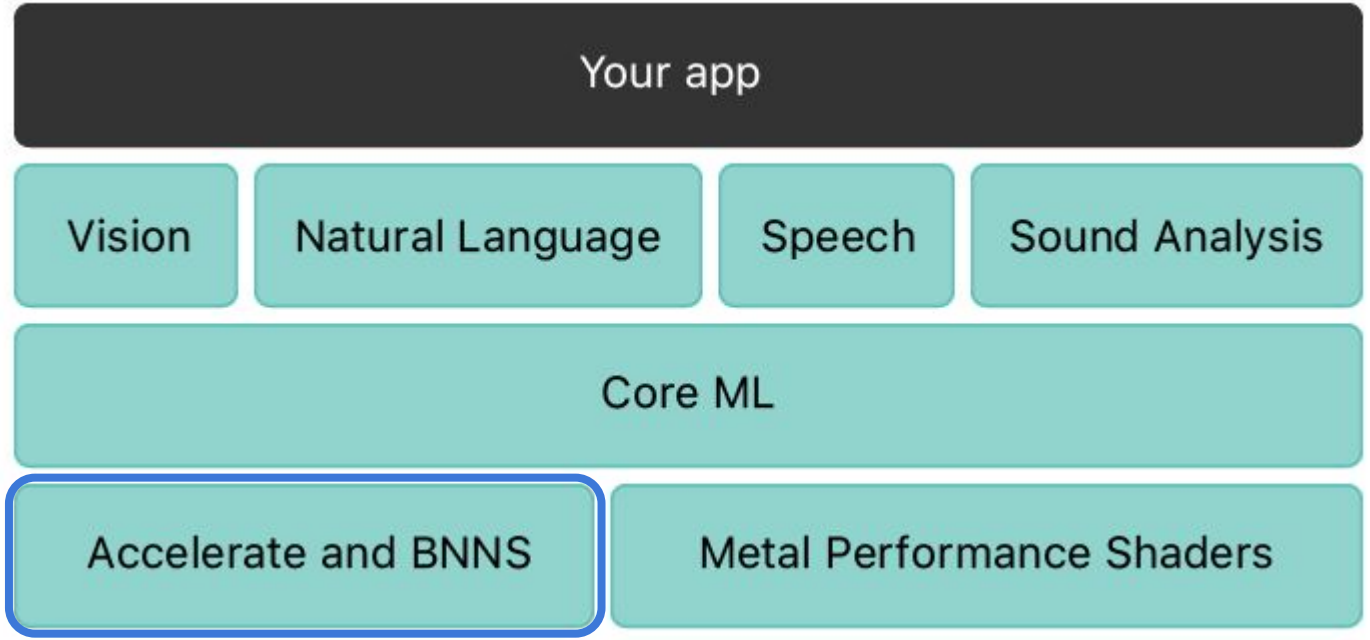
Inception V3

Phone	CPU (ms)	GPU (ms)	Core ML delegate (ms)
iPhone 8+	179.28	40.97	46.71 (GPU)
iPhone XS	150.30	30.01	14.89 (NPU)
iPhone 11 Pro	130.94	21.27	12.10 (NPU)

CoreML


Your app
Vision
Natural Language
Speech
Sound Analysis
Core ML
Accelerate and BNNS
Metal Performance Shaders

CoreML

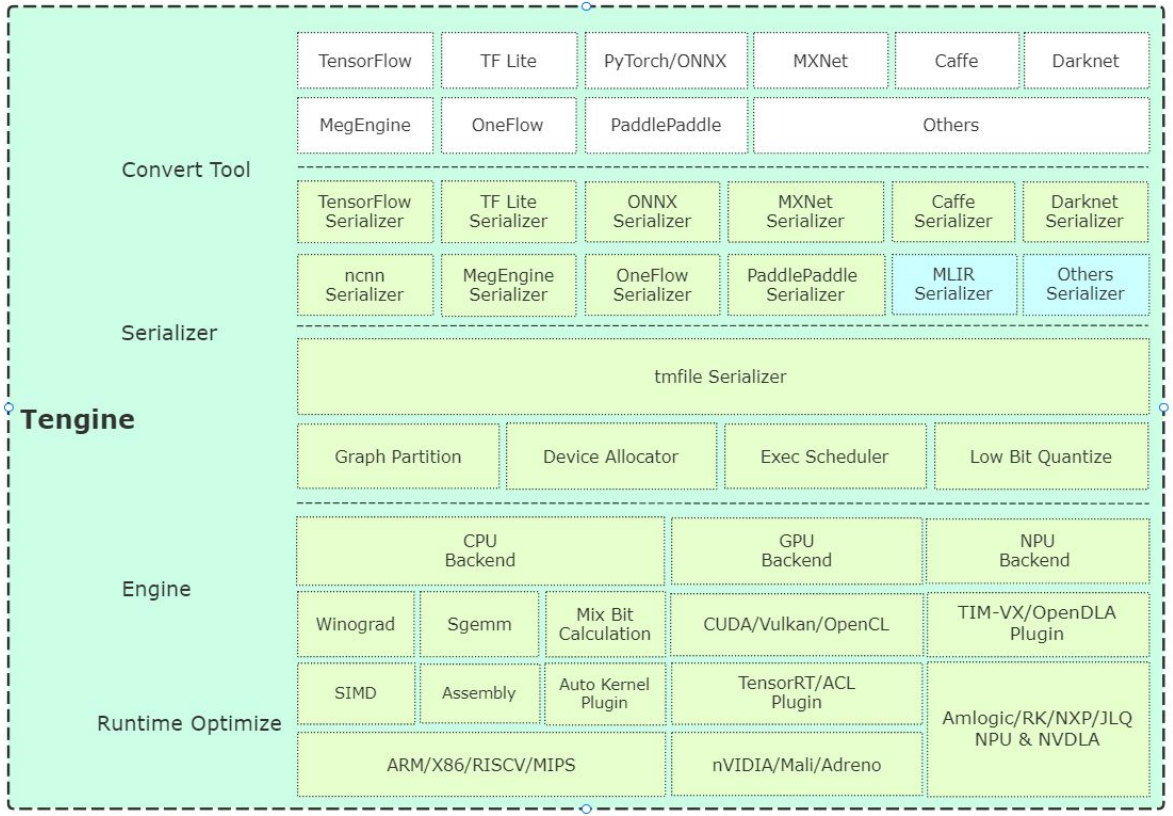

Your app
Vision
Natural Language
Speech
Sound Analysis
Core ML
Accelerate and BNNS
Metal Performance Shaders

Swift-AI

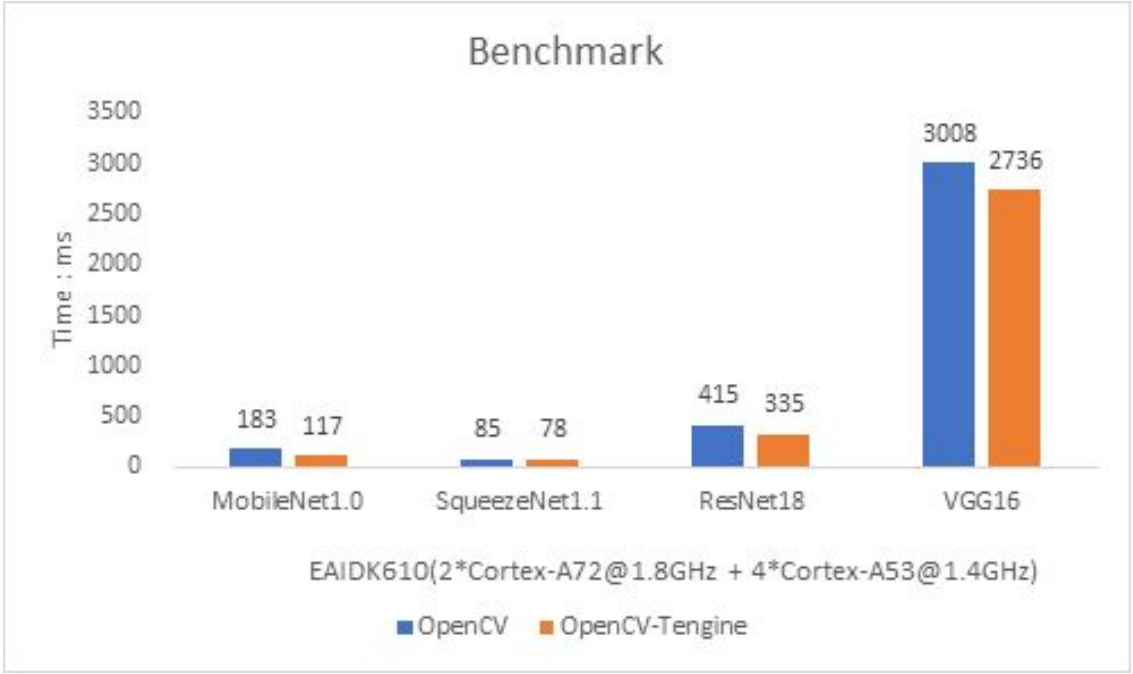
Swift A.I.

A.I. Library for Swift

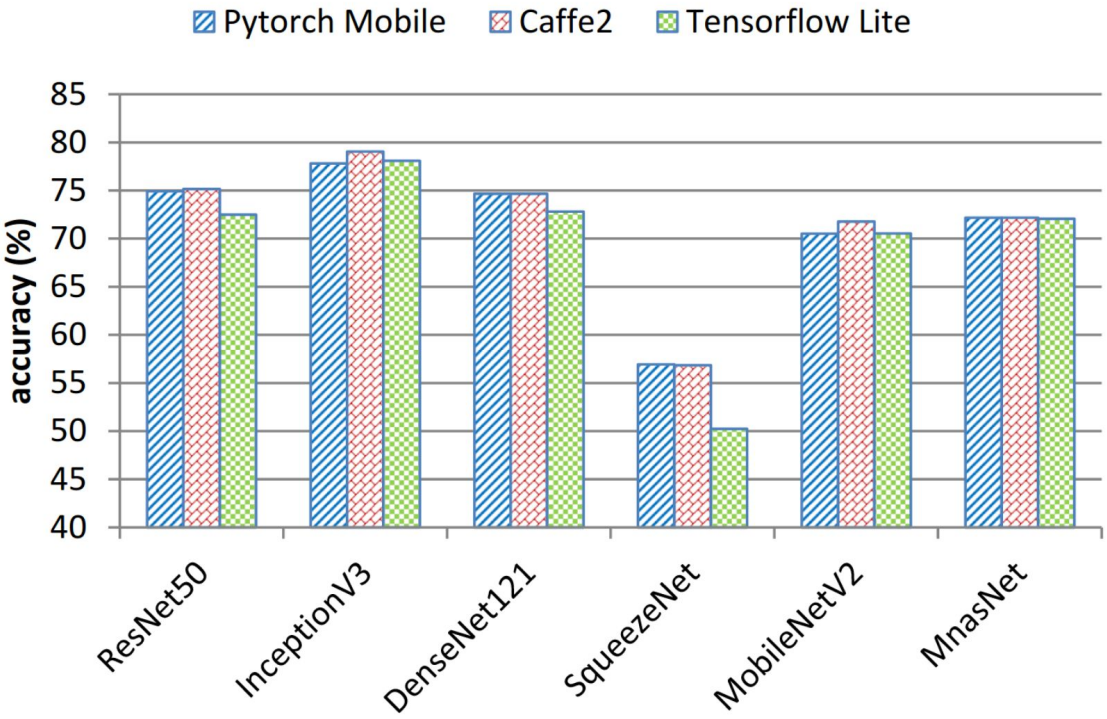
Tengine


Tengine
Convert Tool
TensorFlow
TF Lite
PyTorch/ONNX
MXNet
Caffe
Darknet
MegEngine
OneFlow
PaddlePaddle
Others
Serializer
TensorFlow Serializer
TF Lite Serializer
ONNX Serializer
MXNet Serializer
Caffe Serializer
Darknet Serializer
ncnn Serializer
MegEngine Serializer
OneFlow Serializer
PaddlePaddle Serializer
MLIR Serializer
Others Serializer
tmfile Serializer
Graph Partition
Device Allocator
Exec Scheduler
Low Bit Quantize
Engine
CPU Backend
GPU Backend
NPU Backend
Winograd
Sgemm
Mix Bit Calculation
CUDA/Vulkan/OpenCL
TIM-VX/OpenDLA Plugin
Runtime Optimize
SIMD
Assembly
Auto Kernel Plugin
TensorRT/ACL Plugin
Amlogic/RK/NXP/JLQ NPU & NVDLA
ARM/X86/RISCV/MIPS
nVIDIA/Mali/Adreno

Tengine


Benchmark
Time : ms
3500
3000
2500
2000
1500
1000
500
0
183
117
85
78
415
335
3008
2736
MobileNet1.0
SqueezeNet1.1
ResNet18
VGG16
EAIDK610(2*Cortex-A72@1.8GHz + 4*Cortex-A53@1.4GHz)
OpenCV
OpenCV-Tengine

Проблема падения точности модели



Статья: Comparison and Benchmarking of AI Models and Frameworks on Mobile Devices

Примеры внедрения нейронных сетей

Лекция 3

Искусственный интеллект:
математические модели и прикладные решения
2022

Структура лекции

- Примеры решений с реализацией экспорта моделей под мобильные устройства
- Примеры готовых реализаций проектов под Android и iOS

YOLOv5


YOLOv5 by
Download on the App Store
Coming Soon on Google Play

YOLOv5

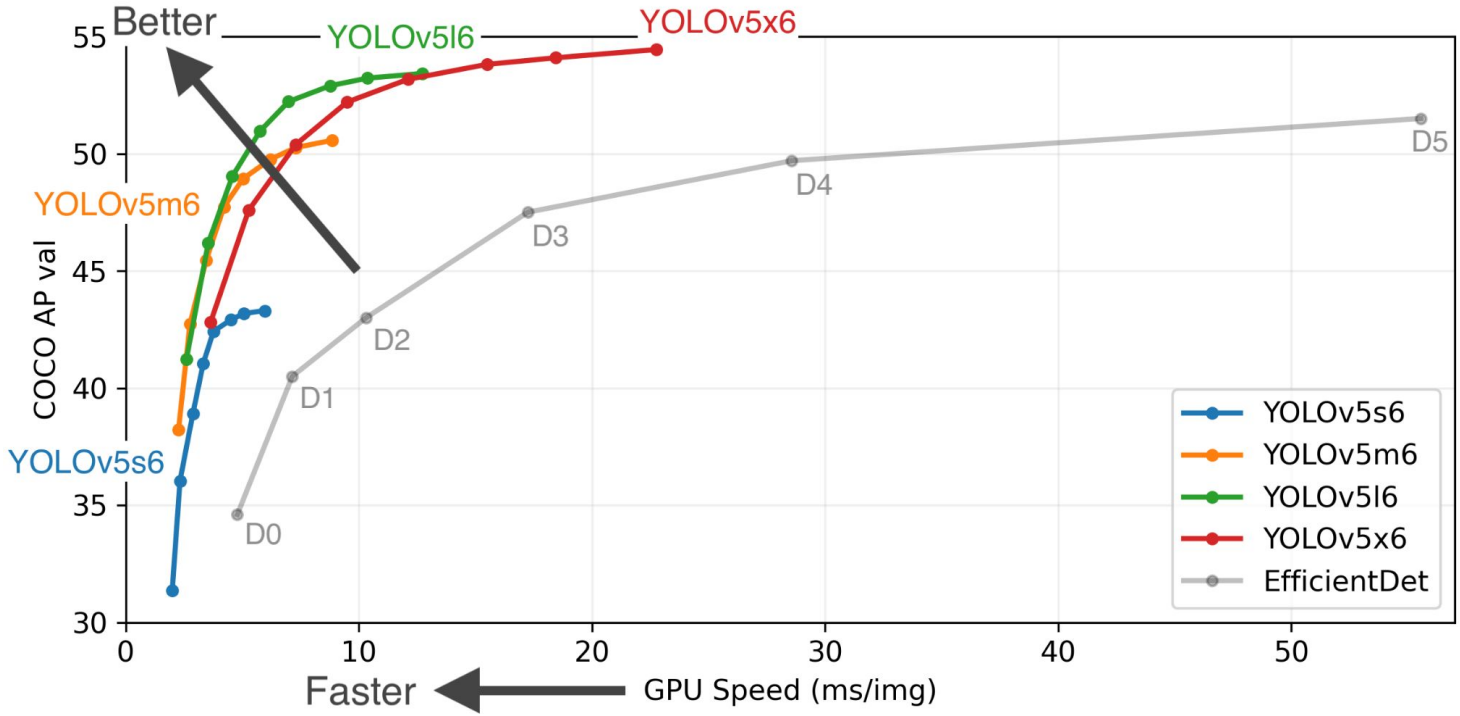
Решаемая задача: нахождение (детекция) объектов

Исходный фреймворк: PyTorch

Поддерживаемый экспорт:

- TorchScript
- ONNX
- CoreML
- TensorRT

YOLOv5


Better
YOLOv5l6
YOLOv5x6
YOLOv5m6
YOLOv5s6
D0
D1
D2
D3
D4
D5
COCO AP val
55
50
45
40
35
30
0
10
20
30
40
50
Faster
GPU Speed (ms/img)
YOLOv5s6
YOLOv5m6
YOLOv5l6
YOLOv5x6
EfficientDet

YOLOX

YOLOX

Решаемая задача: нахождение (детекция) объектов

Исходный фреймворк: PyTorch / MegEngine

Поддерживаемый экспорт:

- ONNX
- TensorRT
- MegEngine -> Tengine
- OpenVINO

D2Go

Решаемые задачи моделей из пула:

- Нахождение (детекция) объектов
- Сегментация объектов
- Нахождение ключевых точек

Исходный фреймворк: PyTorch + Detectron2

Поддерживаемый экспорт:

- TorchScript

D2Go: Зоопарк моделей

- Сегментация:
  - Faster-RCNN-FBNetV3A
  - Faster-RCNN-FBNetV3A-dsmask
  - Faster-RCNN-FBNetV3G-FPN
- Детекция:
  - Mask-RCNN-FBNetV3A
  - Mask-RCNN-FBNetV3A-dsmask
  - Mask-RCNN-FBNetV3G-FPN
- Нахождение ключевых точек
  - Keypoint-RCNN-FBNetV3A-dsmask

Готовые реализации TF-Lite: Android + NLP

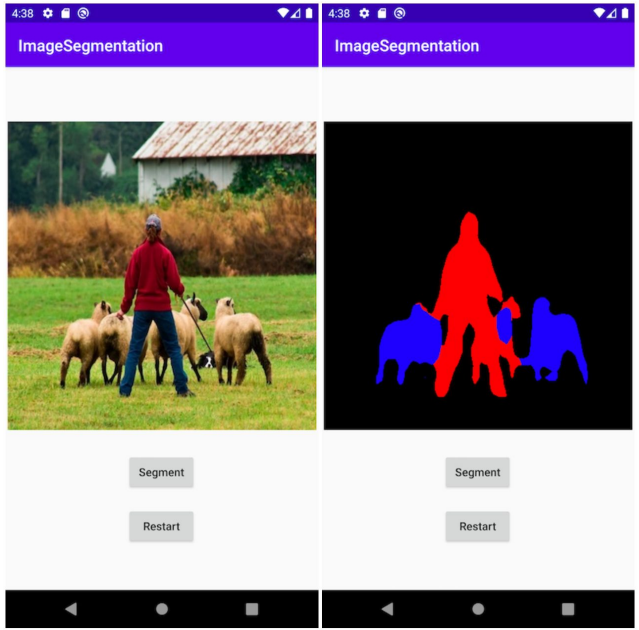
- BERT
- DistilBERT
- GPT-2
- DistilGPT-2

Готовые реализации TF-Lite: iOS + NLP

- BERT
- DistilBERT
- GPT-2
- DistilGPT-2

Готовые реализации PyTorch: Android + CV

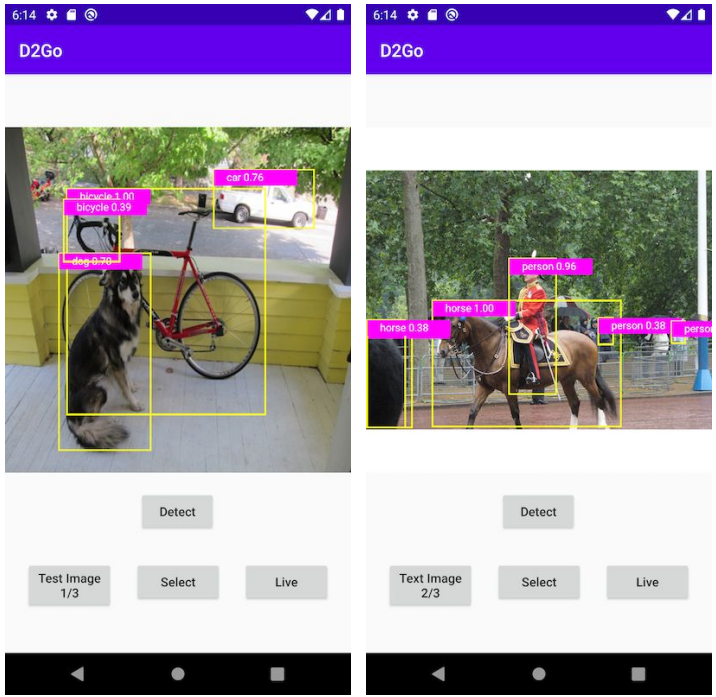
- DeepLabV3



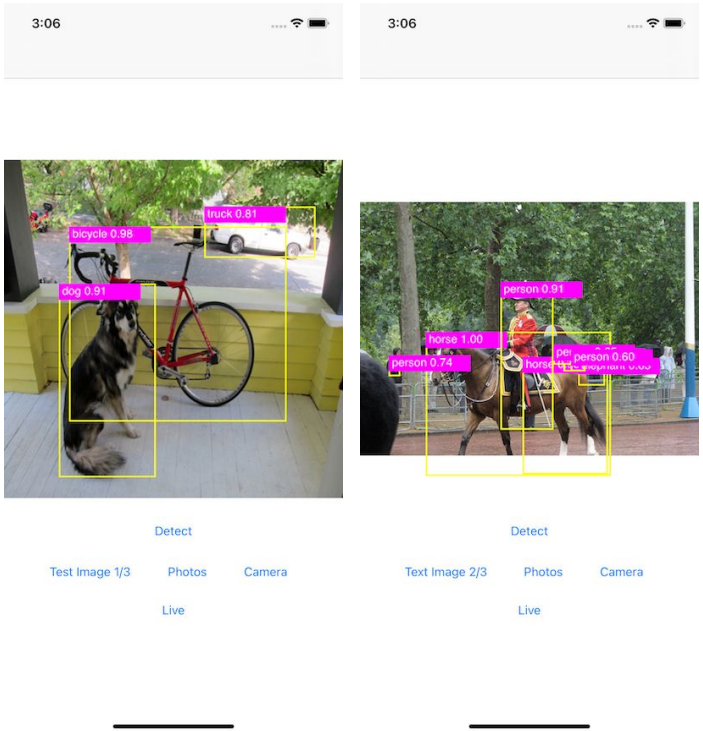
Готовые реализации PyTorch: iOS + CV

- DeepLabV3

Готовые реализации PyTorch: Android + D2Go



Готовые реализации PyTorch: iOS + D2Go



Готовые реализации PyTorch: (Android, iOS) + Sound

- Wav2Vec + Classification

Android

iOS

Готовые реализации TF-Lite

Платформы:

- Android
- iOS
- Raspberry Pi

Готовые реализации: Nvidia Jetson

Решаемые задачи моделями из пула:

- Image classification
- Object detection
- Semantic segmentation
- Pose estimation (keypoints)
- Mono Depth

Готовые реализации: Nvidia Jetson + DeepStream

Готовые реализации: Nvidia Jetson + DeepStream

Решаемые задачи моделями из пула:

- Object detection
- Semantic segmentation

Зоопарк моделей:

- Faster-RCNN
- YoloV3/4
- SSD/DSSD
- RetinaNet
- PeopleSegNet
- UNET

Нейронные сети и web

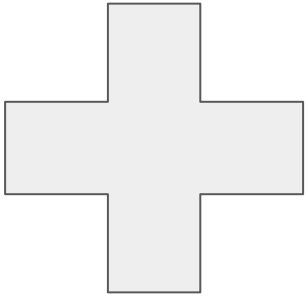
Лекция 4

Искусственный интеллект:
математические модели и прикладные решения
2022

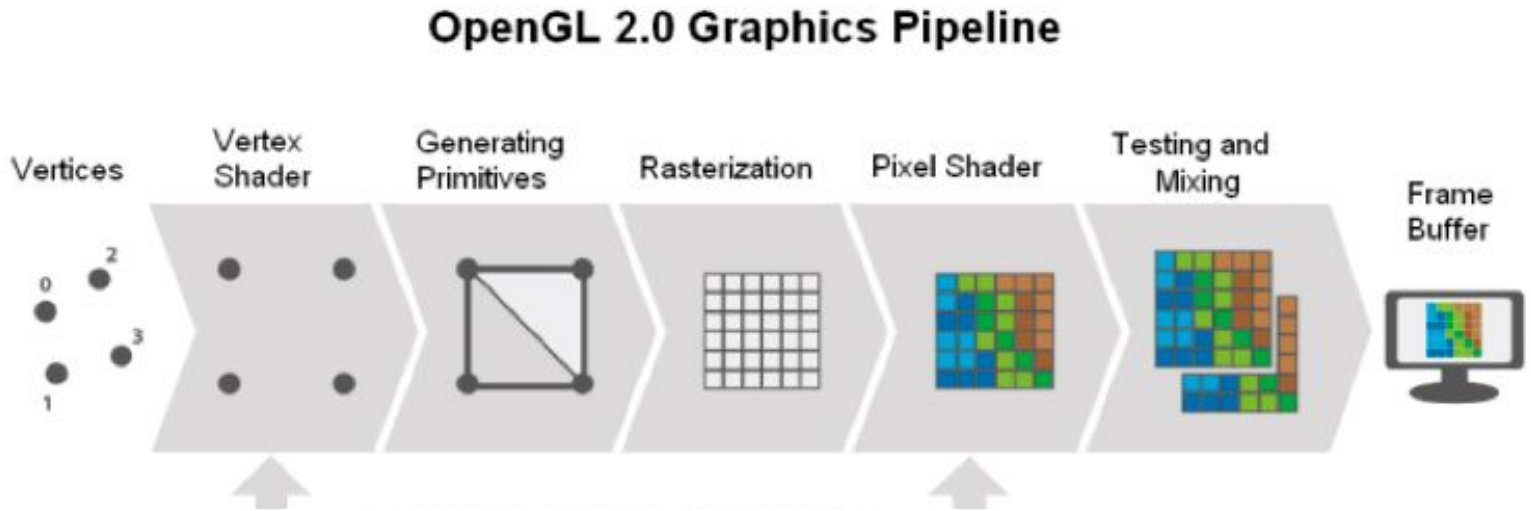
Структура лекции

- Технологии интеграции нейронных сетей в web
- Фреймворки запуска нейронных сетей для web

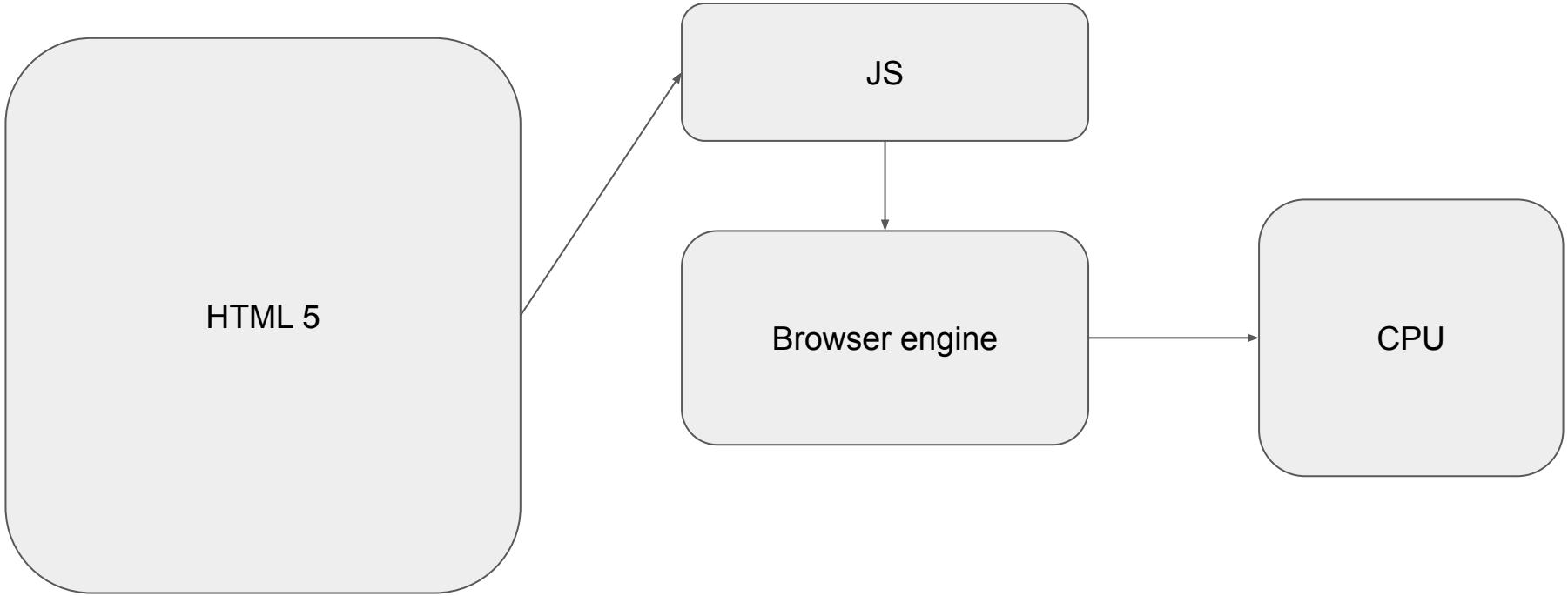
Технологии интеграции нейронных сетей в web



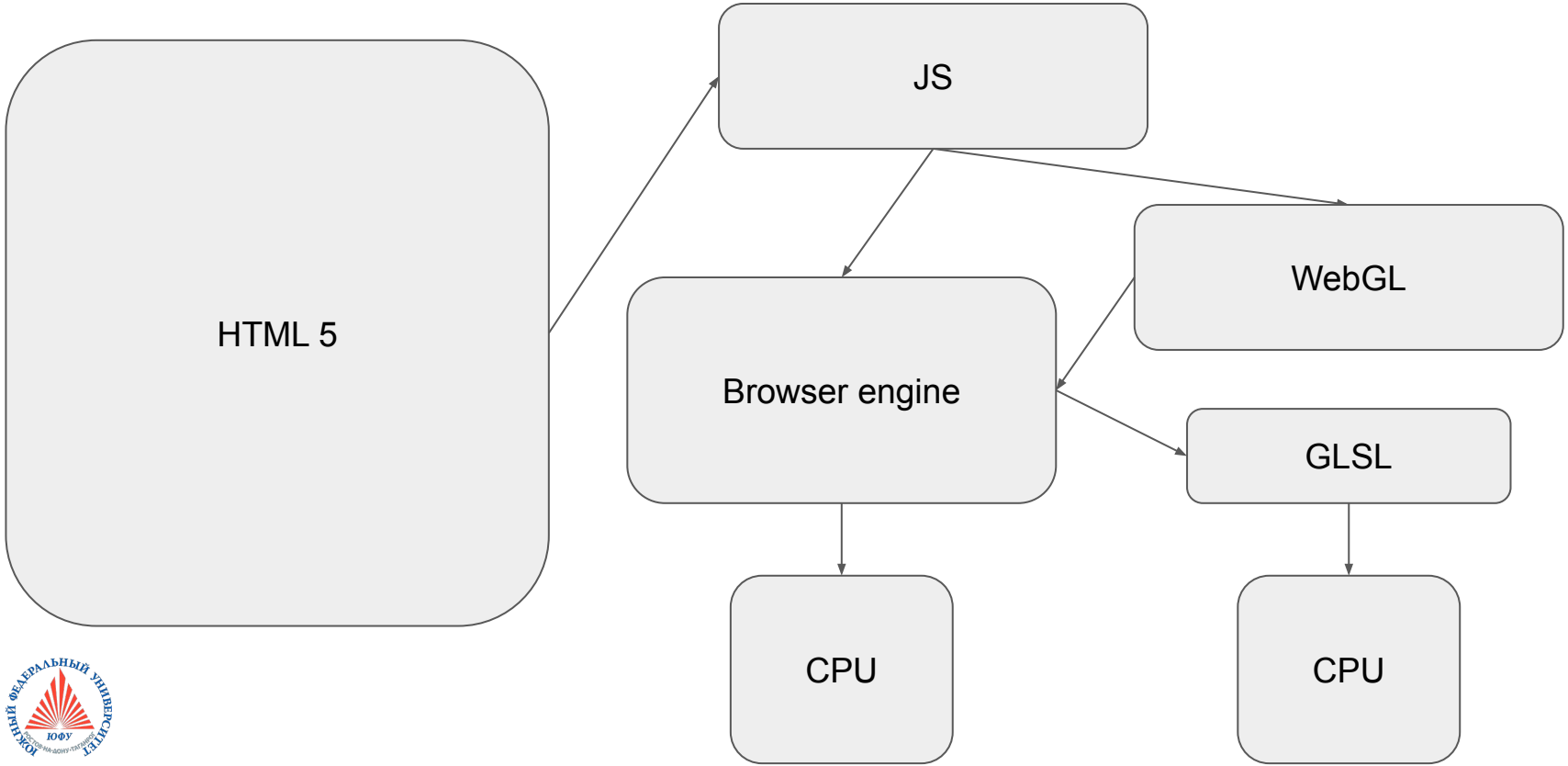
Технология OpenGL



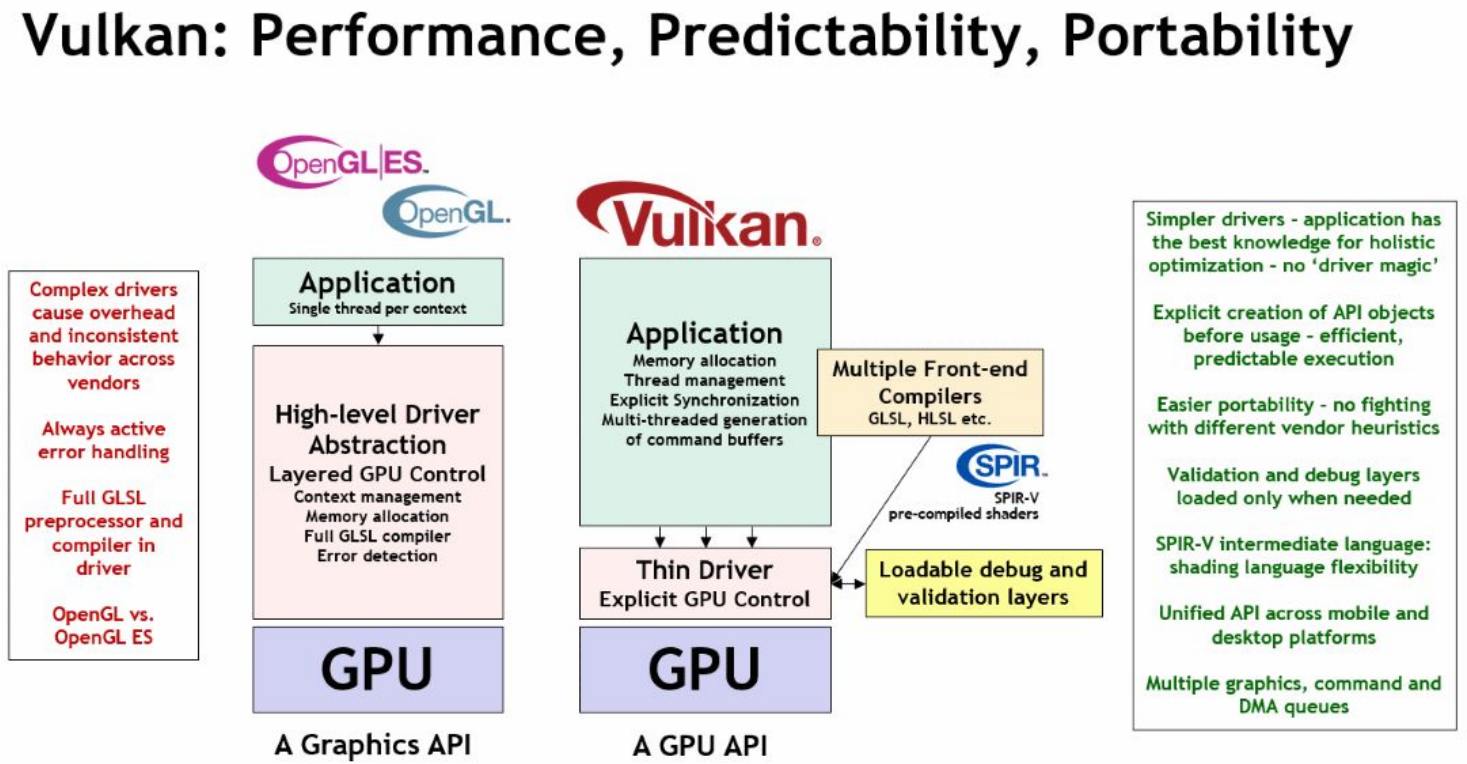
Запуск моделей без WebGL



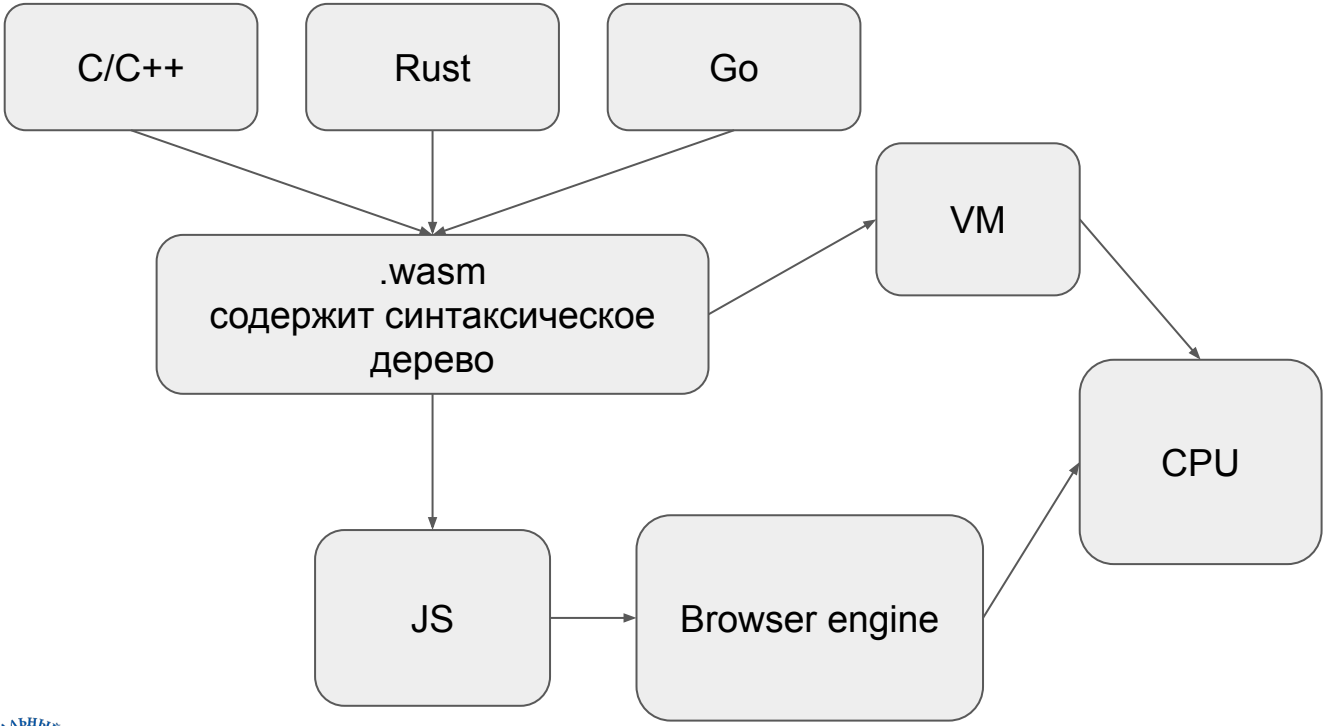
Технология WebGL



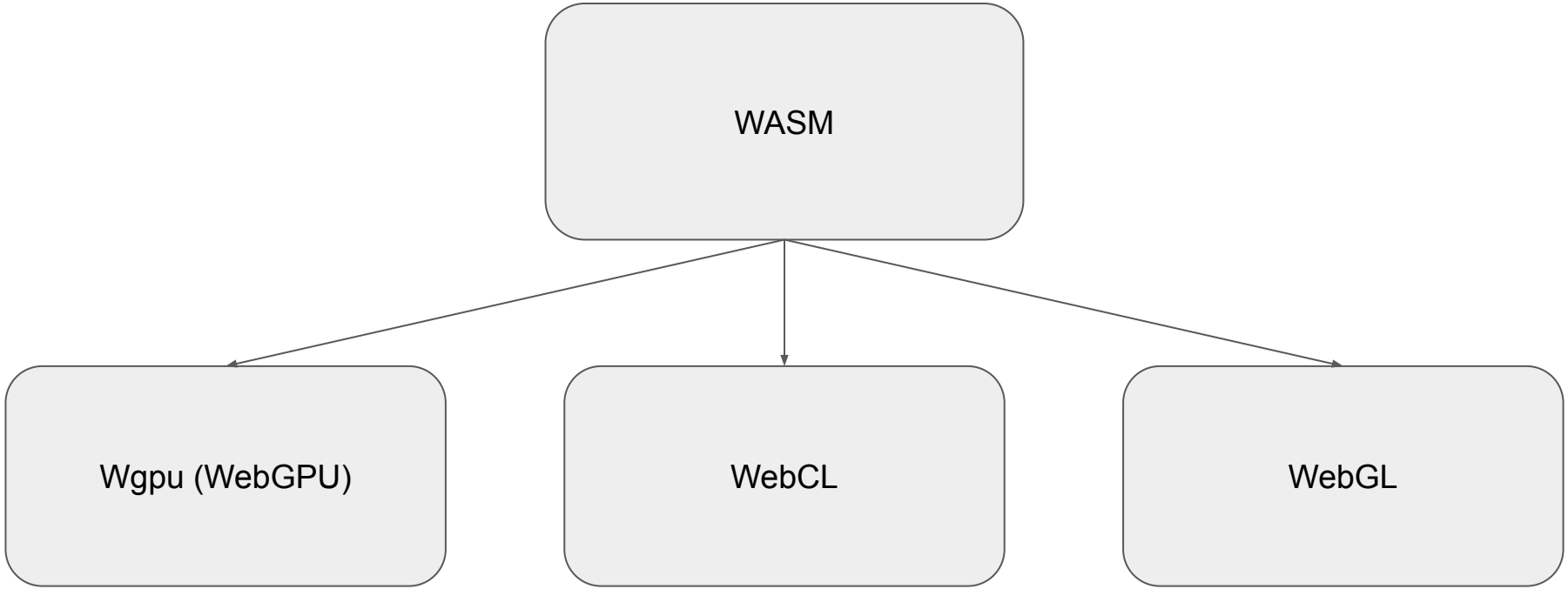
Vulkan


Vulkan: Performance, Predictability, Portability
OpenGL ES
OpenGL
Complex drivers cause overhead and inconsistent behavior across vendors
Always active error handling
Full GLSL preprocessor and compiler in driver
OpenGL vs. OpenGL ES
Application
Single thread per context
High-level Driver Abstraction
Layered GPU Control
Context management
Memory allocation
Full GLSL compiler
Error detection
GPU
A Graphics API
Vulkan
Application
Memory allocation
Thread management
Explicit Synchronization
Multi-threaded generation of command buffers
Multiple Front-end Compilers
GLSL, HLSL etc.
SPIR
SPIR-V pre-compiled shaders
Thin Driver
Explicit GPU Control
Loadable debug and validation layers
GPU
A GPU API
Simpler drivers - application has the best knowledge for holistic optimization - no 'driver magic'
Explicit creation of API objects before usage - efficient, predictable execution
Easier portability - no fighting with different vendor heuristics
Validation and debug layers loaded only when needed
SPIR-V intermediate language: shading language flexibility
Unified API across mobile and desktop platforms
Multiple graphics, command and DMA queues

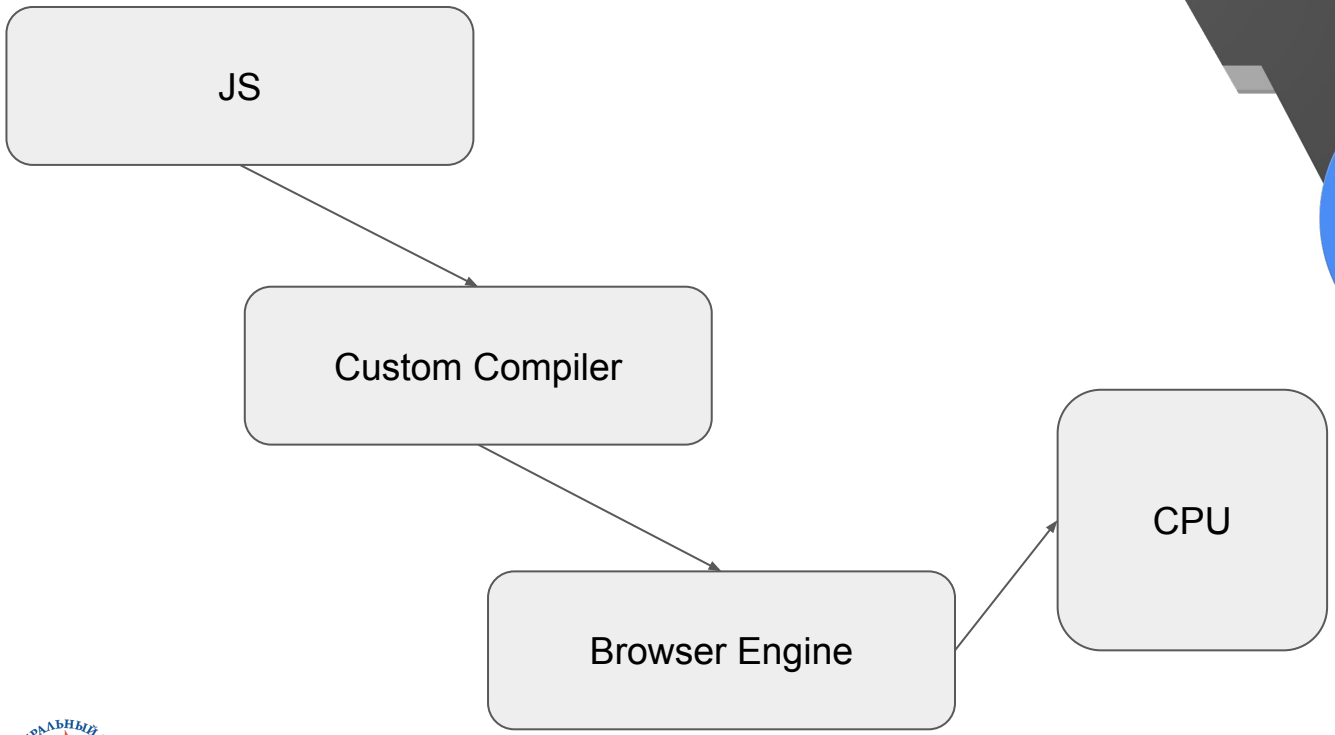
WebAssembly



WebAssembly + GPU



Chrome V8



TensorFlow.js

Поддерживаемые технологии:

- WebGL
- WASM
- WebGPU

Полезный функционал:

- AutoML

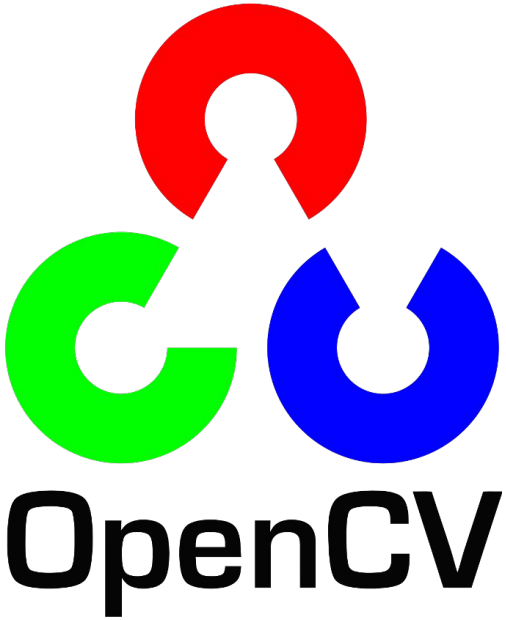
OpenCV.js

Поддерживаемые технологии:

- Emscripten (LLVM)
- W3C (WebAssembly)

Полезный функционал:

- Обширный функционал обработки изображений
- DNN модуль



Brain.js

Поддерживаемые технологии:

- W3C (WebAssembly)

Полезный функционал:

- Обучение нейронных сетей
  в браузере, как главное преимущество

WebDNN

Поддерживаемые технологии:

- W3C (WebAssembly)
- WebGPU
- WebGL

Keras.js

Keras.js

Поддерживаемые технологии:

- WebGL

Полезный функционал:

- Запуск моделей с фреймворков TensorFlow и CNTK

Ещё фреймворки

- Synaptic
- Neataptic
- Neuro.js
- mljs
- Mind
- MXnet.js
- Deepforge

Подготовка моделей к серверному использованию

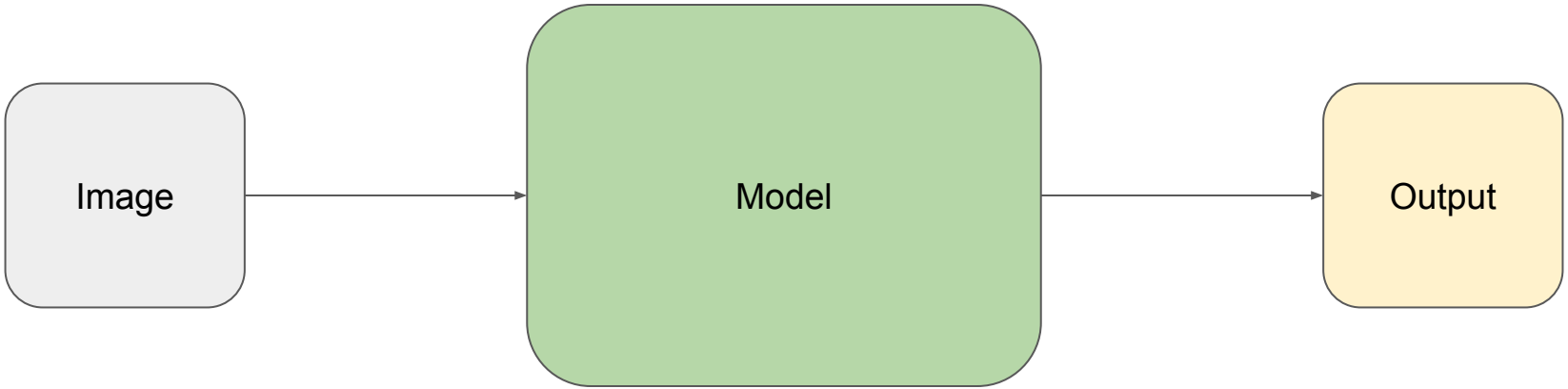
Лекция 5

Искусственный интеллект:
математические модели и прикладные решения
2022

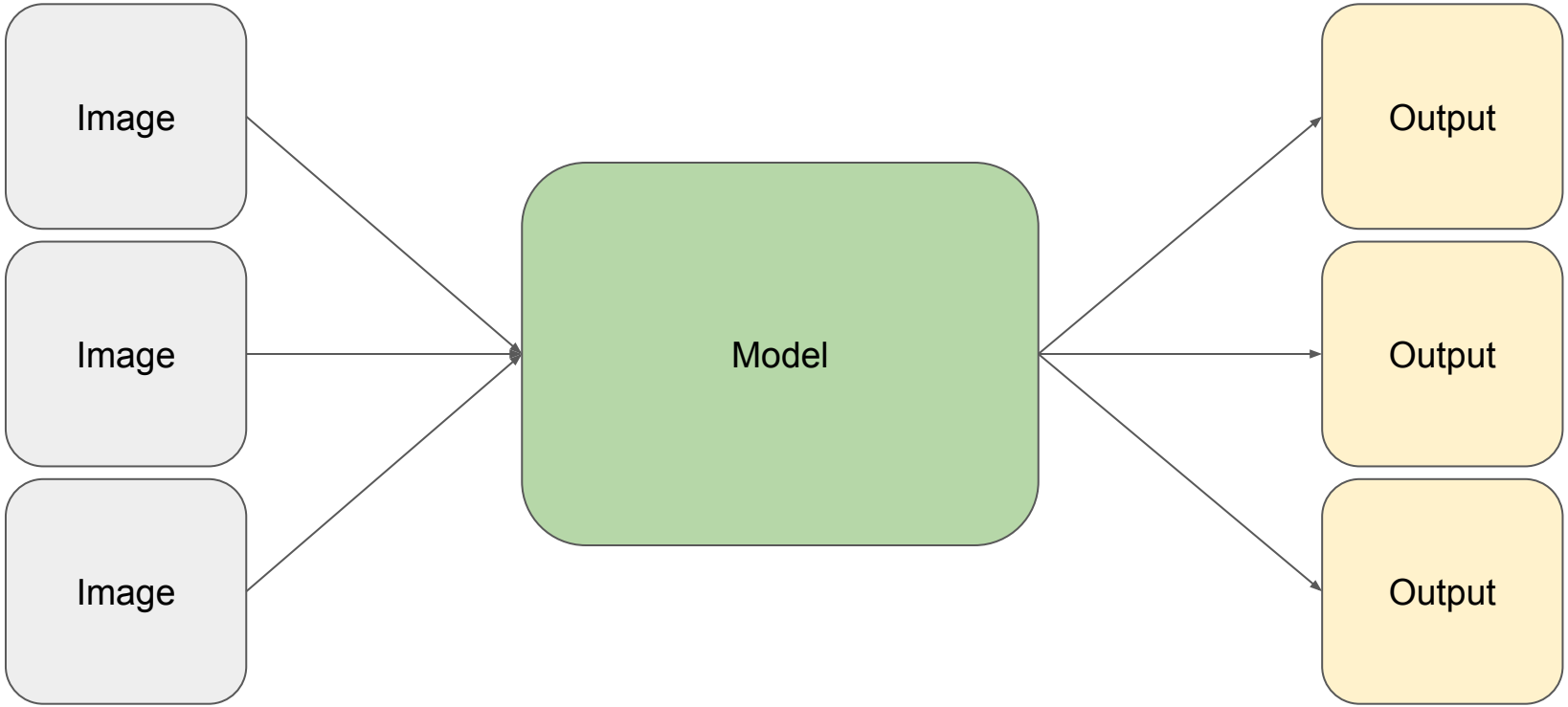
Структура лекции

- Основные проблемы интеграции моделей
- Готовые решения для интеграции
- Пример интеграции модели

Inference



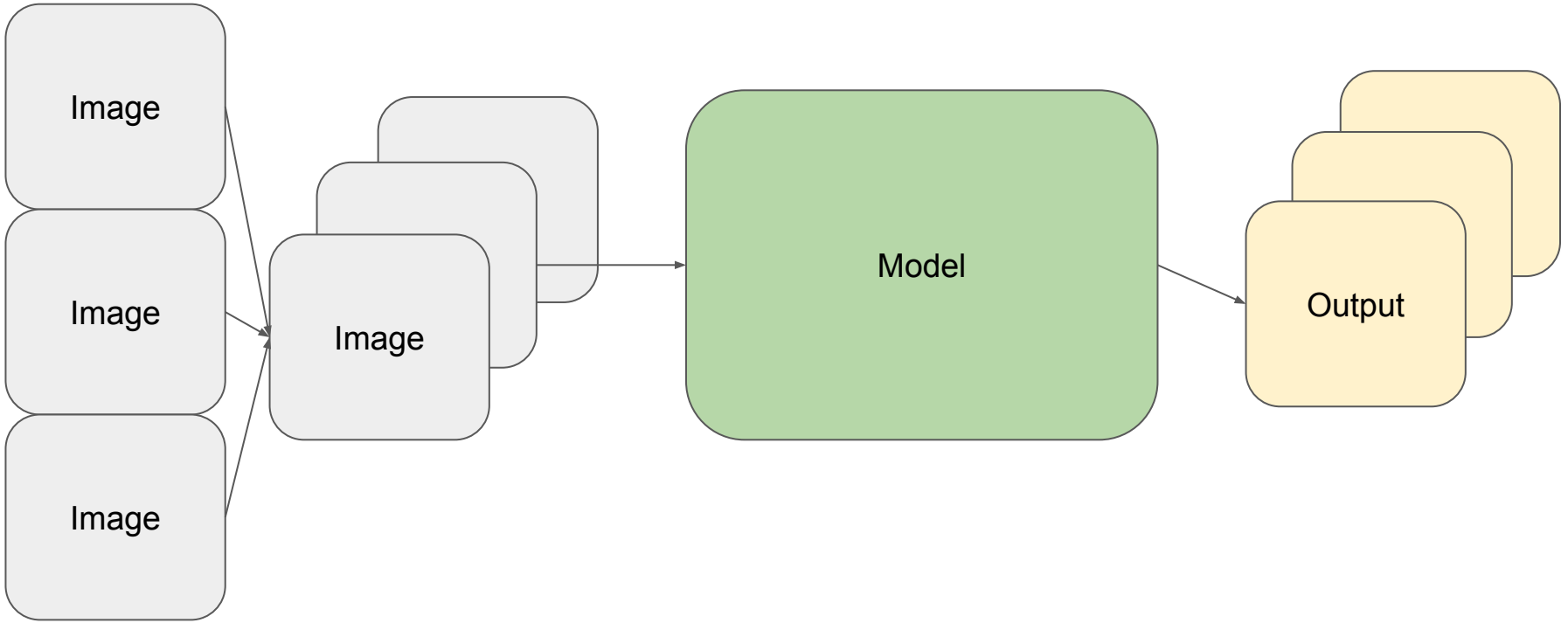
Inference



Inference on batch

```
img = Image.open(...).convert('RGB').resize((224, 224))

tensor = torchvision.transforms.ToTensor()(img)

out = model(tensor.unsqueeze(0)).detach()
```

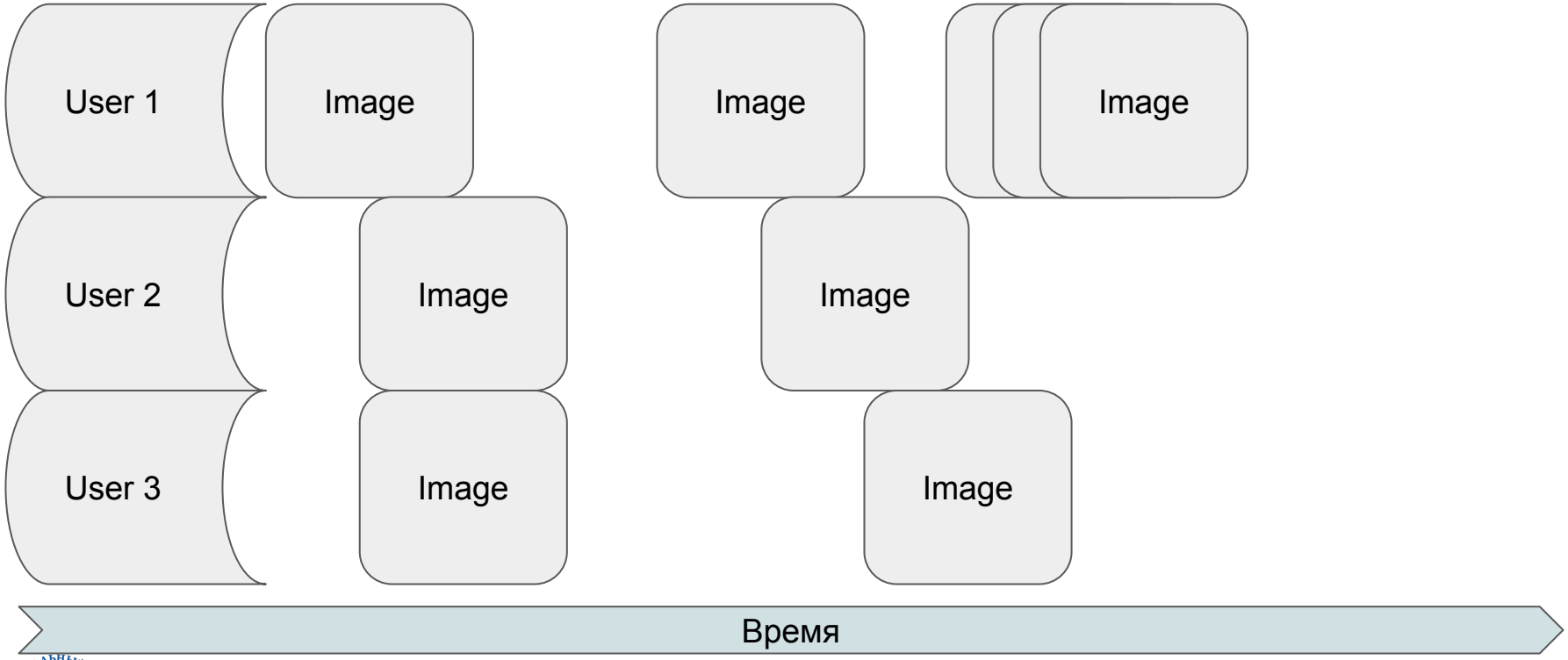
Inference on batch



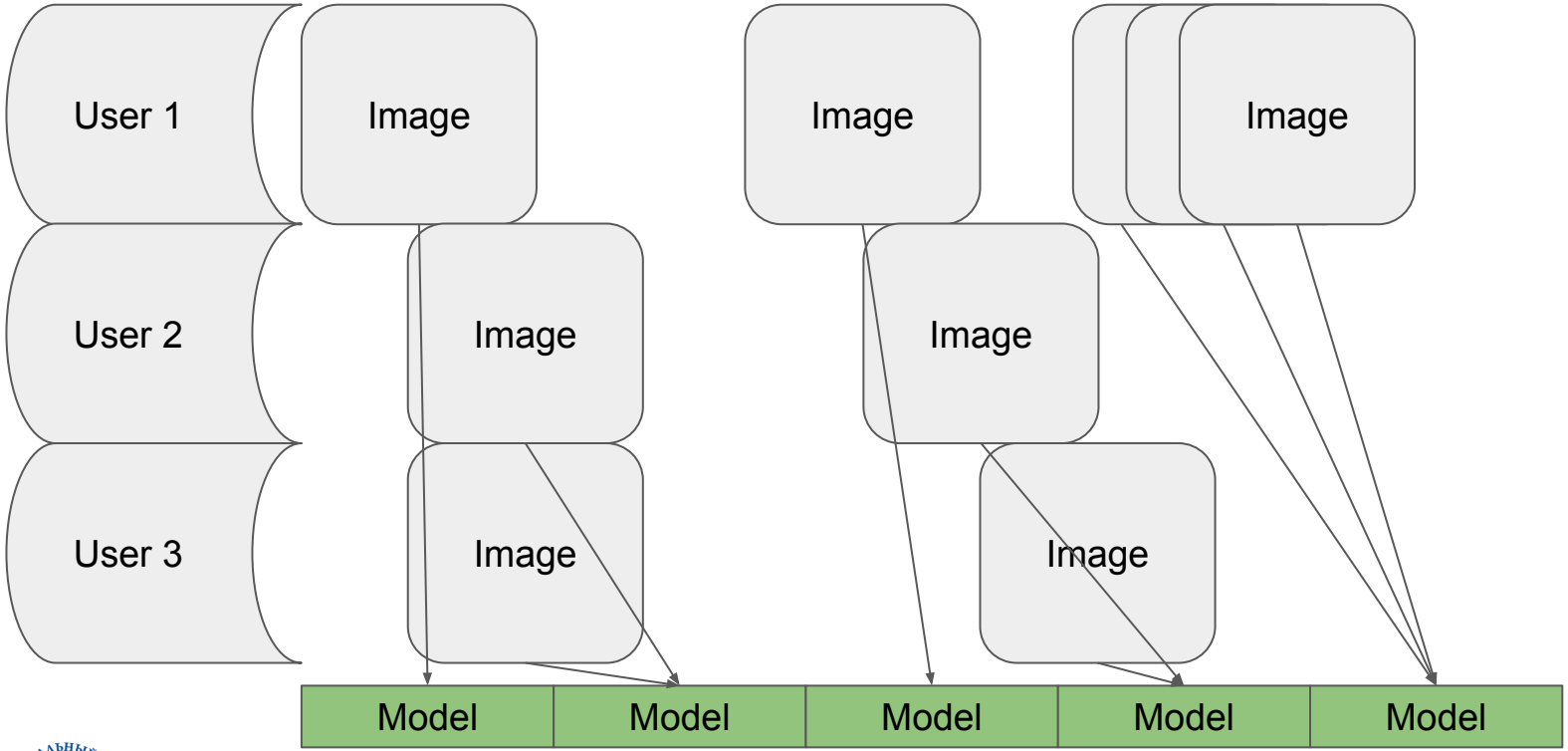
Inference on batch

```
images = [...]

tensors = [

    torchvision.transforms.ToTensor()(img)

    for img in images

]

input_tensor = torch.stack(tensors, dim=0)

out = model(input_tensor).detach()
```

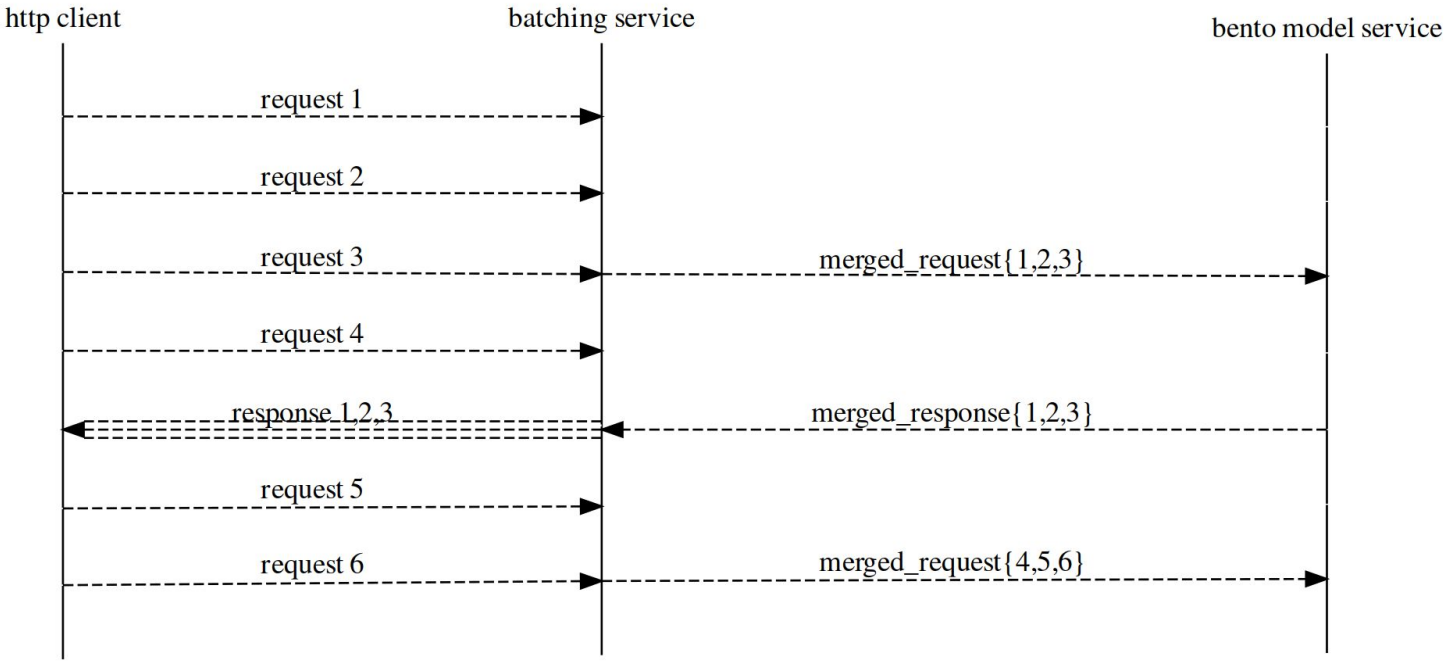
Проблема асинхронности запросов



Проблема асинхронности запросов

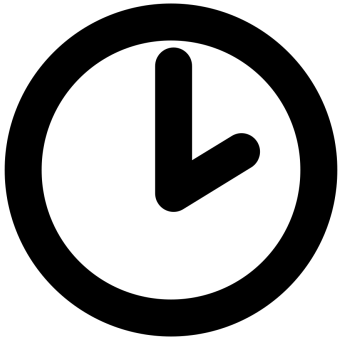


Проблема асинхронности запросов

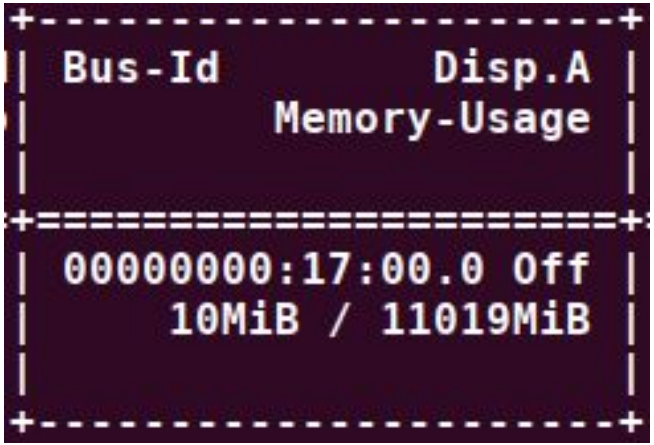


Проблема ограничения ресурсов

- Время обработки входного тензора



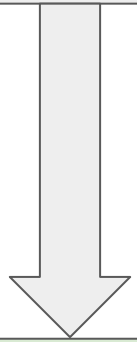
- Необходимая память
  для обработки входного тензора



Возможные оптимизации для решения проблем

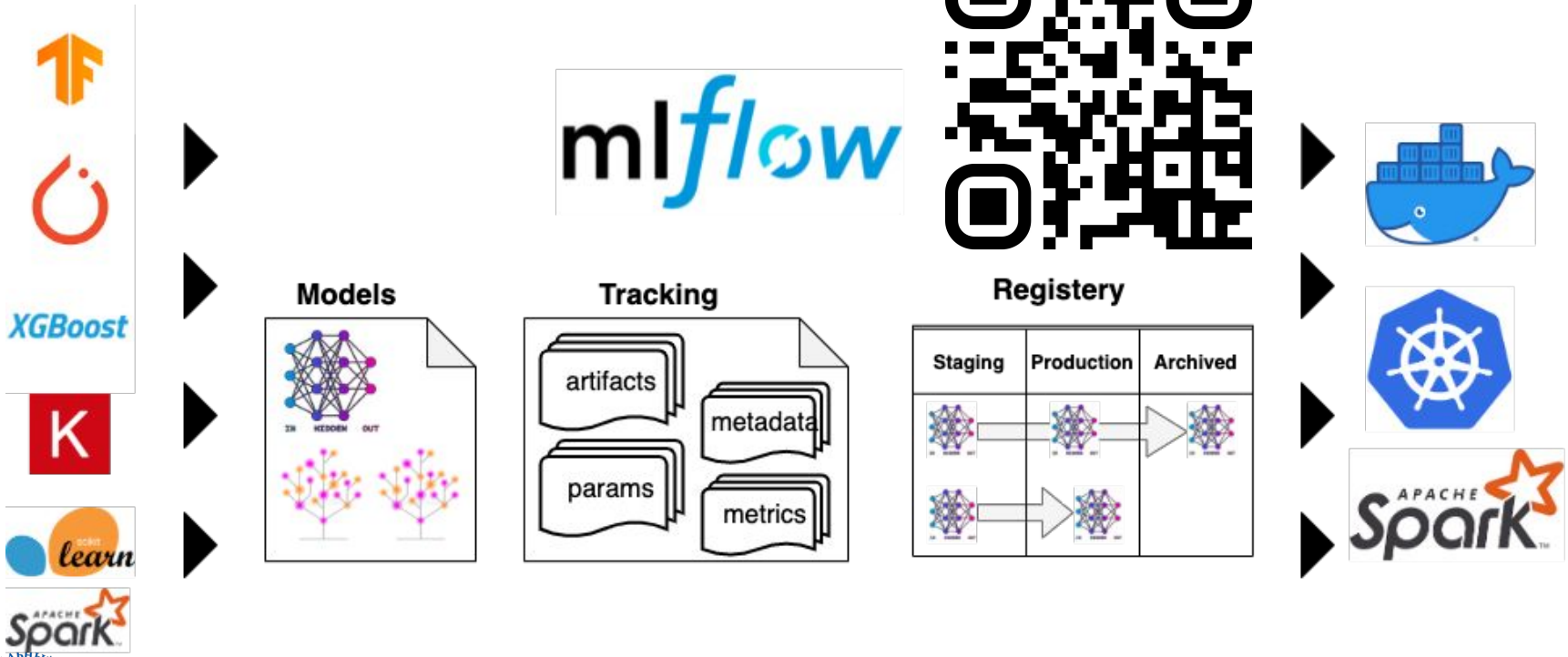
Ограничение по времени ожидания запроса пользователем

Ограничение по количеству занимаемой памяти при работе сети

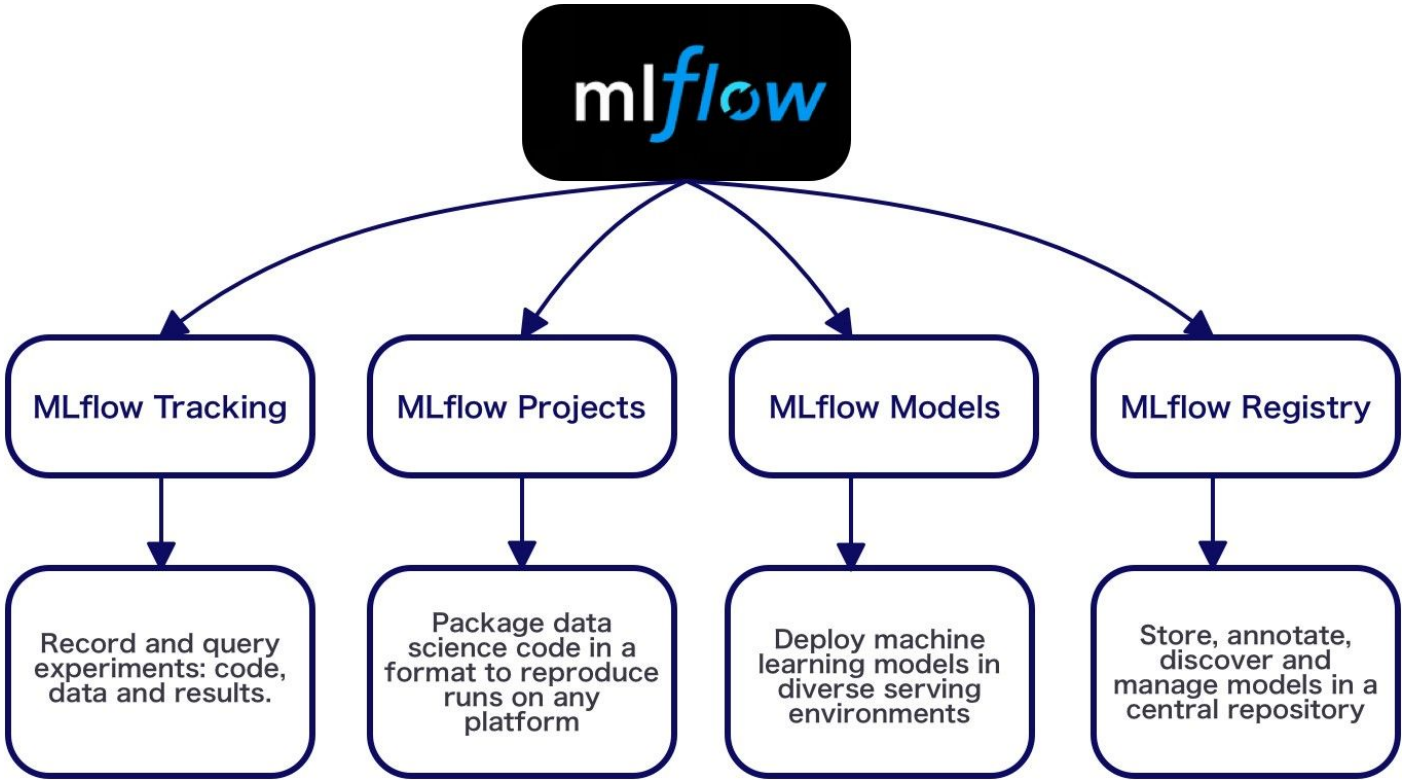


Ограничение размера входного тензора (batch)

MLflow


XGBoost
learn
Spark
mlflow
Models
IN HIDDEN OUT
Tracking
artifacts
metadata
params
metrics
Registery
Staging
Production
Archived
APACHE
Spark

MLflow

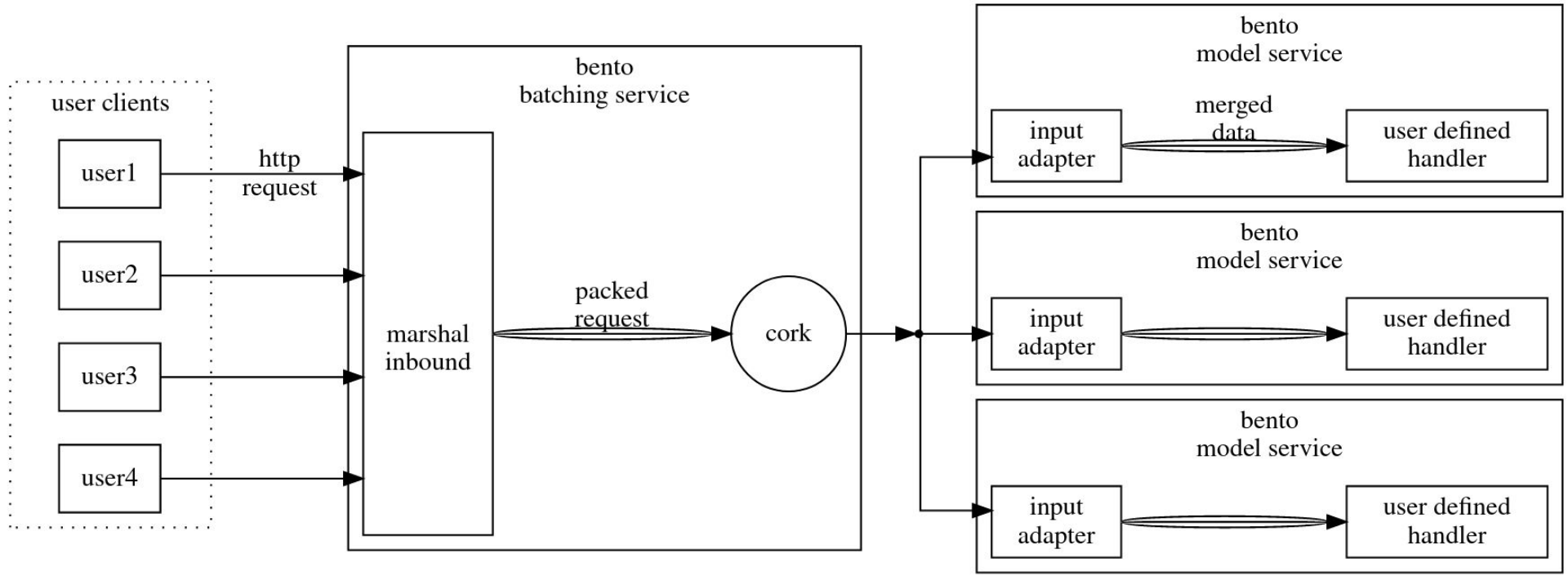

mlflow
MLflow Tracking
MLflow Projects
MLflow Models
MLflow Registry
Record and query experiments: code, data and results.
Package data science code in a format to reproduce runs on any platform
Deploy machine learning models in diverse serving environments
Store, annotate, discover and manage models in a central repository

BentoML

BentoML


KERAS
SCIKIT LEARN
TENSOR FLOW
PYTORCH
XGBOOST
BENTOML
SWAGGER
PROMETHEUS
REST API
DOCKER
KUBERNETES

BentoML


user clients
user1
user2
user3
user4
http
request
bento
batching service
marshal
inbound
packed
request
cork
bento
model service
input
adapter
merged
data
user defined
handler
bento
model service
input
adapter
user defined
handler
bento
model service
input
adapter
user defined
handler

BentoML: схема использования

BentoML: пример использования

```
if __name__ == '__main__':
    model = torchvision.models.resnet18(True)
    example_input = torch.rand(1, 3, 224, 224)
    traced = torch.jit.trace(model, example_input)


    bento_classifier = PyTorchImageNetClassifier()
    bento_classifier.pack('classifier', traced)
    saved_path = bento_classifier.save()
```

BentoML: пример использования

```
import bentoml

from bentoml.frameworks.pytorch import PytorchModelArtifact


@bentoml.env(pip_packages=['torch', 'numpy', 'torchvision'])

@bentoml.artifacts([PytorchModelArtifact('classifier')])

class PyTorchImageNetClassifier(bentoml.BentoService):
```

BentoML: пример использования

```
from typing import BinaryIO, List

from bentoml.adapters import FileInput, JsonOutput


@bentoml.api(input=FileInput(), output=JsonOutput(), batch=True)

def predict(self, file_streams: List[BinaryIO]) -> List[dict]:

    img_tensors = [self.transform(Image.open(fs).convert(mode='RGB')) for fs in file_streams]

    input_batch = torch.stack(img_tensors,dim=0)

    pred_batch = self.artifacts.classifier(input_batch).detach().argmax(dim=1).numpy().tolist()

    return [{'class_index': idx} for idx in pred_batch]
```

BentoML: пример использования

```
from torchvision import transforms


@bentoml.utils.cached_property
def transform(self):
    return transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor()
    ])
```

BentoML: пример использования

Запуск (bash / shell):

```
python3 main.py

bentoml serve PyTorchImageNetClassifier:latest
```

Использование (REST API):

```
curl -X POST "http://127.0.0.1:5000/predict" -F image=@img1.jpg
```

BentoML: пример использования

GitHub Gist

Обзор ориентированных на мобильное использование архитектур

Лекция 6

Искусственный интеллект:
математические модели и прикладные решения
2022

Структура лекции

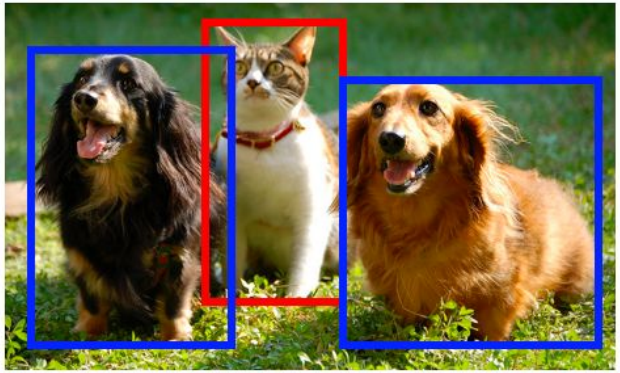
- Обзор архитектур для задачи классификации
- Обзор архитектур для задачи нахождения объектов (детекции)
- Обзор архитектур для задач сегментации (выделение масик и задачи выделения маски каждого объекта)
- Обзор архитектур для задач обработки естественных языков (NLP)
- Обзор архитектур для задач обработки звука

Задачи компьютерного зрения

Classification

CAT

Object Detection

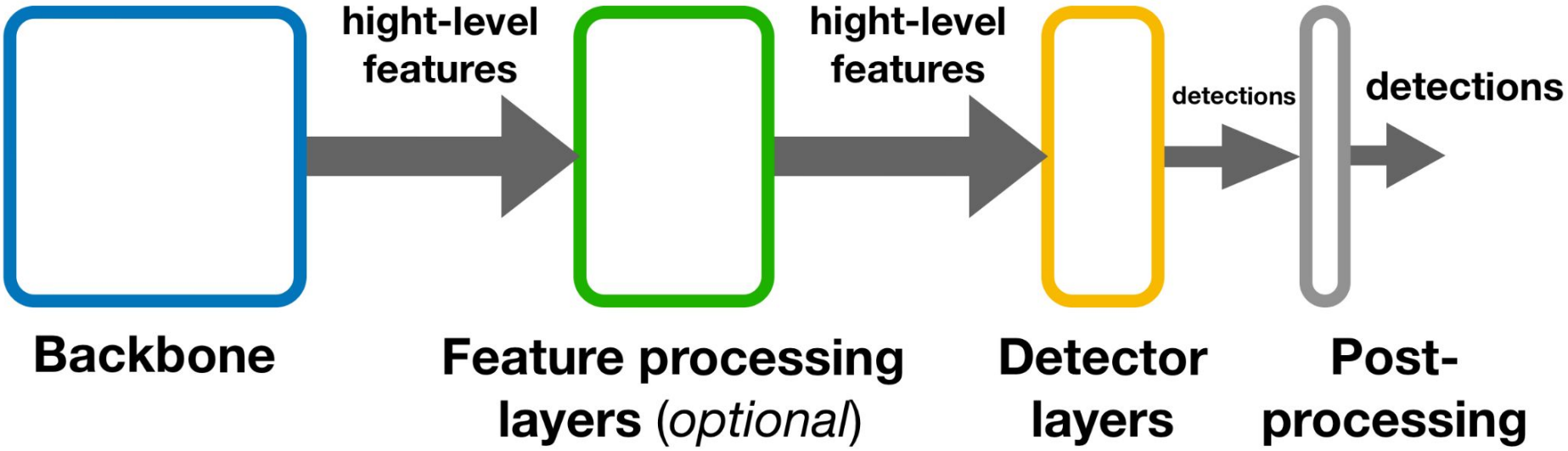


CAT, DOG

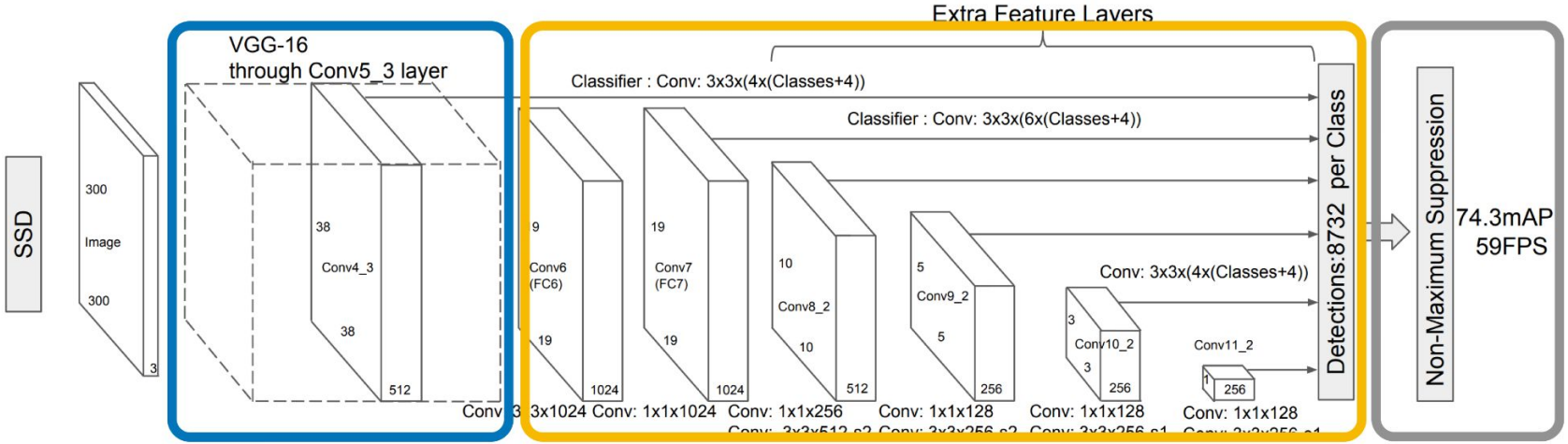
Instance Segmentation

CAT, DOG

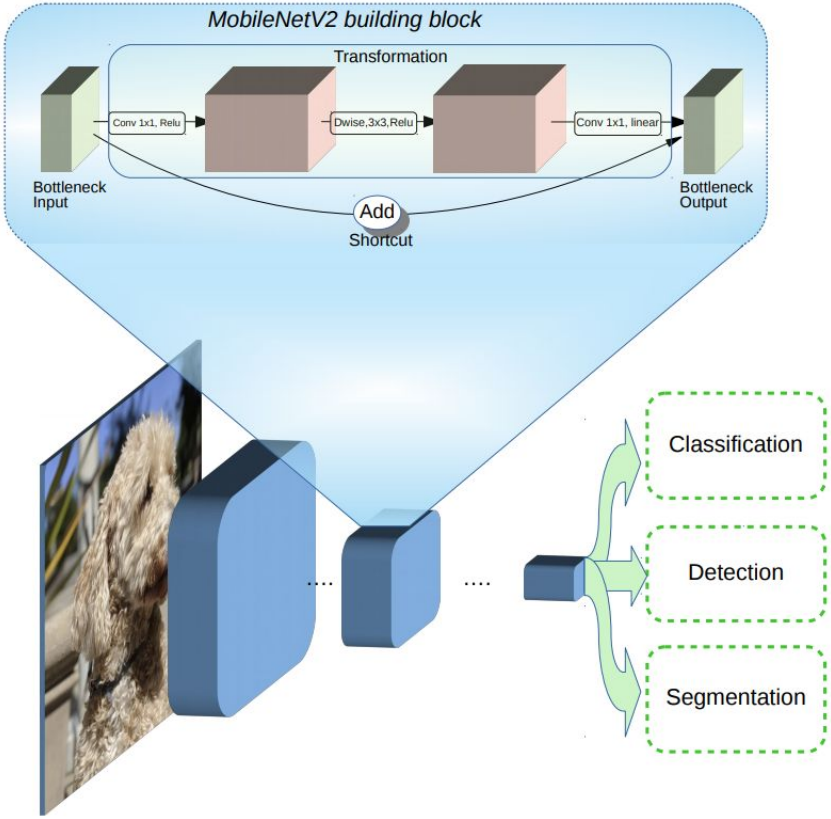
Общая схема нейросетевых архитектур



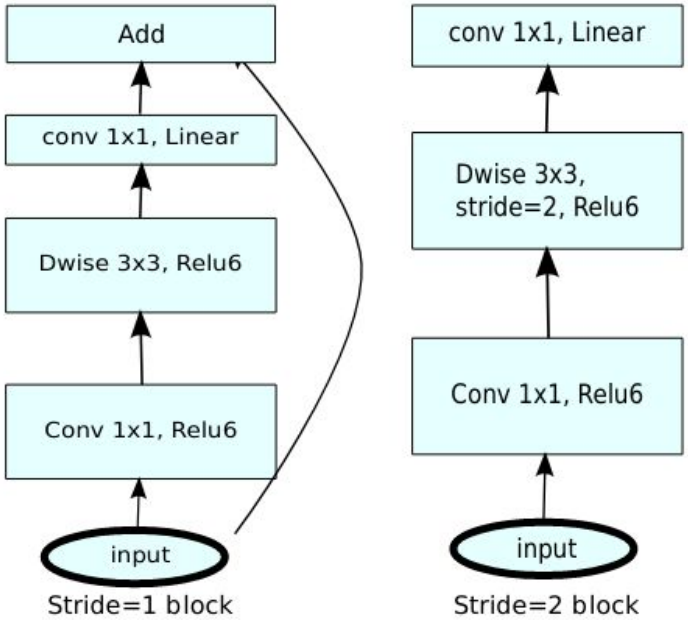
SSD: Single Short MultiBox Detector



Использование предобученных моделей



Классификация: MobileNetV2



Классификация: SqueezeNet

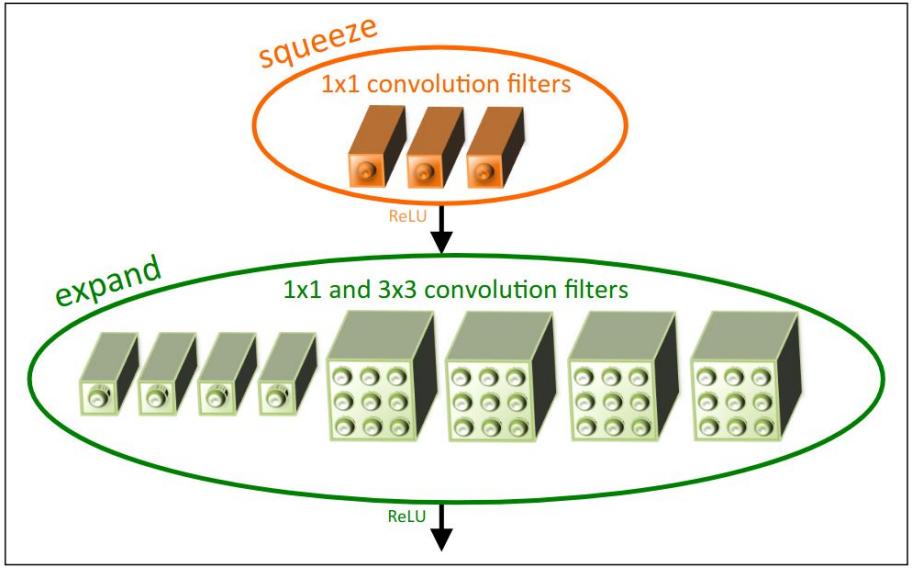
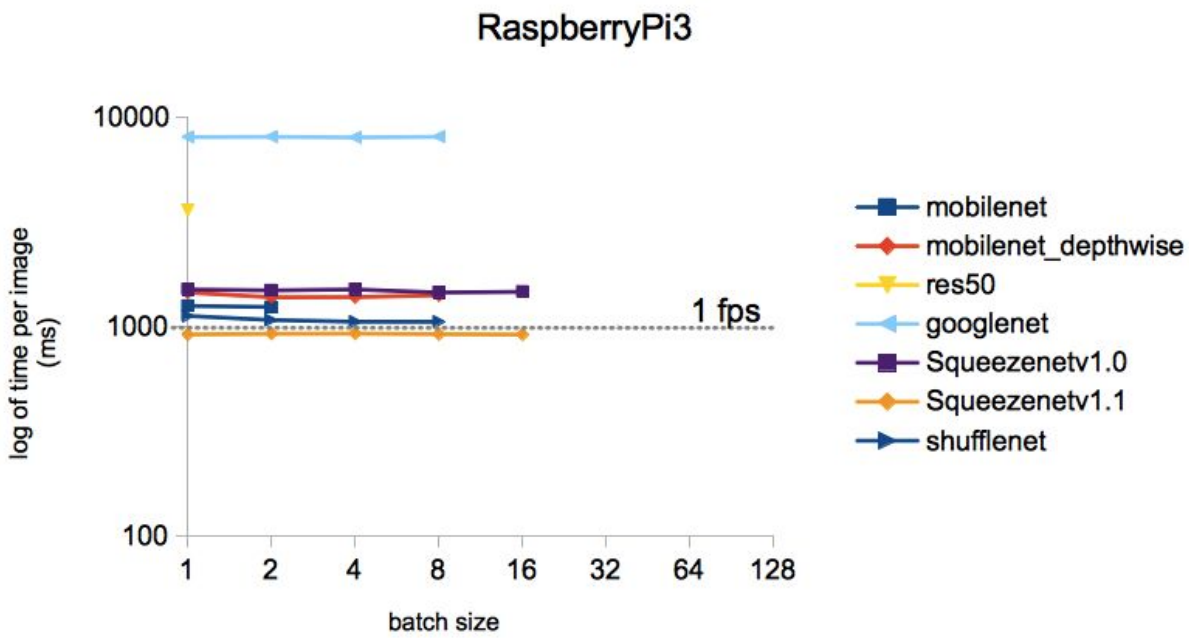
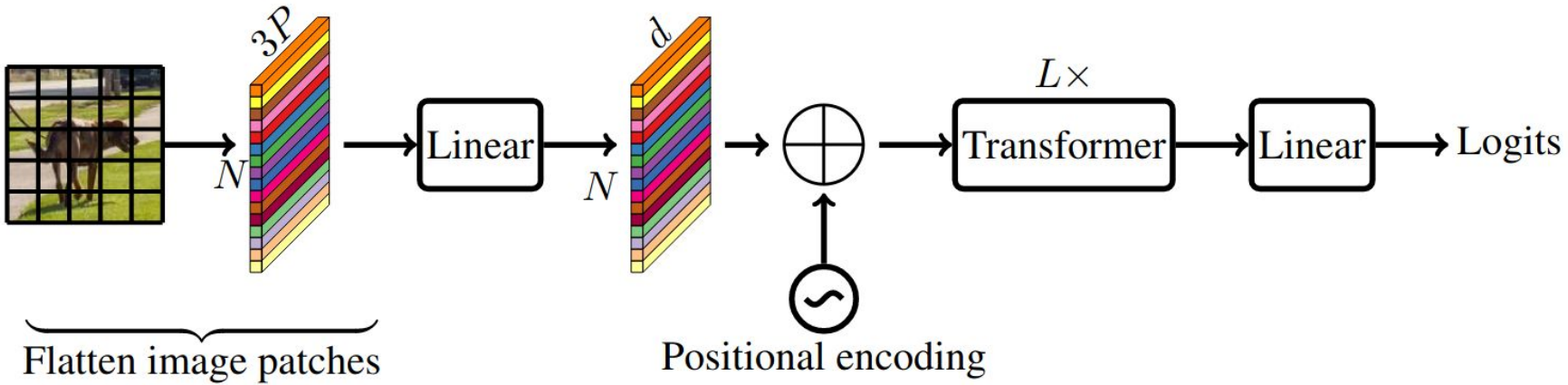


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

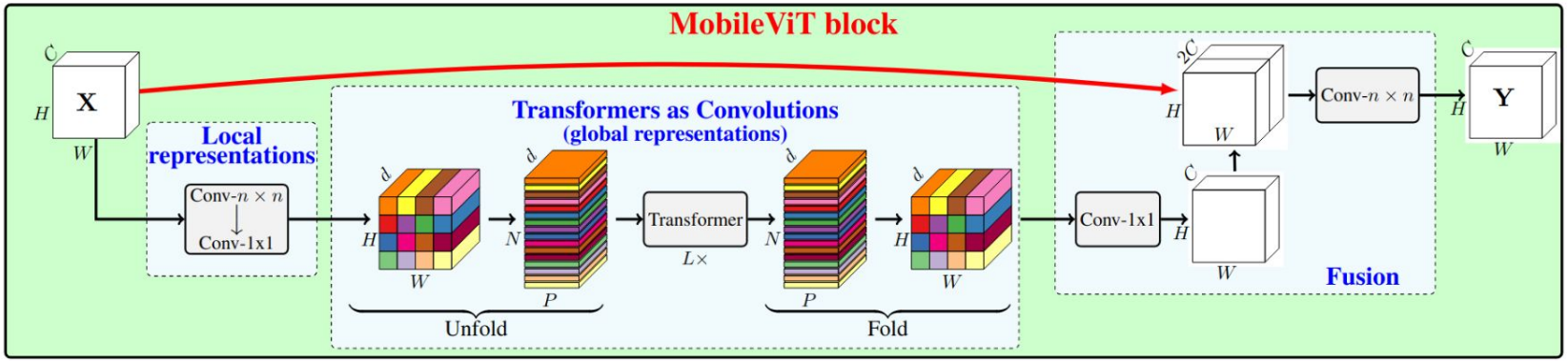
Сравнение производительности

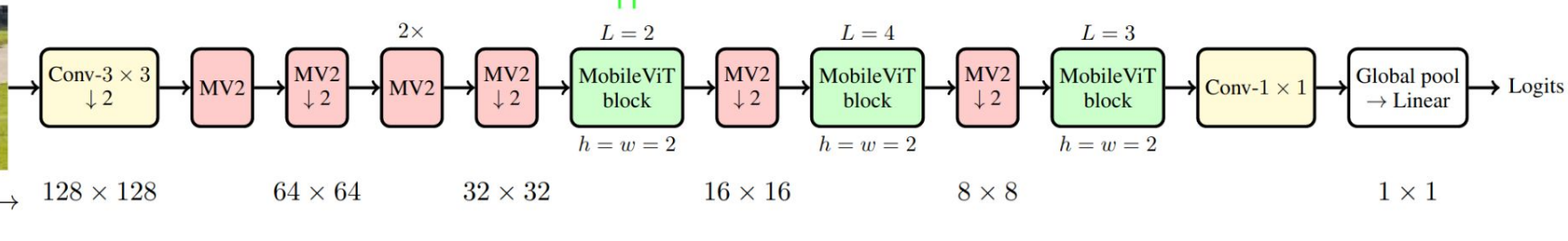


Классификация: трансформеры (ViT)



Классификация: Mobile ViT

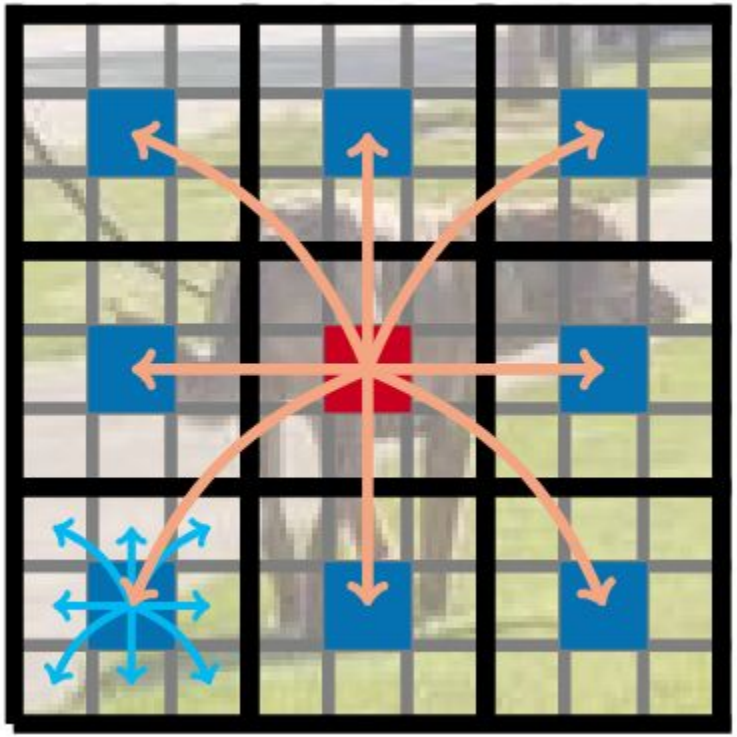




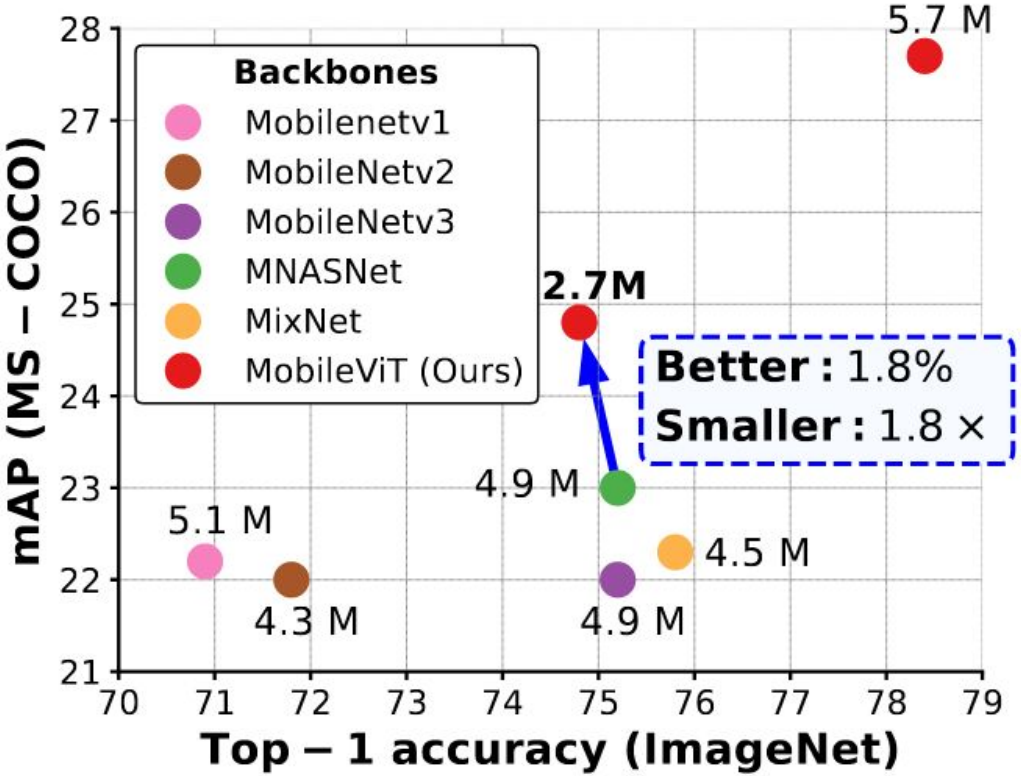
Output spatial → dimensions

128×128 64×64 32×32 16×16 8×8 1×1

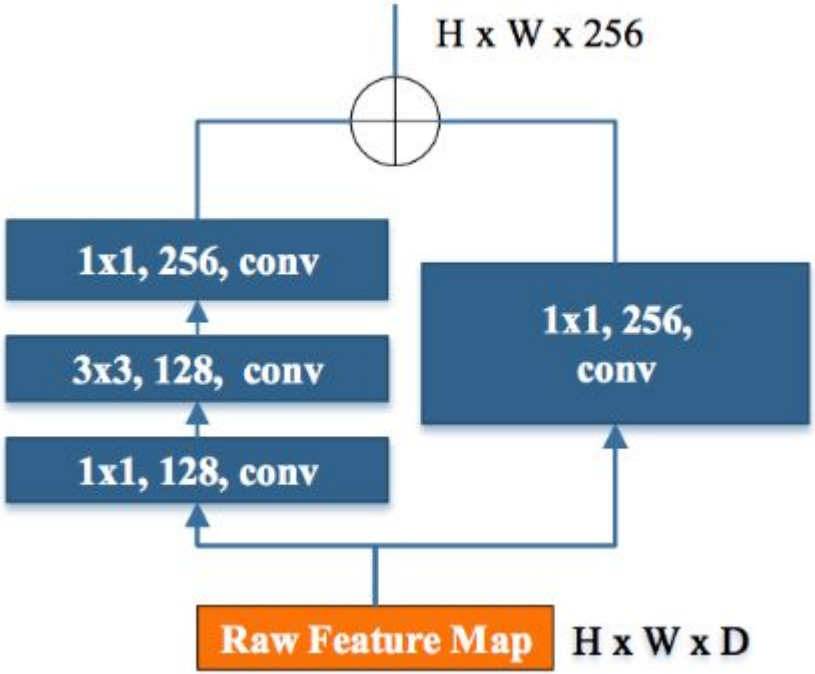
Классификация: Mobile ViT



Сравнение производительности



Классификация/детекция: Pelee



Классификация: Pelee

Table 4: **Speed on NVIDIA TX2** (The larger the better) The benchmark tool is built with NVIDIA TensorRT4.0 library.

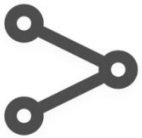
Model	**Top-1 Accuracy on ILSVRC2012** @224x224	**FLOPs** @224x224	**Speed (images per second)** Input Dimension		
			224x224	320x320	640x640
1.0 MobileNet	70.6	569 M	136.2	75.7	22.4
1.0 MobileNetV2	72.0	300 M	123.1	68.8	21.6
ShuffleNet 2x (g = 3)	**73.7**	524 M	110	65.3	19.8
PeleeNet (ours)	72.6	508 M	**240.3**	**129.1**	**37.2**

Детекция: Pelee

Table 10: Results on COCO test-dev2015

Model	Input Dimension	Speed on TX2 (FPS)	Model Size (Parameters)	Avg. Precision (%), IoU:		
				0.5:0.95	0.5	0.75
Original SSD	300x300	-	34.30 M	25.1	43.1	25.8
YOLOv2	416x416	32.2	67.43 M	21.6	44.0	19.2
YOLOv3	320x320	21.5	62.3 M	-	51.5	-
YOLOv3-Tiny	416x416	105	12.3 M	-	33.1	-
SSD+MobileNet	300x300	80	6.80 M	18.8	-	-
SSDlite + MobileNet v2	320x320	61	4.3 M	22	-	-
Pelee (ours)	304x304	120	5.98 M	**22.4**	**38.3**	**22.9**

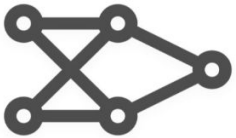
Детекция: YOLOv5



Small

YOLOv5s

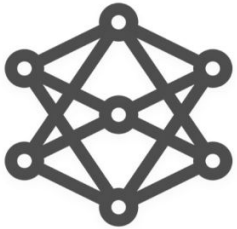
14 MB_{FP16}
2.0 ms_{V100}
37.2 mAP_{COCO}



Medium

YOLOv5m

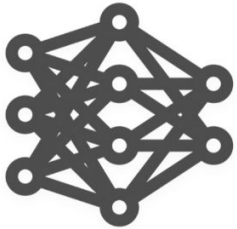
41 MB_{FP16}
2.7 ms_{V100}
44.5 mAP_{COCO}



Large

YOLOv5l

90 MB_{FP16}
3.8 ms_{V100}
48.2 mAP_{COCO}

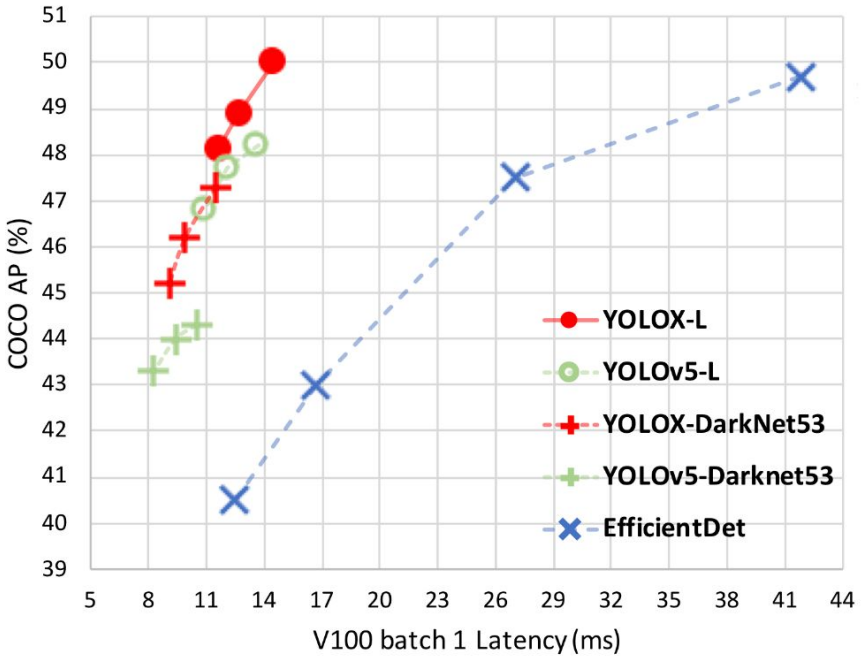


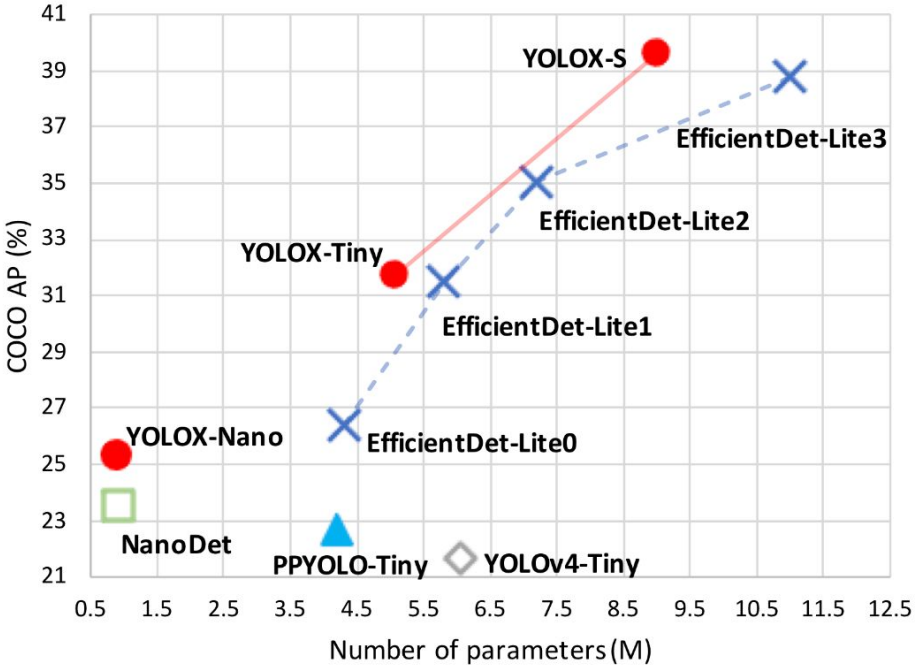
XLarge

YOLOv5x

168 MB_{FP16}
6.1 ms_{V100}
50.4 mAP_{COCO}

Детекция: YOLOX

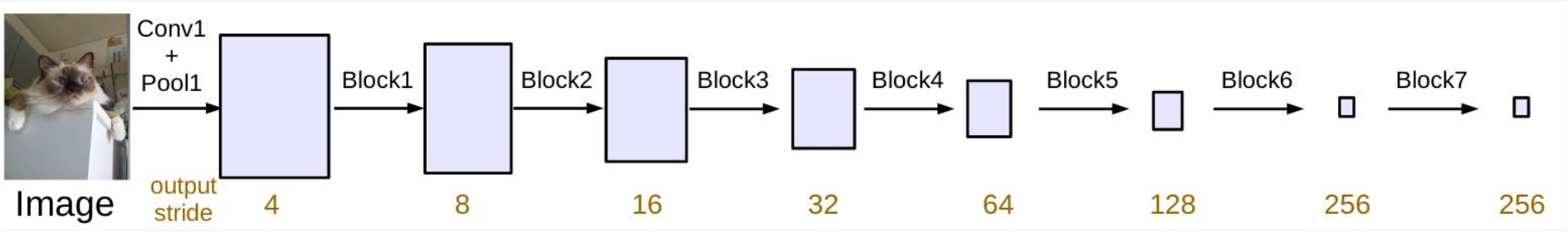




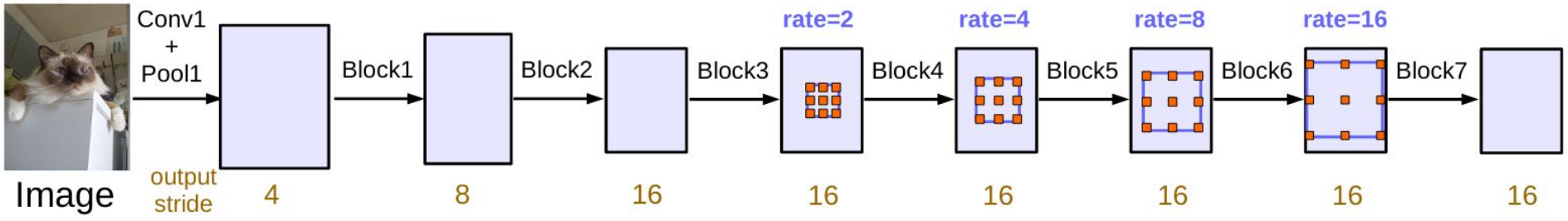
Детекция: YOLOX

Model	size	mAP^{val} 0.5:0.95	Params (M)	FLOPs (G)
YOLOX-Nano	416	25.8	0.91	1.08
YOLOX-Tiny	416	32.8	5.06	6.45

Сегментация: DeepLabV3



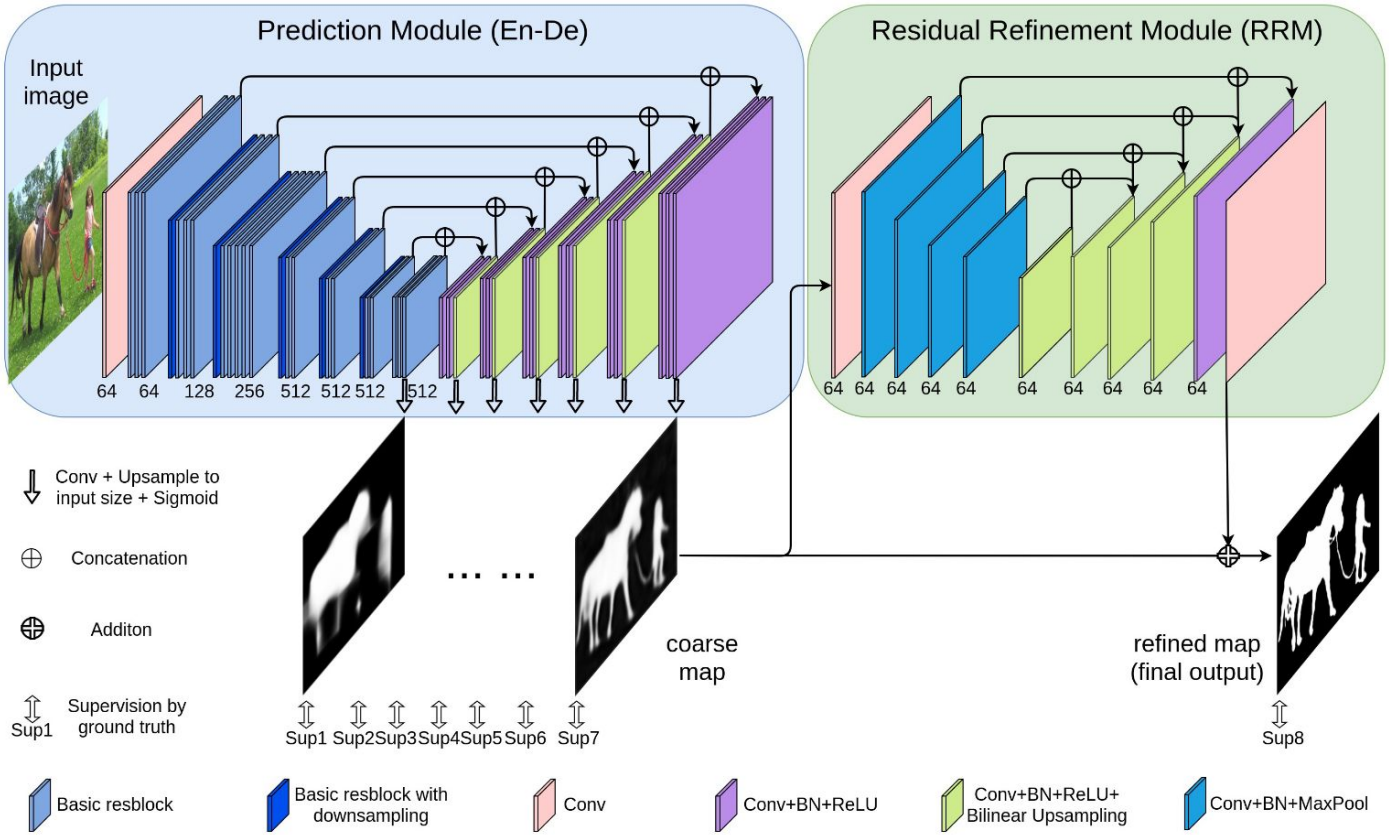
(a) Going deeper without atrous convolution.



Сравнение качества

S.No	Method	Precision	Recall	F1-score	Pixel Accuracy	*IoU*	*mIoU*
1	Otsu [57]	50%	80%	70%	78%	70%	55%
2	Watershed [56]	52%	82%	74%	80%	76%	60%
3	Gaussian mixture-based model [58]	60%	82%	76%	82%	79%	70%
4	Gaussian mixture-based model [59]	60%	84%	79%	82%	80%	72%
5	Background subtraction-based [60]	65%	84%	74%	84%	80%	70%
6	Re-weighted HOG & image segmentation [55]	68%	82%	75%	82%	79%	70%
7	Adaptively splitted GMM [61]	70%	84%	75%	82%	80%	74%
8	FCN [17]	62%	92%	76%	91%	83%	80%
9	U-Net [21]	74%	92%	81%	92%	84%	82%
10	DeepLabV3 [52]	80%	96%	83%	93%	86%	84%

Сегментация: BASNet



Сегментация: пример применения

(a) Copy (b) Move (c) Paste

Сегментация (каждого объекта): YOLACT

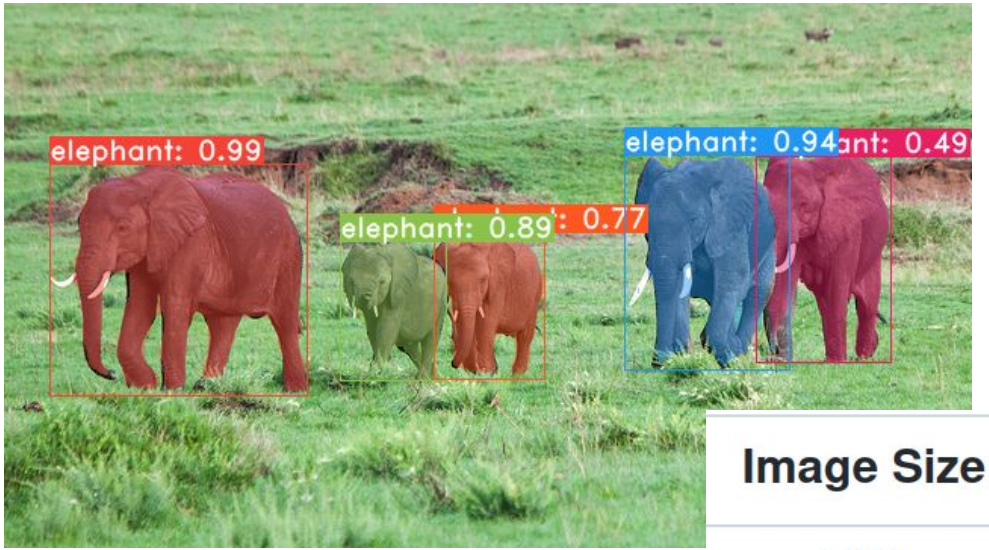


Image Size	Backbone	FPS	mAP
550	Resnet50-FPN	33.5	34.1
550	Resnet101-FPN	27.3	34.6

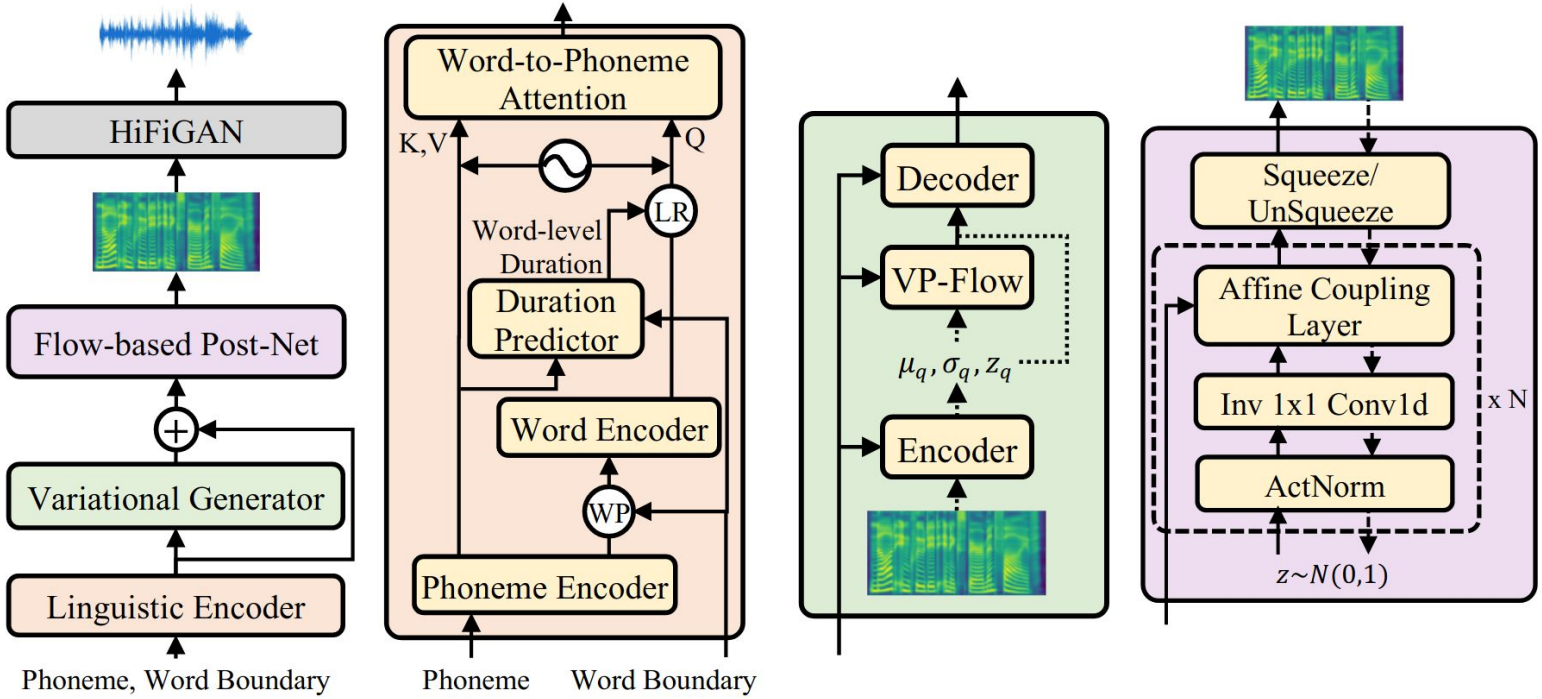
NLP: MobileBERT

			$\text{BERT}_{\text{LARGE}}$	$\text{BERT}_{\text{BASE}}$	$\text{IB-BERT}_{\text{LARGE}}$	MobileBERT	$\text{MobileBERT}_{\text{TINY}}$
embedding		$h_{\text{embedding}}$	1024	768	128		
			no-op	no-op	3-convolution		
		h_{inter}	1024	768	512		
body	Linear	h_{input} h_{output}	$\left[\begin{pmatrix}1024\\16\\1024\end{pmatrix}\begin{pmatrix}1024\\4096\\1024\end{pmatrix}\right]\times 24$	$\left[\begin{pmatrix}768\\12\\768\end{pmatrix}\begin{pmatrix}768\\3072\\768\end{pmatrix}\right]\times 12$	$\left[\begin{pmatrix}512\\1024\end{pmatrix}\begin{pmatrix}512\\4\\1024\end{pmatrix}\begin{pmatrix}1024\\4096\\1024\end{pmatrix}\begin{pmatrix}1024\\512\end{pmatrix}\right]\times 24$	$\left[\begin{pmatrix}512\\128\end{pmatrix}\begin{pmatrix}512\\4\\128\end{pmatrix}\begin{pmatrix}128\\512\\128\end{pmatrix}\times 4\begin{pmatrix}128\\512\end{pmatrix}\right]\times 24$	$\left[\begin{pmatrix}512\\128\end{pmatrix}\begin{pmatrix}128\\4\\128\end{pmatrix}\begin{pmatrix}128\\512\\128\end{pmatrix}\times 2\begin{pmatrix}128\\512\end{pmatrix}\right]\times 24$
	MHA	h_{input} #Head h_{output}					
	FFN	h_{input} h_{FFN} h_{output}					
	Linear	h_{input} h_{output}					
#Params			334M	109M	293M	25.3M	15.1M

Сравнение качества

	#Params	#FLOPS	Latency	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 5.7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	GLUE
ELMo-BiLSTM-Attn	-	-	-	33.6	90.4	84.4	72.3	63.1	74.1/74.5	79.8	58.9	70.0
OpenAI GPT	109M	-	-	47.2	93.1	87.7	84.8	70.1	80.7/80.6	87.2	69.1	76.9
$\text{BERT}_{\text{BASE}}$	109M	22.5B	342 ms	**52.1**	**93.5**	**88.9**	**85.8**	71.2	**84.6/83.4**	90.5	66.4	78.3
$\text{BERT}_{\text{BASE}}$-6L-PKD*	66.5M	11.3B	-	-	92.0	85.0	-	70.7	81.5/81.0	89.0	65.5	-
$\text{BERT}_{\text{BASE}}$-4L-PKD†*	52.2M	7.6B	-	24.8	89.4	82.6	79.8	70.2	79.9/79.3	85.1	62.3	-
$\text{BERT}_{\text{BASE}}$-3L-PKD*	45.3M	5.7B	-	-	87.5	80.7	-	68.1	76.7/76.3	84.7	58.2	-
$\text{DistilBERT}_{\text{BASE}}$-6L†	62.2M	11.3B	-	-	92.0	85.0		70.7	81.5/81.0	89.0	65.5	-
$\text{DistilBERT}_{\text{BASE}}$-4L†	52.2M	7.6B	-	32.8	91.4	82.4	76.1	68.5	78.9/78.0	85.2	54.1	-
TinyBERT*	14.5M	1.2B	-	43.3	92.6	86.4	79.9	**71.3**	82.5/81.8	87.7	62.9	75.4
$\text{MobileBERT}_{\text{TINY}}$	15.1M	3.1B	40 ms	46.7	91.7	87.9	80.1	68.9	81.5/81.6	89.5	65.1	75.8
MobileBERT	25.3M	5.7B	62 ms	50.5	92.8	88.8	84.4	70.2	83.3/82.6	90.6	66.2	77.7
MobileBERT w/o OPT	25.3M	5.7B	192 ms	51.1	92.6	88.8	84.8	70.5	84.3/**83.4**	**91.6**	**70.4**	**78.5**

Работа со звуком: PortaSpeech

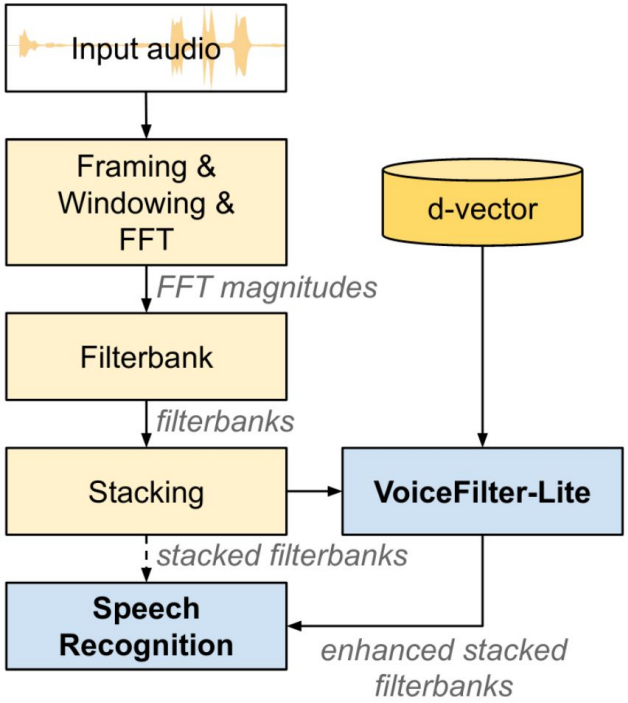


(a) PortaSpeech (b) Linguistic Encoder (c) Variational Generator (d) Flow-based Post-Net

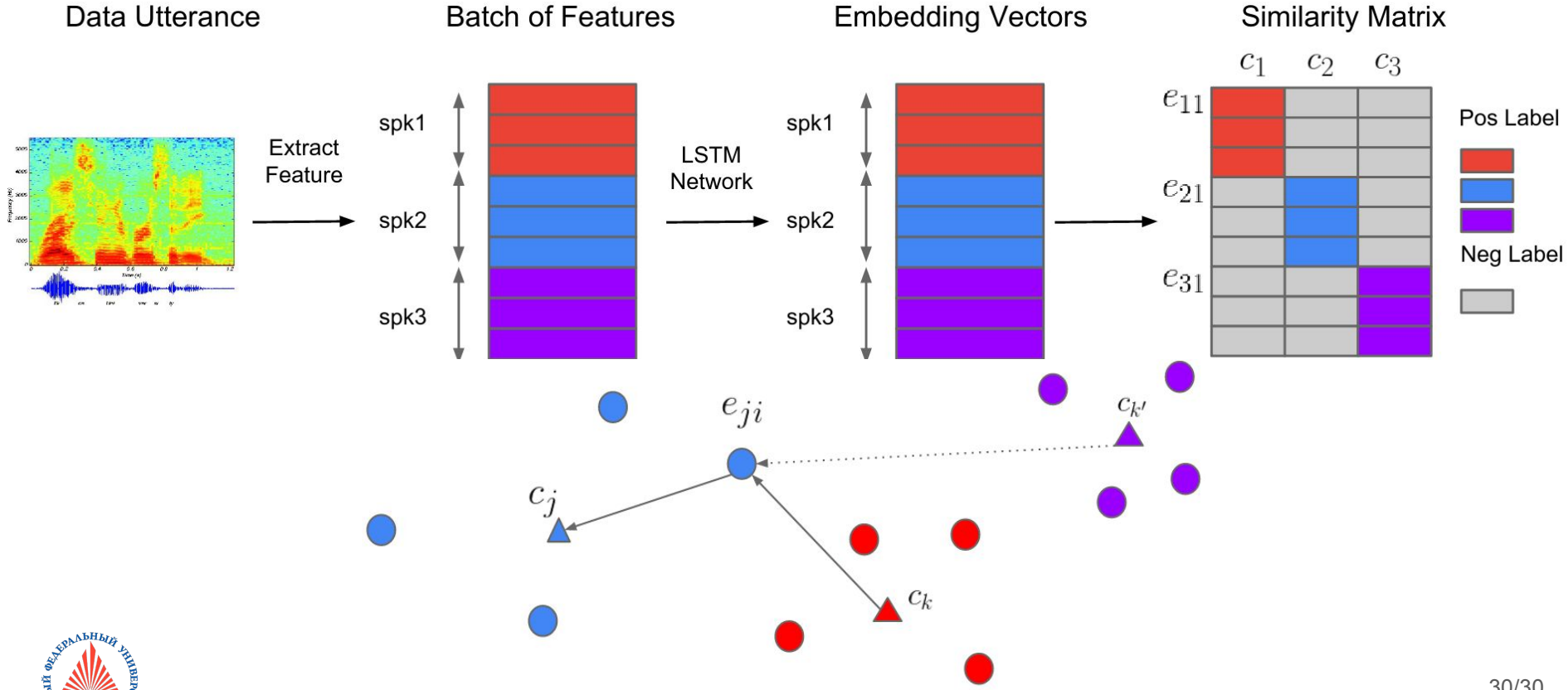
Работа со звуком: PortaSpeech

Module	Normal	Small
Total	24M	7.6M
LinguisticEncoder	3.7M	1.4M
VariationalGenerator	11M	2.8M
FlowPostNet	9.3M	3.4M

Работа со звуком: VoiceFilter-Lite



Работа со звуком: GE2E



Применение нейронных сетей в игровых движках

Лекция 7

Искусственный интеллект:
математические модели и прикладные решения
2022

Структура лекции

- Фреймворки для запуска нейронных сетей в игровых движках
- Сценарии использования нейронных сетей в игровых движках

Рассматриваемые движки (и Blender)

UNREAL
ENGINE

Unity

Поддерживаемый фреймворк:

- Barracuda

Поддерживаемые задачи:

- ML
- DL

UnrealEngine

Поддерживаемый фреймворк:

- TensorFlow (UE4 build)

Поддерживаемые задачи:

- ML
- DL

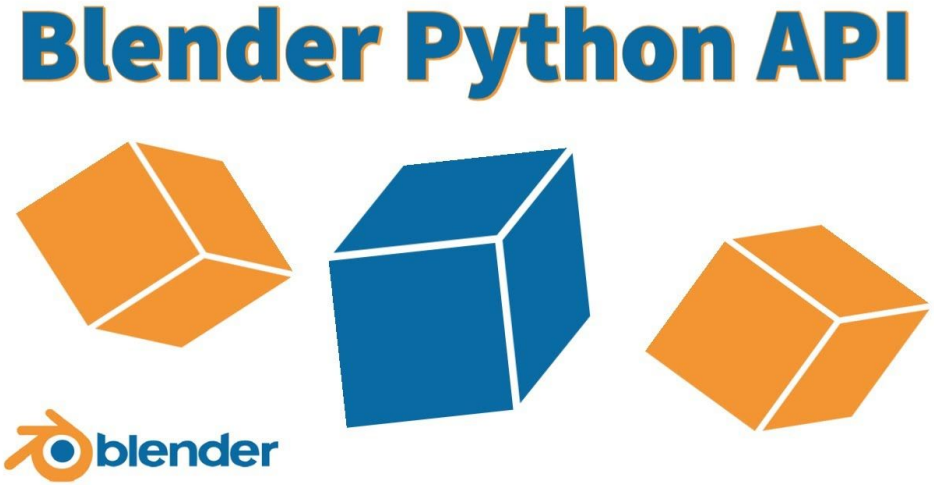
Blender

Поддерживаемый фреймворк:

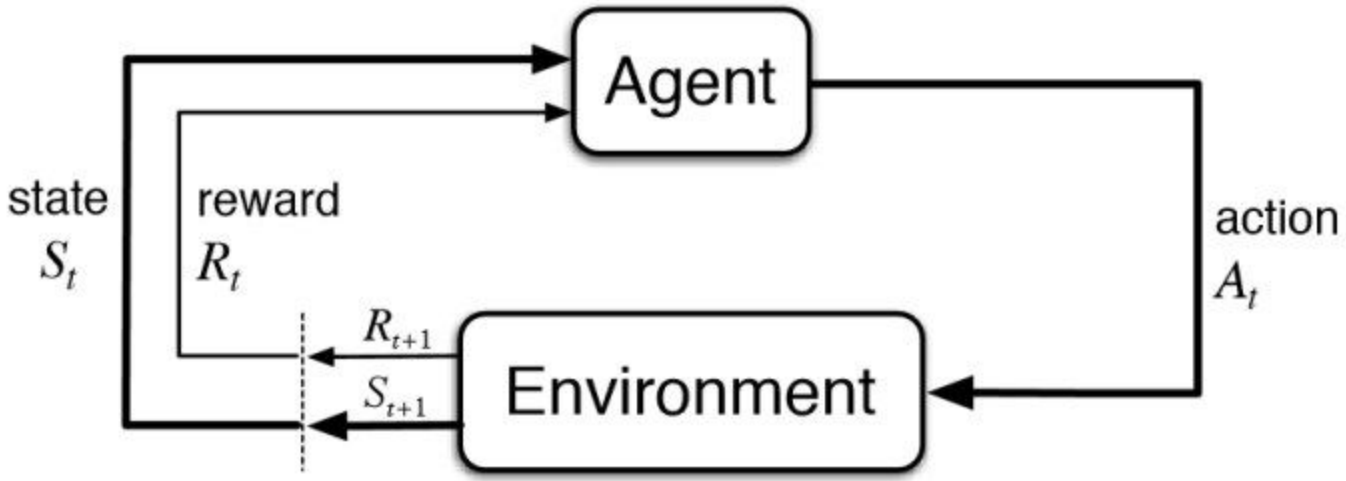
- Все доступные на Python:
  - PyTorch
  - TensorFlow
  - ONNX
  - и т. д.

Поддерживаемые задачи:

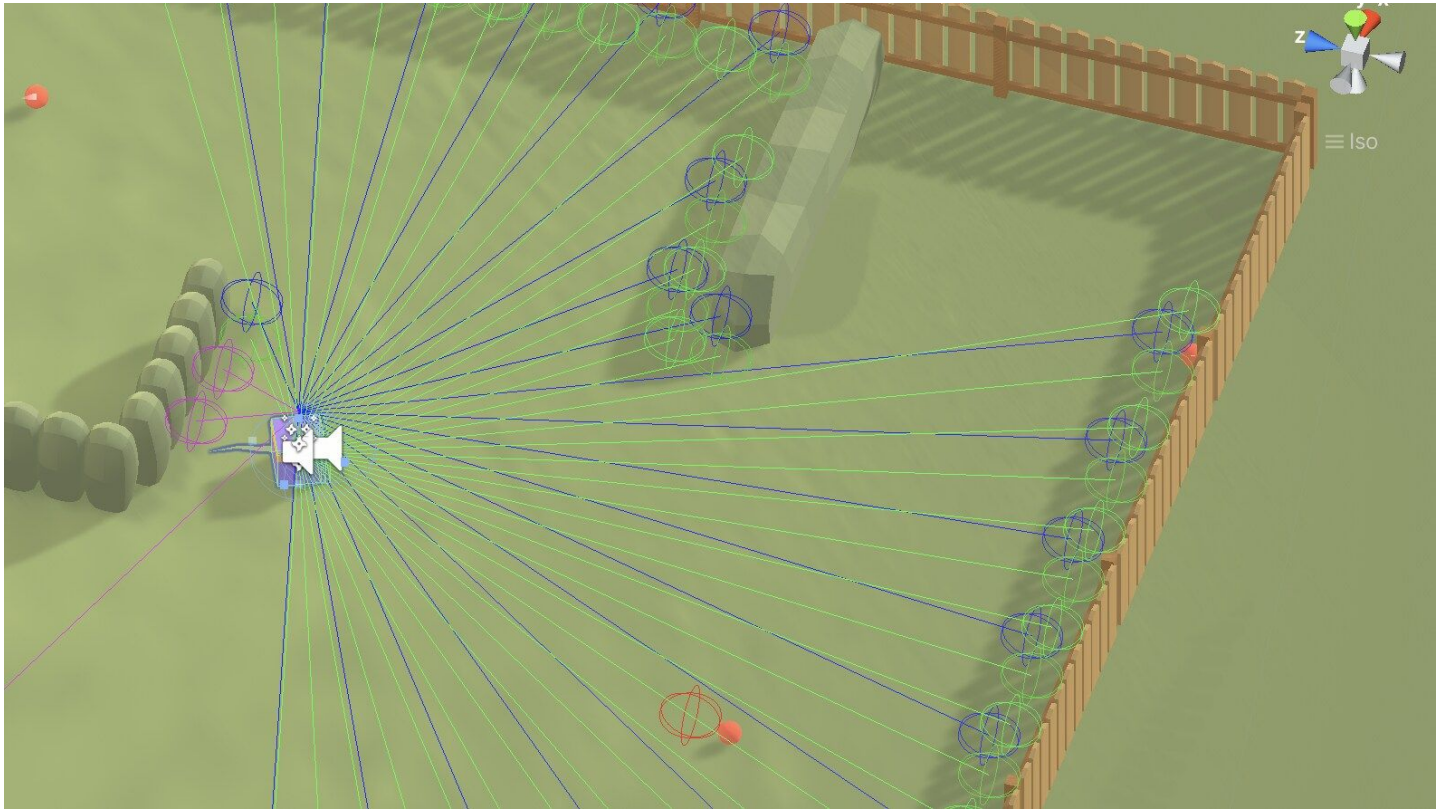
- ML
- DL



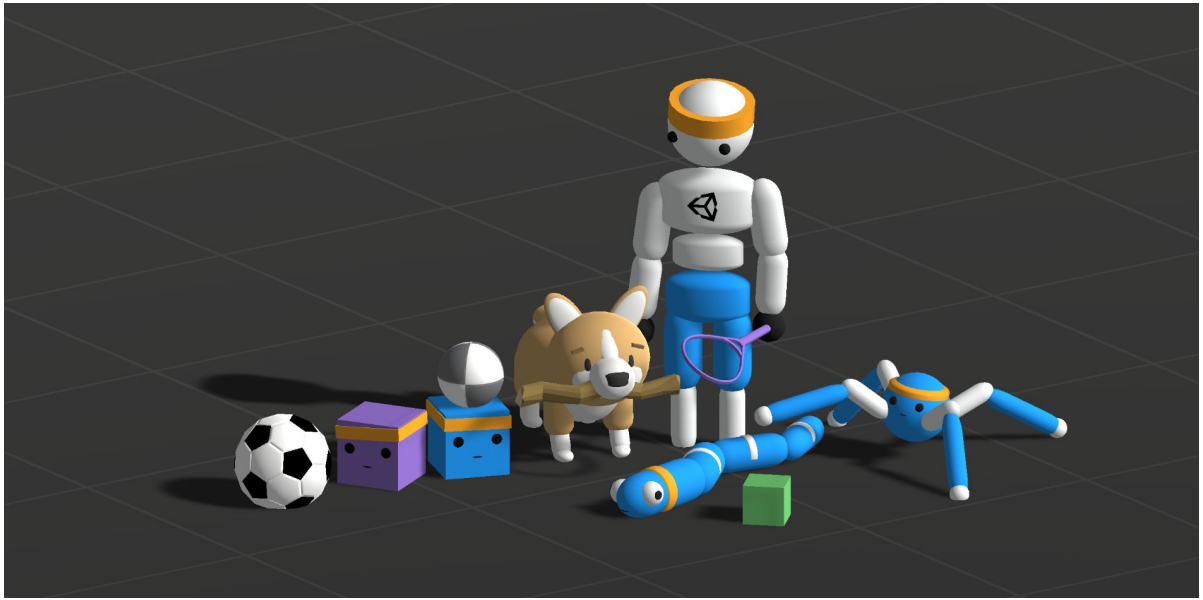
Reinforcement Learning (RL)



Reinforcement Learning (RL)

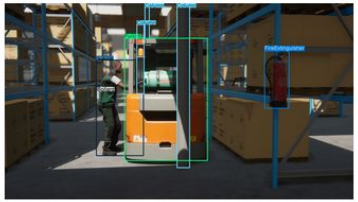


Unity ML-Agents Toolkit

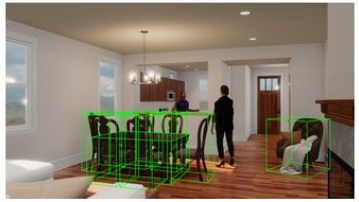
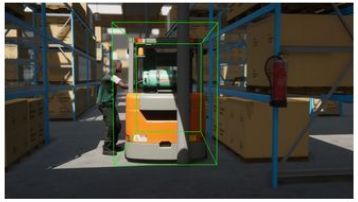


Datasets generation

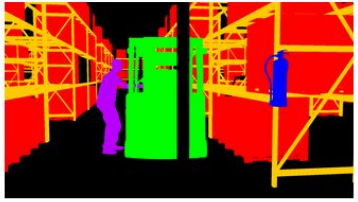
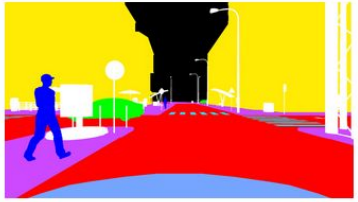
2D bounding boxes

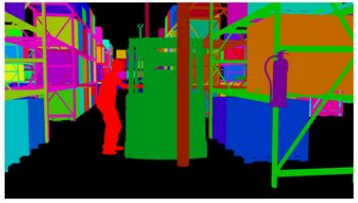
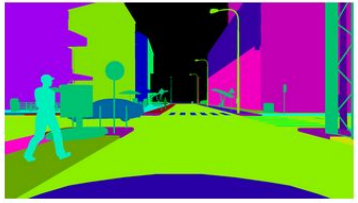

3D bounding boxes

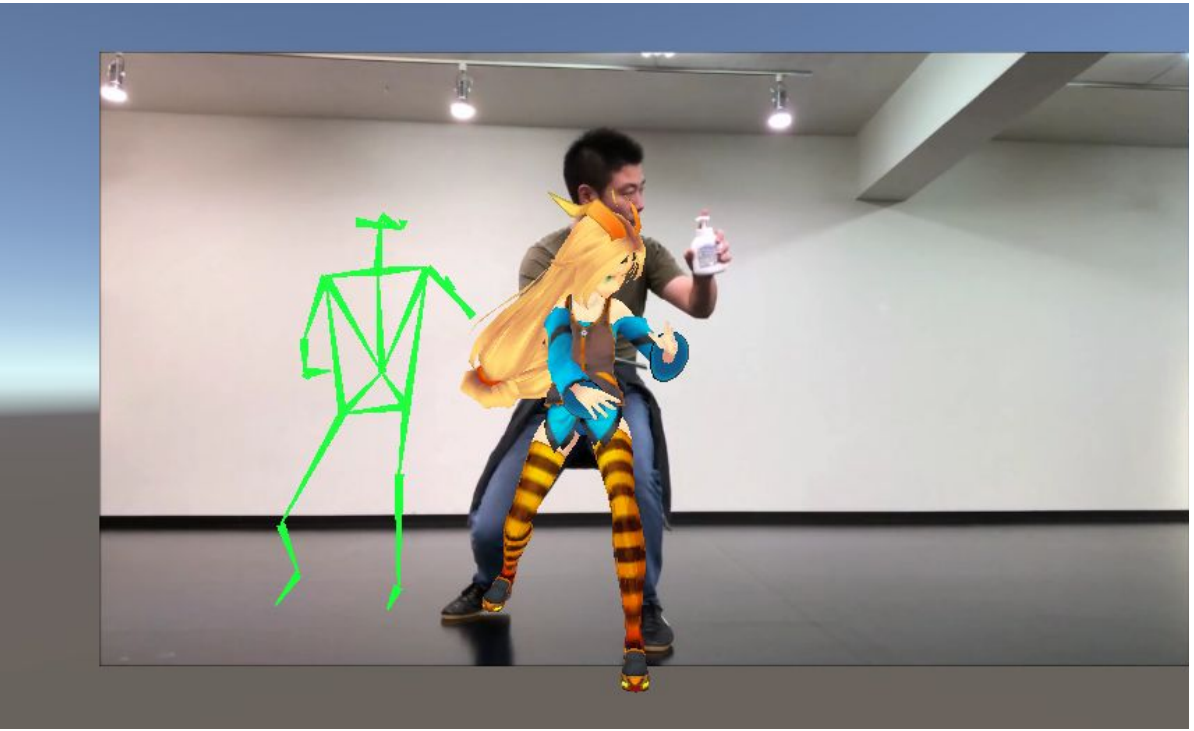

Class segmentation

Instance segmentation

Real world connection



Unity Robotics

Оптимизация моделей

Лекция 8

Искусственный интеллект:
математические модели и прикладные решения
2022

Структура лекции

- Оптимизация моделей для PyTorch
- Оптимизация моделей для ONNXRuntime
- Оптимизация моделей для TensorFlow-Lite
- Сравнение результатов

Инициализация модели

```
import torch

import torchvision

from PIL import Image

model = torchvision.models.resnet152(True)

_ = model.eval()
```

Инициализация входных данных

```
rand_tensor = torch.rand(1, 3, 224, 224)

img = Image.open('dog.png').convert('RGB')

transf = torchvision.transforms.Compose(

    [ torchvision.transforms.Resize((224, 224)),

        torchvision.transforms.ToTensor(),

        torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])])

img_tensor = transf(img).unsqueeze(0)
```

Чистый запуск на PyTorch

```
%%time
out = model(img_tensor).detach()
```

```
CPU times: user 418 ms, sys: 62 ms, total: 480 ms
Wall time: 441 ms
```

Трассировка, запуск и сравнение результатов

```
traced_model = torch.jit.trace(model, rand_tensor)
_ = traced_model.eval()
```

```
%%time
out2 = traced_model(img_tensor).detach()
```

```
CPU times: user 414 ms, sys: 64.2 ms, total: 479 ms
Wall time: 432 ms
```

```
torch.linalg.norm(out - out2), out.argmax(dim=1), out2.argmax(dim=1)
```

```
(tensor(0.), tensor([208]), tensor([208]))
```

Динамическая квантизация

```
model_int8 = torch.quantization.quantize_dynamic(
    model,
    {torch.nn.Linear, torch.nn.Conv2d},
    dtype=torch.qint8
)
```

```
%%time
out3 = model_int8(img_tensor).detach()
```

```
CPU times: user 427 ms, sys: 70.1 ms, total: 497 ms
Wall time: 454 ms
```

```
torch.linalg.norm(out - out3), out.argmax(dim=1), out3.argmax(dim=1)
```

```
(tensor(1.5398), tensor([208]), tensor([208]))
```

Статическая квантизация

```
static_model_int8 = torch.quantization.convert(model)
```

```
%%time
out4 = static_model_int8(img_tensor).detach()
```

```
CPU times: user 414 ms, sys: 62.5 ms, total: 476 ms
Wall time: 433 ms
```

```
torch.linalg.norm(out - out4), out.argmax(dim=1), out4.argmax(dim=1)
```

```
(tensor(0.), tensor([208]), tensor([208]))
```

TORCH.FX

- PyTorch >= 1.10

- Symbolic tracing
- Intermediate representation
- Python code generation

TORCH.FX: Пример

```
class MyModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = torch.nn.Linear(100, 5)
    def forward(self, x):
        return torch.sigmoid(self.fc(x))


module = MyModule()
```

TORCH.FX: Пример

```
from torch.fx import symbolic_trace
symbolic_traced : torch.fx.GraphModule = symbolic_trace(module)
print(symbolic_traced.graph)
```

```
graph():
    %x : [#users=1] = placeholder[target=x]
    %fc : [#users=1] = call_module[target=fc](args = (%x,), kwargs = {})
    %sigmoid : [#users=1] = call_function[target=torch.sigmoid](args = (%fc,), kwargs = {
    return sigmoid
```

```
print(symbolic_traced.code[3:])
```

```
def forward(self, x):
    fc = self.fc(x);  x = None
    sigmoid = torch.sigmoid(fc);  fc = None
    return sigmoid
```

TORCH.FX: Пример

```
symbolic_traced.graph.print_tabular()
```

```
opcode         name     target                                                  args         kwargs
-------------  -------  ------------------------------------------------------  -----------  --------
placeholder    x        x                                                       ()           {}
call_module    fc       fc                                                      (x,)         {}
call_function  sigmoid  <built-in method sigmoid of type object at 0x113b5e5c0>  (fc,)        {}
output         output   output                                                  (sigmoid,)   {}
```

Квантизация с помощью TORCH.FX

```
import torch.quantization.quantize_fx as quantize_fx
import copy

model_to_quantize = copy.deepcopy(model)
qconfig_dict = {"": torch.quantization.default_dynamic_qconfig}
model_to_quantize.eval()

model_prepared = quantize_fx.prepare_fx(model_to_quantize, qconfig_dict)

model_quantized_fx = quantize_fx.convert_fx(model_prepared)
```

```
%%time
out5 = model_quantized_fx(img_tensor).detach()
```

```
CPU times: user 381 ms, sys: 33.2 ms, total: 414 ms
Wall time: 384 ms
```

```
torch.linalg.norm(out - out5), out.argmax(dim=1), out5.argmax(dim=1)
```

```
(tensor(1.5398), tensor([208]), tensor([208]))
```

Запуск через ONNXRuntime

```
model.eval()
input_names = ['data']
output_names = ['output']
torch.onnx.export(model, rand_tensor, 'resnet152.onnx', input_names=input_names, output_names=output_names)
```

```
import onnxruntime as onnxrt
onnx_session= onnxrt.InferenceSession("resnet152.onnx")
onnx_inputs= {onnx_session.get_inputs()[0].name: img_tensor.numpy()}
```

```
%%time
onnx_output = onnx_session.run(None, onnx_inputs)
out6 = onnx_output[0]
```

```
CPU times: user 355 ms, sys: 2.88 ms, total: 358 ms
Wall time: 89.8 ms
```

```
torch.linalg.norm(out - out6), out.argmax(dim=1), out6.argmax(axis=1)
```

```
(tensor(5.6543e-05), tensor([208]), array([208]))
```

Конвертация модели в TensorFlow-Lite

```
import onnx2keras; from onnx2keras import onnx_to_keras

import keras; import onnx

onnx_model = onnx.load('resnet152.onnx')

keras_model = onnx_to_keras(onnx_model, ['data'])

keras.models.save_model(

   keras_model,'resnet152_keras.h5', overwrite=True,
include_optimizer=False)
```

Конвертация модели в TensorFlow-Lite

```
import tensorflow as tf

model = tf.keras.models.load_model('resnet152_keras.h5')

converter = tf.lite.TFLiteConverter.from_keras_model(model)

tflite_model = converter.convert()

open('resnet152.tflite', 'wb').write(tflite_model)
```

Скорость работы TensorFlow-Lite модели

```
interpreter = tf.lite.Interpreter(model_path='resnet152.tflite')
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
interpreter.allocate_tensors()
```

```
interpreter.set_tensor(input_details[0]['index'], [img_tensor.squeeze(0).numpy()])
```

```
%%time
interpreter.invoke()
```

```
CPU times: user 657 ms, sys: 17.9 ms, total: 675 ms
Wall time: 205 ms
```

```
out7 = interpreter.get_tensor(output_details[0]['index'])
torch.linalg.norm(out - out7), out.argmax(dim=1), out7.argmax(axis=1)
```

```
(tensor(7.5545e-05), tensor([208]), array([208]))
```

Оптимизация моделей в TensorFlow-Lite

```
model = tf.keras.models.load_model('resnet152_keras.h5')

converter = tf.lite.TFLiteConverter.from_keras_model(model)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite_model = converter.convert()

open('resnet152.tflite', 'wb').write(tflite_model)
```

Оптимизация моделей в TensorFlow-Lite

```
%%time
interpreter.invoke()
```

```
CPU times: user 24.4 s, sys: 7.85 ms, total: 24.4 s
Wall time: 24.4 s
```

```
out7 = interpreter.get_tensor(output_details[0]['index'])
torch.linalg.norm(out - out7), out.argmax(dim=1), out7.argmax(axis=1)
```

```
(tensor(3.8195), tensor([208]), array([208]))
```

Сравнение для архитектуры x86

Optimization	Inference time (ms)	Acceleration
Original PyTorch	441	x1.0
Traced PyTorch	432	x1.02
Quantized PyTorch	433	x1.02
Quantized TORCH.FX	384	x1.15
ONNX	90	x4.9
TensorFlow-Lite	205	x2.15

Слабые стороны мобильных устройств

Лекция 9

Искусственный интеллект:
математические модели и прикладные решения
2022

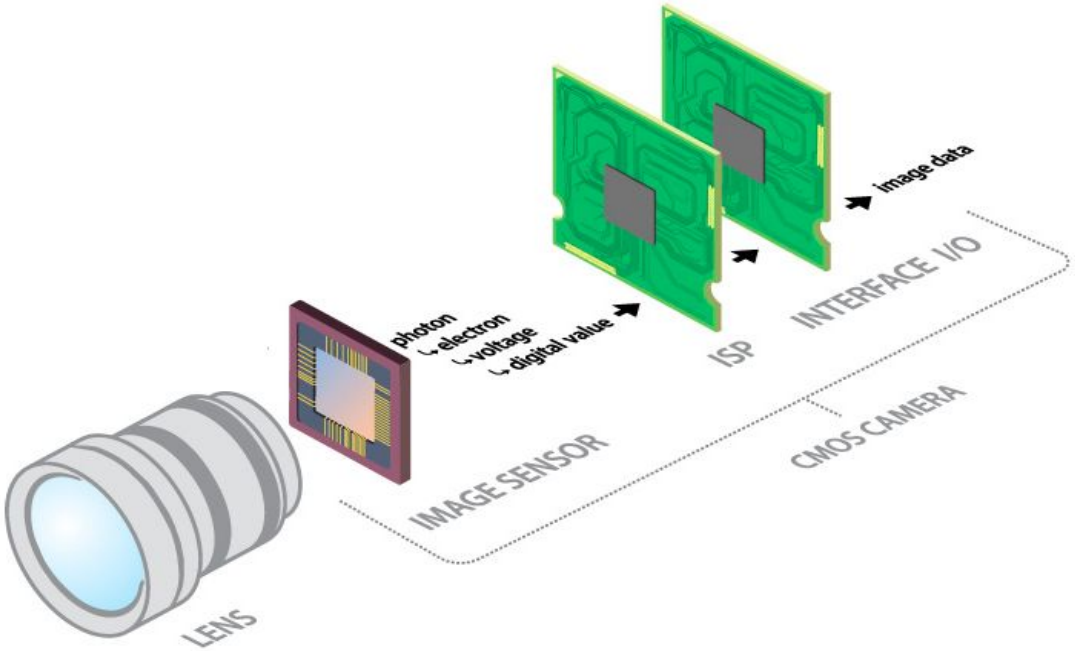
Структура лекции

- Слабые стороны мобильных устройств
- Улучшение слабых сторон

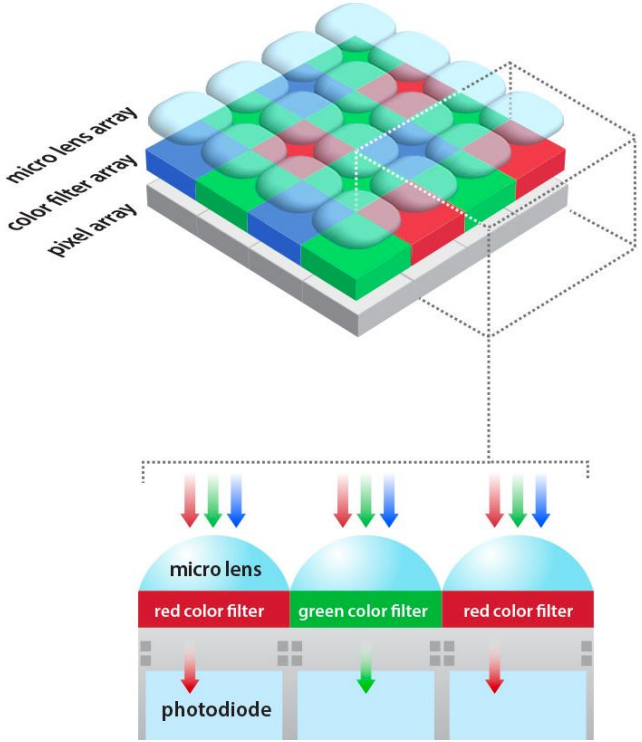
Некоторые слабые стороны мобильных устройств

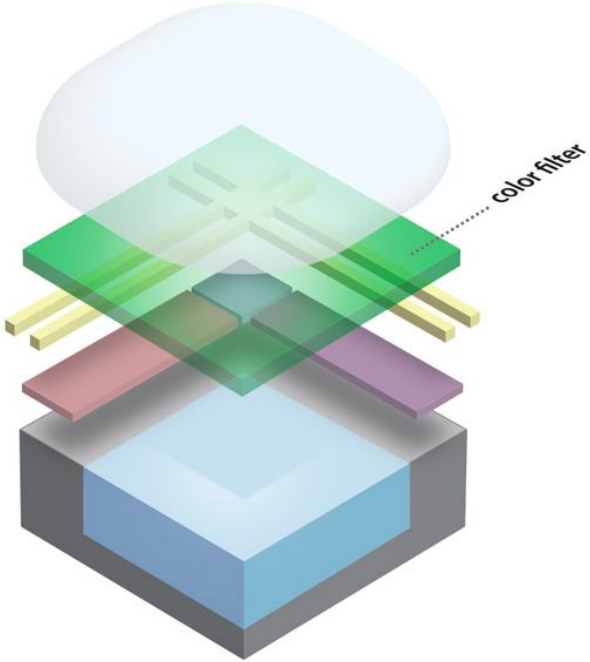
- Камера
- Процессор
- Снижение производительности при длительной работе
- Автономность

Устройство камеры

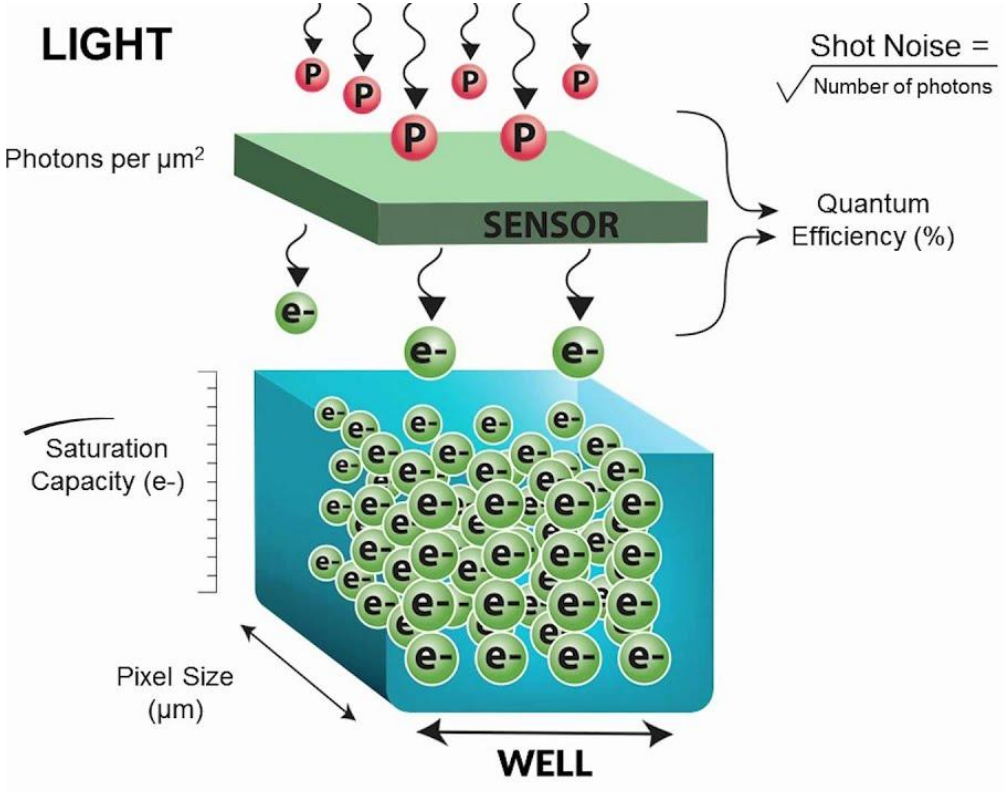


Устройство CMOS матрицы

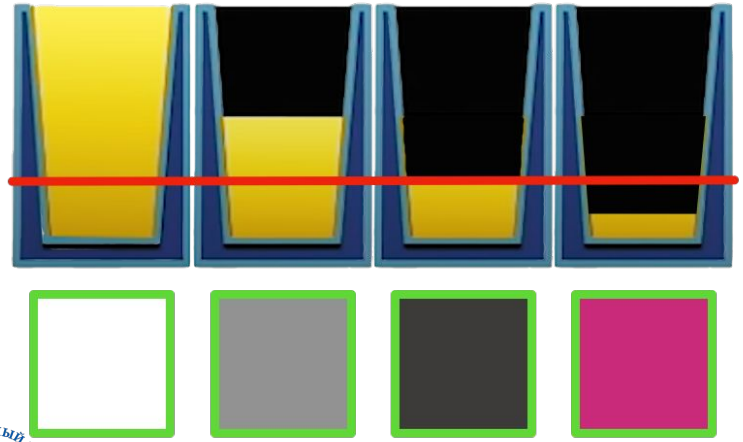




Механизм получения значения пикселя



Появление шума



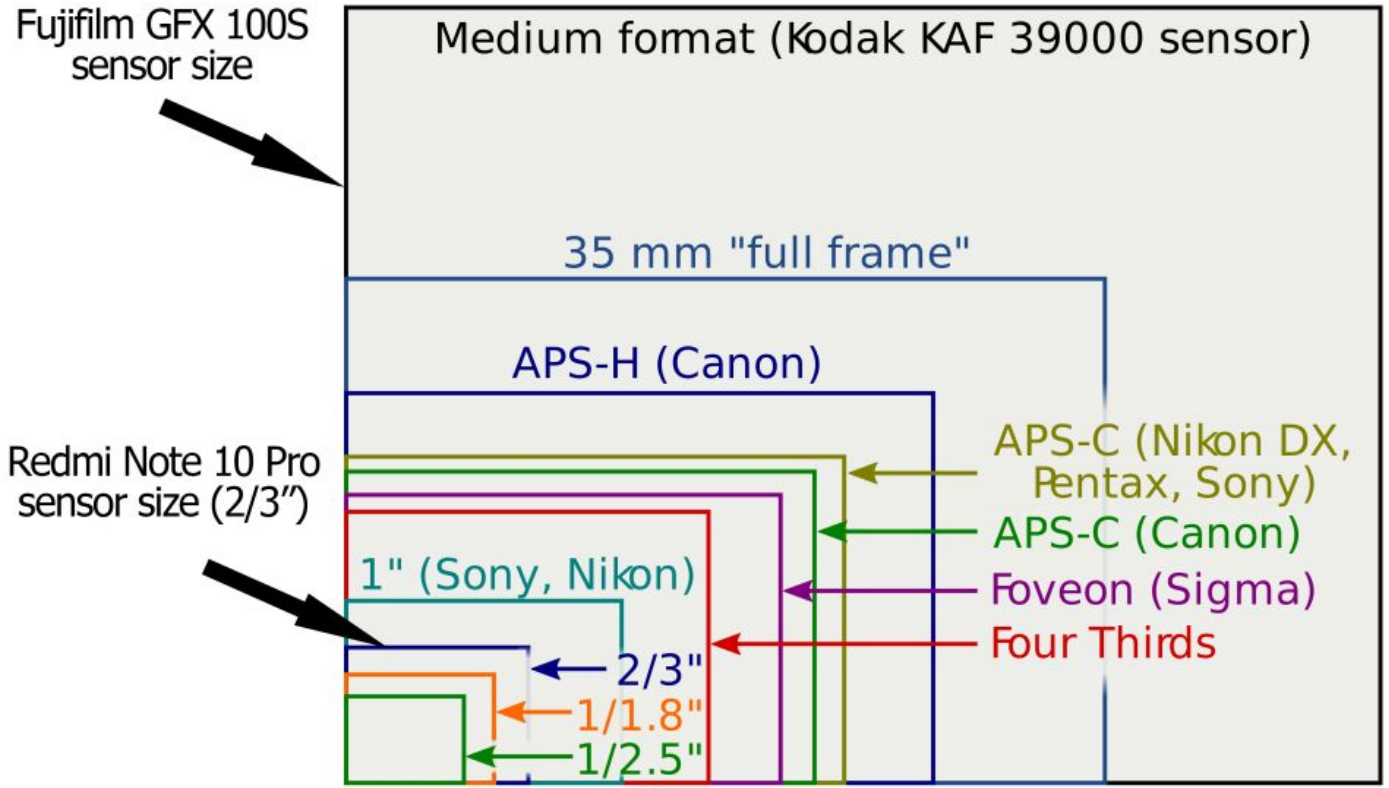
Камера

- iPhone 13 Pro: 12 MP sensor, 1.9μm pixels
- Samsung ISOCELL HM2: 108 MP sensor, 0.7μm pixels

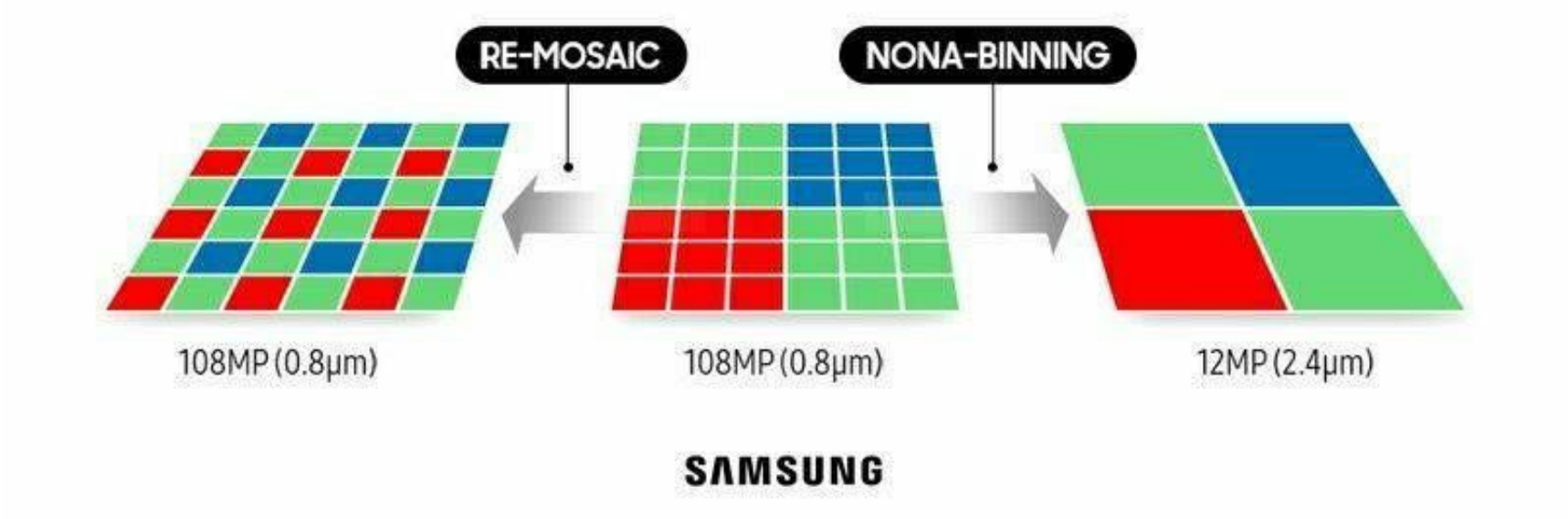
- Sony Alpha A7 III (full-frame): 24 MP, 5.91μm pixels
- Sony a6400 (1.5” crop): 24 MP, 3.89μm pixels

- DJI Mavic 3 (4/3” microframe): 20 MP

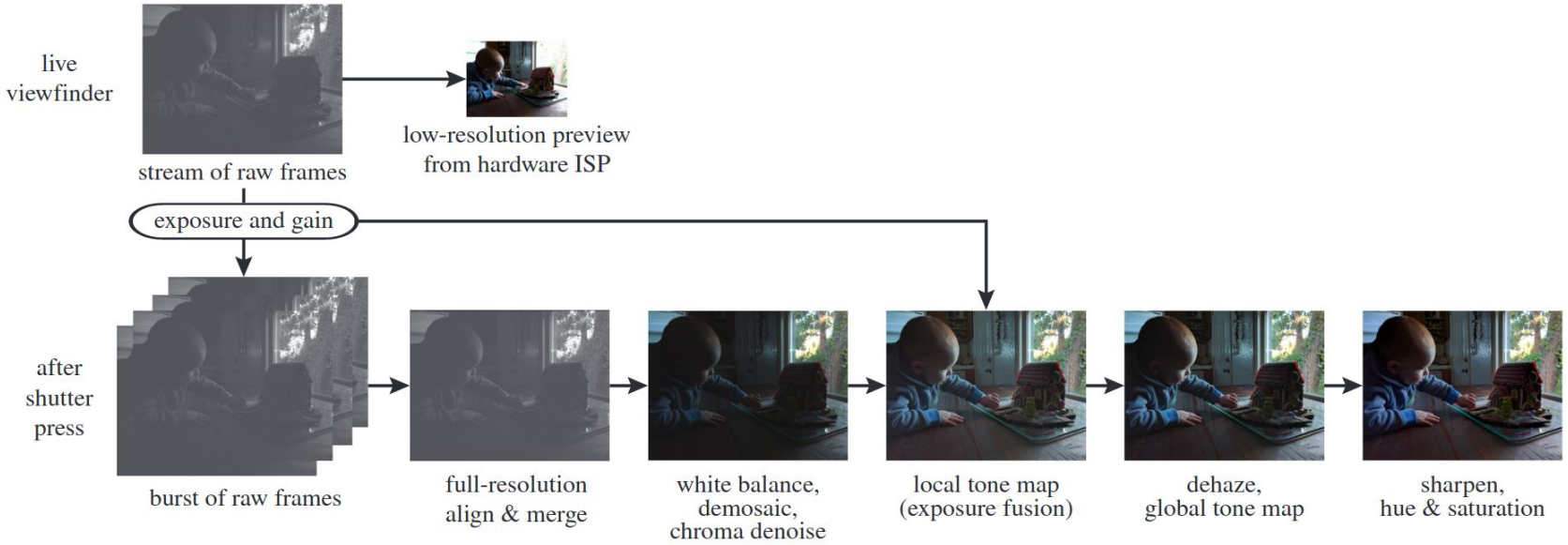
Камера


Fujifilm GFX 100S sensor size
Medium format (Kodak KAF 39000 sensor)
35 mm "full frame"
APS-H (Canon)
Redmi Note 10 Pro sensor size (2/3")
APS-C (Nikon DX, Pentax, Sony)
APS-C (Canon)
Foveon (Sigma)
Four Thirds
1" (Sony, Nikon)
2/3"
1/1.8"
1/2.5"

Способы улучшения фотографии: pixel binning



Способы улучшения фотографии: сложные пайплайны (Google)



Способы улучшения фотографии: сложные пайплайны (Apple)

Процессор

- Qualcomm Snapdragon 888 (1x 2,84 ГГц - Kryo 680 Prime (Cortex-X1), 3x 2,42 ГГц - Kryo 680 Gold (Cortex-A78), 4x 1,8 ГГц - Kryo 680 Silver (Cortex-A55))

Распределение нагрузки

2 new high-performance cores

4 new high-efficiency cores

Faster Neural Engine

New display engine

New video encoders

New video decoders

Double system cache

New image signal processor