Deployment on the Edge

PyTorch in Munich at Microsoft

Munich Applied Deep Learning Meetup

Nov 15th, 2019

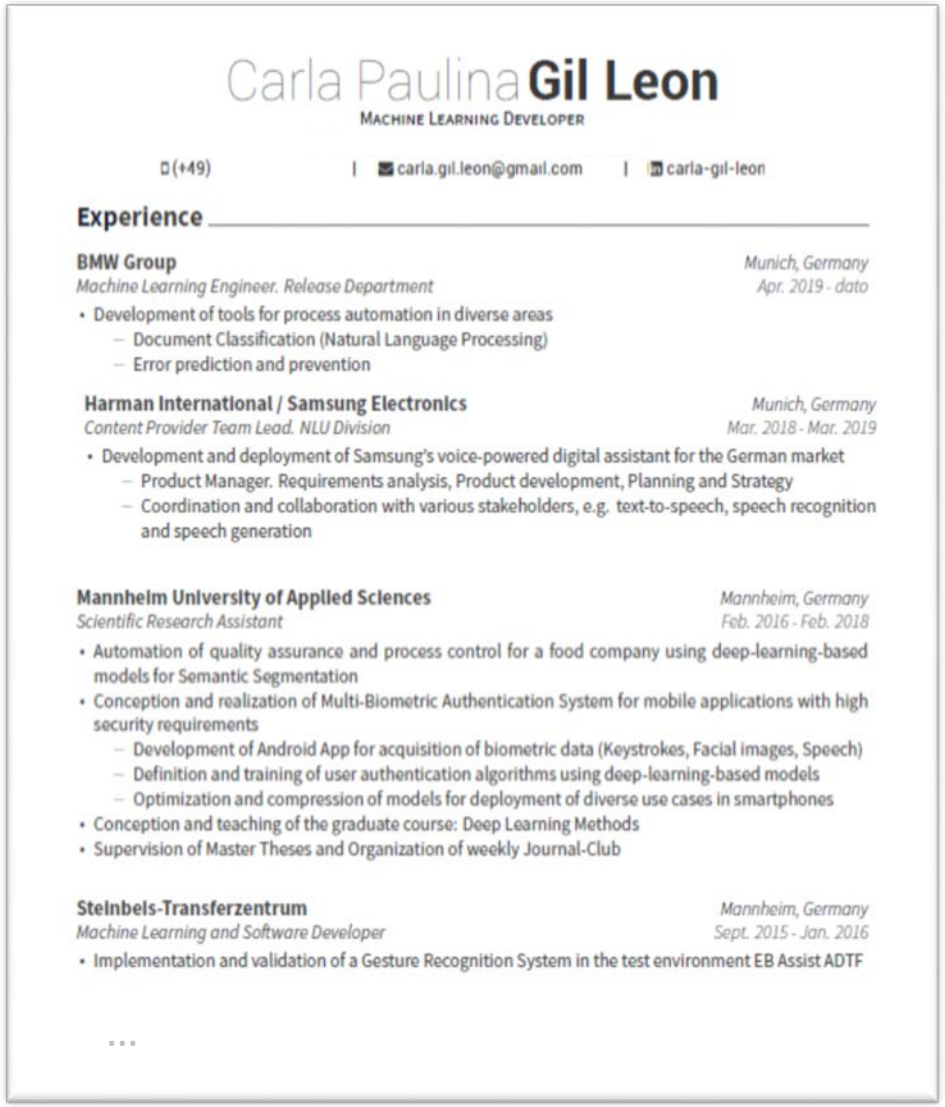
Carla Gil

Personal Card

Machine Learning Developer

M.Sc. Information Technology

Telecommunications Engineer



Carla Paulina **Gil Leon**

MACHINE LEARNING DEVELOPER

(+49) | carla.gil.leon@gmail.com | carla-gil-leon

Experience

BMW Group — *Munich, Germany*
Machine Learning Engineer. Release Department — *Apr. 2019 - dato*
- Development of tools for process automation in diverse areas
  - Document Classification (Natural Language Processing)
  - Error prediction and prevention

Harman International / Samsung Electronics — *Munich, Germany*
Content Provider Team Lead. NLU Division — *Mar. 2018 - Mar. 2019*
- Development and deployment of Samsung's voice-powered digital assistant for the German market
  - Product Manager. Requirements analysis, Product development, Planning and Strategy
  - Coordination and collaboration with various stakeholders, e.g. text-to-speech, speech recognition and speech generation

Mannheim University of Applied Sciences — *Mannheim, Germany*
Scientific Research Assistant — *Feb. 2016 - Feb. 2018*
- Automation of quality assurance and process control for a food company using deep-learning-based models for Semantic Segmentation
- Conception and realization of Multi-Biometric Authentication System for mobile applications with high security requirements
  - Development of Android App for acquisition of biometric data (Keystrokes, Facial images, Speech)
  - Definition and training of user authentication algorithms using deep-learning-based models
  - Optimization and compression of models for deployment of diverse use cases in smartphones
- Conception and teaching of the graduate course: Deep Learning Methods
- Supervision of Master Theses and Organization of weekly Journal-Club

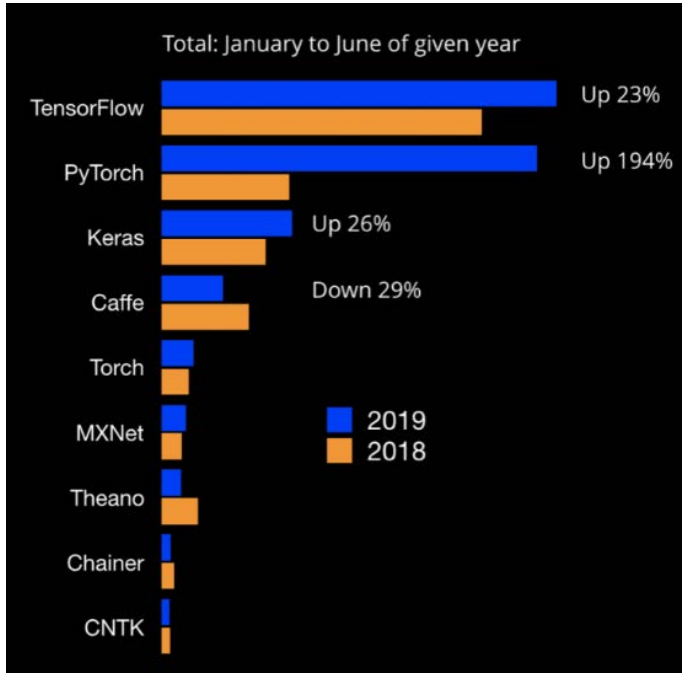
Steinbeis-Transferzentrum — *Mannheim, Germany*
Machine Learning and Software Developer — *Sept. 2015 - Jan. 2016*
- Implementation and validation of a Gesture Recognition System in the test environment EB Assist ADTF

...

Usage of PyTorch

Research:

Up 194%



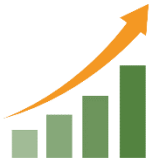
Measured by the number of papers on arxiv.org that mention Pytorch 2018 vs 2019.
Source: RISELab

Up 240%

CONFERENCE	PT 2018	PT 2019	PT GROWTH	TF 2018	TF 2019	TF GROWTH
CVPR	82	280	240%	116	125	7.7%
NAACL	12	66	450%	34	21	-38.2%
ACL	26	103	296%	34	33	-2.9%
ICLR	24	70	192%	54	53	-1.9%
ICML	23	69	200%	40	53	32.5%

Measured by the number of papers at the top research conferences 2018 vs 2019.
Source: TheGradient

Usage of PyTorch

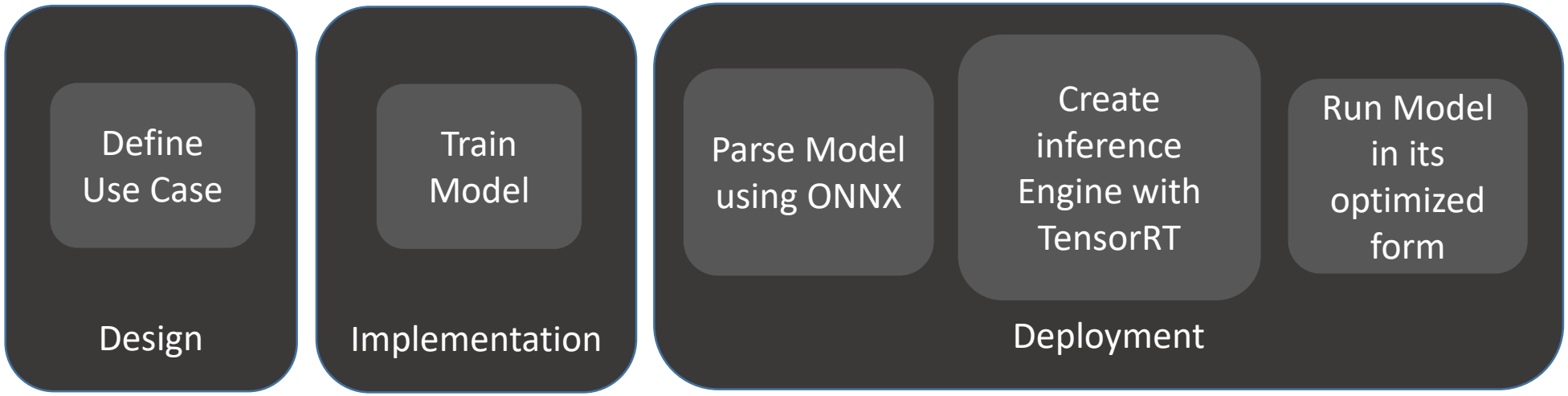


Industry:

- 1437 new job listings from 2018 to 2019 for PyTorch – against 1541 for TensorFlow (LinkedIn, Indeed, Simply Hired, Monster, Angel List, …)
- 3230 new TensorFlow Medium articles vs. 1200 PyTorch

Source: TheGradient

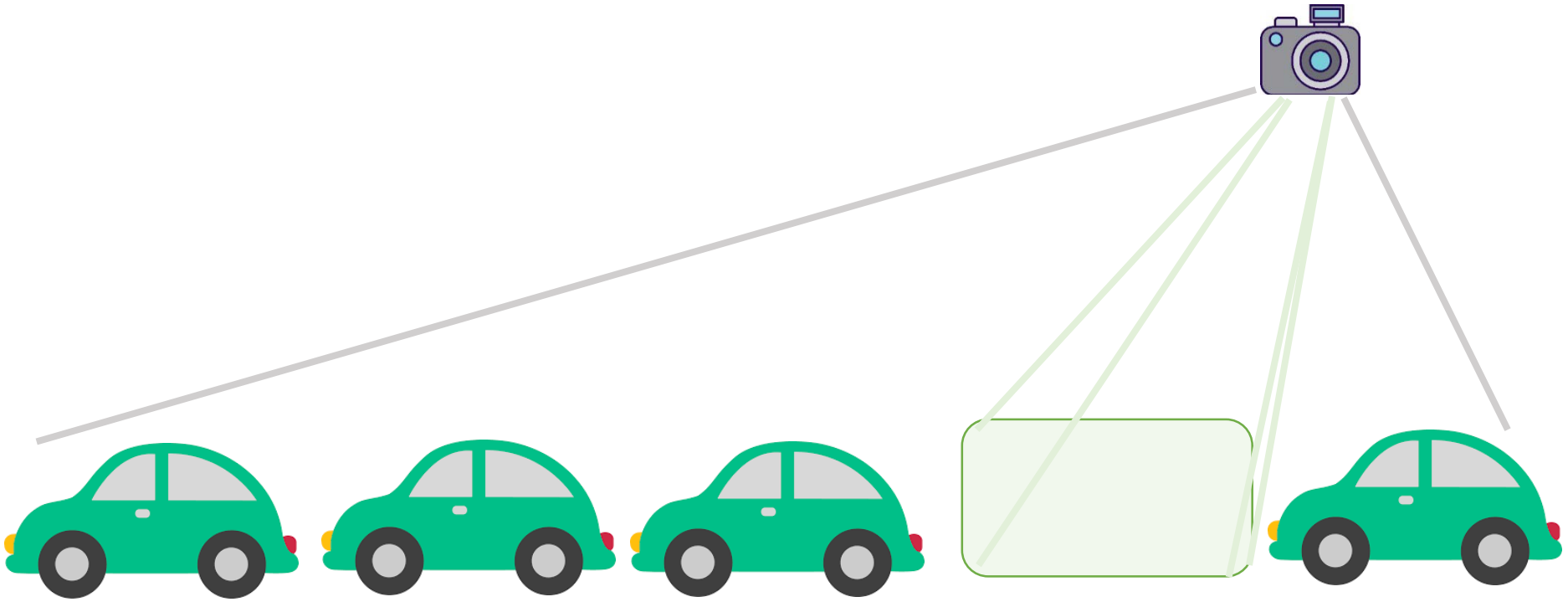
From Design to Deployment



Design

Use Case

Parking Spot Detection System



Equipment

- **Nvidia JETSON Nano.** Small, powerful computer with GPU
- Raspberry Pi Camera Module v2
- Smartphone

NVIDIA.
128-core Maxwell

Equipment

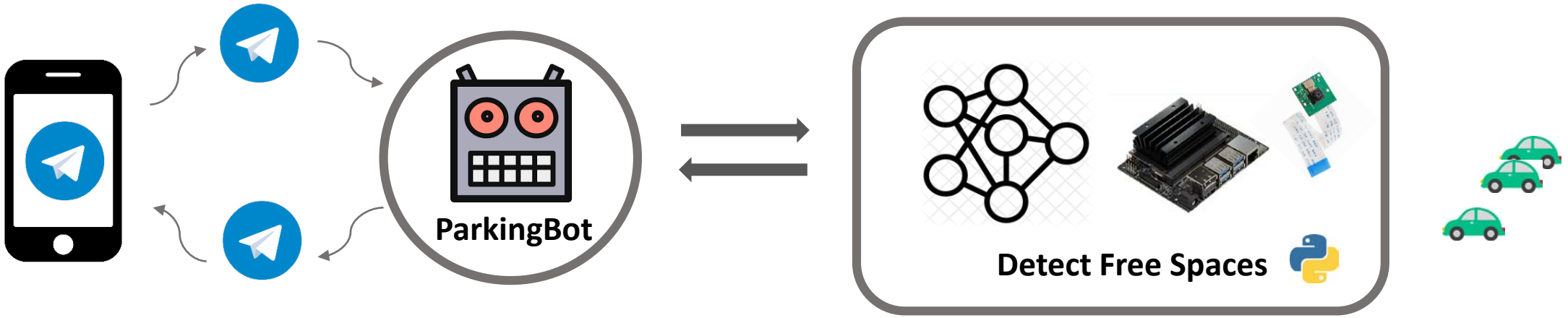
- Nvidia JETSON Nano. Small, powerful computer with GPU
- **Raspberry Pi Camera Module v2**
- Smartphone

Equipment

- Nvidia JETSON Nano. Small, powerful computer with GPU
- Raspberry Pi Camera Module v2
- **Smartphone**

Setup

- The camera attached to the JETSON Nano is placed on the window pointing to the street.
- A Telegram BOT is controlled over a Python Application also running on the JETSON Nano.
- The detection is activated when the BOT receives a command.



Implementation

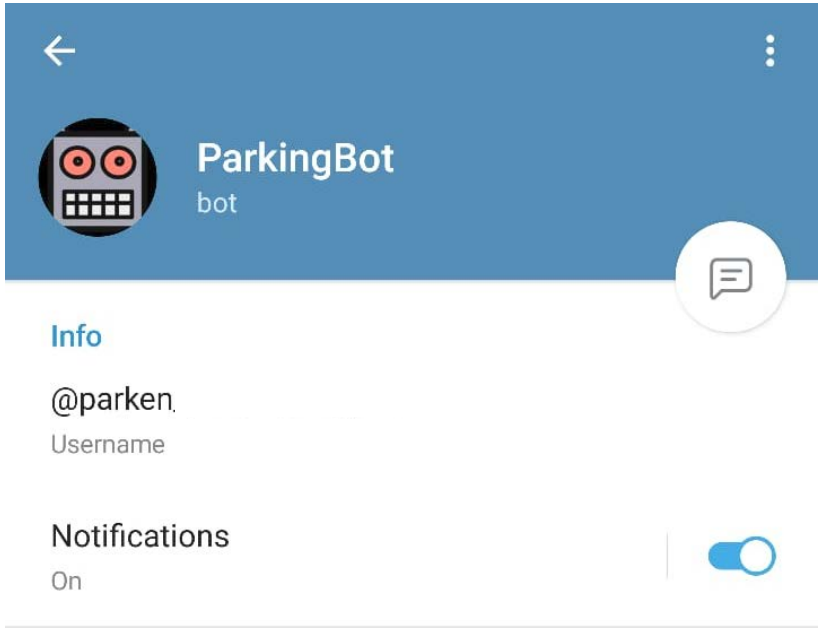
ParkingBot - Telegram BOT

The **Bot API** from Telegram allows us to communicate with the server using a simple HTTPS-interface. The library **python-telegram-bot** provides a pure Python interface

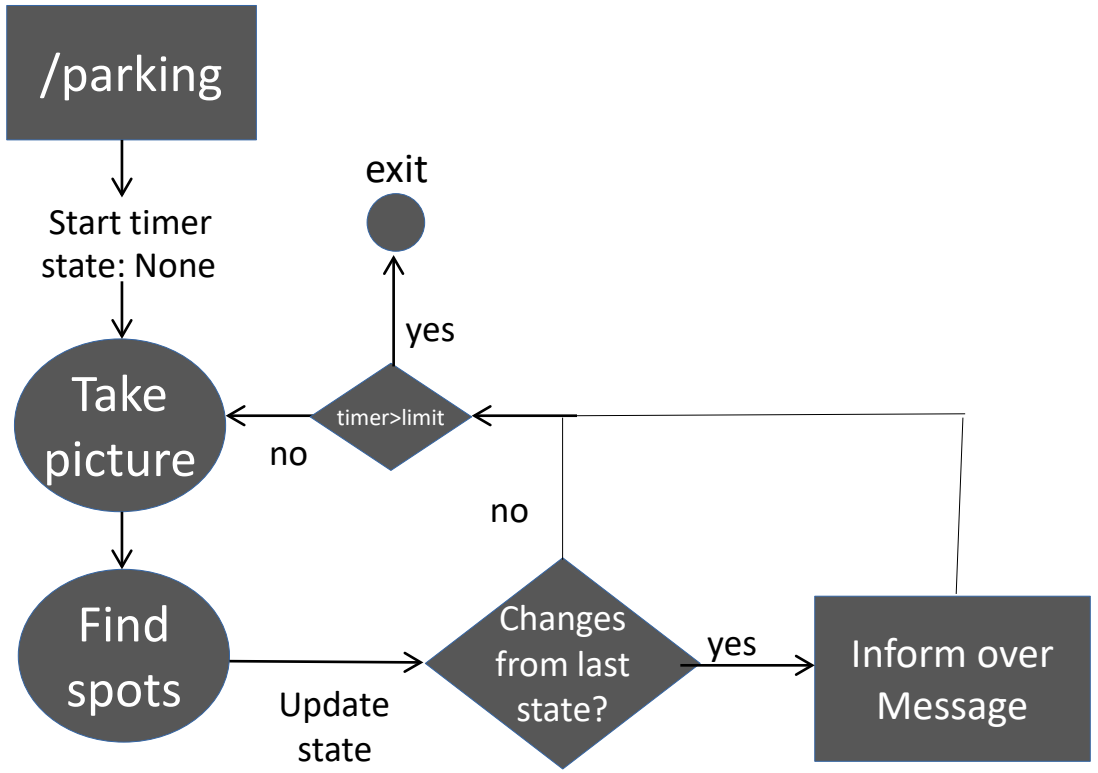
python-telegram-bot

ParkingBot is controlled using commands

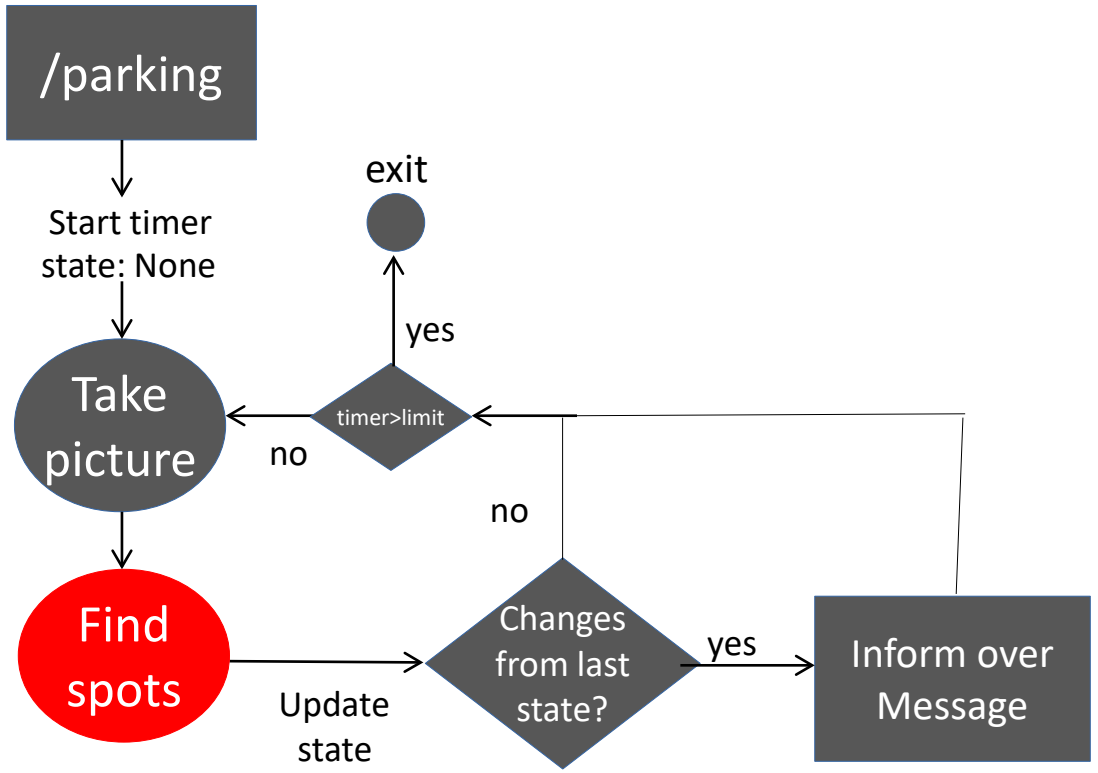
- /start – gives information about itself
- /parking – initializes the parking detection and observes for 5 minutes informing about any changes



Detection Workflow



Detection Workflow



Detection Workflow

Find spots

Image Segmentation

Architecture: Fully-Convolutional Network with a Resnet-101 Backbone
The pre-trained model from **PyTorch Hub** has been trained on a subset of COCO train2017 for semantic segmentation, on the 20 categories that are present in the Pascal VOC dataset.

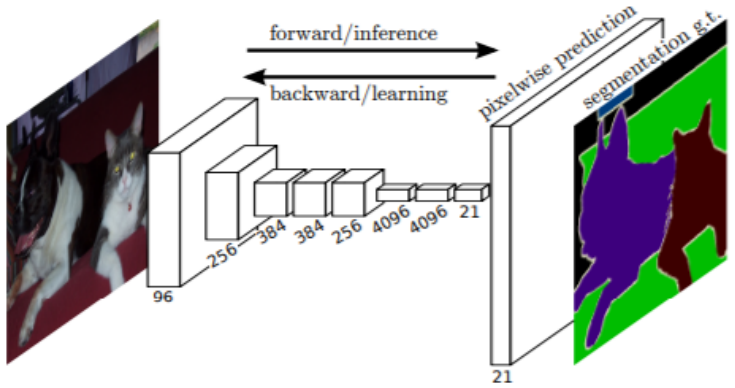
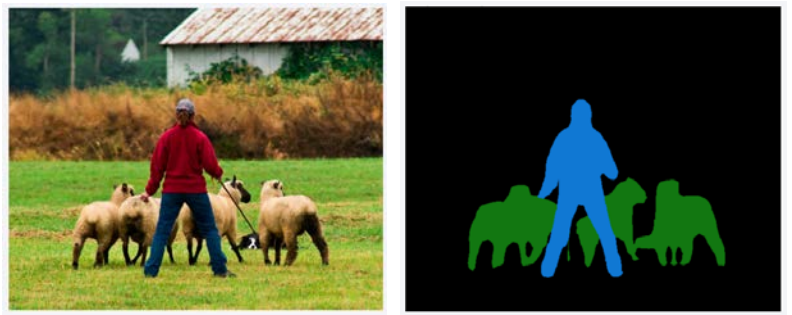


Fig. 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.



Source: https://pytorch.org/hub/pytorch_vision_fcn_resnet101/

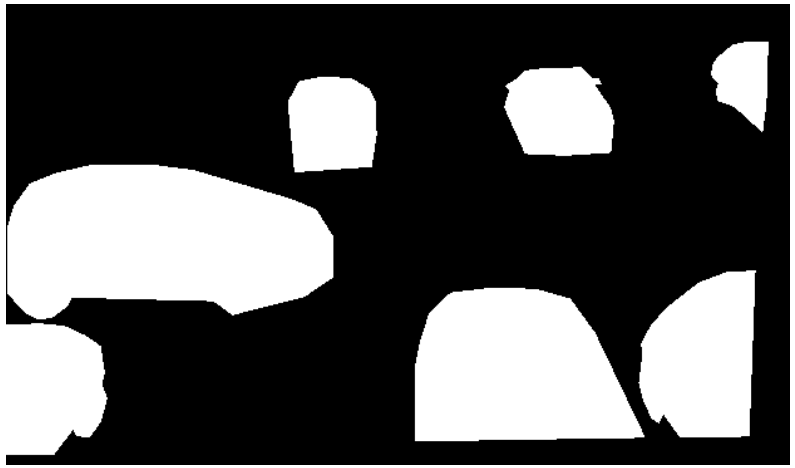
Detection Workflow

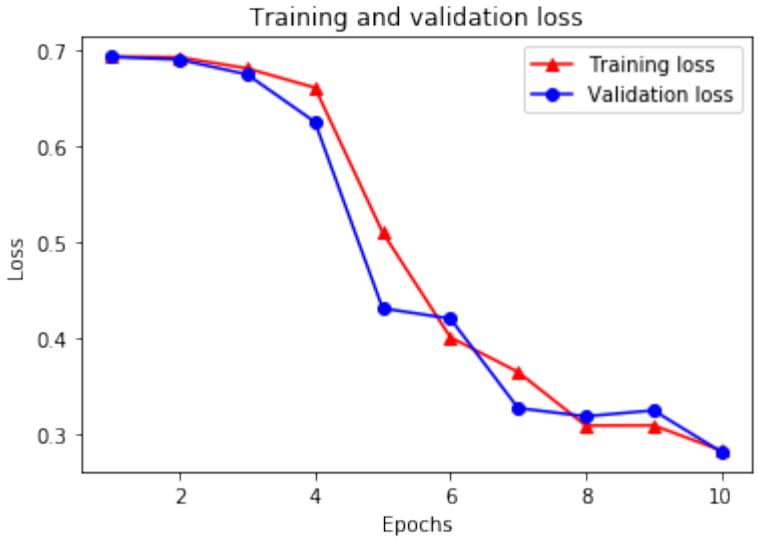
Find spots

Image Segmentation

Architecture: FCN Resnet 101

Training: Fine Tuning on small dataset (~800 images) for 10 Epochs



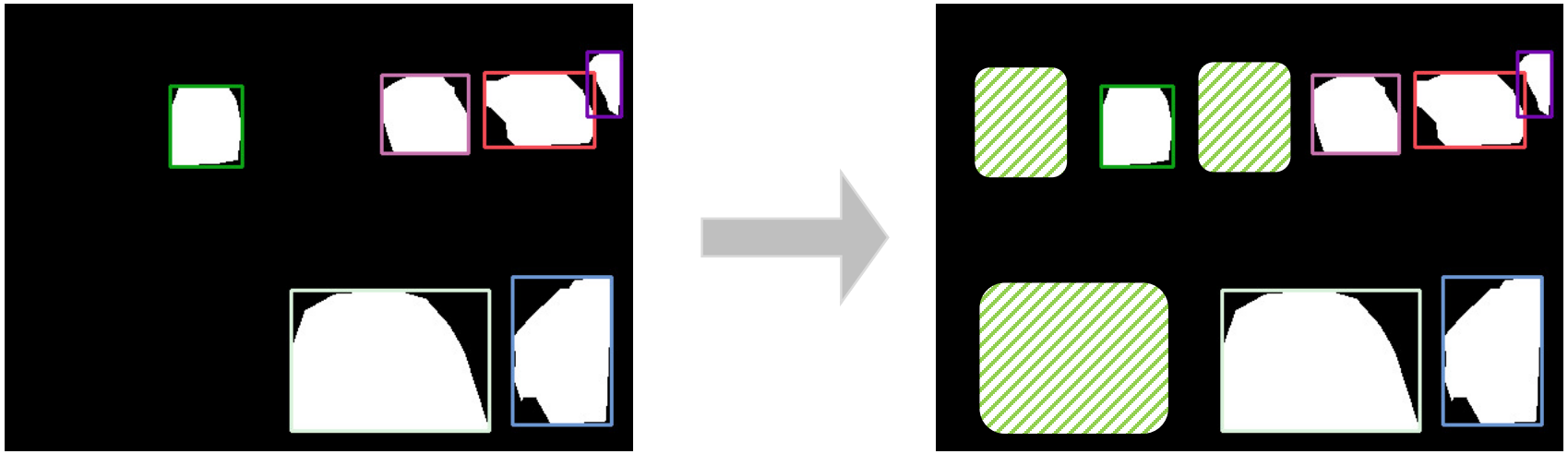


Detection Workflow

Find spots

Detect free spaces

A combination of partially fixed allowed areas and calculation of dynamic sizes was used to determine the amount of available parking spaces.

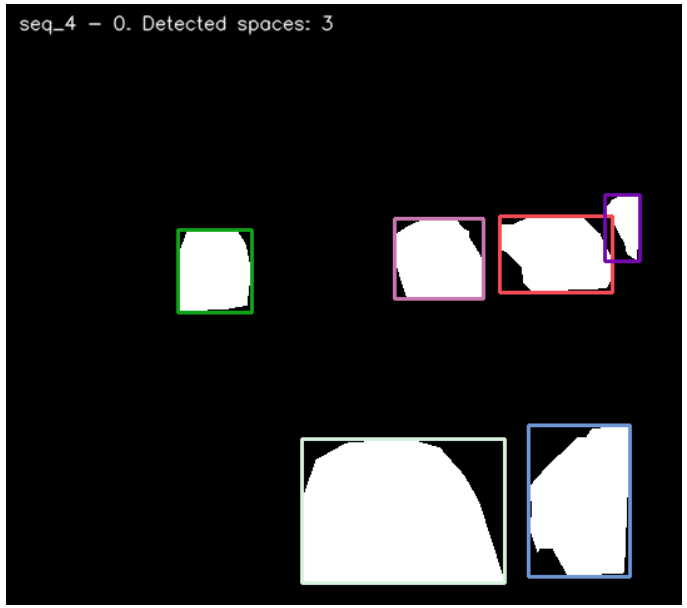


Detection Workflow

Find spots

Detect free spaces

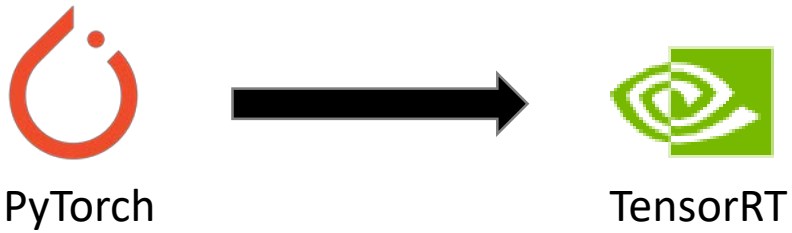
A combination of partially fixed allowed areas and calculation of dynamic sizes was used to determine the amount of available parking spaces.



Deployment

From PyTorch to TensorRT

Our application will run on a JETSON Nano which has GPU support. In order to take the best out of our hardware we will use TensorRT, which is a platform for high-performance deep learning inference that is built on CUDA, NVIDIA‘s parallel programming model.



From PyTorch to TensorRT

Challenges:

- Several conversions.
- PyTorch's computational graphs, from dynamic to static.

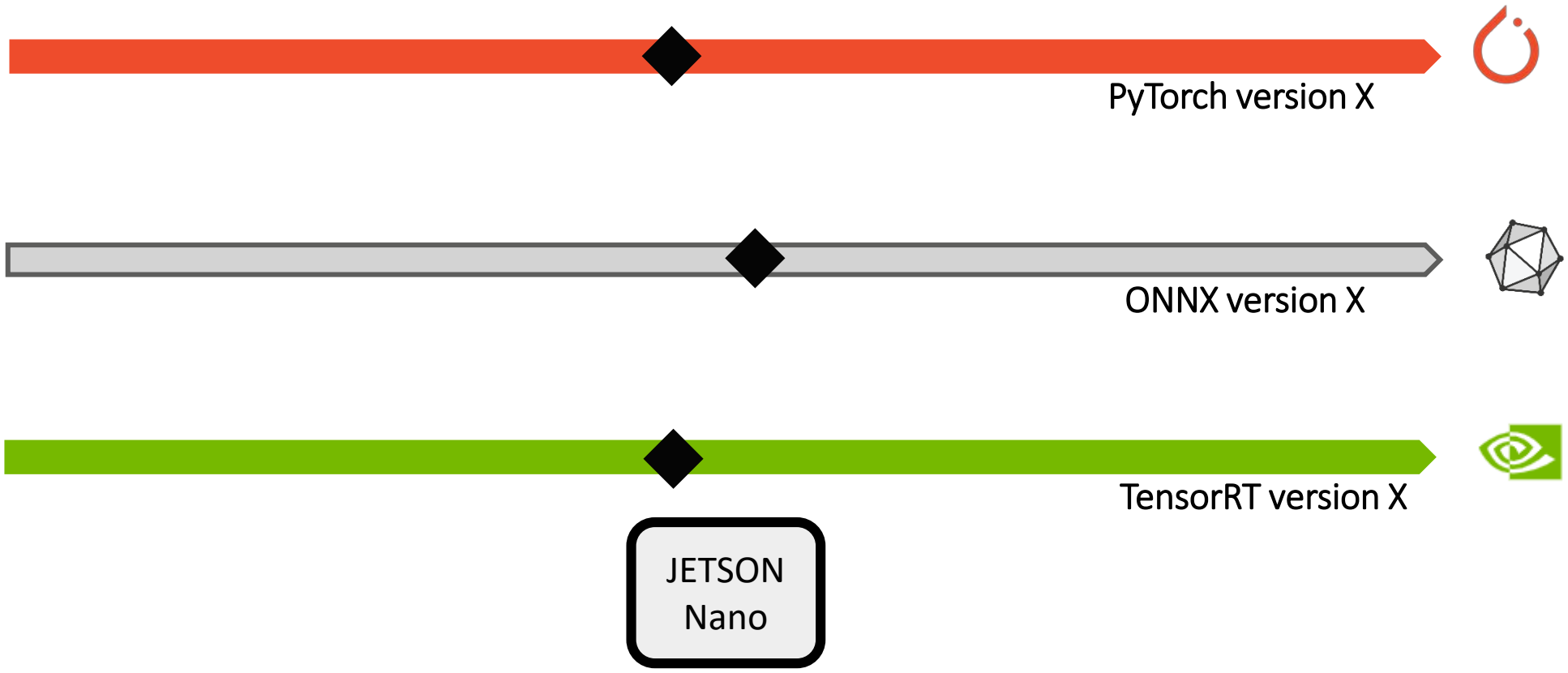
A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

Source: https://pytorch.org/

From PyTorch to TensorRT



PyTorch JIT Compiler and TorchScript

PyTorch JIT is an intermediate representation (IR) for PyTorch called TorchScript. TorchScript is the “graph” representation of PyTorch.

TorchScript is a way to create serializable and optimizable models from PyTorch code. Any TorchScript program can be saved from a Python process and loaded in a process where there is no Python dependency.

PyTorch provides tools to incrementally transition a model from a pure Python program to a TorchScript program that can be run independently from Python, such as in a standalone C++ program. This makes it possible to train models in PyTorch using familiar tools in Python and then export the model via TorchScript to a production environment where Python programs may be disadvantageous for performance and multi-threading reasons.

Source: https://pytorch.org/

PyTorch JIT Compiler and TorchScript

TorchScript provides tools to capture the definition of the model:

- Scripting: takes a function/class and interprets the Python code directly outputting the TorchScript IR.
- Tracing: takes a function and an input, records the executed operations with that input and constructs the IR.

TORCH.ONNX

The ONNX exporter from PyTorch can be both trace-based and script-based.

Source: https://pytorch.org/

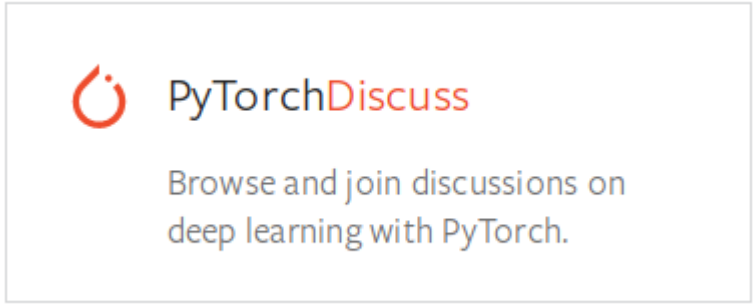
PyTorch JIT Compiler and TorchScript

There are some known limitations for the exporter. If your model contains some of these exceptions you will have to modify it in order to successfully export it.

PyTorch JIT Compiler and TorchScript

There are some known limitations for the exporter. If your model contains some of these exceptions you will have to modify it in order to successfully export it.

BUT… There is a lot of support in the community and there are many people working hard to solve the known issues.



Add Support for ONNX scripting Interpolate with missing shape #29489

Closed lara-hdr wants to merge 3 commits into pytorch:master from lara-hdr:lahaidar/interpolate_shape

Conversation 10 Commits 3 Checks 5 Files changed 3

lara-hdr commented 6 days ago Member

- Add support for missing case where interpolate is exported with missing shape information in scripting
- Add warnings

Reviewers
facebook
BowenBa
houseroa

https://github.com/pytorch/vision/issues/1002

https://github.com/pytorch/vision/pulls

Enable ONNX Test for FasterRcnn #1555

Merged fmassa merged 4 commits into pytorch:master from lara-hdr:lahaidar/enable_faster_rcnn 9 days ago

Conversation 5 Commits 4 Checks 2 Files changed 4

lara-hdr commented 9 days ago Member

Faster Rcnn should now be exportable to ONNX.
The PyTorch version should include commit ebc216a0765d85f345f9a5cd1dfd2ec360de3a52 (any nightly version after Nov 5th).
Opset 11 is the minimum ONNX version supported.
Only a batch size of 1 with fixed image size is supported.

The test test_faster_rcnn() in test_onnx.py has an example of exporting the model;
1 - create an input of valid size to export the model.
2 - run the model with the input then export it by calling torch.onnx.export() with the model and input.
(3- you can optionally test/run the model with ONNX Runtime like in ort_validate().)

Open ONNX model import #116

hadim opened this issue on 5 Nov 2018 · 7 comments

fmassa commented on 6 Nov 2018 Contributor

I think it might be difficult to have our detection models be ONNX-exportable in the near future.

The reason being that they have a few custom operators, and those operators are not in the ONNX standard.
Adding them to the ONNX standard could be possible, but I believe this requires some time and discussion between different teams. Also, having other frameworks implement those operators is yet another story.

fmassa added the needs discussion label on 6 Nov 2018

Open Neural Network Exchange ONNX

ONNX is an open format to represent deep learning models. With ONNX, AI developers can more easily move models between state-of-the-art tools and choose the combination that is best for them.

ONNX is developed and supported by a community of partners.

Source: https://onnx.ai/

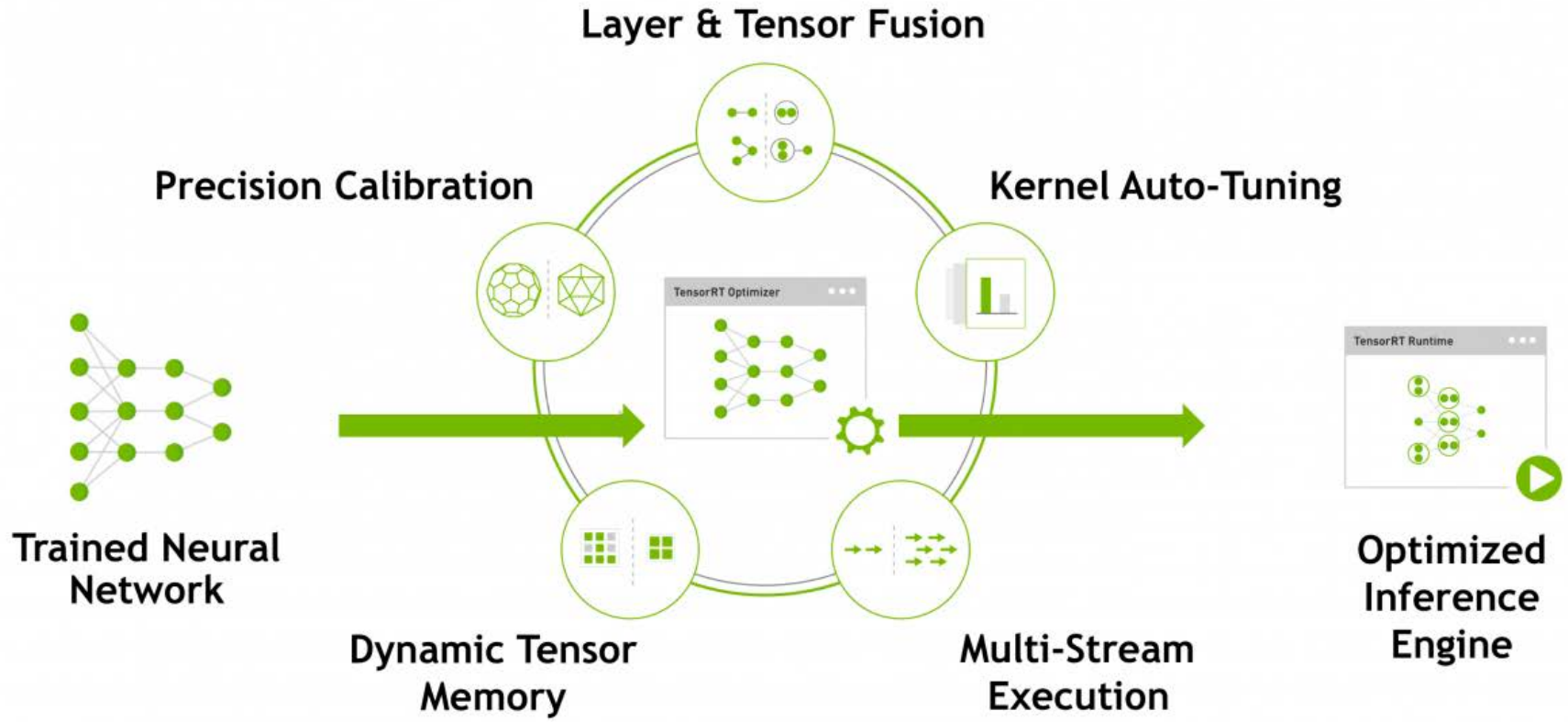
TensorRT

NVIDIA TensorRT™ is a platform for high-performance deep learning inference. It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications.

With TensorRT, you can optimize neural network models trained in all major frameworks, calibrate for lower precision with high accuracy, and finally deploy to hyperscale data centers, embedded, or automotive product platforms.

Source: https://developer.nvidia.com/tensorrt

TensorRT



Source: https://developer.nvidia.com/tensorrt

TensorRT backend for ONNX

onnx_tensorrt parses ONNX models for execution with TensorRT.

https://github.com/onnx/onnx-tensorrt

You can:

- Directly convert an ONNX Model to serialized TensorRT engines.
- Use ONNX Python backend to run the TensorRT engine in Python.

Final System Setup

- torch: 1.2.0
- torchvision: 0.4.0
- torch.onnx export opset9
- Model from current torch.hub:

```
model = torch.hub.load('pytorch/vision', 'fcn_resnet101', pretrained=True, force_reload=False)
```

Final System Setup

Some modifications had to be made to the model in order to export it:

- Remove interpolation
- The output of the Model has to return a Tensor; no dict, remove aux_output
- Remove unknown (new in torch 1.3.0) JIT functions (i.e.: @torch.jit.unused)

```
my_model = MyFCNResNet101(model)
my_model.eval()
output_data = my_model(x)
output_data_interp = F.interpolate(output_data,
                                   size=(224, 224),
                                   mode='bilinear',
                                   align_corners=False)
```

```
class MyFCNResNet101(nn.Module):
    def __init__(self, fcn):
        super(MyFCNResNet101, self).__init__()
        self.fcn = fcn


    def forward(self, x):
        input_shape = (224, 224)

        features = self.fcn.backbone(x)


        x = features["out"]
        x = self.fcn.classifier(x)
        #x = F.interpolate(x, size=input_shape, mode='bilinear', align_corners=False)

        return x
```

Performance Gain

Speed up of 1.37x

```
pytorch with interp (on GPU): iters/sec 1.25986337847
trt with interp (on GPU): iters/sec 1.72348039751
speedup: 1.3679899x

pytorch without interp: iters/sec 1.26707609834
trt without interp: iters/sec 2.17546154538
speedup: 1.7169x
```

All roads lead to Rome…

I followed these steps for this implementation but there are many libraries that offer different approaches, i.e. going directly from PyTorch to TensorRT:

- **torch2trt** (https://github.com/NVIDIA-AI-IOT/torch2trt)
- **TensorRT Python API** where you can create the inference engine with TensorRT by creating the network definition from scratch, which involves replicating the network architecture using the TensorRT Python API and then copying the weights from PyTorch.
- **jetson-inference** (https://github.com/dusty-nv)
- …

Thank you…

- Microsoft for hosting the event.
- Thomas Viehmann for the organization.
- Piotr Bialecki for the insightful discussions.