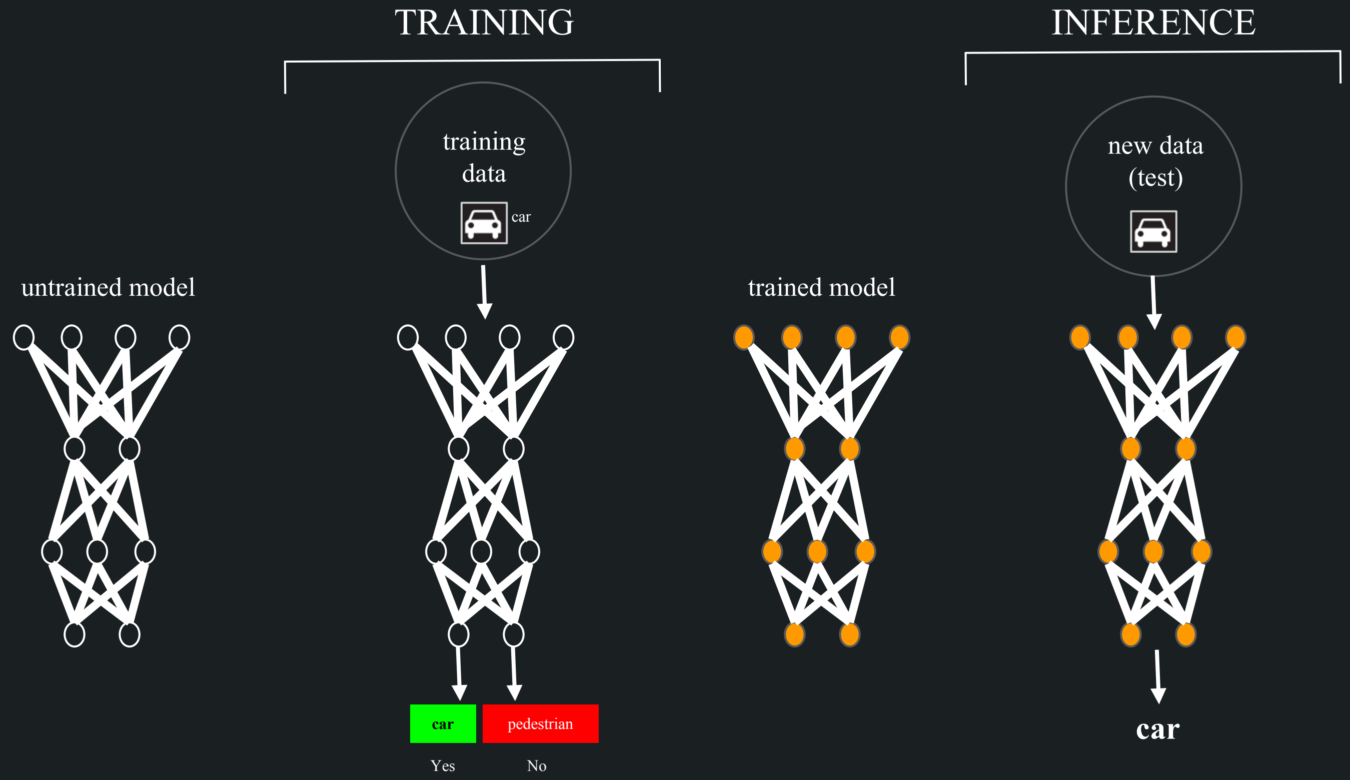
Deep Learning on Everyday Devices

Amir Alush, CTO & co founder, IMVC 2018

Brodmann17

Deep Learning “Cycle”



Brodmann17

Inference on Everyday devices

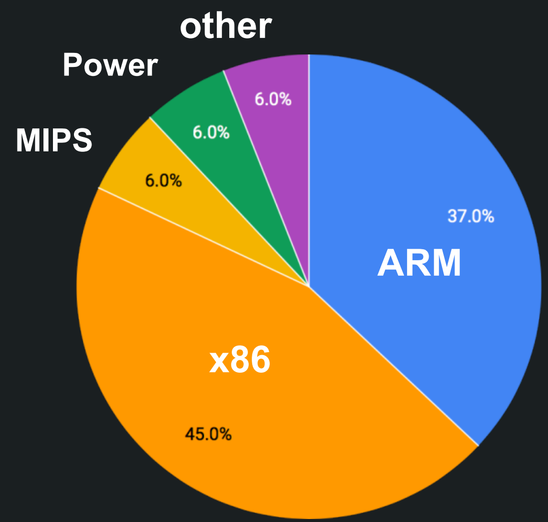
8-Core ARM

Qualcomm 6-core

Snapdragon 820

TBD

Embedded processors market share (Source: AMD 2016)



Qualcomm Kryo (2.15GHz): 17.2 GFLOPS, 2.05Watts
TitanXP: 12 TeraFlops, 250W

Brodmann

Inference on the Edge

Motivation

- Low Latency
- Keep Privacy
- Small/no Bandwidth
- Utilizing Existing HW

Challenges

- NN High complexity
- Low HW resources
- Complex porting

The Deep Learning Stack

Algorithms
NN Architectures, Meta-Architectures

Frameworks
TF, CAFFE, PyTorch, MXNet

Engines
TensorRT, Core ML, SNPE

Primitives Libraries
BLAS, NNPACK,CUDNN*

HARDWARE
GPU, CPU, TPU, FPGA, DSP, ASIC

It’s the algorithms!

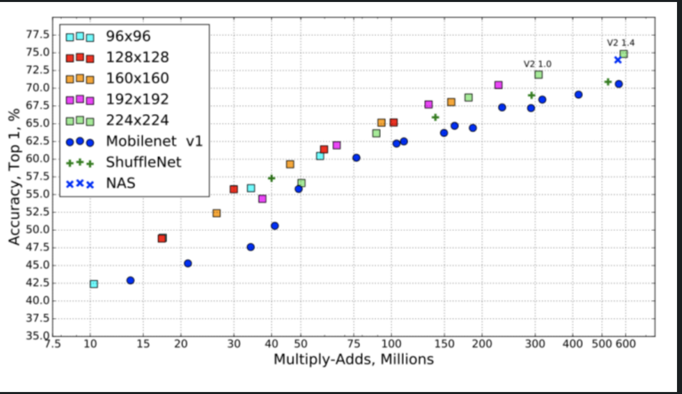
Efficient algorithms:

1. More AI on any processor
2. Critical for everyday devices with low power processors

Speed-accuracy tradeoff

Running **off-the-shelf** DL algorithms on the edge requires sacrificing accuracy. The tradeoffs:

1. Reduce input resolution → reduce accuracy
2. Reduce model size (backbone) → reduce accuracy
3. Reduce model bit precision → reduce accuracy?
4. Less accurate algorithm



Model	Top-1 accuracy	Num. Params.
VGG-16	71.0	14,714,688
MobileNet	71.1	3,191,072
Inception V2	73.9	10,173,112
ResNet-101	76.4	42,605,504
Inception V3	78.0	21,802,784
Inception Resnet V2	80.4	54,336,736

	mAP	GPU msecs
faster-rcnn + resnet 101 + high resolution	35.6	140
ssd + mobilenetV1 + low resolution	18.8	50

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Application", Howard et al. 2017
nverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation", Sandler et al. 2018

Reduce network size (pruning)

- Should be structured, non structured is usually not HW supported
- Requires re-training
- How effective on already small models?
- Can hurt accuracy!

Faster-RCNN	Baseline	25 %	50%	75%
mAP	0.66	0.655	0.648	0.530
fps	7.5	10	13	16

Table 6: Object detection results when directly pruning (random) a fully trained Faster-RCNN model.

Heuristics	25 %	50%	75%
Random	**0.647**	0.600	0.505
Mean Activation	**0.647**	0.601	0.489
Entropy	0.635	0.584	0.501
Scaled Entropy	0.640	0.593	0.507
l_1-norm	0.628	**0.608**	**0.520**
APoZ	0.646	0.598	0.514
Sensitivity	0.636	0.592	0.485

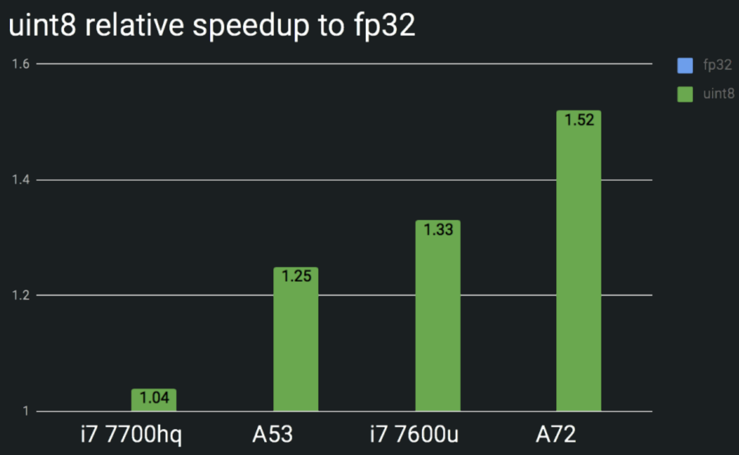
Table 5: Object detection results obtained by plugging-in different pruned VGG-16 models into Faster-RCNN.

“Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks”, Mittal 2018

Brodmann17

Reduce model bit precision (quantization)

- Reduce from 32 bits to 8/4/2/1 bits ?
- Activations & Weights are within a narrow range
- Networks are robust to small changes
- 8 bits:
    - Need to be supported in processors instructions
    - Reduces DRAM bandwidth (more in the SRAM)
    - Supported natively by various engines
- Below 8 bits:
    - Accuracy drops
    - Not supported in existing HW



Brodmann

How to deploy your models?

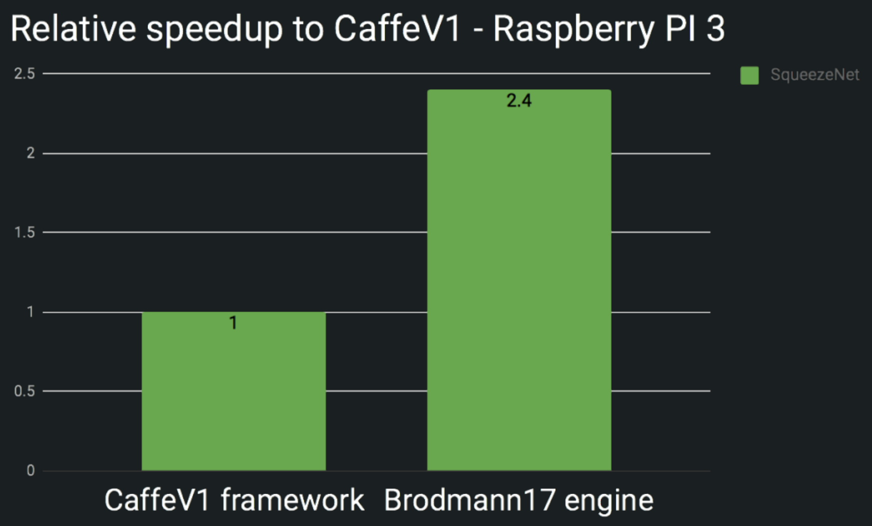
Training Frameworks

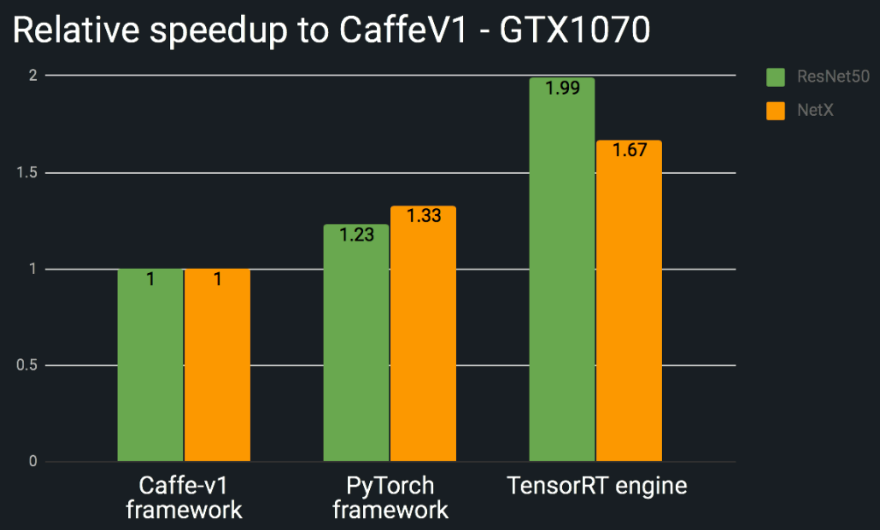
- Research flexibility:
  - Fast POC from idea to results
  - Easy to extended: new data loaders, layers (loss, operations)
  - Easy debugging
- Good training speed (+ parallelization)
  - Fast research iterations
- Large active community:
  - Explore new research ideas fast
- Personal flavour
- Portability should not a factor

Inference Engines

- Optimized for latency (forward pass)
- Should be efficient on a specific hardware
  - Takes advantage of specific hw instructions
- Little / no overhead
- Efficient working memory
- Small code size
- Support all of your NN layers

Training frameworks vs Inference Engines





Brodmann17

Build your own Inference Environment?

1. **Use the engines (TensorRT, SNPE...):**
    - Faster to production (if the conversion works out of the box)
    - Current generation has some limitations and minimal tech support
    - Not as mature as the training frameworks or DL primitives libraries
    - Not all NN capabilities are supported
2. **Write your own inference engine:**
    - Slower to production, but, with low risk
    - Use the DL primitives libraries (CuDNN, BLAS, NNPACK..) they are more matures than the engines and they are optimized!
    - You'll need to write your own logic on top!
    - You're in control and can adjust to your needs

Brodmann17

Brodmann17 R&D workflow

Data

collect
clean
annotate

Research

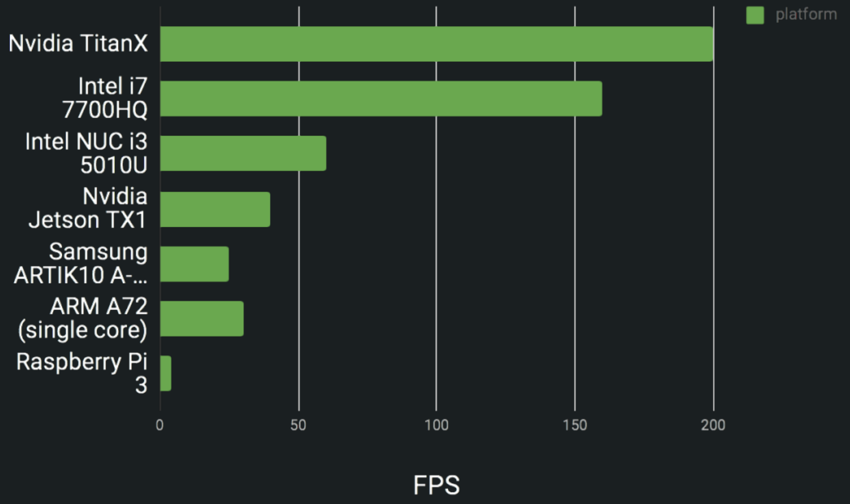
inventing
Improving
training (PYTORCH)

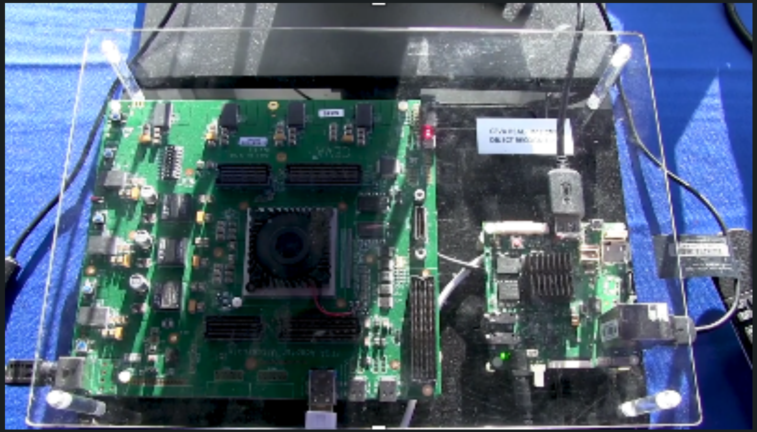
Deploy

In-house inference engine
*also using existing engines
or primitives:

VeriSilicon CDNN2 SNPE
arm TensorRT

Brodmann17 Use Case - IoT

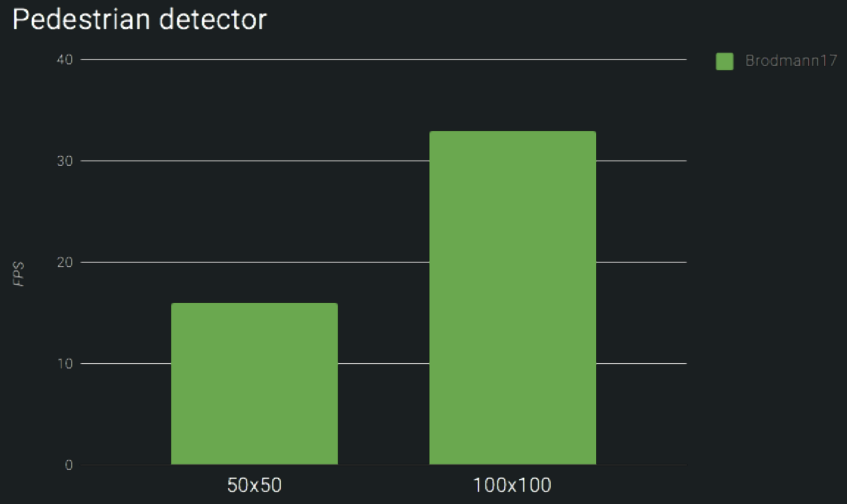




- MWC 2018 recent cooperation with intuition robotics & arm
- Embedded World 2018 with INTRINSYC
- CES 2018 with CEVA

Brodmann17

Brodmann17 Use Case - Smart City

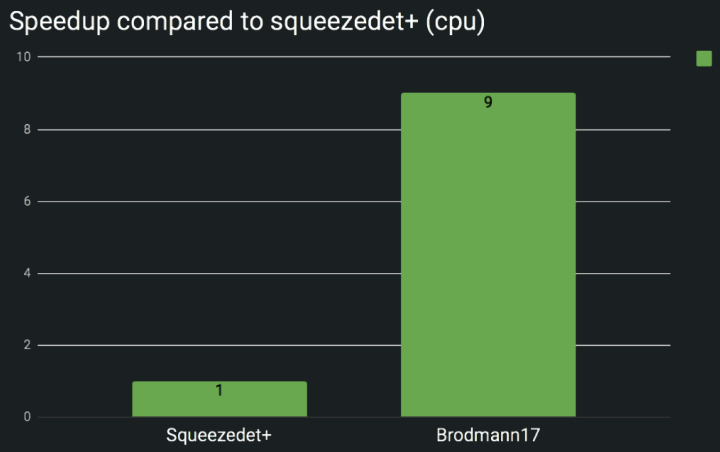


* 1 CPU core

Brodmann17 Use Case - Adas

Benchmarking on the KITTI dataset car detection

	Easy	Medium
Brodmann17	**90.63**	**89**
Squeezedet+	90.4	87.1



* 1 CPU core

Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite, Geiger et al, 2012

Summary

1. motivation and challenges in NN edge processing
2. Speed accuracy tradeoffs made today
3. Brodmann17 key design principles for keeping efficiency and accuracy on the edge
4. key considerations when choosing your inference environment
5. Brodmann17 example use cases