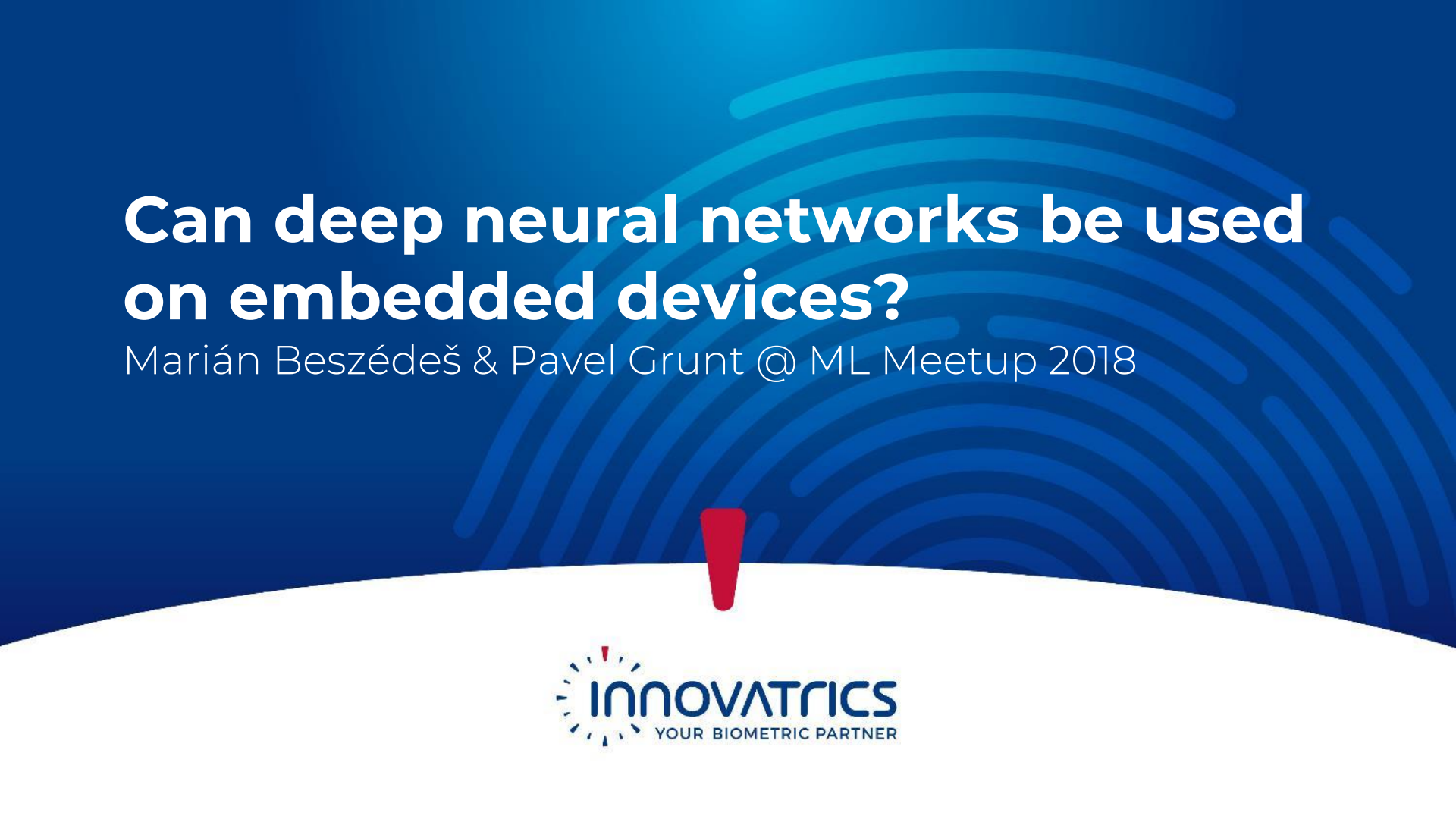
# Can deep neural networks be used on embedded devices?

Marián Beszédeš & Pavel Grunt @ ML Meetup 2018



INNOVATRICS
YOUR BIOMETRIC PARTNER

# Agenda

- Us & Innovatrics
- Motivations
- Optimizations
  - design fast NN architectures
  - slimming designed NN
- Tools
- Lessons learned

# About us ...

## Computer Vision Team @ Innovatrics

- Facial biometrics
- 10 team members
- Brno / Bratislava

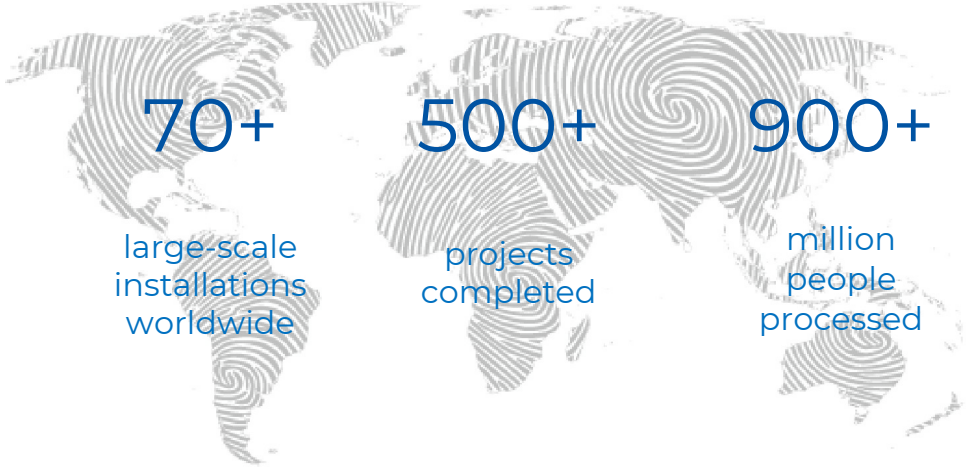
## Marián Beszédeš

- Team leader
- 5 Years

## Pavel Grunt

- Embedded devices optimization specialist
- 1 Year

# About Innovatrics

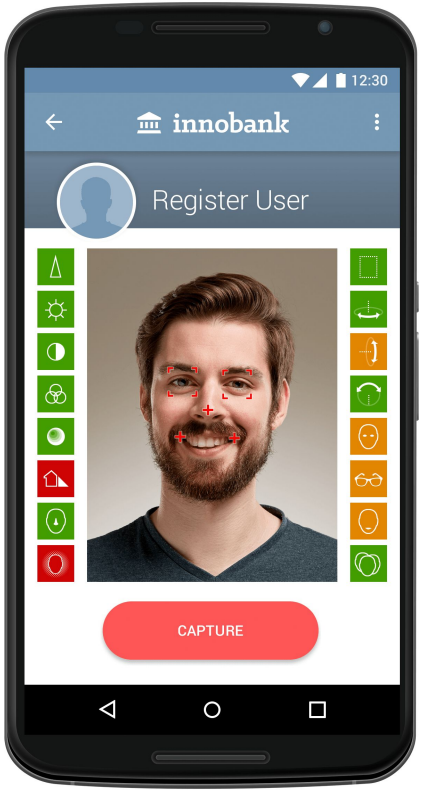



- 100 Employees
- 70+ countries
- 20+ industry awards

- The World's fastest AFIS
  - Automated Fingerprint Identification System
  - 1,04+ billion fingerprint matches in second
  - 99,98% accuracy
  - Awarded accuracy / speed – NIST VPVTE / Minex III / PFT.
- Cutting edge face detection / recognition technology
  - NIST evaluation - top 10 face verification vendors

# About IFace SDK

- Face detection
- Facial landmarks detection
- Face attributes recognition
  - mouth status,  eyes status, ...
- Face tracking (people counting / tracking)
- Face segmentation
- Face liveness
- Face recognition
  - Age / Gender
  - Verification / Identification




Demo

# Why deep networks on “Small” devices ?

Standard “deep NN” environment

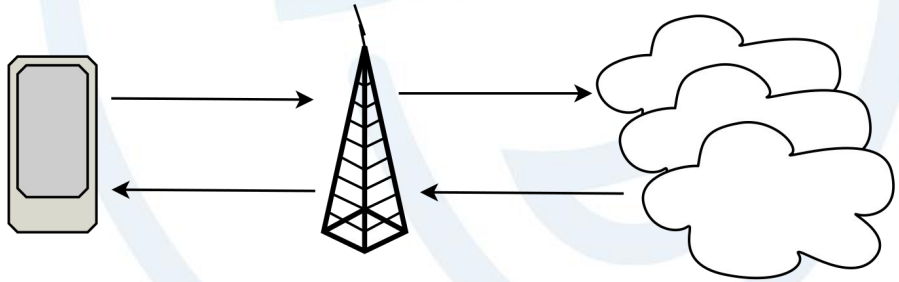
- **Server in cloud** with lot of GPUs.

BUT :

- Cloud is not accessible everywhere
- Data can be too big to transfer them
- Real-time operations are necessary
  - Robotics / Self-driving cars / Drones / Augmented reality
- Devices without GPU are cheaper
- Where they store your personal data ?

Solution:

- Do it on the device !!!



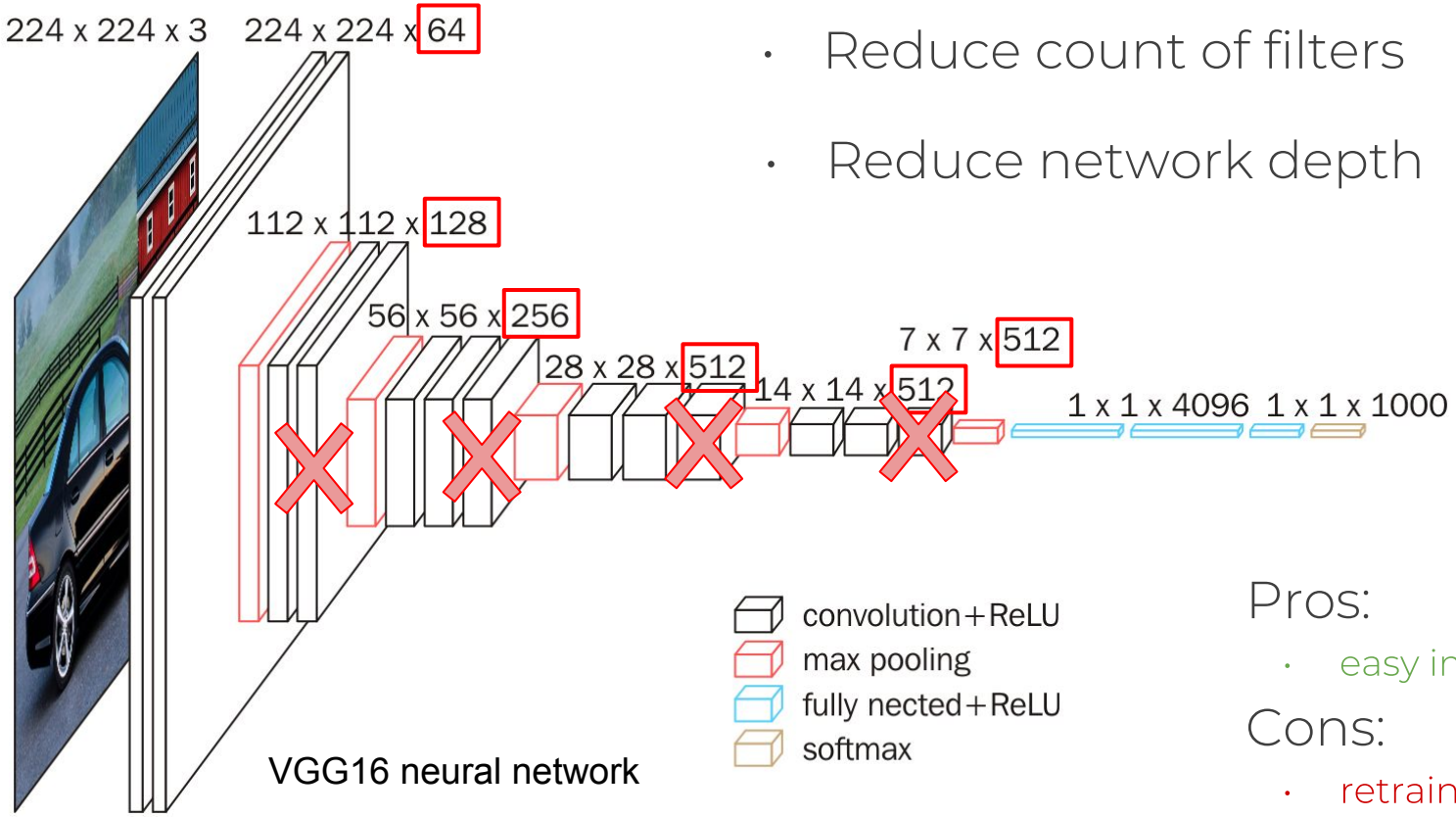
# Cloud vs Mobile neural nets

INNOVATRICS
YOUR BIOMETRIC PARTNER

Mobile (embedded) devices need special “Mobile Neural Nets”

| Neural Network type | Cloud NN | Mobile NN |
|---|---|---|
| *Runtime* | Fast on GPU, Slow on CPU | Super fast on GPU, Fast on CPU |
| *Energy usage* | High | Low |
| *Size* | 0.1GB - 0.5GB | 1 - 5 MB |
| *Training* | Slow | Fast |
| *Update* | Direct | Over the air |
| *Accuracy* | High | Lower |

# (Very) Simple tricks




- Reduce count of filters
- Reduce network depth

Pros:

- easy implementation

Cons:

- retrain from scratch

# Simple tricks ...

| Layer | #Weights |
|---|---|
| conv1 | 35K |
| conv2 | 307K |
| conv3 | 885K |
| conv4 | 663K |
| conv5 | 442K |
| fc6 | 38M |
| fc7 | 17M |
| fc8 | 4M |
| Total | 61M |

96% of all parameters

AlexNet 240 MB

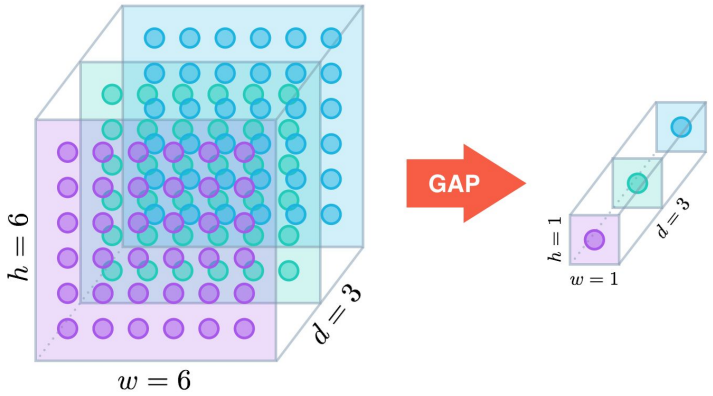
| Layer | #Weight |
|---|---|
| conv1_1 | 2K |
| conv1_2 | 37K |
| conv2_1 | 74K |
| conv2_2 | 148K |
| conv3_1 | 295K |
| conv3_2 | 590K |
| conv3_3 | 590K |
| conv4_1 | 1M |
| conv4_2 | 2M |
| conv4_3 | 2M |
| conv5_1 | 2M |
| conv5_2 | 2M |
| conv5_3 | 2M |
| fc6 | 103M |
| fc7 | 17M |
| fc8 | 4M |
| Total | 138M |

90% of all parameters

VGG-16 552 MB

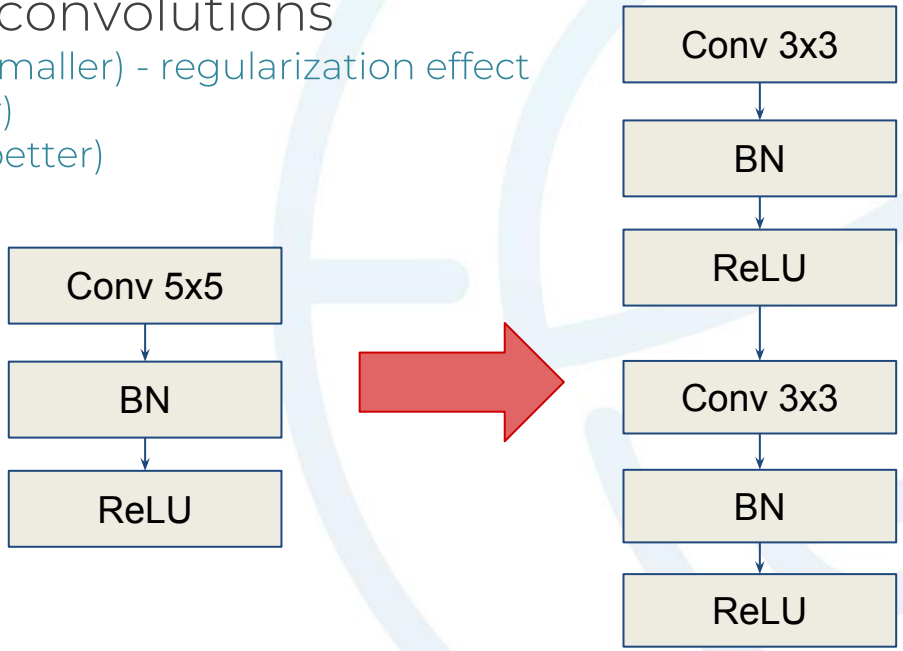
Most parameters (size) in fully connected layers

- replace them by global Average Pooling

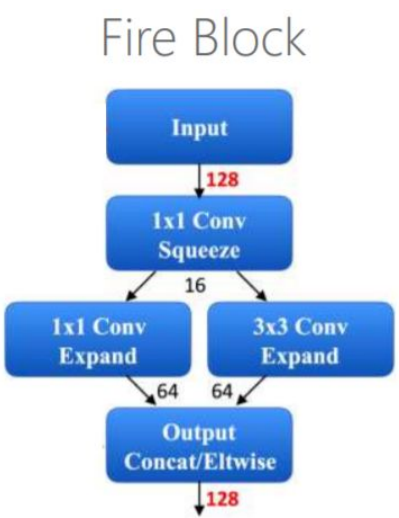


# Simple tricks ...

- Most time is spent in convolutions
- Replace large (5x5, 7x7) convolutions with stacks of 3x3 convolutions
  - Fewer parameters (smaller) - regularization effect
  - Less compute (faster)
  - More non-linearity (better)

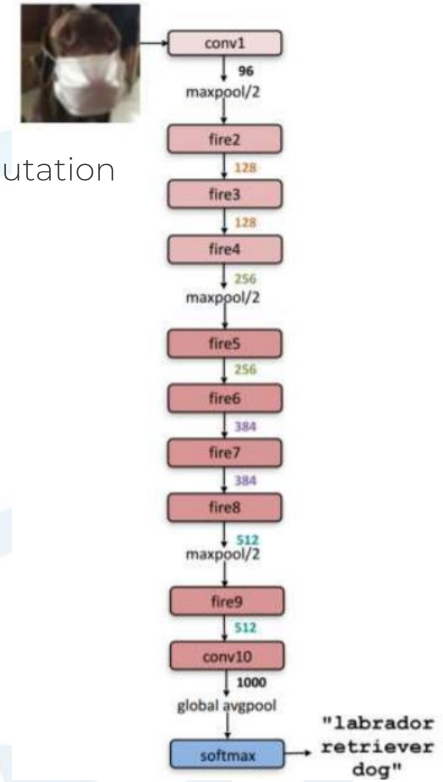


# SqueezeNet tricks



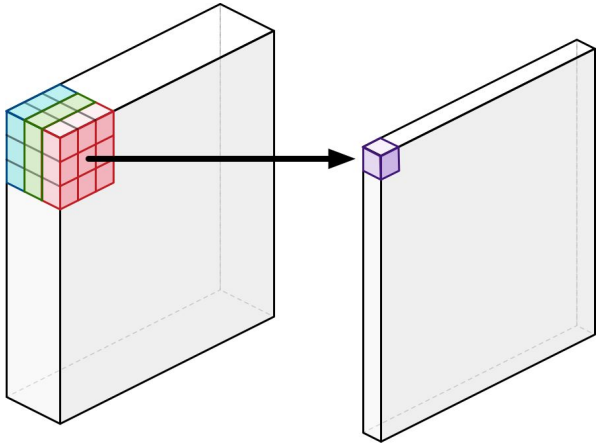

- Fire Block Architecture
  - Squeeze Blocks
    - Squeeze number of channels to reduce computation in expand block
    - Use 1x1 filters (point-wise) instead of 3x3
      - Reduction of computations 9x
    - Apply nonlinearity (RELU)
  - Expand Block
    - Do convolution using 1x1 and 3x3 filters
    - Apply one more nonlinearity

- SqueezeNet base vs AlexNet
  - 4.8 MB vs 240 MB (50x smaller)
  - Same accuracy
  - Slower than AlexNet in runtime
    - Downsampling (pooling) done late in network
    - Big memory footprint

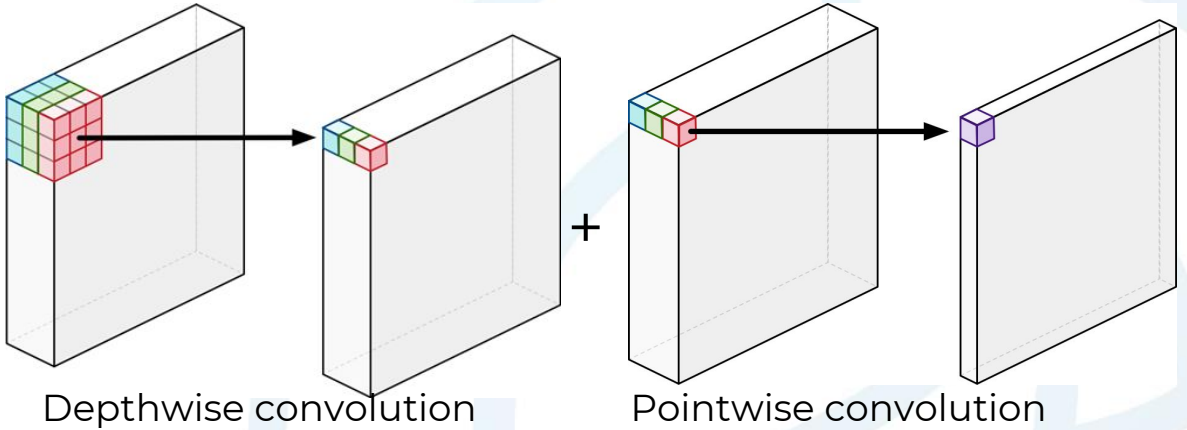



Landola et. al, SqueezeNet: AlexNet level accuracywith 50x fewer parameters and <0.5MB model size, 2016
http://github.com/DeepScale/SqueezeNet

# MobileNets tricks



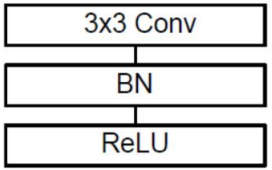
Regular convolution

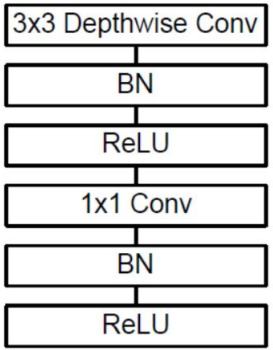


Depthwise Separable convolution

Depthwise convolution

Pointwise convolution




| Filter size | Filter count | Computation reduction |
|---|---|---|
| 3 | 32 | 7,0 x |
| 3 | 64 | 7,9 x |
| 3 | 128 | 8,4 x |
| 5 | 32 | 14,0 x |
| 7 | 32 | 19,4 x |

Howard et. al, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Google, 2017

# MobileNets tricks

**Resolution multiplier** $\rho$ - shrinks size of input image

**Width multiplier** $\alpha$ - shrinks number of filters

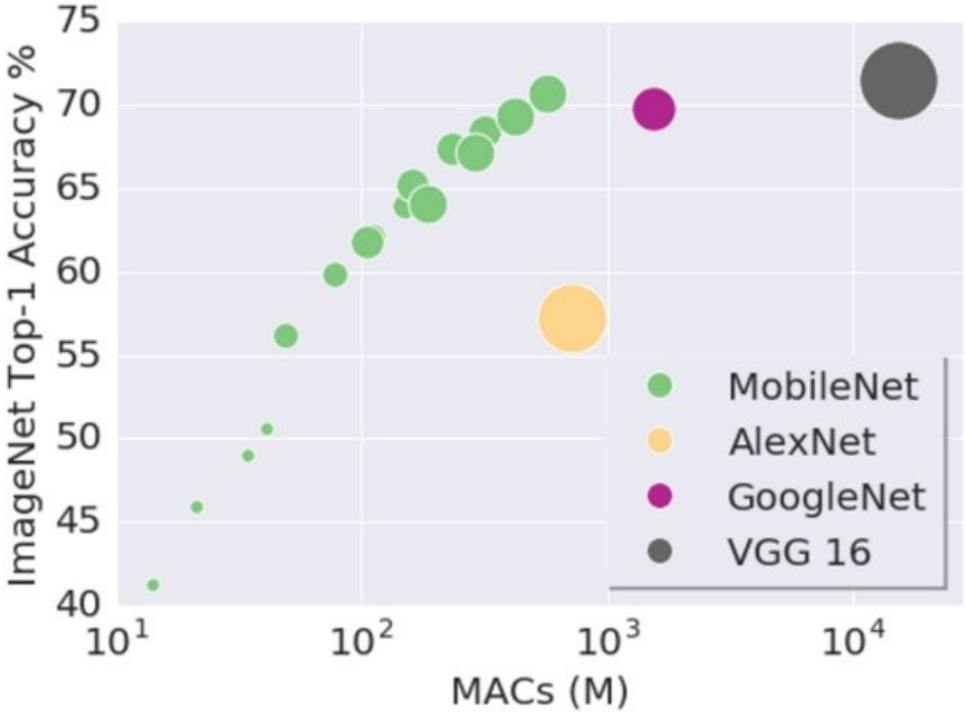
| Input size | Operator | Filters count | Module Repeated | Stride |
|---|---|---|---|---|
| 224 x 224 x 3 | conv2d | 32 | 1 | 2 |
| 112 x 112 x 32 | conv DWS | 64 | 1 | 1 |
| 112 x 112 x 64 | conv DWS | 128 | 1 | 2 |
| 56 x 56 x 128 | conv DWS | 128 | 1 | 1 |
| 56 x 56 x 128 | conv DWS | 256 | 1 | 2 |
| 28 x 28 x 256 | conv DWS | 256 | 1 | 1 |
| 28 x 28 x 256 | conv DWS | 512 | 1 | 2 |
| 14 x 14 x 512 | conv DWS | 512 | 5 | 1 |
| 14 x 14 x 512 | conv DWS | 1024 | 1 | 2 |
| 7 x 7 x 1024 | conv DWS | 1024 | 1 | 1 |
| 7 x 7 x 1024 | avgpool 7x7 | - | 1 | - |
| 1 x 1 x 1024 | fullconnect | k | - | - |

- Optimized architectures

| Layer/Modification | Million Mult-Adds | Million Parameters |
|---|---|---|
| Convolution | 462 | 2.36 |
| Depthwise Separable Conv | 52.3 | 0.27 |
| $\alpha = 0.75$ | 29.6 | 0.15 |
| $\rho = 0.714$ | 15.1 | 0.15 |

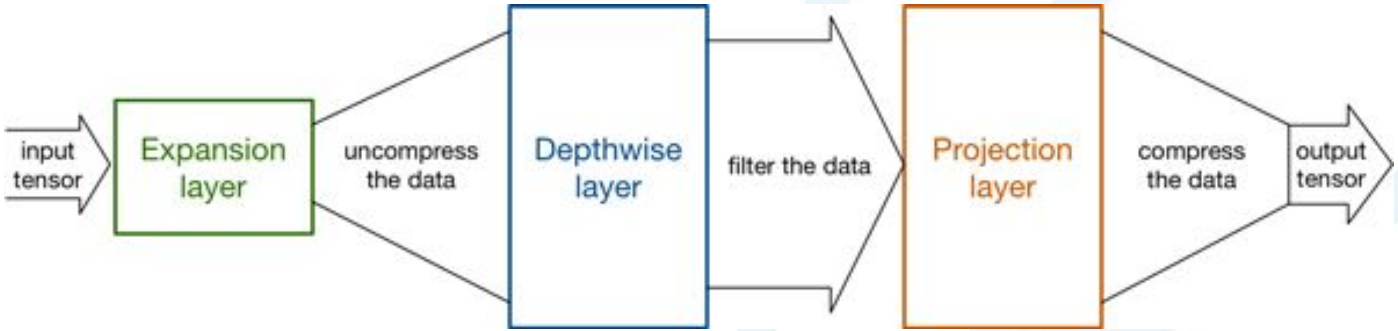
| Model | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| 0.50 MobileNet-160 | 60.2% | 76 | 1.32 |
| Squeezenet | 57.5% | 1700 | 1.25 |
| AlexNet | 57.2% | 720 | 60 |

# MobileNets
# Accuracy / Operations / Size comparison



# MobileNets v2 tricks

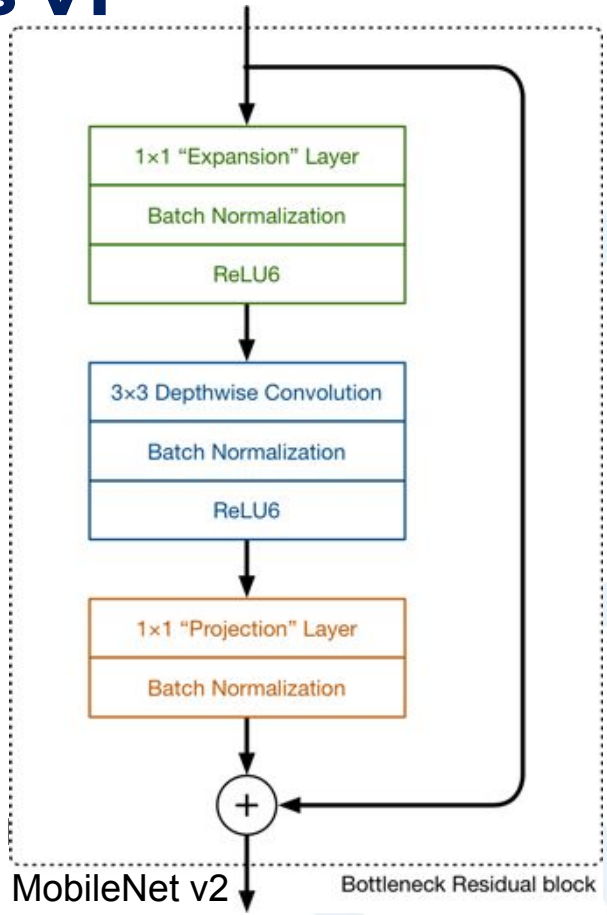
- Bottleneck residual block

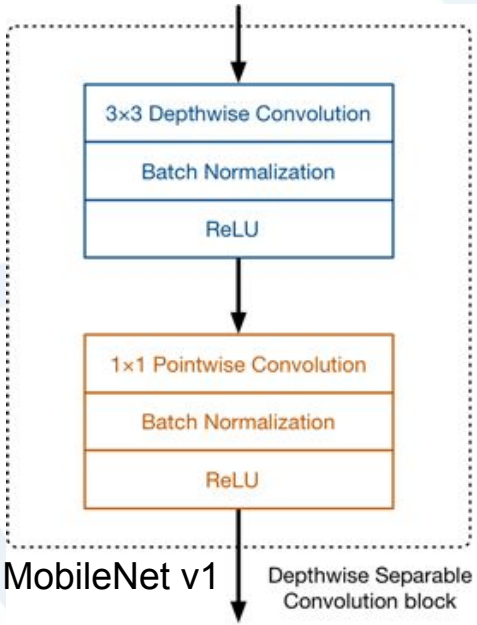



Sandler et. al, MobileNetV2: Inverted Residuals and Linear Bottlenecks, Google, 2018

# MobileNets v2 vs v1

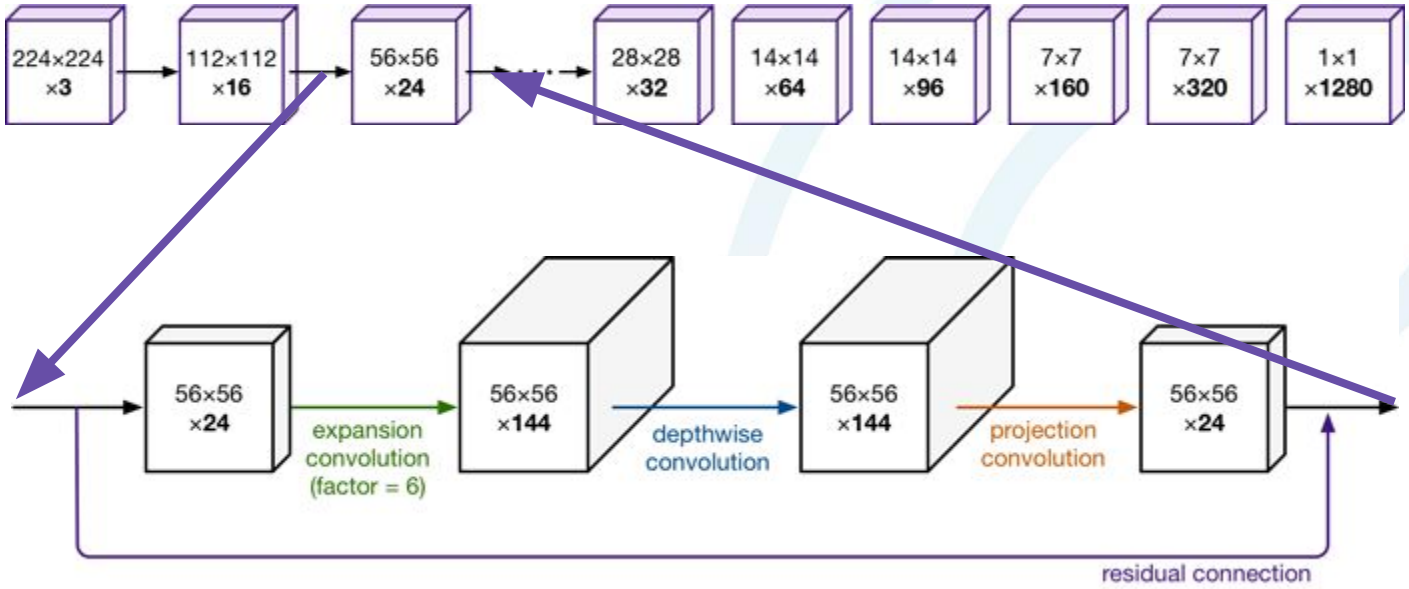
- Residual connections (inspired by ResNets)
- Small memory footprint
- Faster
- Smaller




Sandler et. al, MobileNetV2: Inverted Residuals and Linear Bottlenecks, Google, 2018

# MobileNets v2 tricks



# MobileNet v1 vs v2

| Input size | Operator | Filters count | Module Repeated | Stride |
|---|---|---|---|---|
| 224 x 224 x 3 | conv2d | 32 | 1 | 2 |
| 112 x 112 x 32 | conv DWS | 64 | 1 | 1 |
| 112 x 112 x 64 | conv DWS | 128 | 1 | 2 |
| 56 x 56 x 128 | conv DWS | 128 | 1 | 1 |
| 56 x 56 x 128 | conv DWS | 256 | 1 | 2 |
| 28 x 28 x 256 | conv DWS | 256 | 1 | 1 |
| 28 x 28 x 256 | conv DWS | 512 | 1 | 2 |
| 14 x 14 x 512 | conv DWS | 512 | 5 | 1 |
| 14 x 14 x 512 | conv DWS | 1024 | 1 | 2 |
| 7 x 7 x 1024 | conv DWS | 1024 | 1 | 1 |
| 7 x 7 x 1024 | avgpool 7x7 | - | 1 | - |
| 1 x 1 x 1024 | fullconnect | k | - | - |

MobileNet v1 architecture
28 layers, 4.2M parameters, 575M MAdds
70.6 % Top1 ImageNet Acc, 113 ms on Pixel1

| Input size | Operator | Expansion Factor | Filters count | Module Repeated | Stride |
|---|---|---|---|---|---|
| 224 x 224 x 3 | conv2d | - | 32 | 1 | 2 |
| 112 x 112 x 32 | bootleneck | 1 | 16 | 1 | 1 |
| 112 x 112 x 16 | bootleneck | 6 | 24 | 2 | 2 |
| 56 x 56 x 24 | bootleneck | 6 | 32 | 3 | 2 |
| 28 x 28 x 32 | bootleneck | 6 | 64 | 4 | 2 |
| 14 x 14 x 64 | bootleneck | 6 | 96 | 3 | 1 |
| 14 x 14 x 96 | bootleneck | 6 | 160 | 3 | 2 |
| 7 x 7 x 160 | bootleneck | 6 | 320 | 1 | 1 |
| 7 x 7 x 320 | conv2d 1x1 | - | 1280 | 1 | 1 |
| 7 x 7 x 1280 | avgpool 7x7 | - | - | 1 | - |
| 1 x 1 x 1280 | conv2d 1x1 | - | k | - | - |

MobileNet v2 architecture
55 layers, **3.4M** parameters, **300M** MAdds
**72.0 %** Top1 ImageNet Acc, **75** ms on Pixel1

# MobileFaceNets

- MobileNet v2 building blocks
- Optimized topology
- Tuned for face recognition

PERFORMANCE COMPARISON AMONG MOBILE MODELS TRAINED ON CASIA-WEBFACE

| Network | LFW Acc. | AgeDB-30 Acc. | Params | Speed (CPU) |
|---|---|---|---|---|
| MobileNetV1 | 98.63% | 88.95% | 3.2M | 60ms |
| MobileNetV2 | 98.58% | 88.81% | 2.1M | 49ms |
| **MobileFaceNet** | **99.28%** | **93.05%** | **0.99M** | 24ms |
| MobileFaceNet (112 ×96) | 99.18% | 92.96% | 0.99M | 21ms |
| MobileFaceNet (96 ×96) | 99.08% | 92.63% | 0.99M | **18ms** |

PERFORMANCE COMPARISON WITH PREVIOUS PUBLISHED FACE VERIFICATION MODELS ON LFW

| Method | Training Data | #Net | Model Size | LFW Acc. |
|---|---|---|---|---|
| Deep Face | 4M | 3 | - | 97.35% |
| DeepFR | 2.6M | 1 | 0.5GB | 98.95% |
| DeepID2+ | 0.3M | 25 | - | 99.47% |
| Center Face | 0.7M | 1 | 105MB | 99.28% |
| DCFL | 4.7M | 1 | - | 99.55% |
| SphereFace | 0.49M | 1 | - | 99.47% |
| CosFace | 5M | 1 | - | 99.73% |
| ArcFace (LResNet100E-IR) | 3.8M | 1 | 250MB | **99.83%** |
| FaceNet | 200M | 1 | 30MB | 99.63% |
| ArcFace (LMobileNetE) | 3.8M | 1 | 112MB | 99.50% |
| Light CNN-29 | 4M | 1 | 50MB | 99.33% |
| MobileID | - | 1 | 4.0MB | 97.32% |
| ShiftFaceNet | - | 1 | **3.1MB** | 96.00% |
| **MobileFaceNet** | 3.8M | 1 | **4.0MB** | 99.55% |
| MobileFaceNet (112 ×96) | 3.8M | 1 | **4.0MB** | 99.53% |
| MobileFaceNet (96 ×96) | 3.8M | 1 | **4.0MB** | 99.52% |

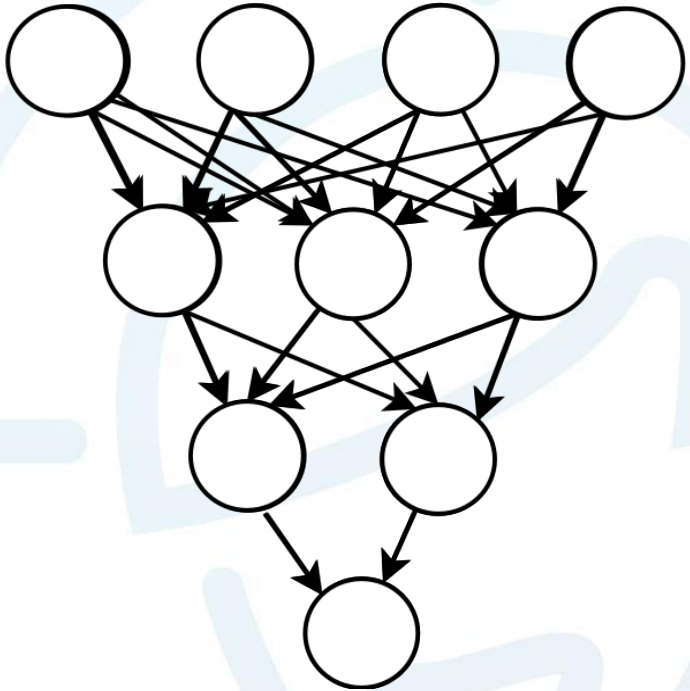
Chen et al., MobileFaceNets: Efficient CNNs for Accurate Real-time Face Verification on Mobile Devices, April 2018

# Network slimming

- Pruning
- Quantization

# Network Pruning

Reducing number of params
while keeping the same accuracy

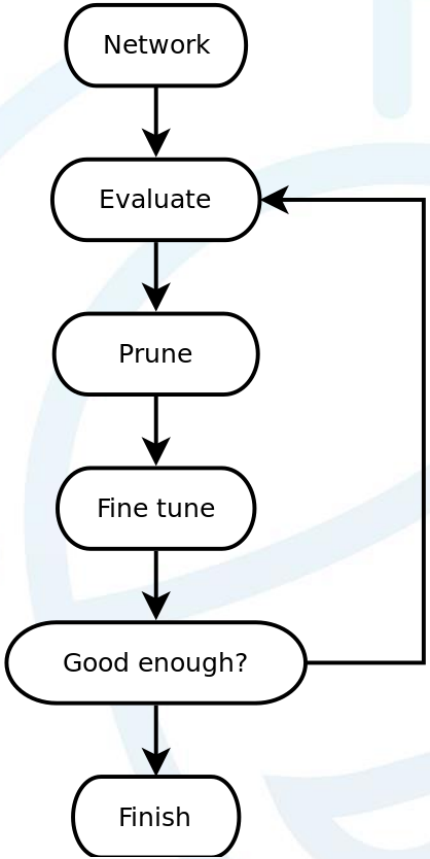


# Network Pruning

- Pruning for size
  - Focus on Fully Connected layers
- Pruning for speed
  - Focus on Convolution layers
    - reduce weights in each filter
    - removing filters

# Network Pruning

- Iterative process
  - prune, train, repeat
  - remove weights below certain threshold
  - too much pruning at once may significantly damaged the model



# Network Pruning

| | Top-1 Error | Parameters | Compression rate |
|---|---|---|---|
| AlexNet (Caffe) | 42.78% | 240MB | 1 |
| Pruning | 42.77% | 27MB | 9 |

Song Han et al.: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding

# Quantization

- NN is resistant to noise
  - weights can have less bits for inference
  - how many?
- Various techniques
  - quantization of trained model
  - simulate quantization during training
  - deep compression

# Quantization of Trained Model

- Range & Step
  - get layers weights min & max
  - map min & max to desired type min & max

| | min | max | zero | step |
|---|---|---|---|---|
| float32 | -20 | 40 | 0 | ~0.235 |
| int8 | -127 | 127 | -42 | 1 |

- Code book / dictionary
  - put weights to a table
  - count of different weights determine memory requirements
- Fixed Point Float
  - easy to implement & use
  - possible speed up thanks to computation in integer arithmetics

# Quantization of Trained Model

| Model | Million MACs | Million Parameters | Top-1 Accuracy | Top-5 Accuracy | Size [MB] |
|---|---|---|---|---|---|
| *MobileNet_v1_1.0_224* | 569 | 4.24 | 70.9 | 89.9 | 16.9 |
| *MobileNet_v1_1.0_224_quant* | 569 | 4.24 | 69.7 | 89.5 | 4.3 |

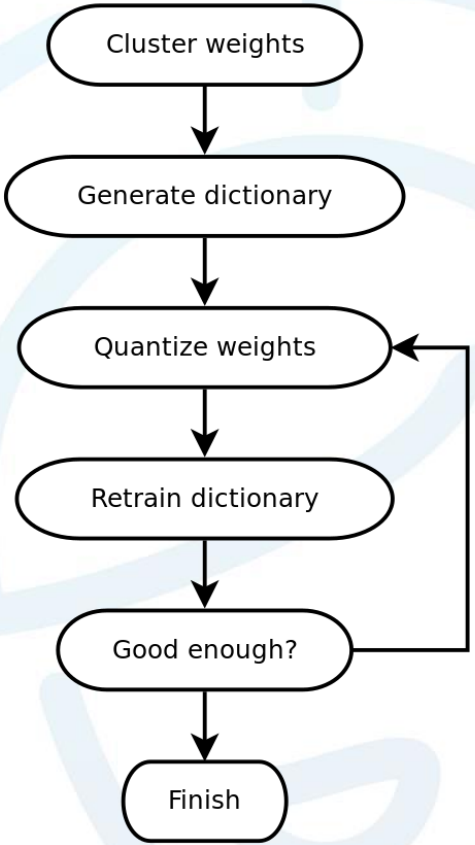
https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

# Training with Quantization

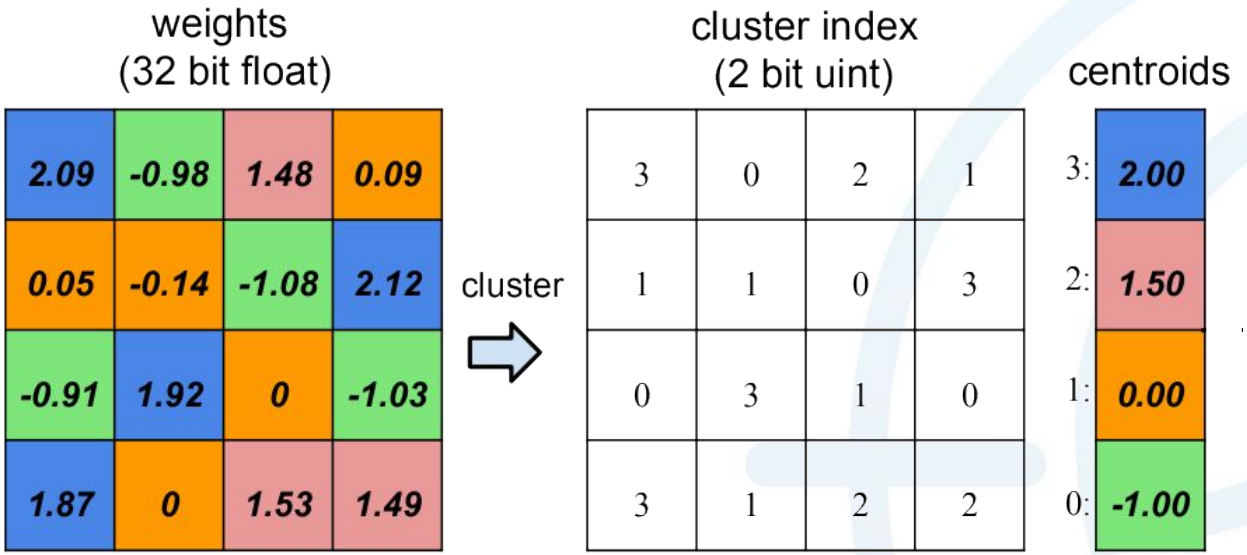
- Learn ranges during training
- Forward pass - quantized values
- Backward pass - float values

# Deep compression

Significantly reduce storage taken by weights by clustering them

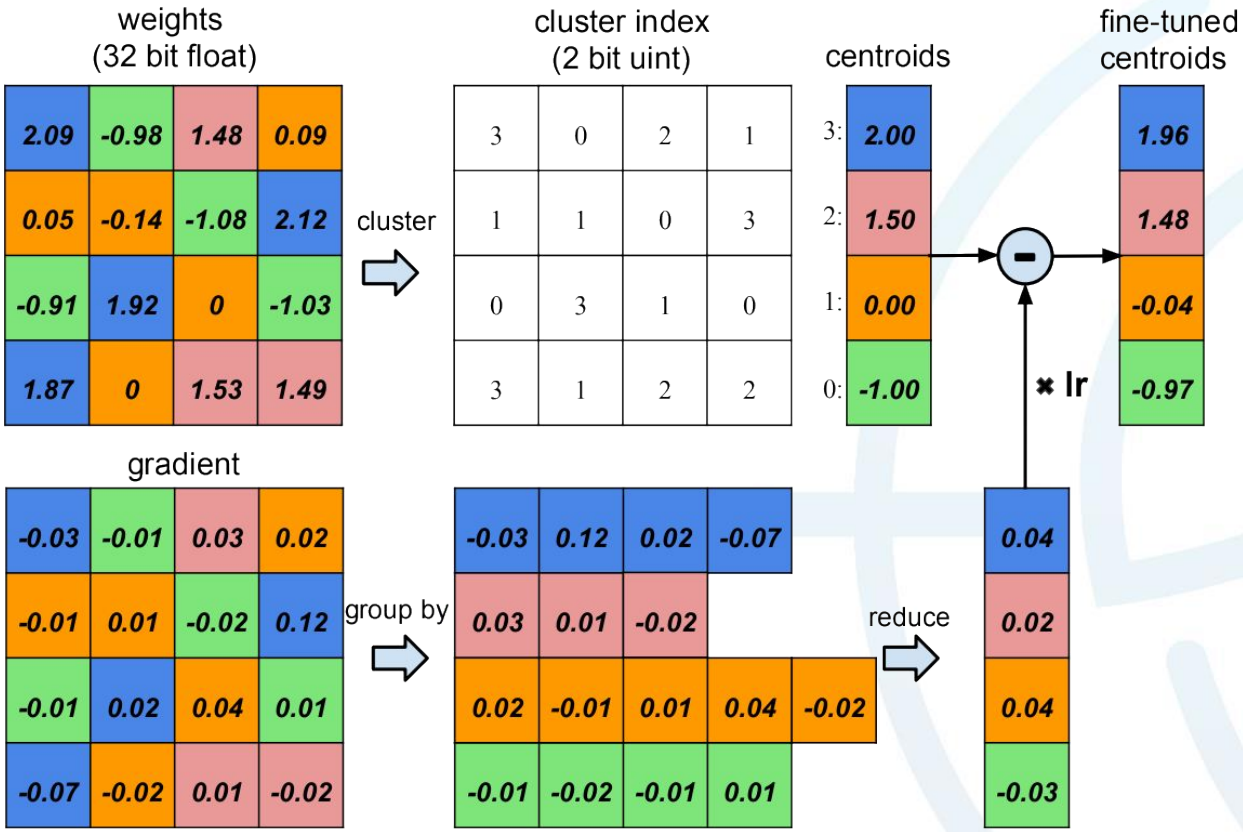


# Deep compression




Song Han et al.: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding

# Deep compression



# Network Slimming

- There is a free lunch!
- Size vs Speed

# Tools and Implementations

- Ristretto
- TensorFlow Lite
- Android NNAPI
- ML Kit
- NCNN
- DIY :-)

# Ristretto | CNN Approximation

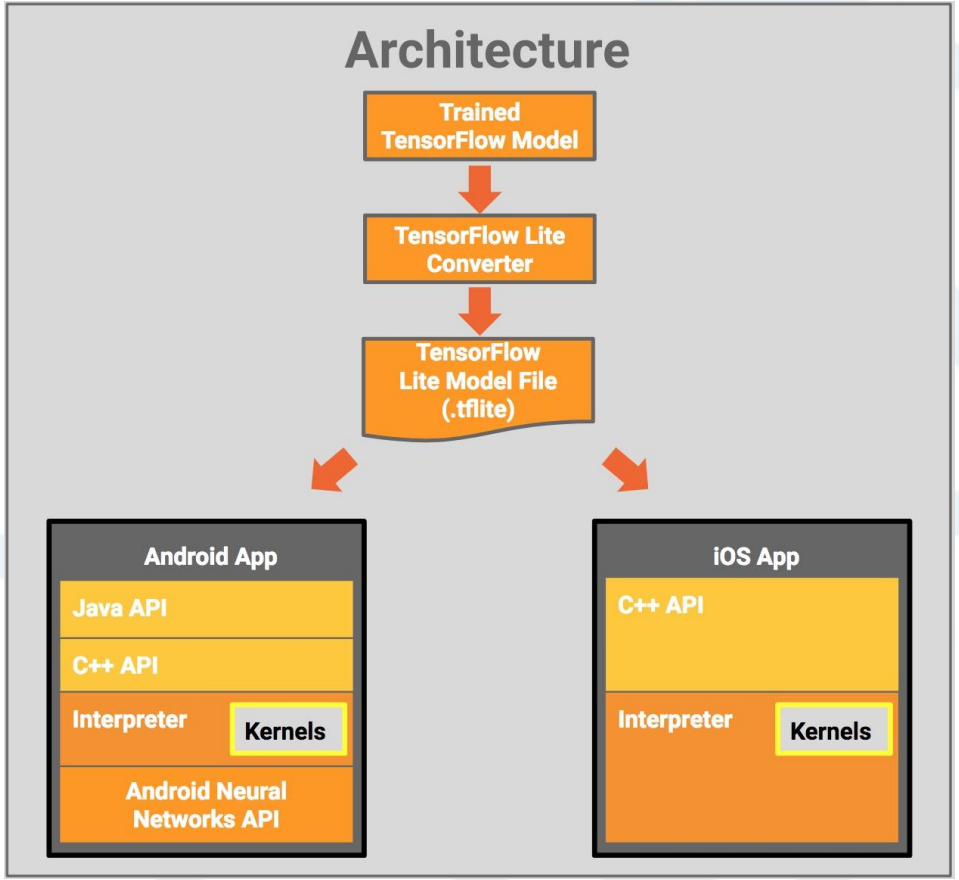
- A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks
- Extension of Caffe
- Tool to quantize trained models
- Offers various approximation schemes
  - dynamic fixed point, minifloat, power of two params
- http://ristretto.lepsucd.com/

# TensorFlow Lite

- A TensorFlow's lightweight solution for inference on mobile and embedded devices
- Evolution of TensorFlow Mobile
- Supports quantized operators
- TensorFlow ships tools to slim NNs
- https://www.tensorflow.org/mobile/tflite/

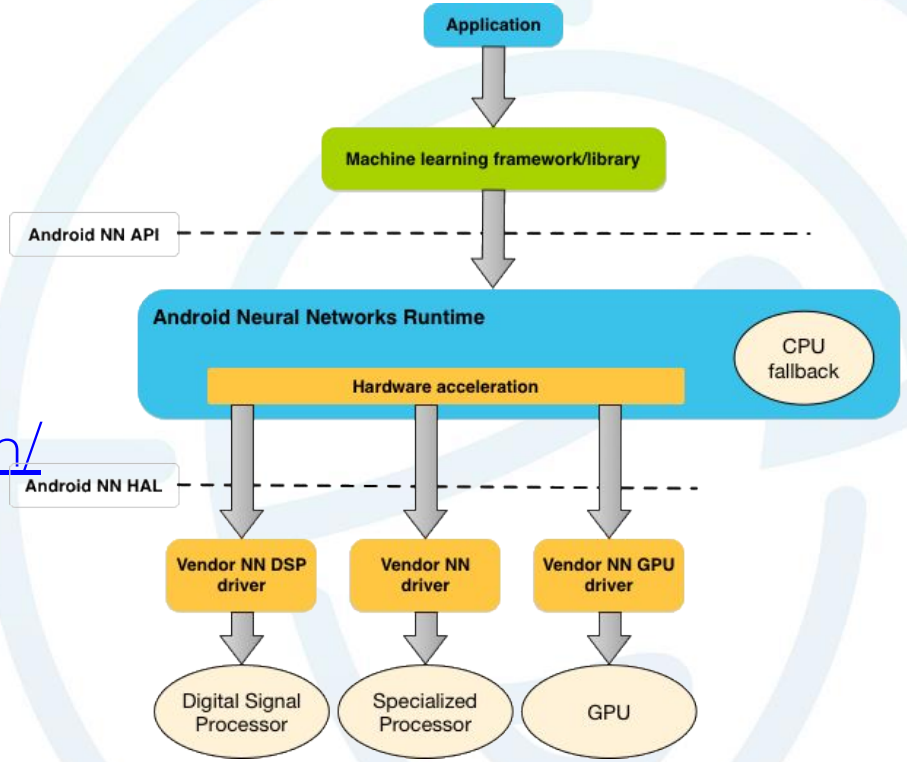
# TensorFlow Lite

- Take a TF model
- Convert it
- Use it!



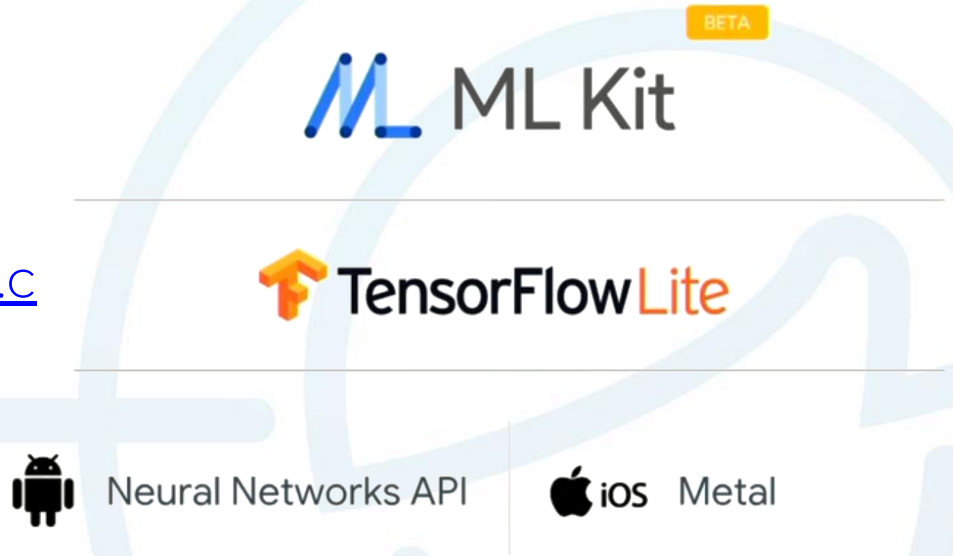
# Android Neural Networks API

- Designed to provide a base layer of functionality for ML Frameworks
- HW acceleration
- Supports Quantization
- Available since Android 8.1
- https://developer.android.com/ndk/guides/neuralnetworks/



# ML Kit

- Google's Machine Learning SDK
- Google I/O '18
- On-device & Cloud APIs
- https://developers.google.com/ml-kit/



# NCNN

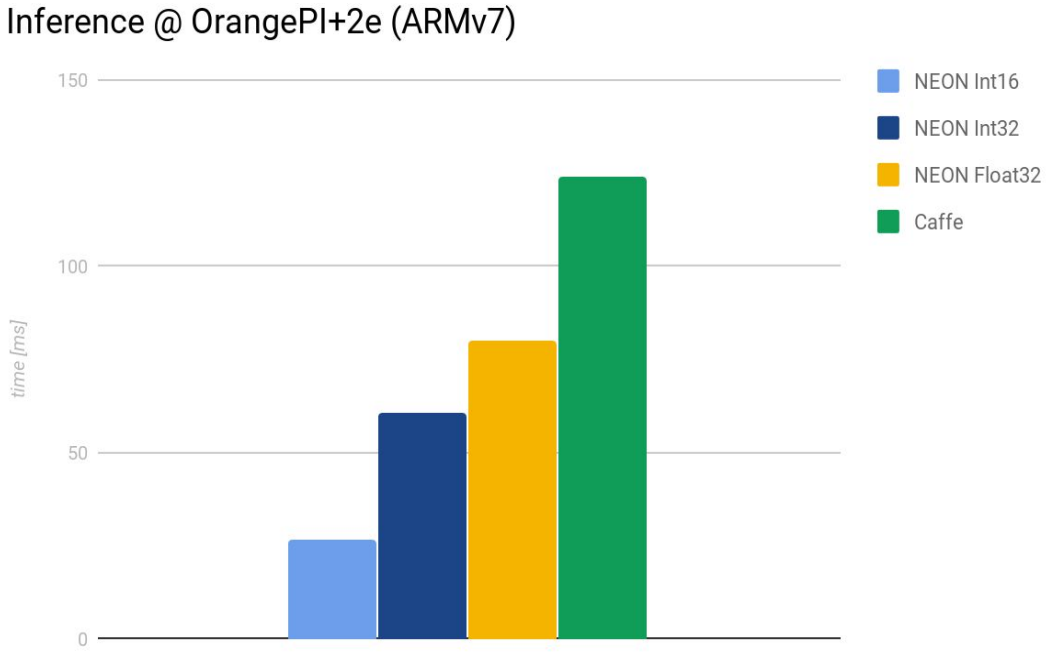
- Framework for inference on ARM
- Supports 8bit quantization and half-precision floating point storage
- Take a caffe/pytorch/mxnet/onnx model
- Convert it
- Use it!
- https://github.com/Tencent/ncnn

# Do It Yourself

- When you really care about the size
- When there is no other option
- When you have specific needs - memory usage, speed, portability
- When you want to know how it is implemented

# DIY - InnoPlayer

- Data order
- Memory handling
- Quantization
  - Fixed Point



# Lessons Learned - NN face detection

- Find similar POC project based on any framework
  - Is there a benchmark (WiderFace) ? Use the “best” solution.
- Domain tuning
  - Try to reach the published results on source platform
  - Tune it on your data to reach your goals
  - Representative Training / Testing data are **very** important
  - Change just one thing and test it
  - Convert trained network to target platform
- Use pretrained models
  - Choose the one which fits your needs
  - Domain adaptation
  - Stable training / less bugs
- Carefully test it
  - Accuracy - Now much higher ….
  - Performance - 10-20x optimized (usable on CPU)
  - Integration tests - Prototyping (Python) vs Runtime (C++ binaries)

# Practise

- Architectures @ Innovatrics
  - Filters / Layers count reduction
  - SqueezeNet for background segmentation
  - MobileNets for object detection
  - Quantization
  - Small ARM CPU Cortex M4 implementation
    - small fingerprints matching
    - INTS@InnoPlayer
    - Memory optimizations
    - Size : NN + Inference module = 100kB


QUESTIONS ?

# Thank you

Marián Beszédeš
Image Processing Team Leader
at Innovatrics

marian.beszedes@innovatrics.com
www.innovatrics.com

# Sources

- https://medium.com/@smallfishbigsea/notes-of-squeezenet-4137d51feef4
- https://www.slideshare.net/JinwonLee9/mobilenet-pr044
- https://www.slideshare.net/ForrestIandola/small-deepneuralnetworks-their-advantages-and-theirdesign
- https://www.slideshare.net/anirudhkoul/squeezing-deep-learning-into-mobile-phones
- http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone
- http://mmlab.ie.cuhk.edu.hk/projects/WIDERFace/
- https://jacobgil.github.io/deeplearning/pruning-deep-learning
- https://petewarden.com/