Designing Efficient Architectures for Mobile Computer Vision

Mark Sandler

Google AI

Research at Google

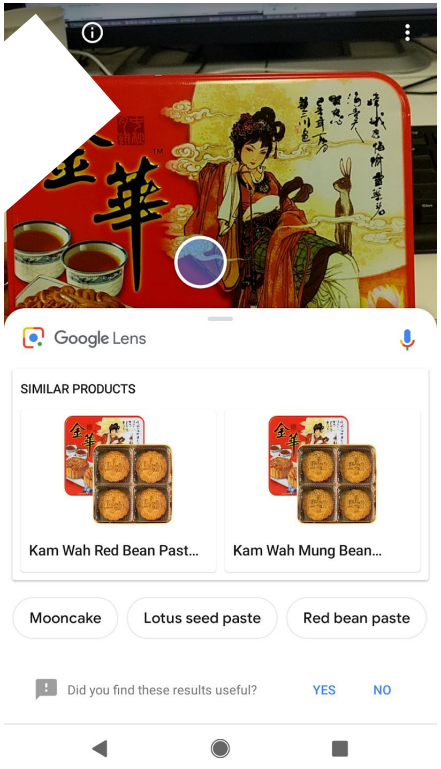
Goal: realtime vision on mobile and embedded devices

- Smart viewfinder apps
- Always-on recognition
- New smart devices

Means optimizing 4 factors:

1) Latency
2) Power consumptio
3) Inference memo print
4) Storage foot model size)

Keep your accuracy high!



Performance Metrics

Many Metrics!

- Inference/storage memory, latency, power consumption
- Hard to optimize

Synthetic metrics

  - # of FLOPS (Multiply adds)
  - # Parameters
  - # Memory reads per math operation

- Synthetic metrics are only a proxy for the real thing

Finding best architectures

Optimizing real metrics is hard!

- Depend on hardware/software
- Automatic search techniques allow to optimize directly

Synthetic metrics are easier to reason about (for software engineer anyway)

- Good performance predictors (with caveats)

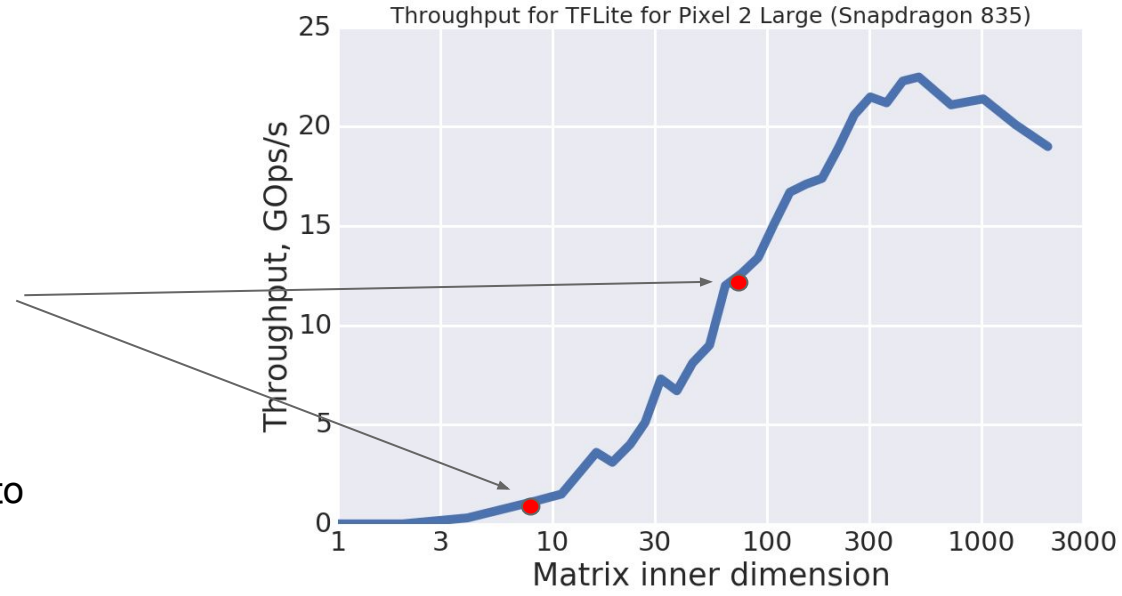
What's wrong with synthetic metrics?

The same type of operations have wildly different costs

Memory accesses outside of cache are hugely expensive

Example

1 op in 10x10 matrix multiplication is ~7x more expensive than in 100x100.



If we optimize FLOPS won't be able to capture this!

Overview of the rest of talk

1. How to make my networks smaller and faster
   - ~~One~~ ~~two~~ weird tricks that nobody wants you to know about

2. Architecture design: a case study

   Explore several popular architectures in detail

3. Automatic architecture search (don't try this at home!)

Universal tricks!

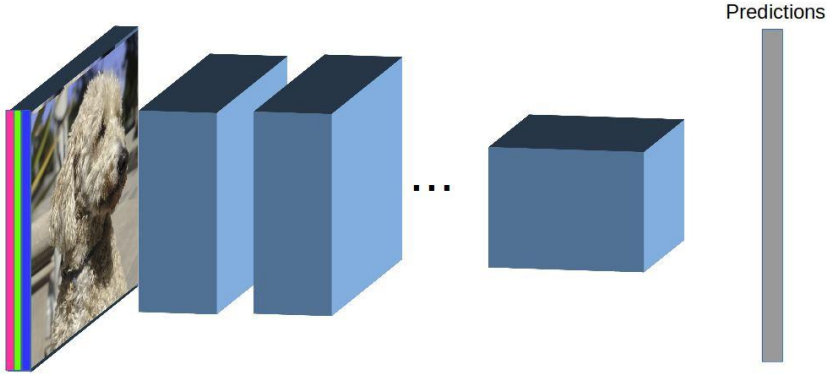
You have an architecture how to make it
- smaller and faster.

There are several ways that provide a strong baseline

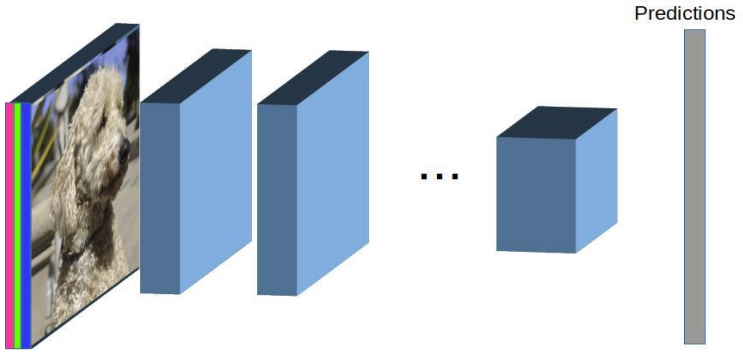
- Resolution
- Multiplier
- Quantization

Width multiplier

- Start with a network:



- Reduce each layer by a multiplier



- Retrain.
  Can be applied to any network

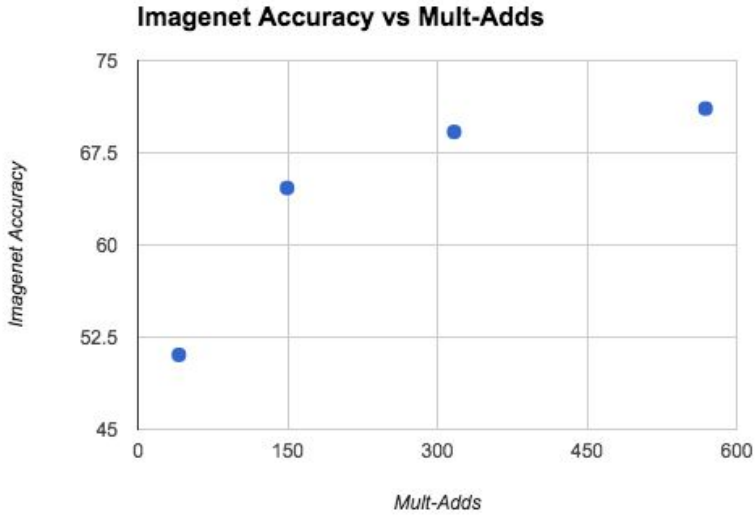
Width multiplier

1) For multiplier γ
   each layer is γ times smaller

2) Parameters, #ops drop by γ^2

3) Very gradual drop in accuracy:

Mobilenet V1



Howard et al, 2017

Multiplier

1. Very strong baseline
2. Reduces model size and increases speed.

Cons:

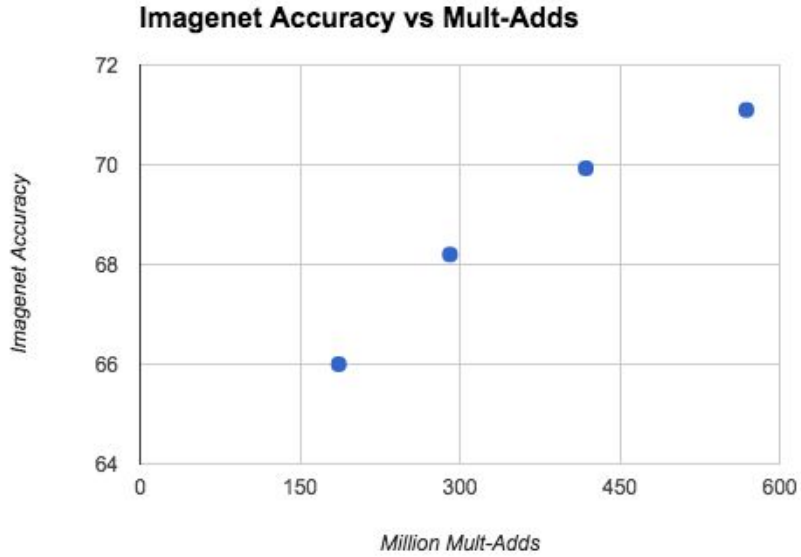
a) Small multiplier end up having lower throughput (Ops/s)

b) Need full retrain.

Reduce your networks: Trick 2: Resolution multiplier

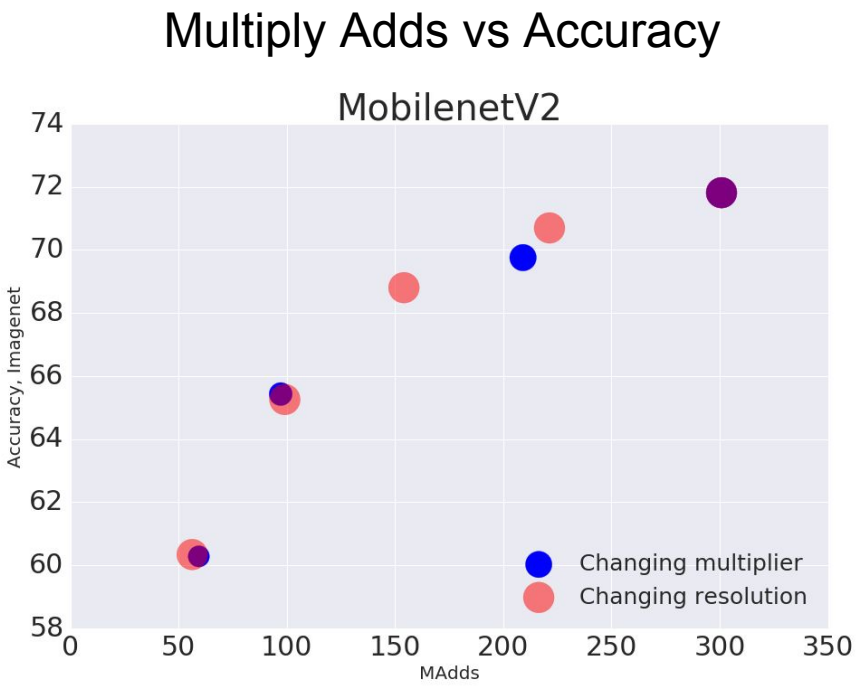
Convolutions can be applied
to any input resolution

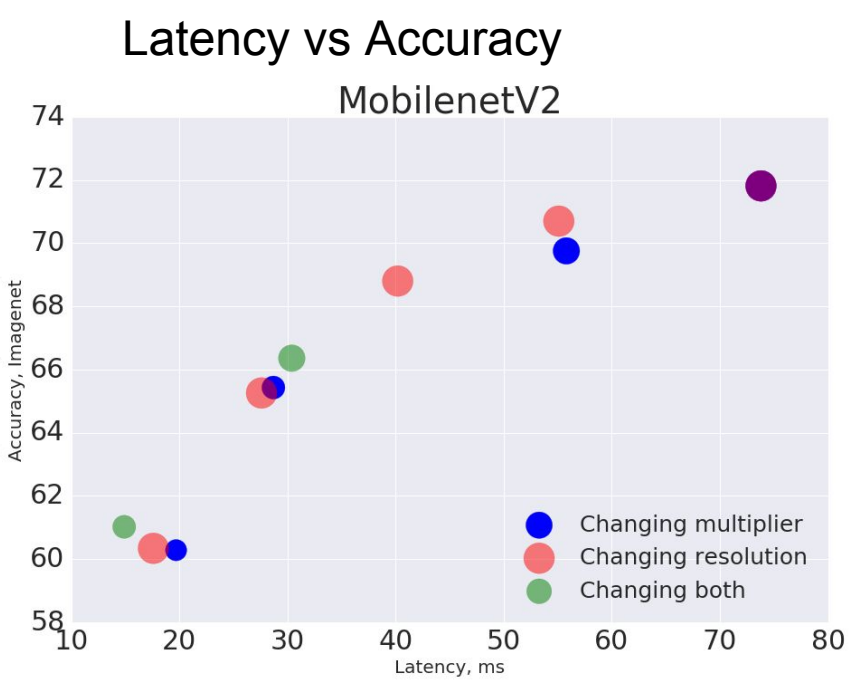
=>



- The number of operations ~ # pixels
- 2x resolution change -> 4x speed up (same number of parameters)
- Still need to re-train (else: accuracy is dramatically lower)

Width vs Resolution multiplier

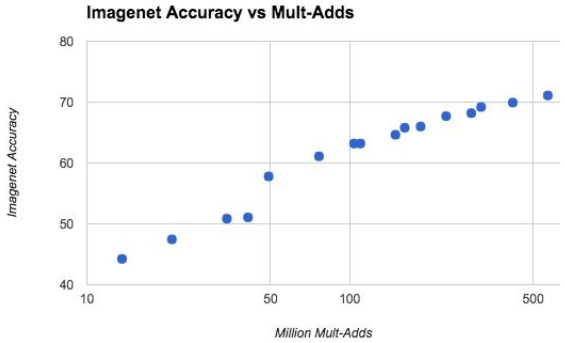




Conclusion: If you have a network you can make it smaller

These techniques can be applied to any architecture

- Allows to create a range of models trading accuracy for speed.



Enables evaluation the quality of a new architectures:

- Does it beat old architecture with multiplier applied to match other parameters?

Bonus trick: quantization:

Quantization: replace weights/activations w/low bit representation.

- Most common 8-bit

- Some hardware architectures support 4 or less
- Improves speed by ~2x with negligible drop in accuracy.

- Important: multiple quantization themes
  - per channel boundaries, etc

  - Mismatch between themes breaks things

Quantization:

(Krishnamoorthi, 2018)

- Naive: quantize everything and hope for the best
  - Accuracy drops dramatically

- Two common ways:
  - Post-training quantization.
    - Can be done very cheaply
  - Quantization aware training
    - Requires re-training gives higher results (0.5% - 1% on Imagenet)

Post training Quantization (Krishnamoorthi, 2018)

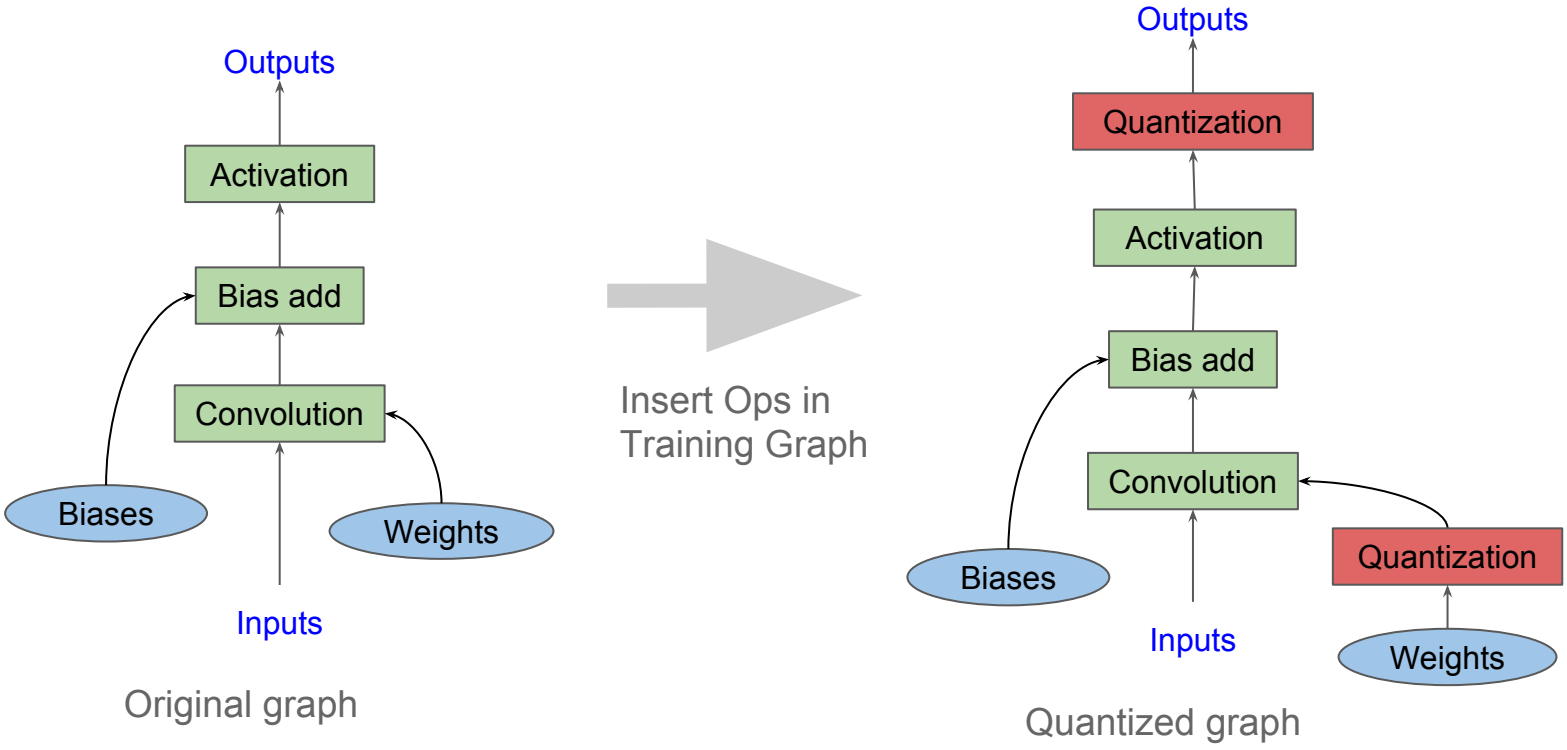
- Modern networks typically have Batch Normalization
  a. Accumulate typical distribution statistics to normalize all activtion and moments to be 0/1

- Quantization shifts them - enough to degrade performance

- Solution: recompute them with quantized weights

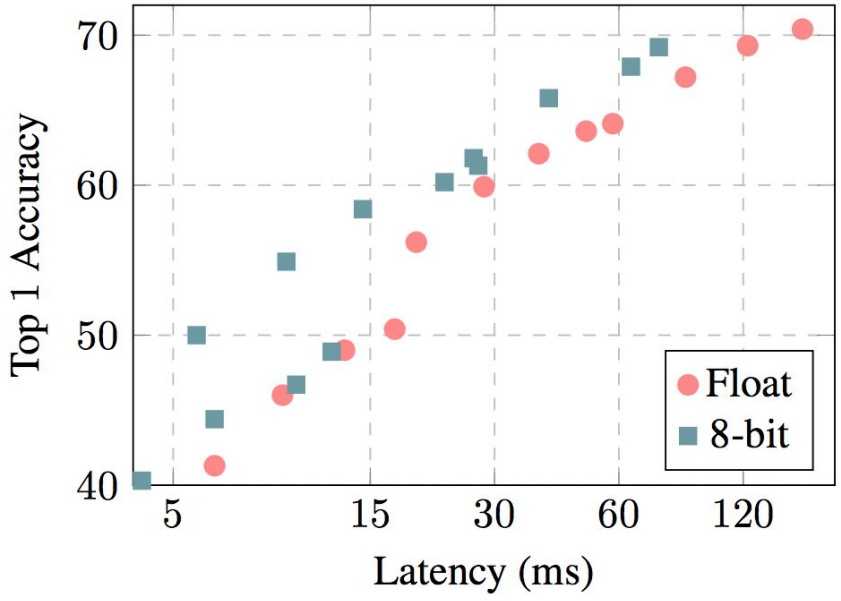
- Practical TF trick: train network with 0 learning rate for ~XK steps to accumulate updated stats

Quantization aware training



Quantization impact on speed:

Roughly 2x improvement on TFLite



(Jacob et al, 2017)

Architectures: A case study

Explore 3 recent architectures.

1) Mobilenet V1 (Howard et al 2017)

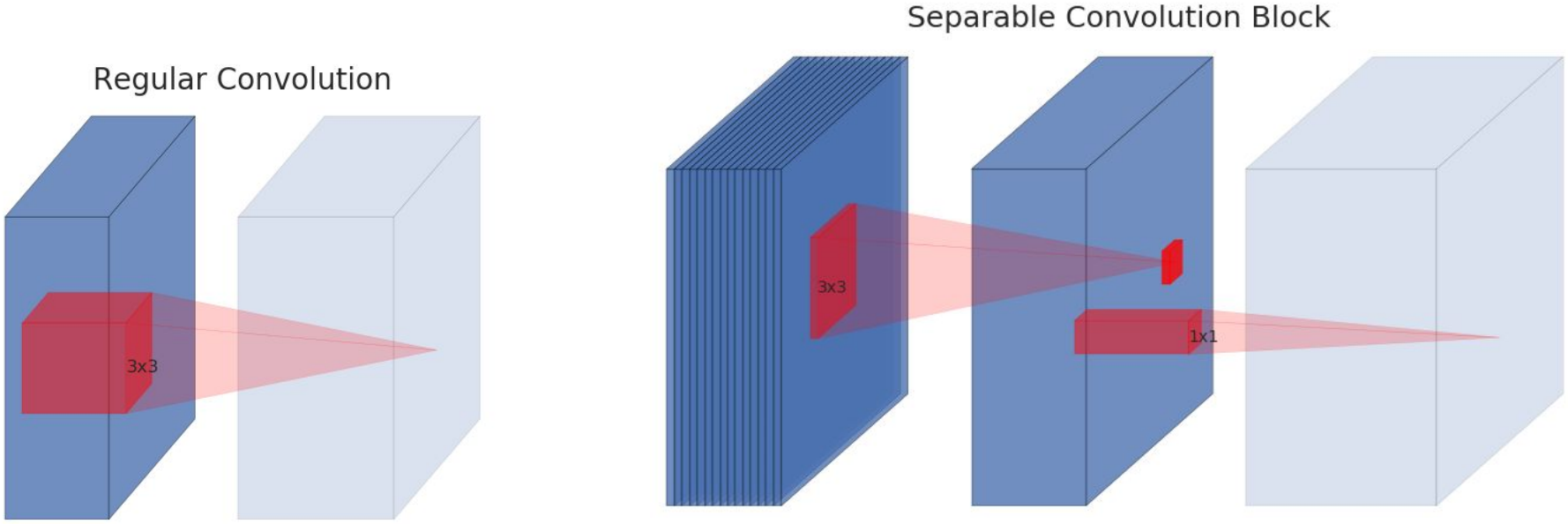
2) Mobilenet V2 (S. at al 2018)

3) Shufflenet V2 (Ma, et al 2018)

These are architectures are all simple yet push sota.

Architectures: Mobilenet V1 (Howard et al, 2017)

1) Introduced depthwise separable convolution for mobile
2) Very simple architecture

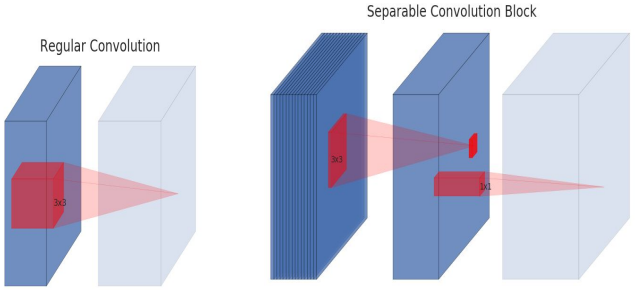


Sandler et al, 2018

Architectures: Mobilenet V1

Depthwise:
Replace one slow convolution

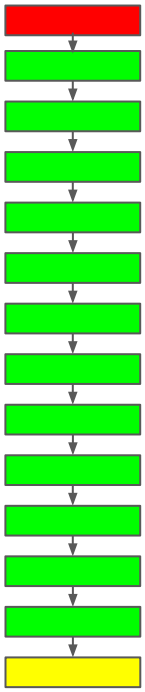
$$Y_{hwc} = \sum_{i,j,d} W_{cijd} X_{(h+i)(w+j)d}$$



With two convolutions::
1) Per-channel convolution (typically 3x3)
2) 1x1 convolution

Empirical fact: cost about 1% in accuracy (on Imagenet)
about 6-7 times faster and 8 times fewers parameters.

Mobilenet Architecture



Type	Output Depth	Output Resolution
Convolution	32	112
Separable Convolution	64	112
Separable Convolution	128	56
Separable Convolution	128	56
Separable Convolution	256	28
Separable Convolution	256	28
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	1024	7
Separable Convolution	1024	7
Avg Pool + FC	1000	1

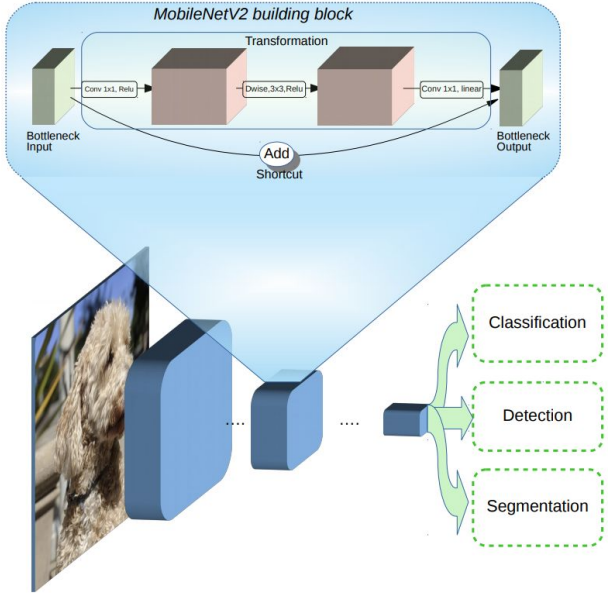
- 95% of computation is 1x1 convolutions efficiently implemented with GEMMs.

Mobilenet V2 (S. et al, 2018)

- Follows similar design principles:
  1) Identical blocks stacked on top of each other

  2) Uses depthwise convolution as Mobilenet V1

  3) Uses shortcut connection and linear bottlenecks



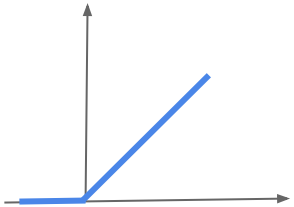
Mobilenet V2: intuition

- Common assumption: is low dimensional
- Why not reduce dimensionality (e.g. apply multiplier?)
- Turns out that non-linearity requires large number of dimensions to work properly.

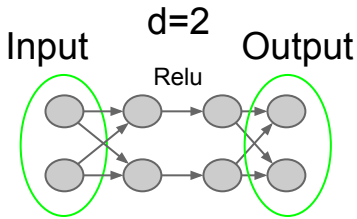
Non linearity in low dimensions

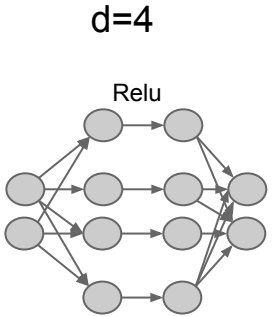
1-dimensional

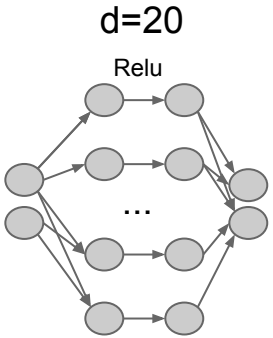
y = relu x



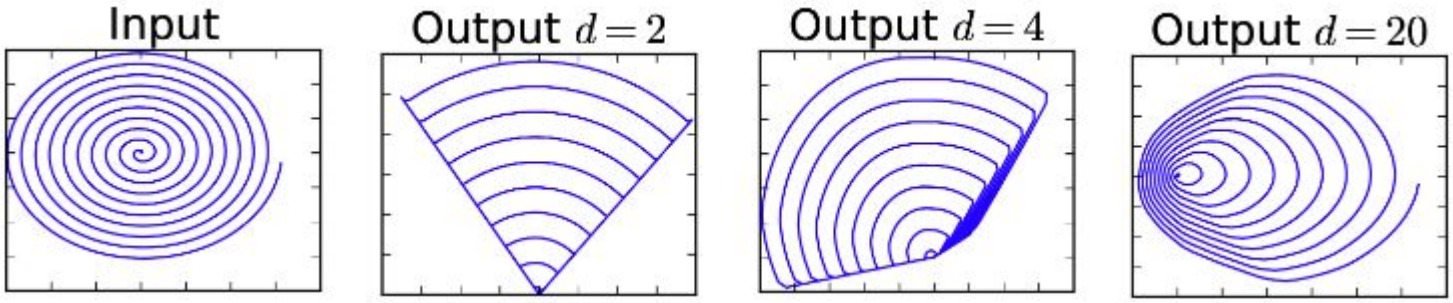
Information is lost if x < 0







2D:

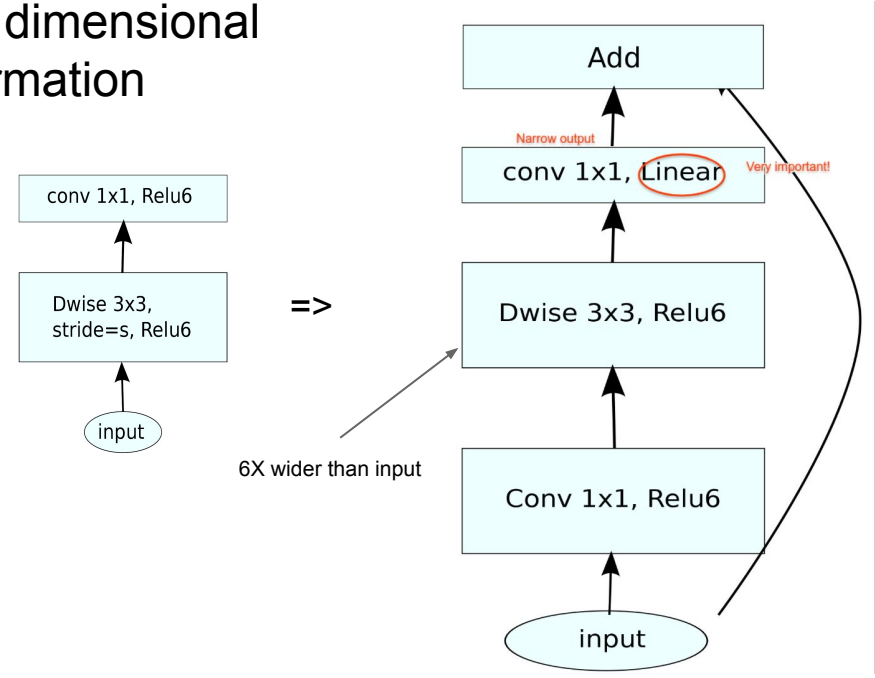


Mobilenet V2

- If the internal space is inherently low dimensional
- Can project down without losing information

- Allows to reduce compute time
  By ~30% on mobile CPU

- Mobilenet V2 have somewhat
  lower op-throughput uses
  much fewer ops.



Efficient Memory Inference

- Convolutional block looks:

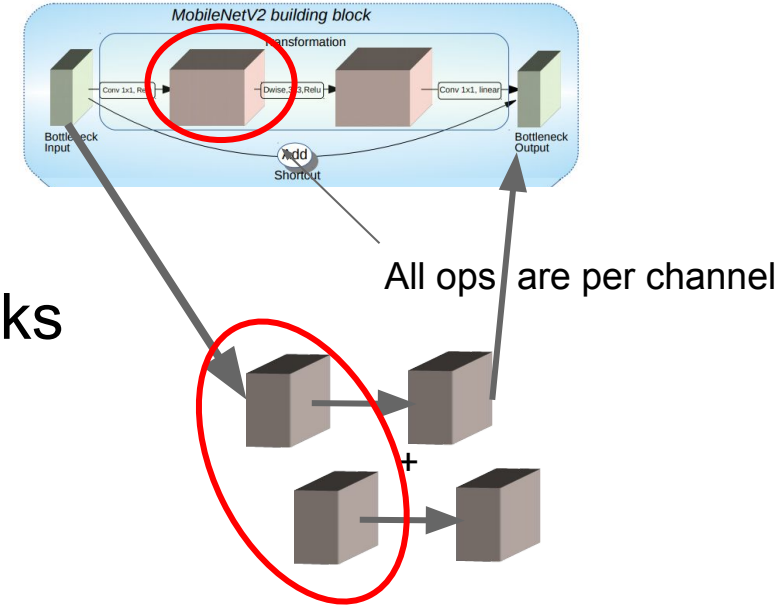
$$\mathcal{L}(x) = [A \circ \mathcal{N} \circ B]x,$$

- Split inner tensors into *t* blocks

$$\mathcal{L}(x) = \sum_{i=1}^{t} (A_i \circ \mathcal{N}_i \circ B_i)(x)$$

- Compute them one by one

- No need to fully materialize inner tensors.

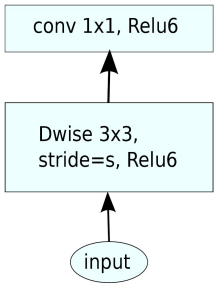


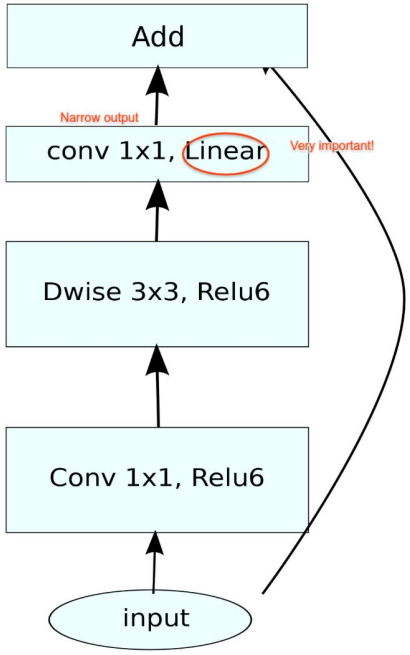
Shufflenet V2 (Ma, et al 2018)

- Goal of minimizing number of memory accesses
- Uses depthwise convolutions (same as Mobilenets)
- Passes through half of previous layer
  => reduce number of channels.
- Uses shuffling to prevent segregation
- Uses aggressive spatial downsampling and larger # of channels
- Latency better than MobileNet V2 despite having comparable #ops.

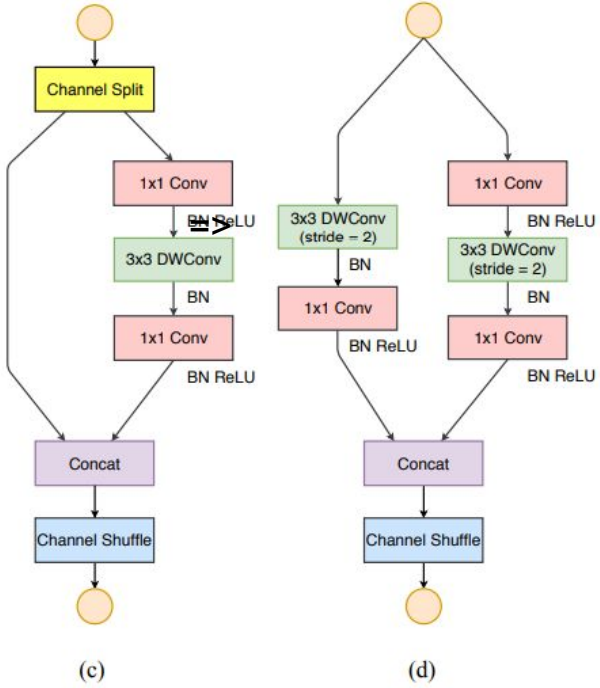
Convolutional Block

Mobilenets





Shufflenet V2



Ma, 2018

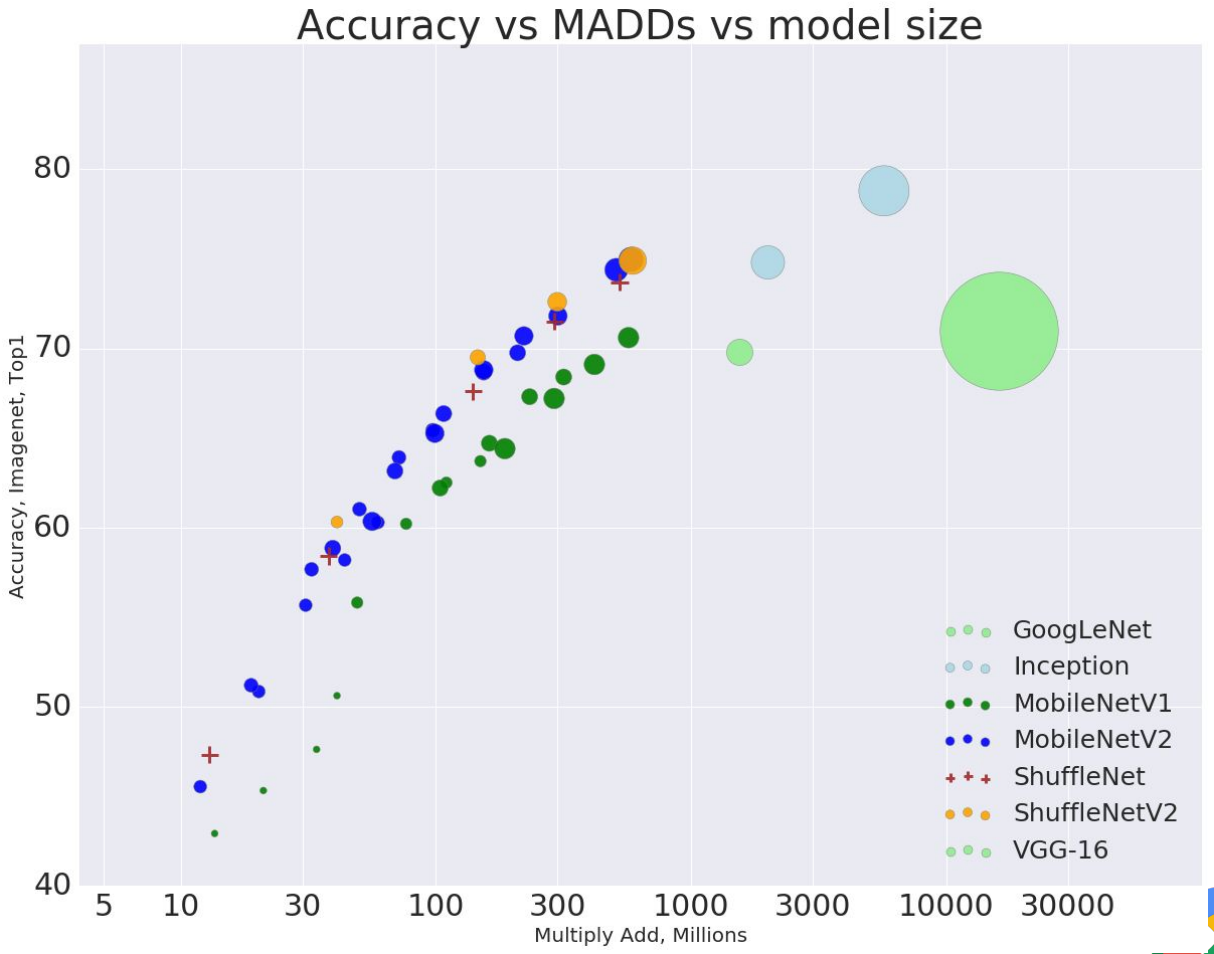
Performance comparison (Madds)

Latency:

MobilenetV2 is ~20-40% faster than MobilenetV1 on CPU

ShufflenetV2 is ~30-50% faster than MobilenetV2

Caveat: depends on hardware a lot



Concluding thoughts

- Mobile architectures tend to be very simple
- Designing efficient convolution blocks is a key
- Often chicken-and-egg of hardware vs software
- Might need to change the target problem soon - might be overfitting to Imagenet
- Trade off curve: 4% accuracy hit for every 2x speed up

Automated Architecture Search

- What if instead of hand-tuning architectures we use AI

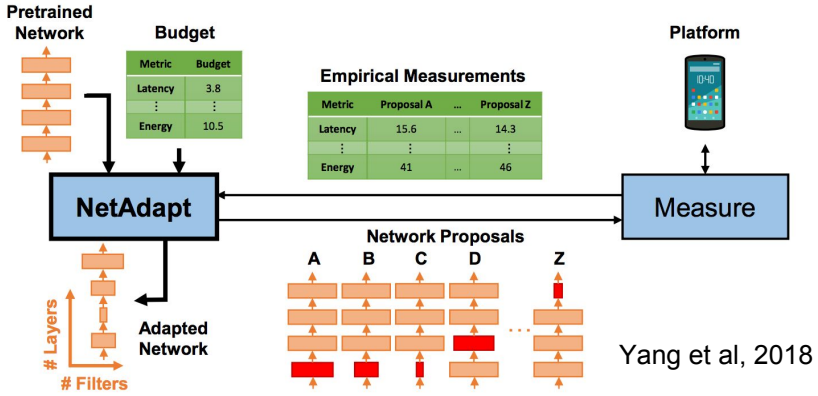
- Lots of work! Will discuss two approaches
  - Use incremental greedy improvements of the architecture
  - Use reinforcement learning to optimize some components.

Automatic architecture search

- Great promise! Can target particular hardware/software
- As good as your search space
- Expensive:
  - Quality of architecture is evaluated by training!
  - Good news: training for several epochs is a great predictor of final quality

NetAdapt: Adapt to target platform (Yang et al, 2018)

Basic idea:



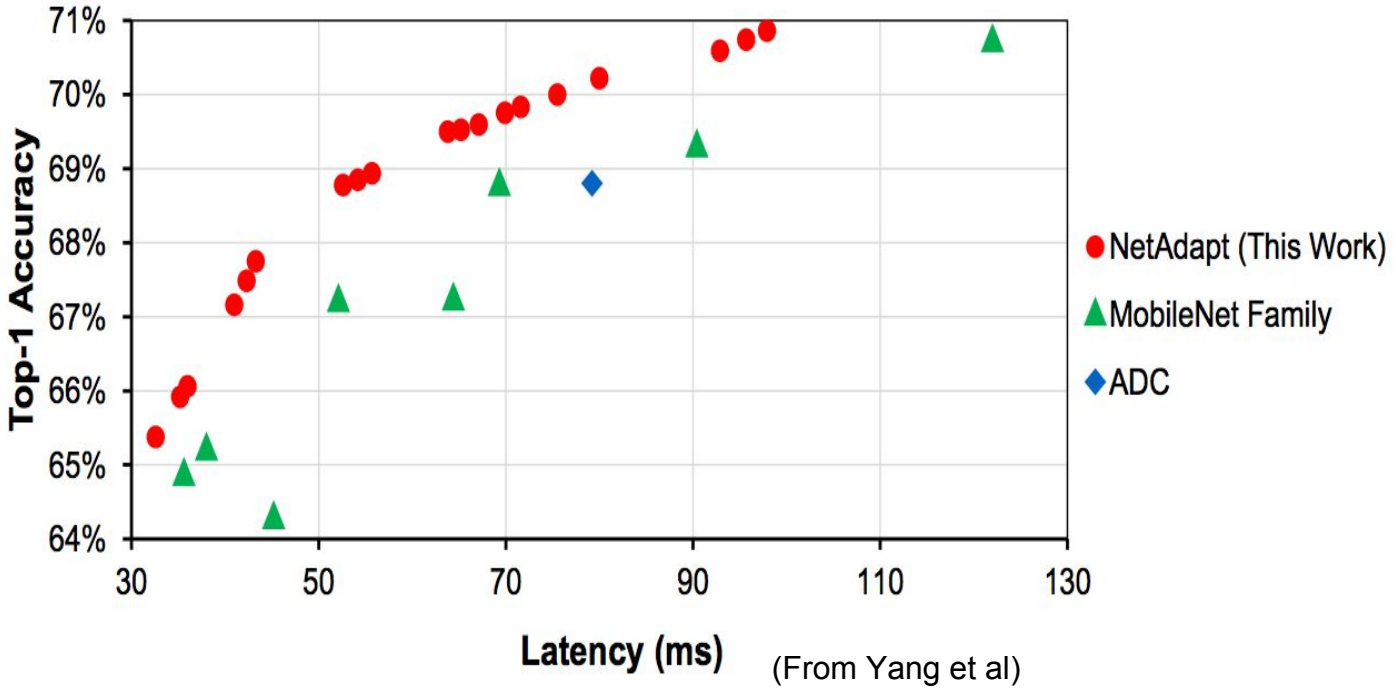
Yang et al, 2018

1. Greedy search
2. Each step:
   pick best proposal minimizing target metric by pre-specified amount, while minimizing accuracy hit

   Proposal generation: reduce width of each layer (and pick the best)

3. Goal: iterate until arrive at architecture with given target metric

NetAdapt applied to MobileNet



(From Yang et al)

Reinforcement learning based achitecture search

1. Early variants were validation accuracy (NasNet)
   a. Had inherent limit on size based on the block algorithm

2. Newer variants target mobile devices (MNasNet)
   a. Target latency by incorporating measured latency into loss function

Reinforcement learning based architecture search:

Use reinforcement learning (Zoph et al, 2017)

- Finds the most efficient convolutional block
- Blocks are stacked on top of each other
- Architectures evaluated using Cifar10
- Updates the selection probability to reflect validation using RL based policy gradients.
- Results transferred well to Imagenet

Building convolutional blocks (Zoph et al)

1. Use fixed number of steps (5)

2. Each step:
   a. Select two inputs from either:
      i. Output of the previous block,
      ii. Intermediate in this block.

   b) Apply one of standard operation (conv, pooling, identity)

   c) Combine using addition/concatenation

   d) Result is the output of this step

3. All unused outputs from all steps are concatenated as “block” output

Controller

1. How each operation is selected?

2. Uses RNN, parameters are updated using REINFORCE (Williams, 1992) gradient updates.

3. But turns out even random selection works pretty well (though there is gain from reinforcement learning)

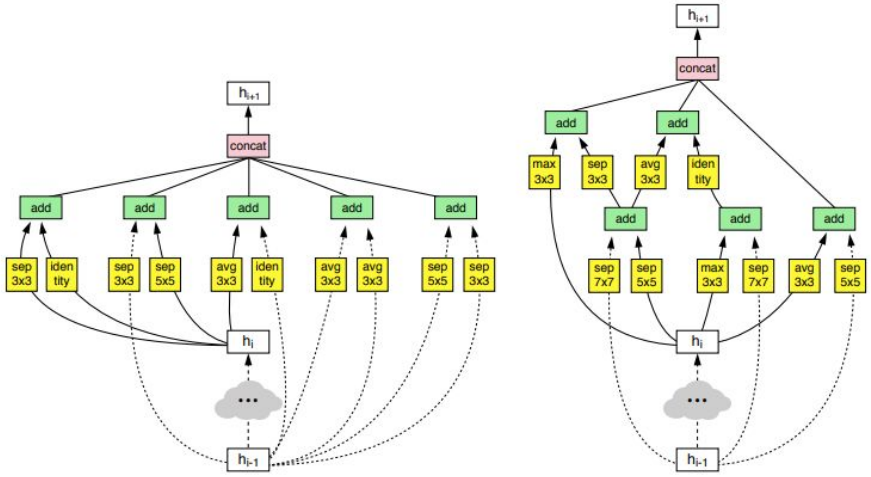
Results

Reaches state of the art in Imagenet

Has good FLOPS counts for mobile. (Same as MobileNet, but 3% higher accuracy)

Because of extreme number of small ops, about 40% slower than MobileNet

Next iteration of this



MNasNet (Tan et al, 2018)

1. Uses MobilenetV2 as starting point
2. Search space is concentrated around changing parameters of each block
3. Cost function: incorporates mobile device latency with accuracy as cost metric

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^{w} \tag{2}$$

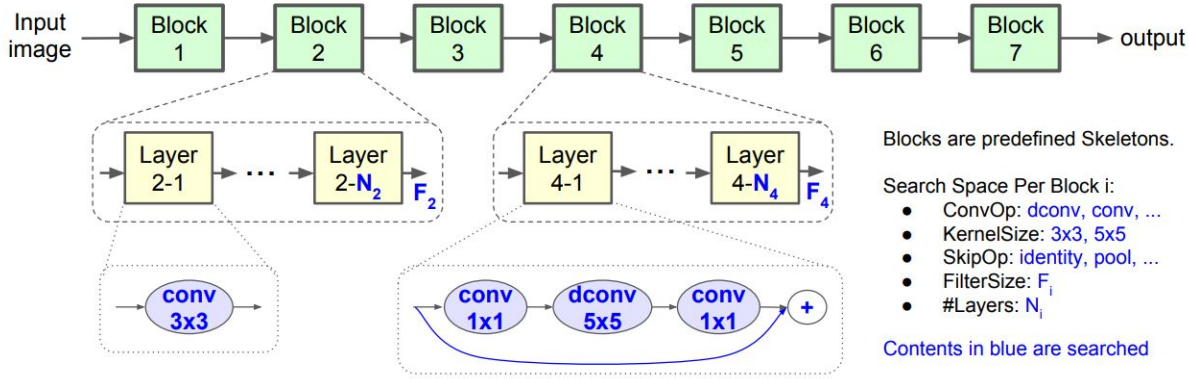
where w is the weight factor defined as:

$$w = \begin{cases} \alpha, & \text{if } LAT(m) \leq T \\ \beta, & \text{otherwise} \end{cases} \tag{3}$$

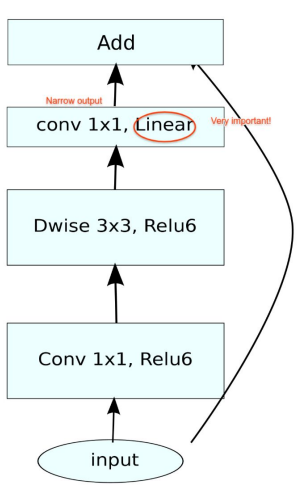
4. Instead of designing 1 block, learns blocks independently.

Block structure

Block structure very similar to MobileNetV2



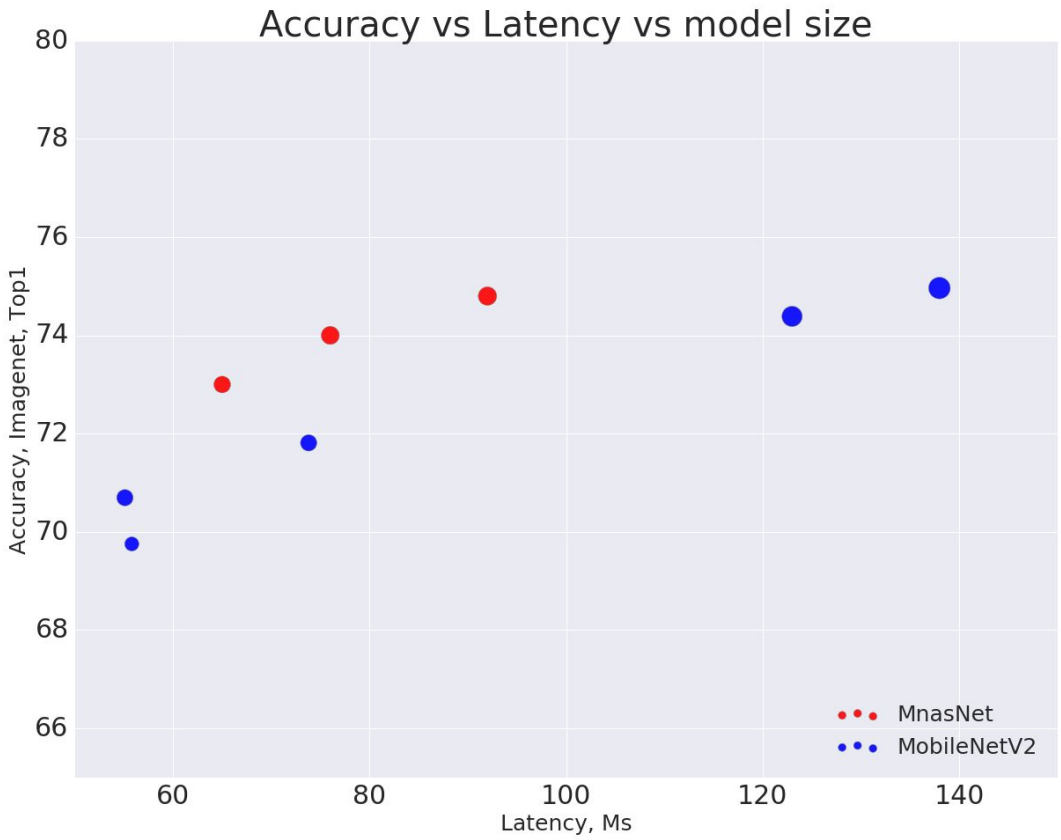
(Tan et al,2018)



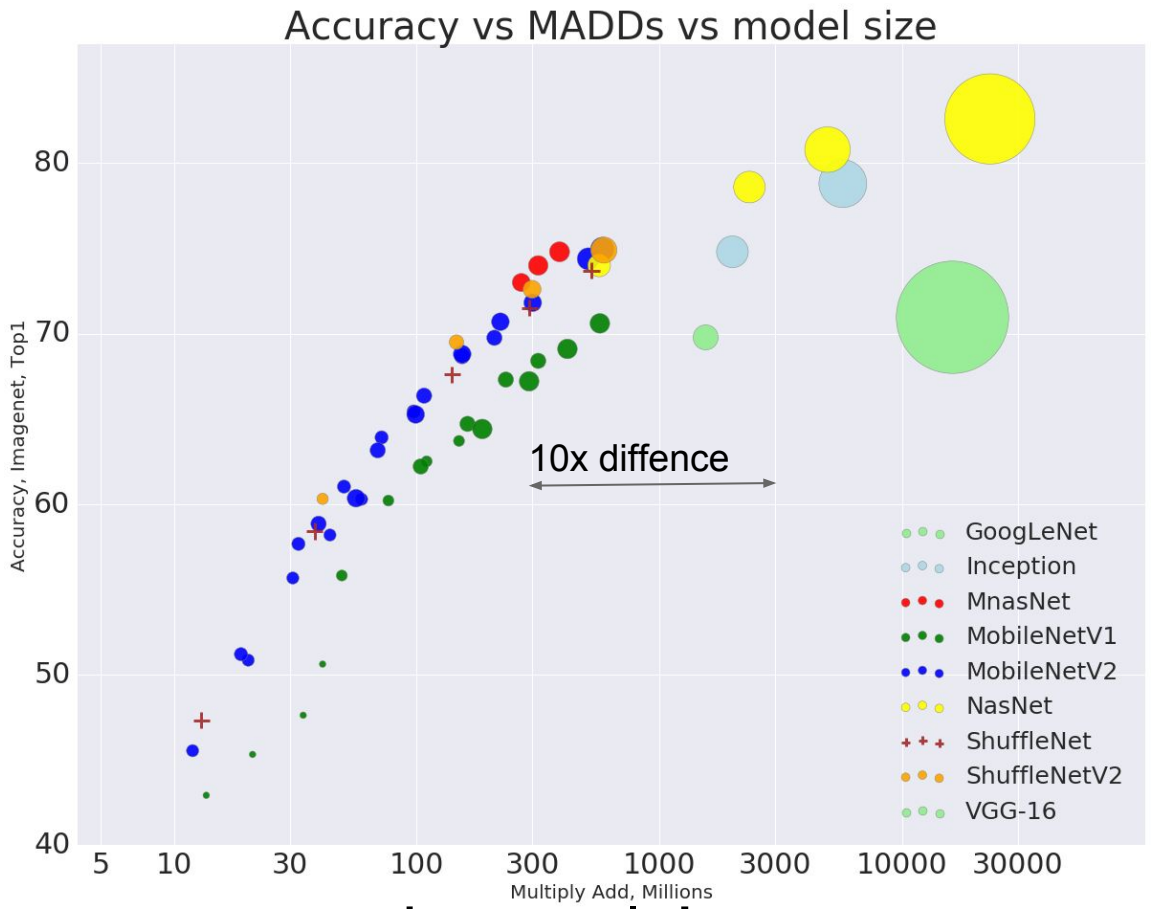
S. et al 2018

MNasNet performance vs Mobilenet V2

About 40% faster for the same accuracy



Where we are with different networks



Log scale!

Wrapping it all up!

- Design of efficient architecture takes a synergy of human designs and automated optimization.
- There are several common ways to build a family of architectures at all performance points that form a good baseline for evaluating new architectures.
- Automated optimization holds a great promise but often is a final touch on the architecture.

Thanks! Questions?

References

Architectures:

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (Howard et al)
MobileNet V2: Inverted Residuals and Linear Bottlenecks (S et al)
Shufflenet V2: Practical Guidelines for Efficient CNN Architecture Design (Ma et al)

Quantization:

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference (Jacob et al)
Quantizing deep convolutional networks for efficient inference: A white paper

Automated search:

Learning Transferable Architectures for Scalable Image Recognition (Zoph et al)
NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications (Yang et al)
MnasNet: Platform-aware Neural Architecture Search For Mobile (Mingxing et al)