EECE695D: Efficient ML Systems

Network Architectures

(part 1)

Announcements

ICLR deadline. Good luck! 🤞 The class ends a bit early.

Team matching. 25 students already found their teams
17 students (and 4 sitting-ins) remain teamless
(let TA know ASAP; one-person team also)
+ Anybody willing to collaborate with strangers?

CoLab. Send TA (1) the name of the team representative
(2) her/his schedule on 10/18

HW#1. Due 10/4, 23:59PM — know about transformers?

Guest Lecture. On 10/6 (Thursday) — Self-supervision

Today

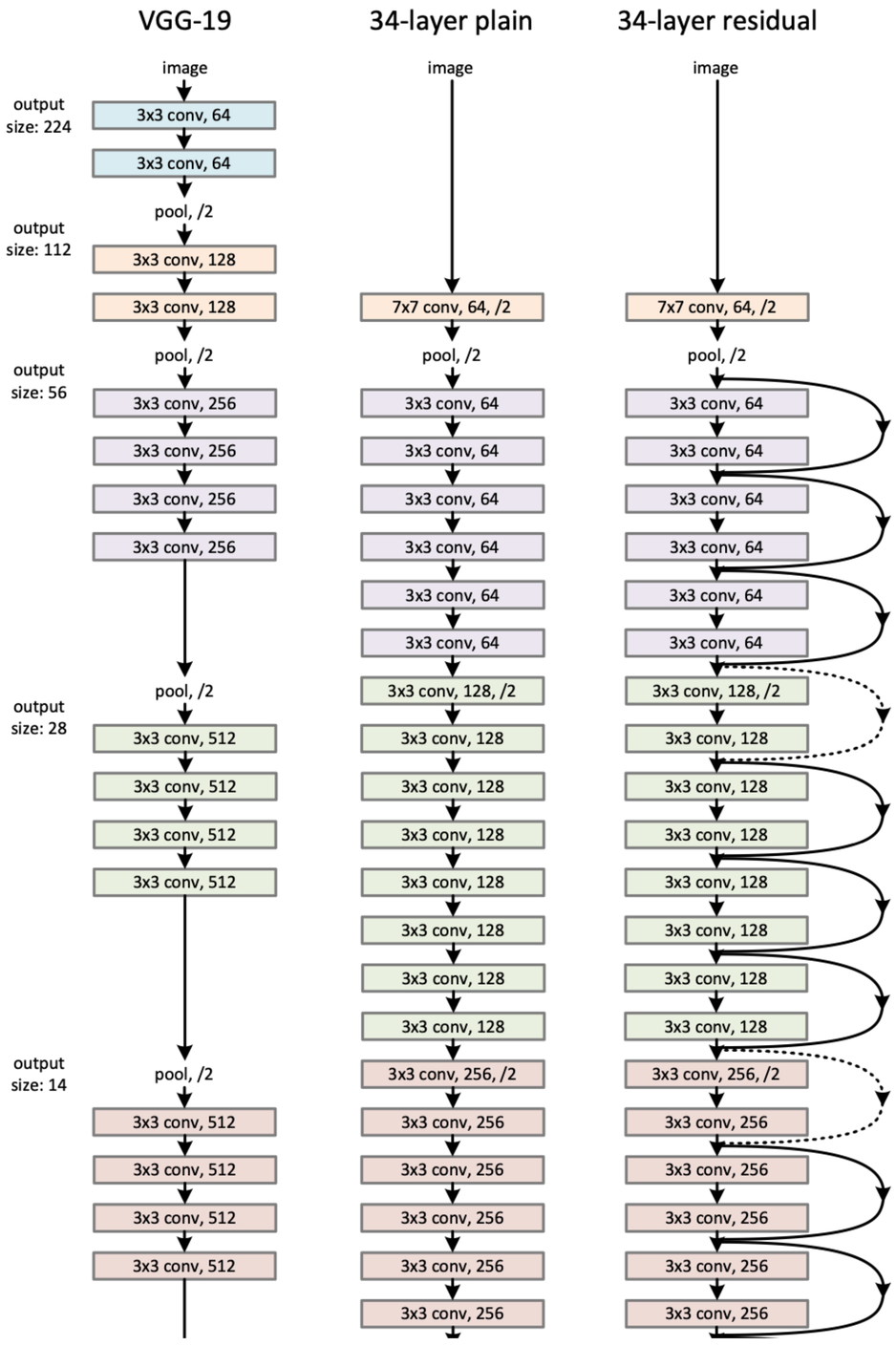
Neural network architectures focused on inference efficiency

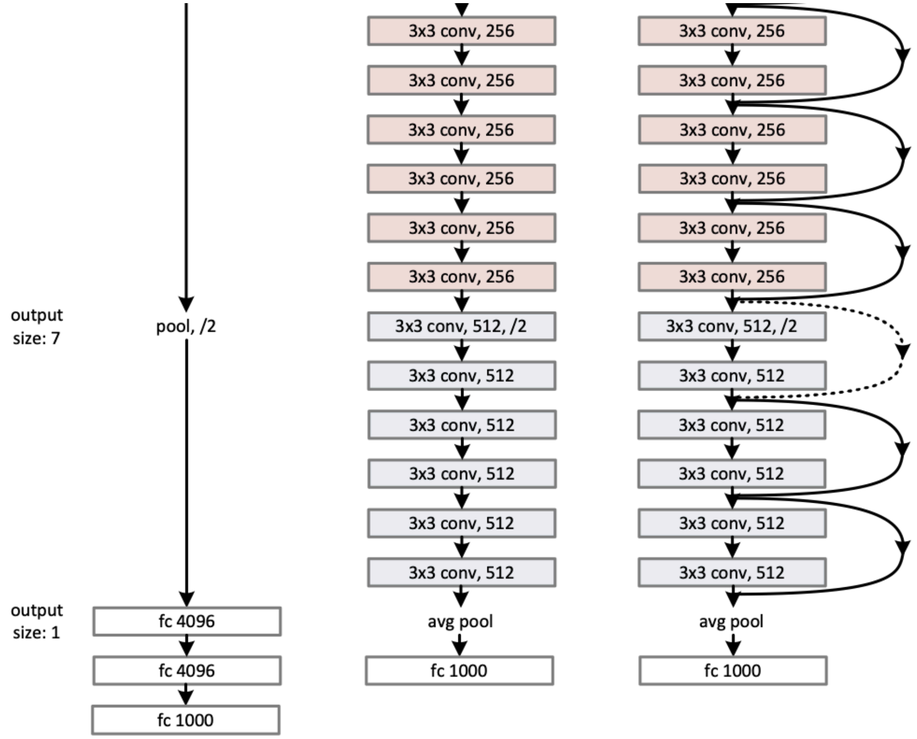
- SqueezeNet
- MobileNets V1/V2/V3
- EfficientNet V1

Next Up

NNs for training efficiency

- EfficientNet V2
- NFNets
- MCUNet V1 /V2 / V3





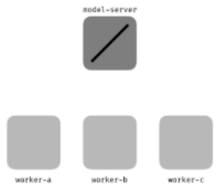
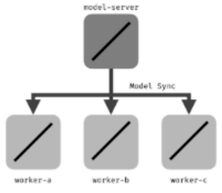
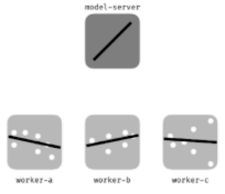
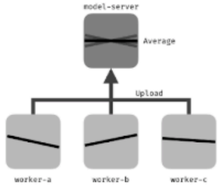
- **Typical Model.** (circa 2016) (1) Mainly built out of full-depth 3x3 convolutional layers
  (2) Number of input channels increase as you get deeper
  (3) Pool once in a while for DS
  (4) Fully connected layers at the end of the models

SqueezeNet
Iandola et al. (ICLR 2017)

Iandola et al., "SqueezeNet: AlexNet-level accuracy with 50x parameters and <0.5MB model size," ICLR 2017.

Motivation

- One of the first popular “small models that work well”
- **Focus.** Making a model with small number of parameters
  - *Distributed Computing.* Need to communicate weight parameters / gradients
  - *Exporting Models to Client.* Over-the-air update
  - *FPGA deployment.* FPGAs (in 2016) often have less than 10MB of on-chip memory
    e.g., Xilinx Virtex-7 FPGA has 8.5MB on-chip, and no off-chip

Step 1	Step 2	Step 3	Step 4
			
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data	Central server pools model results and generate one global mode without accessing any data

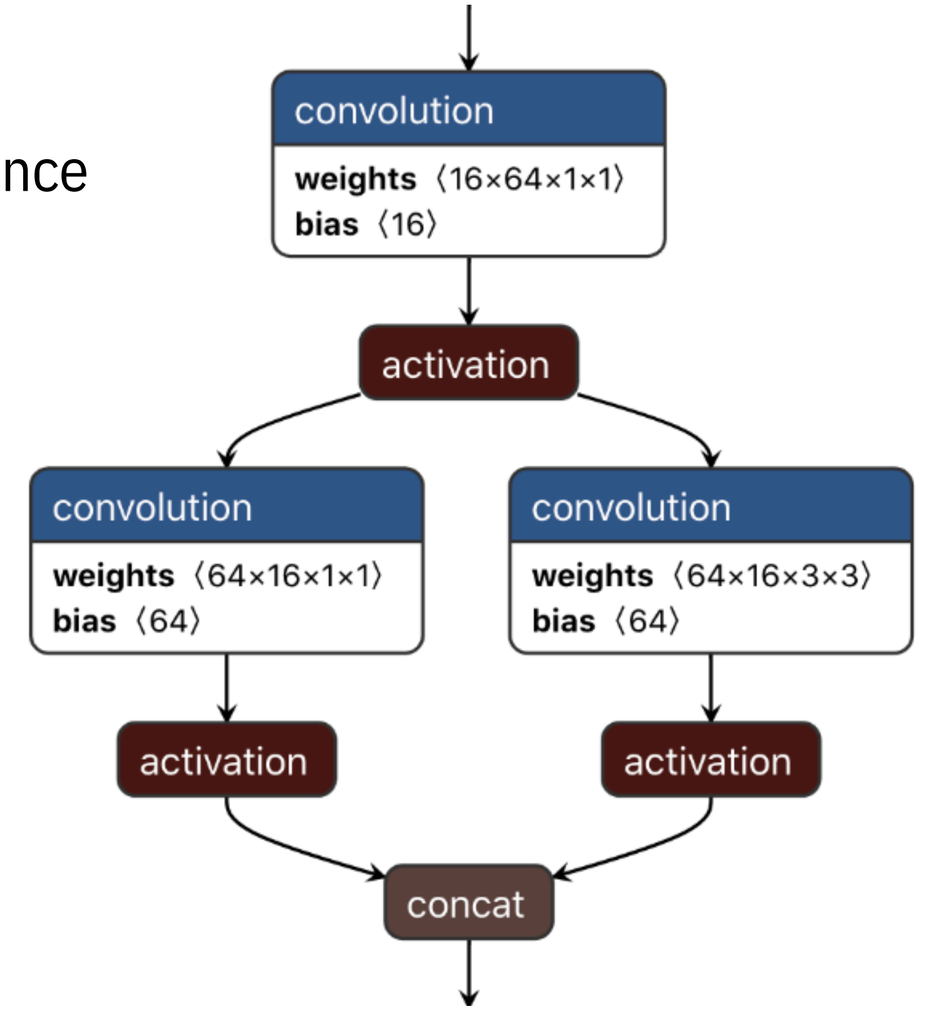
Idea

- **Problem.** 3x3 convolution sucks—avoid overusing it! (also avoid fully-connected)

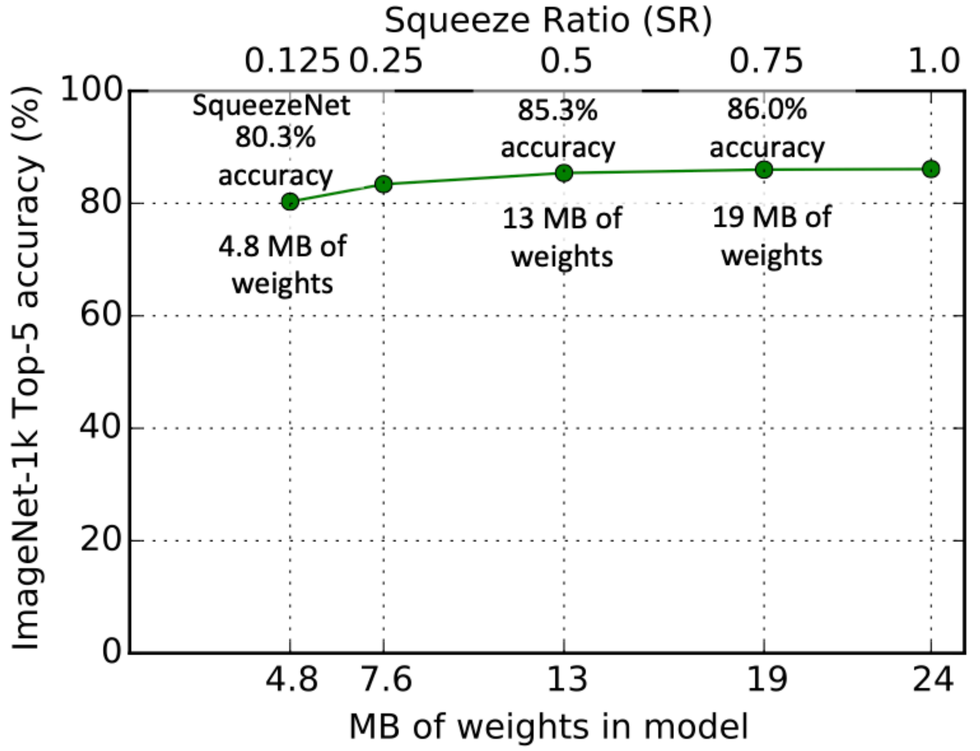
    1. Replace 3x3 filters with 1x1.
    2. Decrease the #input channels to 3x3 filters.
    3. Delay downsampling to later layers—helps performance

- **Approach.** A new module called 🔥Fire Module🔥

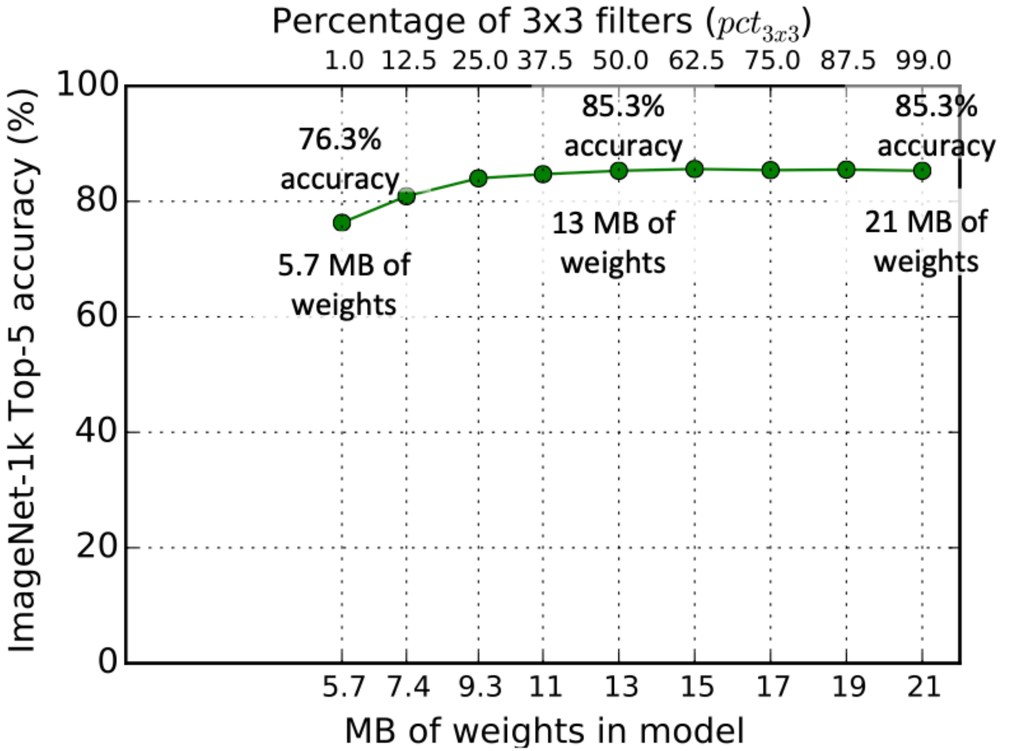
    A. *Squeeze:* Reduce #channels
    B. *Expand:* Use both 1x1 and 3x3, then concat.



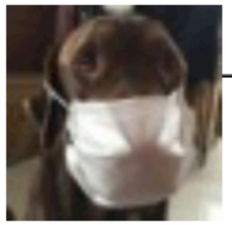
https://machinethink.net/blog/mobile-architectures/

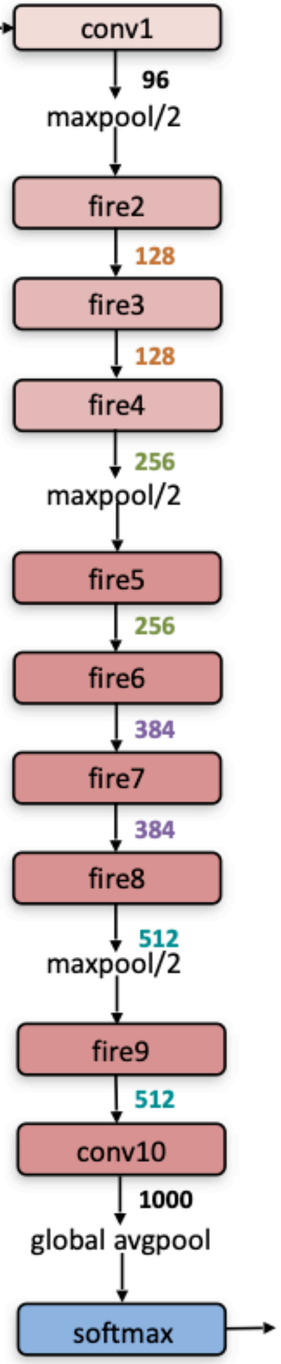


(a) Exploring the impact of the squeeze ratio (SR) on model size and accuracy.



(b) Exploring the impact of the ratio of 3x3 filters in expand layers (pct_{3x3}) on model size and accuracy.





layer name/type	output size	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3		-	-
conv1	111x111x96	6bit	14,208	14,208
maxpool1	55x55x96			
fire2	55x55x128	6bit	11,920	5,746
fire3	55x55x128	6bit	12,432	6,258
fire4	55x55x256	6bit	45,344	20,646
maxpool4	27x27x256			
fire5	27x27x256	6bit	49,440	24,742
fire6	27x27x384	6bit	104,880	44,700
fire7	27x27x384	6bit	111,024	46,236
fire8	27x27x512	6bit	188,992	77,581
maxpool8	13x12x512			
fire9	13x13x512	6bit	197,184	77,581
conv10	13x13x1000	6bit	513,000	103,400
avgpool10	1x1x1000			
			1,248,424 (total)	**421,098** (total)

	parameters	top-1
SqueezeNet v1.1	1.25 M	57.5%

	CPU	GPU	ANE
iPhone 11	0.0079	0.0118	0.0035
iPhone XS	0.0105	0.0164	0.0046
iPhone X	0.0129	0.0205	n/a

https://machinethink.net/blog/mobile-architectures/

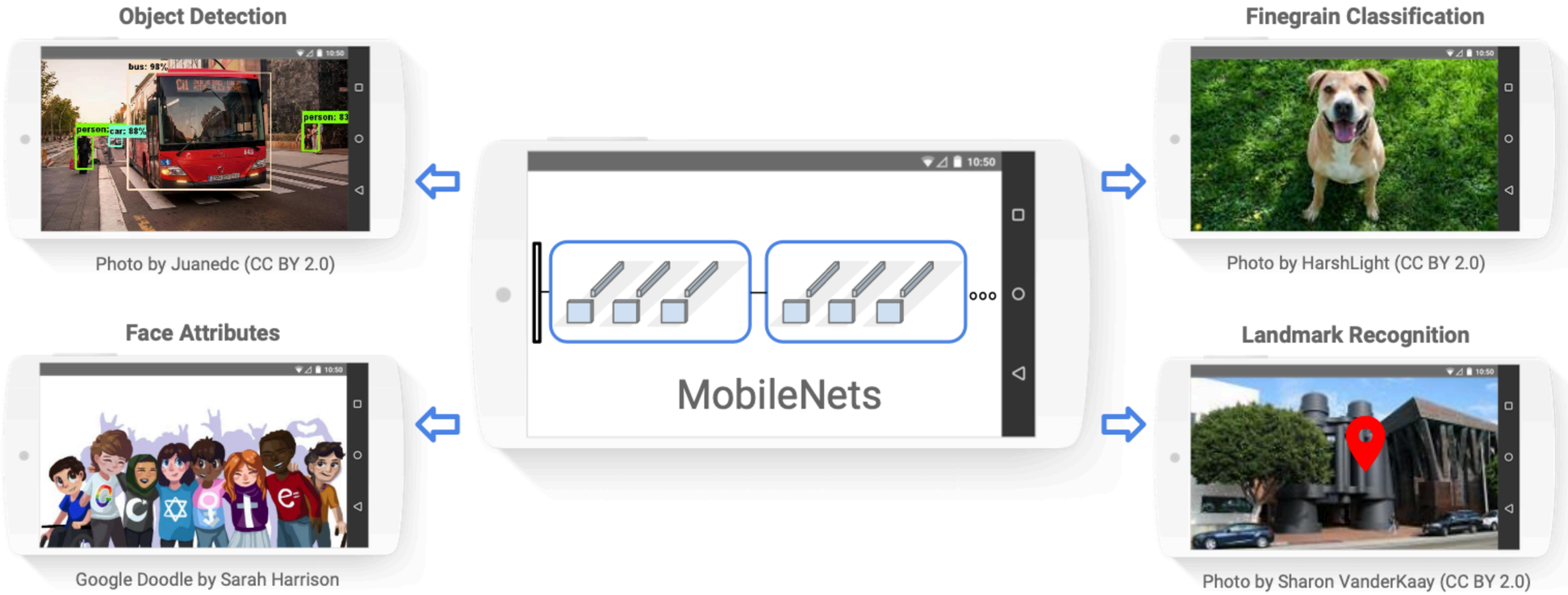
MobileNet
Howard et al. (arXiv 2017)

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

Motivation

- “Small models that work well” —> “Small models work really well”
  (ImageNet accuracy: 57.5% —> 70.9%)

- Also explicitly takes latency into account
  (but actually squeezenet is faster 🤔)

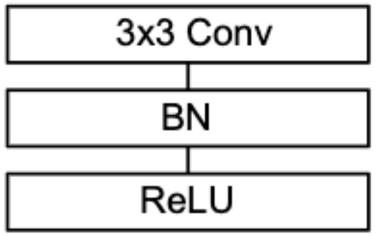


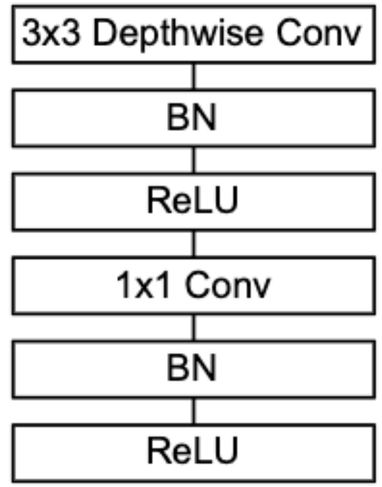
Idea

- **Problem.** Full-depth convolution sucks—use depthwise separable convolutions!
  (Previously used in Inception / GoogLeNet)

- **Approach.** Combine depthwise separable convolutions with 1x1 conv

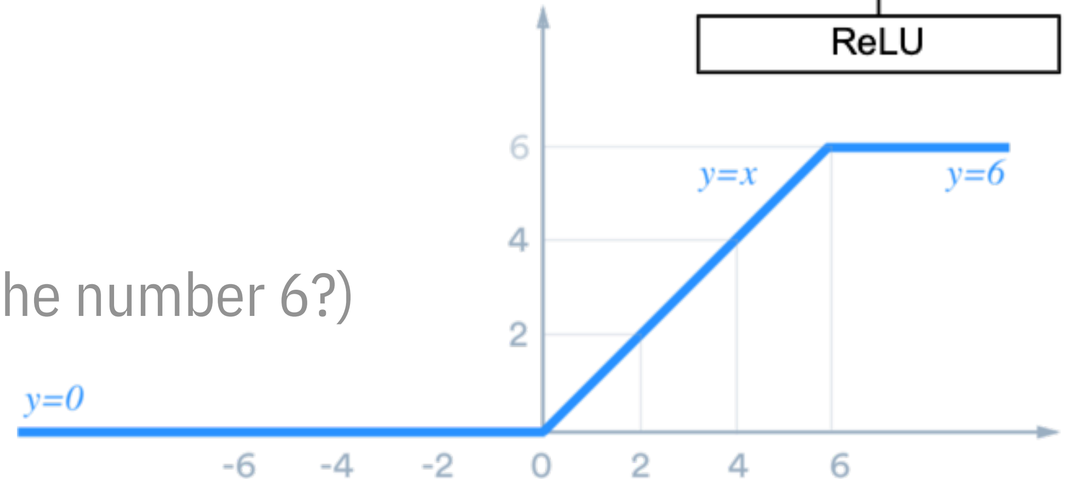
  A. *Depthwise:* Perform 3x3 convolution on each channel
  B. *Mixing:* Use full-depth 1x1 convolutions to mix channels





- **Scaling.** Two HPs: Width & Resolution multiplier
  Width: Number of channels
  Resolution: activation size

- **Details.** No max-pooling—just strided convolutions.
  Makes full use of BNs.
  Use ReLU6 instead of ReLU (quantization; why the number 6?)



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

	CPU	GPU	ANE
iPhone 11	0.0193	0.0243	0.0050
iPhone XS	0.0226	0.0362	0.0064
iPhone X	0.0237	0.0374	n/a

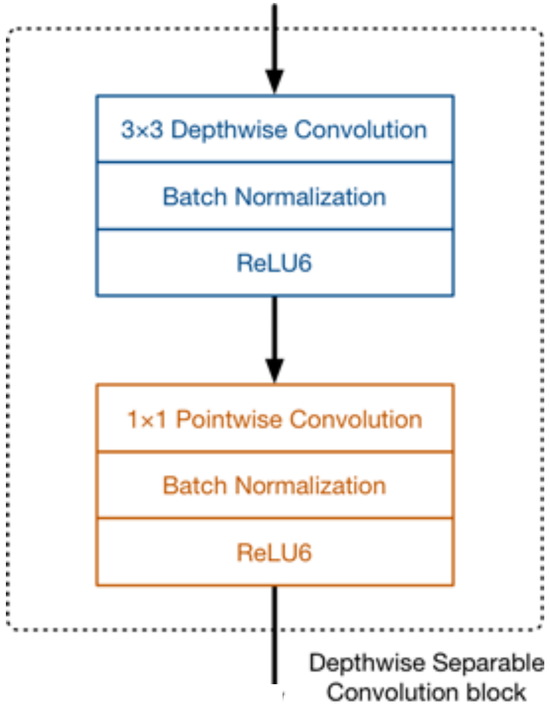
MobileNet v2
Sandler et al. (CVPR 2018)

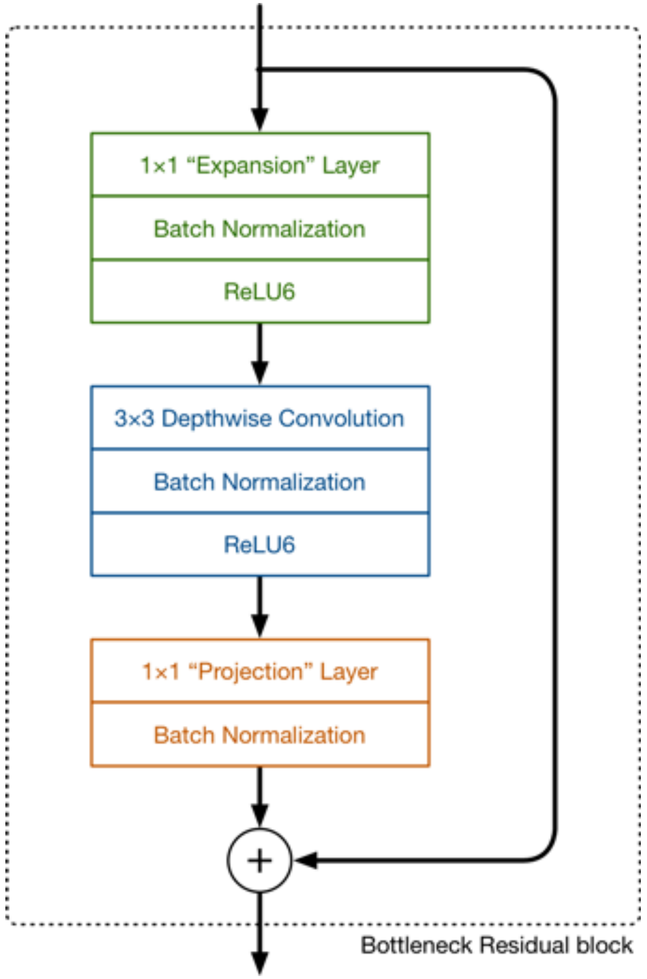
Sandler et al., “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” CVPR 2018

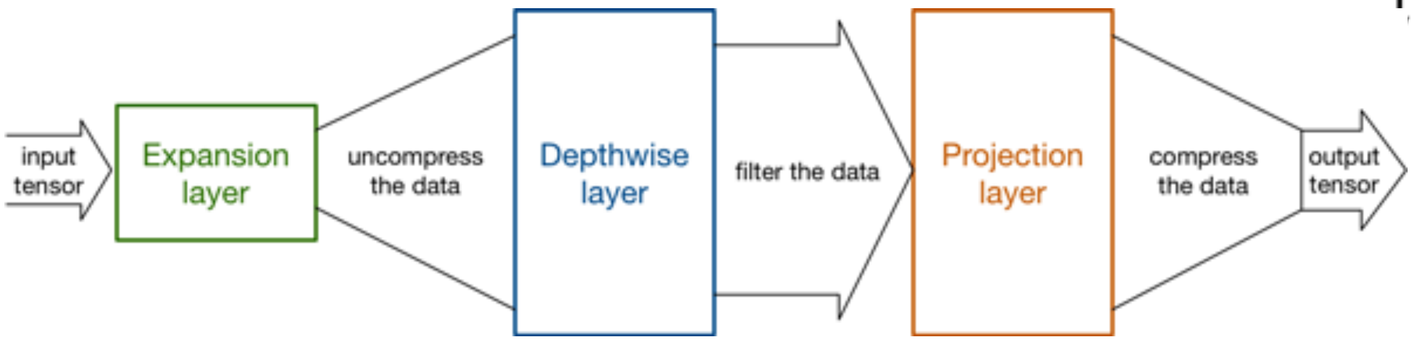
Innovation

- **Key.** Inverted residual with linear bottleneck
  (Re-arrange the depthwise conv!)

1. *Expand:* Increase the number of channels with 1x1 convolutions
2. *Depthwise:* 3x3 depthwise convolutions
3. *Project:* Decrease the number of channels







https://machinethink.net/blog/mobilenet-v2/

Input	Operator
$224^2 \times 3$	conv2d
$112^2 \times 32$	bottleneck
$112^2 \times 16$	bottleneck
$56^2 \times 24$	bottleneck
$28^2 \times 32$	bottleneck
$14^2 \times 64$	bottleneck
$14^2 \times 96$	bottleneck
$7^2 \times 160$	bottleneck
$7^2 \times 320$	conv2d 1x1
$7^2 \times 1280$	avgpool 7x7
$1 \times 1 \times 1280$	conv2d 1x1

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

	CPU	GPU	ANE
iPhone 11	0.0190	0.0185	0.0051
iPhone XS	0.0229	0.0284	0.0064
iPhone X	0.0276	0.0357	n/a

Much faster on GPU than v1, still slower than SqueezeNet.

Version	iPhone 7	iPhone X	iPad Pro 10.5	
MobileNet V1	118	162	204	
MobileNet V2	145	233	220	*(FPS)*

For optimal throughput I used a double-buffering approach where the next request is already being prepared (by the CPU) while the current one is still being processed (by the GPU). This way the CPU and GPU are never waiting for one another.

(Fun fact: for V2 it was actually worth doing triple buffering but for V1 that made no difference in speed. This shows that V2 is much more efficient.)

https://machinethink.net/blog/mobile-architectures/

MobileNet v3
Howard et al. (ICCV 2019)

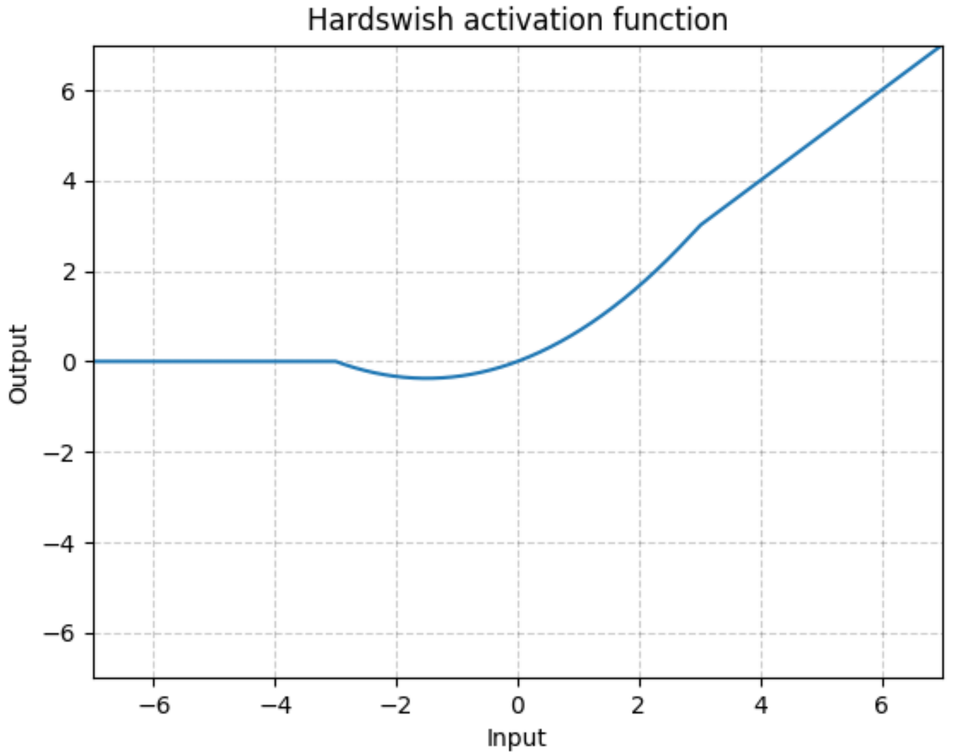
Howard et al., “Searching for MobileNetV3,” ICCV 2019

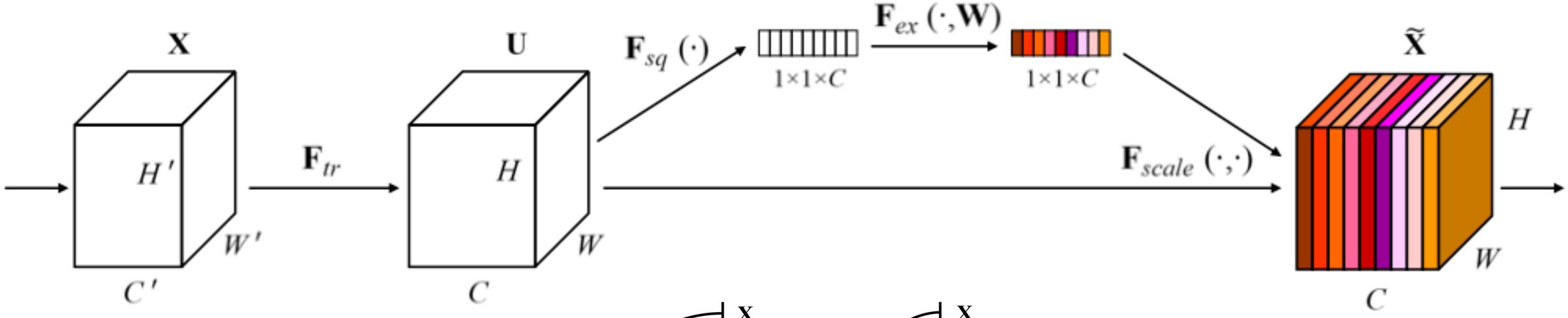
Model

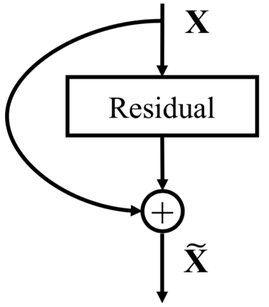
- **Approach.** Use Neural Architecture Search to explicitly search for a model with low latency
  (Similar to MNASNet; will talk more about NAS later)

- **Key Features.** (1) Change layer 1—3x3 conv layer (32 channels —> 16 channels)
  (Saves 2 ms and 10M MAdds)
  (2) ReLU6 -> Hard Swish
  (better accuracy)
  (3) Final layer: Reduced some layers
  (Saves 7ms and 30M MAdds)
  (4) Fixed ratio of the squeeze-and-excite

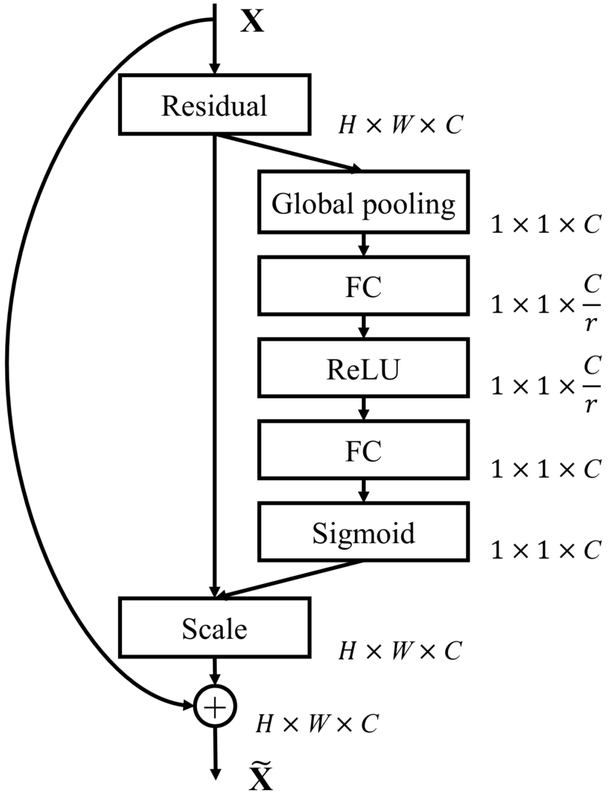
$$\text{Hardswish}(x) = \begin{cases} 0 & \text{if } x \leq -3, \\ x & \text{if } x \geq +3, \\ x \cdot (x+3)/6 & \text{otherwise} \end{cases}$$







ResNet Module



SE-ResNet Module

Hu et al., "Squeeze-and-Excitation networks," CVPR 2018

Input	Operator
$224^2 \times 3$	conv2d
$112^2 \times 16$	bneck, 3x3
$112^2 \times 16$	bneck, 3x3
$56^2 \times 24$	bneck, 3x3
$56^2 \times 24$	bneck, 5x5
$28^2 \times 40$	bneck, 5x5
$28^2 \times 40$	bneck, 5x5
$28^2 \times 40$	bneck, 3x3
$14^2 \times 80$	bneck, 3x3
$14^2 \times 80$	bneck, 3x3
$14^2 \times 80$	bneck, 3x3
$14^2 \times 80$	bneck, 3x3
$14^2 \times 112$	bneck, 3x3
$14^2 \times 112$	bneck, 5x5
$7^2 \times 160$	bneck, 5x5
$7^2 \times 160$	bneck, 5x5
$7^2 \times 160$	conv2d, 1x1
$7^2 \times 960$	pool, 7x7
$1^2 \times 960$	conv2d 1x1, NBN
$1^2 \times 1280$	conv2d 1x1, NBN

Network	Top-1	MAdds	Params	P-1	P-2	P-3
V3-Large 1.0	**75.2**	**219**	5.4M	**51**	**61**	**44**
V3-Large 0.75	73.3	155	4.0M	39	46	40
MnasNet-A1	75.2	315	3.9M	71	86	61
Proxyless[5]	74.6	320	4.0M	72	84	60
V2 1.0	72.0	300	3.4M	64	76	56
V3-Small 1.0	**67.4**	**56**	2.5M	**15.8**	**19.4**	**14.4**
V3-Small 0.75	65.4	44	2.0M	12.8	15.6	11.7
Mnas-small [43]	64.9	65.1	1.9M	20.3	24.2	17.2
V2 0.35	60.8	59.2	1.6M	16.6	19.6	13.9

Table 3. Floating point performance on the Pixel family of phones (P-n denotes a Pixel-n phone). All latencies are in ms and are measured using a single large core with a batch size of one. Top-1 accuracy is on ImageNet.

		CPU	GPU	ANE
MobileNet v3 (small)	iPhone 11	0.0063	0.0106	0.0102
	iPhone XS	0.0071	0.0176	0.0164
	iPhone X	0.0093	0.0211	n/a
MobileNet v3 (large)	iPhone 11	0.0152	0.0192	0.0169
	iPhone XS	0.0174	0.0264	0.0221
	iPhone X	0.0219	0.0332	n/a

ShuffleNets...

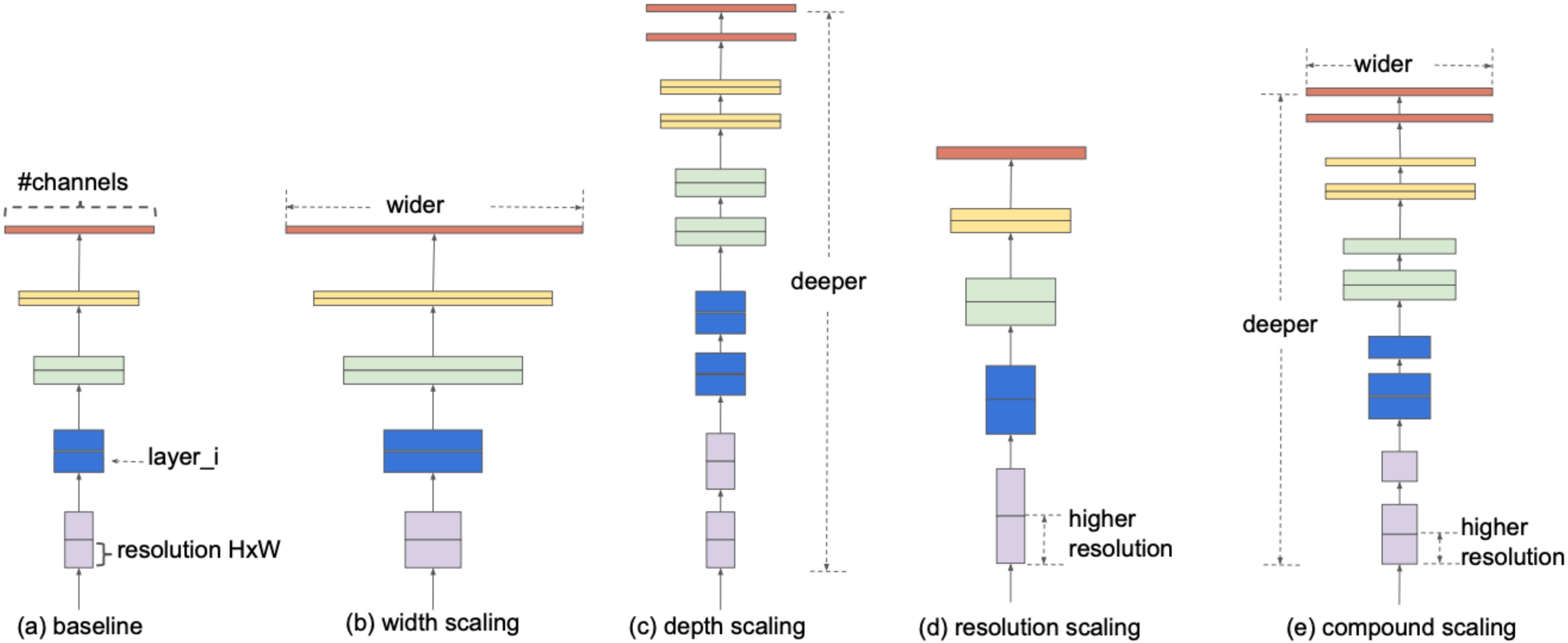
- These are important models—historically—but authors no longer argue that their main module works well! 🤯
  (grouped convolution + shuffle)
- The v2 paper has an interesting discussion regarding the relationship between architectural choice and the number of memory reads—strongly recommended!

EfficientNet
Tan and Le (ICML 2019)

Tan and Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019

Question

- We want to make models of various size... but how to scale up?
  (Depth/Width/Resolution/Compound)



Answer

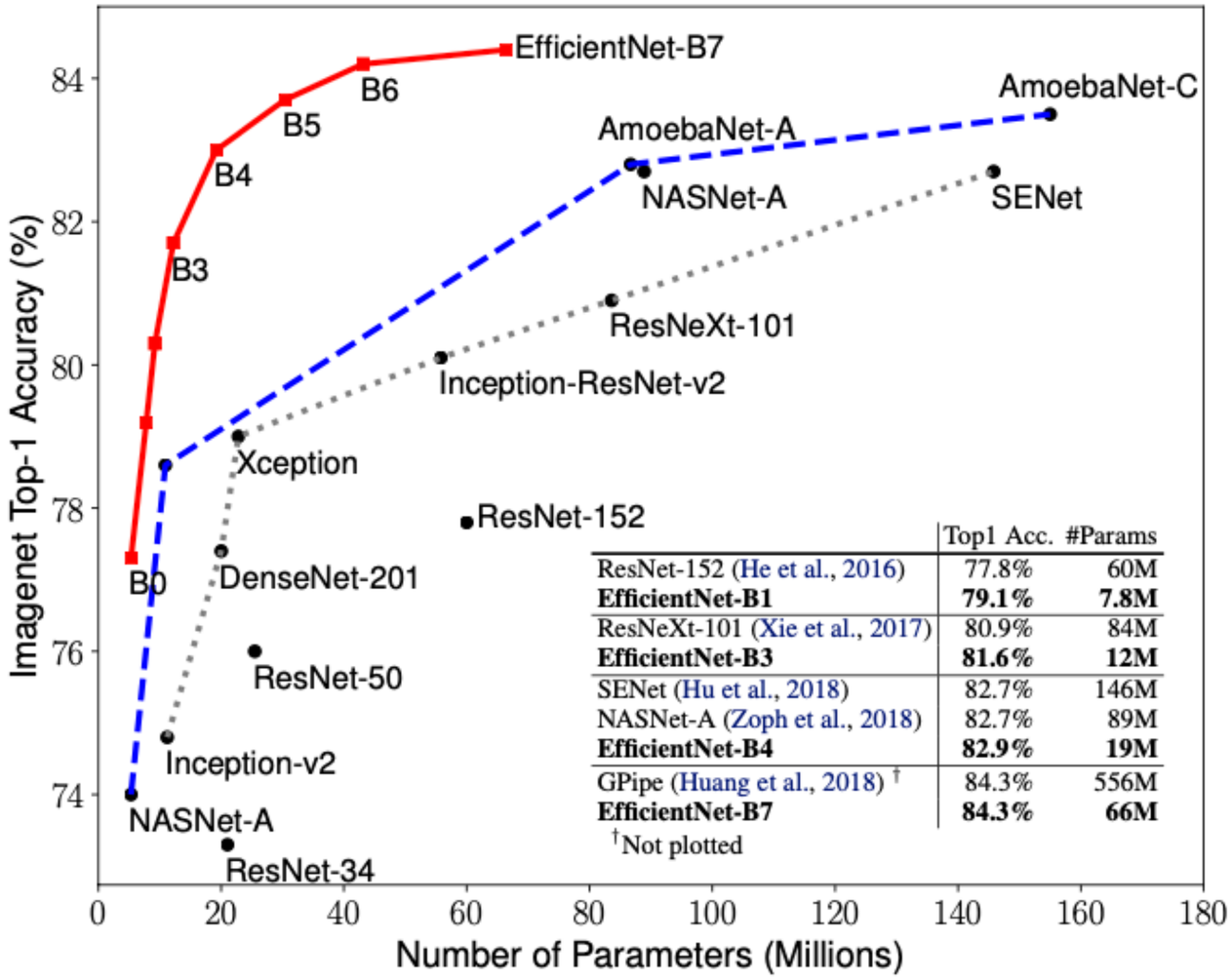
- Suppose that we want to use 2^N times more computational resource. Then, ...
  - Increase the network depth by α^N
  - Increase the width by β^N
  - Increase the image size by γ^N

    α, β, γ are determined by a grid search on the original model,

    $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$

- Also a new base model via NAS

  $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

- **Note.** SiLU activation, RMSProp optimizer

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

		CPU	GPU	ANE
EfficientNet-B0	iPhone 11	0.0309	0.0260	0.0253
	iPhone XS	0.0363	0.0365	0.0353
	iPhone X	0.0478	0.0461	n/a

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Next Up

Models that take “training” into accounts.

- NFNets: Cost of batch normalization
- EfficientNet V2: Fast training
- MCUNets: Models trainable on microcontrollers