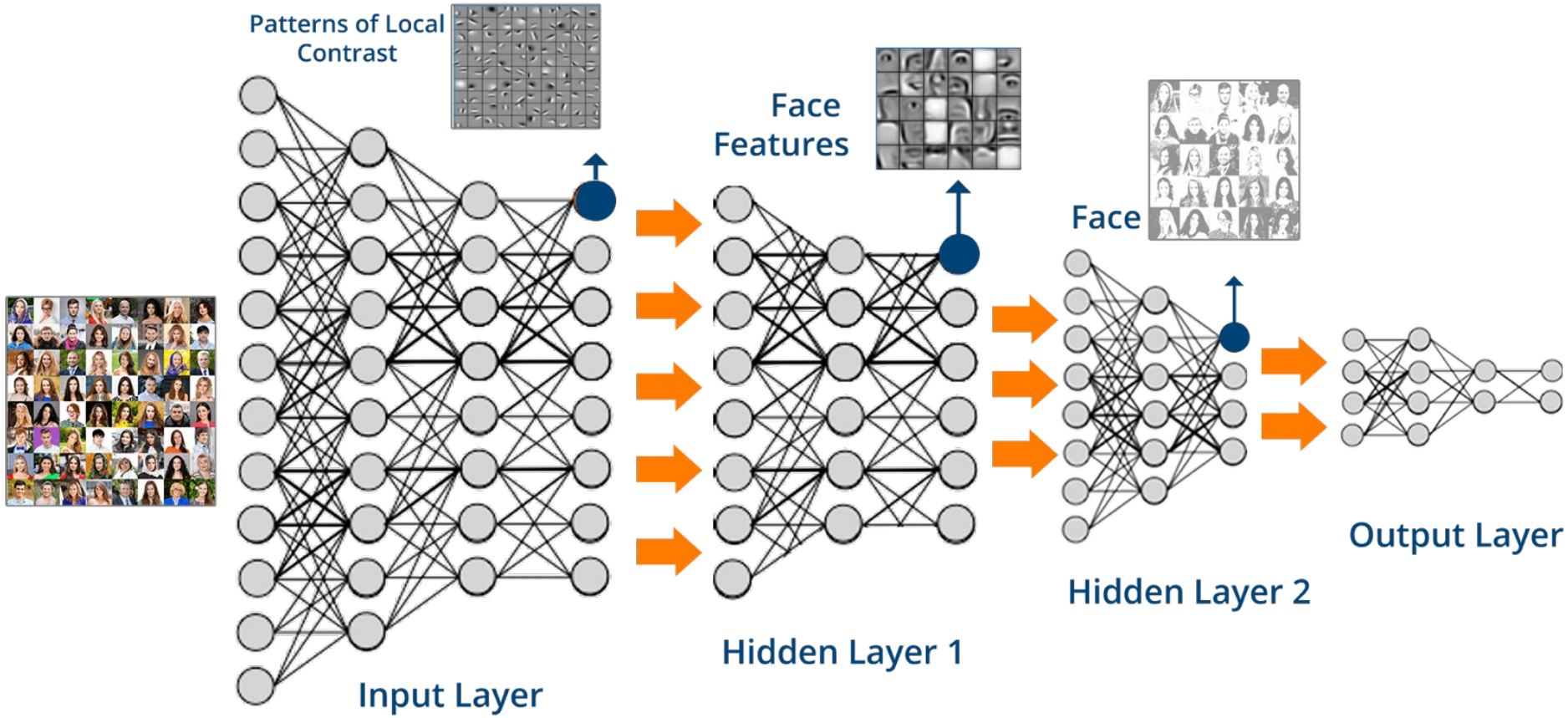
POLITECNICO
MILANO 1863

Hardware Architectures for Embedded and Edge AI

Prof Manuel Roveri – manuel.roveri@polimi.it

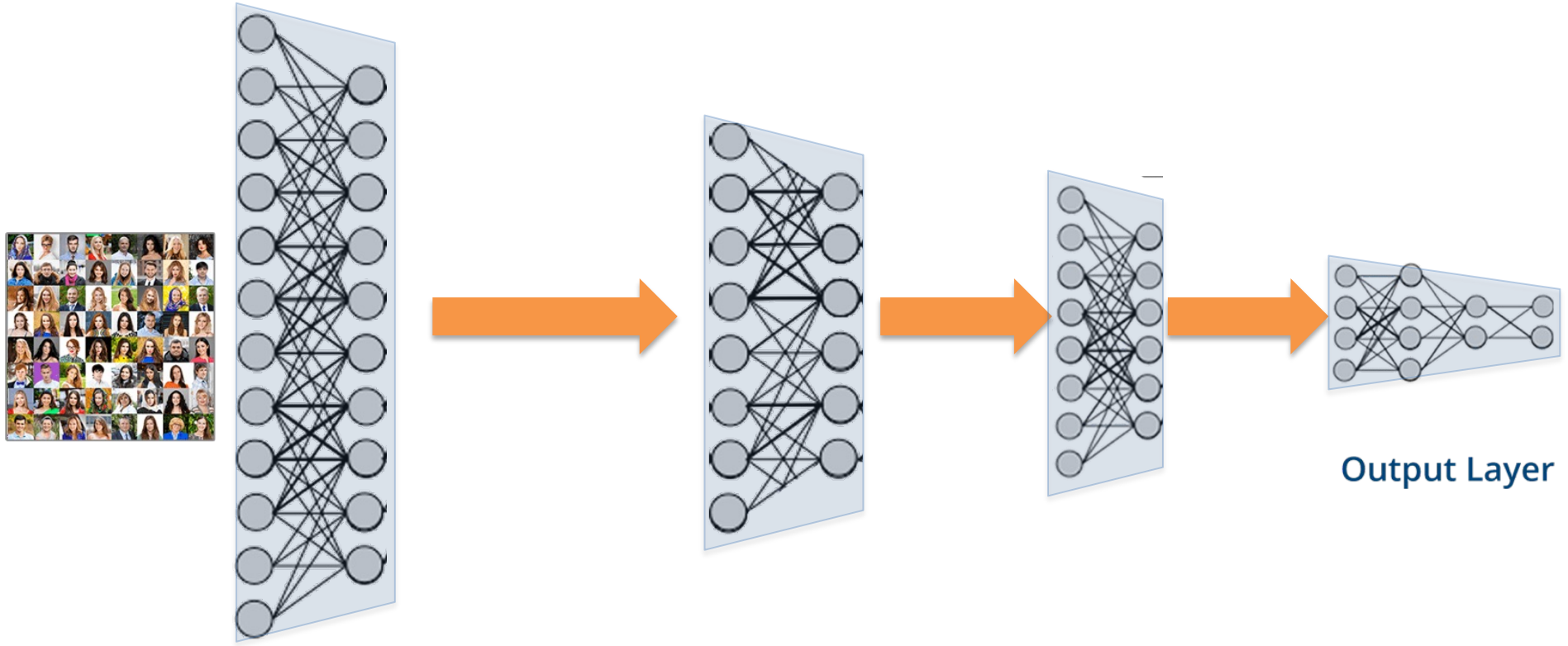
Lecture 6 – Tiny Architectures for Embedded and Edge AI

Approximate Deep Learning for IoT: Tiny Deep Learning



Approximate Deep Learning for IoT: Tiny Deep Learning

Re-design of the CNN architecture → Approximate Computing → Embedded System Code Optimization



Three families of architectures for tiny devices

Cit. Scholar 12/22: 7K

Cit. Scholar 12/22: 16K

Cit. Scholar 12/22: 9.5K

SqueezeNet: from 50x to 500x fewer params than AlexNet

Motivations:

- More efficient distributed training (i.e., communication overhead is directly proportional to the size of the model)
- Less overhead when exporting new models to clients (e.g., over-the-air update in Tesla's Autopilot)
- Feasible FPGA and embedded deployments (FPGAs often have less than 10MB of on-chip memory and no off-chip memory or storage)

Solutions:

- Identify a CNN architecture with fewer parameters but equivalent accuracy compared to AlexNet
- Identify a design space exploration for smaller NNs

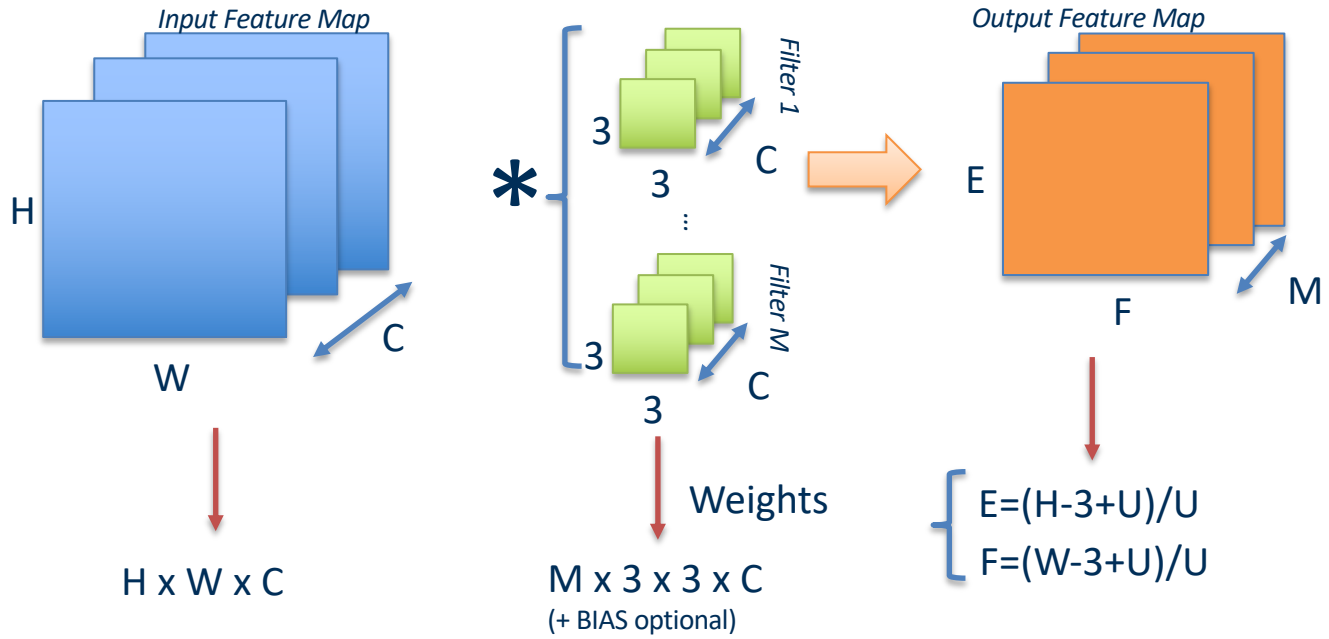
SqueezeNet: the basic blocks

Design Strategies

Fire module

SqueezeNet Architecture

Strategy #1: Replace 3x3 filters with 1x1 filters



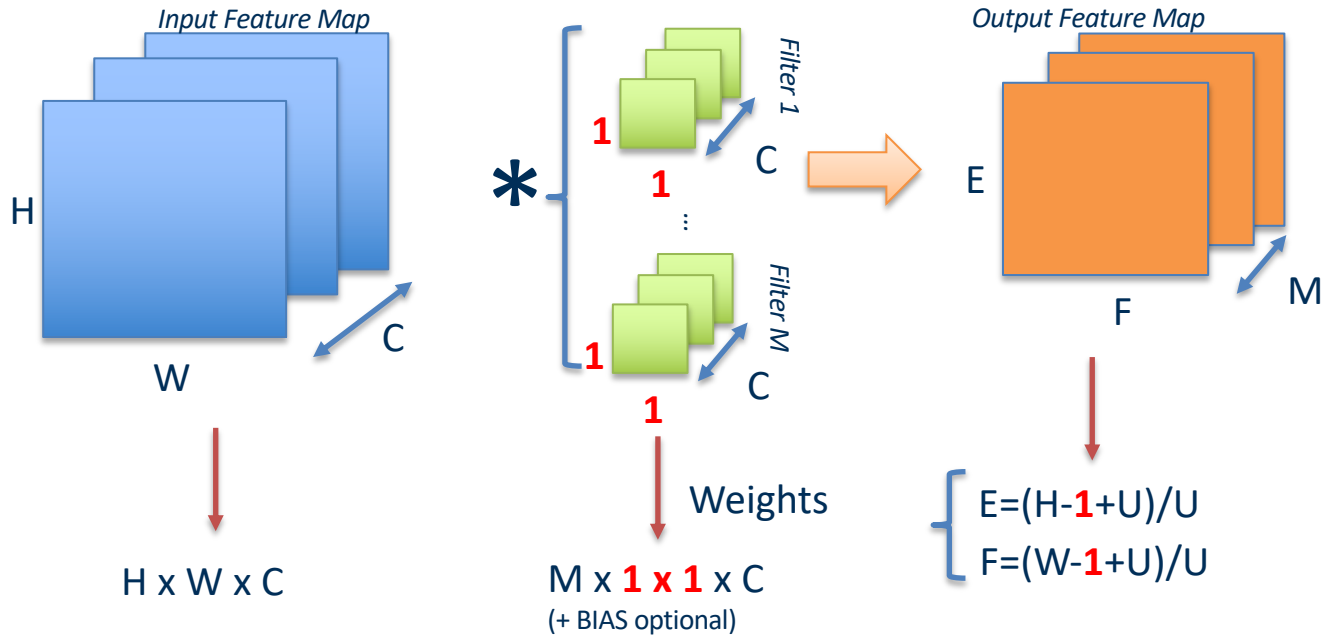
SqueezeNet: the basic blocks

Design Strategies

Fire module

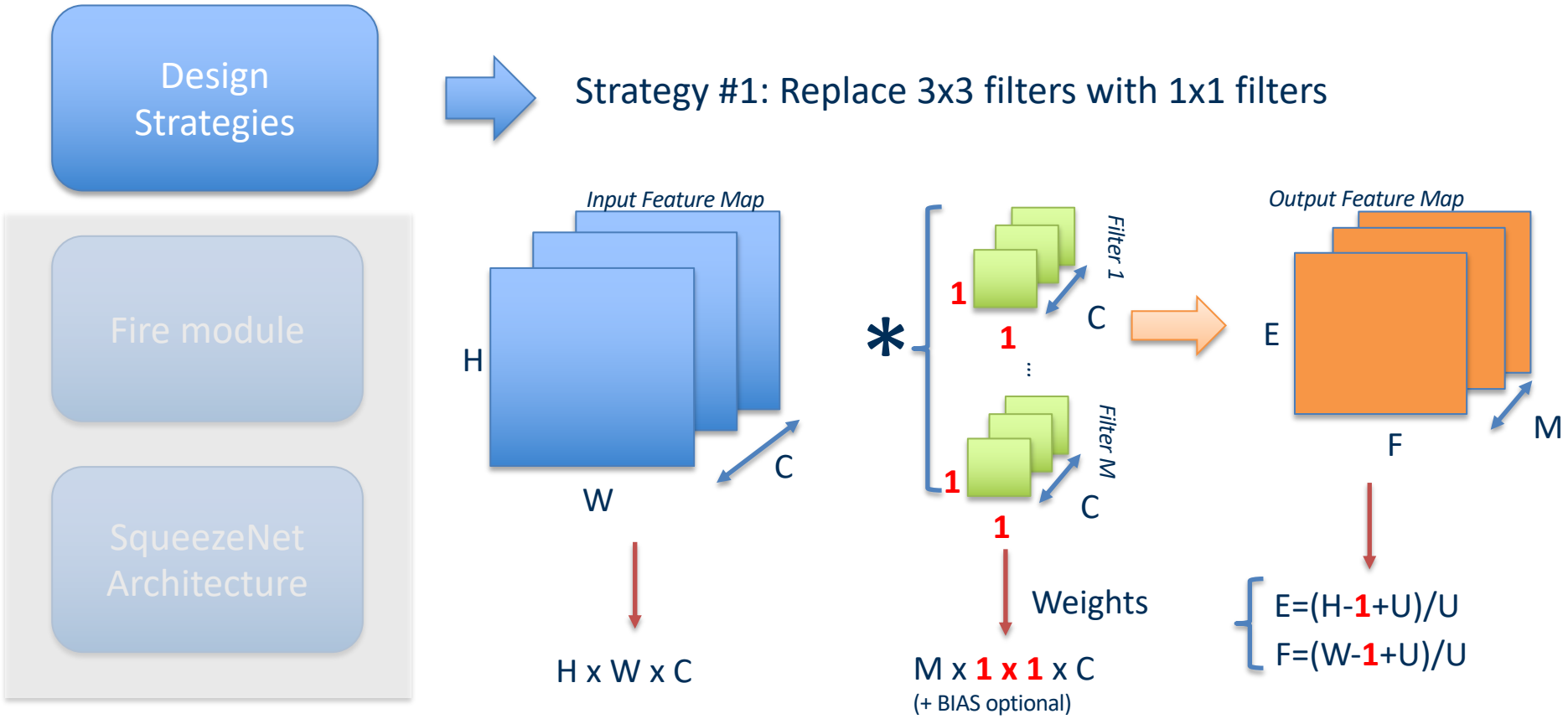
SqueezeNet Architecture

Strategy #1: Replace 3x3 filters with 1x1 filters



9x fewer parameters than 3x3 filters
(but not all the 3x3 filters are replaced)

SqueezeNet: the basic blocks



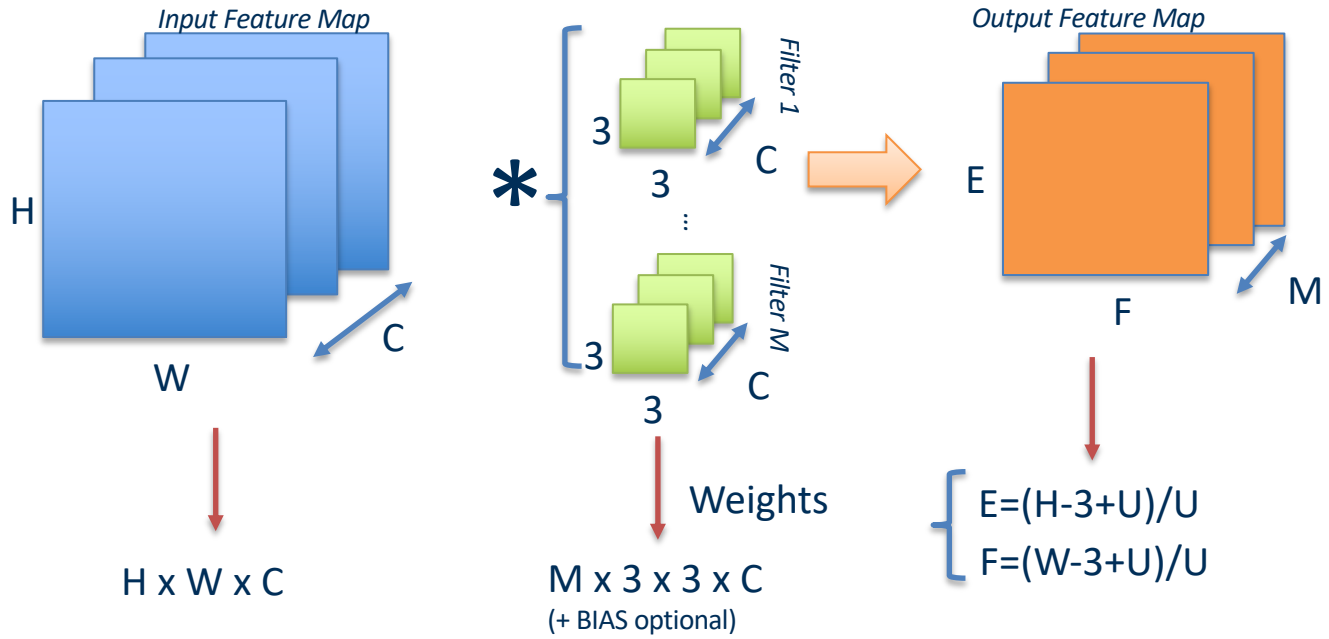
SqueezeNet: the basic blocks

Design Strategies

Fire module

SqueezeNet Architecture

Strategy #2: Reduce the number of input channels to 3x3 filters (by means of squeeze layers)



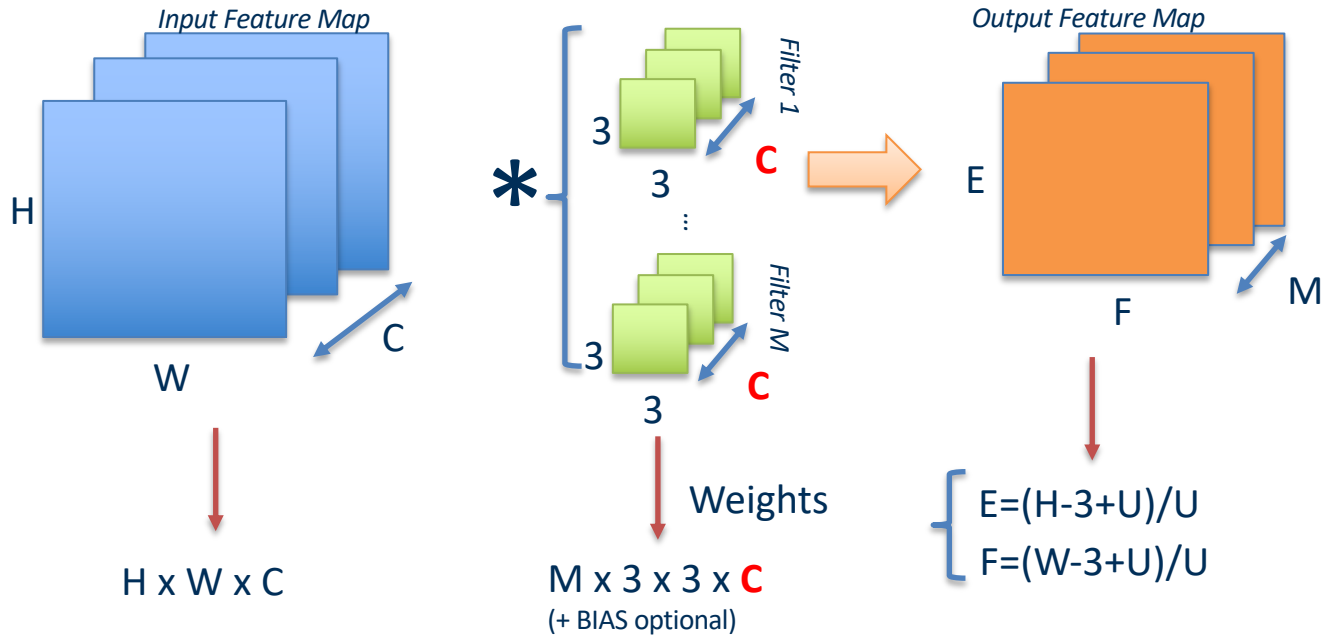
SqueezeNet: the basic blocks

Design Strategies

Fire module

SqueezeNet Architecture

Strategy #2: Reduce the number of input channels to 3x3 filters (by means of squeeze layers)



The total number of MAC is E x F x R x S x C x M

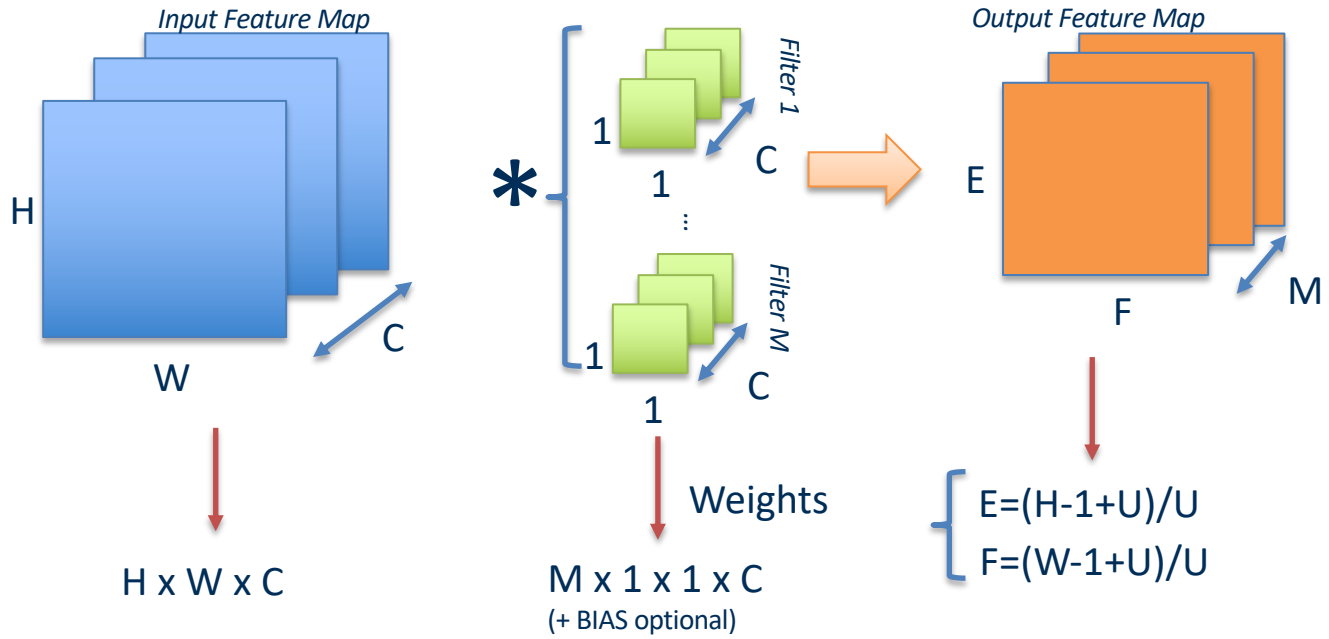
SqueezeNet: the basic blocks

Design Strategies

Fire module

SqueezeNet Architecture

Strategy #3: Downsample late in the NNs



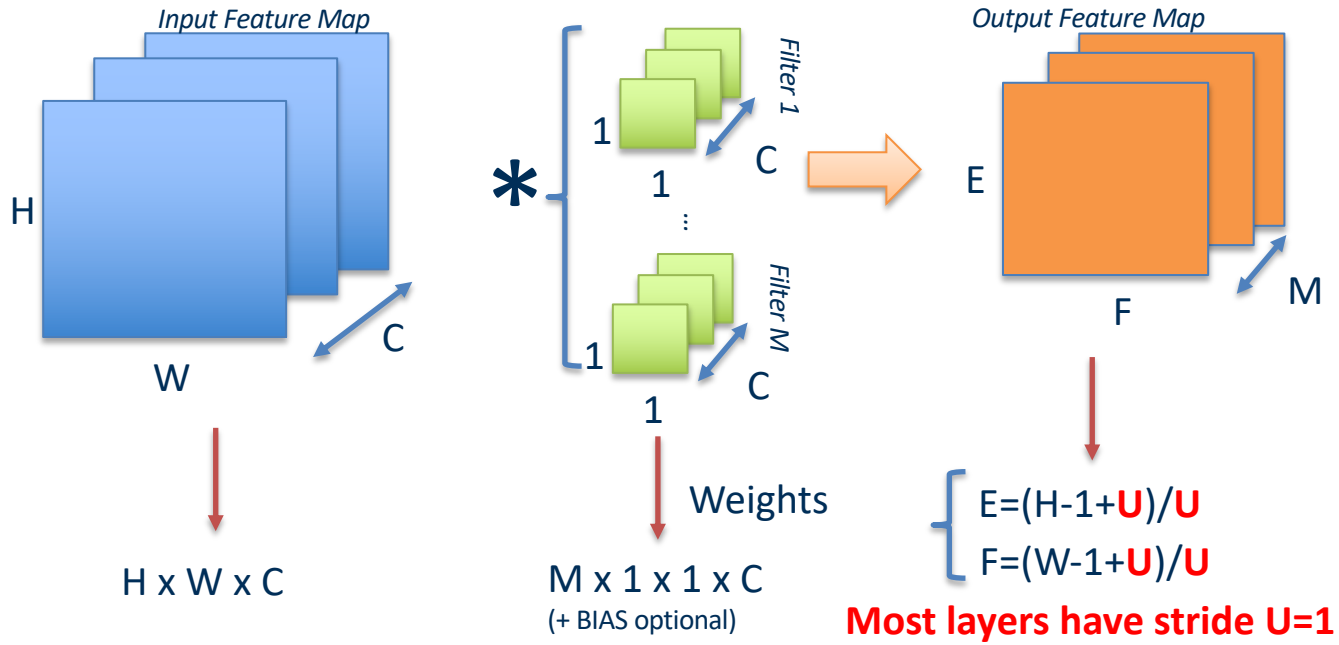
SqueezeNet: the basic blocks

Design Strategies

Fire module

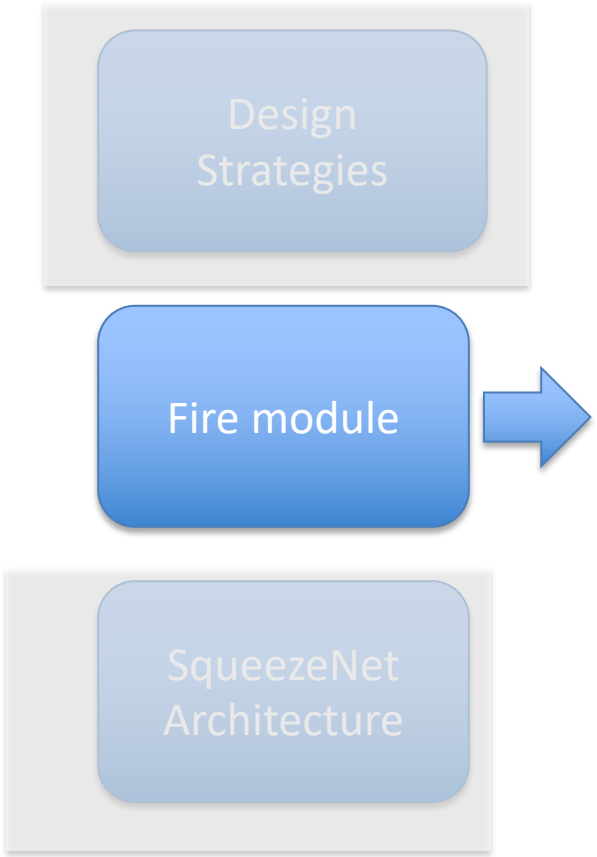
SqueezeNet Architecture

Strategy #3: Downsample late in the NNs

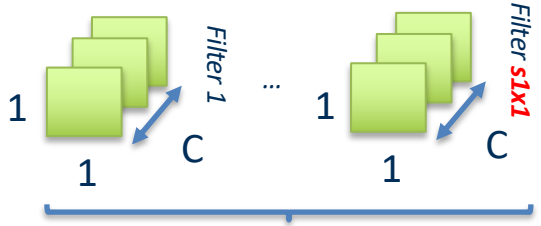


Large activation maps could lead to higher accuracy (with fewer weights) but at the expense of large large activation maps!

SqueezeNet: the basic blocks

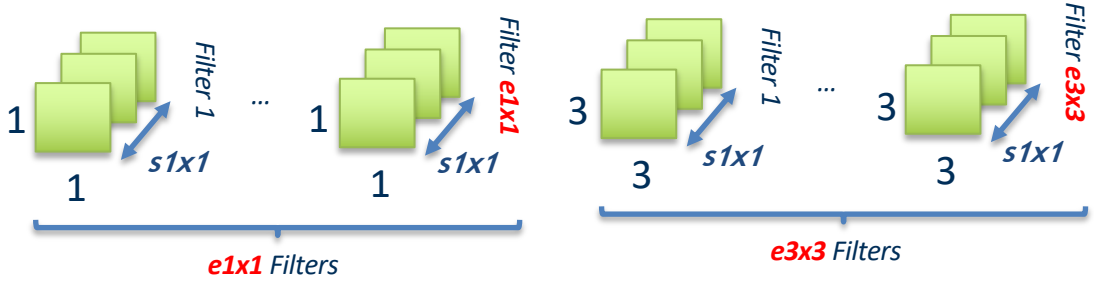


1) Squeeze: only 1x1 conv filters



(Strategy #1)

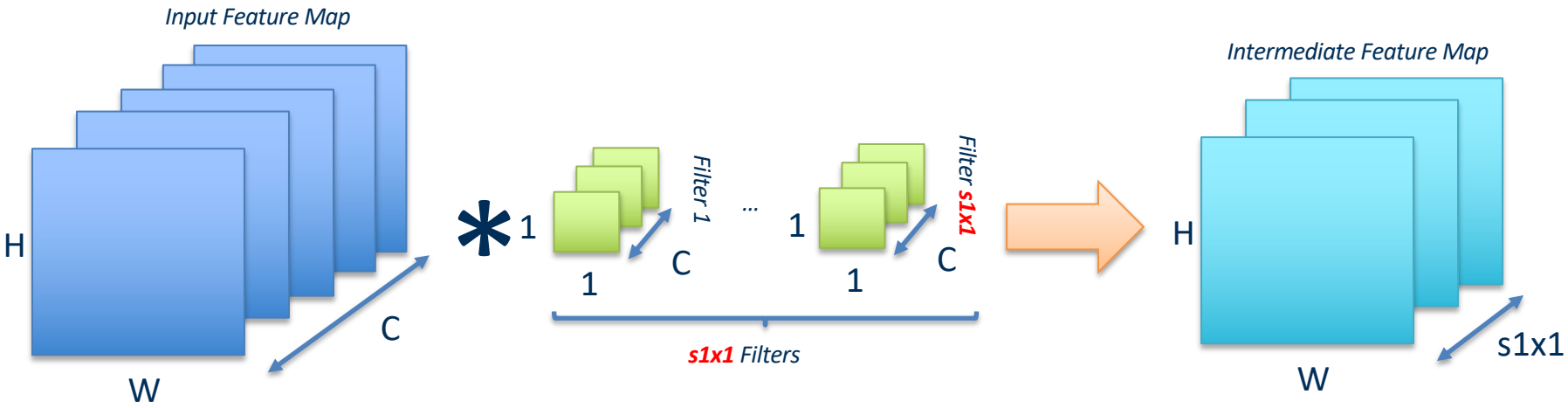
2) Expand: 1x1 and 3x3 conv filter



s1x1 < e1x1 + e3x3 (Strategy #2)

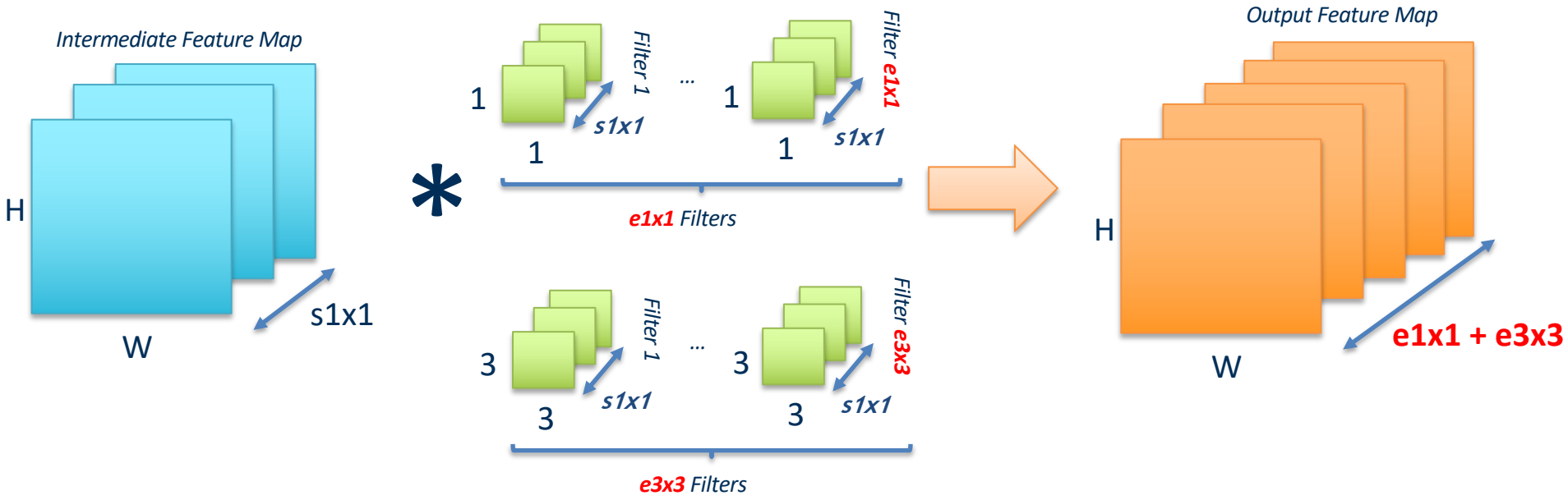
SqueezeNet: the fire module with activations

1) Squeeze: only 1x1 conv filters (with C > s1x1)



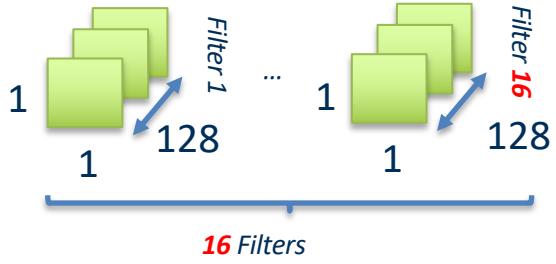
SqueezeNet: the fire module with activations

2) Expand: 1x1 and 3x3 conv filter

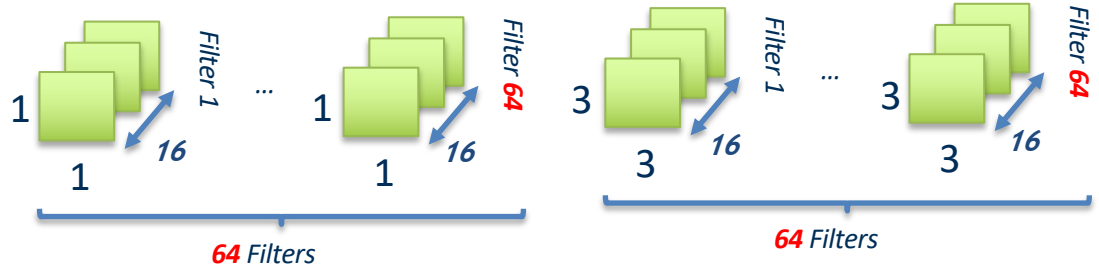


The fire module: a comparison with a 3x3 filter when C=128

1) Squeeze: 16 1x1 conv filters

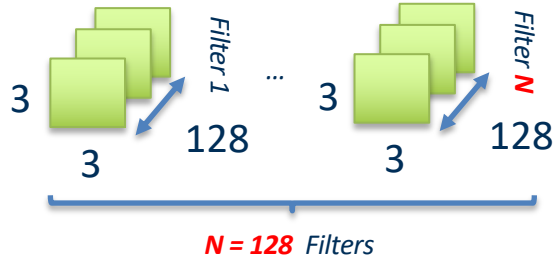


2) Expand: 64 1x1 and 64 3x3 conv filter



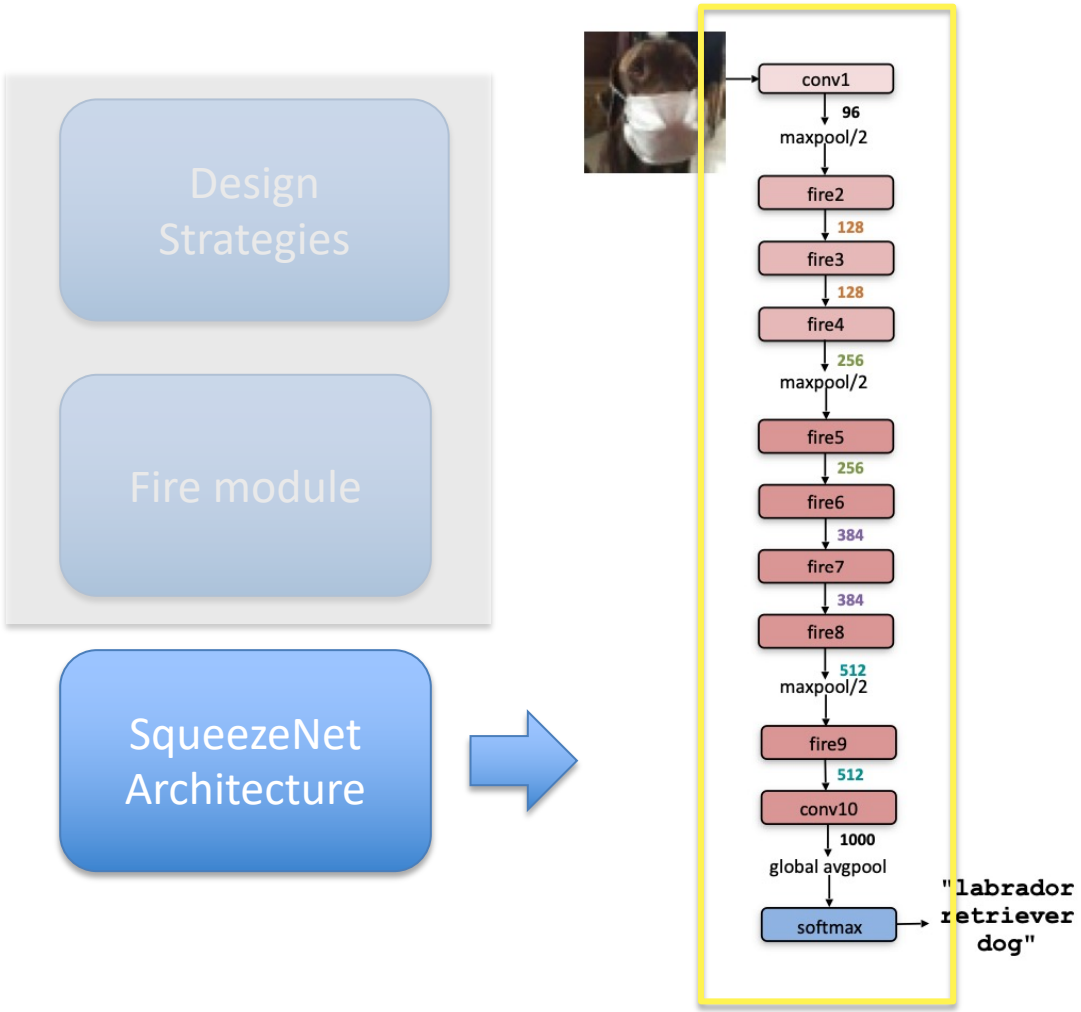
Total Weights = 1x1x128x16 + 1x1x16x64 + 3x3x16x64
= 2048+1024+9216=**12288**
(12432 =~12.5K with bias)

Only 3x3 conv filters



Total Weights = 3x3x128x128
=
147456=~147K

SqueezeNet: the architecture



The mean features:

- An initial conv layer
- 8 fire modules (with increasing number of filters)
- Maxpool after fire 4 and fire 8 (strategy #3)
- No fully connected layer (conv10 with 1000 filters + avg + softmax)

Image taken from [1]

SqueezeNet: the dimensions

Design Strategies

Fire module

SqueezeNet Architecture

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				**20%** (3x3)			6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations			parameters			compression info				1,248,424 (total)	**421,098** (total)

Further approximations can be applied:

- Sparsity of conv filter
- Quantization

Image taken from [1]

SqueezeNet: the accuracy

Design Strategies

Fire module

SqueezeNet Architecture

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

Deepen in next lectures

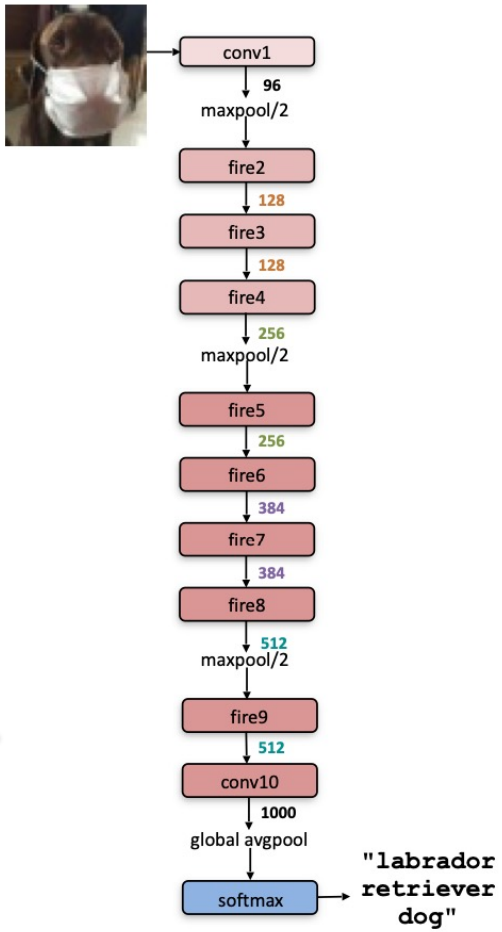
Image taken from [1]

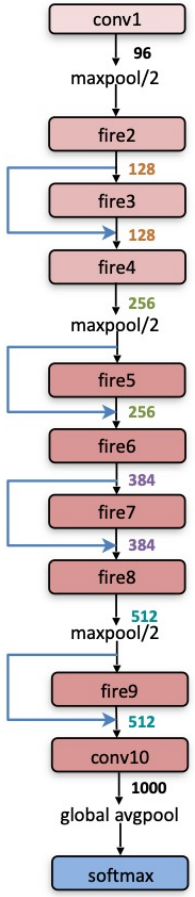
SqueezeNet: macro-architecture design space exploration

Design Strategies

Fire module

SqueezeNet Architecture





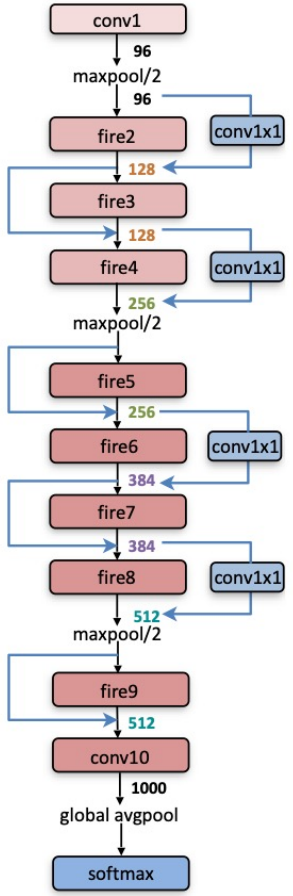


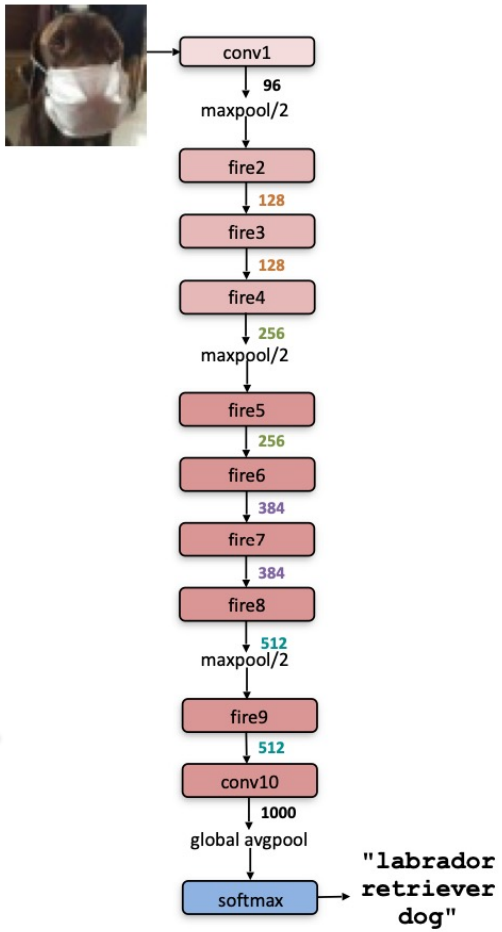
Image taken from [1]

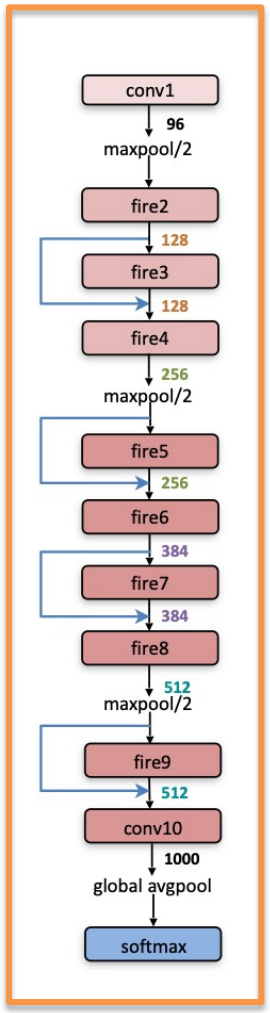
SqueezeNet: macro-architecture design space exploration

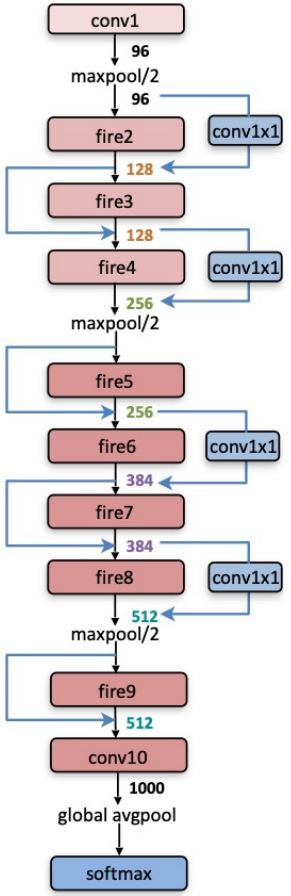
Design Strategies

Fire module

SqueezeNet Architecture







Simple bypass architecture adds bypass connections around Fire modules 3, 5, 7, and 9, requiring these modules to learn a residual function between input and output.

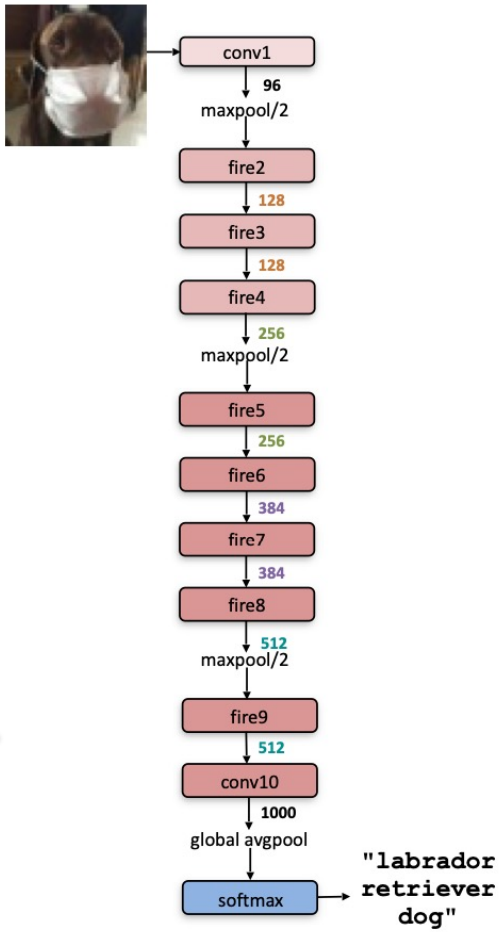
Image taken from [1]

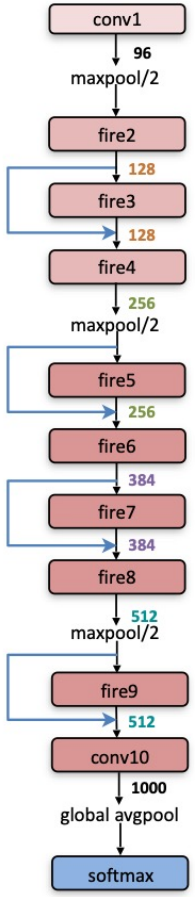
SqueezeNet: macro-architecture design space exploration

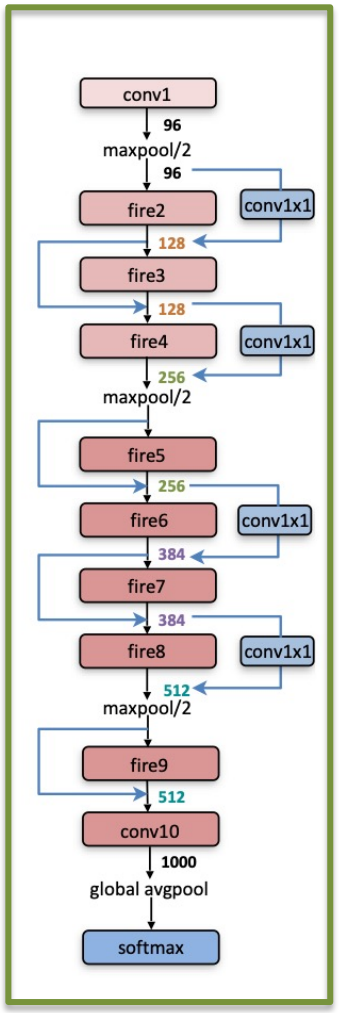
Design Strategies

Fire module

SqueezeNet Architecture







Complex bypass adds a bypass that includes a 1x1 convolution layer with the number of filters set equal to the number of output channels . Extra parameters to be trained

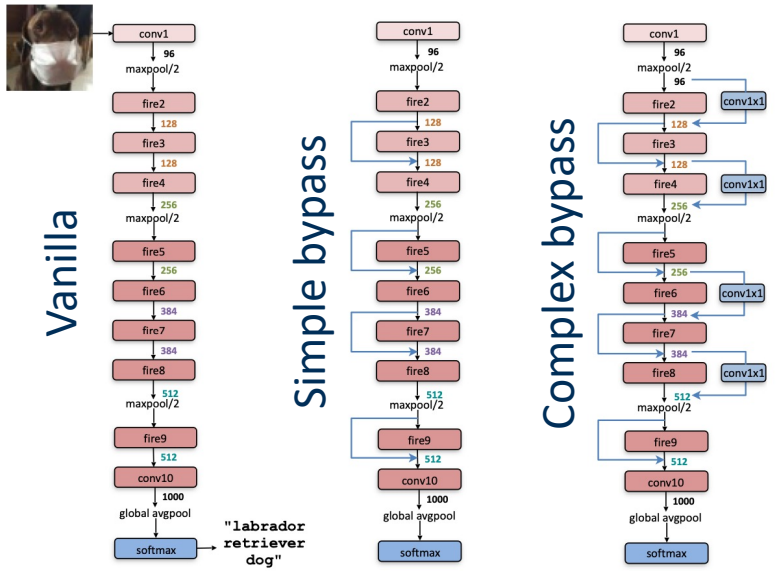
Image taken from [1]

SqueezeNet: macro-architecture design space exploration

Design Strategies

Fire module

SqueezeNet Architecture



Architecture	Top-1 Accuracy	Top-5 Accuracy	Model Size
Vanilla SqueezeNet	57.5%	80.3%	4.8MB
SqueezeNet + Simple Bypass	**60.4%**	**82.5%**	4.8MB
SqueezeNet + Complex Bypass	58.8%	82.0%	7.7MB

Image taken from [1]

Three families of architectures for tiny devices

A class of efficient models for mobile and embedded vision applications

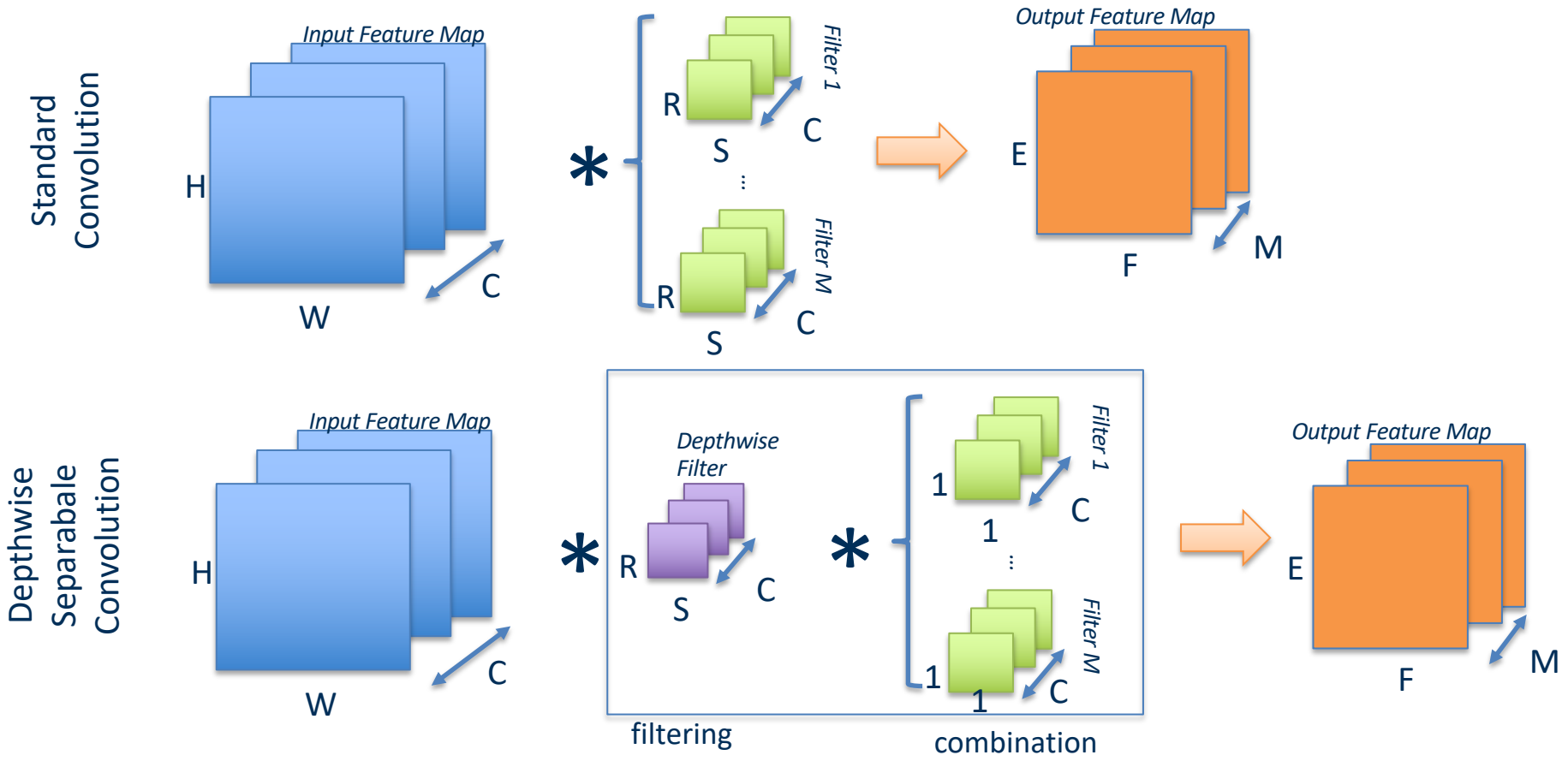
MobileNet (2017)

A streamlined architecture based on depth-wise separable convolutions to build light weight CNNs

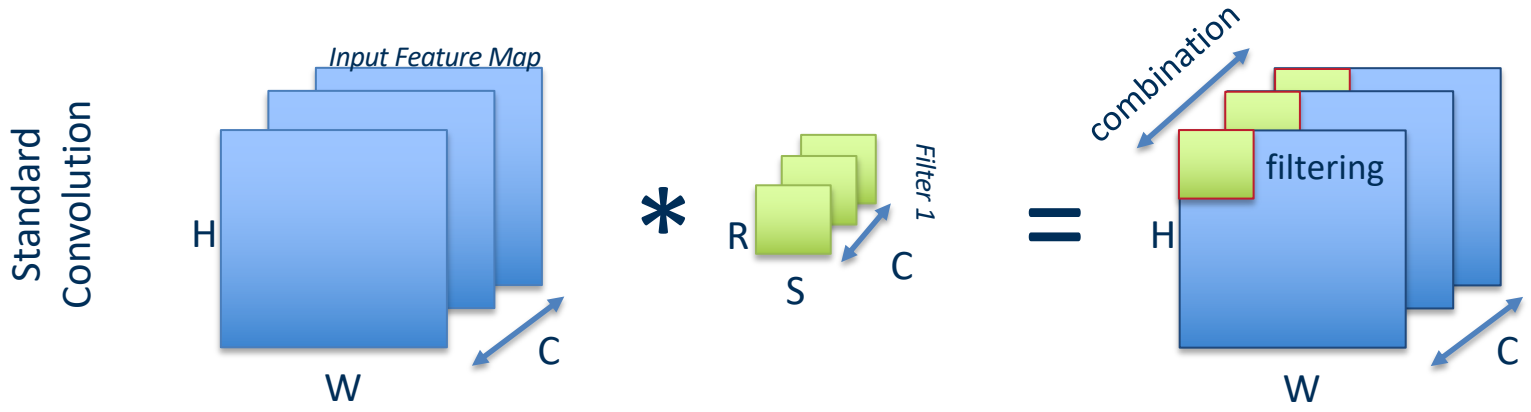
Cit. Scholar 12/22: 16K

Depthwise Separable Convolution

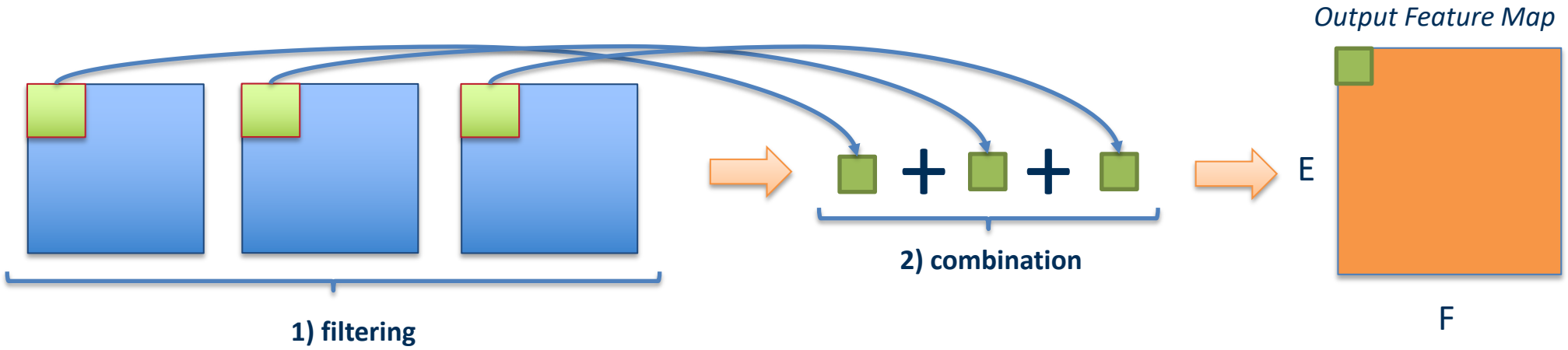
A form of factorized convolutions which factorize a standard convolution into a **depthwise convolution** and a **1 × 1 pointwise convolution**



Let's go back for a while to the standard convolution…



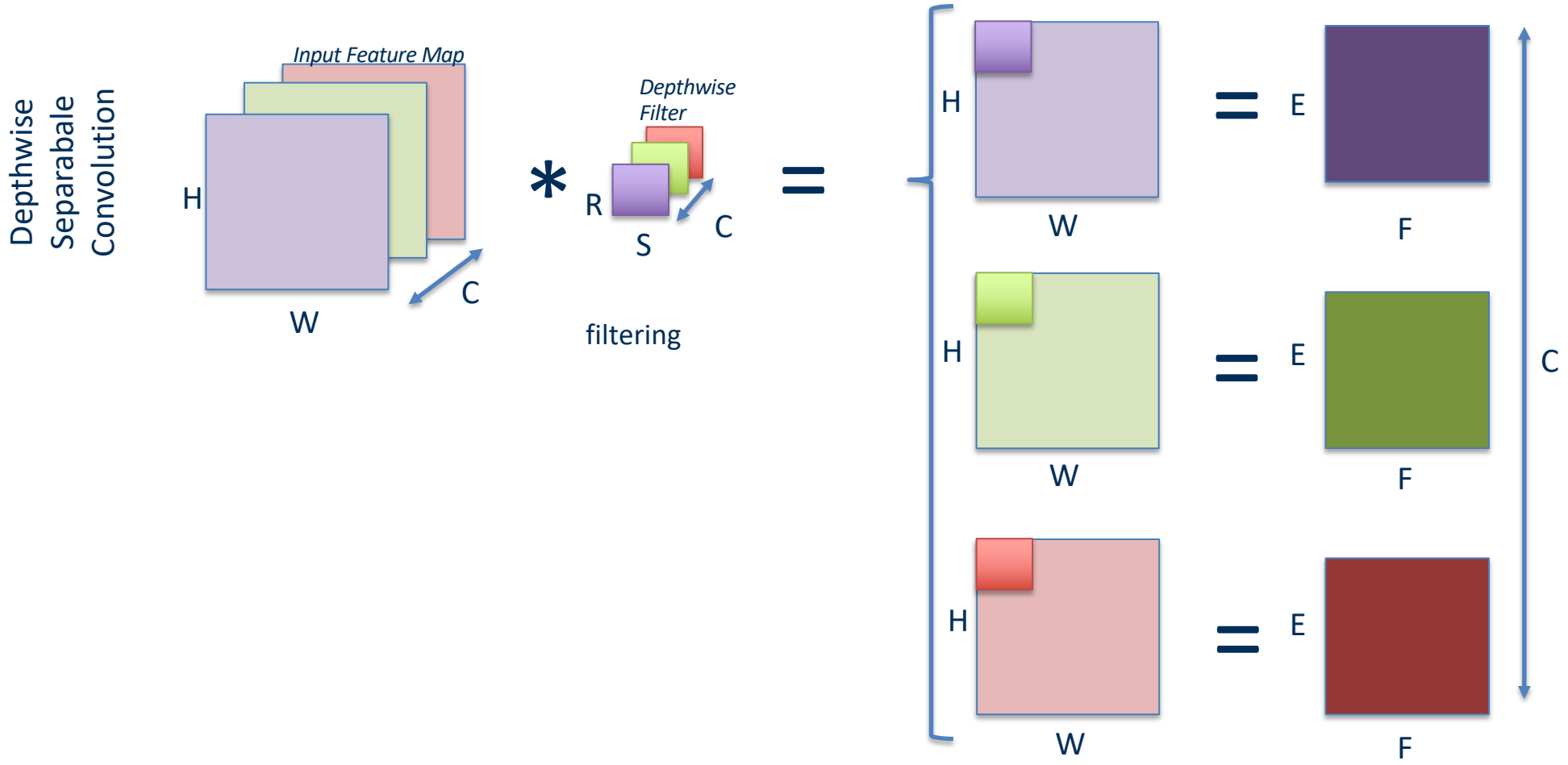
MAC ops : R x S x C



MAC ops: R x S (each)

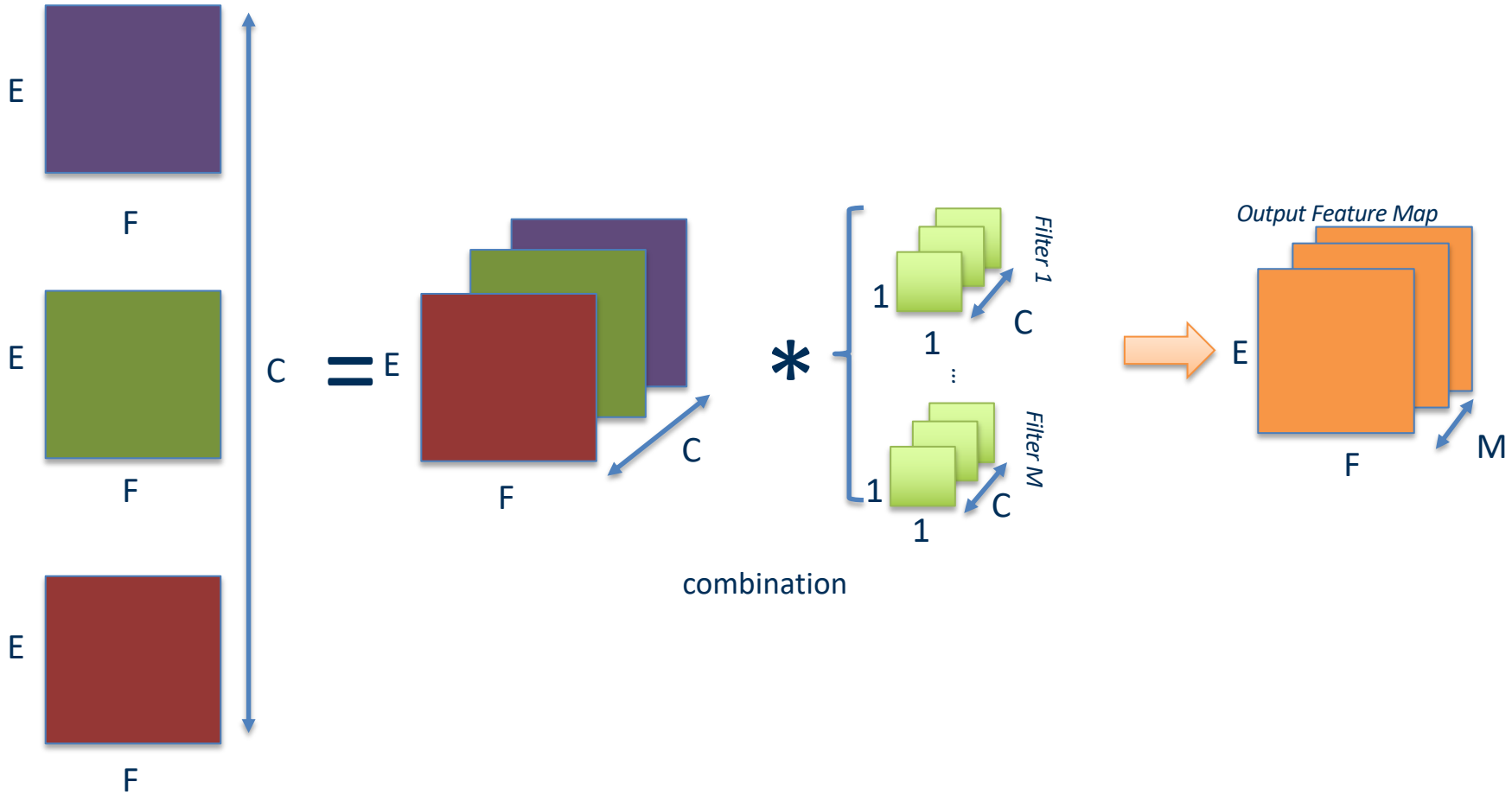
MAC ops per filter:
R x S x C x E x F

Depthwise separable conv. (step 1): depthwise filters



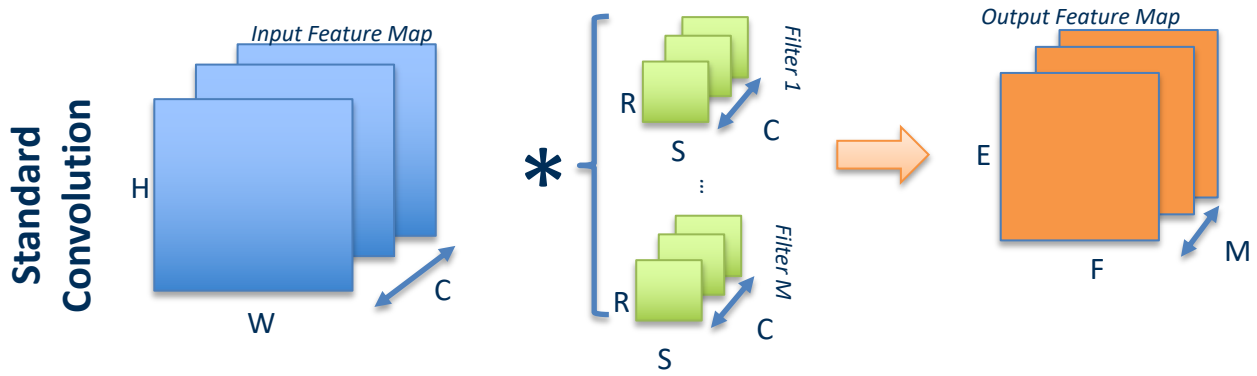
MAC operations: R x S x C x E x F

Depthwise separable conv. (step 2): 1×1 pointwise convolution



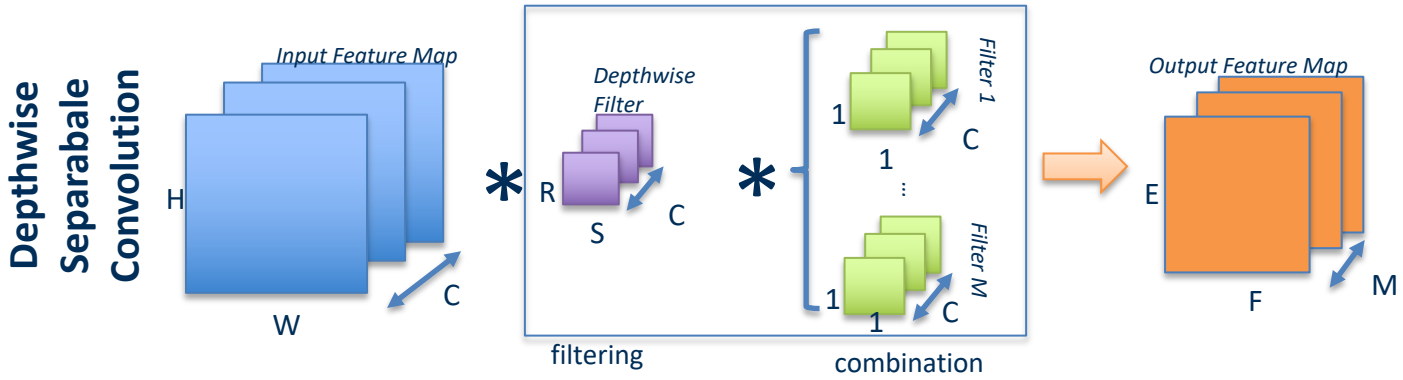
MAC operations: C x M x E x F

Standard vs. Depthwise Separable Convolution



MAC ops :
R x S x C x E x F x M

Number of weights:
R x S x C x M



MAC ops :
R x S x C x E x F
+ C x M x E x F

Number of weights:
R x S x C + C x M

Standard vs. Depthwise Separable Convolution: MAC and Weights

Convolution	MAC ops	#Weights
Standard	R x S x C x E x F x M	R x S x C x M
Depthwise	R x S x C x E x F + C x M x E x F	R x S x C + C x M

MAC ops ratio :

$$\frac{R \times S \times C \times E \times F + C \times M \times E \times F}{R \times S \times C \times E \times F \times M} = \frac{1}{M} + \frac{1}{R \times S} = 0.112$$

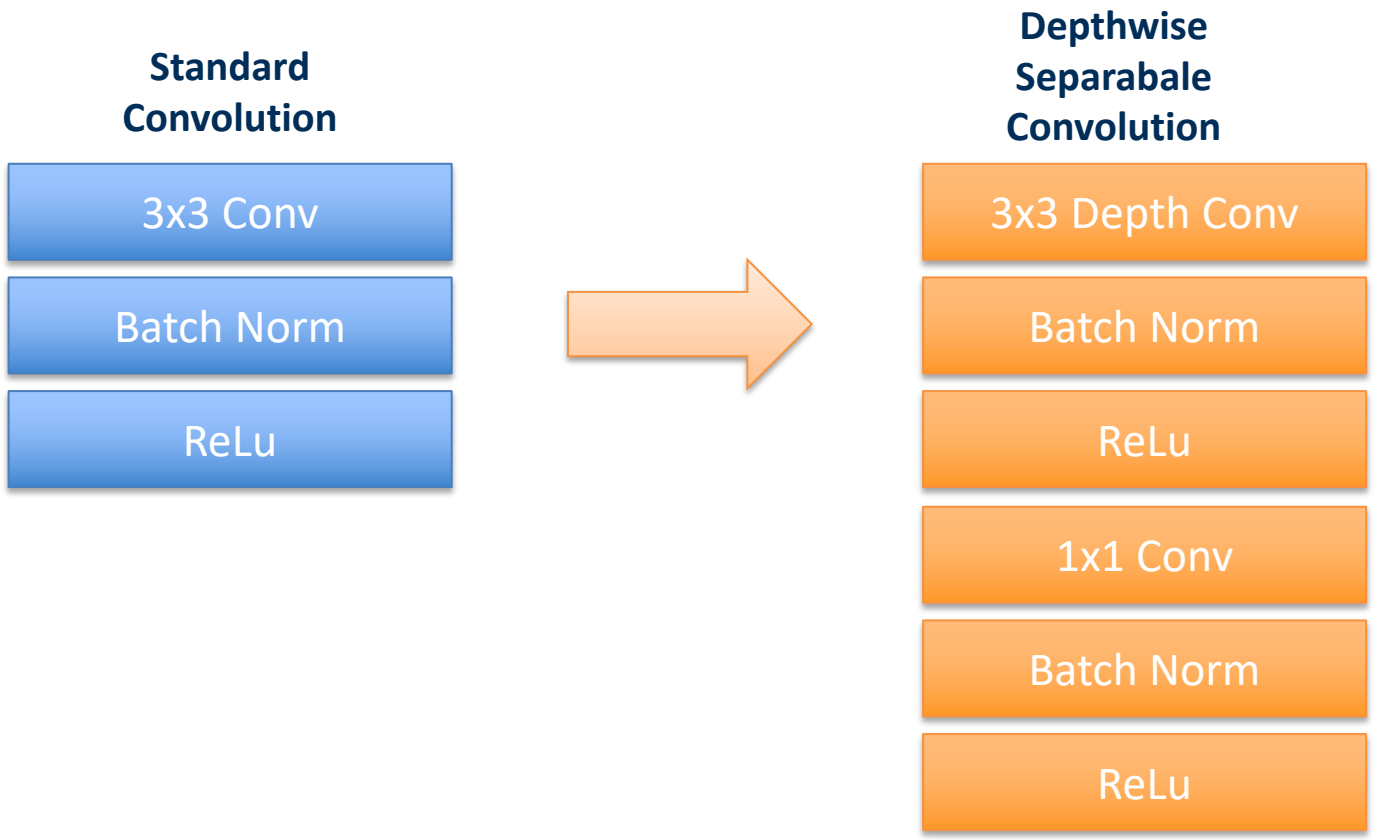
when M = 1024
and R = S = 3

Weights ratio :

$$\frac{R \times S \times C + C \times M}{R \times S \times C \times M} = \frac{1}{M} + \frac{1}{R \times S} = 0.112$$

when M = 1024
and R = S = 3

MobileNet: designing the network



MobileNet: designing the network

Table 1. MobileNet Body Architecture

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	$1 \times 1 \times 1000$

Depthwise Separabale Convolution

- 3x3 Depth Conv
- Batch Norm
- ReLu
- 1x1 Conv
- Batch Norm
- ReLu

Image taken from [2]

MobileNet: designing the network

Table 1. MobileNet Body Architecture

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	$1 \times 1 \times 1000$

- An initial full conv layer
- A sequence of depthwise separable conv (DW + 1x1)
- Fully connect + Softmax
- 28 layers
- less regularization and data augmentation techniques are considered because small models have less trouble with overfitting

Table 2. Resource Per Layer Type

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%

Image taken from [2]

MobileNet: a comparison on effictiveness

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

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

Image taken from [2]

From V1 to V2: Inverted Residuals and Linear Bottlenecks

MobileNetV2 Architecture

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

- Key module: the inverted residual with linear bottleneck

The inverted residual with linear bottleneck module

"the manifold of interest should lie in a low-dimensional subspace of the higher-dimensional activation space…"[3]

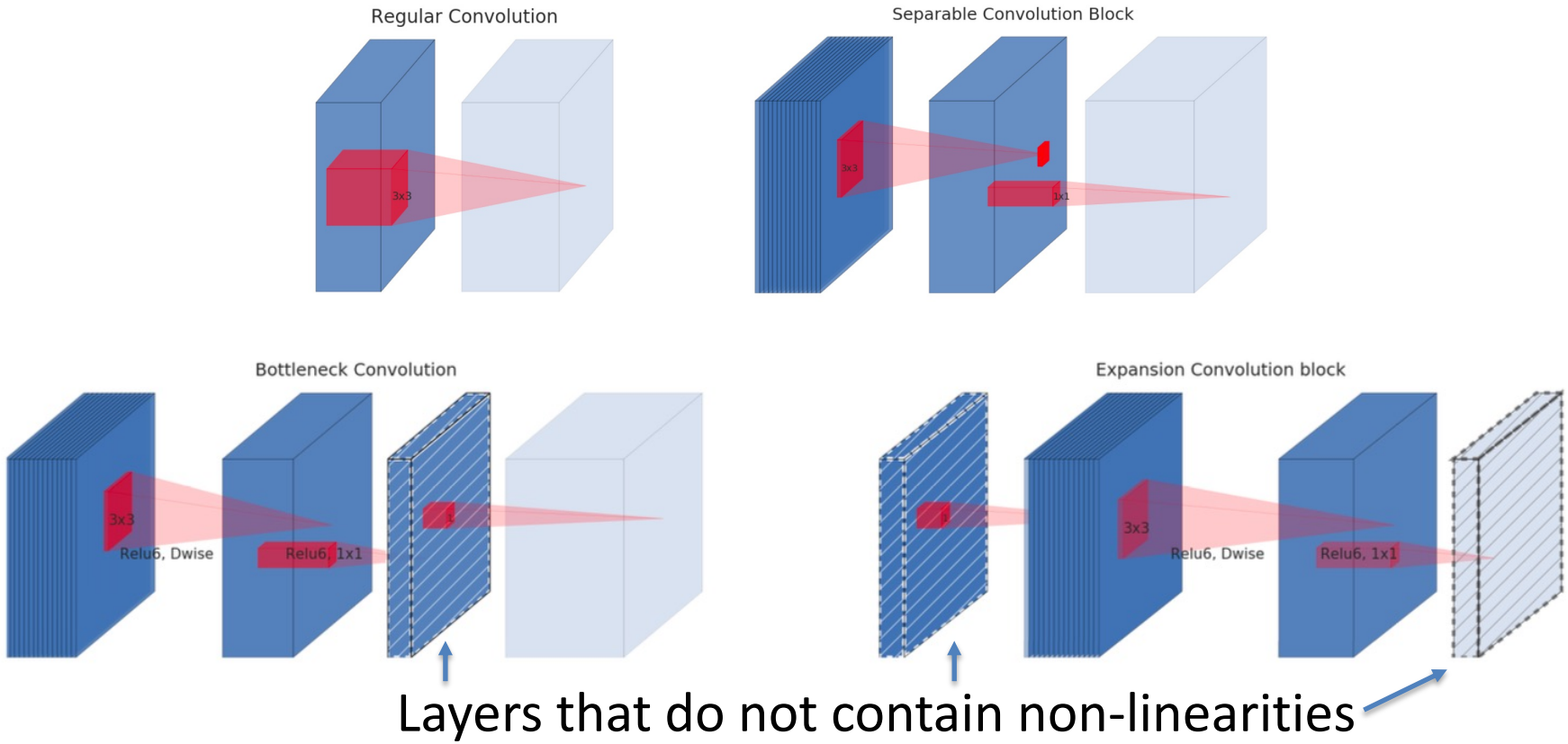
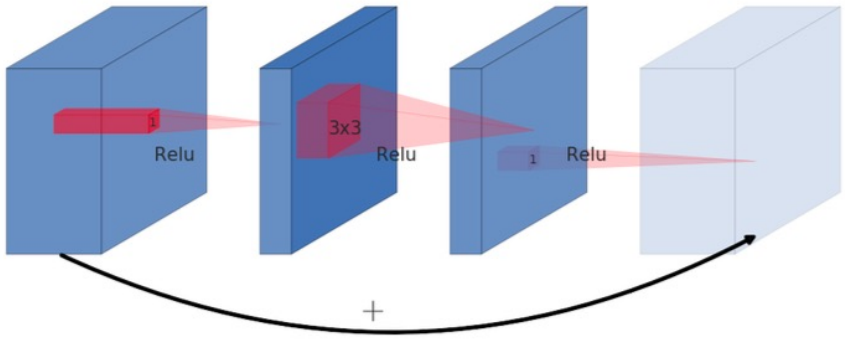
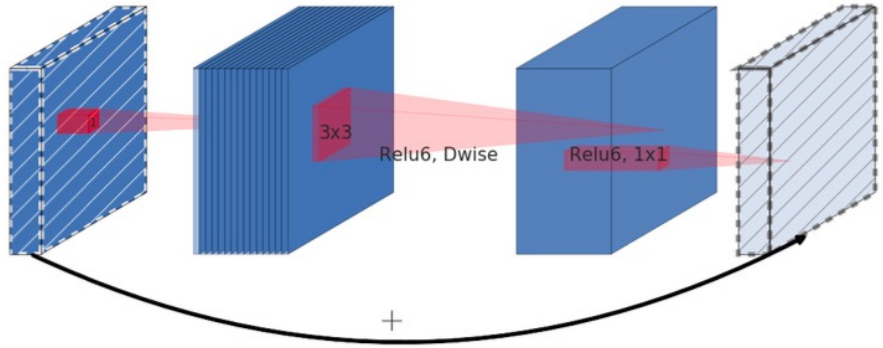


Image taken from [3]

The inverted residual block



Residual block



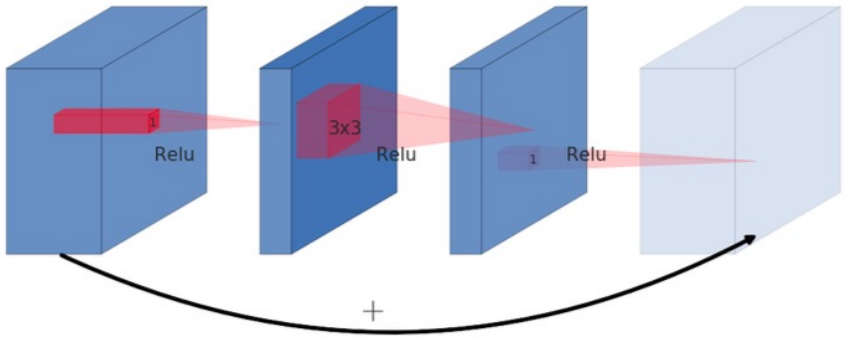
Inverted residual block

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

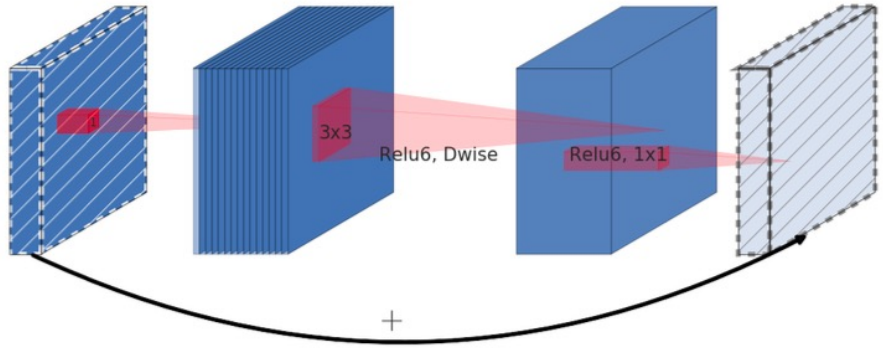
where t is the expansion factor and s is the stride

Image taken from [3]

The inverted residual block



Residual block



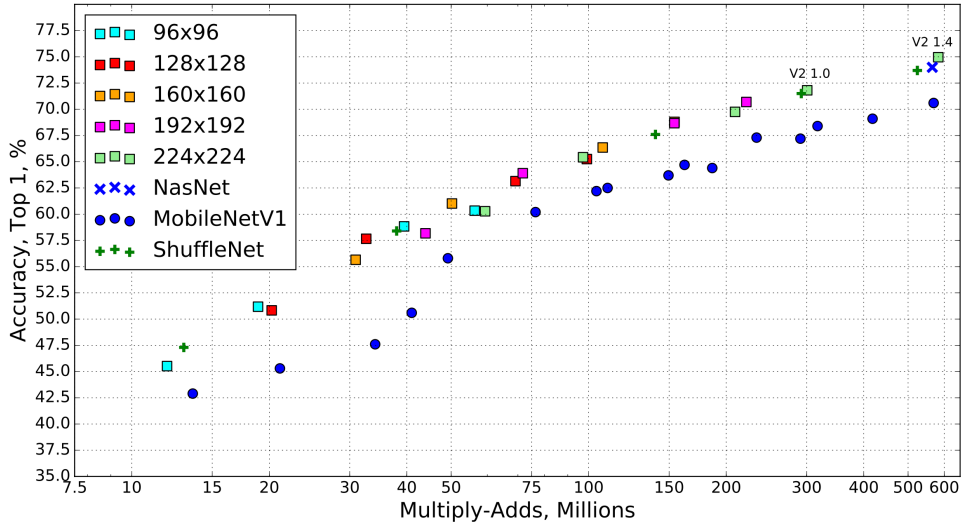
Inverted residual block

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

where t is the expansion factor and s is the stride

Image taken from [3]

MobileNetV2: accuracy vs complexity



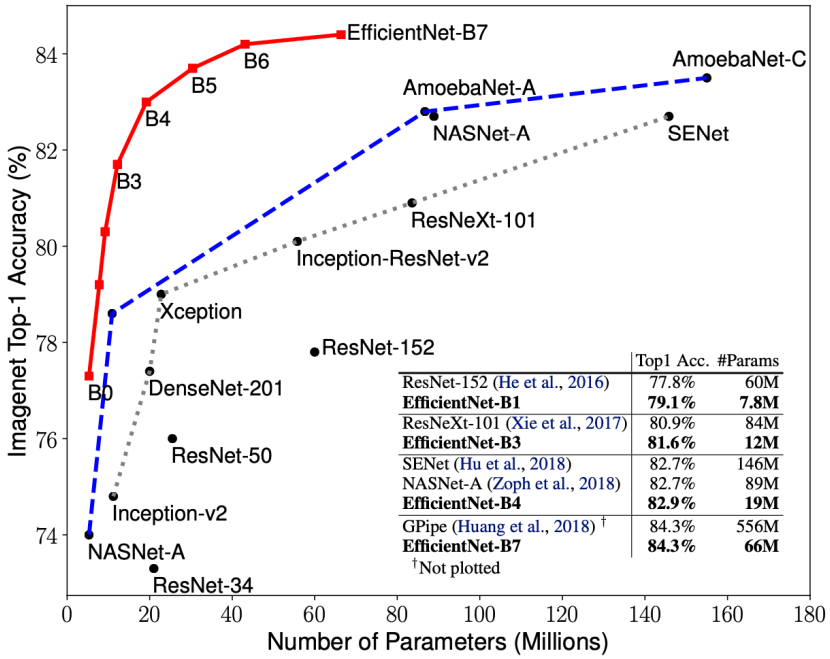
Performance curve of MobileNetV2.
Multipliers: 0.35, 0.5, 0.75, 1.0 and 1.4 (only for 224x224)

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**

Top 1 Accuracy on ImageNet and performance

Image taken from [3]

Three families of architectures for tiny devices



	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**

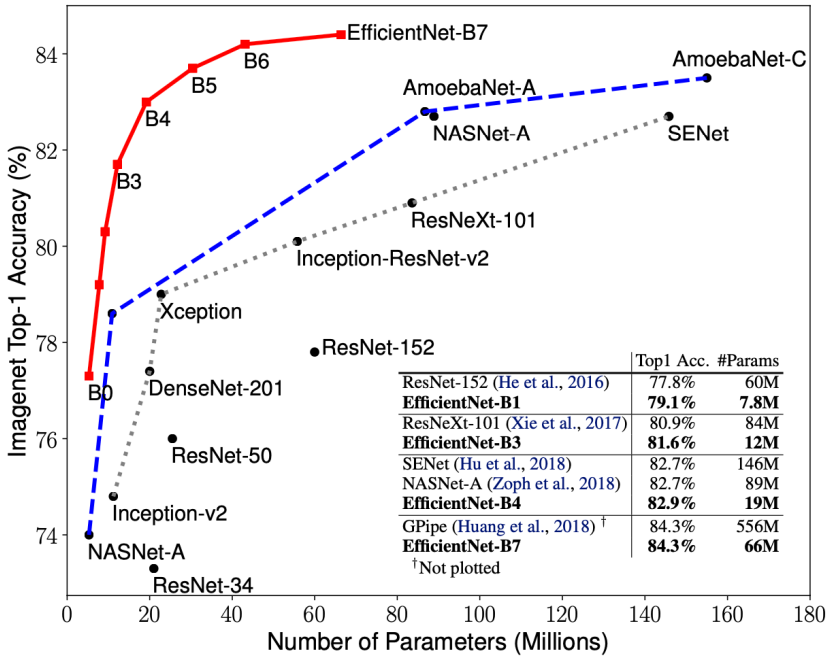
†Not plotted

Rethinking Model Scaling for Convolutional Neural Networks

EfficientNet (2019)

Image taken from [4]

Three families of architectures for tiny devices



	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

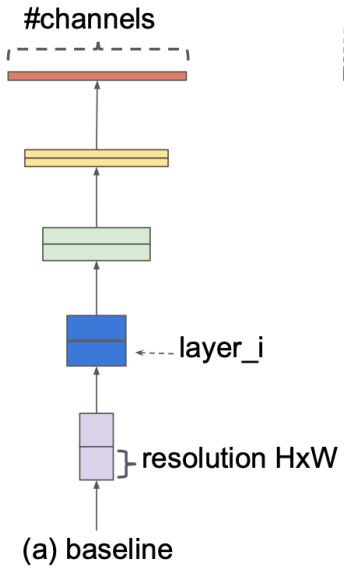
EfficientNet (2019)

Rethinking Model Scaling for Convolutional Neural Networks

Efficient architectures for CNNs
Efficient scaling mechanisms for CNNs

Image taken from [4]

EfficientNet: rethinking how to scale up CNN…

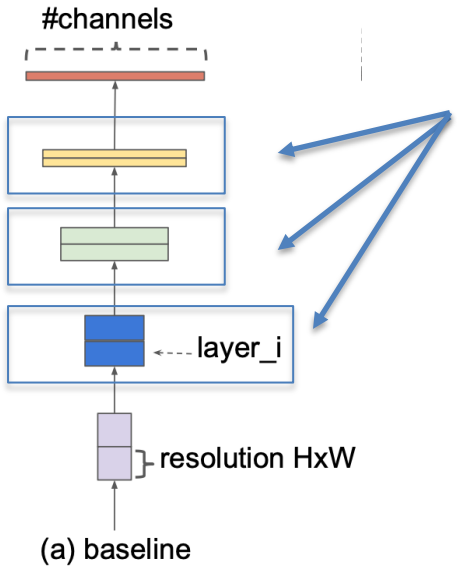


How to scale the CNN?

- Increase width
- Increase depth
- Increase resolution

Image taken from [4]

EfficientNet: rethinking how to scale up CNN…

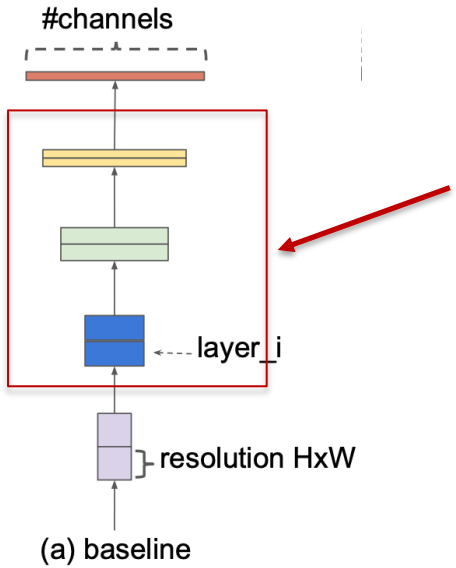


How to scale the CNN?

- Increase width
- Increase depth
- Increase resolution

Image taken from [4]

EfficientNet: rethinking how to scale up CNN…

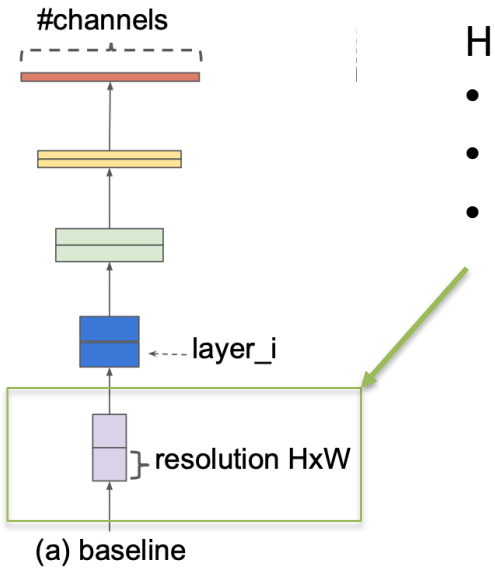


How to scale the CNN?

- Increase width
- Increase depth
- Increase resolution

Image taken from [4]

EfficientNet: rethinking how to scale up CNN…

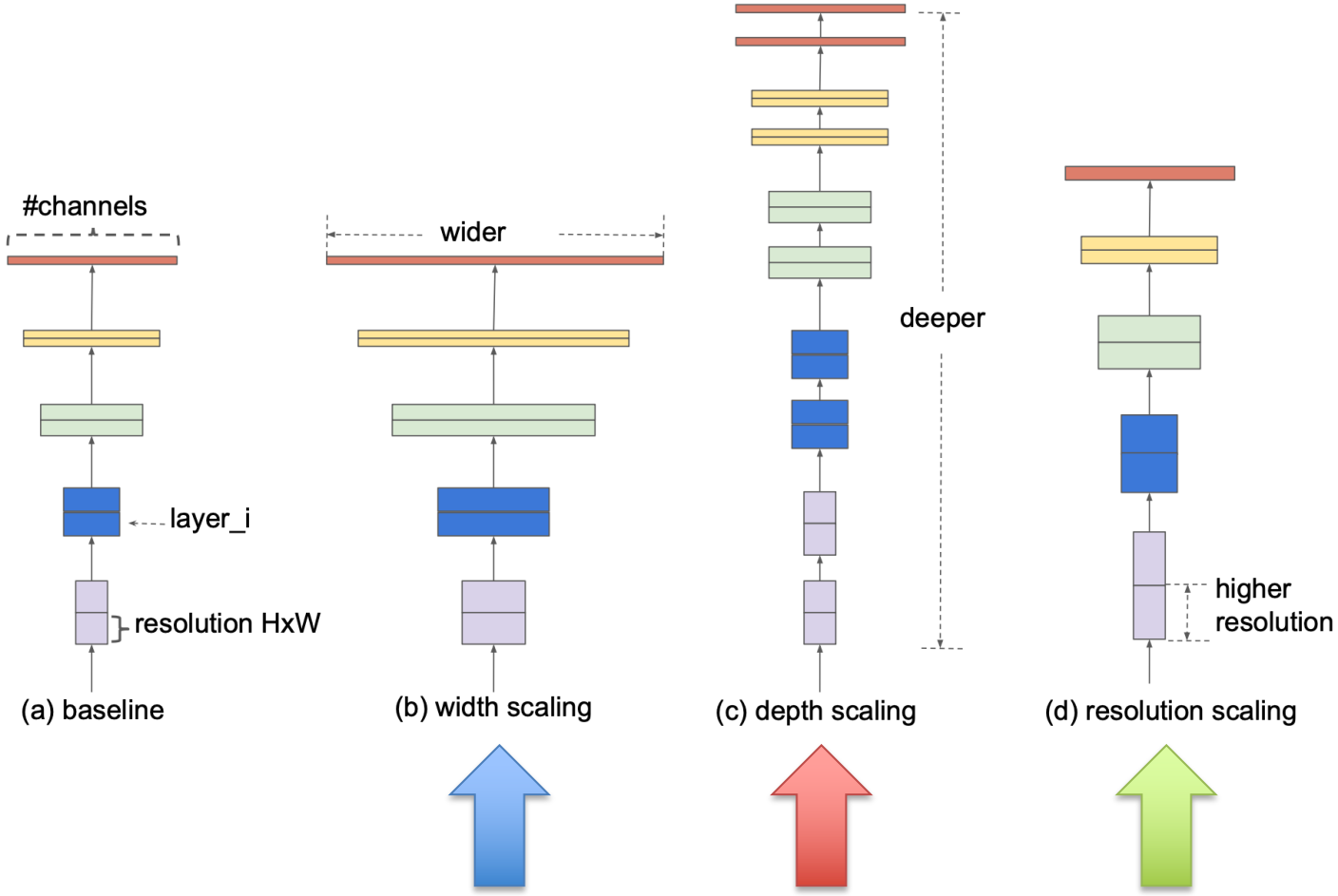


How to scale the CNN?

- Increase width
- Increase depth
- Increase resolution

Image taken from [4]

EfficientNet: rethinking how to scale up CNN…



What about scaling each of them with constant ratio?

Image taken from [4]

EfficientNet: rethinking how to scale up CNN…

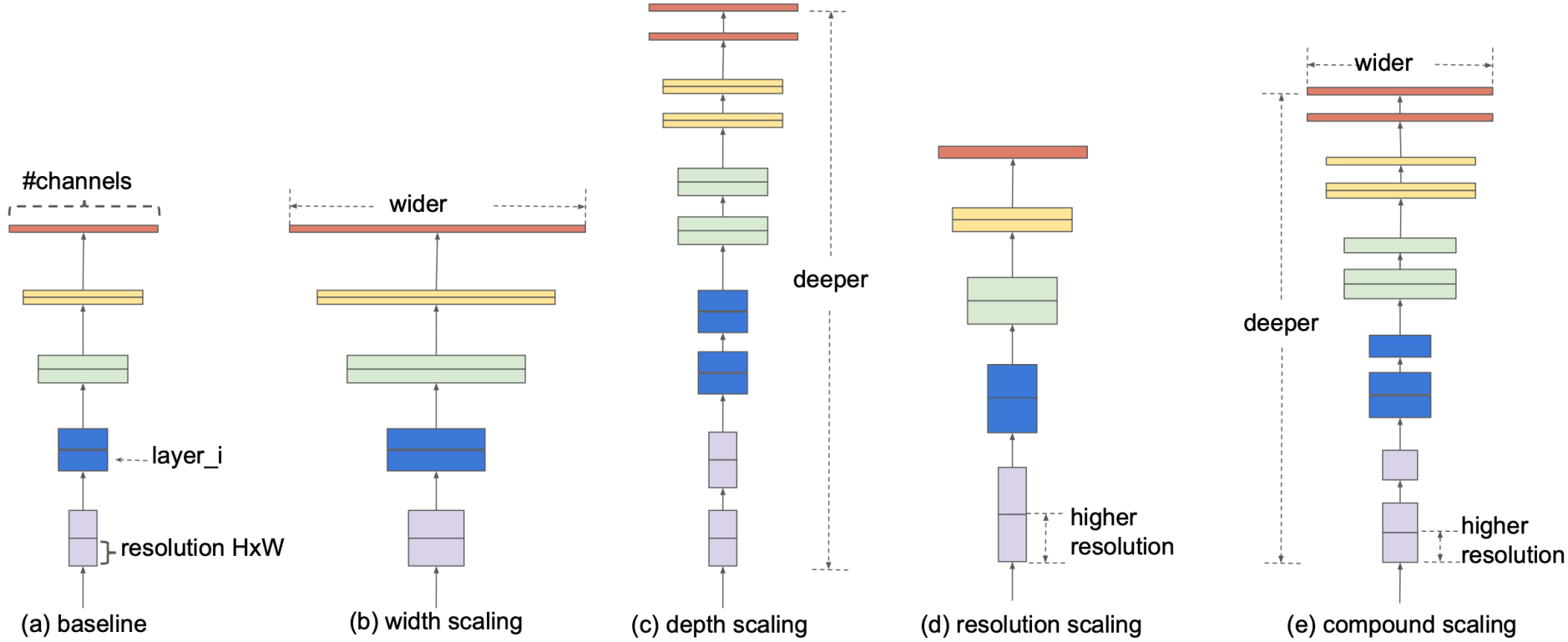


Image taken from [4]

The compound scaling method (the idea)

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

- α, β, γ are constants that can be determined by a small grid search.
- ϕ is a user-specified coefficient controlling how many more resources are available for model scaling

The compound scaling method (the method)

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

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

EfficientNet-B0

- STEP 1: $\phi = 1$, setting a NN and assuming twice more resources available, a small grid search of α, β, γ is carried out, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
  (see EfficientNet-B0)
- STEP 2: we then fix α, β, γ as constants and scale up baseline network with different values of ϕ (see EfficientNet-B1 to B7)

EfficientNet: rethinking how to scale up CNN…

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	-	-

Image taken from [4]