Deep Learning

Convolutional Neural Network

Dr. Mehran Safayani

safayani@iut.ac.ir

safayani.iut.ac.ir

https://www.aparat.com/mehran.safayani

https://github.com/safayani/deep_learning_course

Department of Electrical and computer engineering, Isfahan university of technology, Isfahan, Iran

Computer Vision Problems

Image Classification

64x64

Cat? (0/1)

Object detection

Neural Style Transfer

Content

Style

Generated image

Deep Learning on large images

Cat? (0/1)

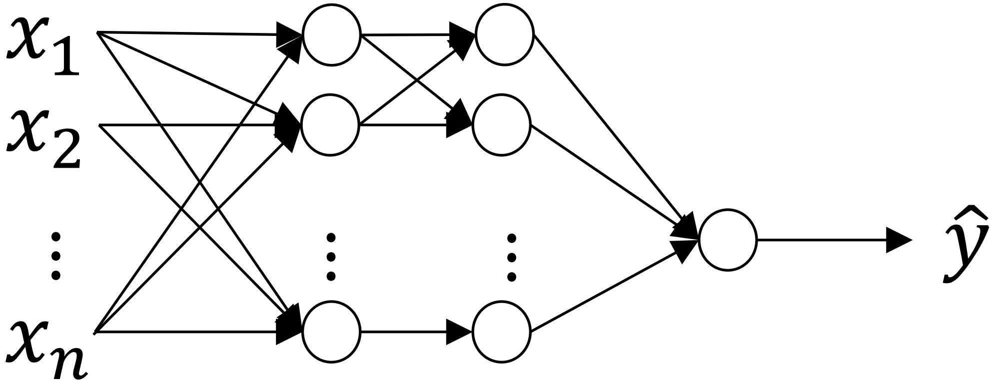
64x64x3=12288

1000x1000x3=3m

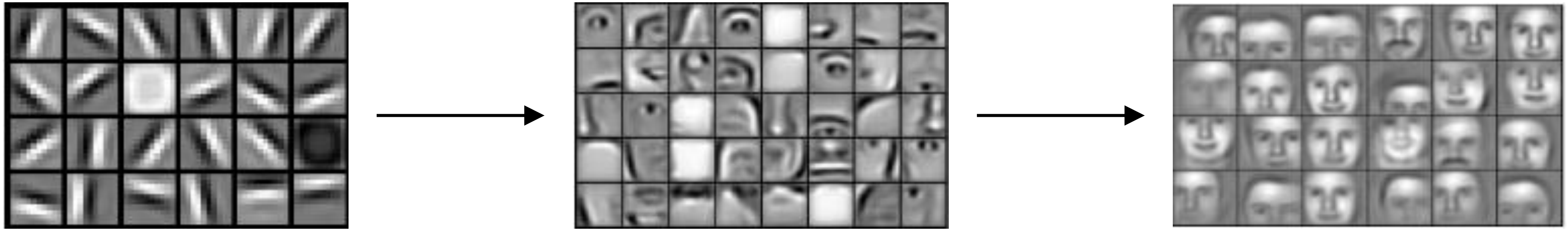
n=3m

$n^1 = 1000$

$W^1_{(1000,3m)}$



Edge detection



vertical edges

horizontal edges

Vertical edge detection

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

1	0	-1
1	0	-1
1	0	-1

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

Vertical edge detection

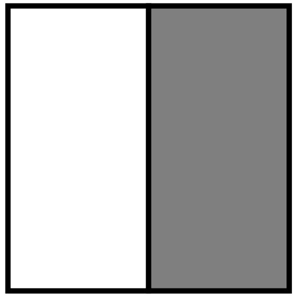
1 10	0 10	-1 10	0	0	0
1 10	0 10	-1 10	0	0	0
1 10	0 10	-1 10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

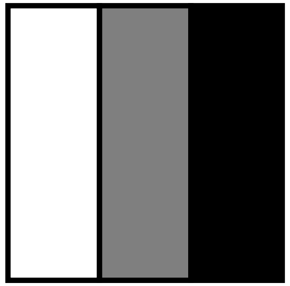
1	0	-1
1	0	-1
1	0	-1

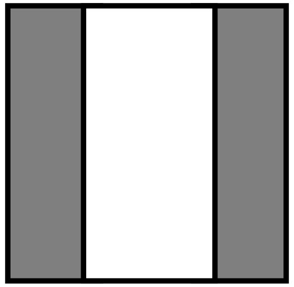
=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



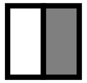
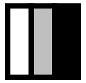
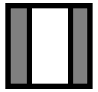
*



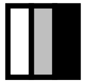


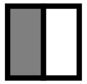
Vertical edge detection examples

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$





Vertical and Horizontal Edge Detection

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$*$

1	1	1
0	0	0
-1	-1	-1

$=$

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Learning to detect edges

1	0	-1
1	0	-1
1	0	-1

1	0	-1
2	0	-2
1	0	-1

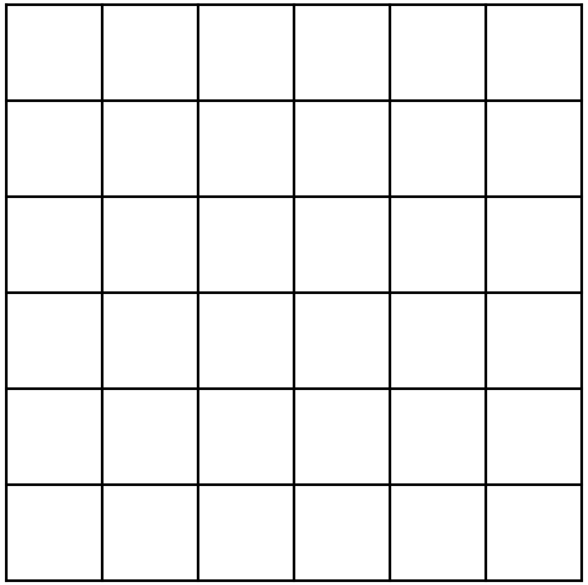
Sobel filter

3	0	-1
10	0	-10
3	0	-3

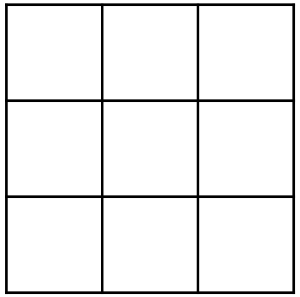
Schass filter

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

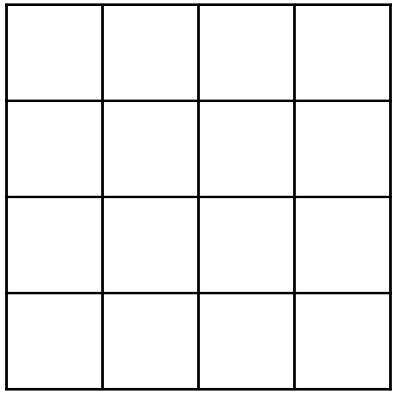
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9



$*$



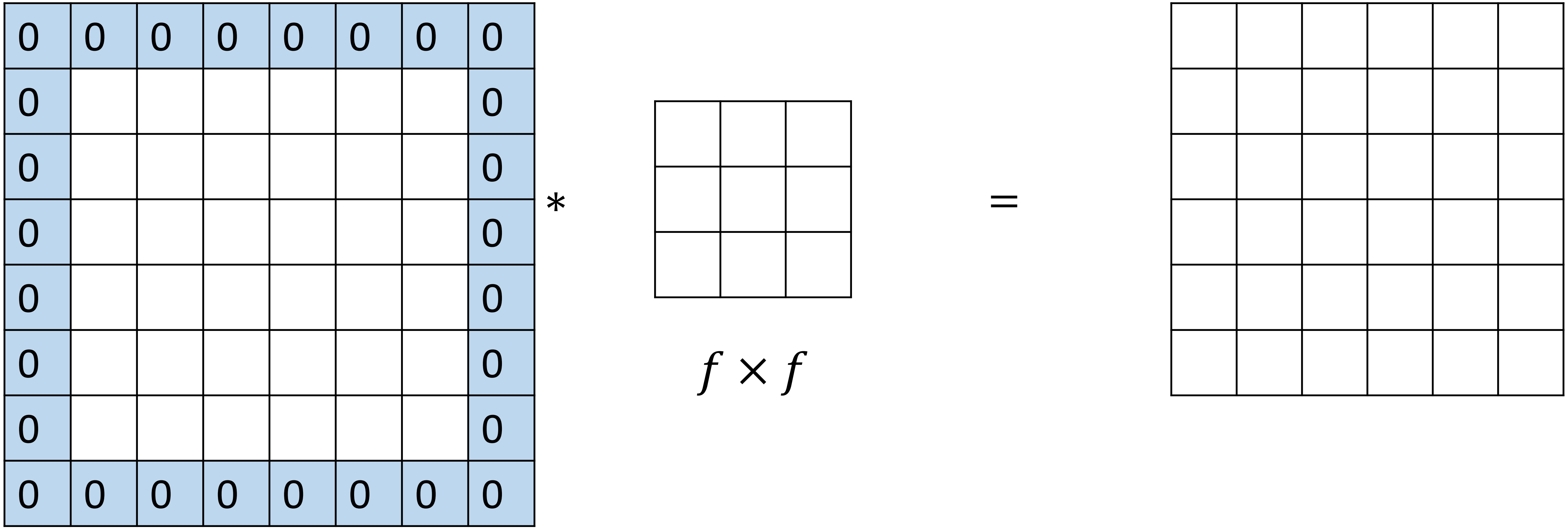
$=$



$n \times n$ $\quad$ $f \times f$ $\quad$ $n - f + 1 \times n - f + 1$

6×6 $\quad$ 3×3 $\quad$ 4×4

Padding



$n + 2p - f + 1 \times \; n + 2p - f + 1$

$n + 2p \times \; n + 2p$

$6 + 2 \times 6{+}2$ $\quad$ 3×3 $\quad$ 6×6

Valid and Same convolutions

"Valid":

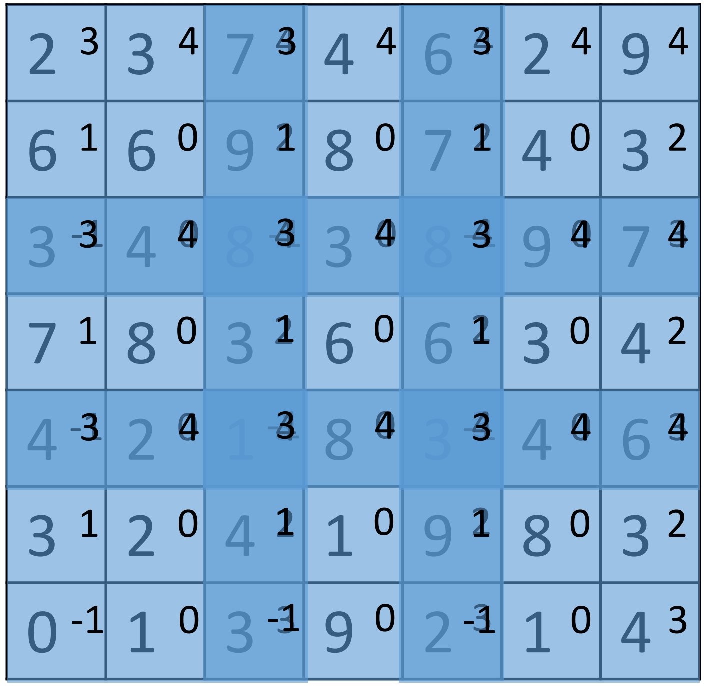
$n \times n$ $\qquad$ $f \times f$ $\qquad$ $n - f + 1 \times n - f + 1$

"Same": Pad so that output size is the same as the input size.

$$n + 2p - f + 1 = n$$

$$p = \frac{f - 1}{2}$$

Strided convolution



$*$

3	4	4
1	0	2
-1	0	3

$=$

91	100	83
69	91	127
44	72	74

$$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor \quad \times \quad \left\lfloor \frac{n-f}{s} + 1 \right\rfloor$$

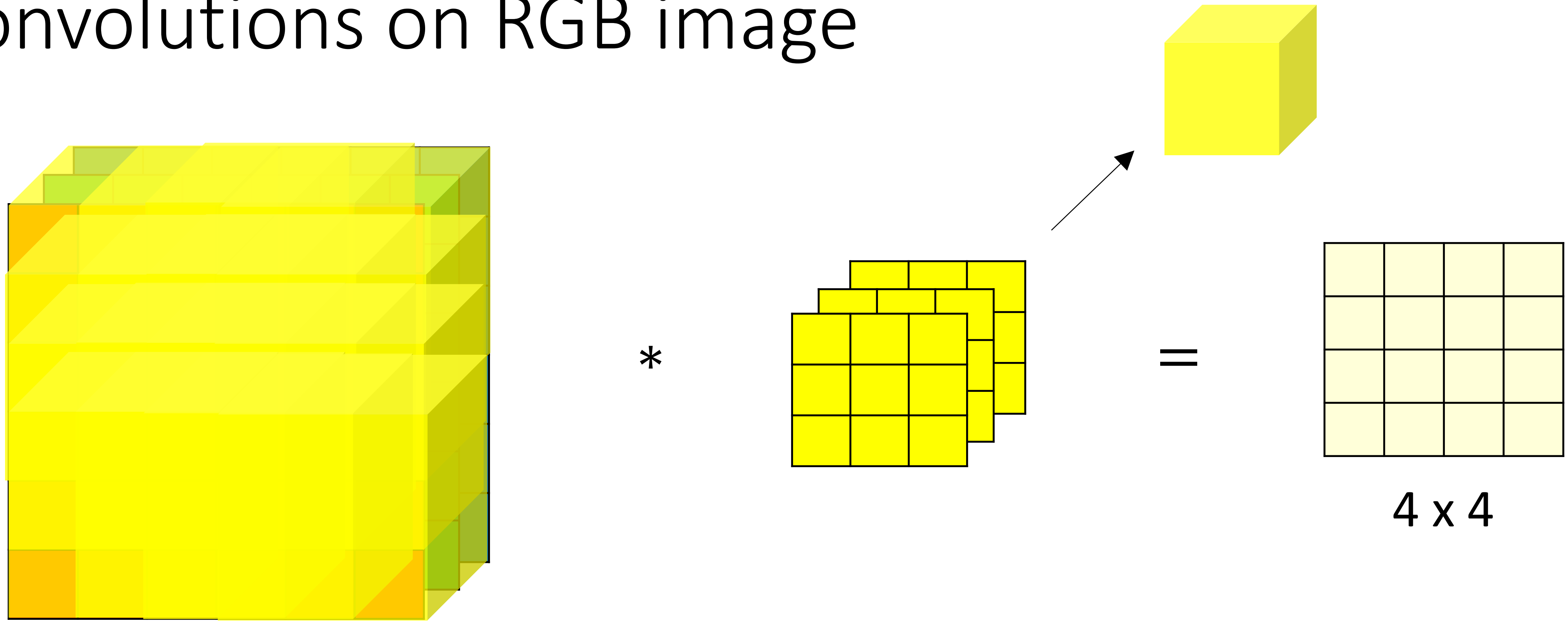
Summary of convolutions

$n \times n$ image $\quad f \times f$ filter

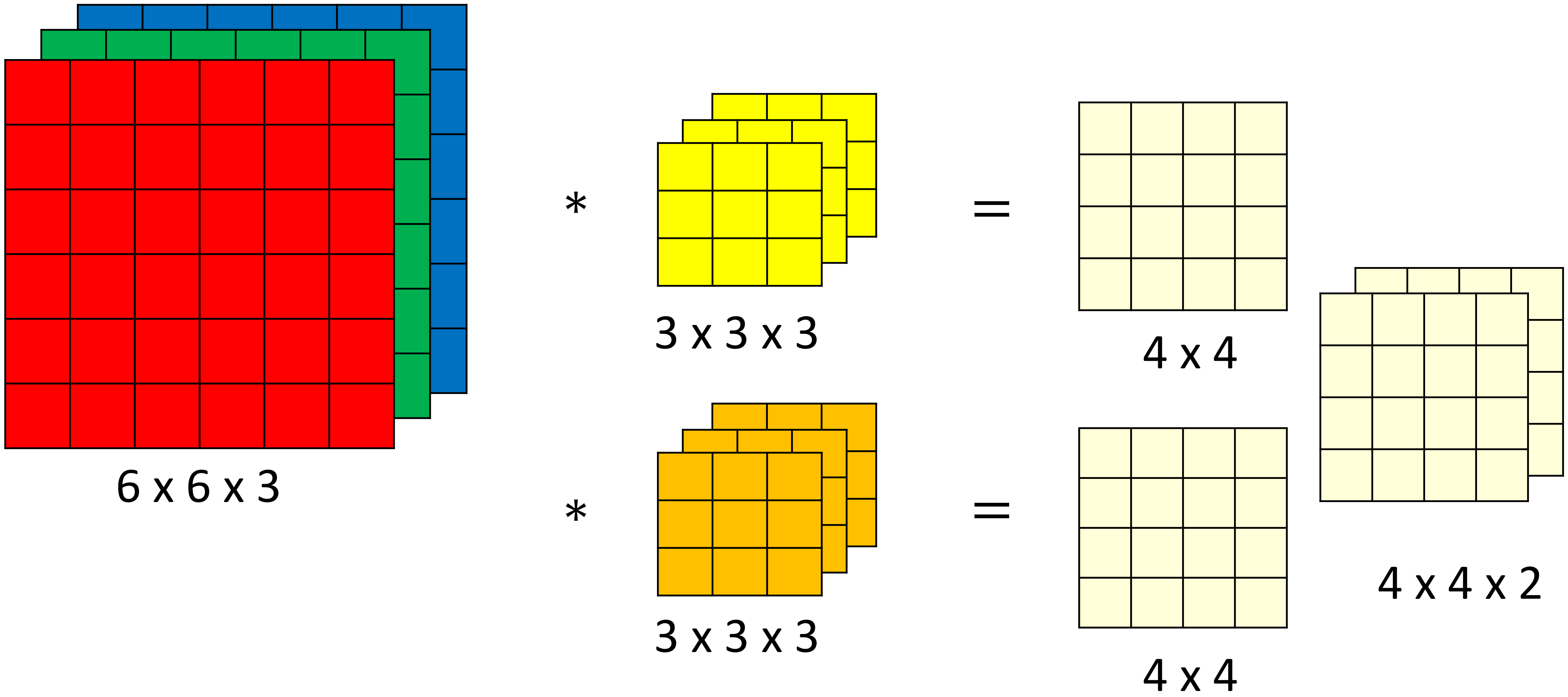
padding $p \quad$ stride s

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

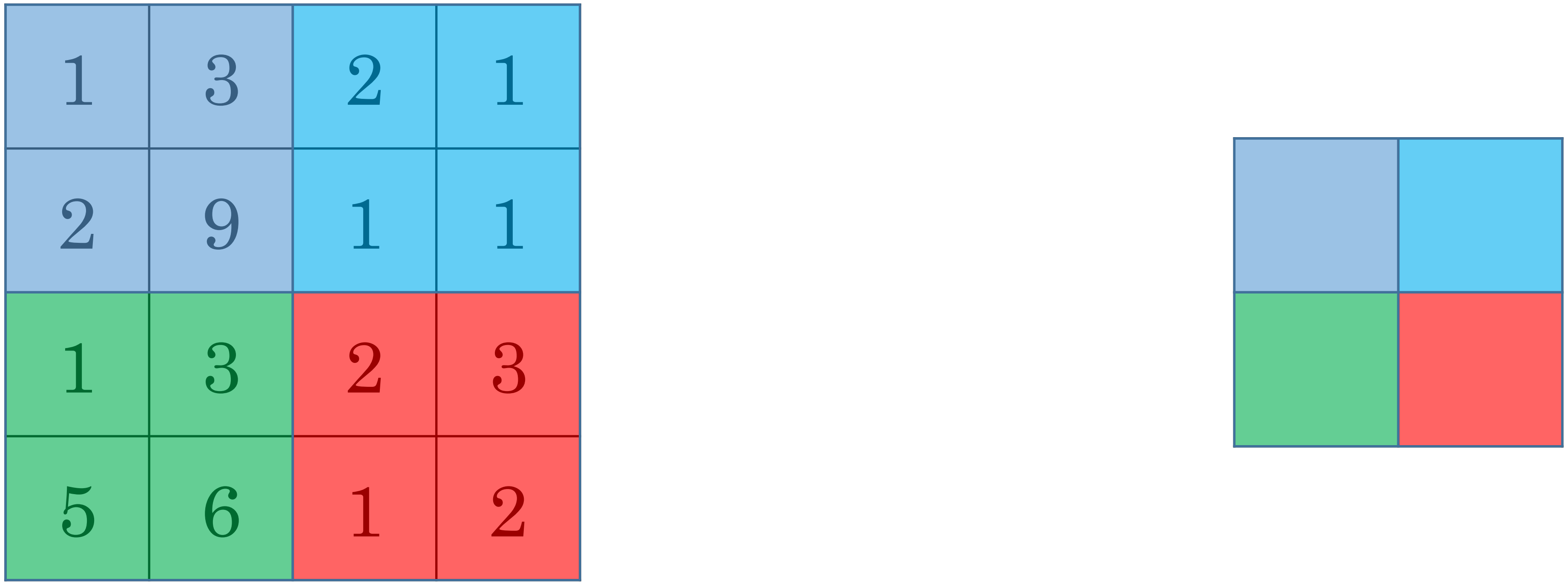
Convolutions on RGB image



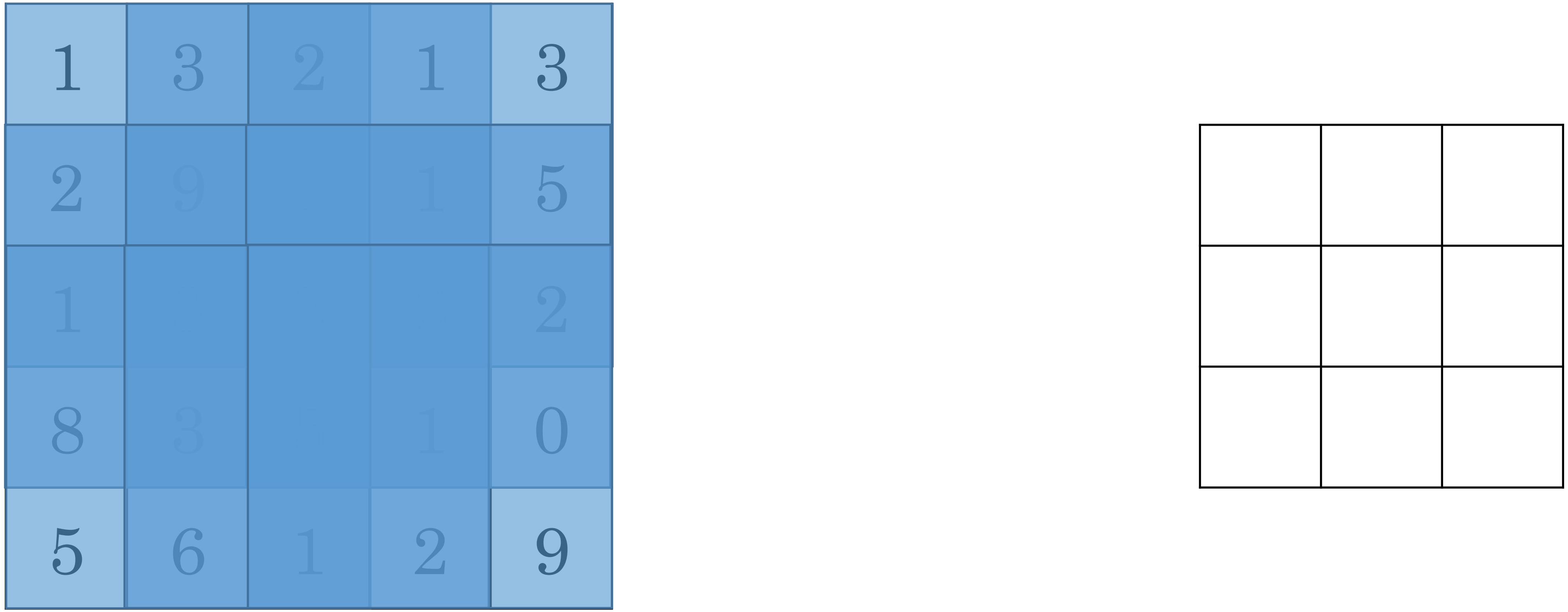
Multiple filters



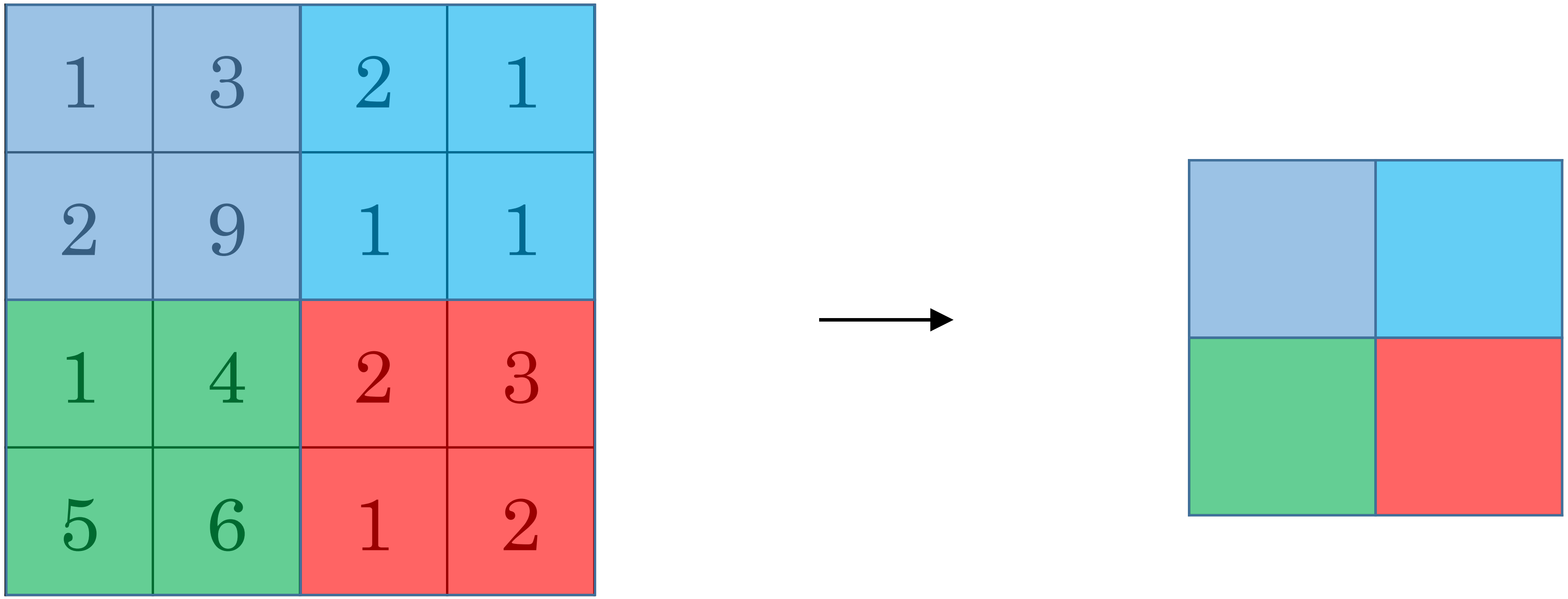
Pooling layer: Max pooling



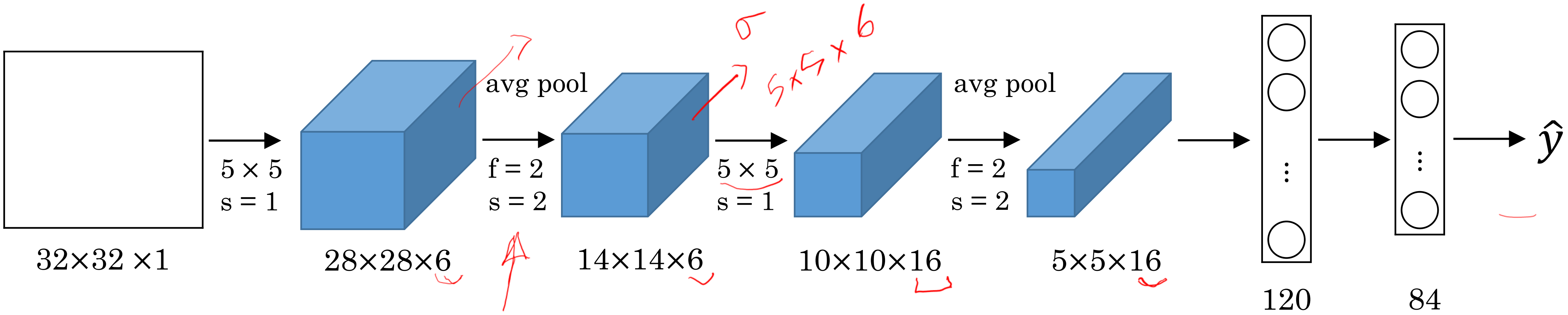
Pooling layer: Max pooling

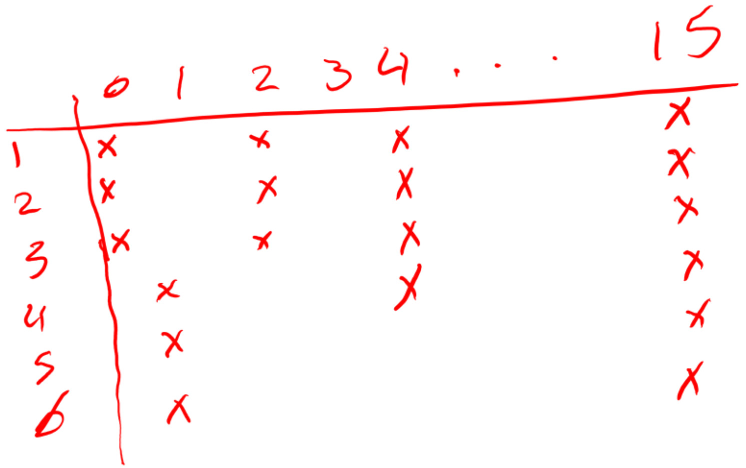


Pooling layer: Average pooling



LeNet - 5



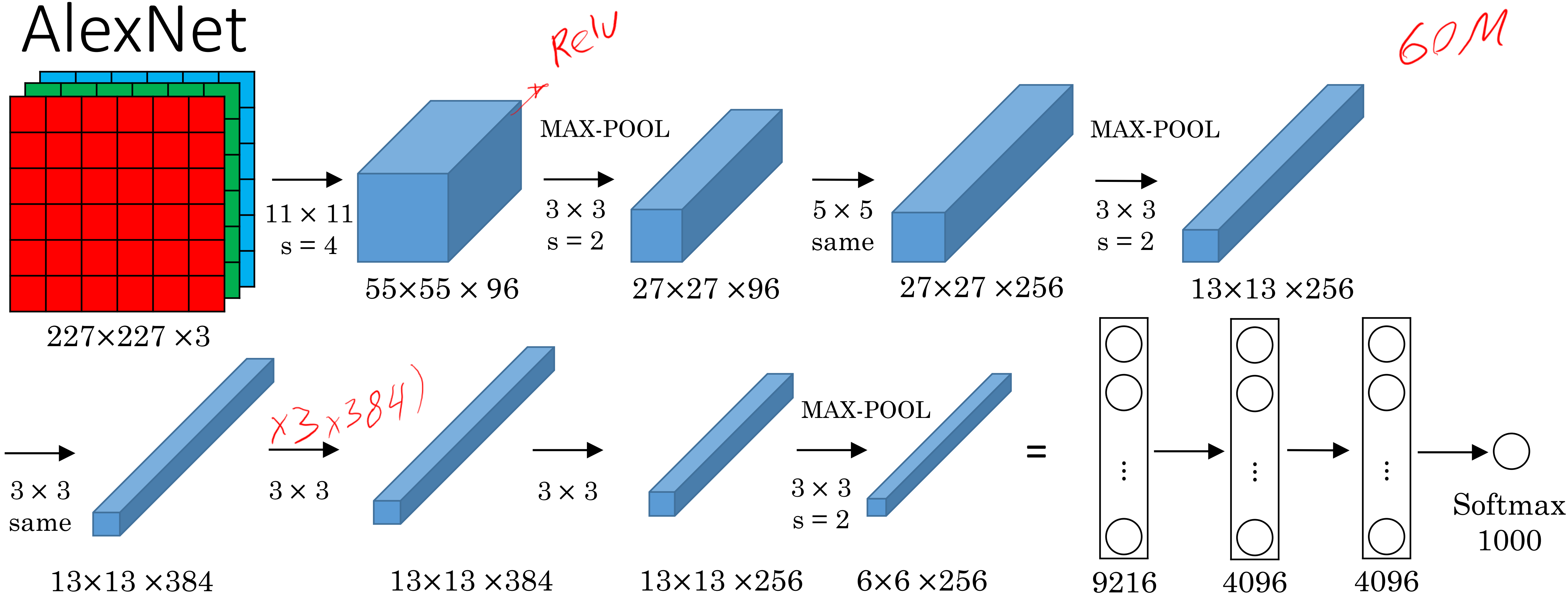


[LeCun et al., 1998. Gradient-based learning applied to document recognition]

Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
			$(5\times5\times3+1)\times8$
$f=2$ $s=2$			
			$(5\times5\times8+1)\times16$
$f=2$ $s=2$			
			$5\times5\times16\times120+120$
			$120\times84+84$
			$84\times10+10$

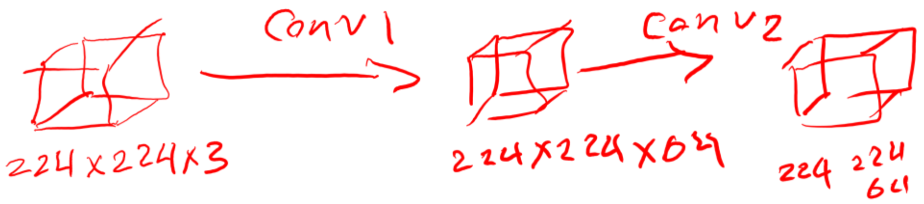
AlexNet

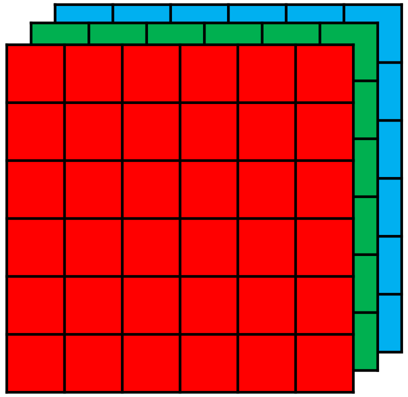


[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

VGG - 16 VGG-19

CONV = 3×3 filter, s = 1, same   MAX-POOL = 2×2 , s = 2





224×224 ×3

→ [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

138M

→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

ResNet

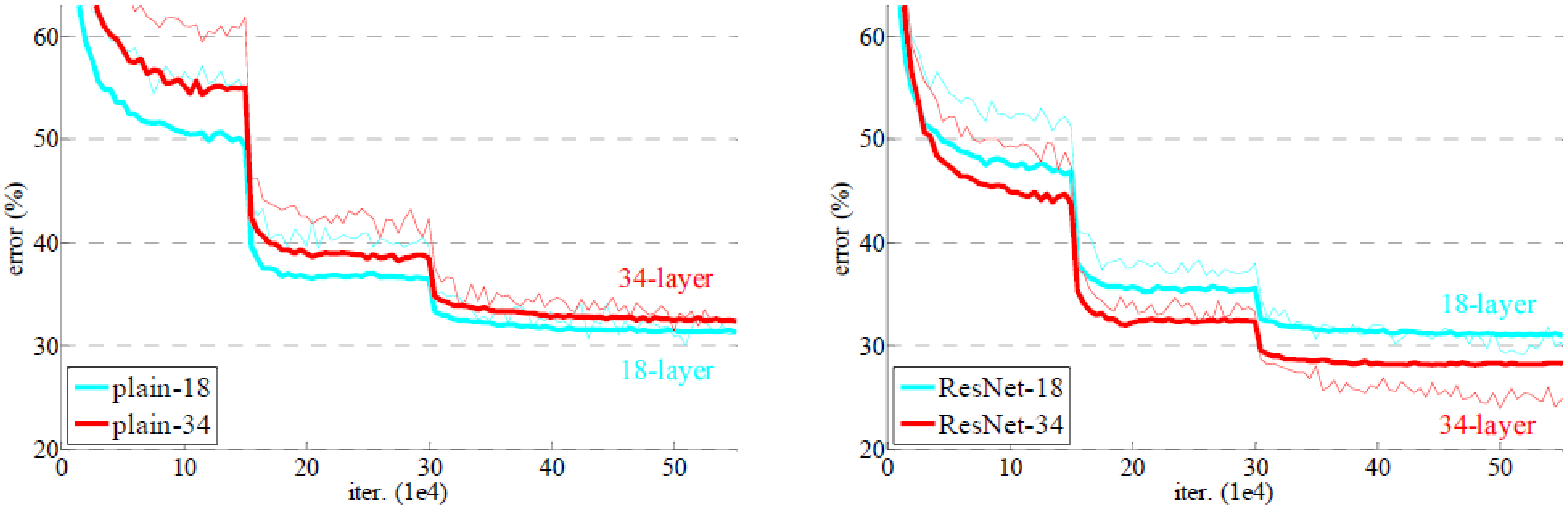
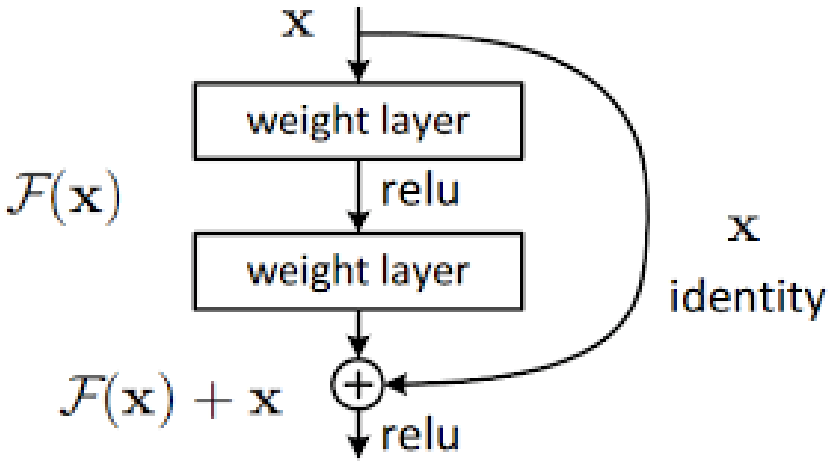


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Residual Networks(ResNets)

- Residual Block



$$z^{[1]} = w^{[1]}x + b^{[1]}$$

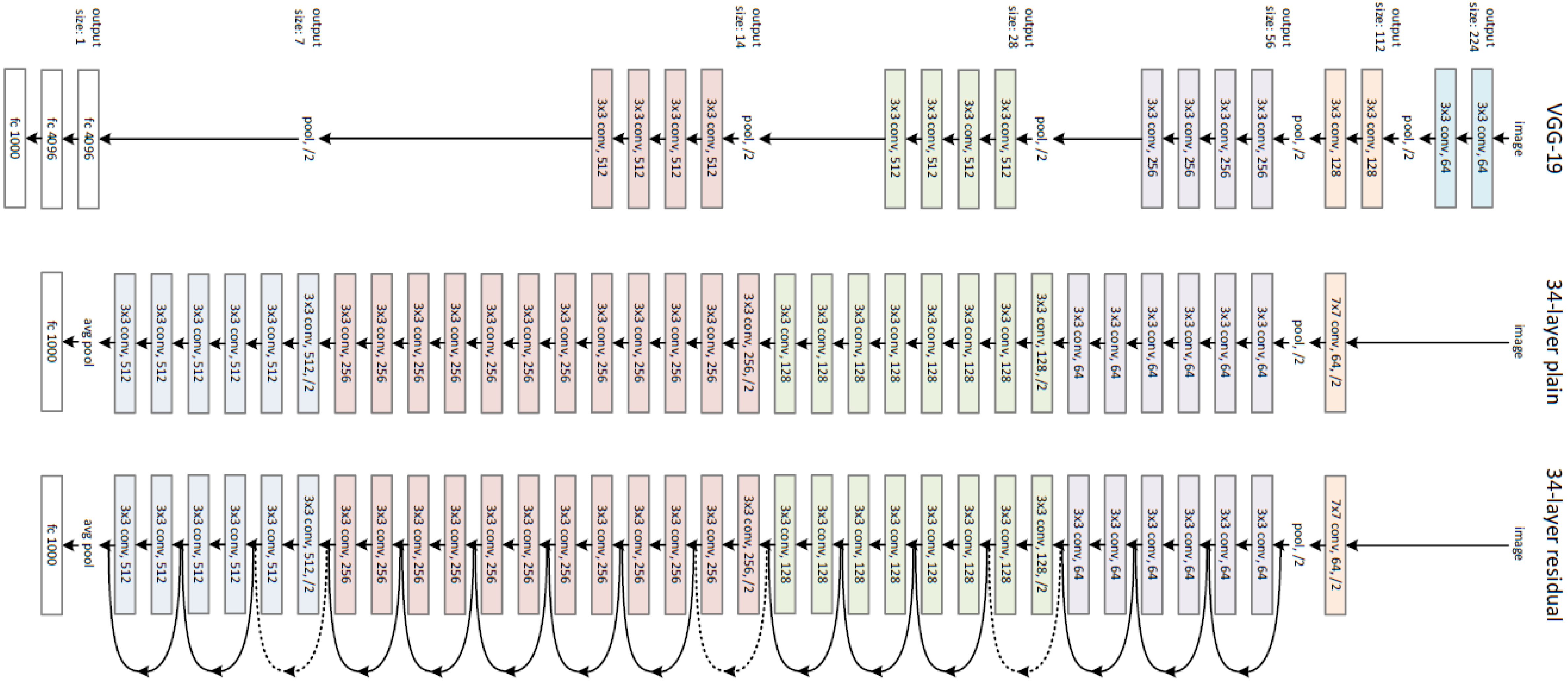
$$a^{[1]} = g(z^{[1]})$$

$$\mathrm{z}^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g(\mathrm{z}^{[2]} + x)$$

$$a^{[2]} = g(\mathrm{z}^{[2]} + w_s x)$$

ResNet



VGG-19
34-layer plain
34-layer residual
image
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
fc 4096
fc 1000
7x7 conv, 64, /2
3x3 conv, 128, /2
3x3 conv, 256, /2
3x3 conv, 512, /2
avg pool

ResNet

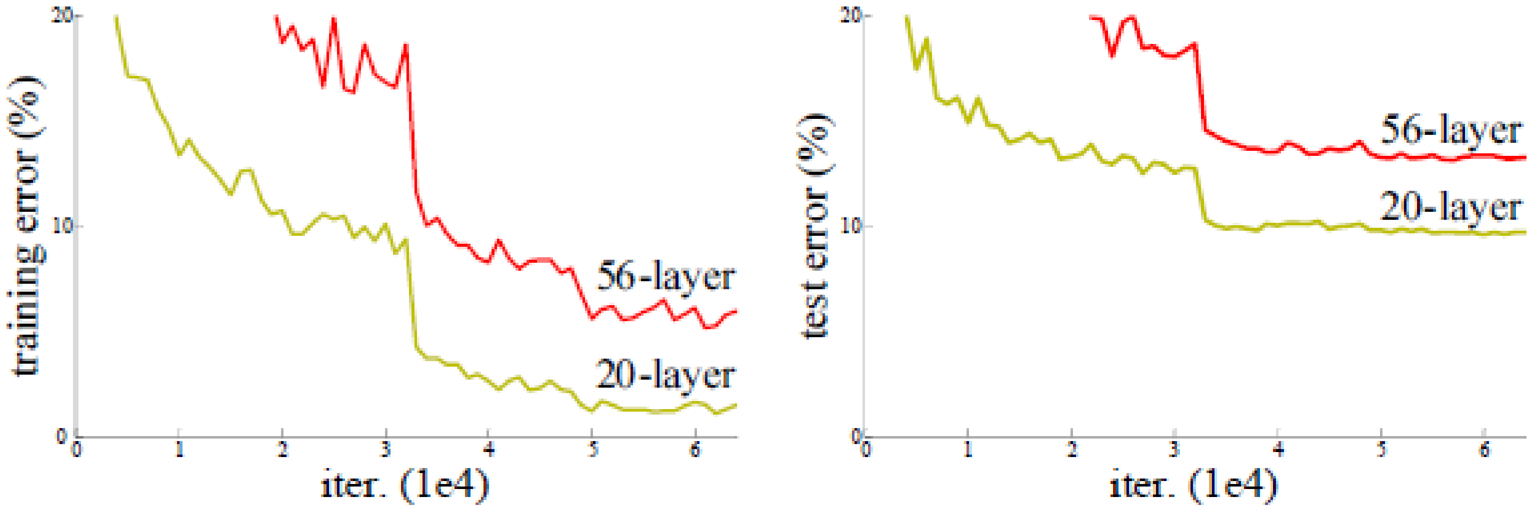


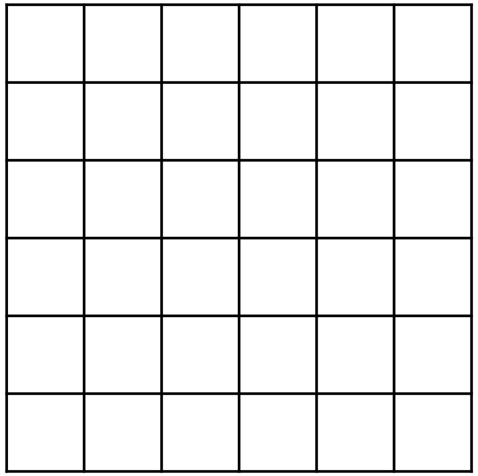
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

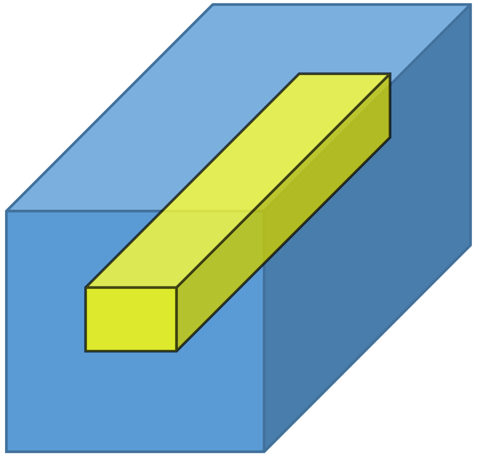
Why does a 1×1 convolution do?

1	2	3	6	5	8
3	5	5	1	3	4
2	1	3	4	9	3
4	7	8	5	7	9
1	5	3	7	4	8
5	4	9	8	3	5

6×6

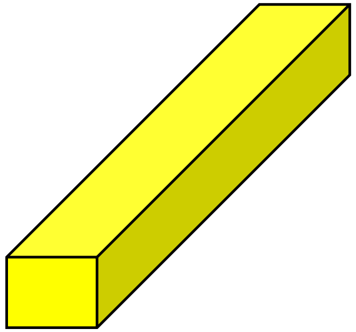
$*$ $\boxed{2}$ $=$





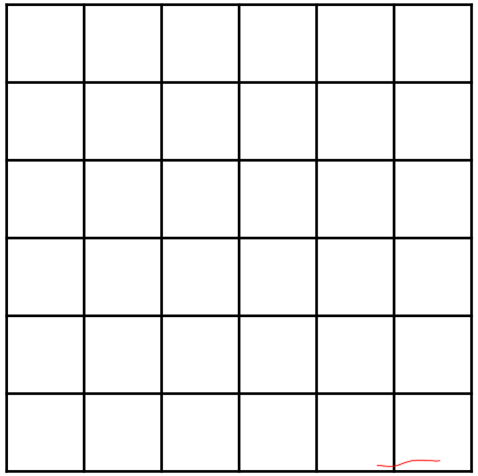
$6 \times 6 \times 192$

$*$



$1 \times 1 \times 192$

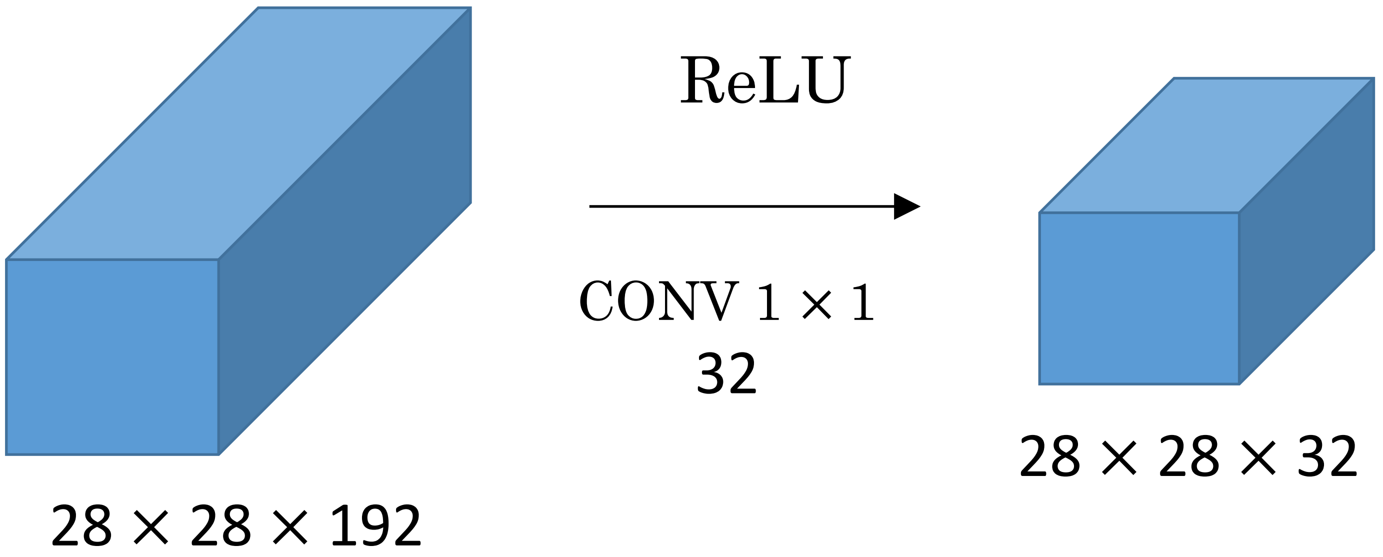
$=$



$6 \times 6 \times \#$ filters

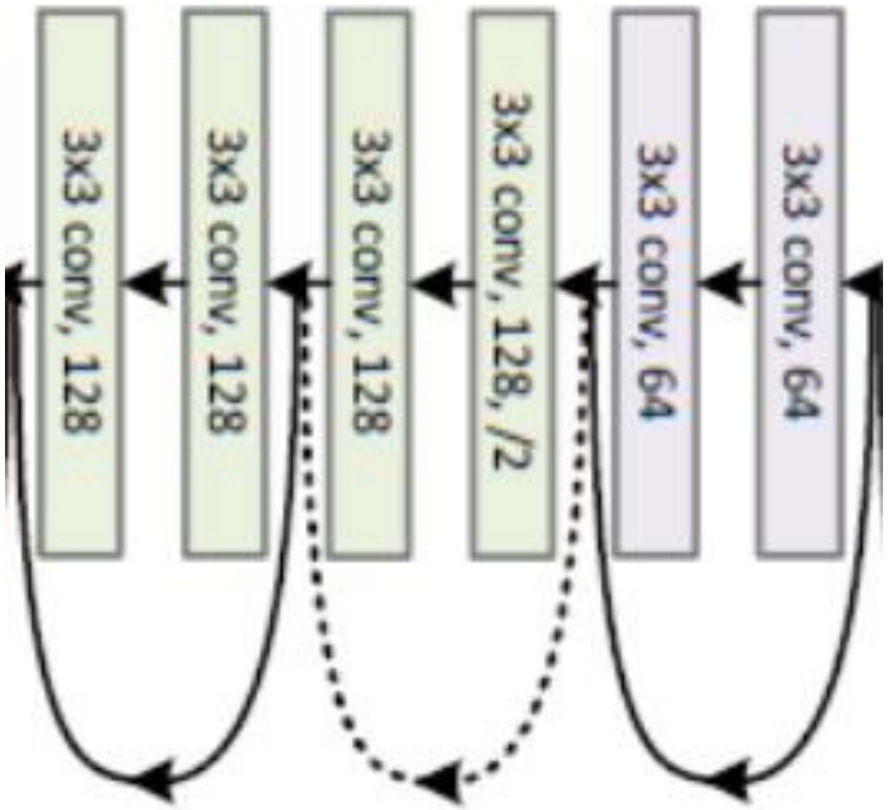
[Lin et al., 2013. Network in network]

Using 1×1 convolutions

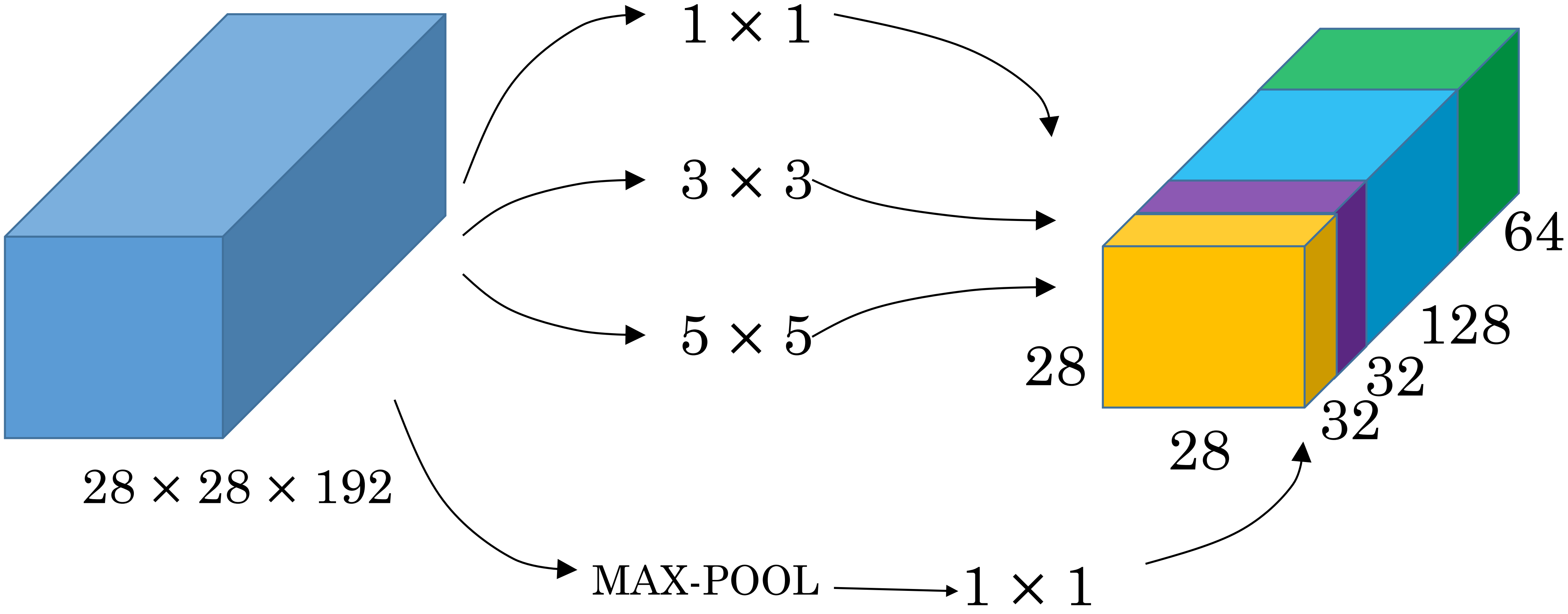


[Lin et al., 2013. Network in network]

Resnet

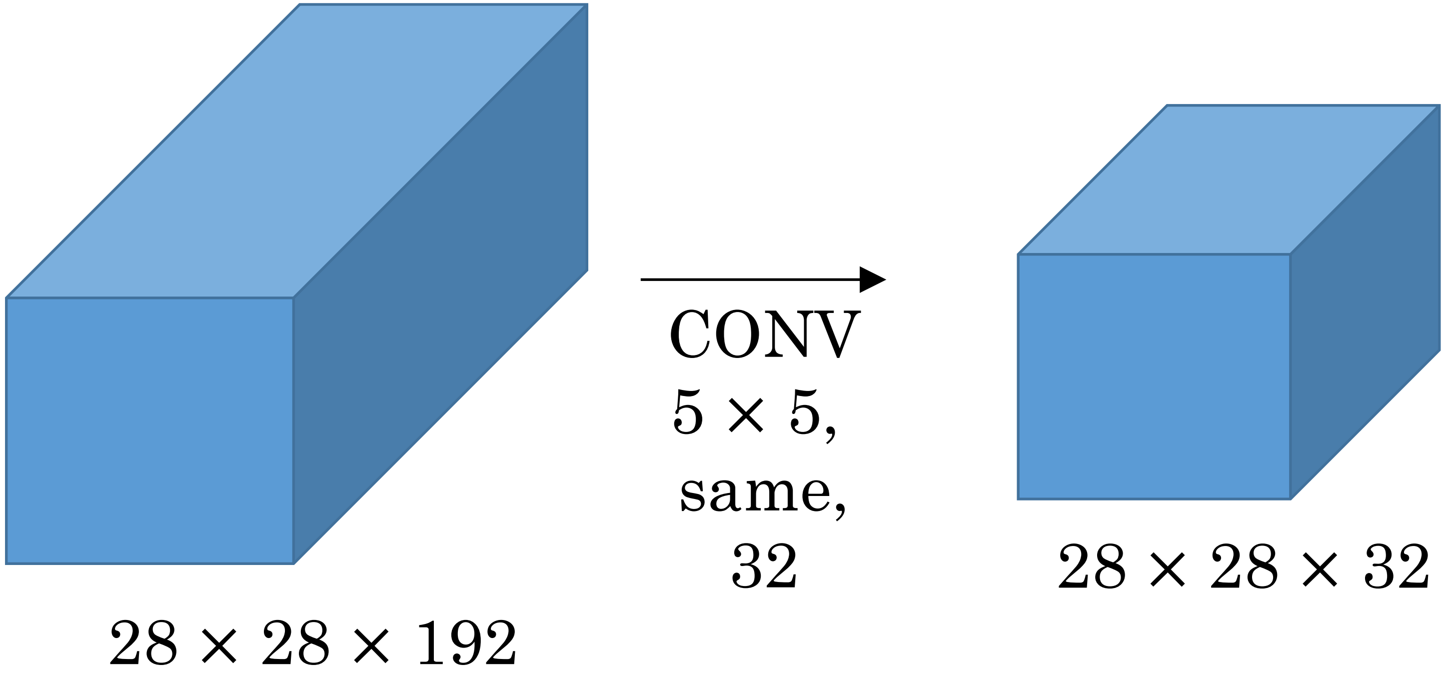

3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128, /2
3x3 conv, 64
3x3 conv, 64

Motivation for inception network



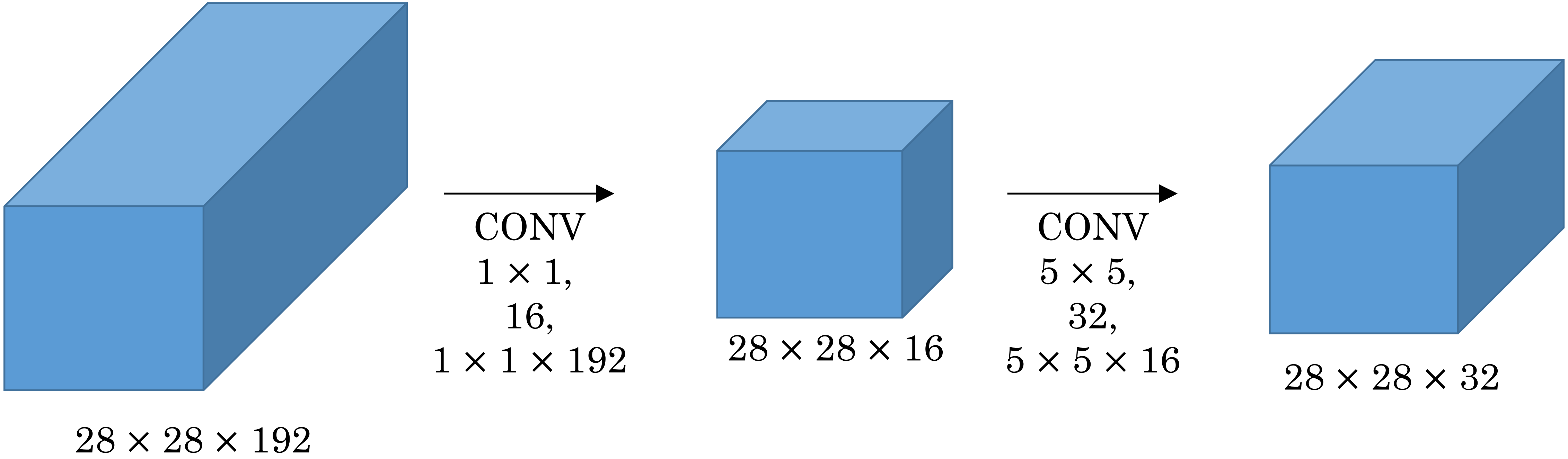
[Szegedy et al. 2014. Going deeper with convolutions]

The problem of computational cost



28x28x32x5x5x192=120m

Using 1×1 convolution

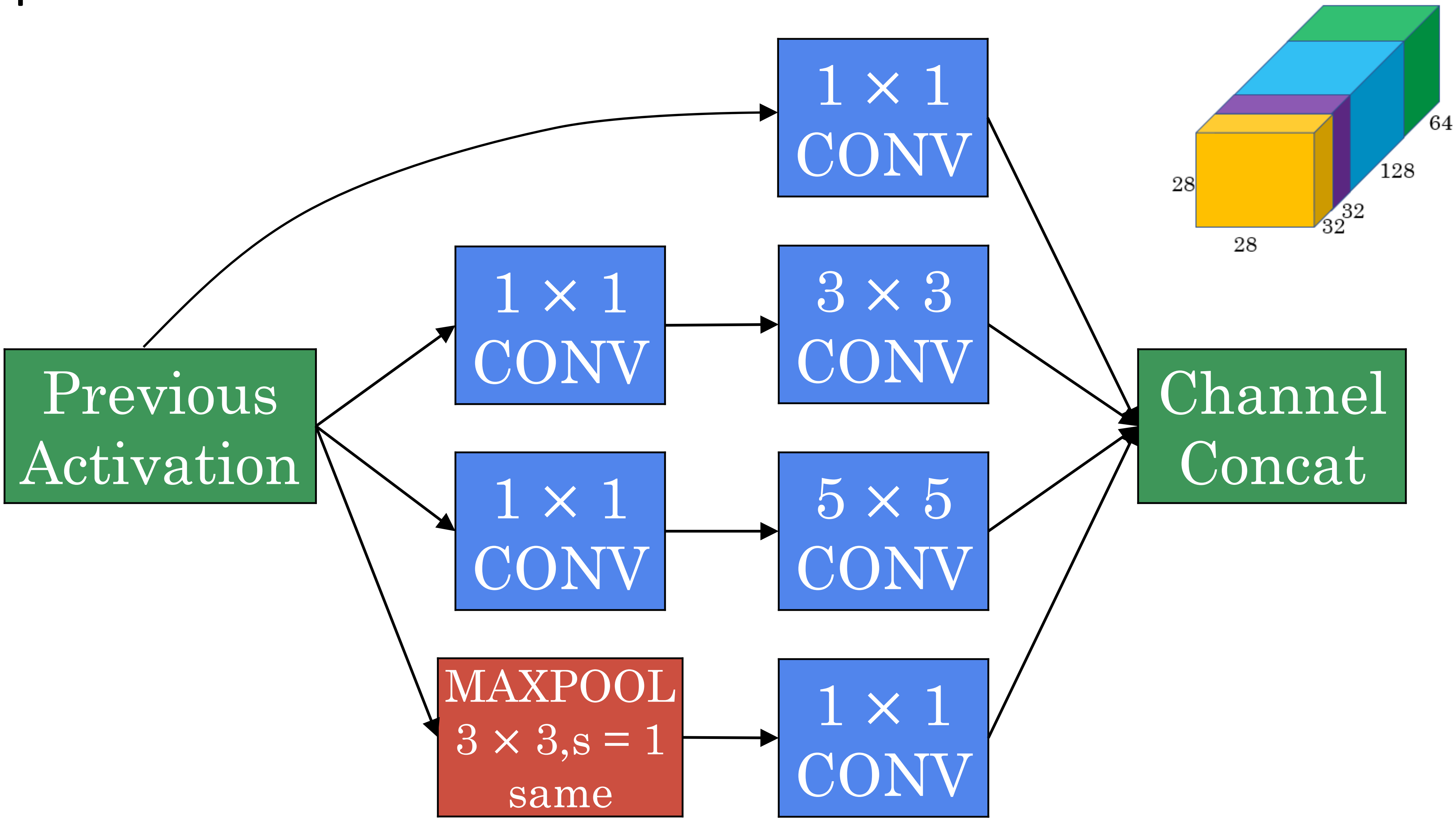


28x28x16x192=2.4m

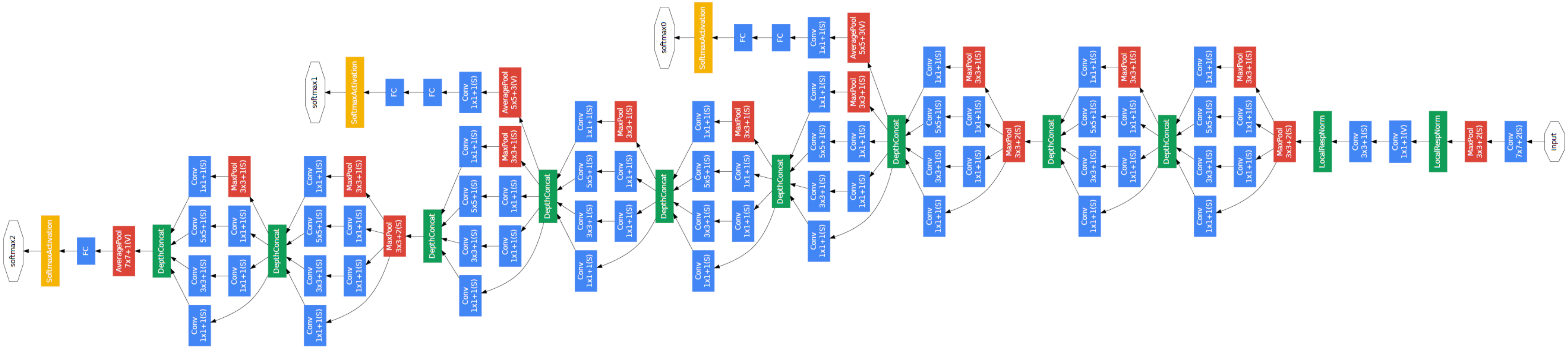
28x28x32x5x5x16=10m

Total computational cost=10m+2.4m=12.4m

Inception module



GoogLeNet



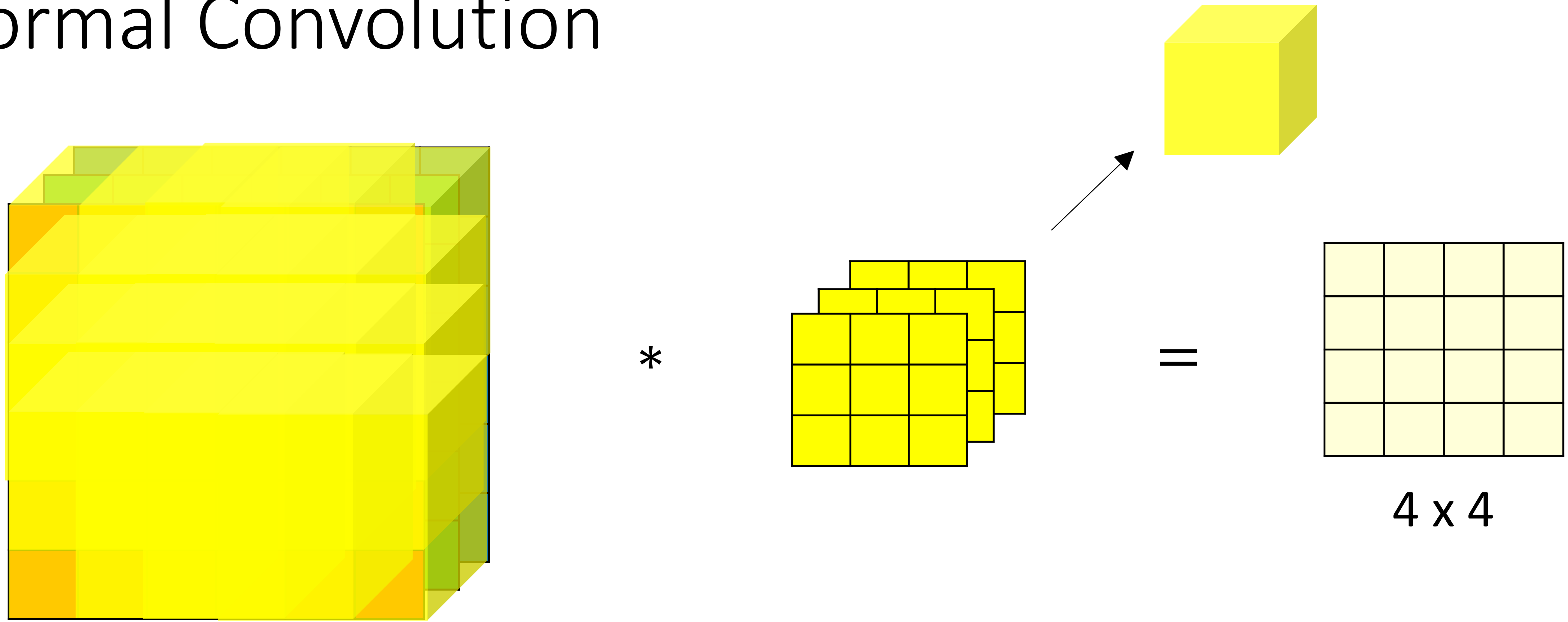
[Szegedy et al. 2014. Going deeper with convolutions]

Motivation for MobileNets

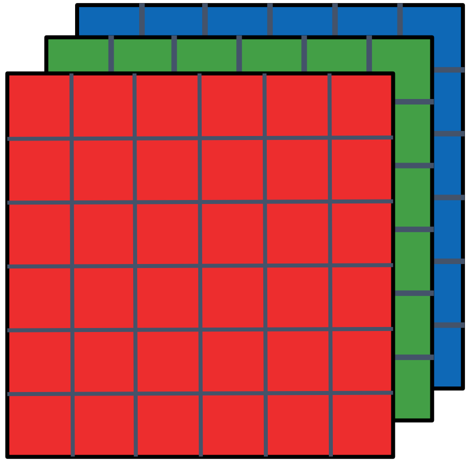
- Low computational cost at deployment
- Useful for mobile and embedded vision applications
- Key idea: Normal vs. depthwise-separable convolutions

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

Normal Convolution

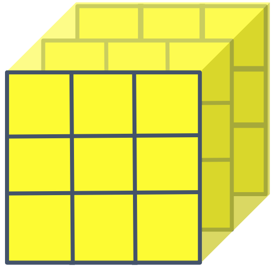


Normal Convolution



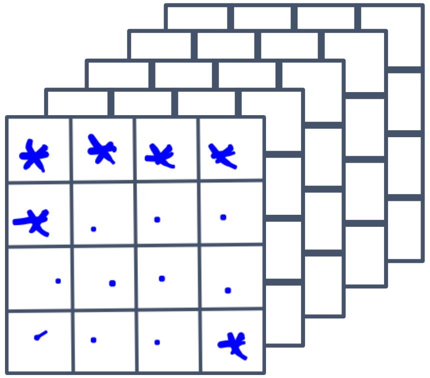
6 x 6 x 3

*



3 x 3 x 3

=



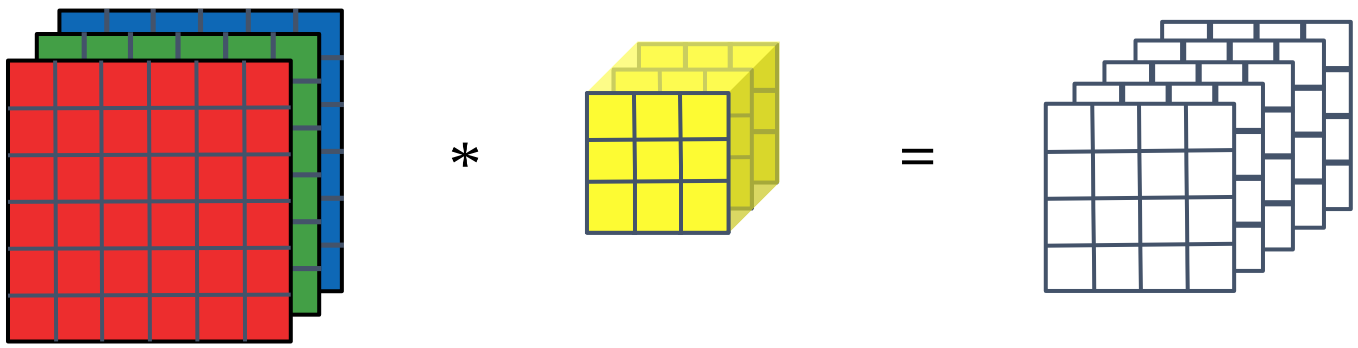
4*4*5

Computational cost = #filter params x # filter positions x # of filters

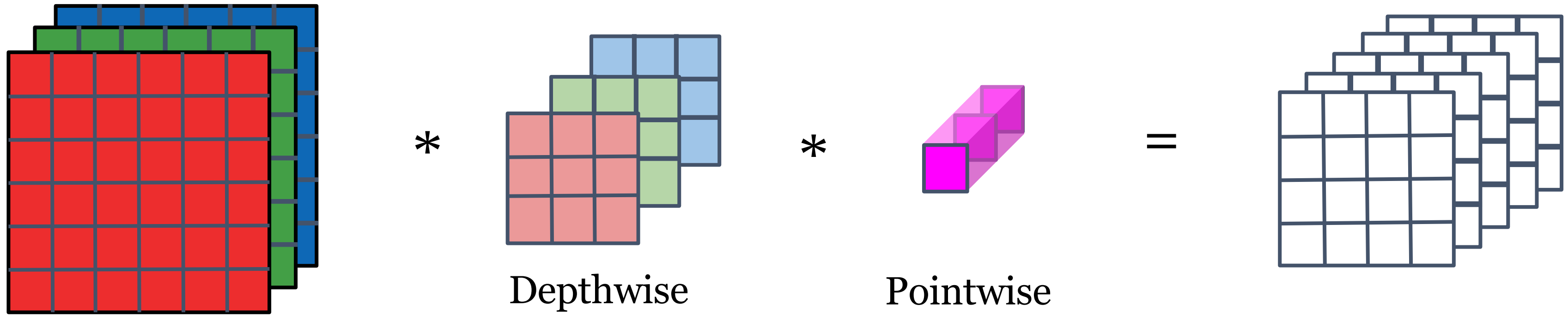
2160=3*3*3*4*4*5

Depthwise Separable Convolution

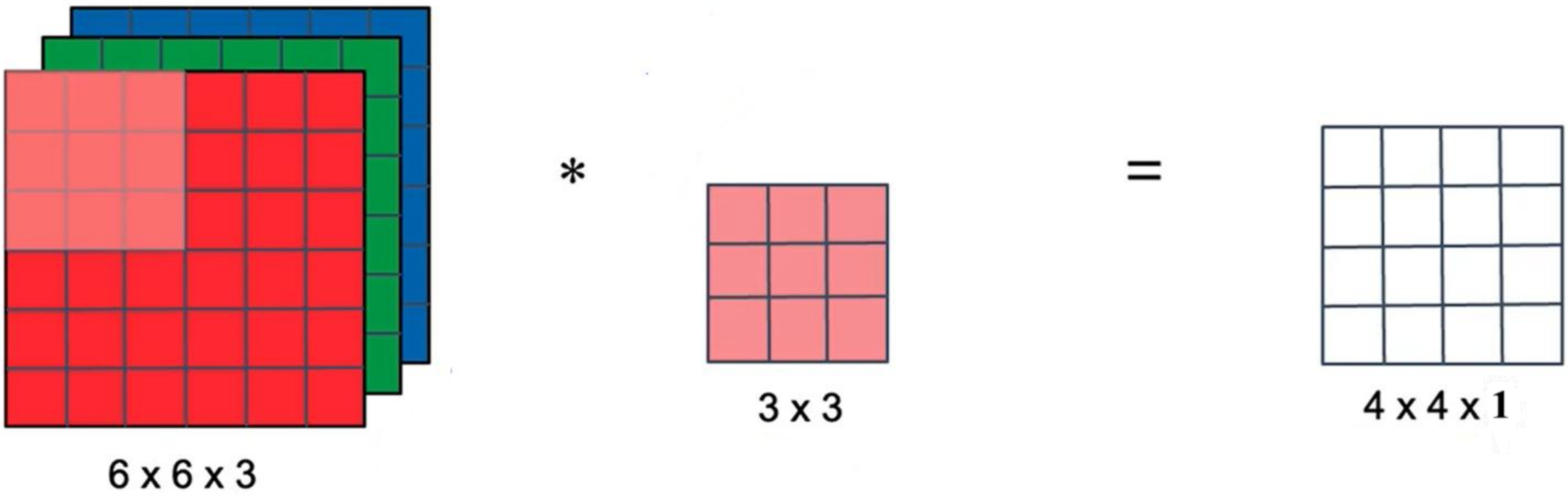
Normal Convolution



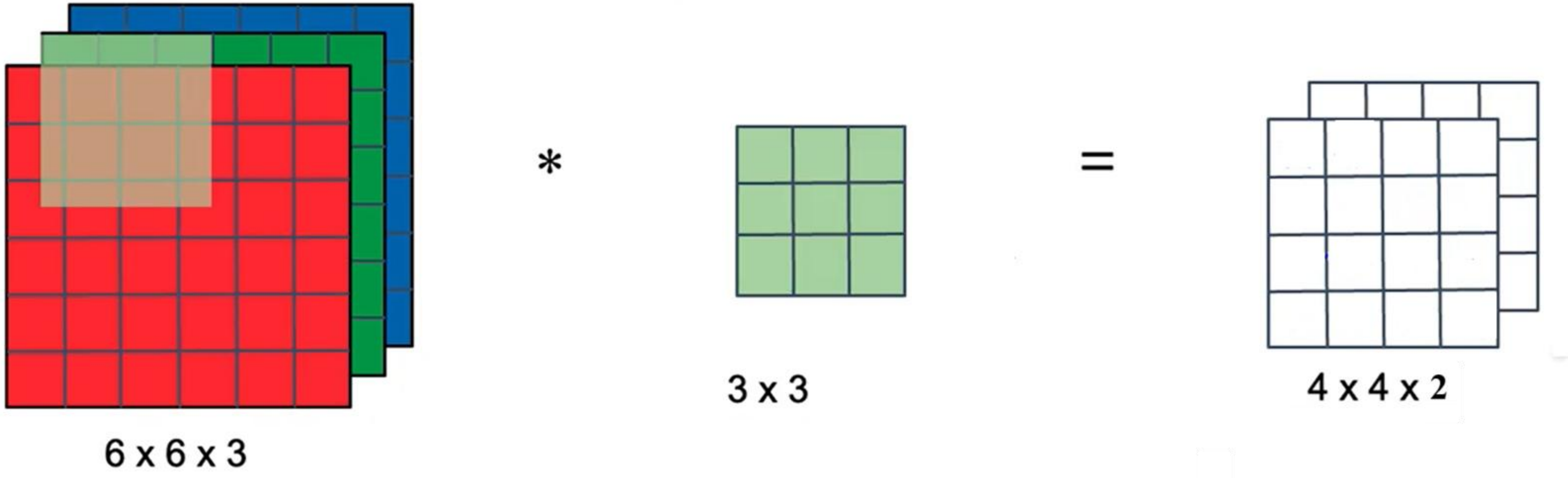
Depthwise Separable Convolution



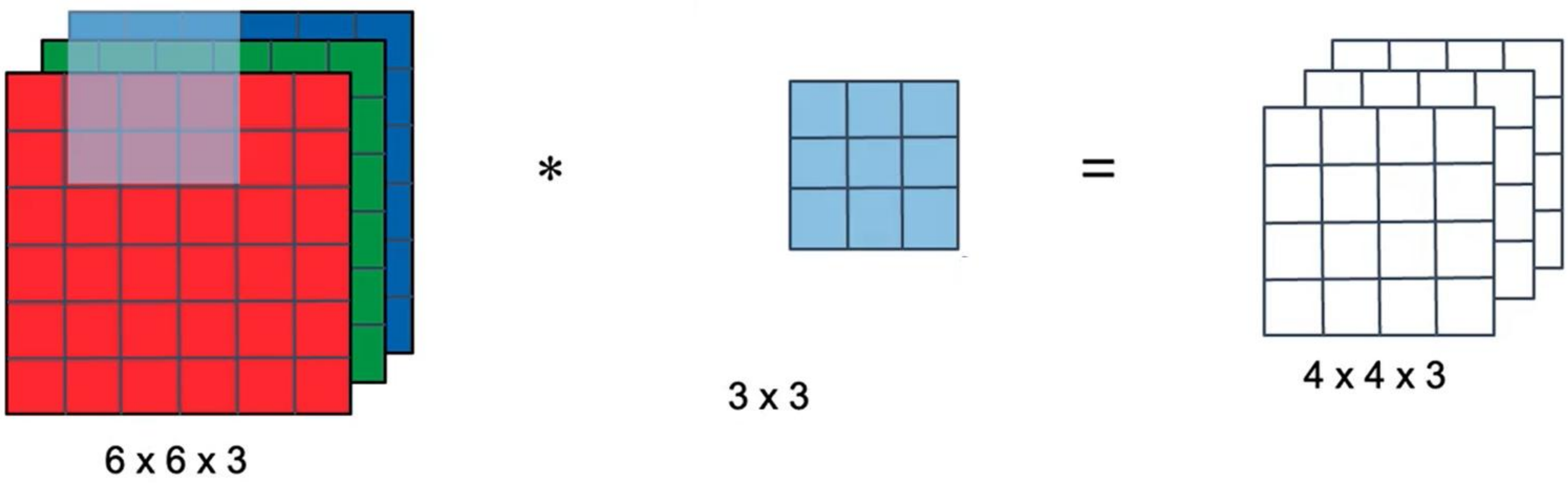
Depthwise Convolution



Depthwise Convolution



Depthwise Convolution

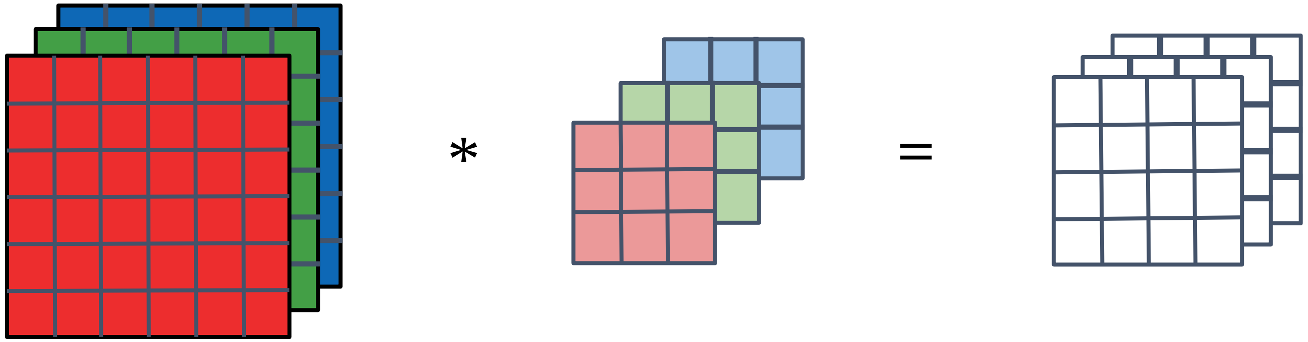


Computational cost = #filter params x # filter positions x # of filters

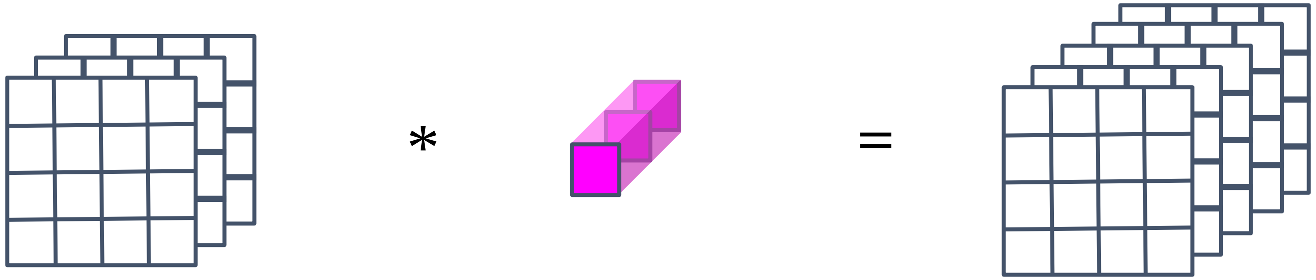
432=3*3*4*4*3

Depthwise Separable Convolution

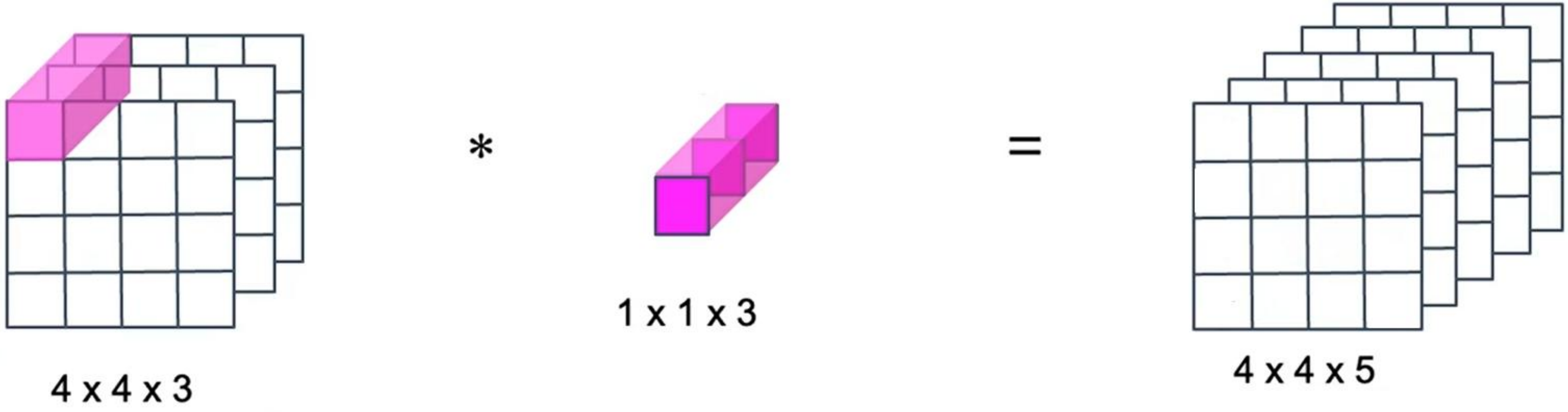
Depthwise Convolution



Pointwise Convolution



Pointwise Convolution

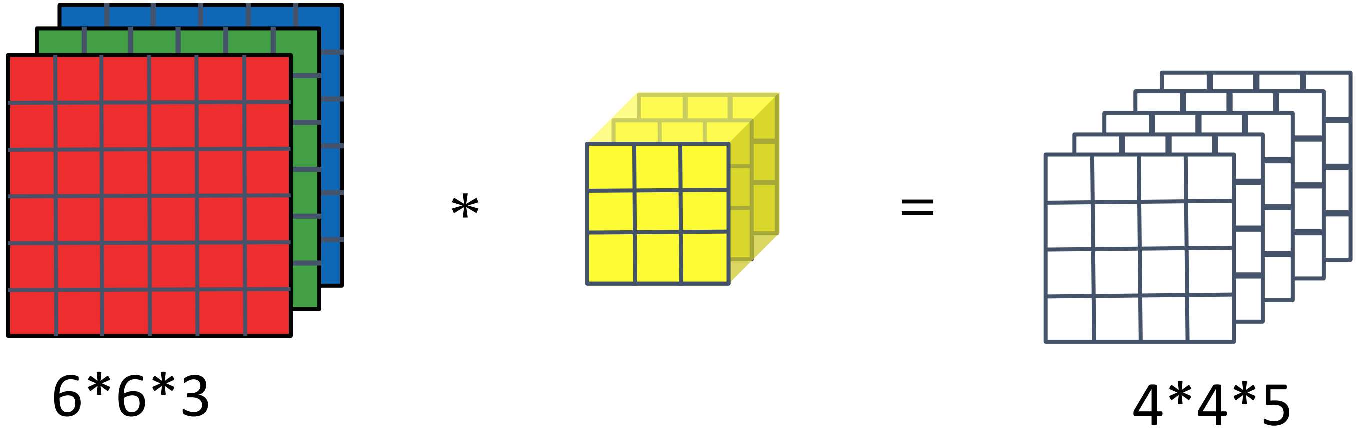


Computational cost = #filter params x # filter positions x # of filters

240=1*1*3*4*4*5

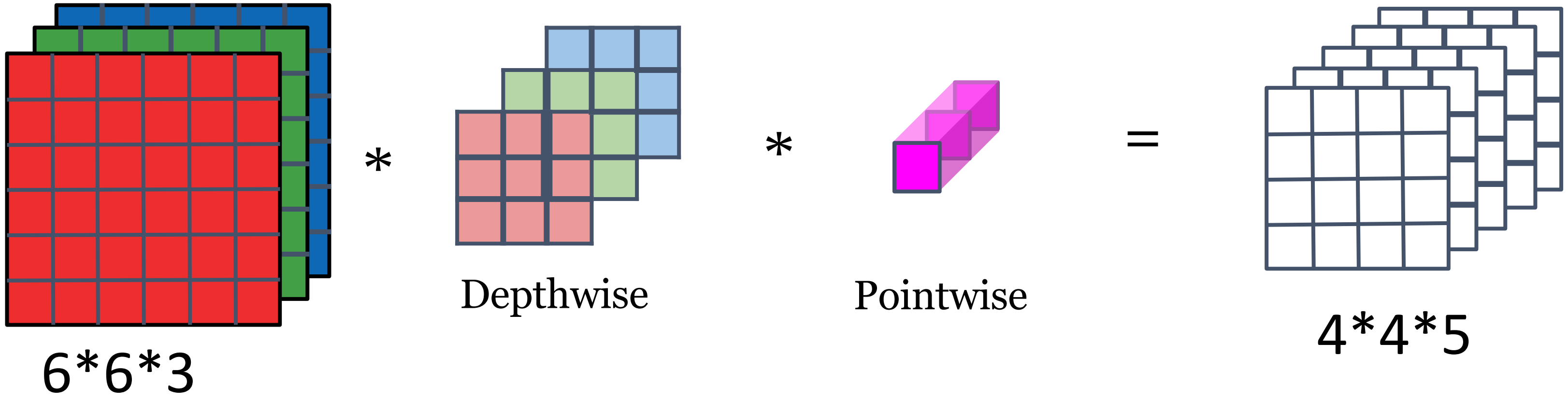
Depthwise Separable Convolution

Normal Convolution



$$3*3*3*4*4*5=2160$$

Depthwise Separable Convolution



$$3*3*4*4*3+1*1*3*4*4*5=432+240=672$$

Cost Summary

Cost of normal convolution 3*3*3*4*4*5=2160

Cost of depthwise separable convolution 3*3*4*4*3+1*1*3*4*4*5=432+240=672

$$\frac{672}{2160} = 0.31$$

n_c: $\#output\ channel$

n'_c: $\#input\ channel$

$filter_{size}=f^2$

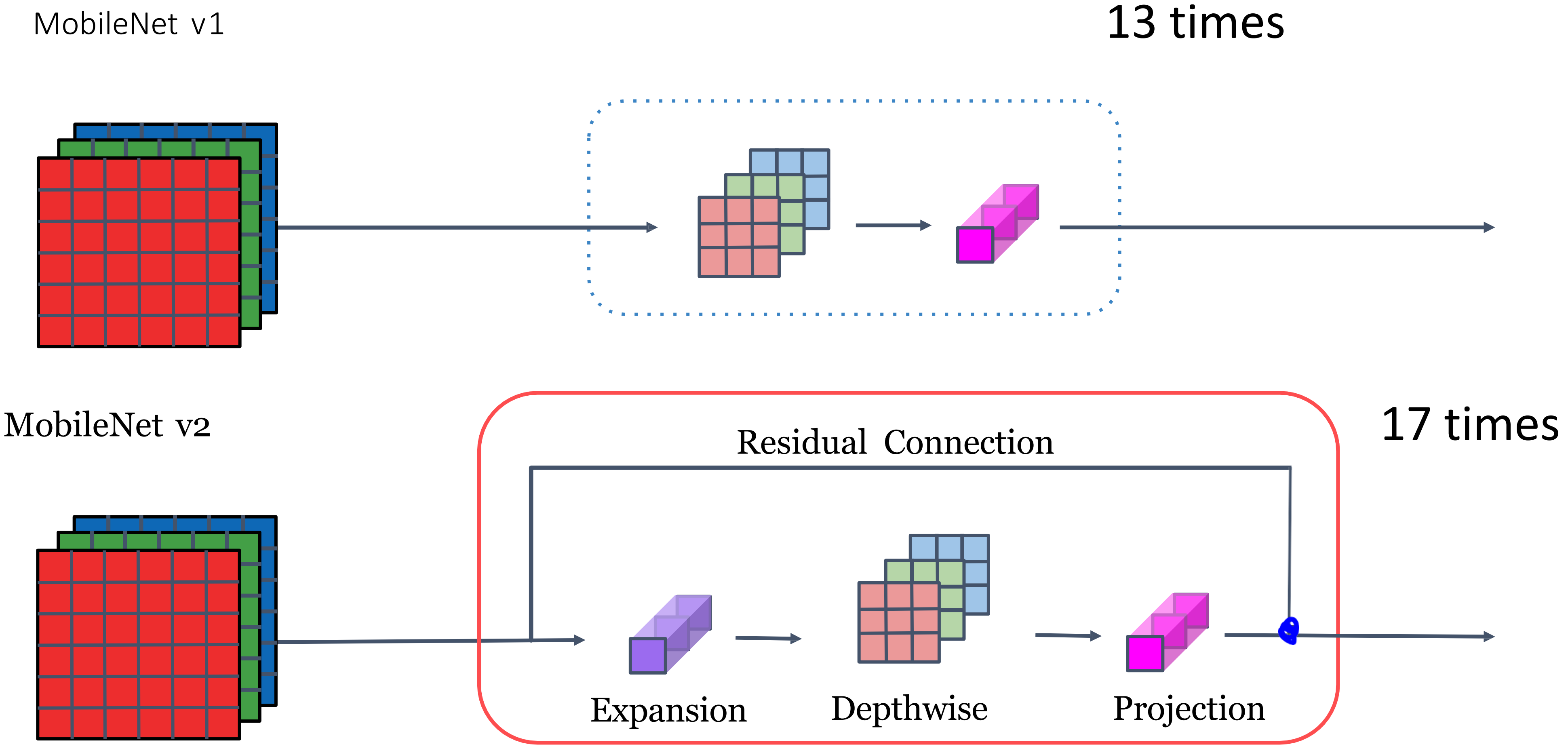
$$\frac{output_{size} * n'_c * f^2 + output_{size} * n_c * n'_c}{output_{size} * n_c * f^2 * n'_c} = \frac{1}{n_c} + \frac{1}{f^2} \sim \frac{1}{512} + \frac{1}{9} \sim \frac{1}{9}$$

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

Table 1. MobileNet Body Architecture

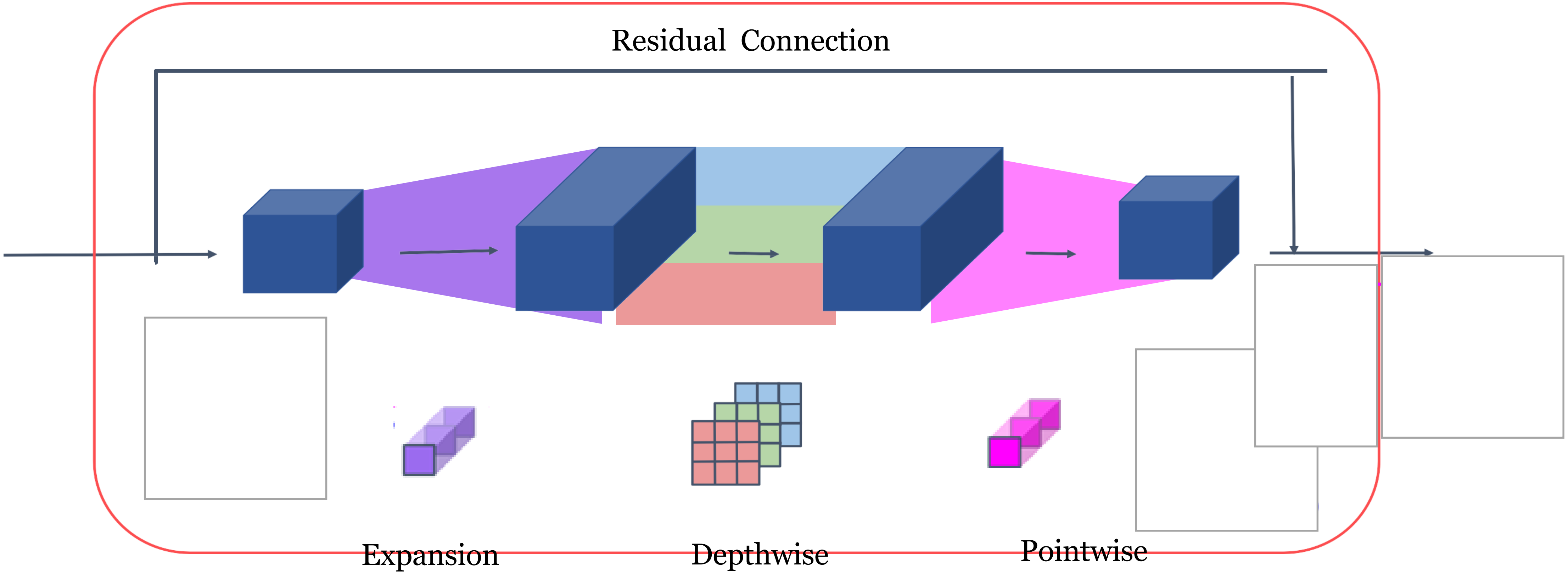
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
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	$1 \times 1 \times 1000$

MobileNet-V2



[Sandler et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks]

MobileNet v2 Bottleneck



[Sandler et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks]

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	$\frac{h}{s} \times \frac{w}{s} \times k'$

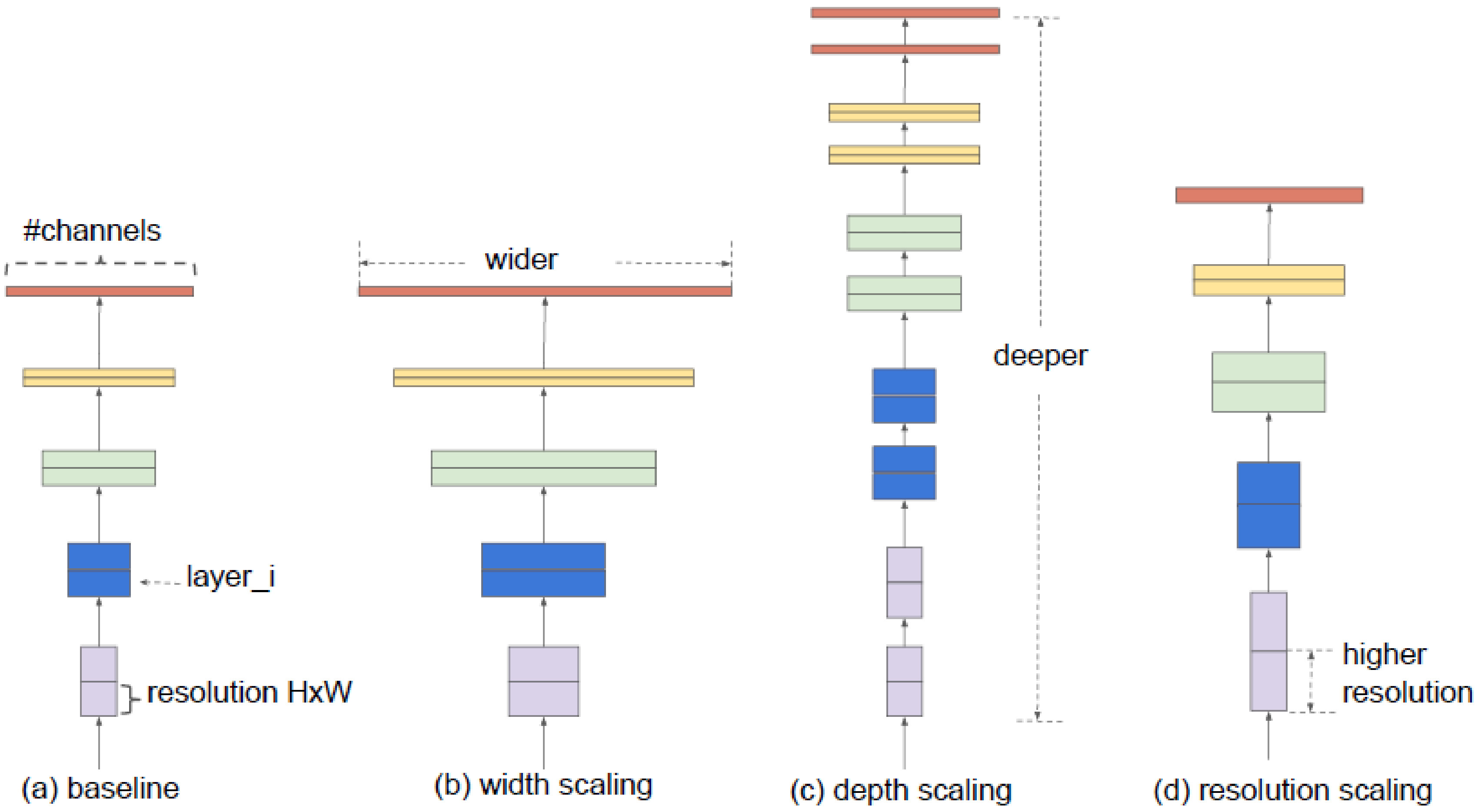
Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.

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	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

[Sandler et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks]

EfficientNet



[Tan and Le, 2019, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks]

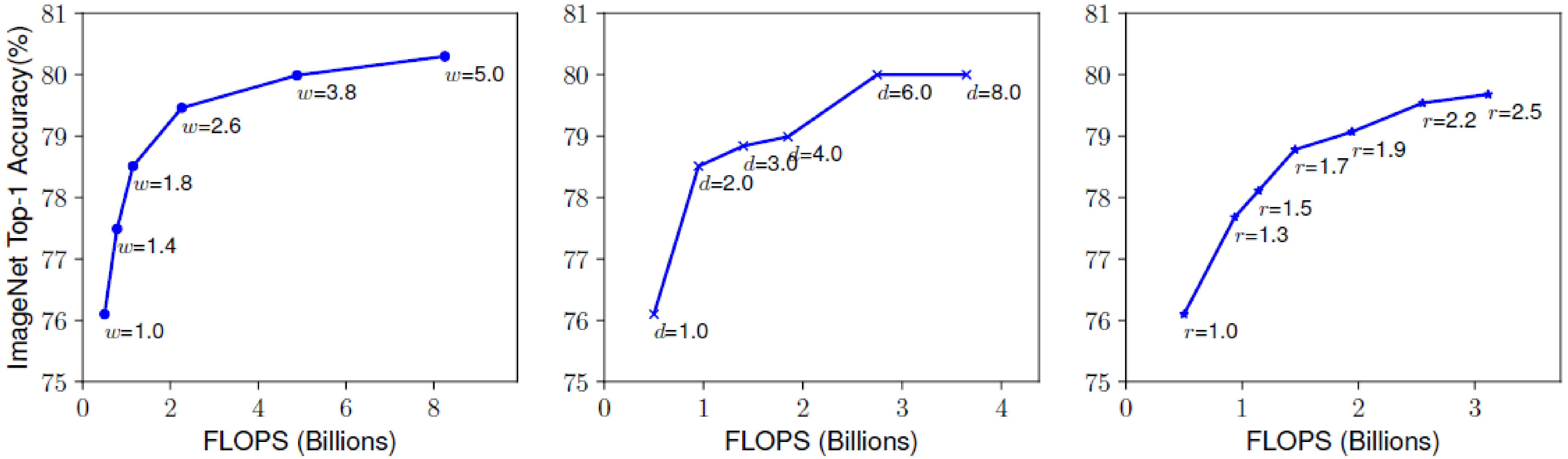


Figure 3. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling. Baseline network is described in Table 1.

- **Observation 1** – Scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models.

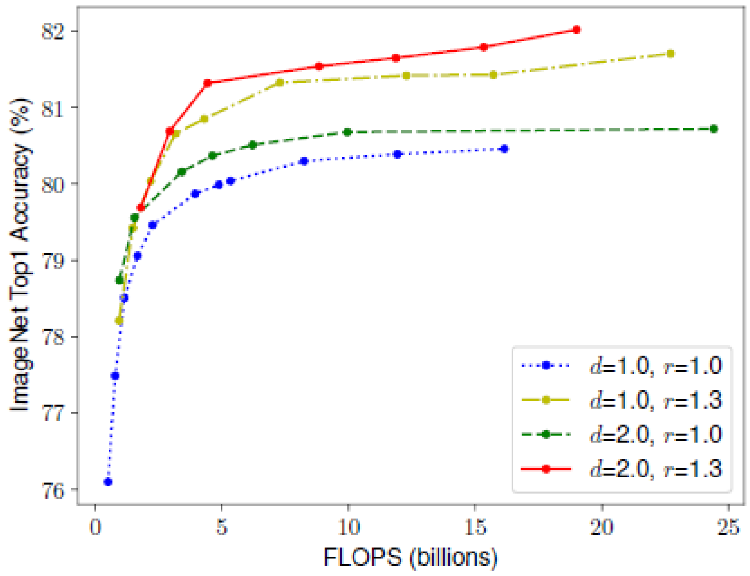


Figure 4. **Scaling Network Width for Different Baseline Networks.** Each dot in a line denotes a model with different width coefficient (w). All baseline networks are from Table 1. The first baseline network (d=1.0, r=1.0) has 18 convolutional layers with resolution 224x224, while the last baseline (d=2.0, r=1.3) has 36 layers with resolution 299x299.

- If we only scale network width w without changing depth (d=1.0) and resolution (r=1.0), the accuracy saturates quickly. With deeper (d=2.0) and higher resolution (r=1.3), width scaling achieves much better accuracy under the same FLOPS cost.

- **Observation 2** – In order to pursue better accuracy and efficiency, it is critical to balance all dimensions of network width, depth, and resolution during ConvNet scaling.

EfficientNet

Starting from the baseline EfficientNet-B0, we apply our compound scaling method to scale it up with two steps:

- STEP 1: we first fix $\phi = 1$, assuming twice more resources available, and do a small grid search of α, β, γ based on Equation 2 and 3. In particular, we find the best values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- STEP 2: we then fix α, β, γ as constants and scale up baseline network with different ϕ using Equation 3, to obtain EfficientNet-B1 to B7 (Details in Table 2).

$$\begin{aligned} \max_{d,w,r} \quad & Accuracy\big(\mathcal{N}(d,w,r)\big) \\ s.t. \quad & \mathcal{N}(d,w,r) = \bigodot_{i=1\dots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\big(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\big) \\ & \mathrm{Memory}(\mathcal{N}) \leq \mathrm{target\_memory} \\ & \mathrm{FLOPS}(\mathcal{N}) \leq \mathrm{target\_flops} \end{aligned} \tag{2}$$

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{3}$$

for any new ϕ, the total FLOPS will approximately increase by 2^ϕ.

[Tan and Le, 2019, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks]

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	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

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

Table 2. **EfficientNet Performance Results on ImageNet** (Russakovsky et al., 2015). All EfficientNet models are scaled from our baseline EfficientNet-B0 using different compound coefficient ϕ in Equation 3. ConvNets with similar top-1/top-5 accuracy are grouped together for efficiency comparison. Our scaled EfficientNet models consistently reduce parameters and FLOPS by an order of magnitude (up to 8.4x parameter reduction and up to 16x FLOPS reduction) than existing ConvNets.

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	-	-

We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

Table 3. **Scaling Up MobileNets and ResNet.**

Model	FLOPS	Top-1 Acc.
Baseline MobileNetV1 (Howard et al., 2017)	0.6B	70.6%
Scale MobileNetV1 by width (w=2)	2.2B	74.2%
Scale MobileNetV1 by resolution (r=2)	2.2B	72.7%
compound scale (*d*=1.4, *w*=1.2, *r*=1.3)	**2.3B**	**75.6%**
Baseline MobileNetV2 (Sandler et al., 2018)	0.3B	72.0%
Scale MobileNetV2 by depth (d=4)	1.2B	76.8%
Scale MobileNetV2 by width (w=2)	1.1B	76.4%
Scale MobileNetV2 by resolution (r=2)	1.2B	74.8%
MobileNetV2 compound scale	**1.3B**	**77.4%**
Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth (d=4)	16.2B	78.1%
Scale ResNet-50 by width (w=2)	14.7B	77.7%
Scale ResNet-50 by resolution (r=2)	16.4B	77.5%
ResNet-50 compound scale	**16.7B**	**78.8%**

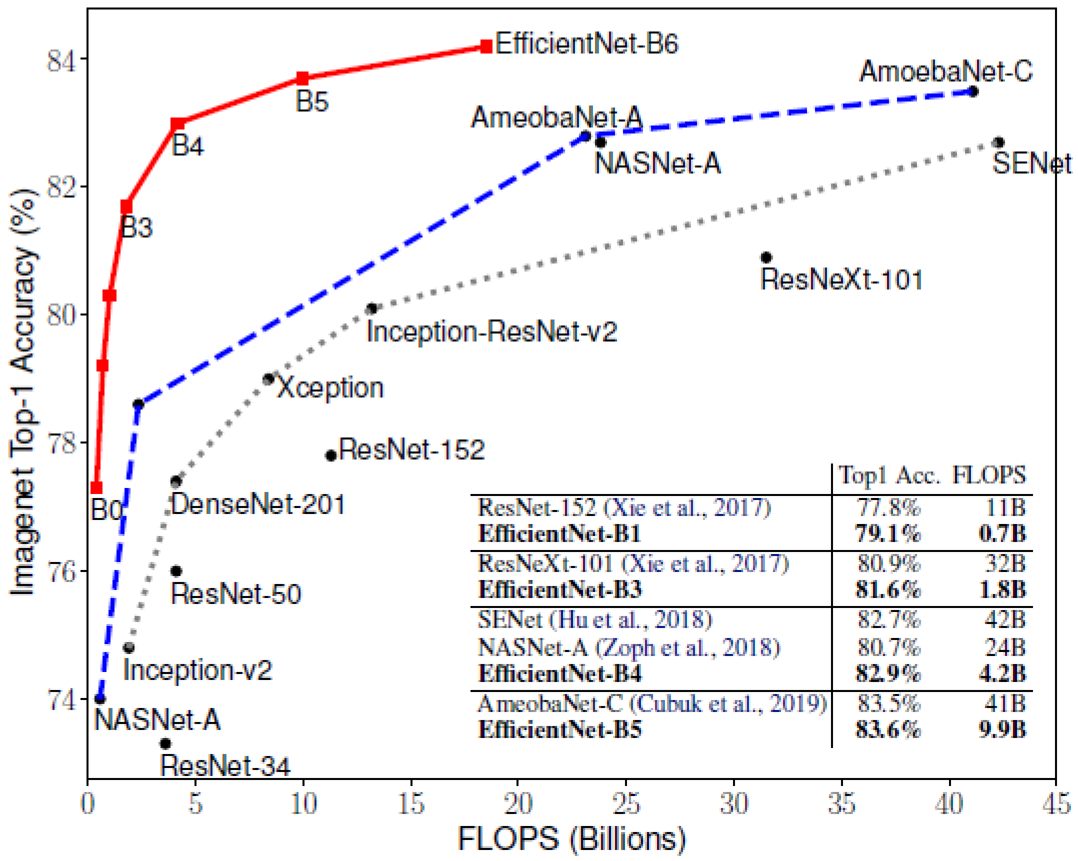
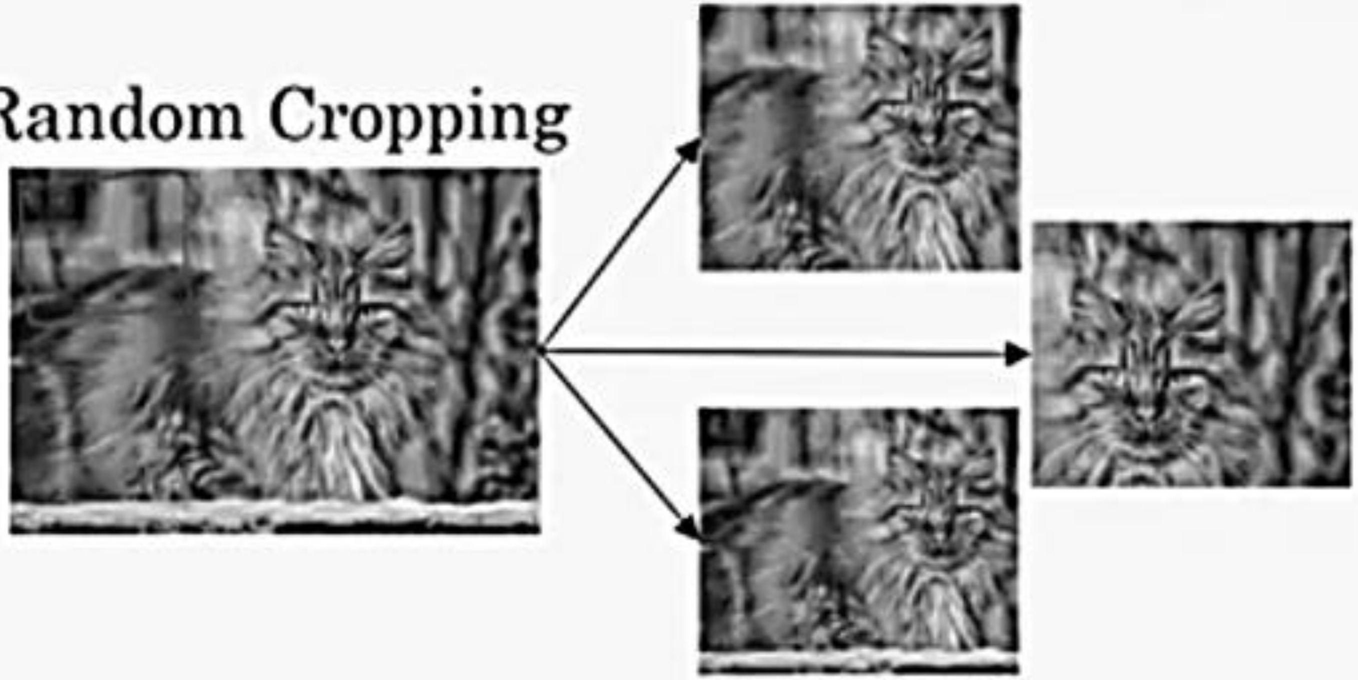


Figure 5. **FLOPS vs. ImageNet Accuracy –** Similar to Figure 1 except it compares FLOPS rather than model size.

Common augmentation method

Mirroring

Random Cropping



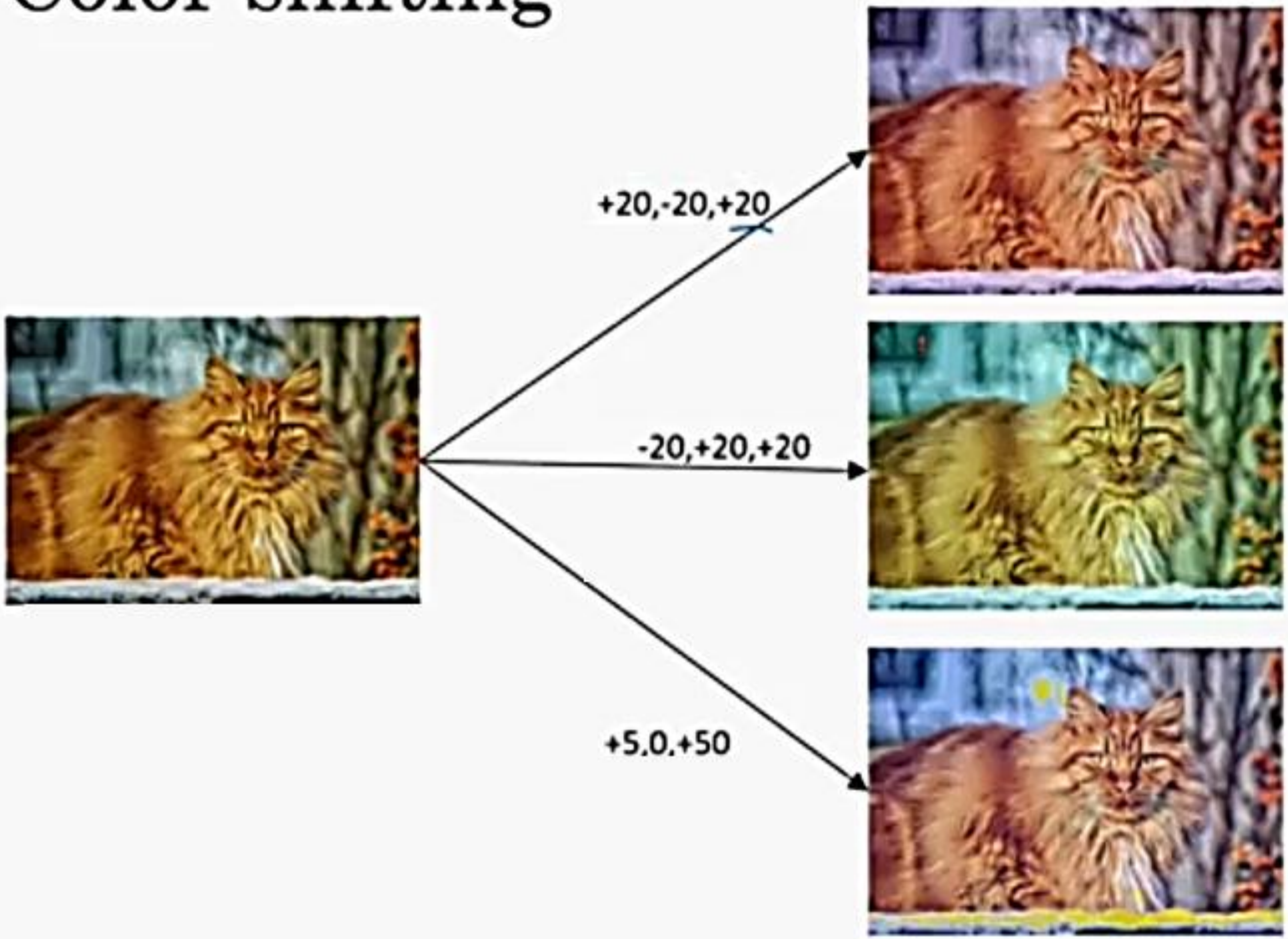
Rotation

Shearing

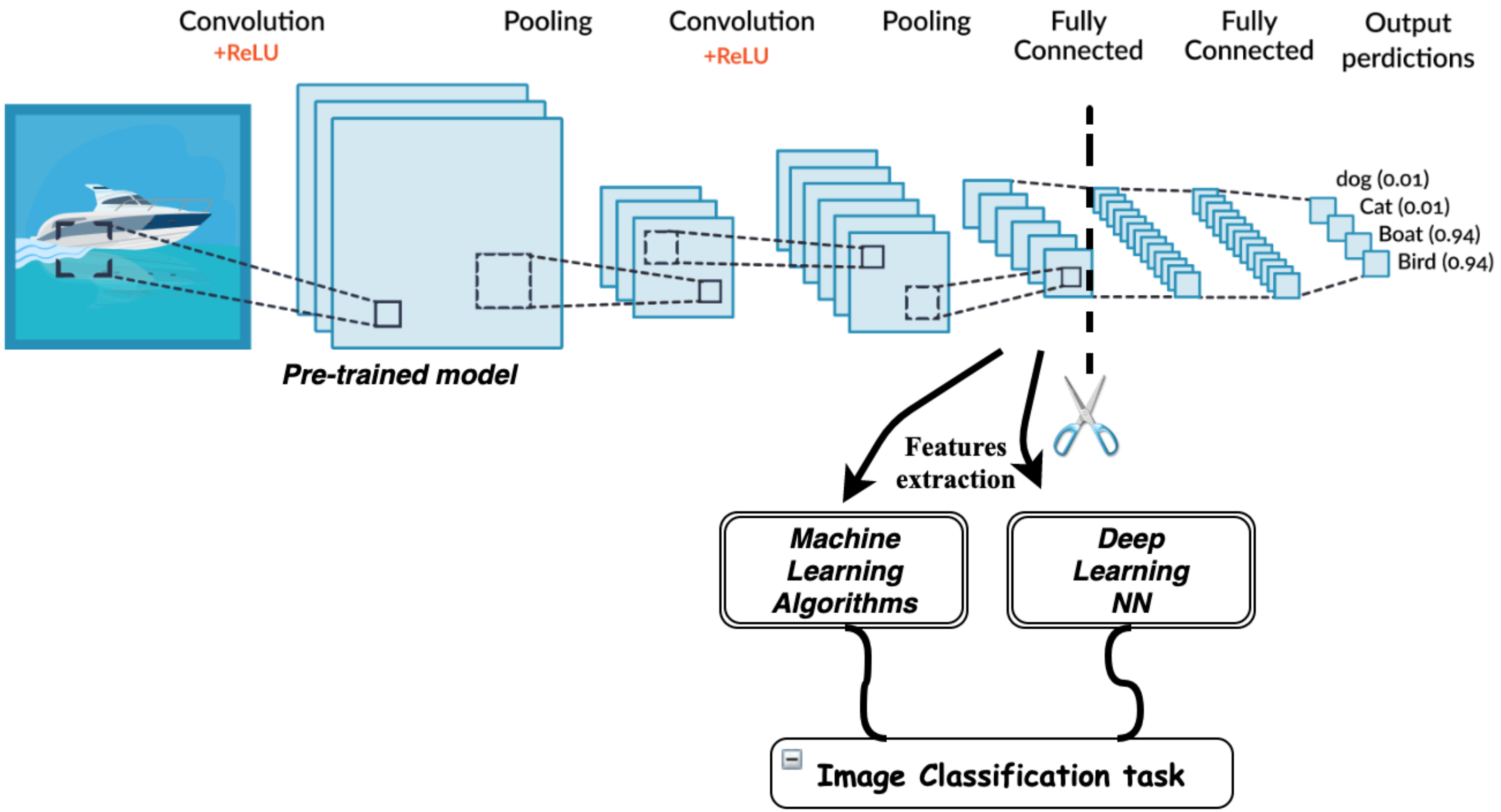
Local warping

...

Color shifting



Transfer Learning

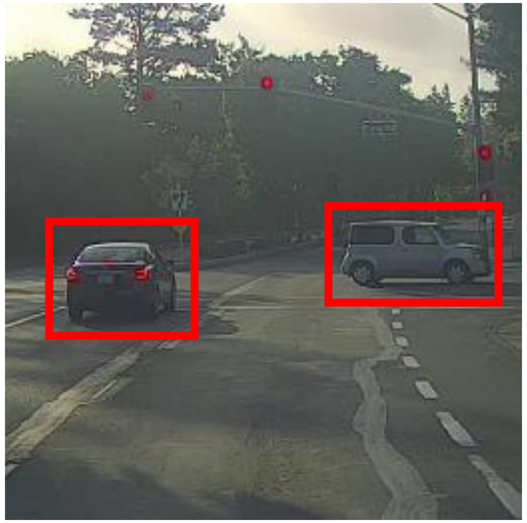


What are localization and detection?

Image classification

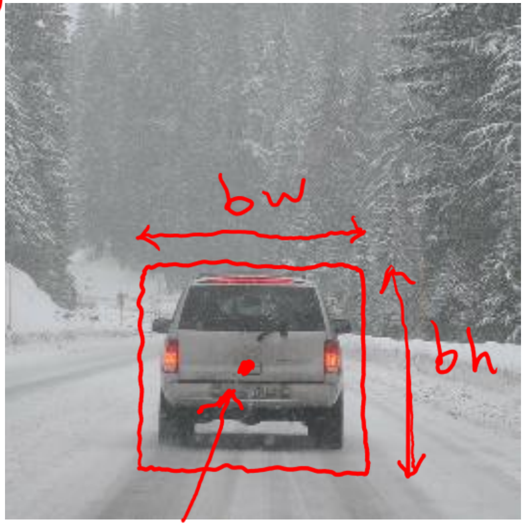
Classification with localization

Detection



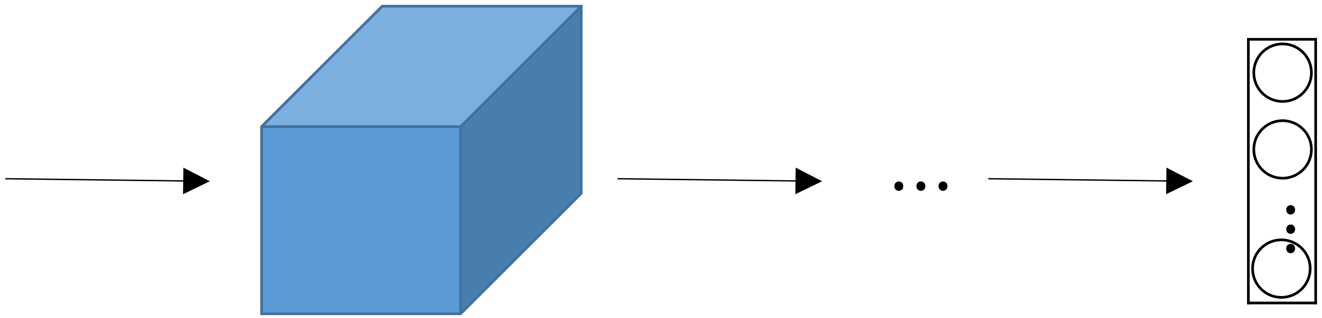
Classification with localization

(0,0)



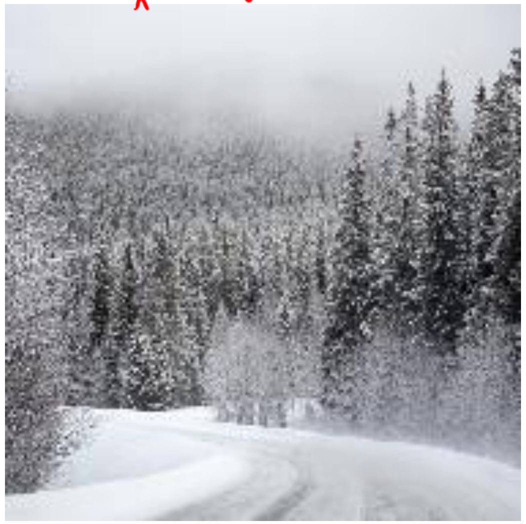
b_x, b_y

(1,1)



$b_x = 0.5$ $b_h = 0.3$

$b_y = 0.7$ $b_w = 0.4$



1 - pedestrian

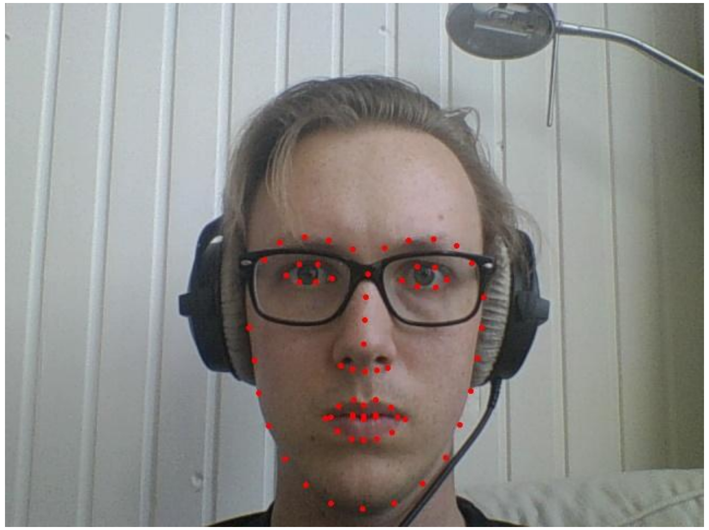
2 - car

3 - motorcycle

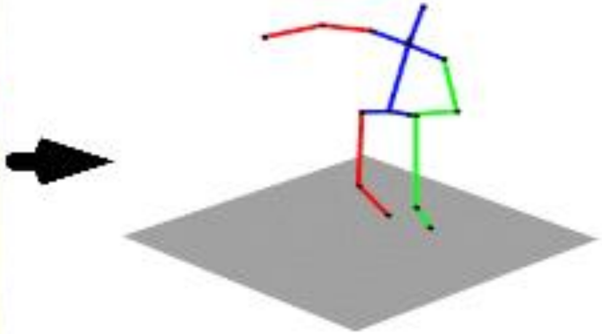
4 - background

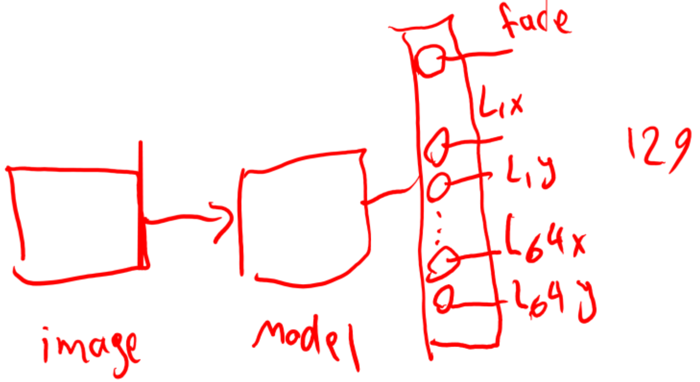
Landmark detection

b_x, b_y, b_h, b_w



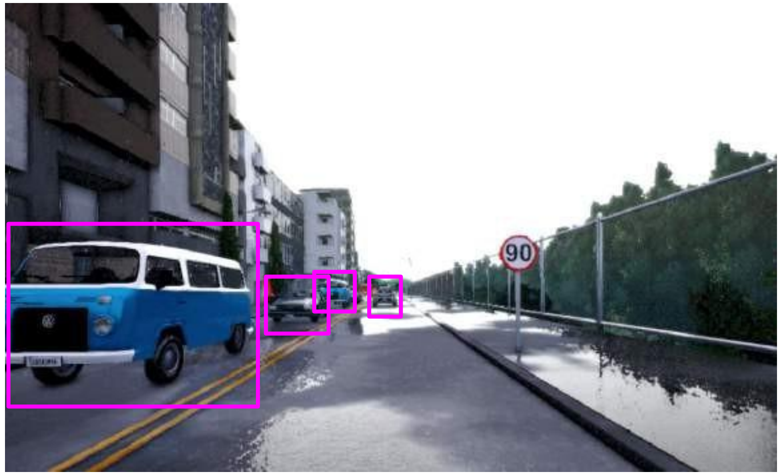
l_{1x}, l_{1y}
l_{2x}, l_{2y}
$\vdots$
l_{64x}, l_{64y}



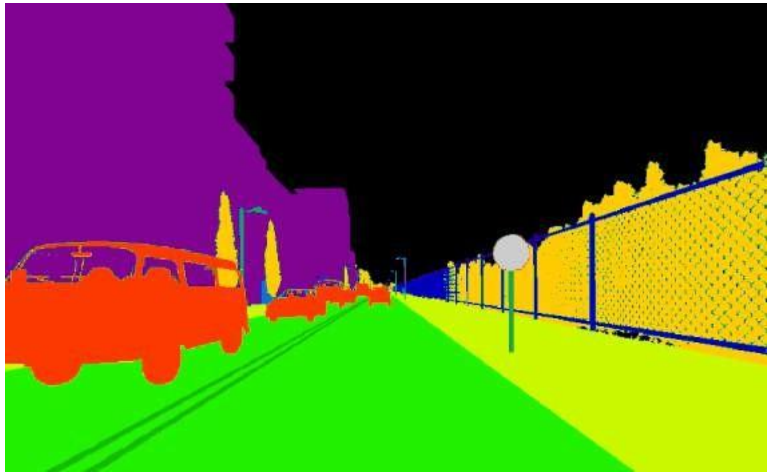


Object Detection vs. Semantic Segmentation

Input image

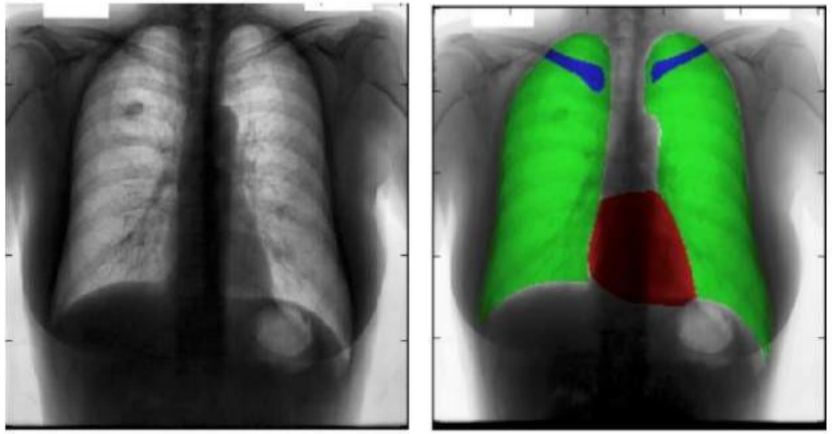


Object Detection

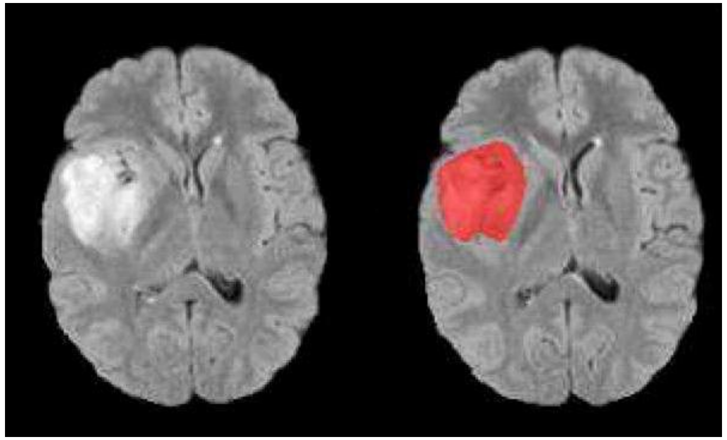


Semantic Segmentation

Motivation for U-Net



Chest X-Ray



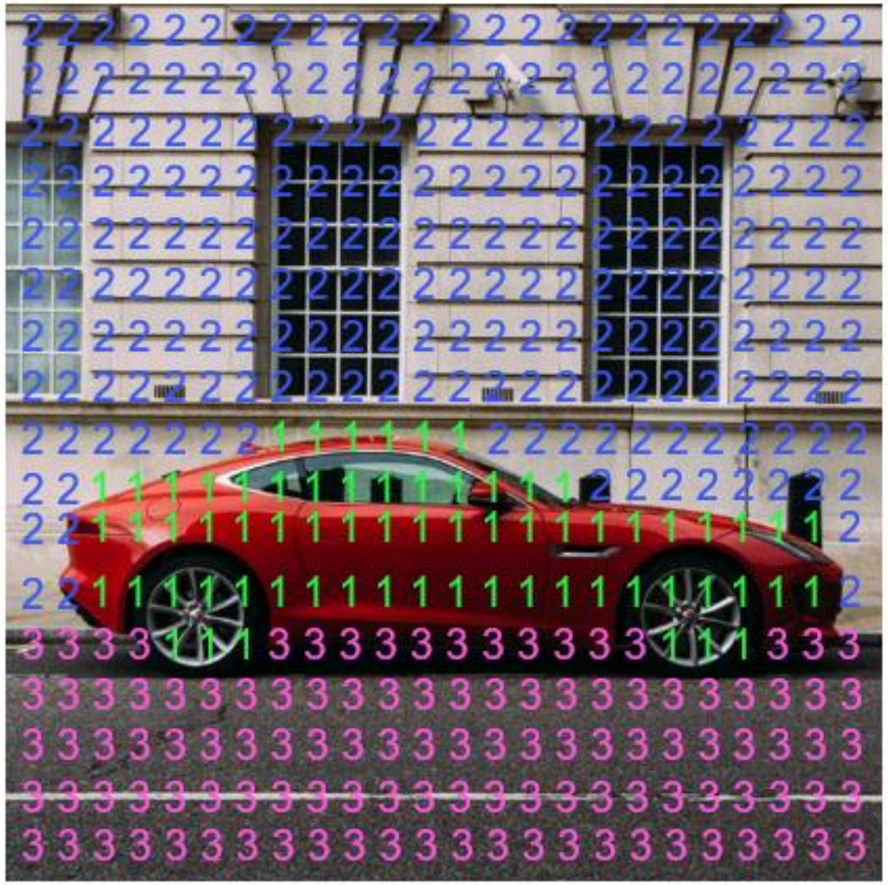
Brain MRI

[Novikov et al., 2017, Fully Convolutional Architectures for Multi-Class Segmentation in Chest Radiographs]
[Dong et al., 2017, Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks]

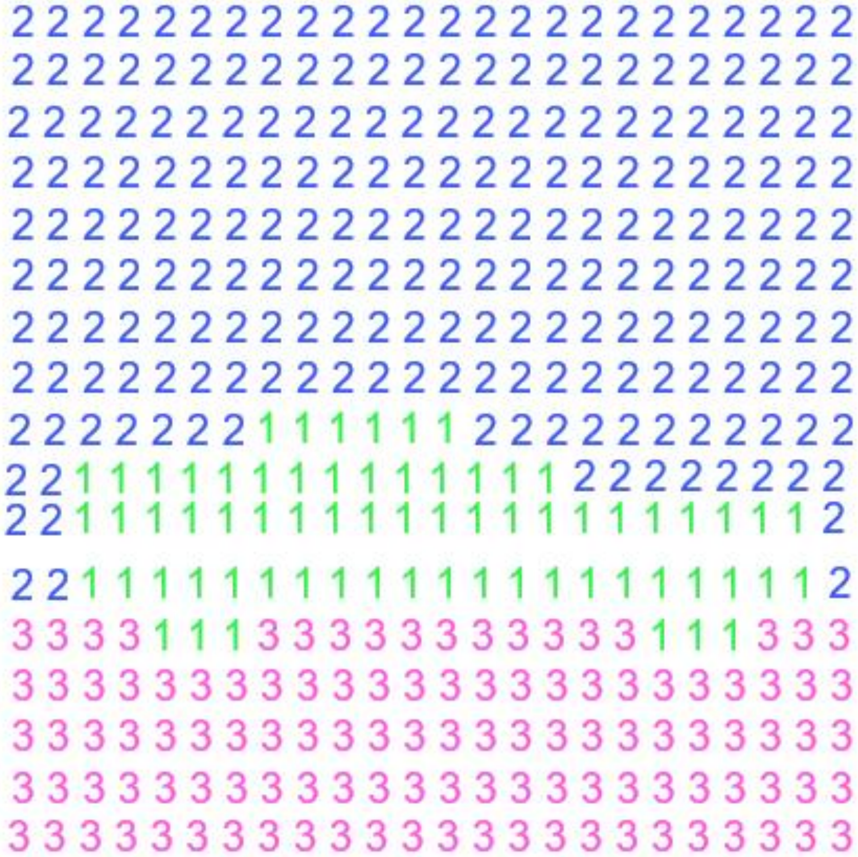
Per-pixel class labels

1. Car
0. Not Car

Per pixel class label

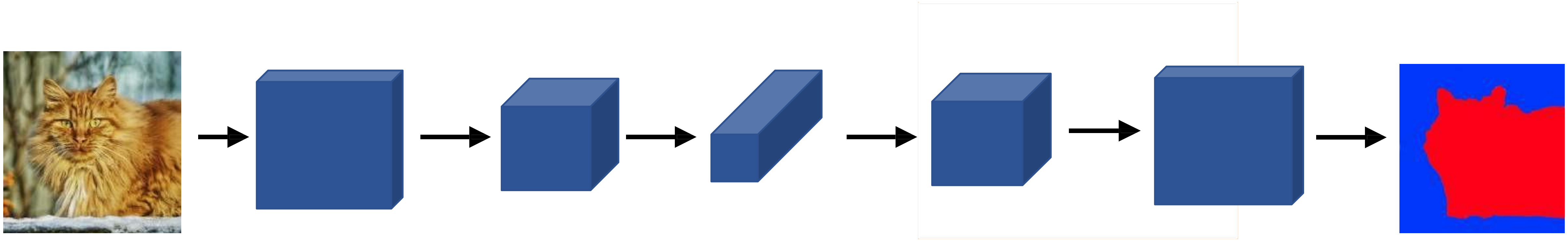


1. Car
2. Building
3. Road

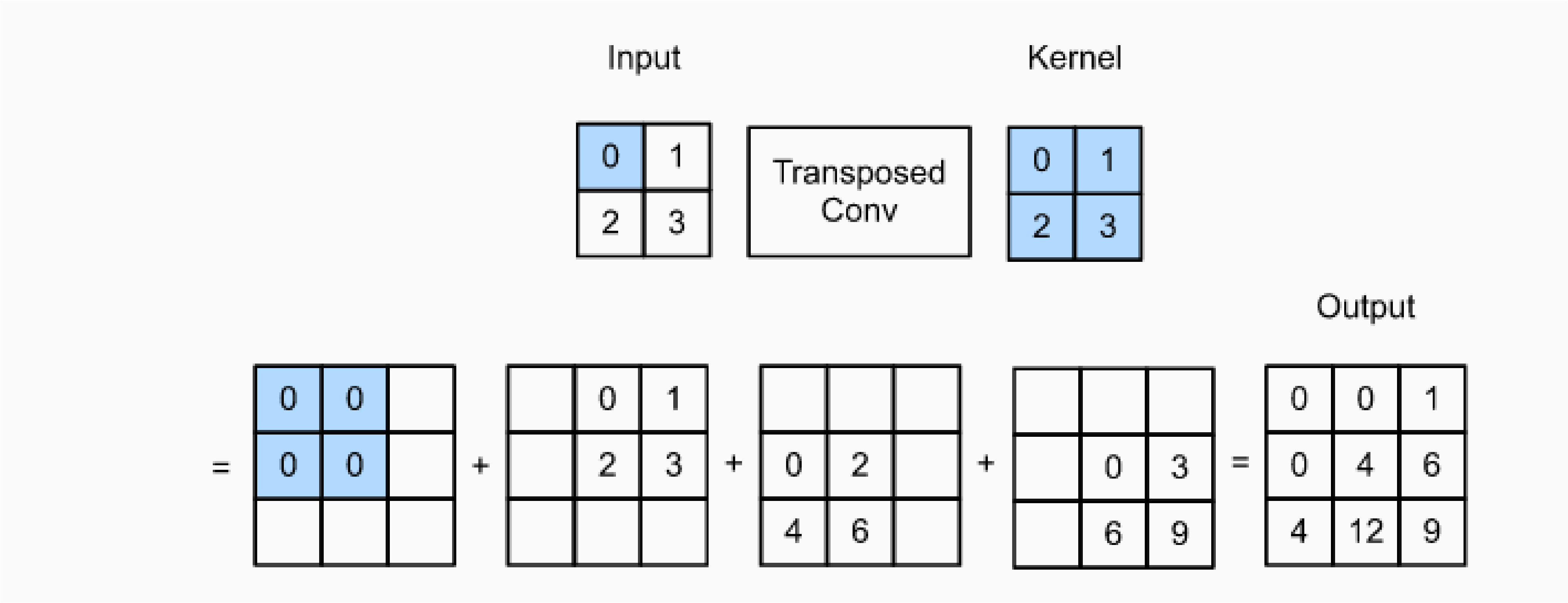


Segmentation Map

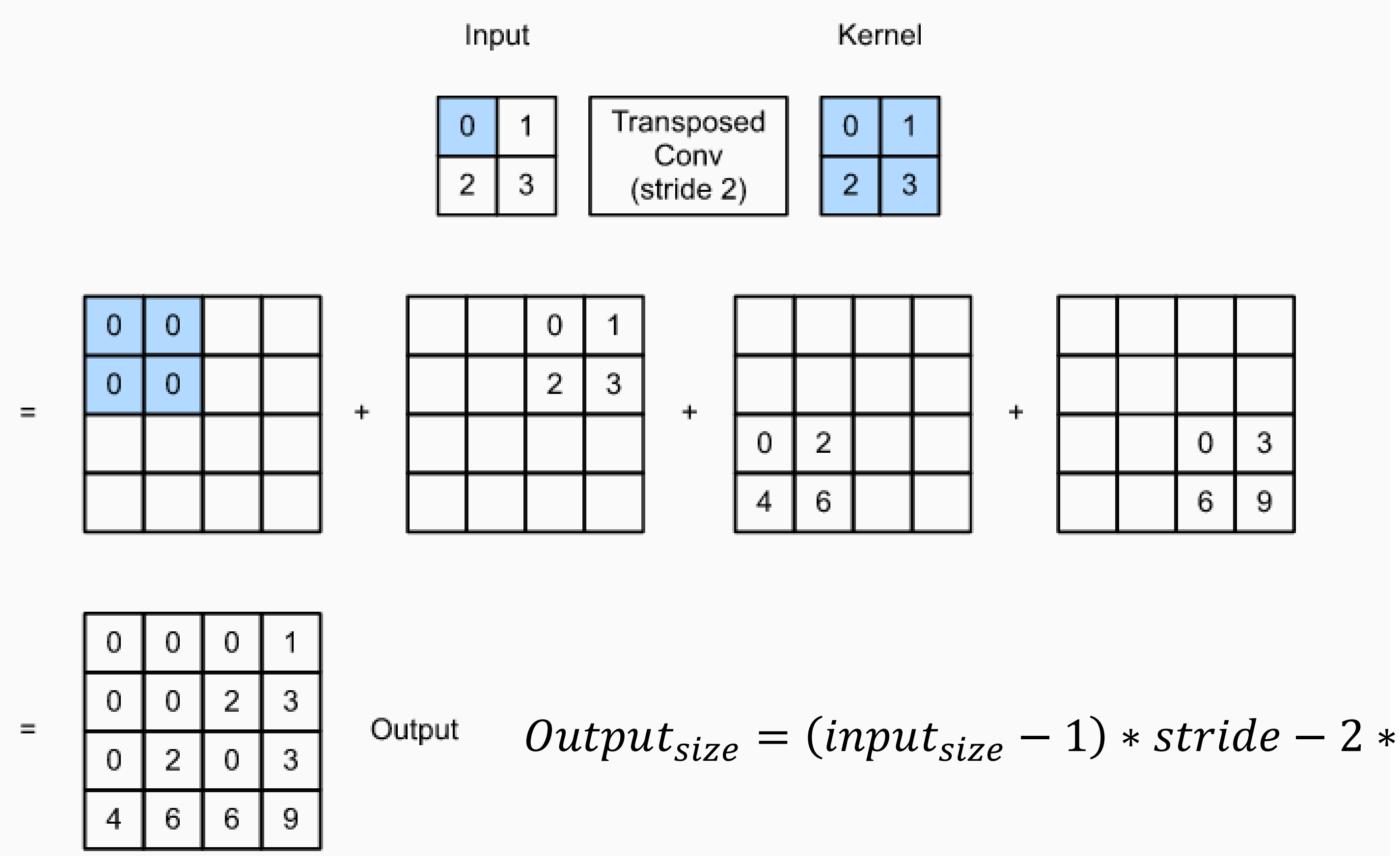
Deep Learning for Semantic Segmentation



Transpose Convolution



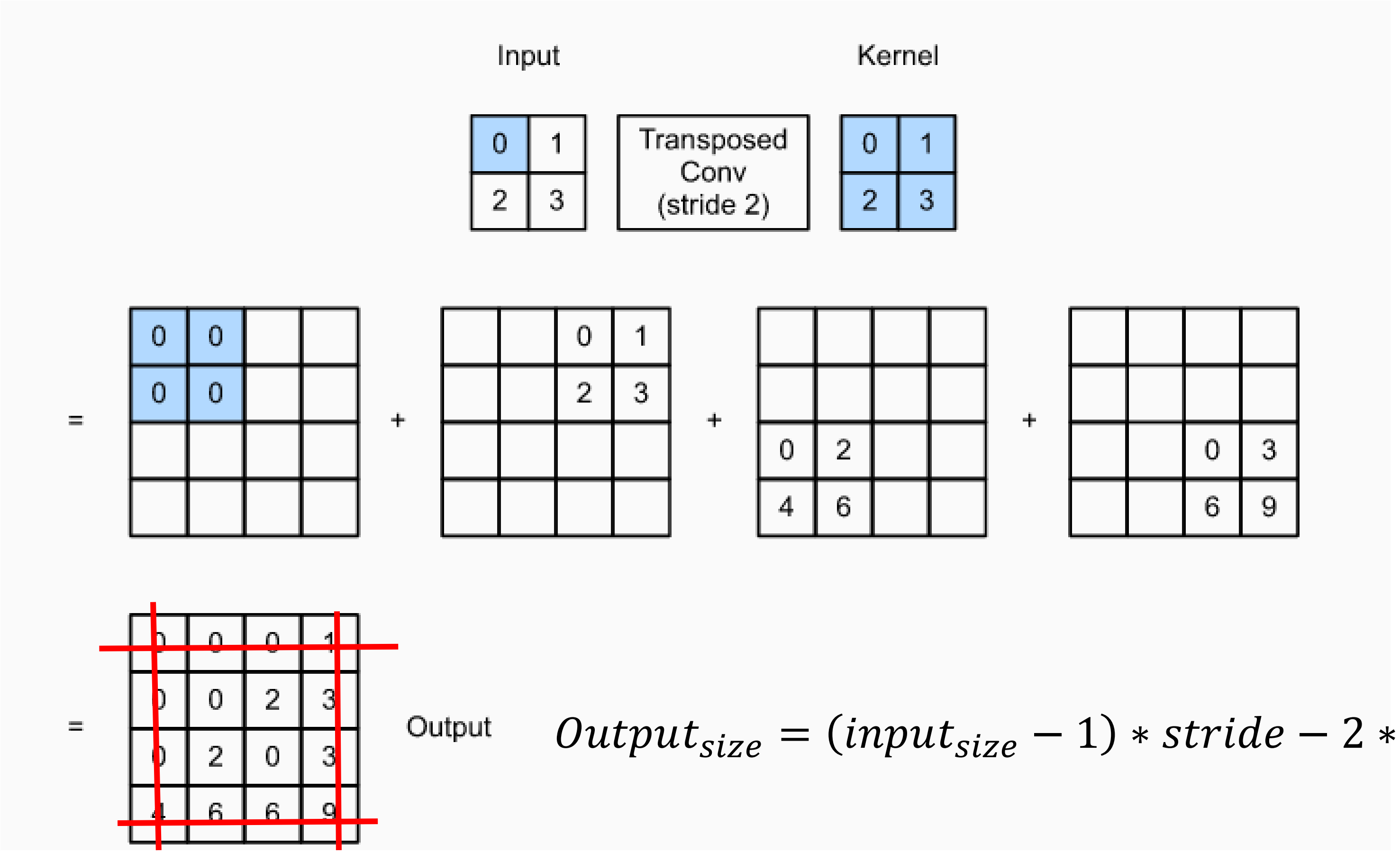
Transpose Convolution with stride



Stride=2

$$Output_{size} = (input_{size} - 1) * stride - 2 * padding + kernel\_size$$

Transpose Convolution with stride and padding

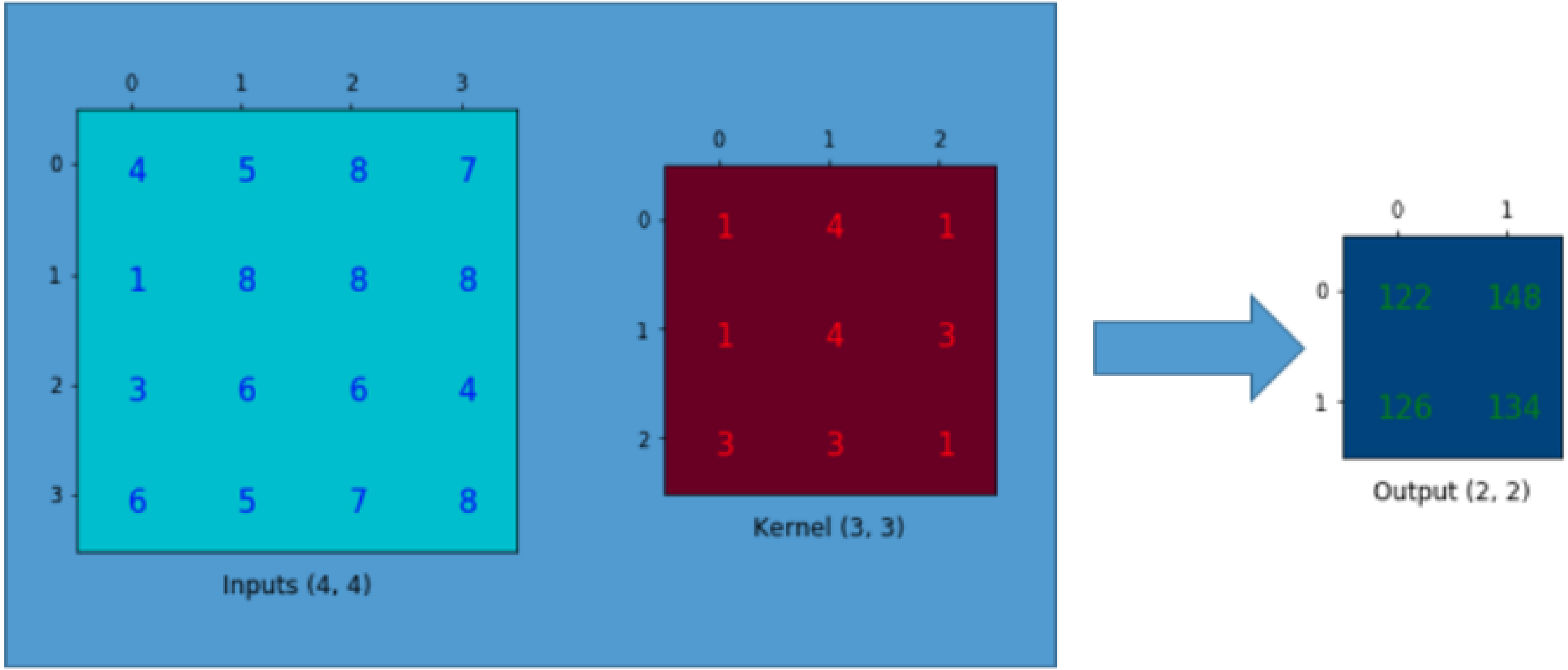


Stride=2

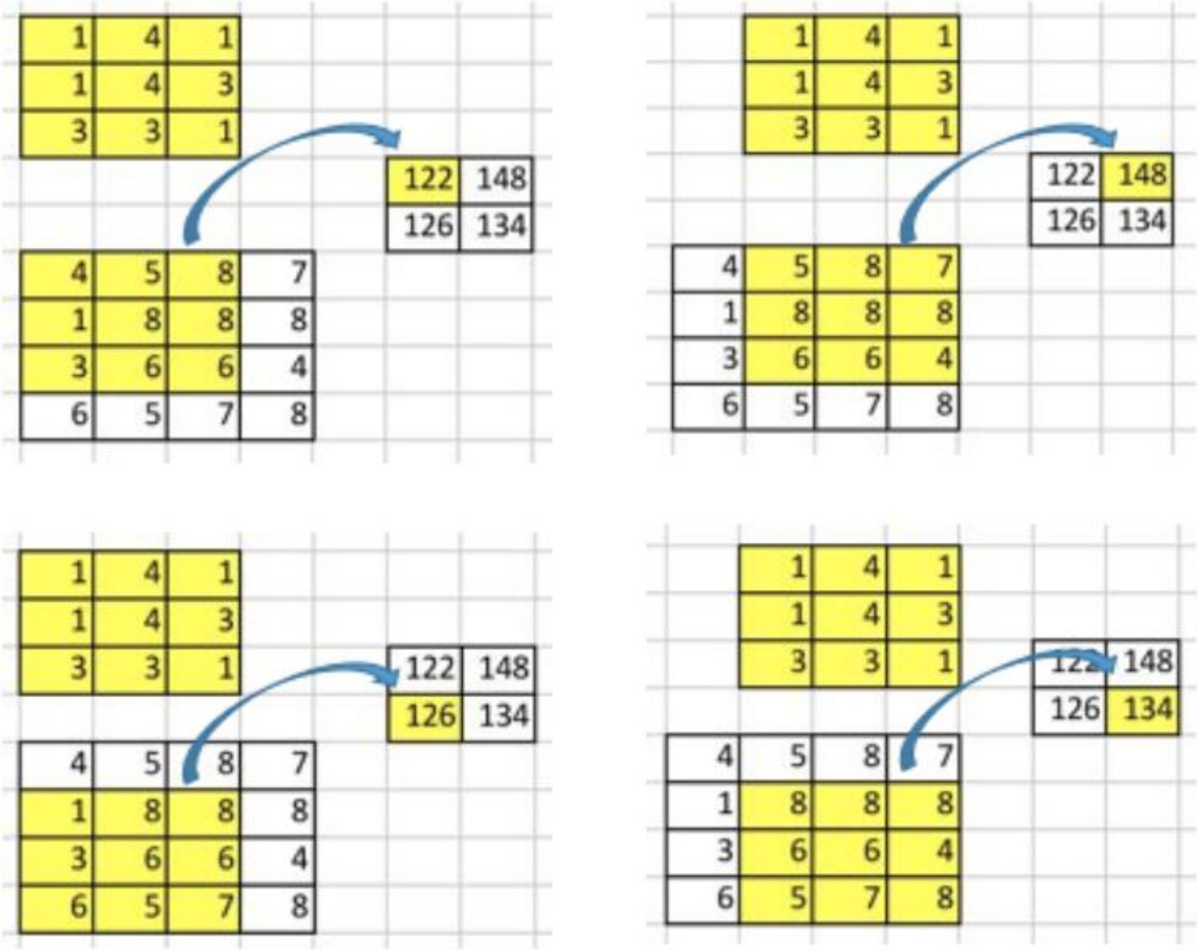
Padding=1

$$Output_{size} = (input_{size} - 1) * stride - 2 * padding + kernel\_size$$

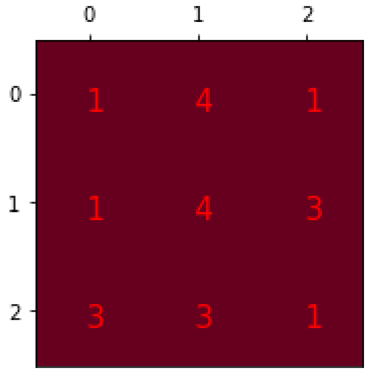
Why transpose convolution?



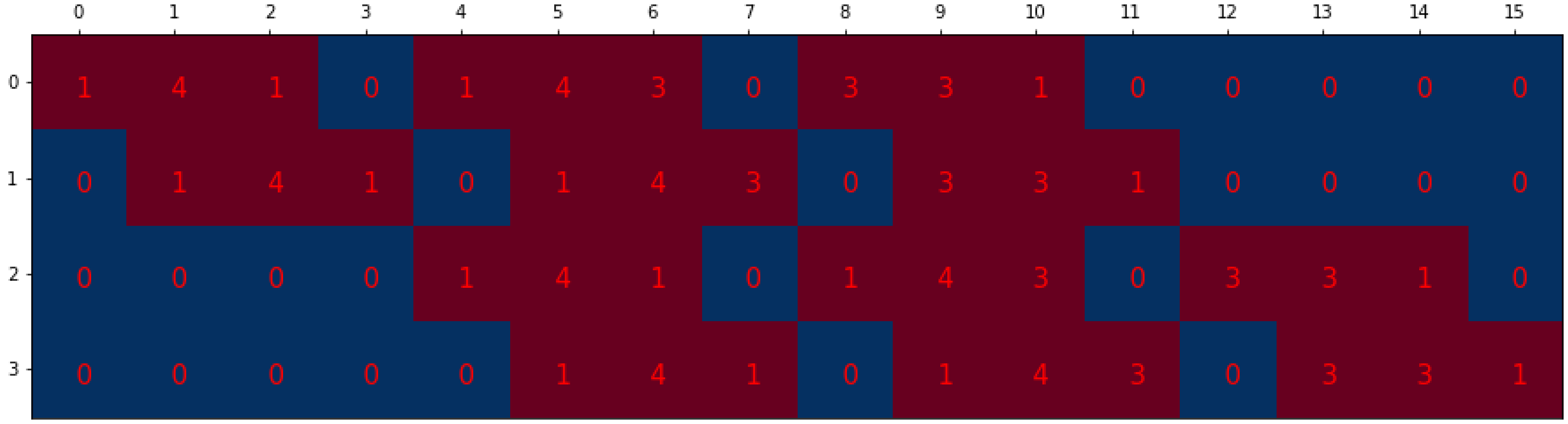
https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0



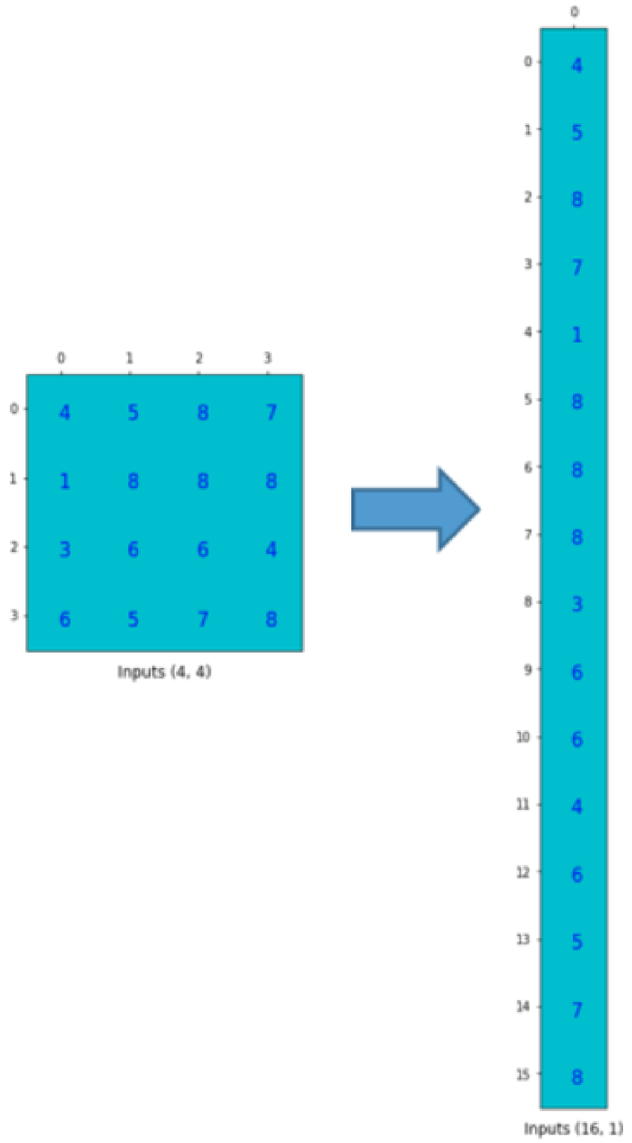
1 4 1
1 4 3
3 3 1
122 148
126 134
4 5 8 7
1 8 8 8
3 6 6 4
6 5 7 8
1 4 1
1 4 3
3 3 1
122 148
126 134
4 5 8 7
1 8 8 8
3 6 6 4
6 5 7 8
1 4 1
1 4 3
3 3 1
122 148
126 134
4 5 8 7
1 8 8 8
3 6 6 4
6 5 7 8
1 4 1
1 4 3
3 3 1
122 148
126 134
4 5 8 7
1 8 8 8
3 6 6 4
6 5 7 8



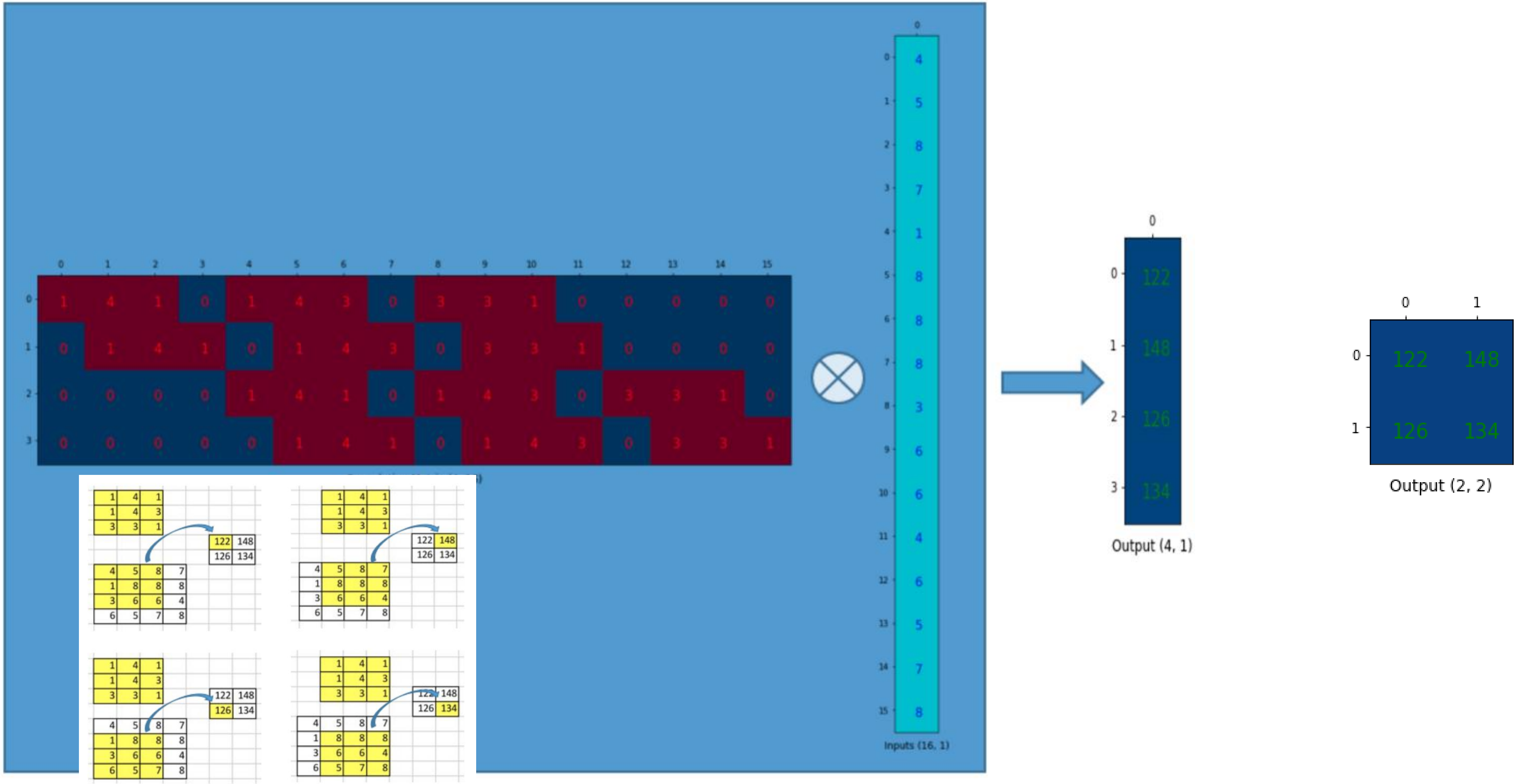
Kernel (3, 3)



Convolution Matrix (4, 16)

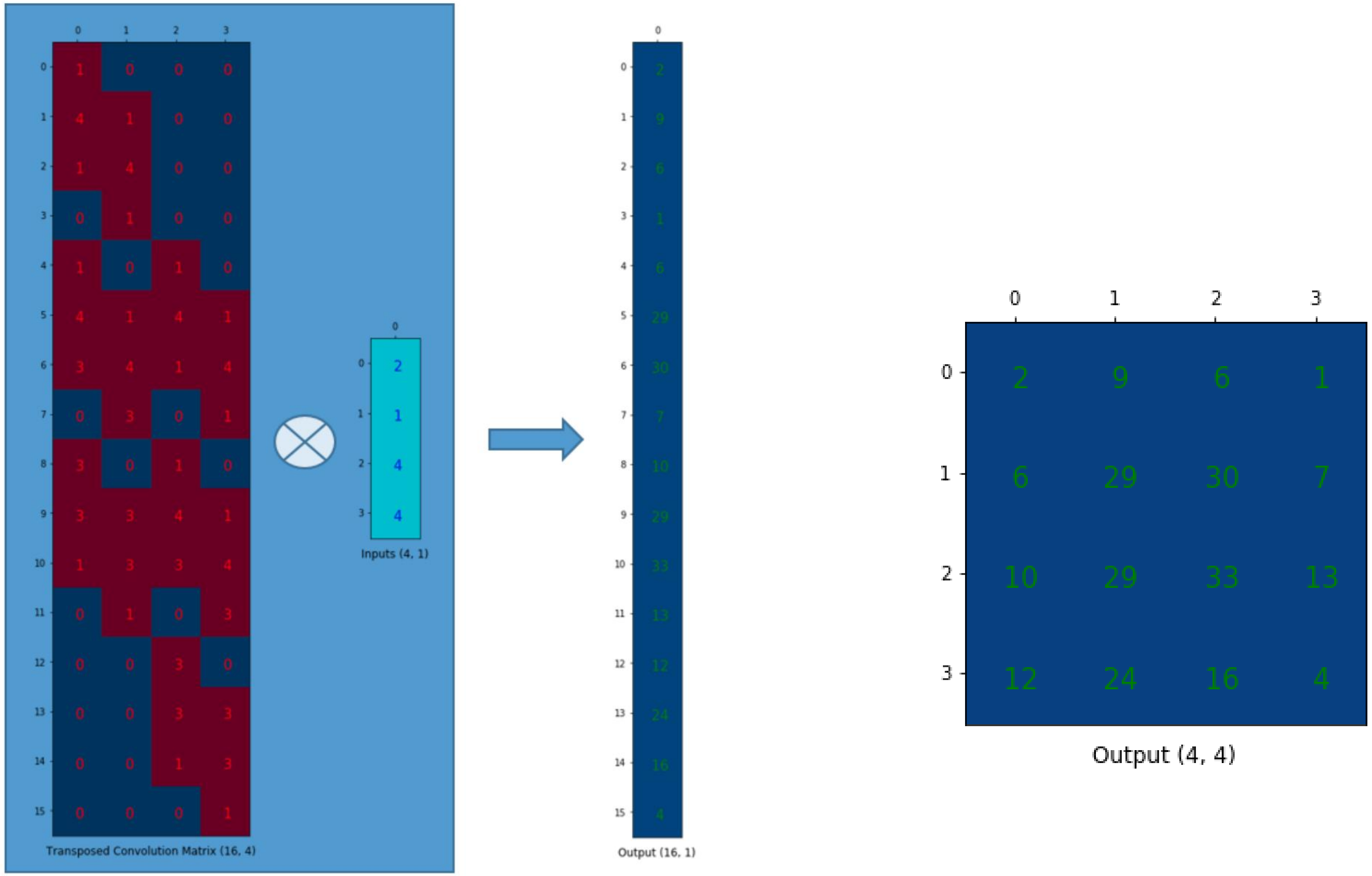


Inputs (4, 4)
Inputs (16, 1)

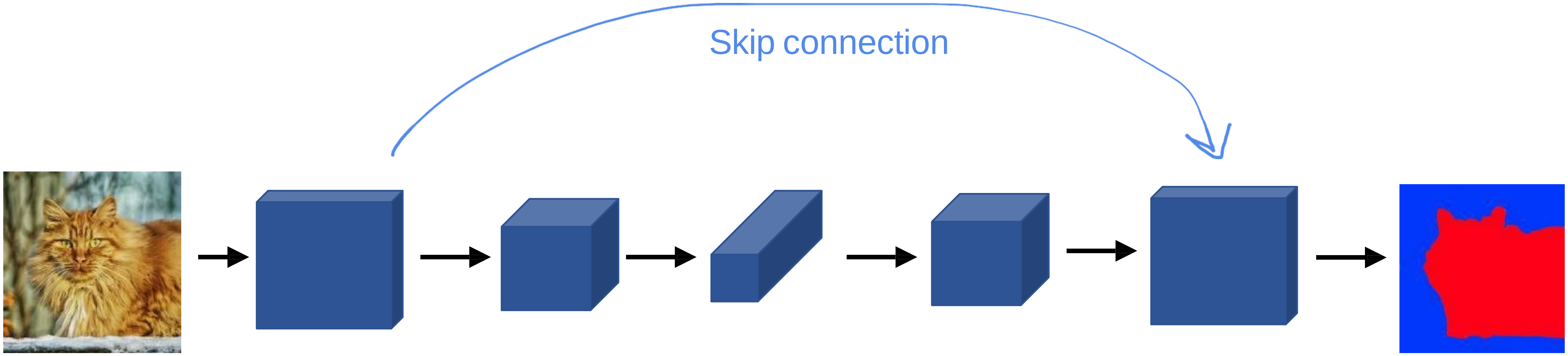


Inputs (16, 1)
Output (4, 1)
122
148
126
134
Output (2, 2)

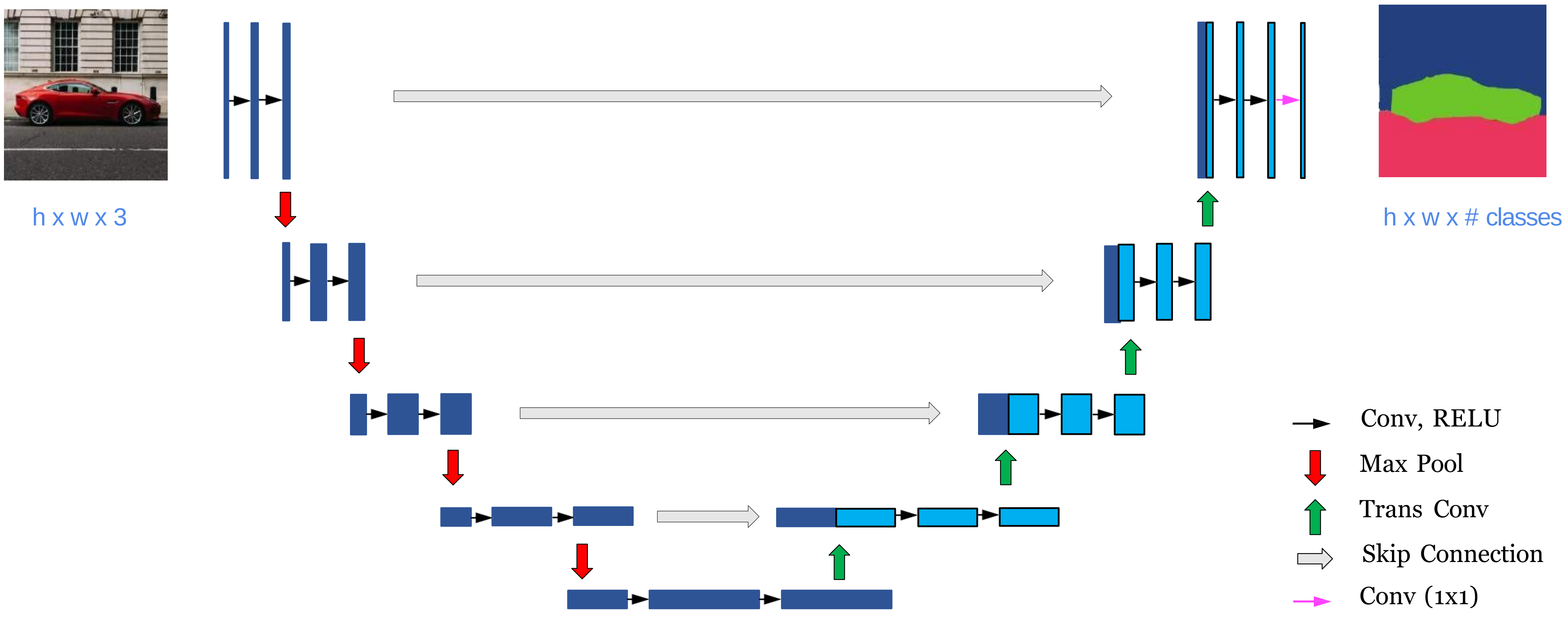
Transposed Convolution Matrix



UNET Motivation

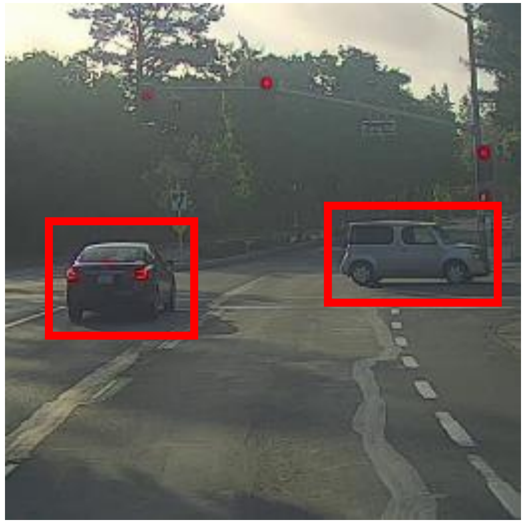


U-Net



[Ronneberger et al., 2015, U-Net: Convolutional Networks for Biomedical Image Segmentation]

Car detection example

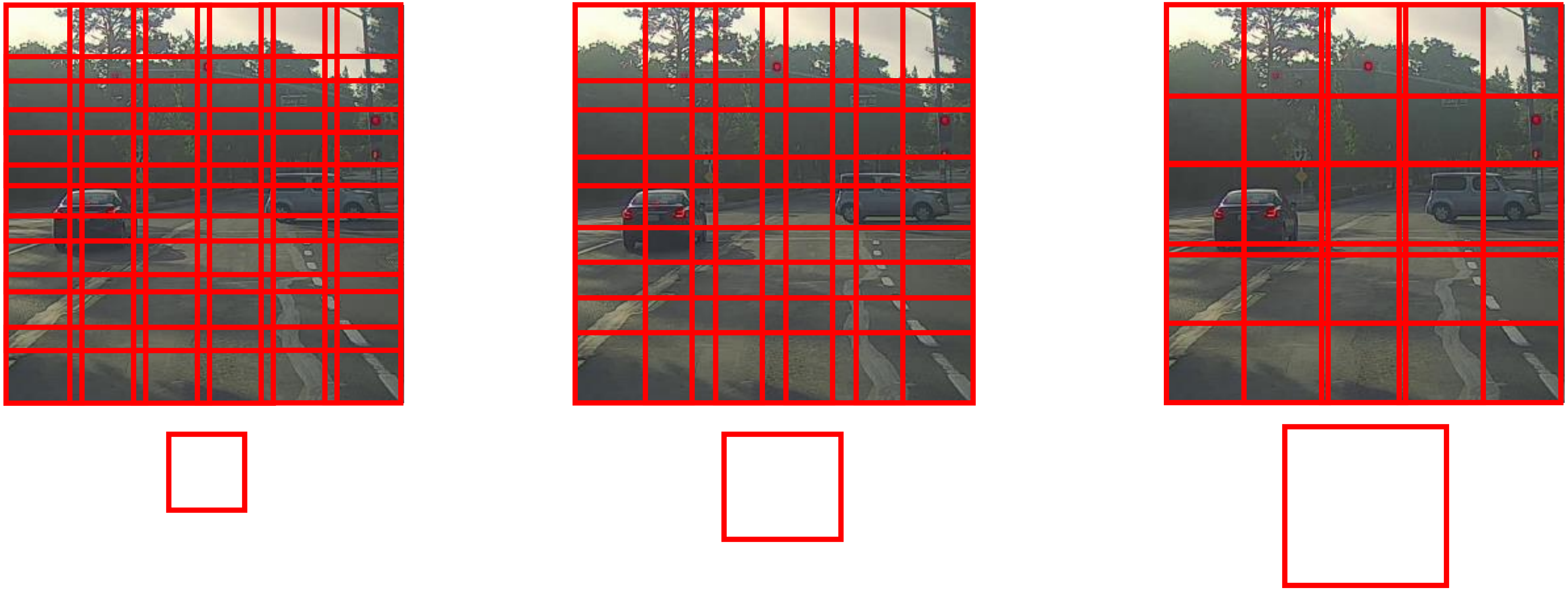


Training set:

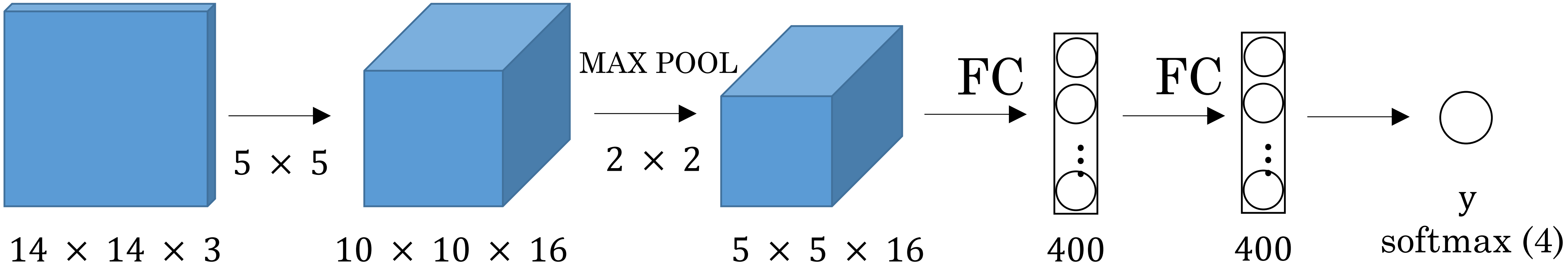
x	y
	1
	1
	1
	0
	0

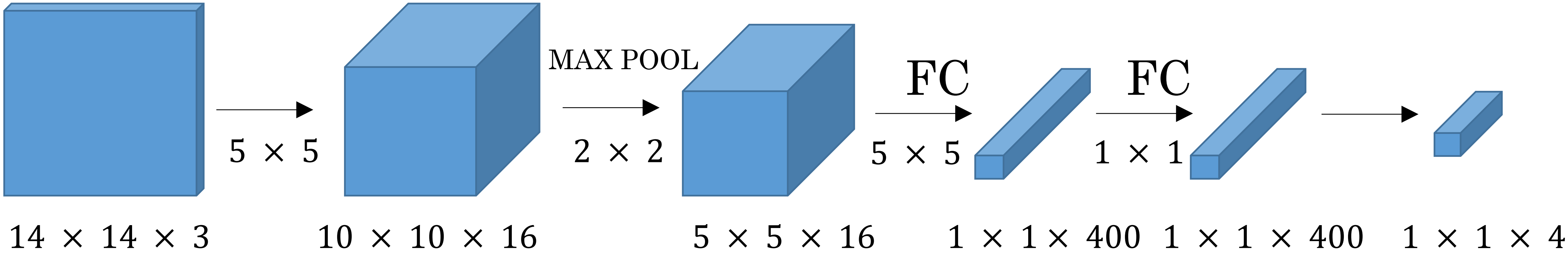
 → ConvNet → y

Sliding windows detection

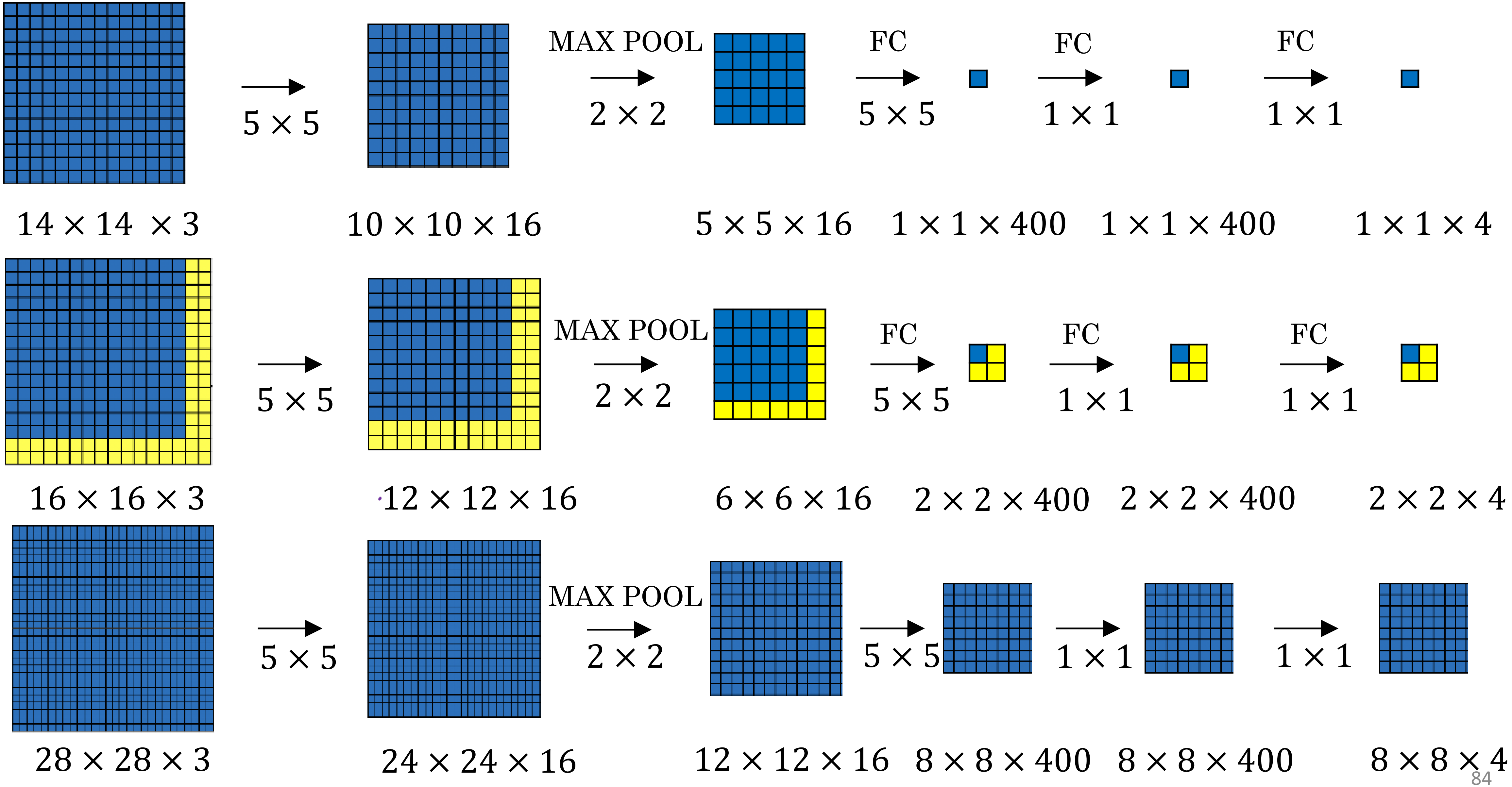


Turning FC layer into convolutional layers



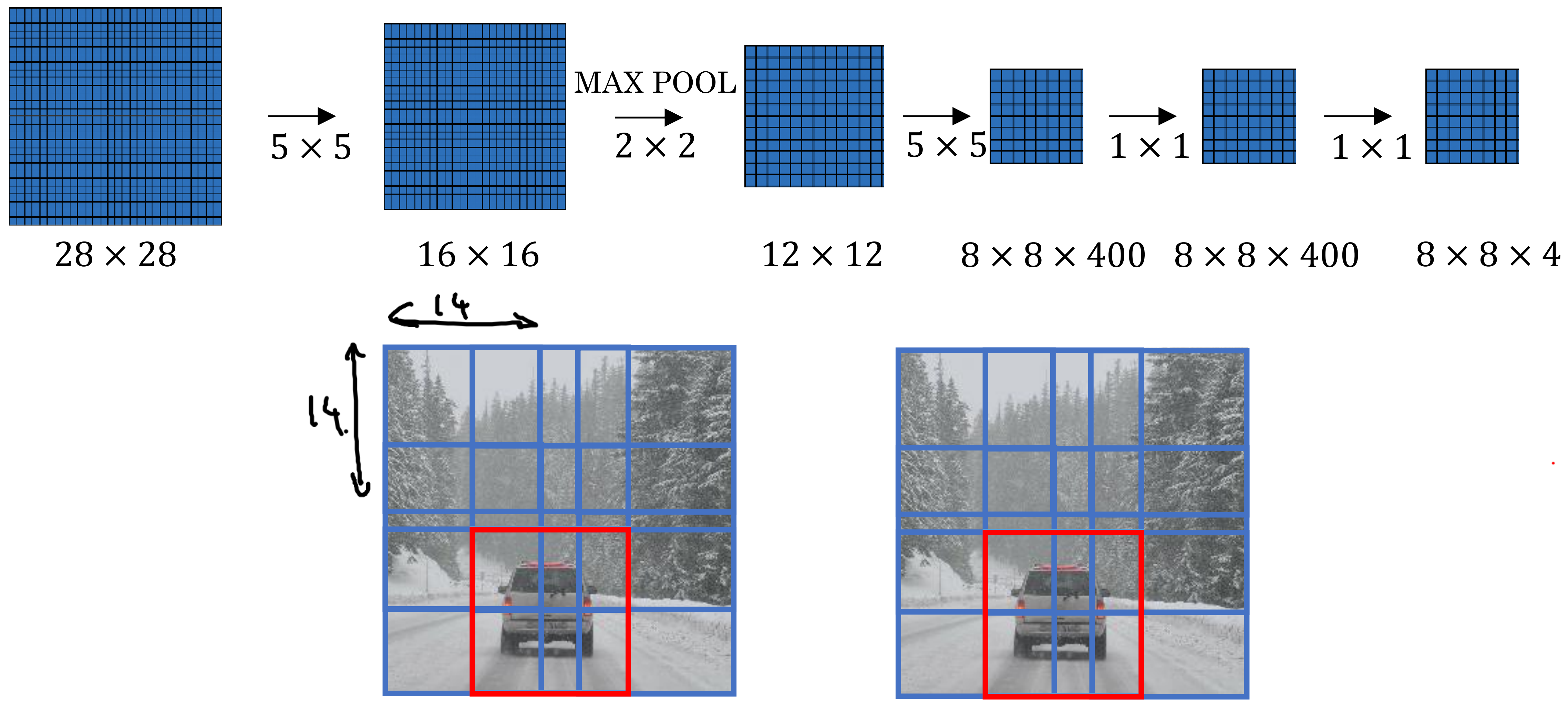


Convolution implementation of sliding windows

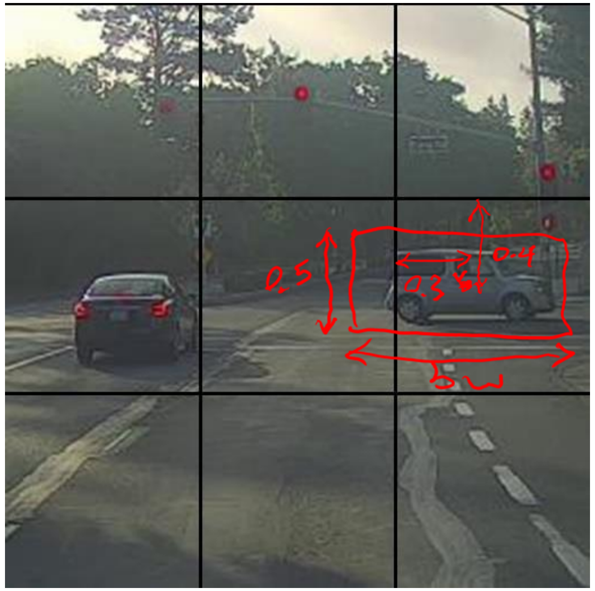


[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

Convolution implementation of sliding windows



Specify the bounding boxes

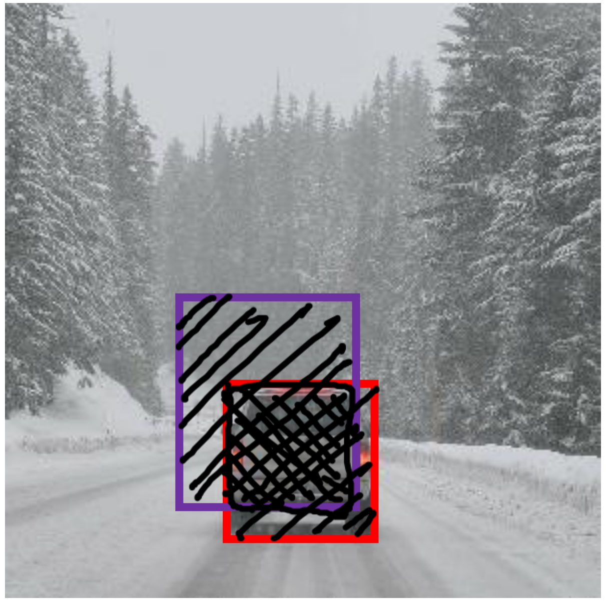


$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$b_x = 0.3$
$b_y = 0.4$
$b_h = 0.5$
$b_w = 1.1$

Redmon, et al., 2015, You Only Look Once, unified real-time object detection

Evaluating object localization



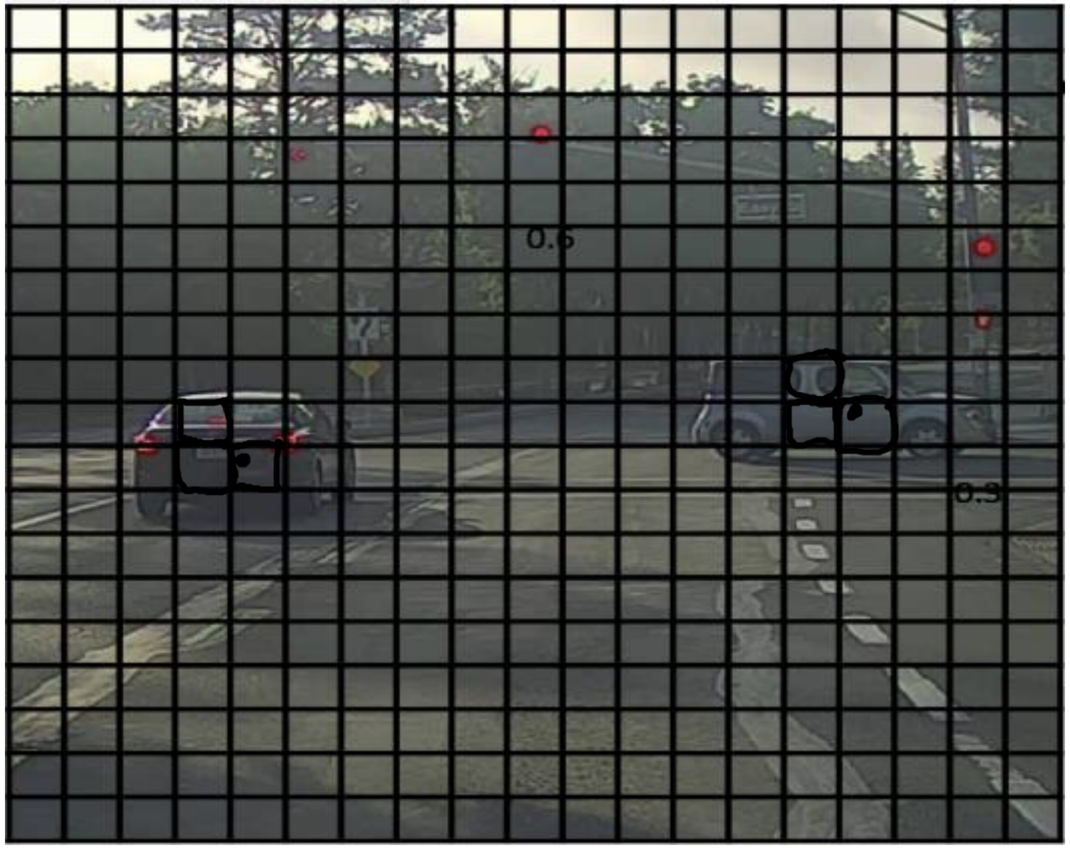
Intersection over Union (IoU)

$$= \frac{\text{size of}}{\text{size of}}$$

“Correct” if IoU $\geq$ 0.5

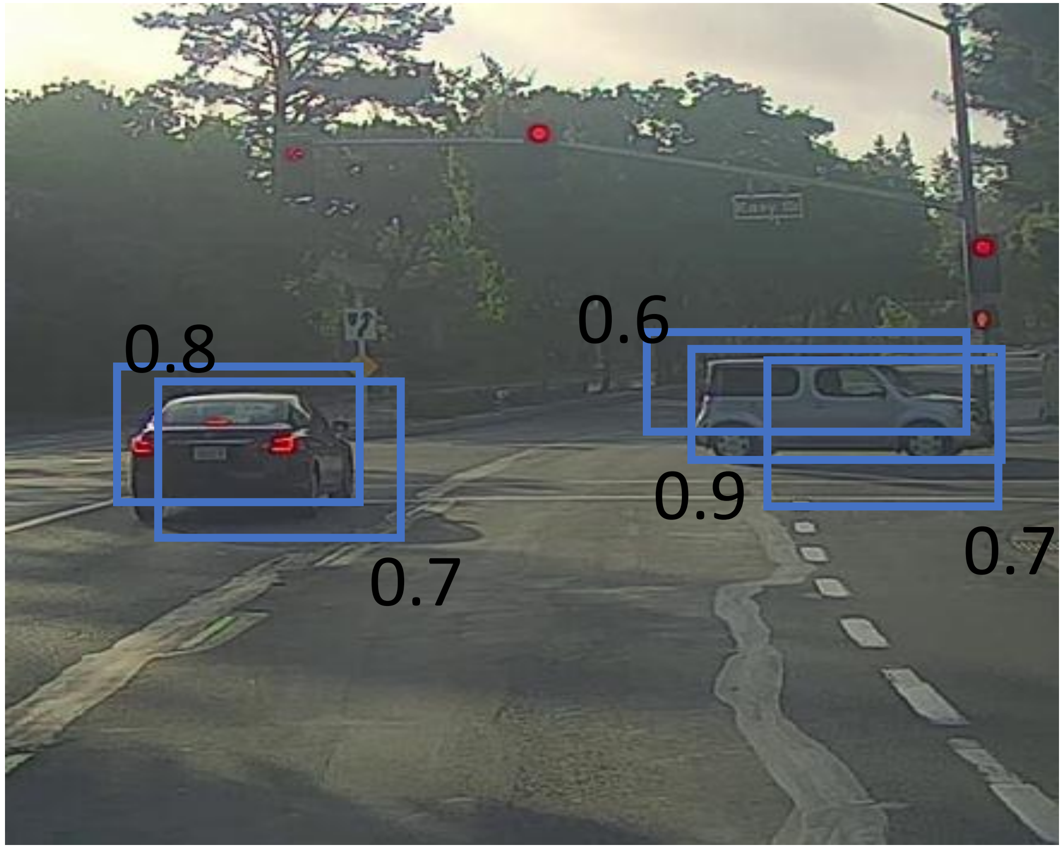
More generally, IoU is a measure of the overlap between two bounding boxes.

Non-max suppression example

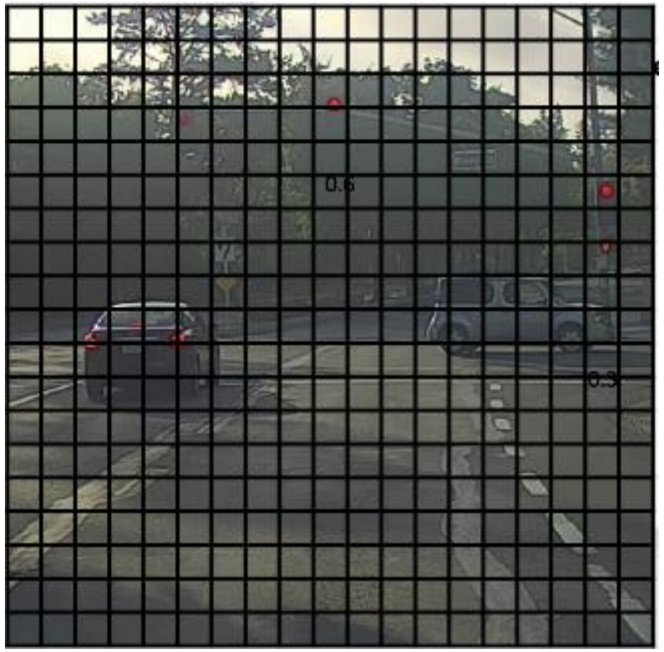


19x19

Non-max suppression example



Non-max suppression algorithm



19× 19

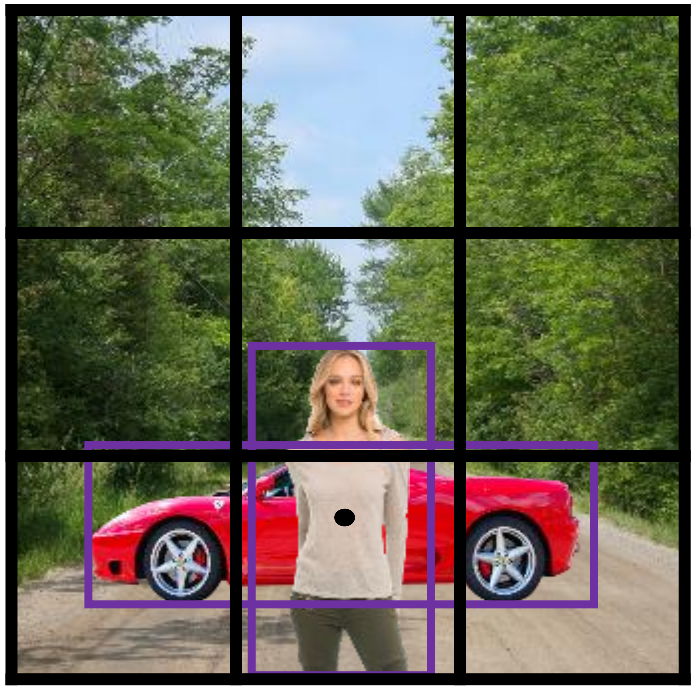
Each output prediction is: $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

Discard all boxes with $p_c \leq 0.6$

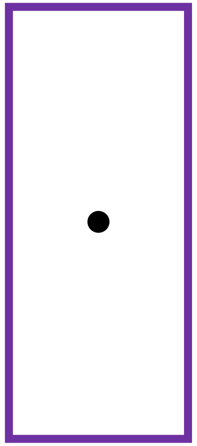
While there are any remaining boxes:

- Pick the box with the largest p_c
  Output that as a prediction.
- Discard any remaining box with
  IoU ≥ 0.5 with the box output
  in the previous step

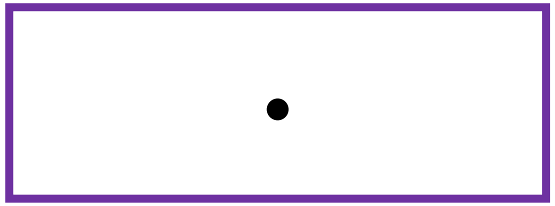
Overlapping objects:



Anchor box 1:



Anchor box 2:



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

[Redmon et al., 2015, You Only Look Once: Unified

Anchor box algorithm

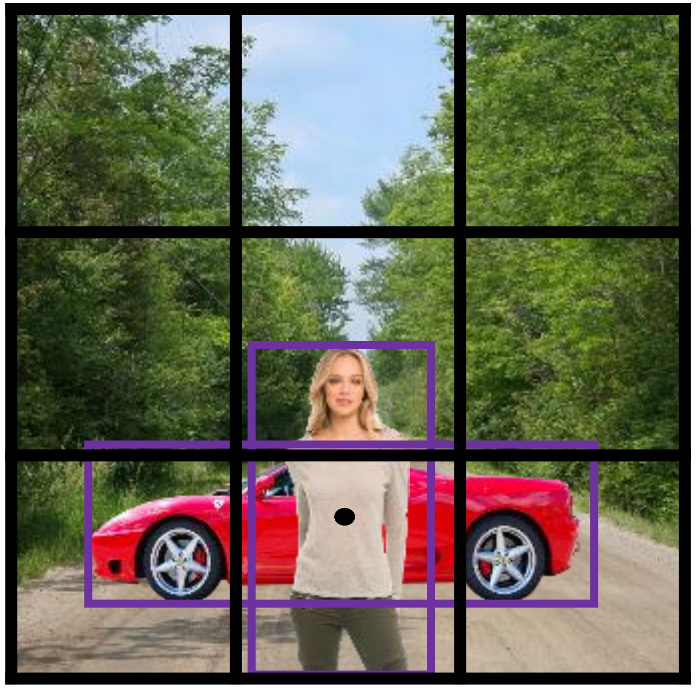
Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint.

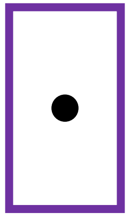
With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

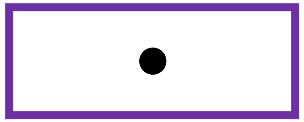
Anchor box example



Anchor box 1:



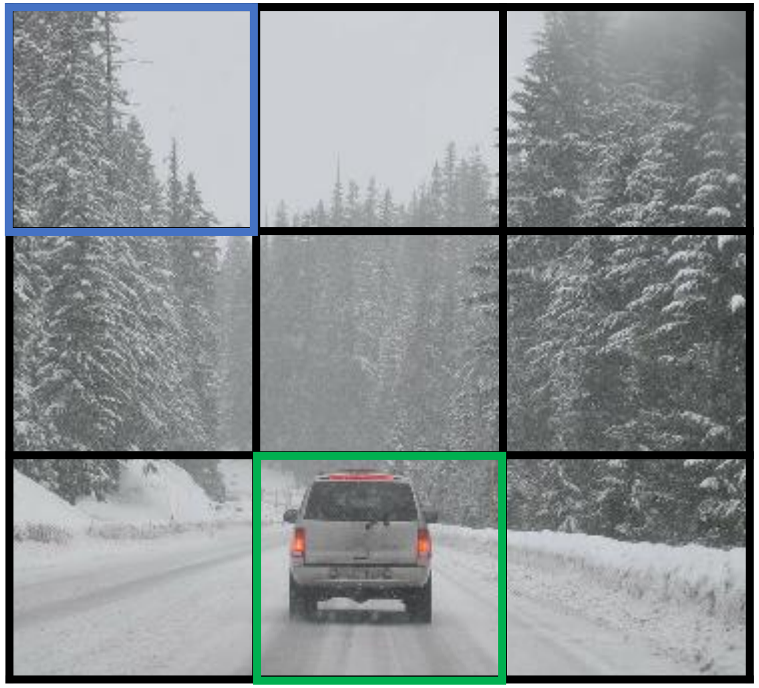
Anchor box 2:



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Please note: In the original yolo paper, only one class set is assigned to a cell

Training



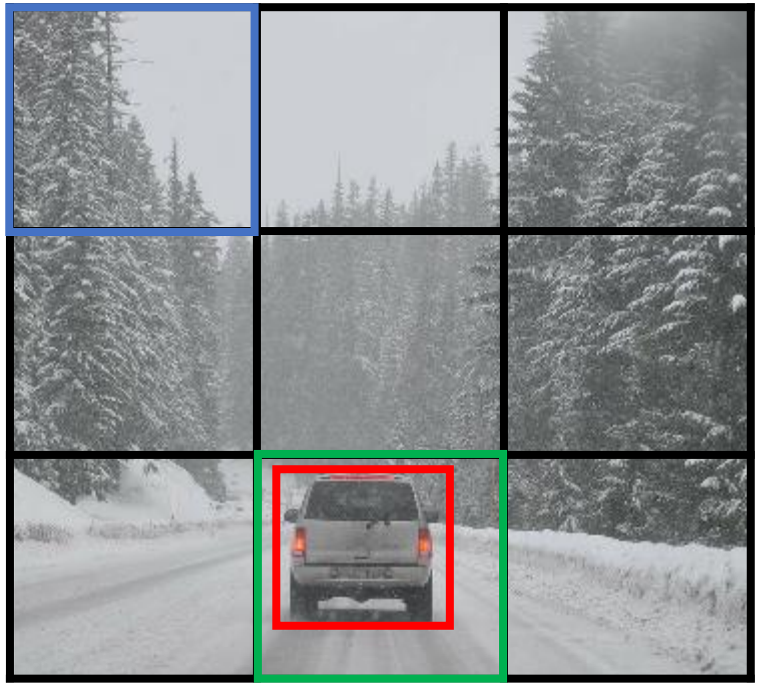
1 - pedestrian
2 - car
3 - motorcycle

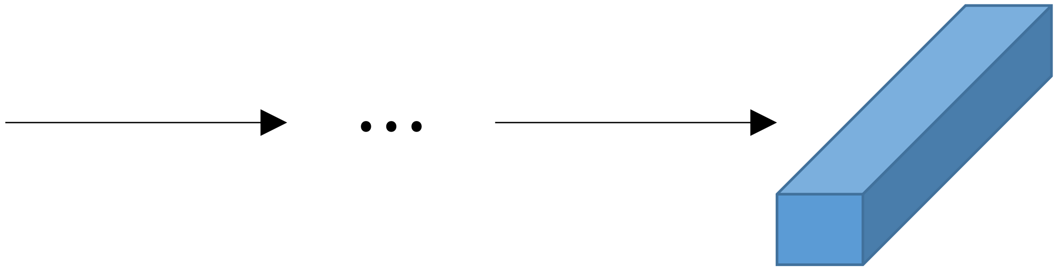
$$
y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \qquad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}
$$

y is $3 \times 3 \times 2 \times 8$

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Making predictions





$3 \times 3 \times 2 \times 8$

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Yolo loss function

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj}[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj}[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj}(C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj}(C_i - \hat{C}_i)^2$$

$$+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

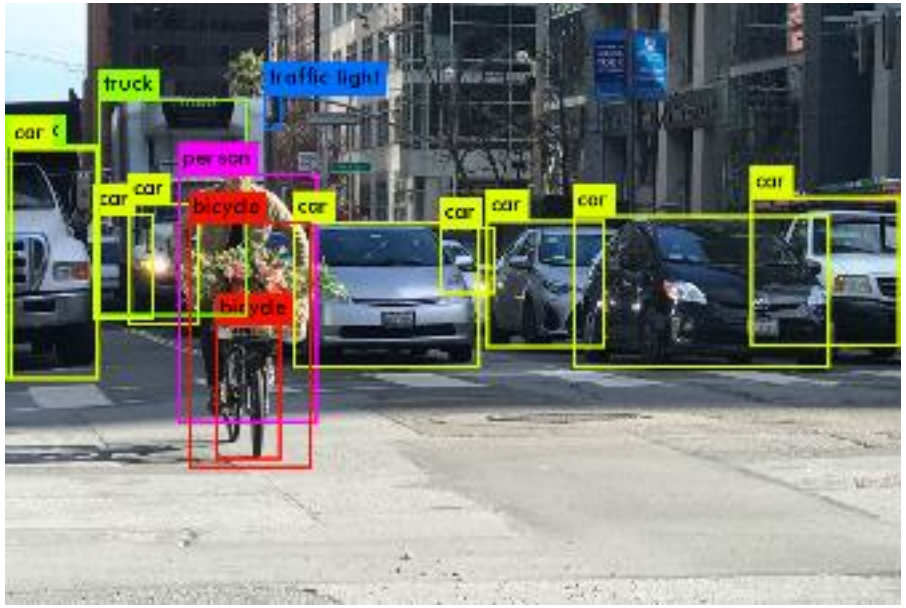
$\lambda_{coord} = 5$

$\lambda_{noobj} = 0.5$

- 1_i^{obj} is 1 when there is an object (precisely, object's center) in cell i and 0 elsewhere.
- 1_{ij}^{obj} is 1 if the jth predictor has the highest IoU with the ground truth "is responsible", else it is 0.

  1_{ij}^{noobj} $is\ 1\ when\ there\ is\ no\ object\ in\ the\ cell\ i\ and\ jth\ predictor\ has\ confidence\ more\ than\ the\ threshold$

Outputting the non-max suppressed outputs



- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Face verification vs. face recognition

Verification

- Input image, name/ID
- Output whether the input image is that of the claimed person

Recognition

- Has a database of K persons
- Get an input image
- Output ID if the image is any of the K persons (or "not recognized")

One-shot learning

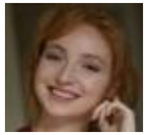
Learning from one example to recognize the person again

Learning a "similarity" function

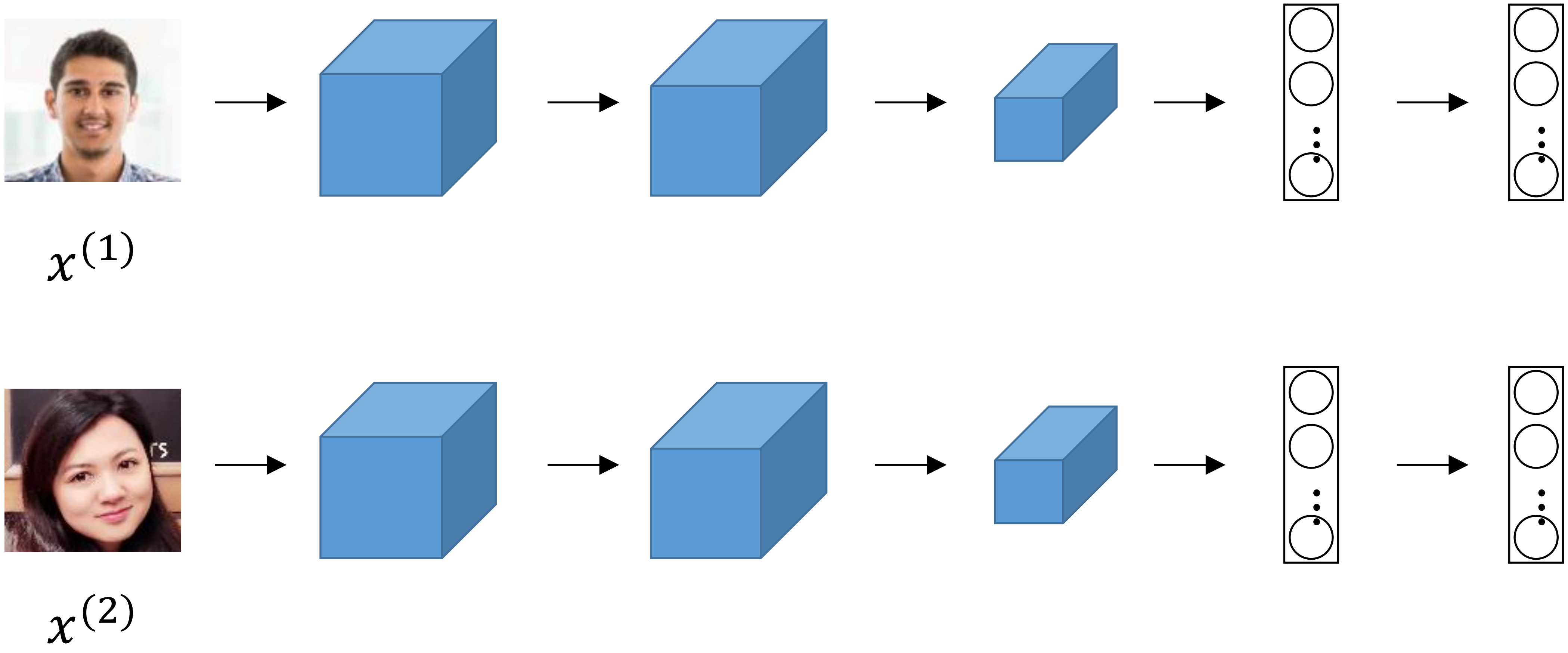
$$\text{d(img1,img2)} = \text{degree of difference between images}$$

$$\text{If d(img1,img2)} \leq \tau \quad \text{same}$$

$$> \tau \quad \text{different}$$

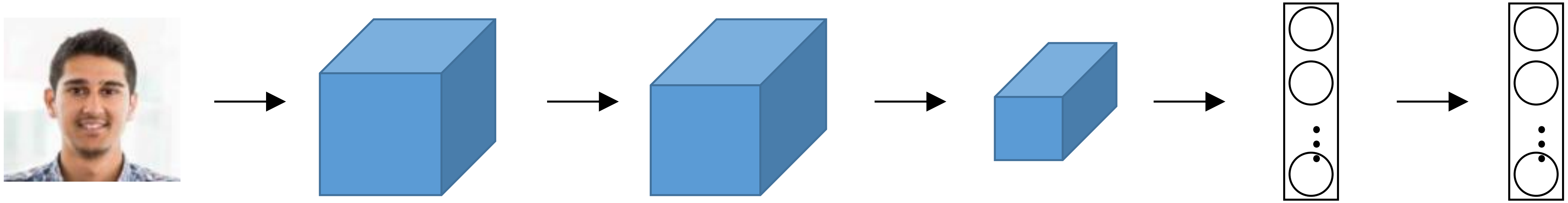


Siamese network



[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Goal of learning



$f(x^{(1)})$

Parameters of NN define an encoding $f(x^{(i)})$

Learn parameters so that:

If $x^{(i)}, x^{(j)}$ are the same person, $\left\|f(x^{(i)}) - f(x^{(j)})\right\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\left\|f(x^{(i)}) - f(x^{(j)})\right\|^2$ is large.

Contrastive loss

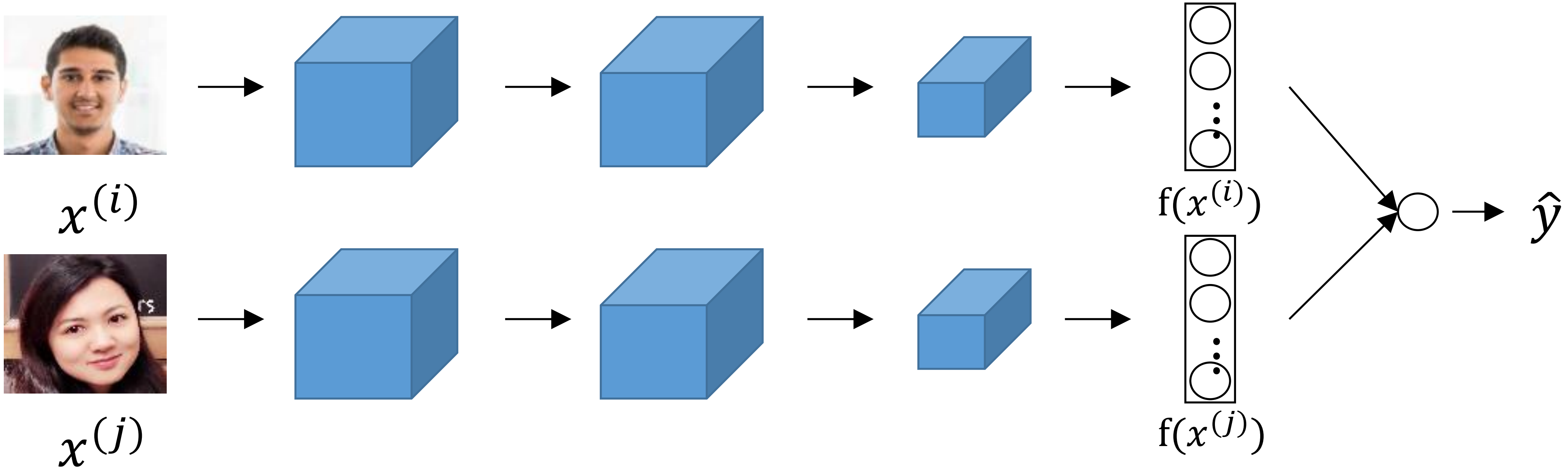
If $x^{(i)}, x^{(j)}$ are the same person, $\left\|f\left(x^{(i)}\right) - f\left(x^{(j)}\right)\right\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\left\|f\left(x^{(i)}\right) - f\left(x^{(j)}\right)\right\|^2$ is large.

$$(Y)\boxed{(D_W)^2} + (1-Y)\boxed{\{max(0, m - D_W)\}^2}$$

Y is 0 for dissimilar pairs and 1 for similar pairs.

Learning the similarity function



$$d(f_1, f_2) = \sum_i \alpha_i |f_1[i] - f_2[i]|$$

$$\chi^2(f_1, f_2) = \sum_i w_i (f_1[i] - f_2[i])^2 / (f_1[i] + f_2[i])$$

[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Face verification supervised learning

x y

  1

  0

  0

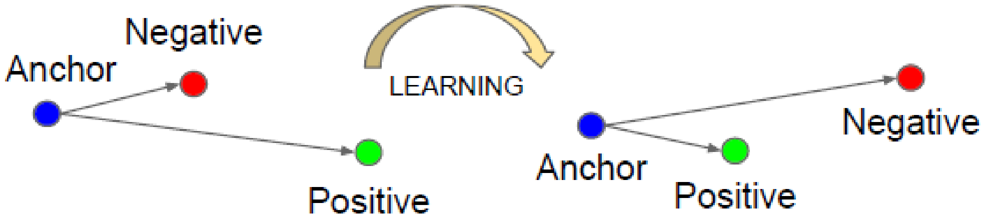
  1

[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Triplet loss


Anchor Positive


Anchor Negative



$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2$$

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+$$

[Schroff et al.,2015, FaceNet: A unified embedding for face recognition and clustering]

Choosing the triplets A,P,N

During training, if A,P,N are chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is easily satisfied.

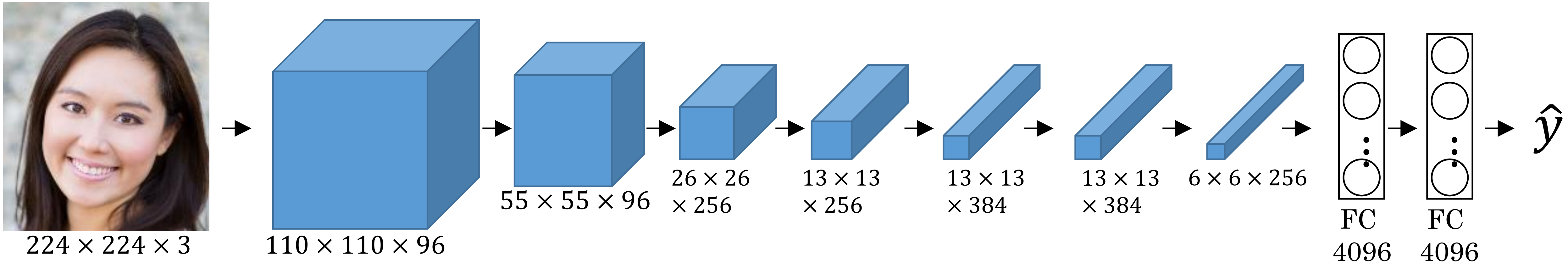
Choose triplets that're "hard" to train on.

[Schroff et al.,2015, FaceNet: A unified embedding for face recognition and clustering]

Training set using triplet loss

Anchor	Positive	Negative
		
		
$\vdots$	$\vdots$	$\vdots$
		

Visualizing what a deep network is learning



Pick a unit in layer 1. Find the nine image patches that maximize the unit's activation.

Repeat for other units.

[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

Visualizing deep layers: Layer 1

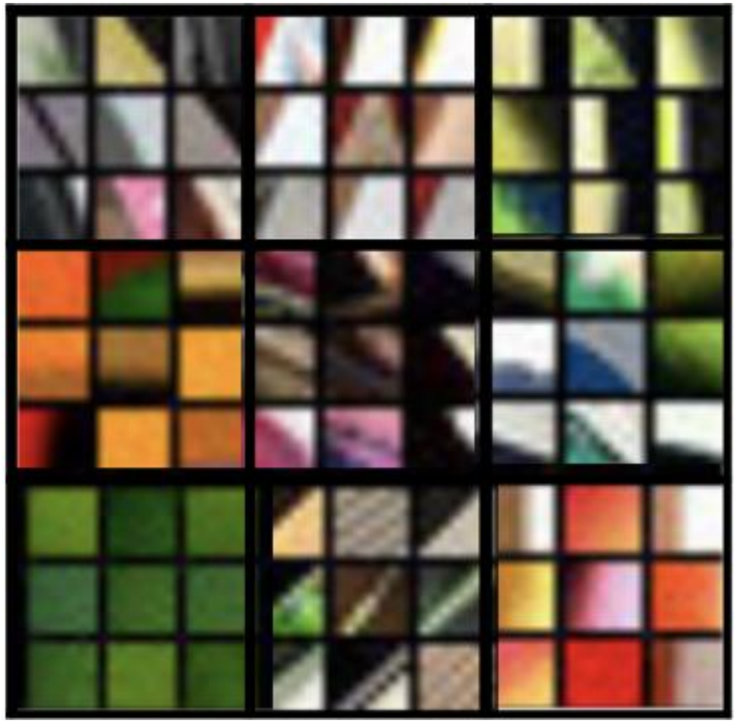

Layer 1


Layer 2


Layer 3


Layer 4


Layer 5



Visualizing deep layers: Layer 2

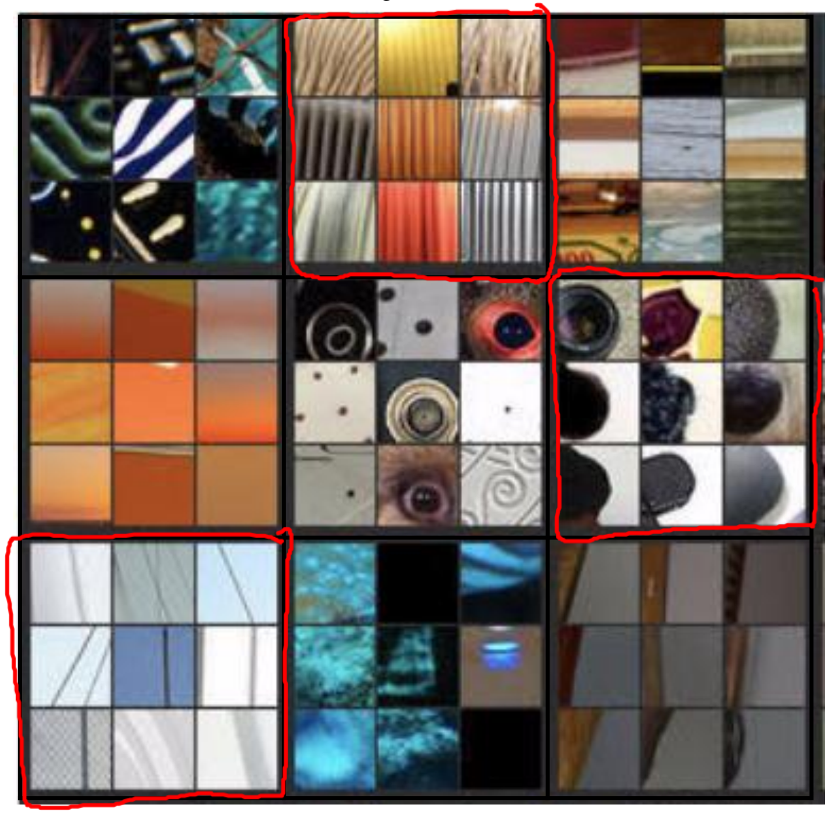
Layer 1

Layer 2

Layer 3

Layer 4

Layer 5



Visualizing deep layers: Layer 3

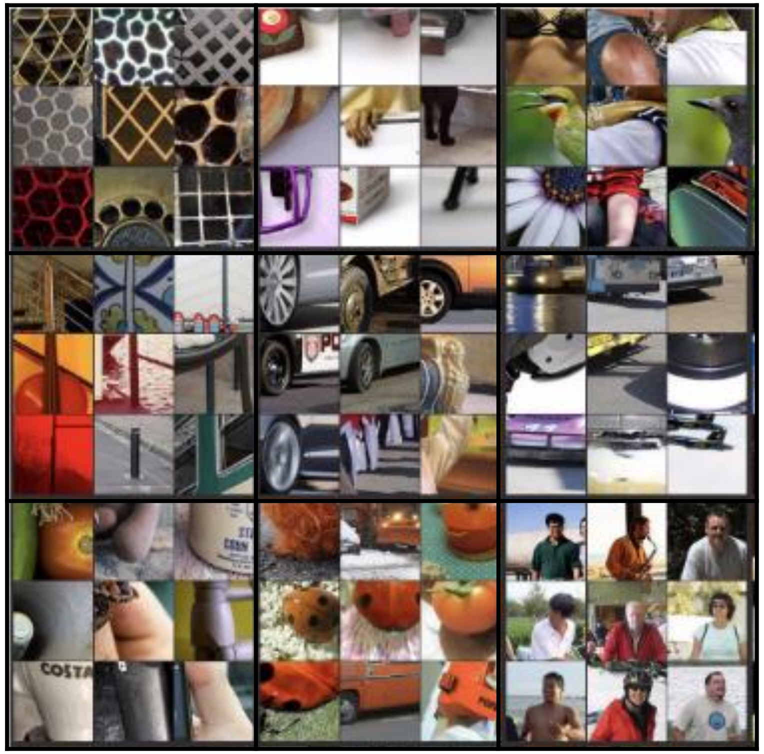
Layer 1

Layer 2

Layer 3

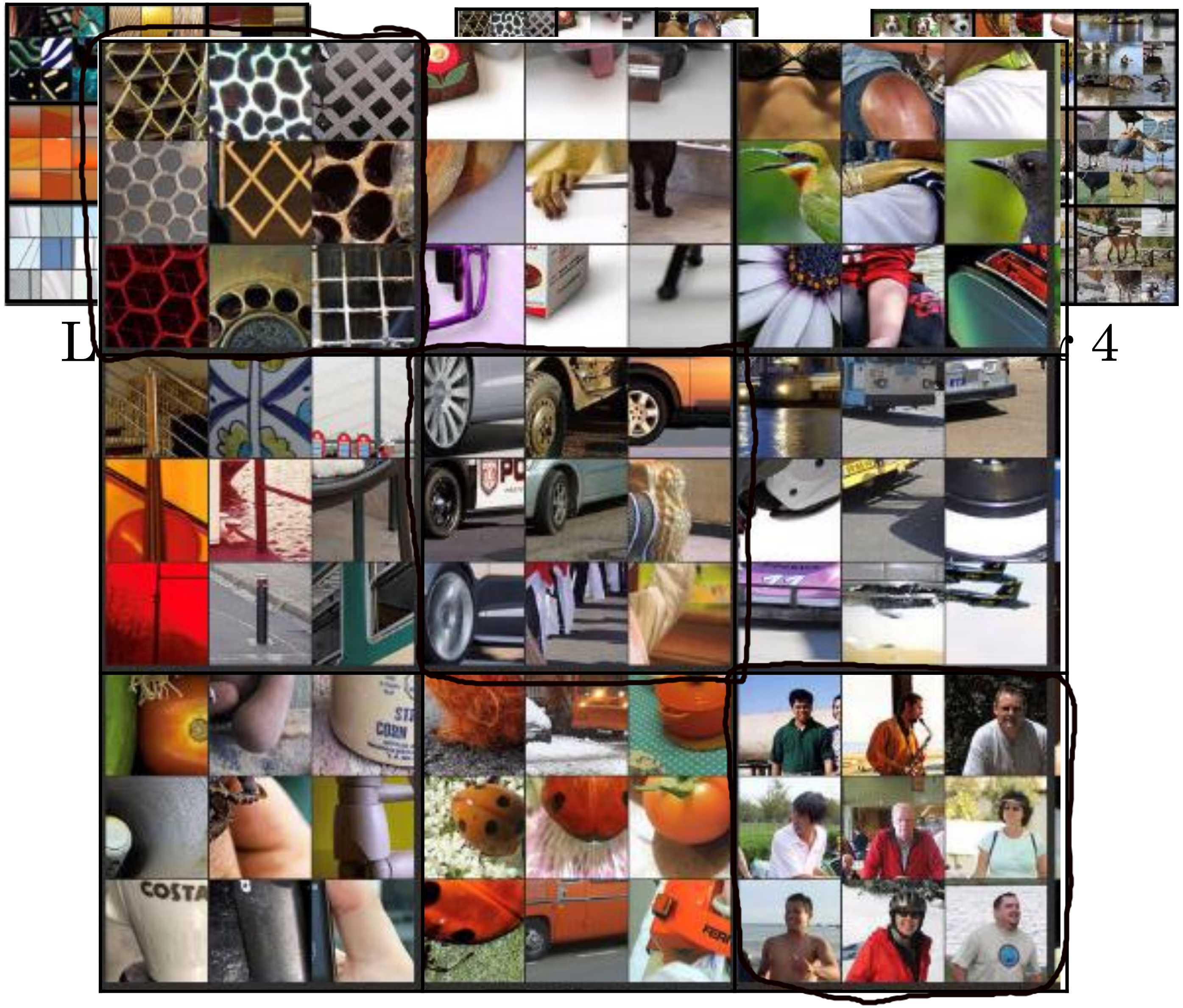
Layer 4

Layer 5



Visualizing deep layers: Layer 3

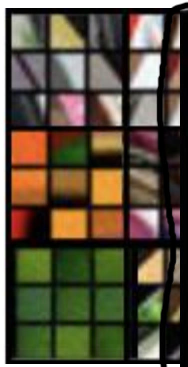
Layer 1



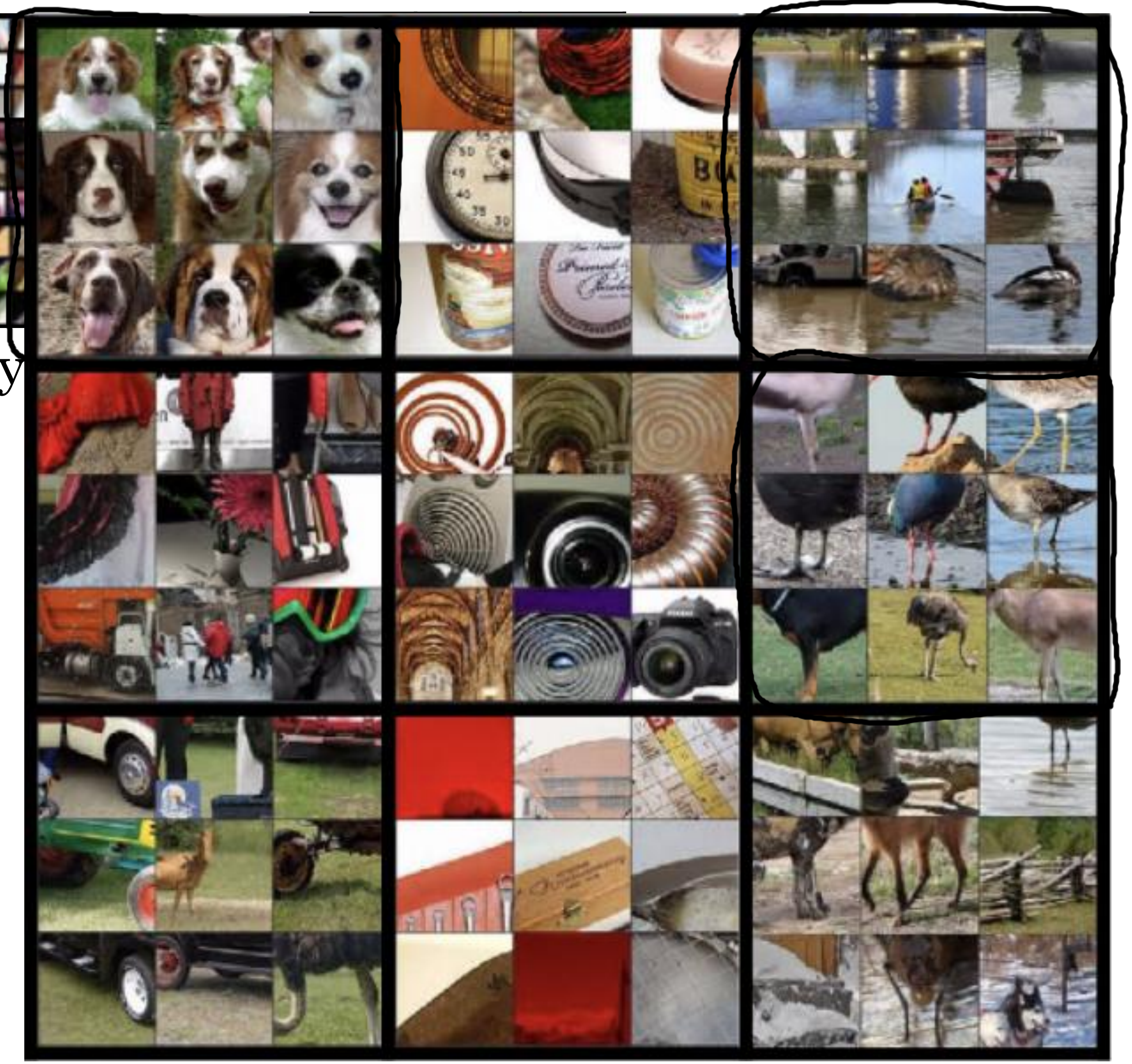
L 4

Layer 5

Visualizing deep layers: Layer 4



Lay

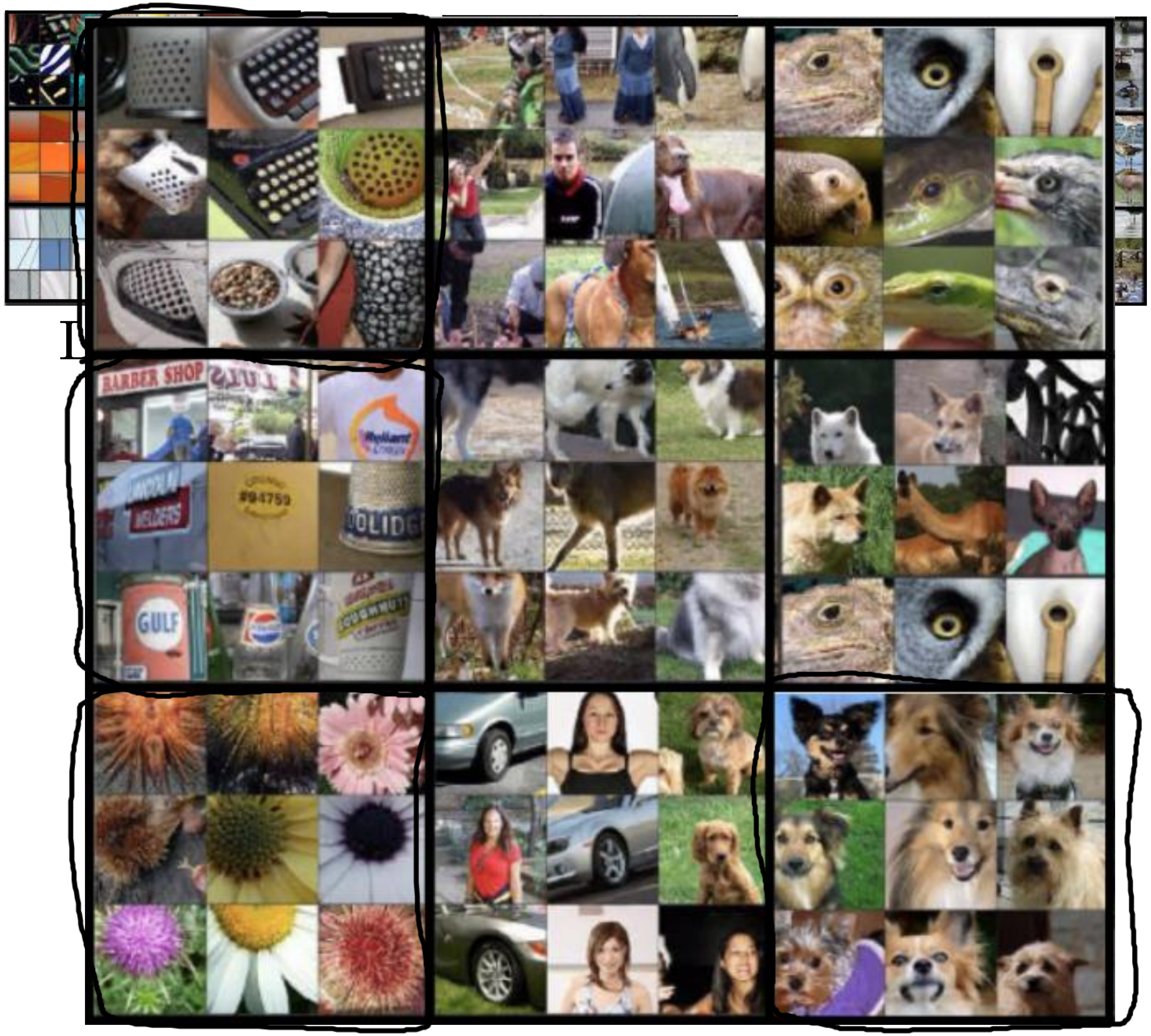


Layer 4

Layer 5

Visualizing deep layers: Layer 5

Layer 1



L

Layer 5

Neural Style Transfer

Content

Style

Generated image

Neural style transfer cost function

Content C

Style S

Generated image G

[Gatys et al., 2015. A neural algorithm of artistic style. Images on slide generated by Justin Johnson]

Find the generated image G

1. Initiate G randomly

   G: $100 \times 100 \times 3$

2. Use gradient descent to minimize $J(G)$

$$G = G - \frac{\delta J(G)}{\delta G}$$

[Gatys et al., 2015. A neural algorithm of artistic style]

Content cost function

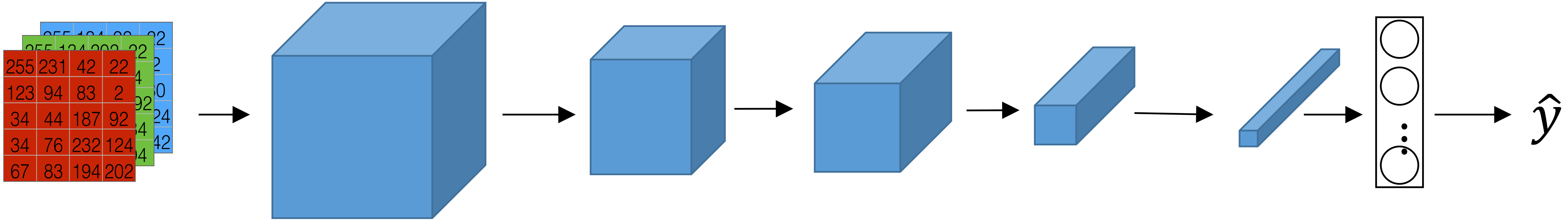
$$J(G) = \alpha\, J_{content}(C,G) + \beta\, J_{style}(S,G)$$

- Say you use hidden layer l to compute content cost.
- Use pre-trained ConvNet. (E.g., VGG network)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content

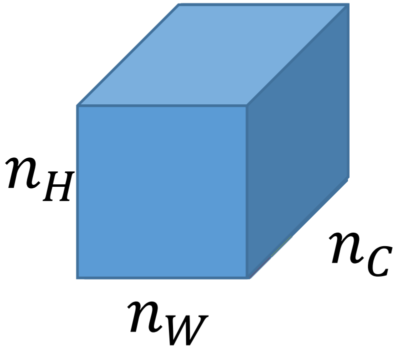
$$\mathrm{Jcontent}(\mathrm{C},\mathrm{G}) = \frac{1}{2} \times \left(\mathrm{a}^{[\mathrm{L}][\mathrm{C}]} - \mathrm{a}^{[\mathrm{L}][\mathrm{G}]}\right)^2$$

[Gatys et al., 2015. A neural algorithm of artistic style]

Meaning of the “style” of an image



Say you are using layer l’s activation to measure “style.”
Define style as correlation between activations across channels.

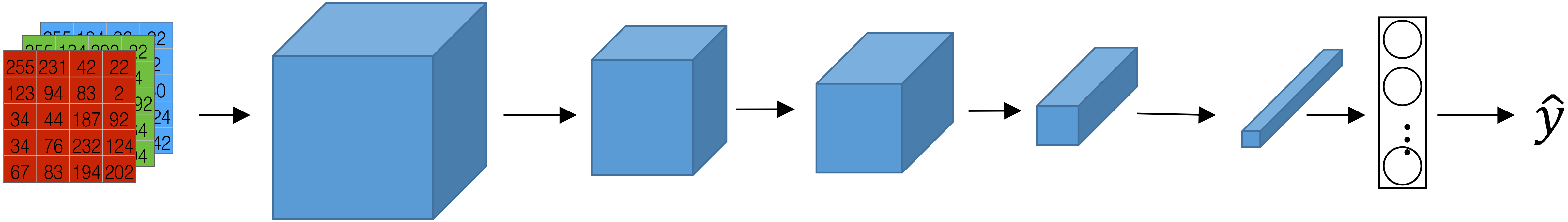


$$\mathrm{Gram}_{KK'}{}^{[L][G]} = \sum_{i=1}^{nh} \sum_{j=1}^{nw} a_{ijk}{}^{[L][G]} \, a_{ijk'}{}^{[L][G]}$$

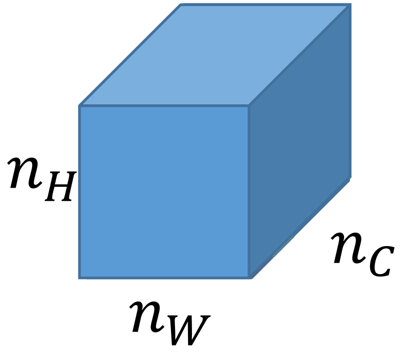
where,
nh = height of gram matrix
nw = width of gram matrix
nc = number of channels in gram matrix

[Gatys et al., 2015. A neural algorithm of artistic style]

Meaning of the "style" of an image



Say you are using layer l's activation to measure "style."
Define style as correlation between activations across channels.

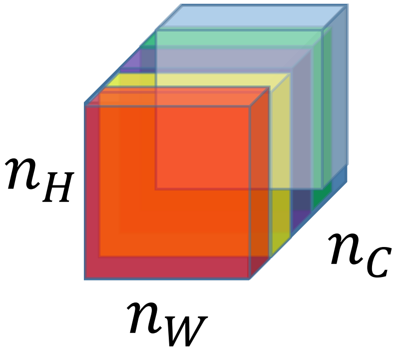


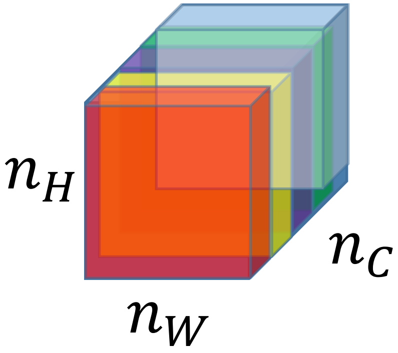
How correlated are the activations across different channels?

[Gatys et al., 2015. A neural algorithm of artistic style]

Intuition about style of an image

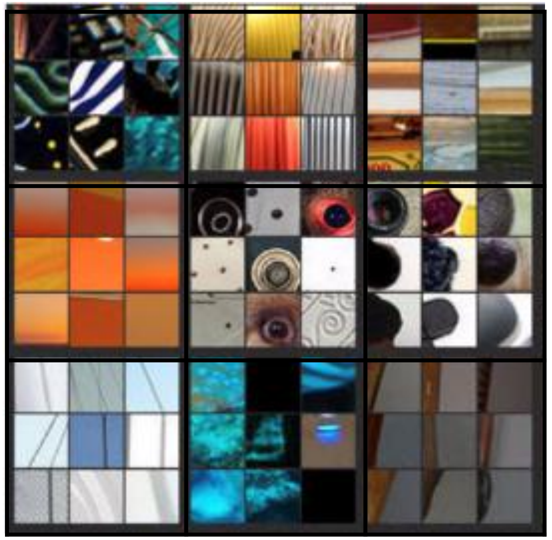
Style image **Generated Image**





$$\text{Gram}_{KK'}^{[L][S]} = \sum_{i=1}^{nh} \sum_{j=1}^{nw} a_{ijk}^{[L][G]}\, a_{ijk'}^{[L][G]}$$

where,
nh = height of gram matrix
nw = width of gram matrix
nc = number of channels in gram matrix



$$\text{Jstyle}(S,G) = \frac{1}{(2 \times nh \times nw \times nc)^2} \times \sum_{K} \sum_{K'} \left(\text{Gram}^{[L][S]} - \text{Gram}^{[L][G]}\right)^2$$

[Gatys et al., 2015. A neural algorithm of artistic style]

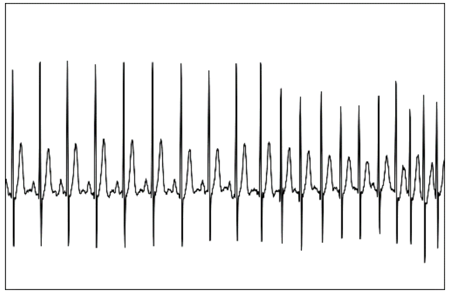
Convolutions in 2D and 1D

2D input image

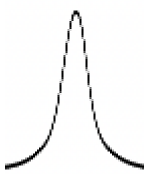
$14 \times 14 \times 3 \quad *$

2D filter

$5 \times 5 \times 3$ → $10 \times 10 \times 16$ $\xrightarrow{5 \times 5 \times 16}$ $6 \times 6 \times 32$



$*$



1	20	15	3	18	12	4	17

1	3	10	3	1

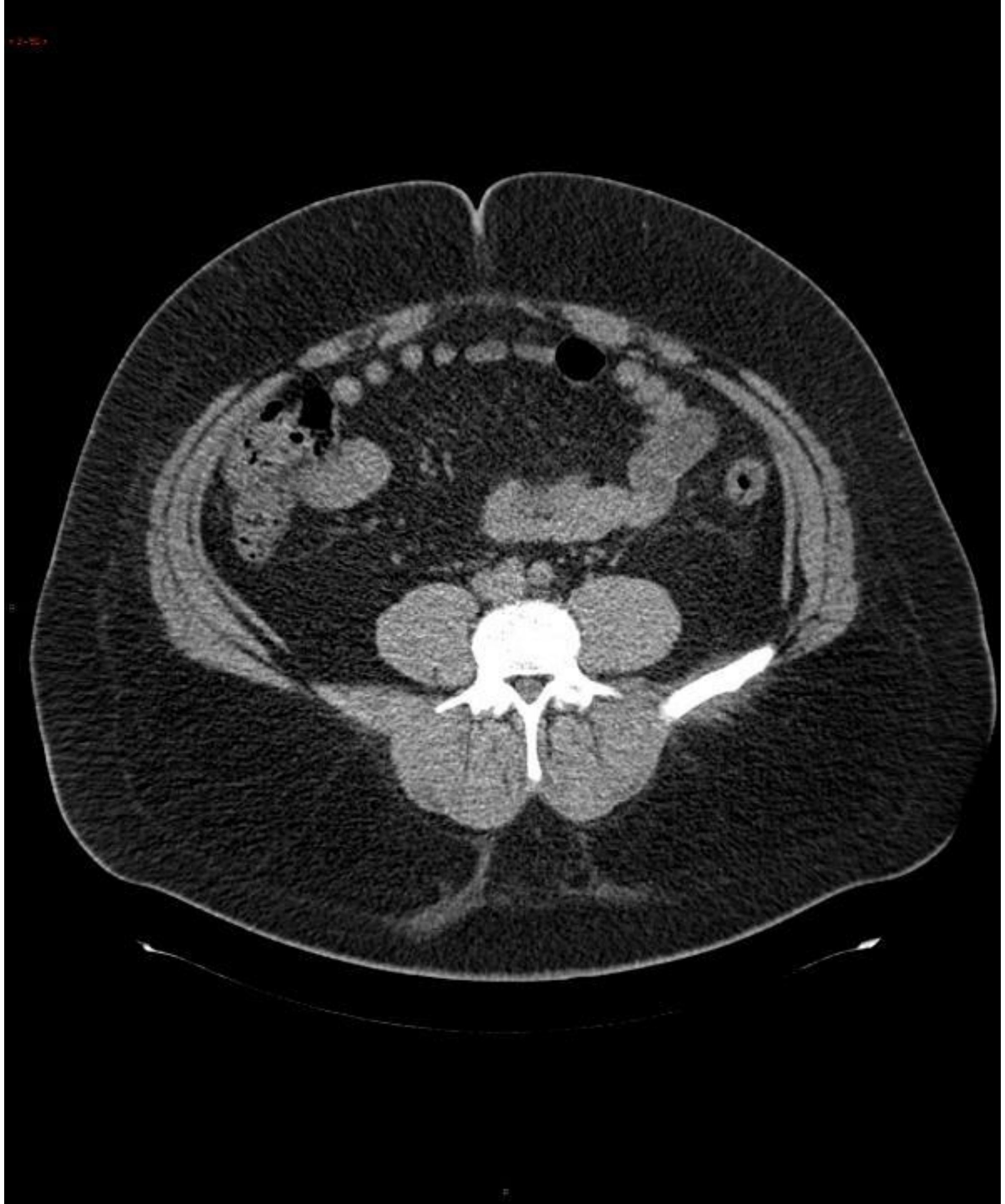
1D input data

$14 \times 1 \quad *$

1D filter

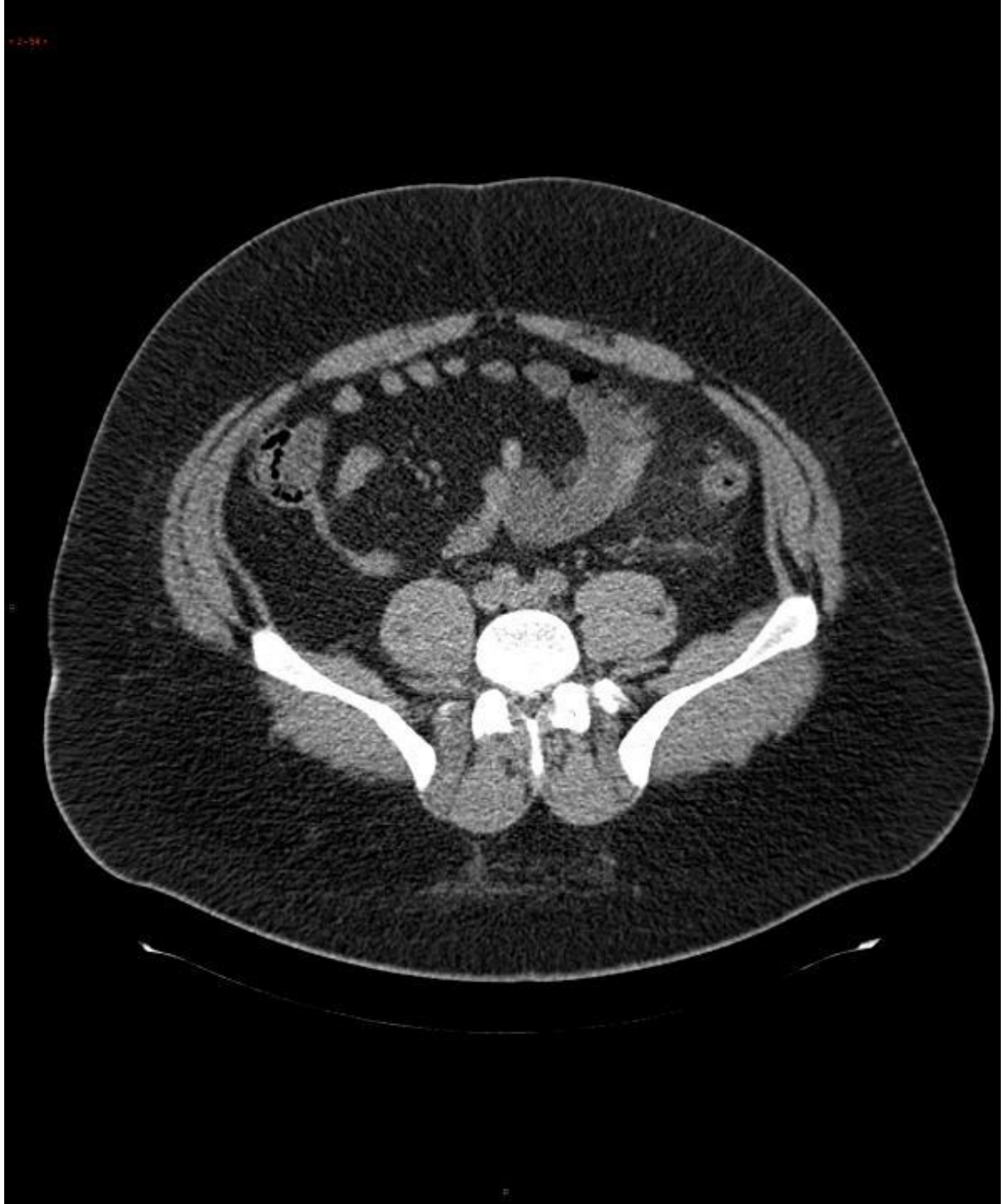
5×1 → 10×16 $\xrightarrow{5 \times 16}$ 6×32

3D data



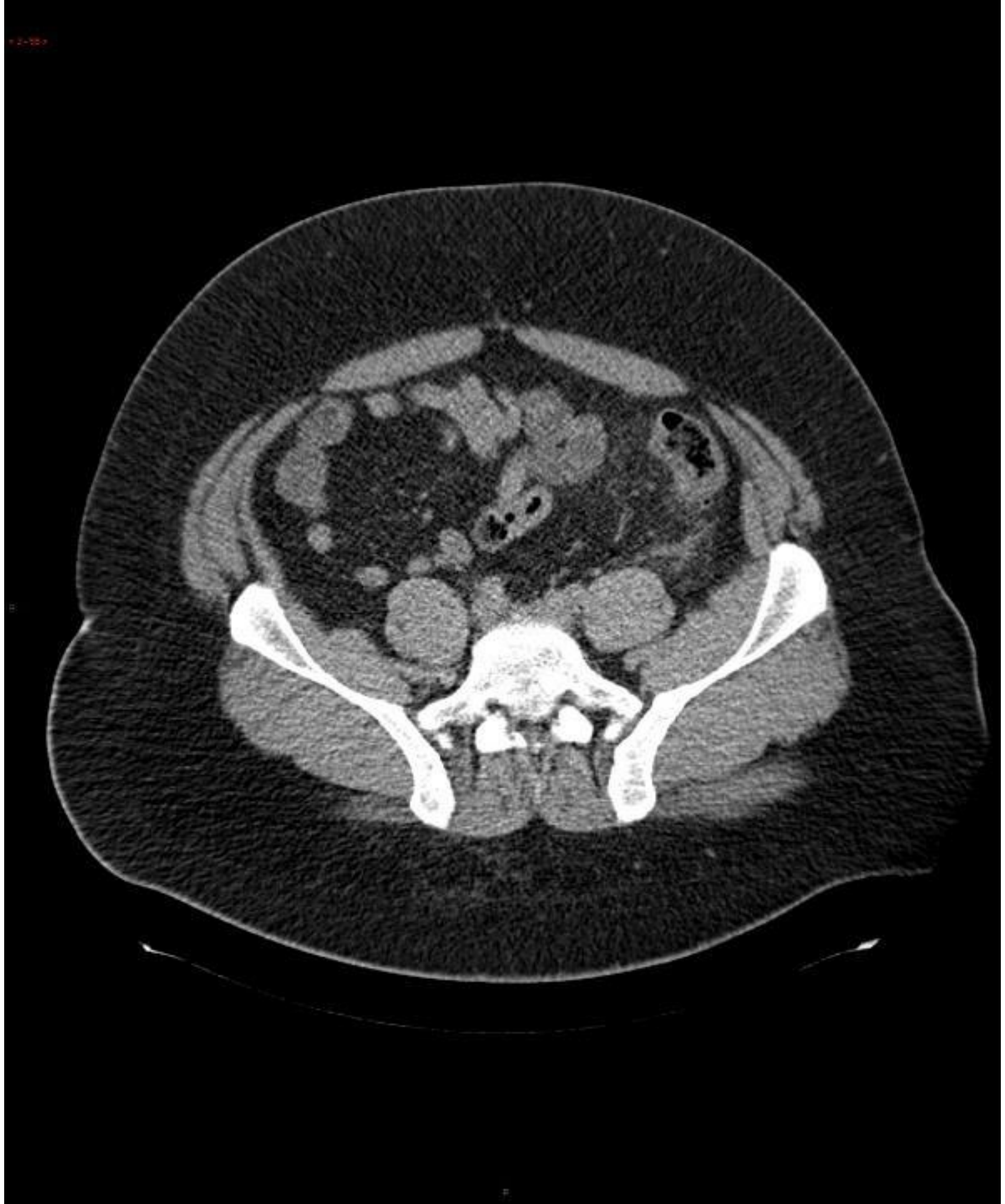
3D data

3D data



3D data

3D data



3D data

3D data

3D data

3D data

3D data

3D data

Convolutions in 3D

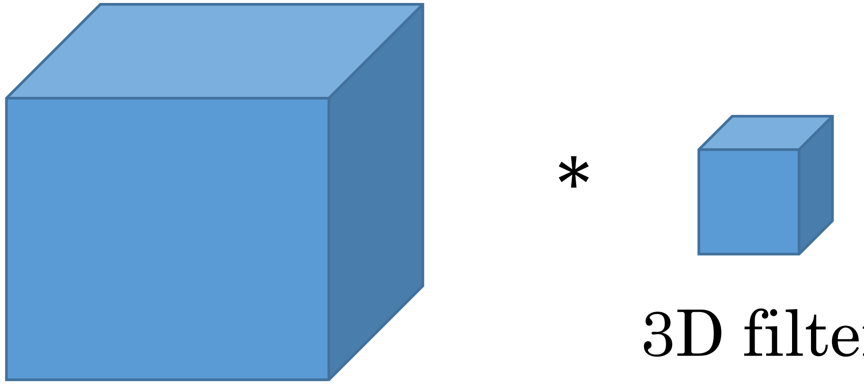
3D input image

3D filter

$5 \times 5 \times 5 \times 1$

$5 \times 5 \times 5 \times 16$

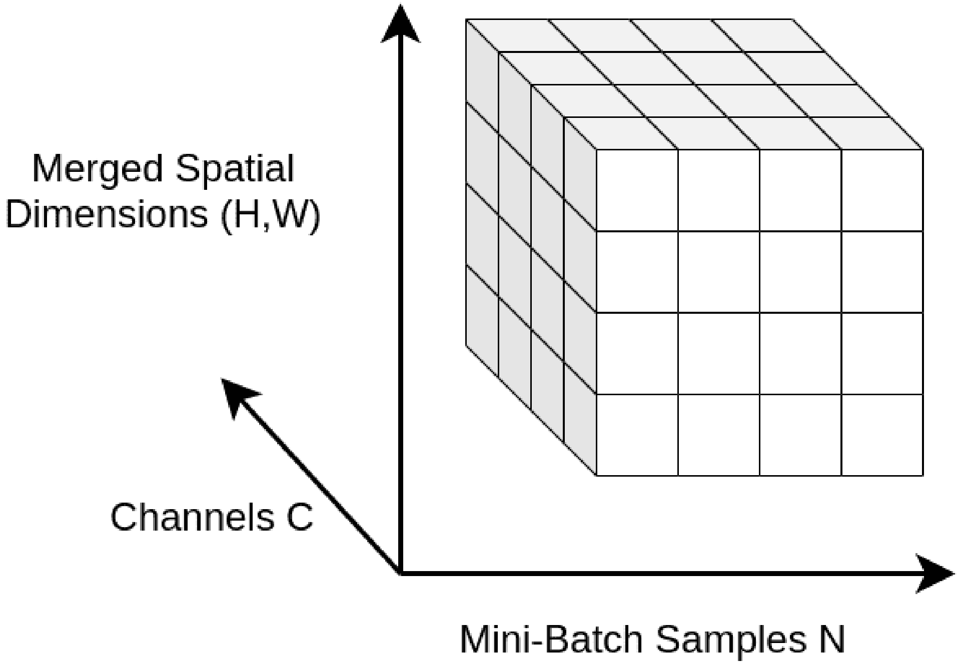
$14 \times 14 \times 14 \times 1 \ *$ → $10 \times 10 \times 10 \times 16$ → $6 \times 6 \times 6 \times 32$



3D filter

3D volume

Normalization in CNN



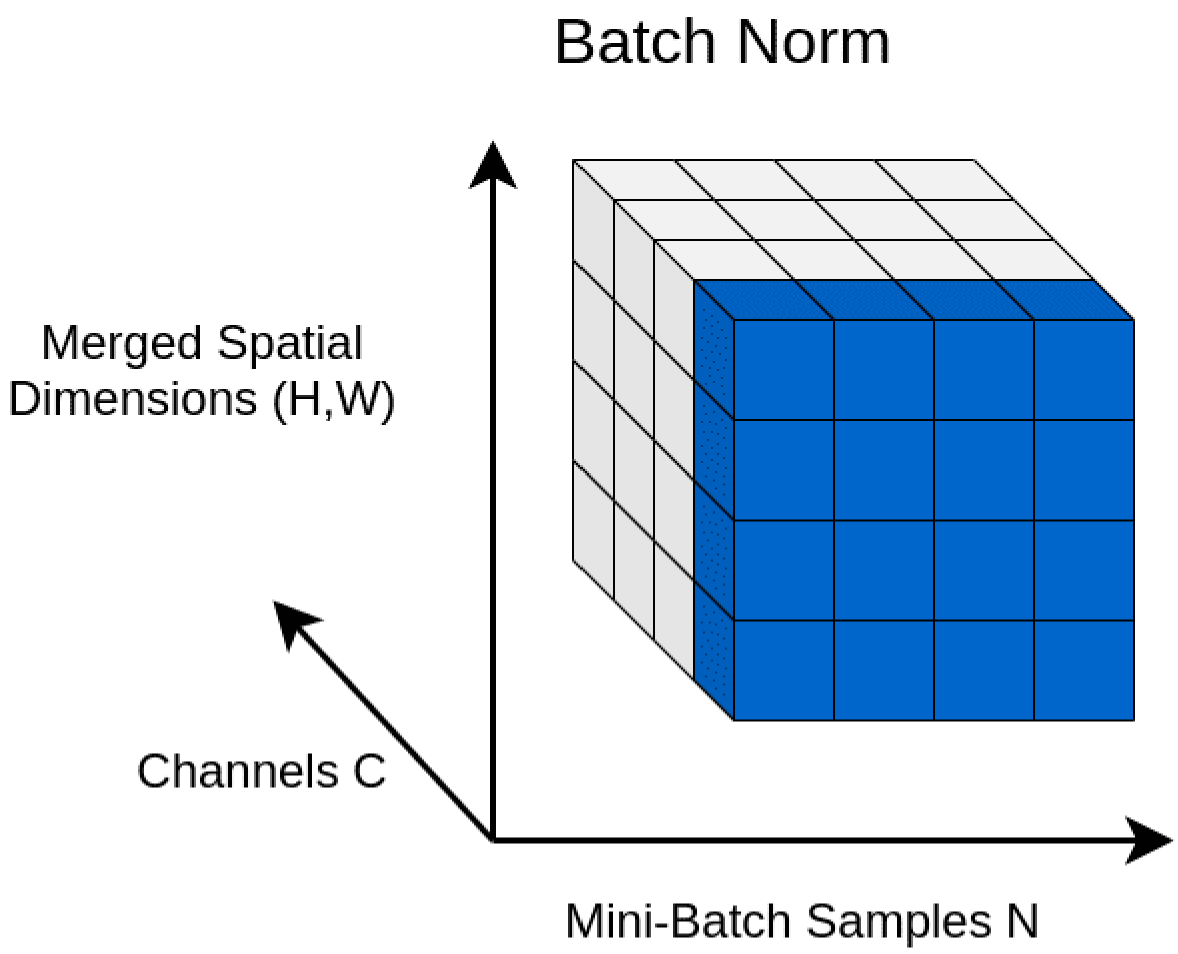
https://theaisummer.com/normalization/

Batch normalization

$$BN(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta$$

$$\mu_c(x) = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_c(x))^2}$$

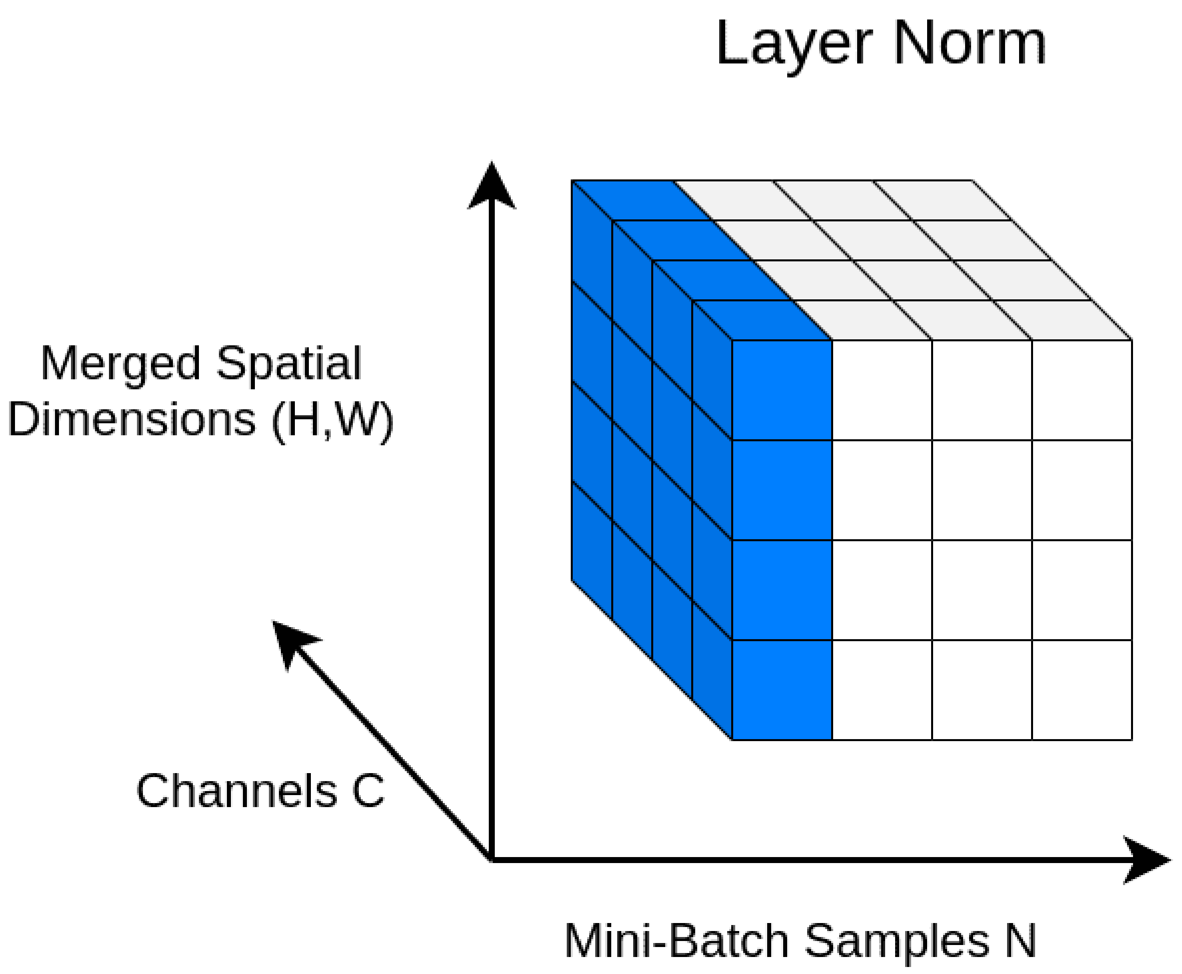


Layer normalization

$$LN(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta$$

$$\mu_n(x) = \frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

$$\sigma_n(x) = \sqrt{\frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_n(x))^2}$$

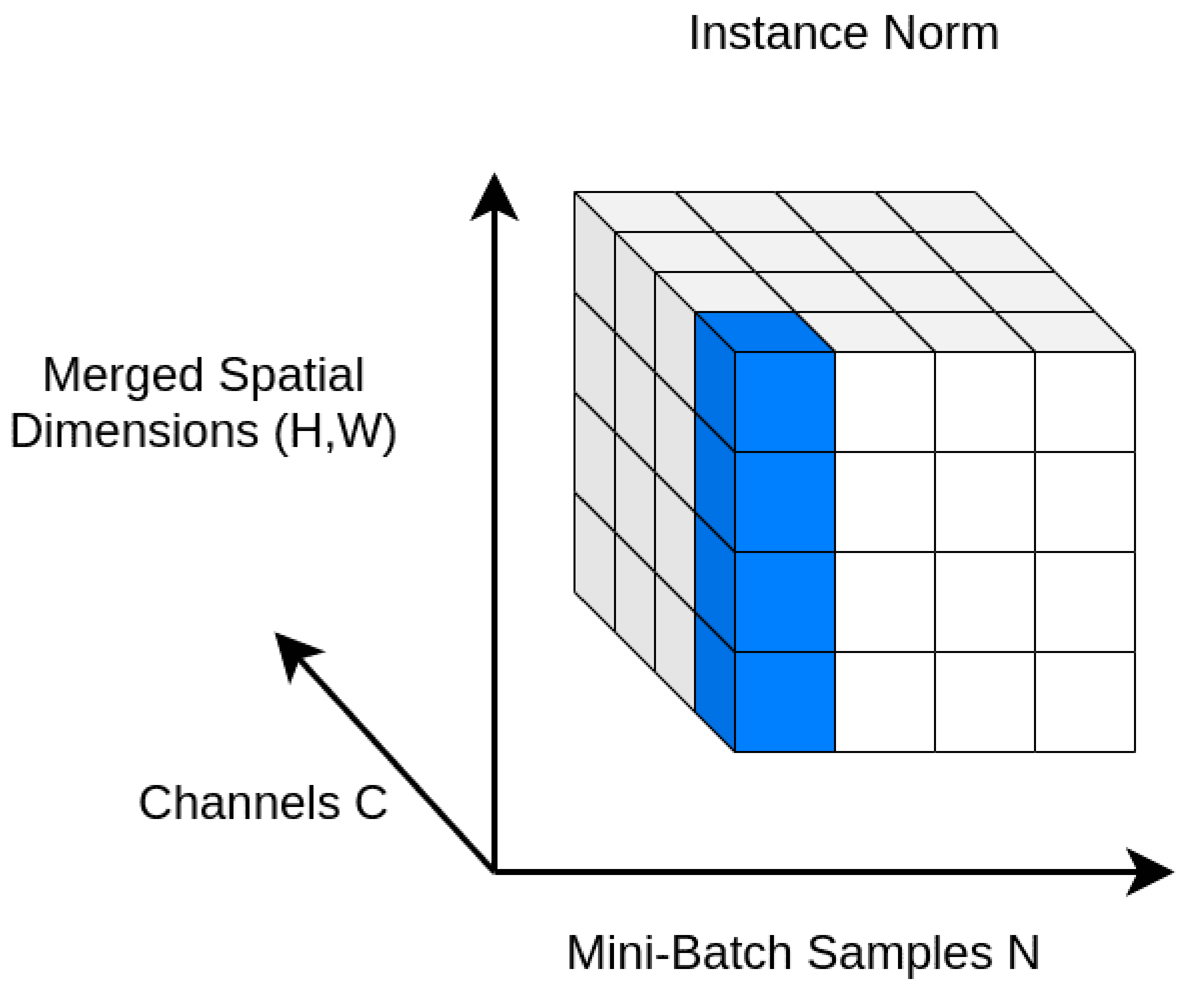


Instance Normalization

$$IN(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta$$

$$\mu_{nc}(x) = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{chw}$$

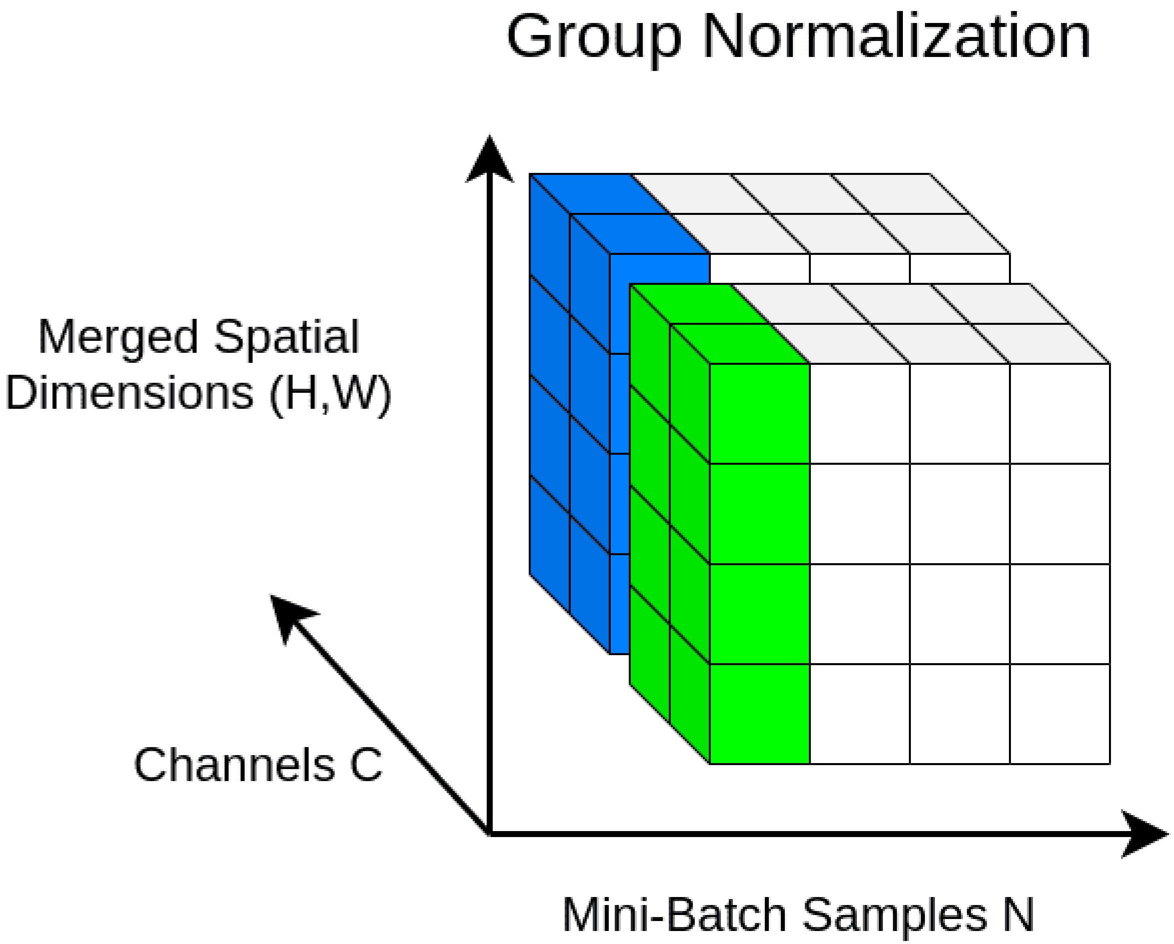
$$\sigma_{nc}(x) = \sqrt{\frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_c(x))^2}$$



Group Normalization

$$\mu_i = \frac{1}{m}\sum_{k\in\mathcal{S}_i} x_k, \quad \sigma_i = \sqrt{\frac{1}{m}\sum_{k\in\mathcal{S}_i}(x_k-\mu_i)^2+\epsilon}$$

$$\mathcal{S}_i = \left\{k \mid k_N = i_N, \left\lfloor \frac{k_C}{C/G} \right\rfloor = \left\lfloor \frac{i_C}{C/G} \right\rfloor \right\}$$



Weight standardization

$$\hat{W} = \left[\hat{W}_{i,j} \mid \hat{W}_{i,j} = \frac{W_{i,j} - \mu_{W_{i,\cdot}}}{\sigma_{W_{i,\cdot}}}\right]$$

$$y = \hat{W} * x$$

$$\mu_{W_{i,\cdot}} = \frac{1}{I}\sum_{j=1}^{I} W_{i,j}, \quad \sigma_{W_{i,\cdot}} = \sqrt{\frac{1}{I}\sum_{i=1}^{I}\left(W_{i,j} - \mu_{W_{i,\cdot}}\right)^2}$$