Introduction of Deep Learning Models

Presented By: Abdul Qayyum

Abdesslam BENZINOU, Kamal NASREDDINE, khawla bengaied

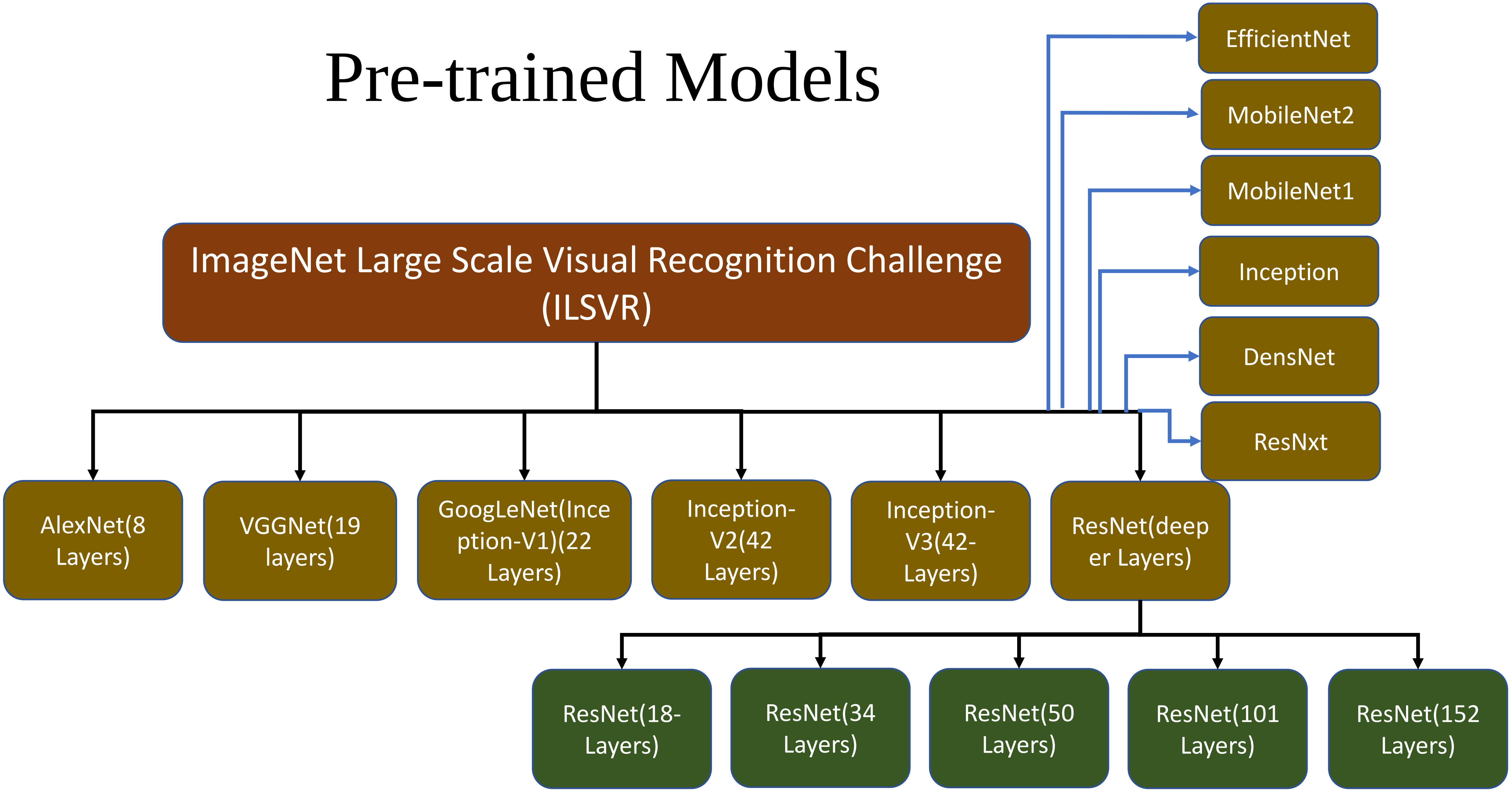
BLUE REVOLUTION

BIODIVERSITY UNDERESTIMATION IN OUR BLUE PLANET :
ARTIFICIAL INTELLIGENCE REVOLUTION
IN BENTHIC TAXONOMY

ÉCOLE NATIONALE D'INGÉNIEURS DE BREST

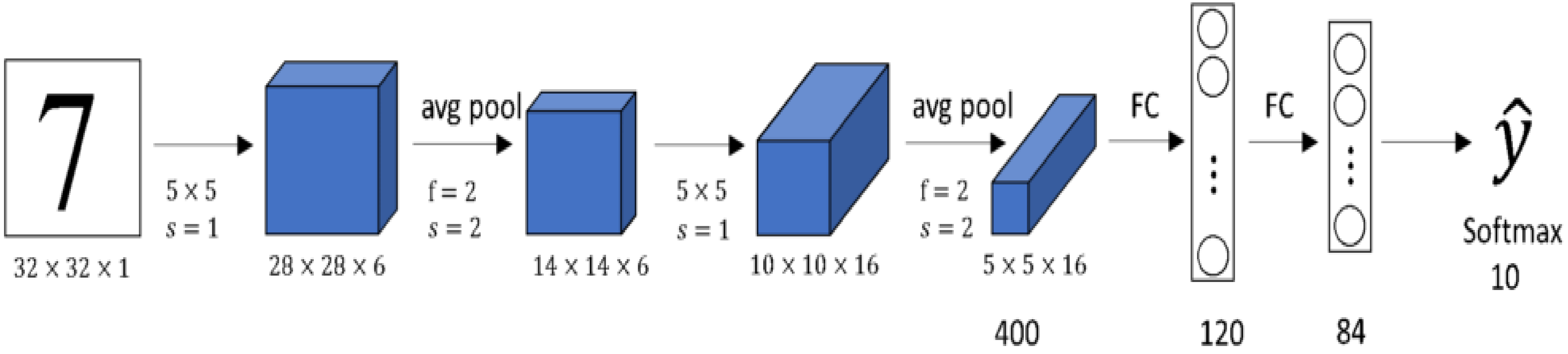
Deep Learning
Modeling

Pre-trained Models



LeNet

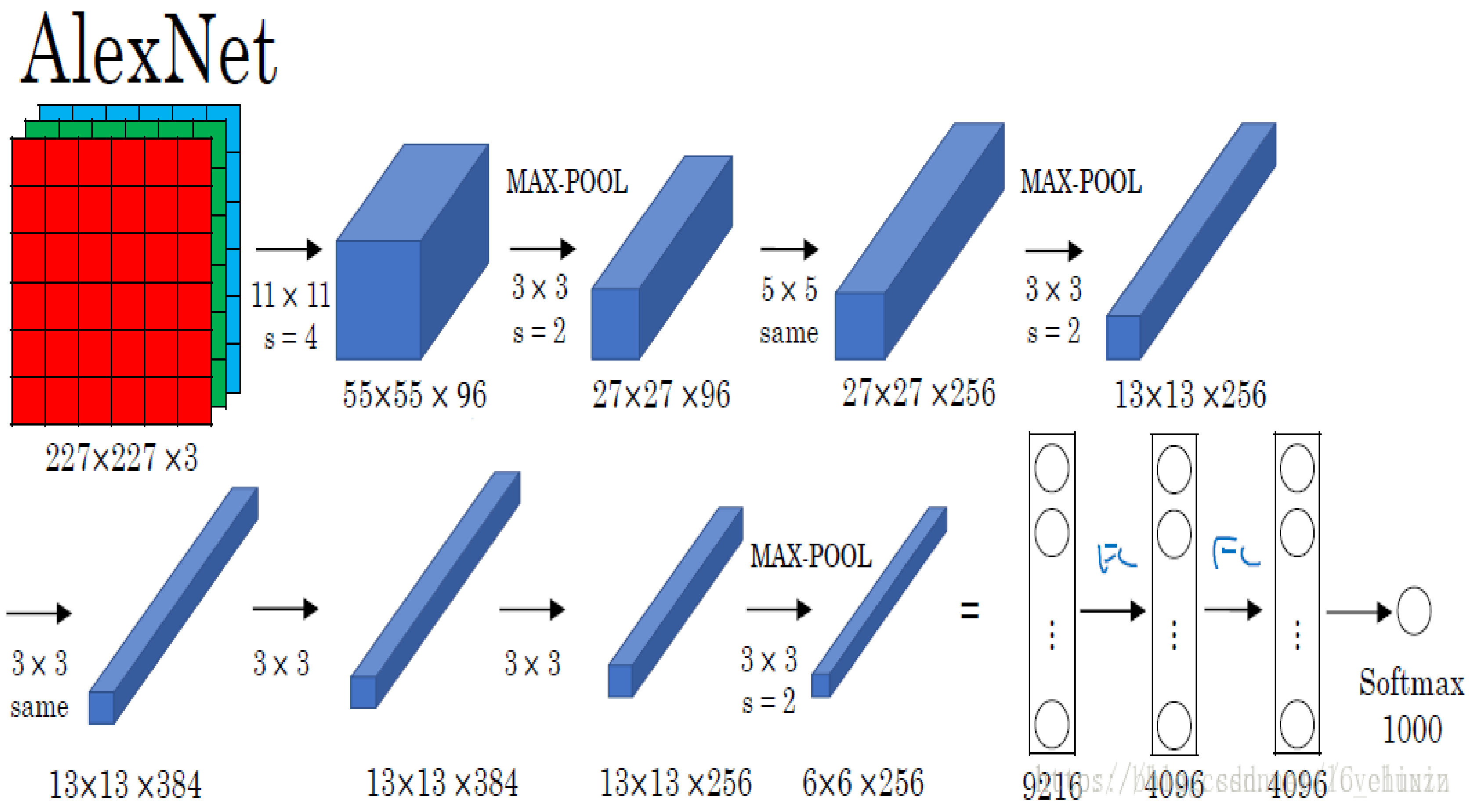
- **LeNet(Layers)**

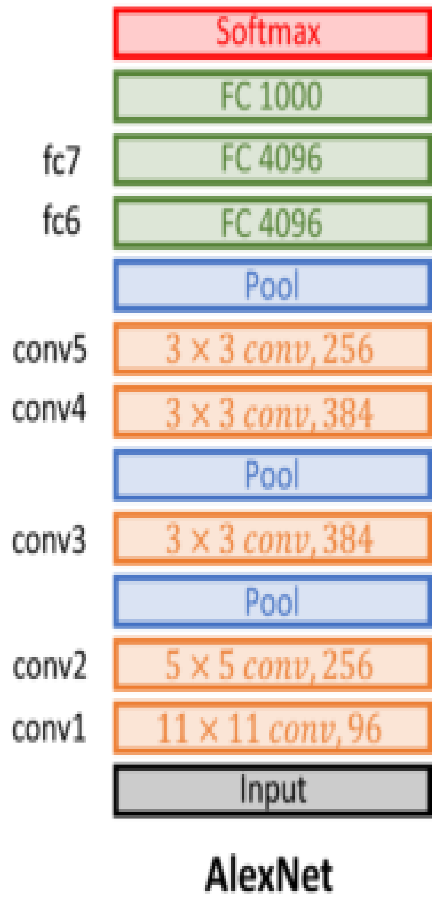


http://datahacker.rs/deep-learning-lenet-5-architecture/

AlexNet

- **AlexNet(8 Layers)**



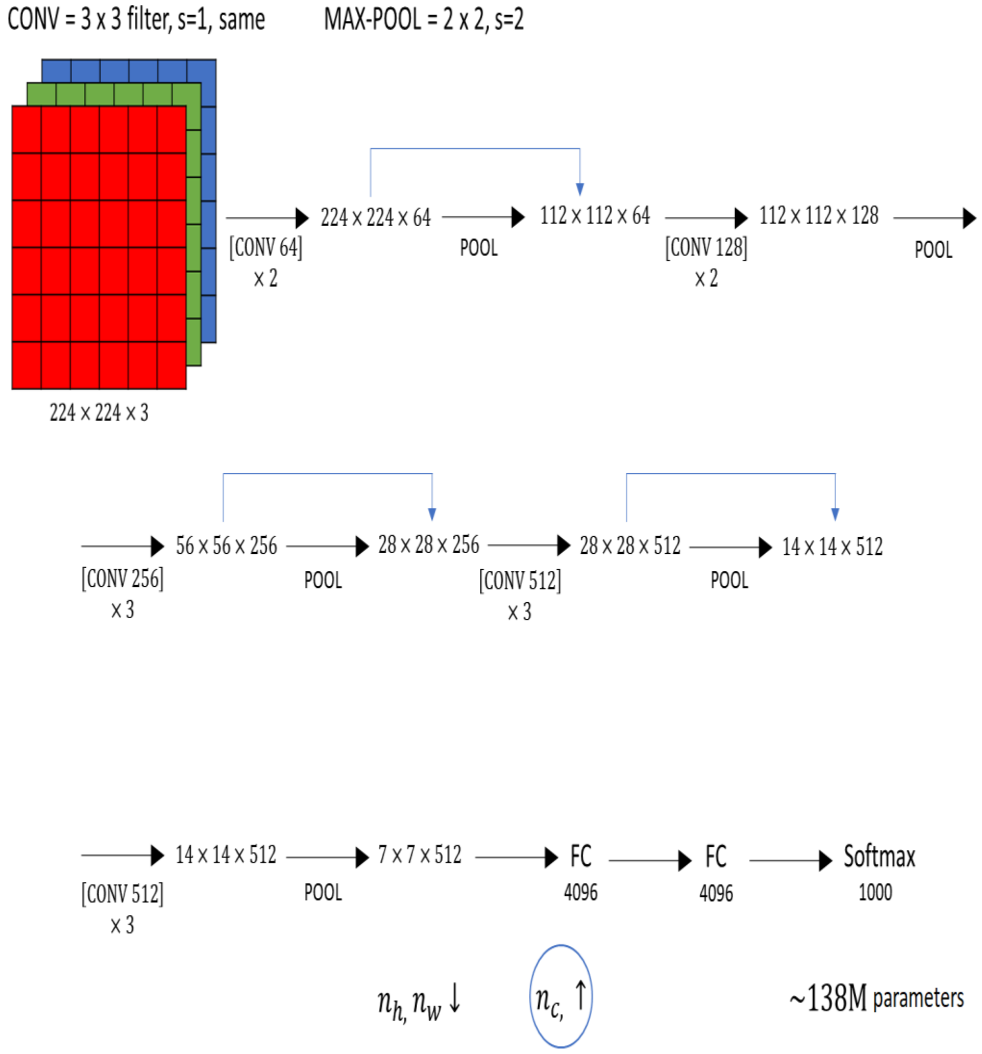


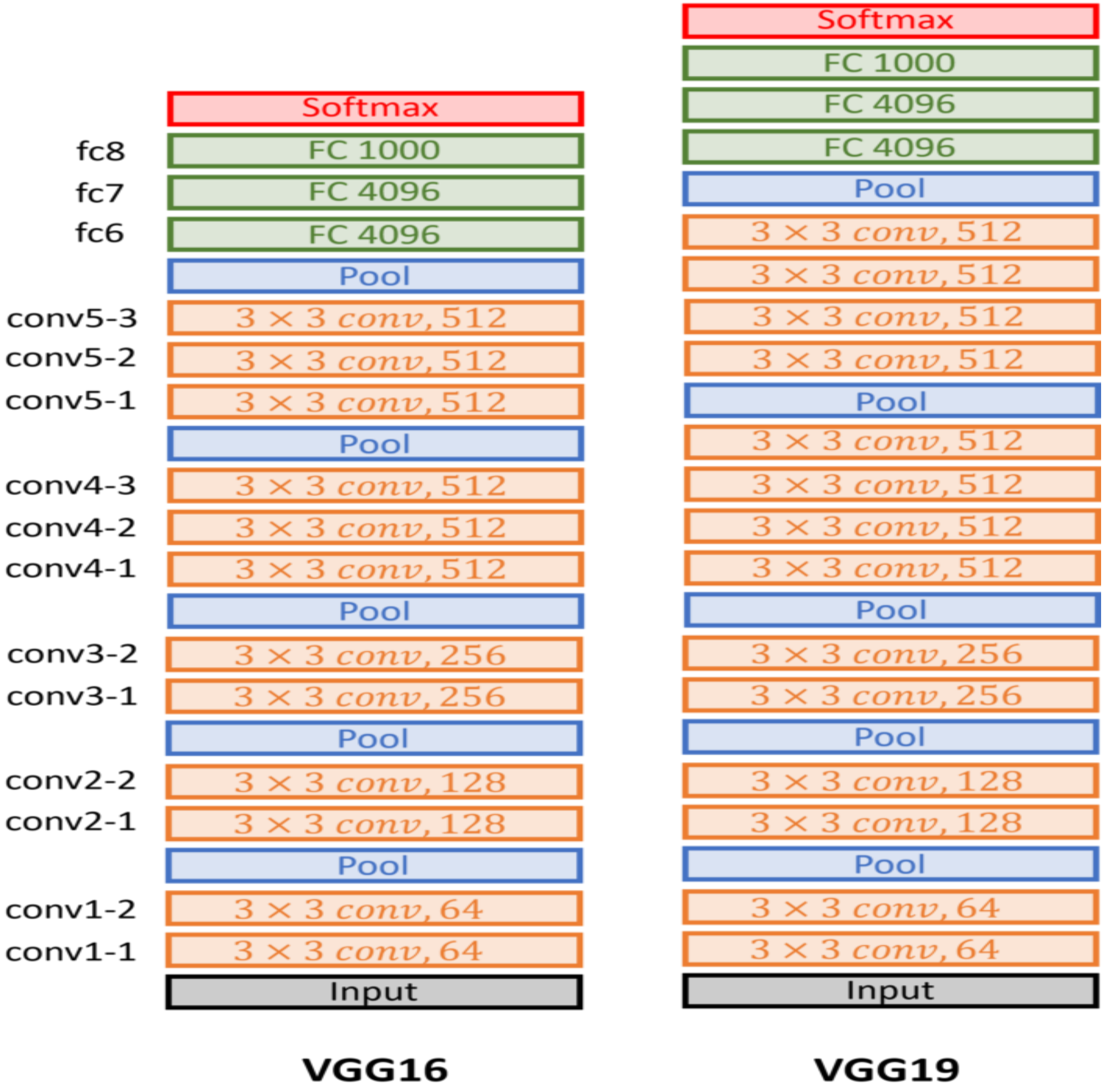
http://datahacker.rs/deep-learning-alexnet-architecture/

http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/

VGG Net

- **VGG16 and VGG19**

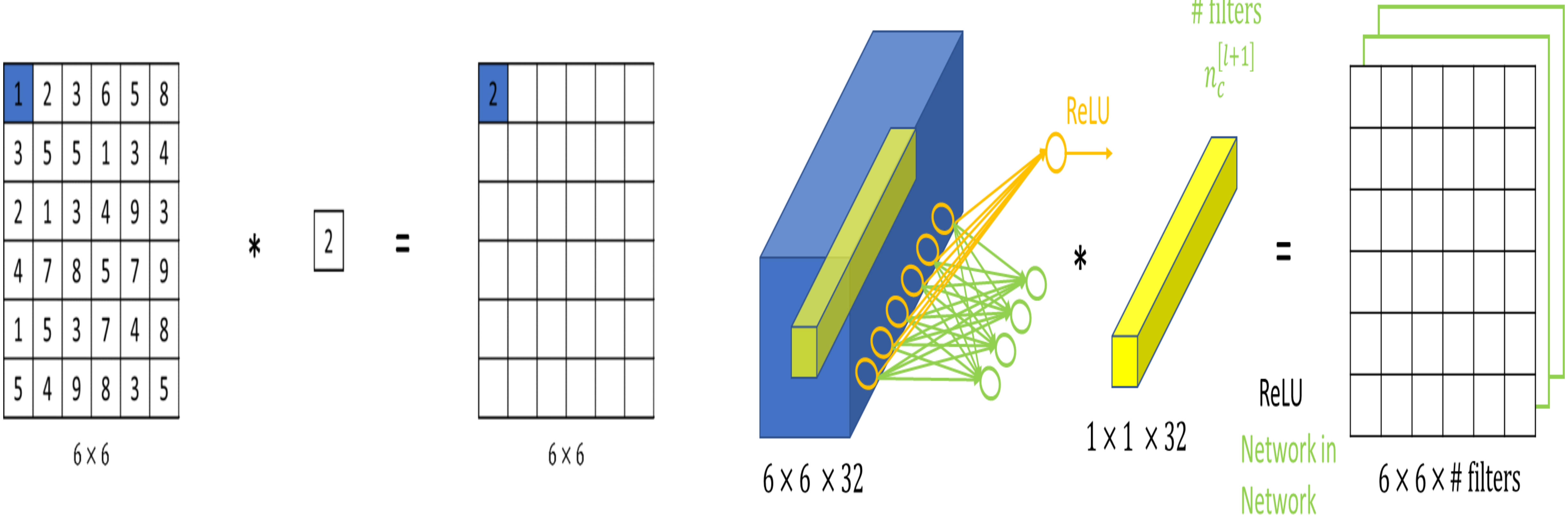




Network in Network

http://datahacker.rs/introduction-into-1x1-convolution/

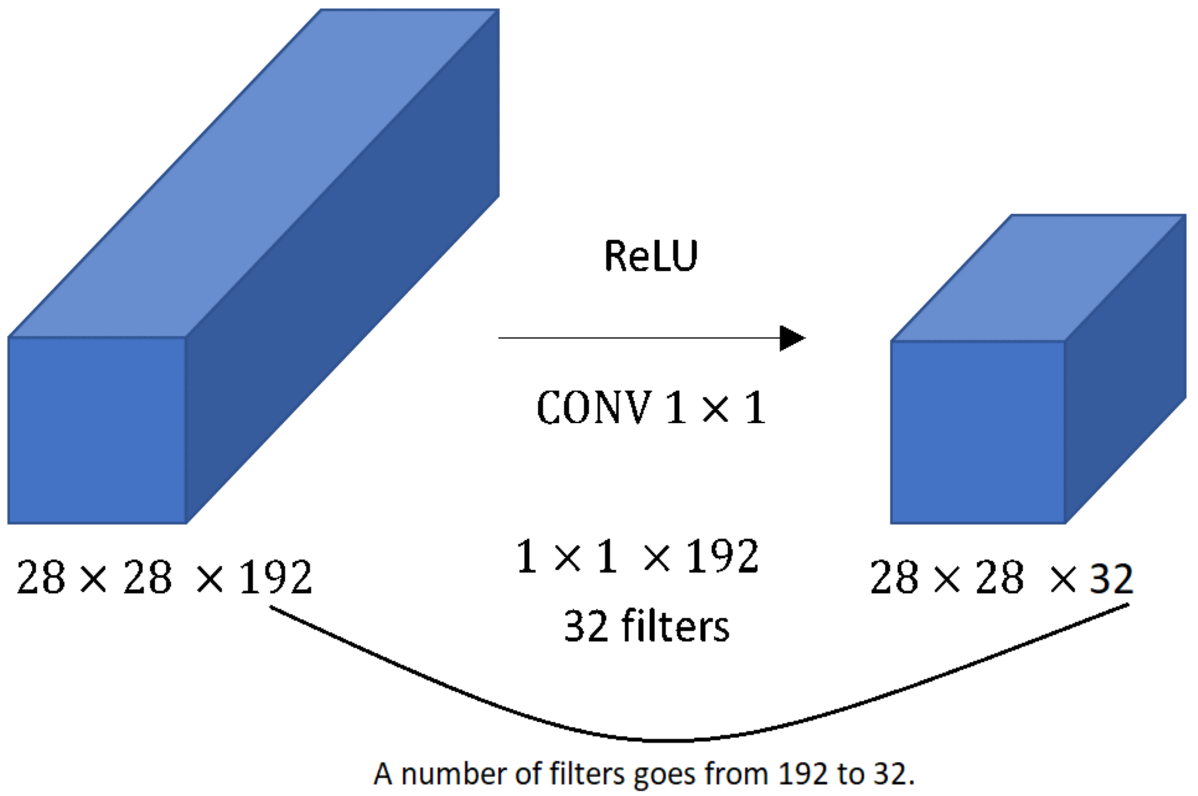
- **Network in Network – 1×1 Convolutions**



In particular a 1×1 convolution will look at each of the 36 different positions (6×6), and it will take the element-wise product between 32 numbers on the left and the 32 numbers in the filter. Then, a ReLU non-linearity will be applied. Looking at one out of the 36 positions, maybe one slice through this volume, we take these 32 numbers, multiply it by 1×1 slice through the volume, and we get a single number.

Network in Network

- **Network in Network – 1×1 Convolutions**

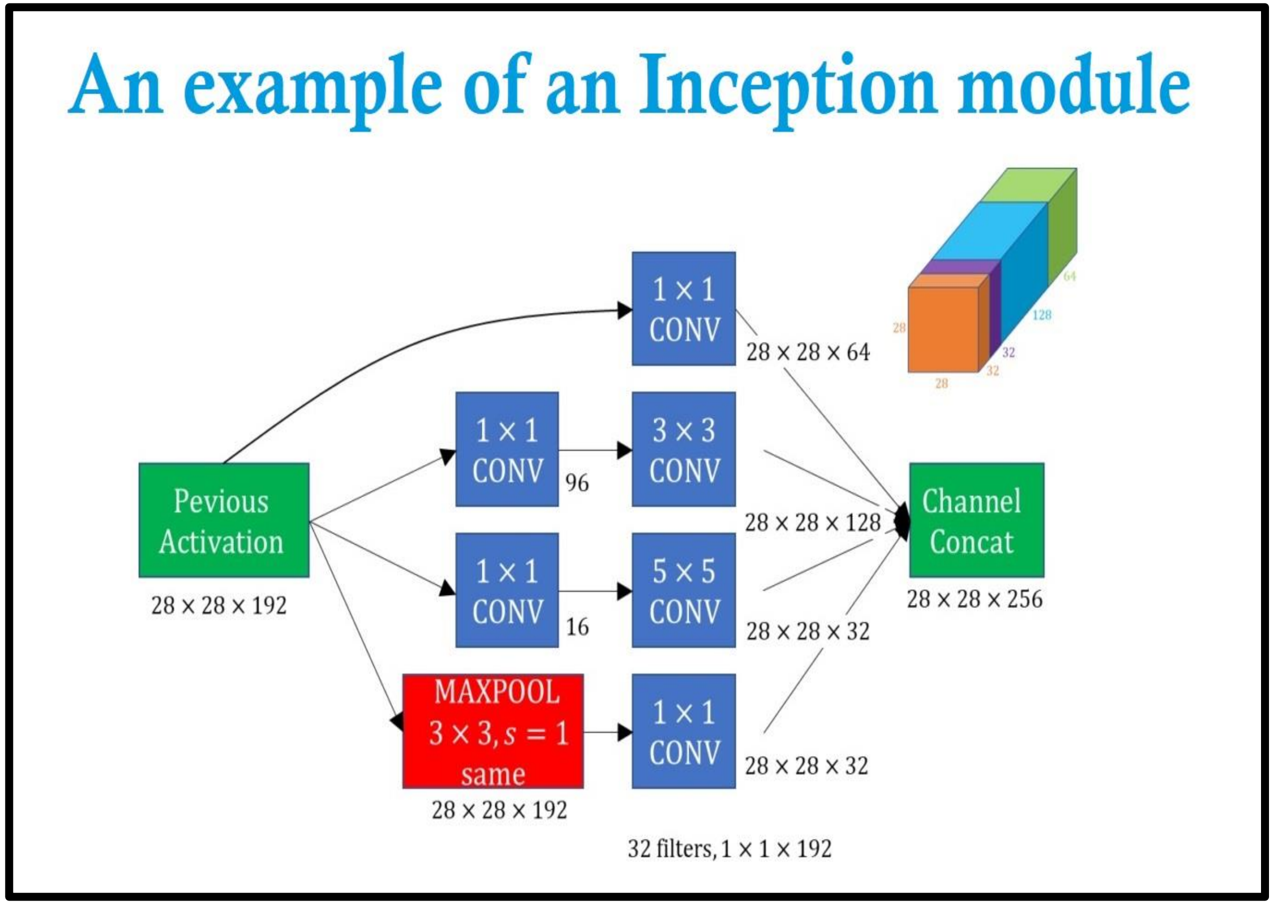


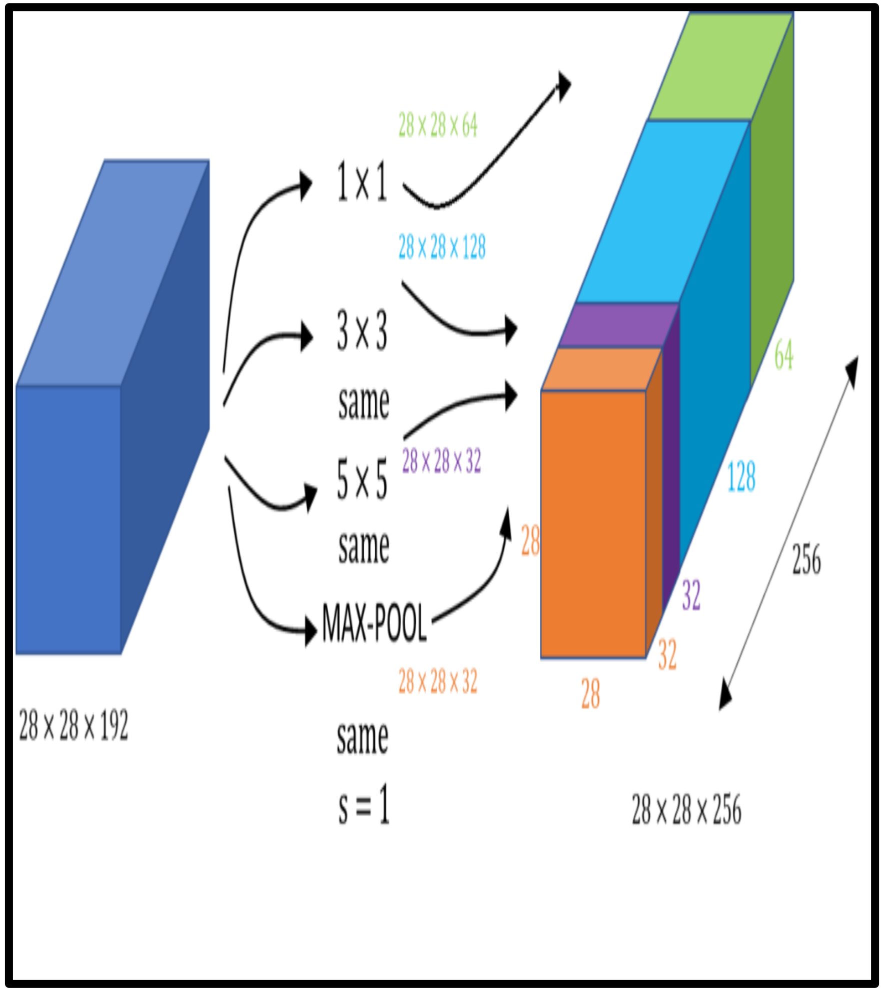
We can do is use 32 filters that are 1×1, and technically each filter would be of dimension 1×1×192 because the number of channels in our filter has to match the number of channels in input volume. But we use 32 filters and the output of this process will be a 28×28×32 volume. This is a way to let us shrink nc (number of channels). We'll see later how this idea of 1×1 convolutions allows us to shrink the number of channels and therefore save on computations and networks.

http://datahacker.rs/deep-learning-inception-network/

Inception Module

- **Inception Module**

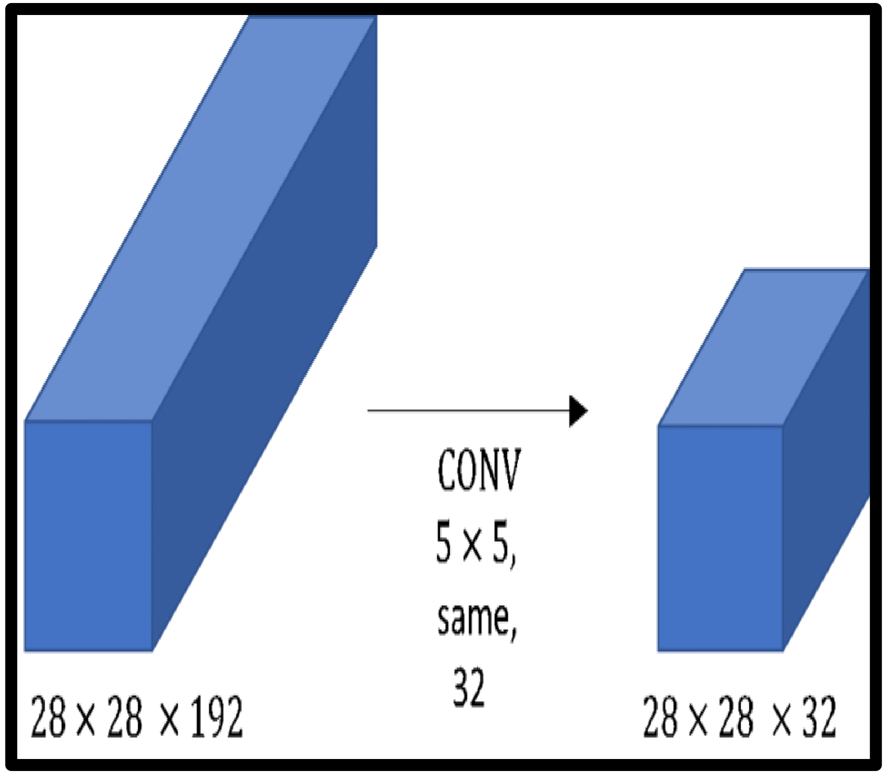


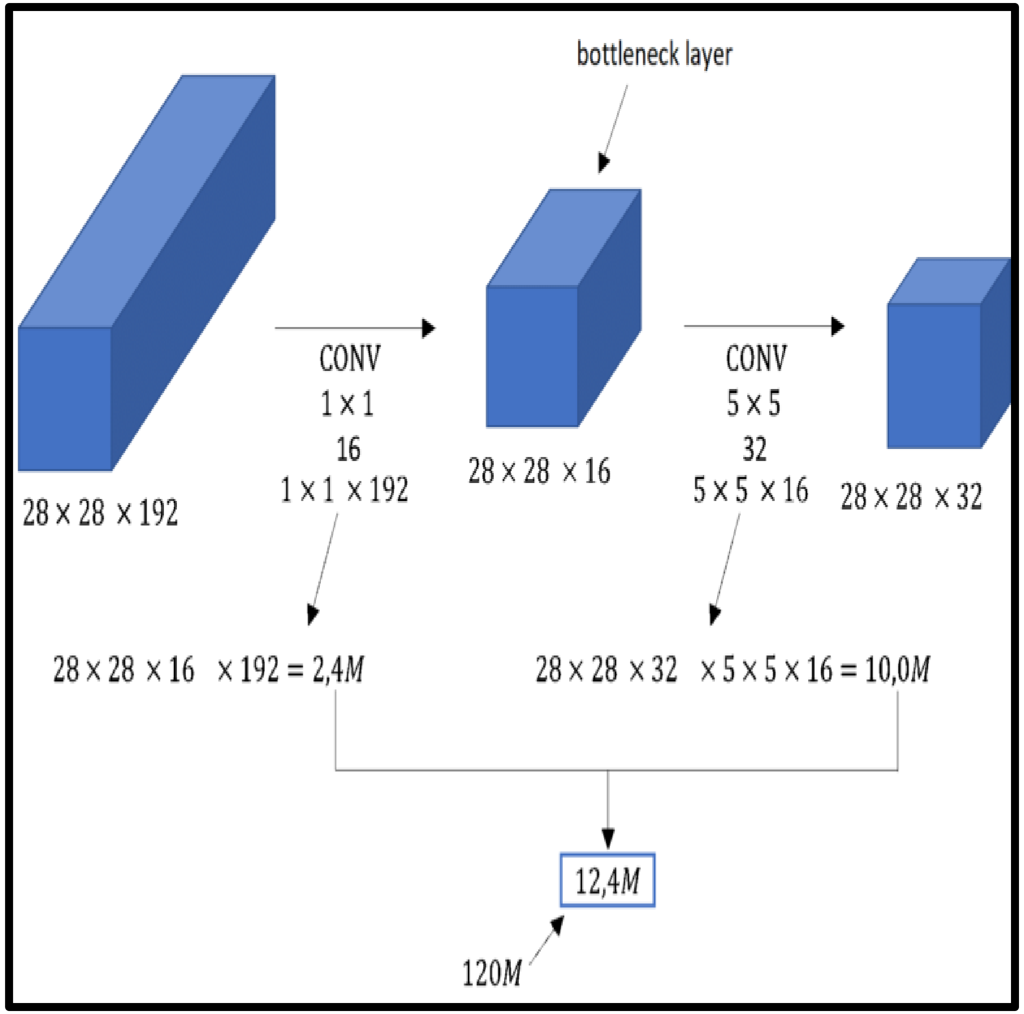


http://datahacker.rs/deep-learning-inception-network/

Inception Module

- **Inception Module**





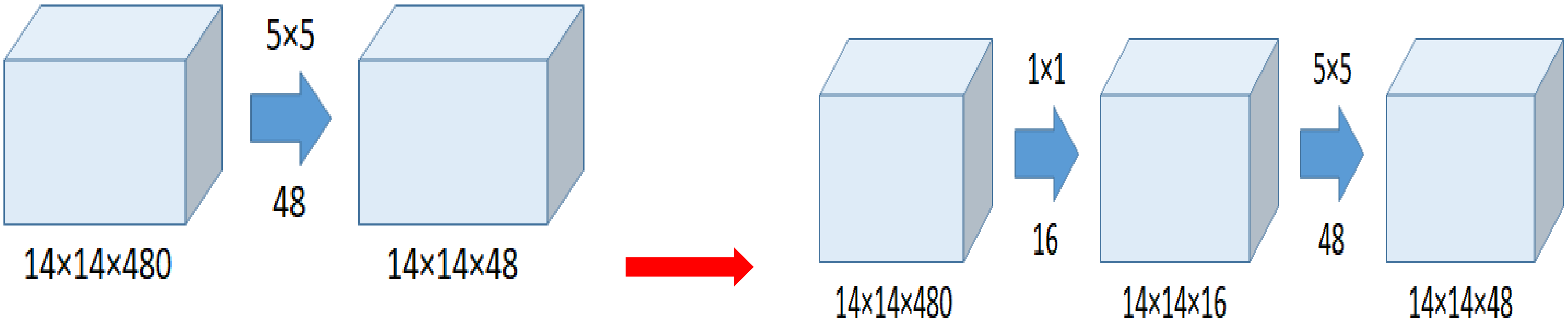
28×28×32x5x5x192=120M

http://datahacker.rs/deep-learning-inception-network/

https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

Inception Module

- **Inception Module**



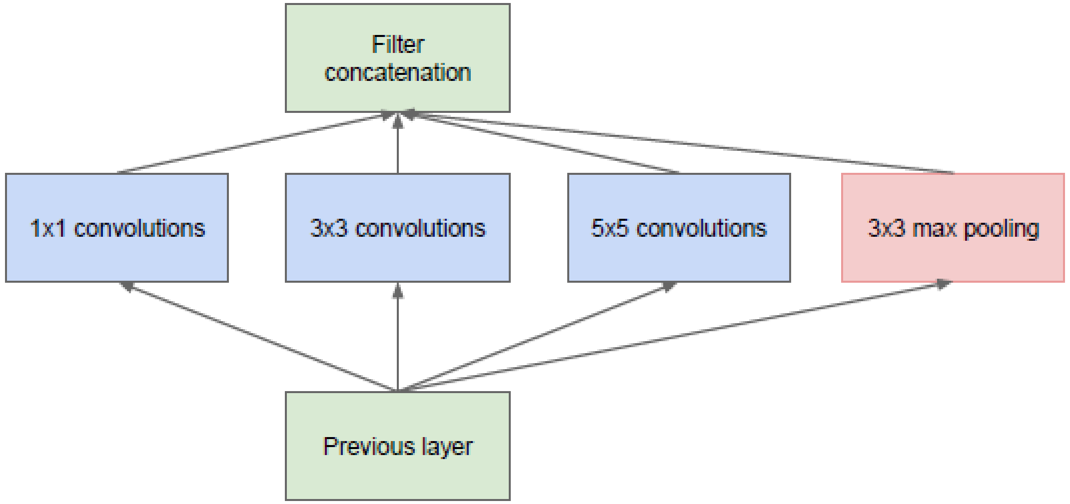
Number of operations = (14×14×48)×(5×5×480) = 112.9M

Number of operations for 1×1 = (14×14×16)×(1×1×480) = 1.5M
Number of operations for 5×5 = (14×14×48)×(5×5×16) = 3.8M
Total number of operations = 1.5M + 3.8M = 5.3M
which is much much smaller than 112.9M !!!!!!!!!!!!!!

Optimization Algorithms

https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

- **Inception Module**



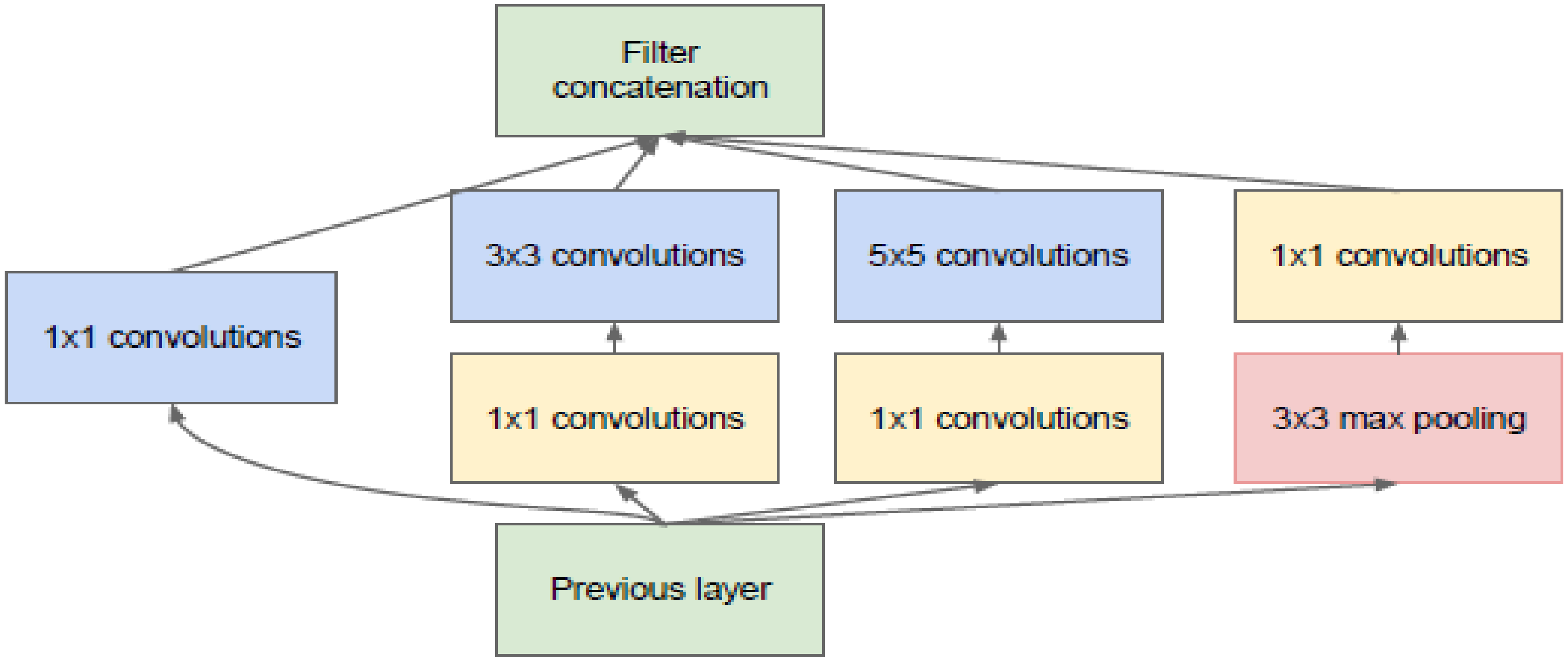
(a) Inception module, naïve version

Previously, such as AlexNet, and VGGNet, conv size is fixed for each layer.
Now, 1×1 conv, 3×3 conv, 5×5 conv, and 3×3 max pooling are done altogether for the previous input, and stack together again at output. When image's coming in, different sizes of convolutions as well as max pooling are tried. Then different kinds of features are extracted.
After that, all feature maps at different paths are concatenated together as the input of the next module.
However, without the 1×1 convolution as above, we can imagine how large the number of operation is!

Optimization Algorithms

https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

- **Inception Module**



(b) Inception module with dimensionality reduction

Optimization Algorithms

https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

- **Inception Module**

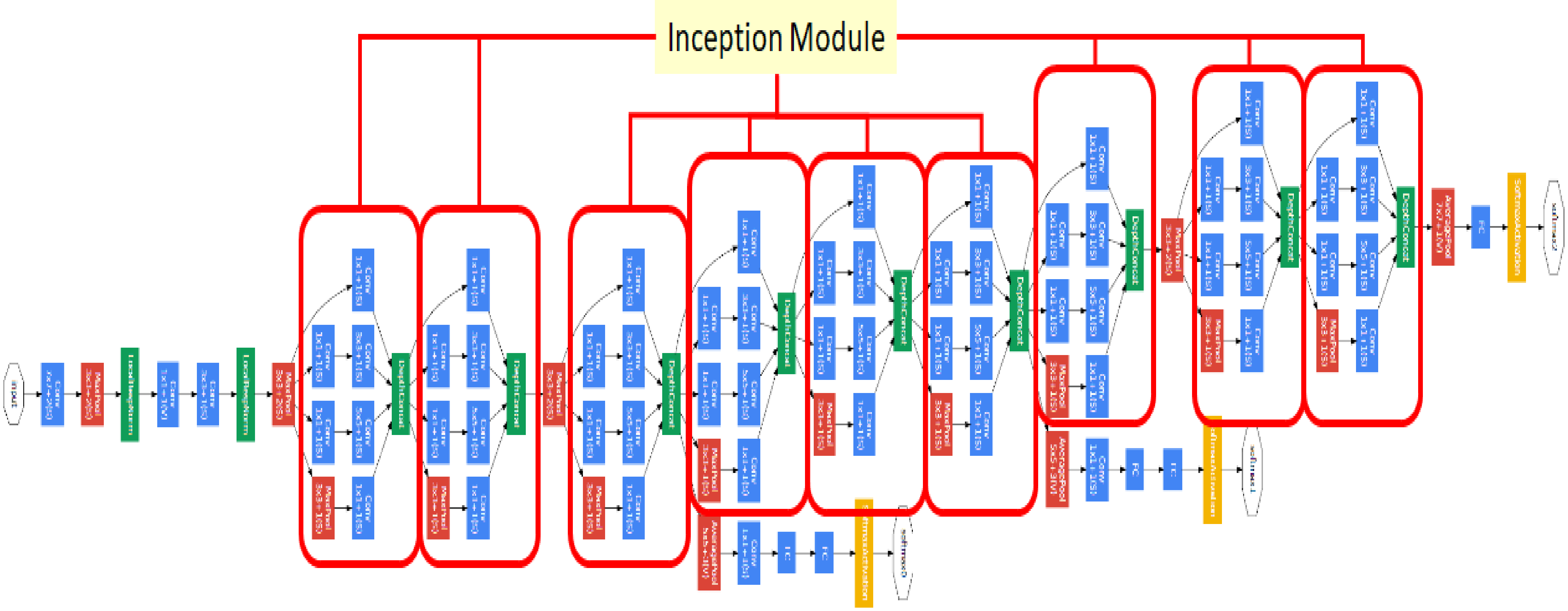
"The Inception deep convolutional architecture was introduced as GoogLeNet in (Szegedy et al. 2015a), here named Inception-v1.

Later the Inception architecture was refined in various ways, first by the introduction of batch normalization (Ioffe and Szegedy 2015) (Inception-v2).
Later by additional factorization ideas in the third iteration (Szegedy et al. 2015b) which will be referred to as Inception-v3 in this report."

GoogleNet

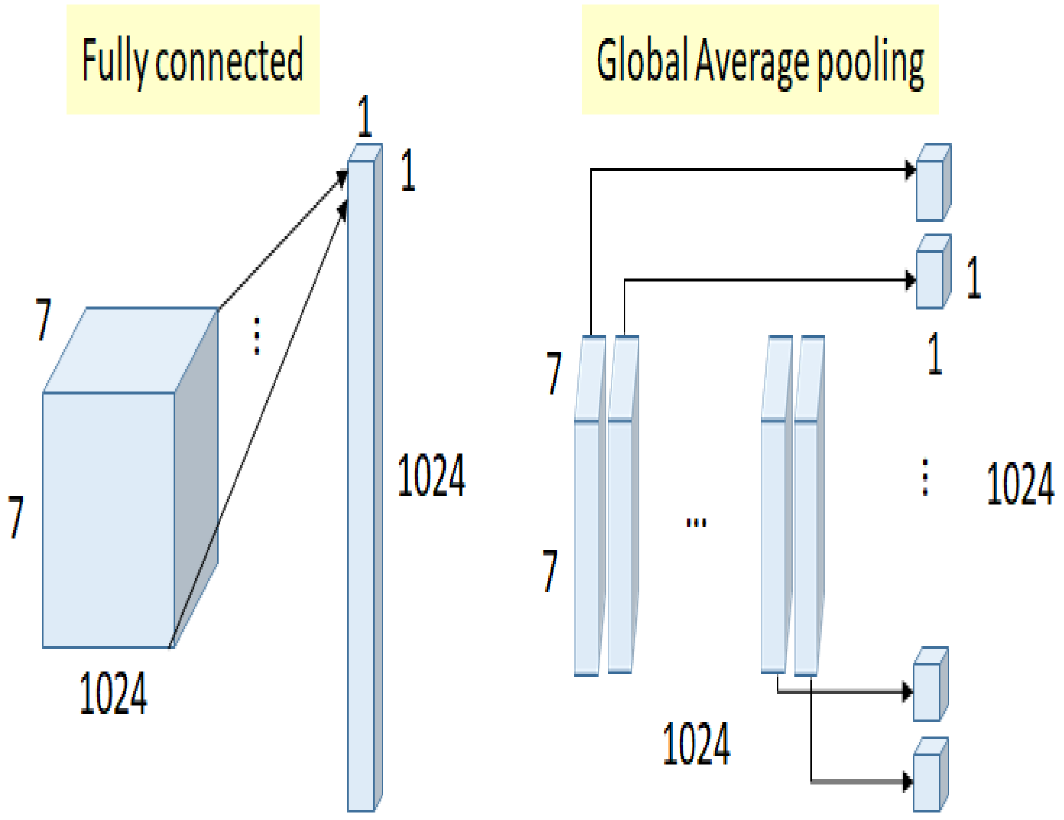
https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

- **GoogLeNet(Inception_V1)**



GoogleNet

- **GoogleNet**

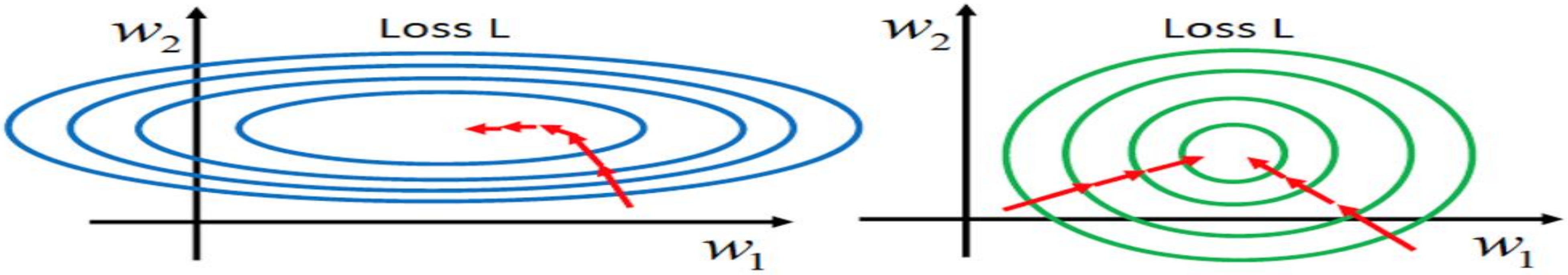


Previously, fully connected (FC) layers are used at the end of network, such as in AlexNet.
All inputs are connected to each output.
Number of weights (connections) above = 7×7×1024×1024 = 51.3M

In GoogLeNet, global average pooling is used nearly at the end of network by averaging each feature map from 7×7 to 1×1
Number of weights = 0
And authors found that a move from FC layers to average pooling improved the top-1 accuracy by about 0.6%.

Inception v2

- **Why we need Batch Normalization (BN)?**
- As we should know, the input X is multiplied by weight W and added by bias b and become the output Y at the next layer after an activation function F:
- $Y = F(W \cdot X + b)$
- Previously, F is sigmoid function which is easily saturated at 1 which easily makes the gradient become zero. As the network depth increases, this effect is amplified, and thus slow down the training speed.
- ReLU is then used as F, where ReLU(x)=max(x,0), to address the saturation problem and the resulting vanishing gradients. However, careful initialization, learning rate settings are required.



Without BN (Left), With BN (Right)

Inception v2

- **Why we need Batch Normalization (BN)?**
- It is advantageous for the distribution of X to remain fixed over time because a small change will be amplified when network goes deeper.
- BN can reduce the dependence of gradients on the scale of the parameters or their initial values. As a result,
- **Higher learning rate can be used.**
- **The need for Dropout can be reduced.**

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

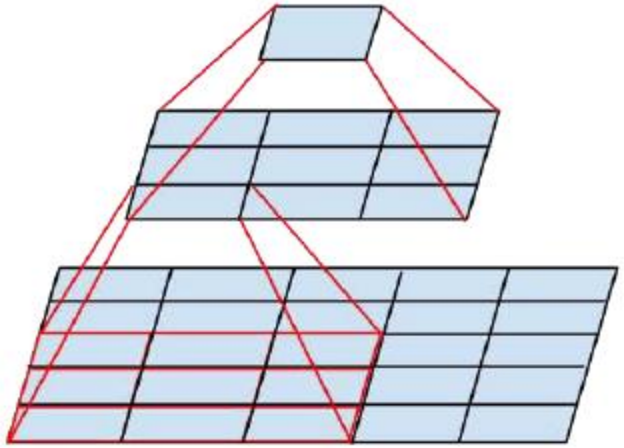
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Y=F(BN(W · X+b))
To have a more precise mean and variance, moving average is used to calculate the mean and variance.

Inception v2 and V3

- **Factorization Into Asymmetric Convolutions**
- Thus, the BN-Inception / Inception-v2 [6] is talking about batch normalization while Inception-v3 [1] is talking about factorization ideas.
- **Factorizing Convolutions**
- The aim of factorizing Convolutions is to reduce the number of connections/parameters without decreasing the network efficiency.

Factorization Into Smaller Convolutions
Two 3×3 convolutions replaces one 5×5 convolution as follows:

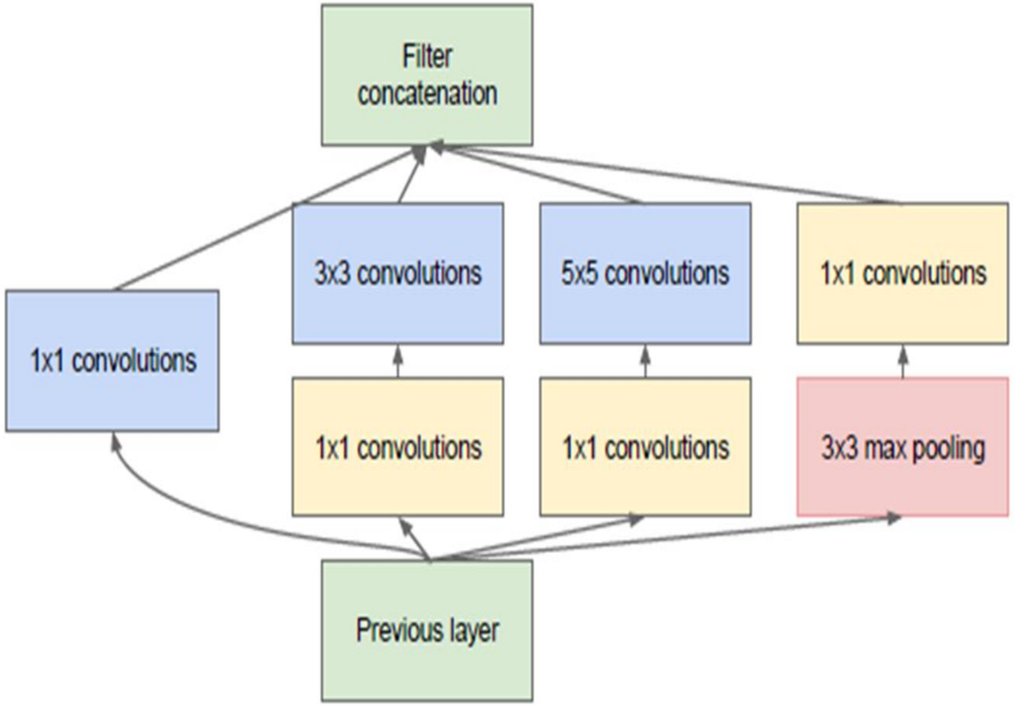


By using 1 layer of 5×5 filter, number of operations = 5×5=25
By using 2 layers of 3×3 filters, number of operations = 3×3+3×3=18
Number of operations is reduced by 28%

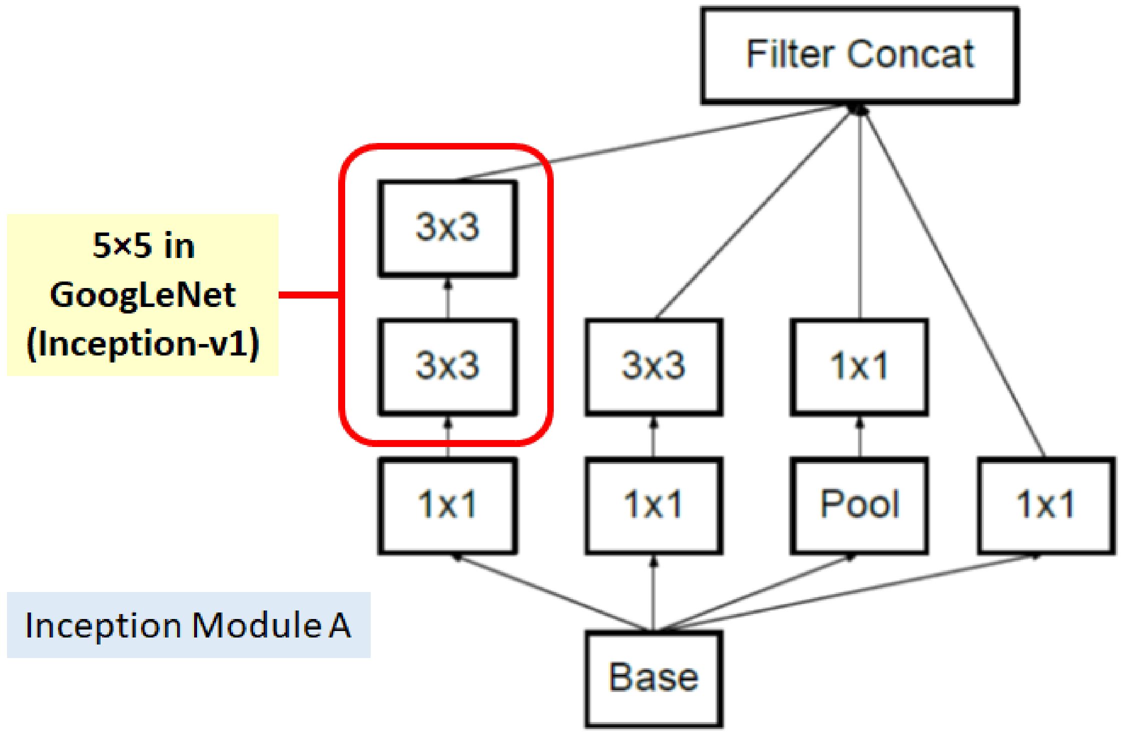
Two 3×3 convolutions replacing one 5×5 convolution

Inception v2 and V3

- **Factorization Into Asymmetric Convolutions**
- With this technique, one of the new Inception modules (I call it Inception Module A here) becomes:

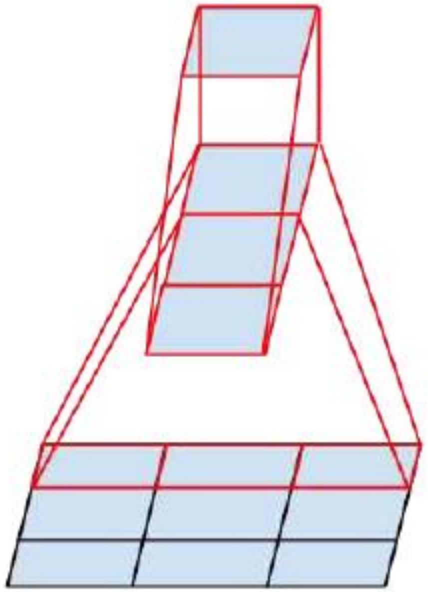


(b) Inception module with dimensionality reduction



Inception v2 and V3

- **Factorization Into Asymmetric Convolutions**
- Factorization Into Asymmetric Convolutions
- One 3×1 convolution followed by one 1×3 convolution replaces one 3×3 convolution as follows:



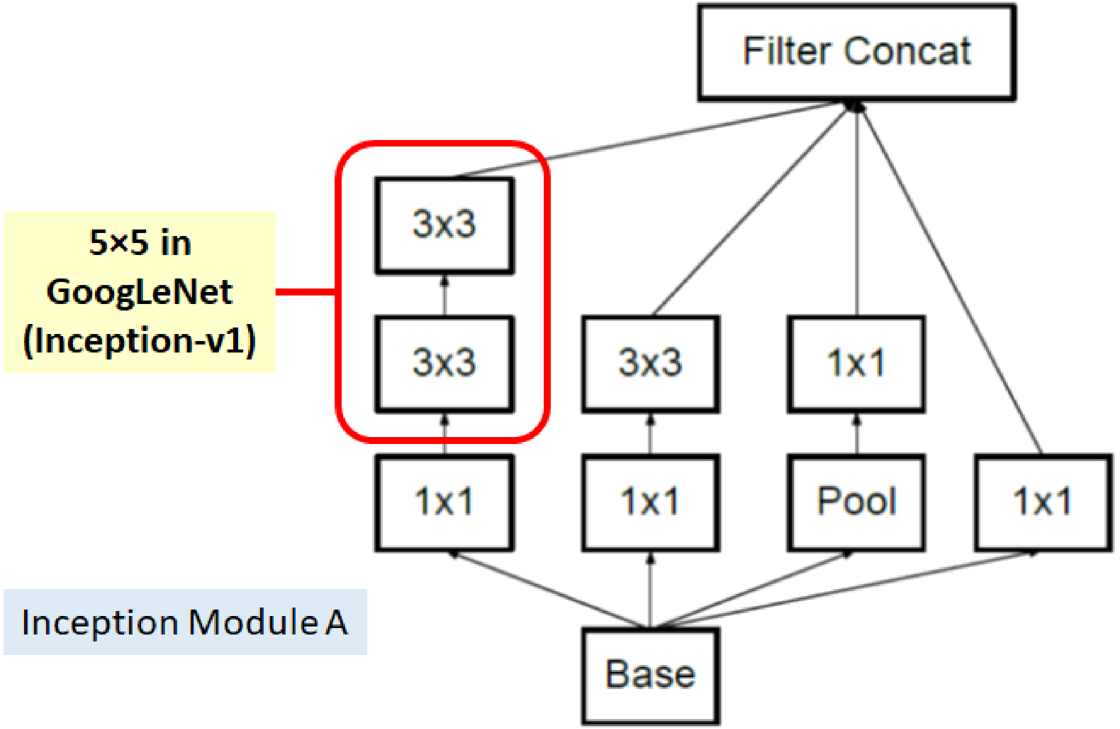
By using 3×3 filter, number of operations = 3×3=9
By using 3×1 and 1×3 filters, number of operations= 3×1+1×3=6
Number of operations is reduced by 33%
You may ask why we don't use two 2×2 filters to replace one 3×3 filter?
If we use two 2×2 filters, number of operations = 2×2×2=8
Number of operations is only reduced by 11%

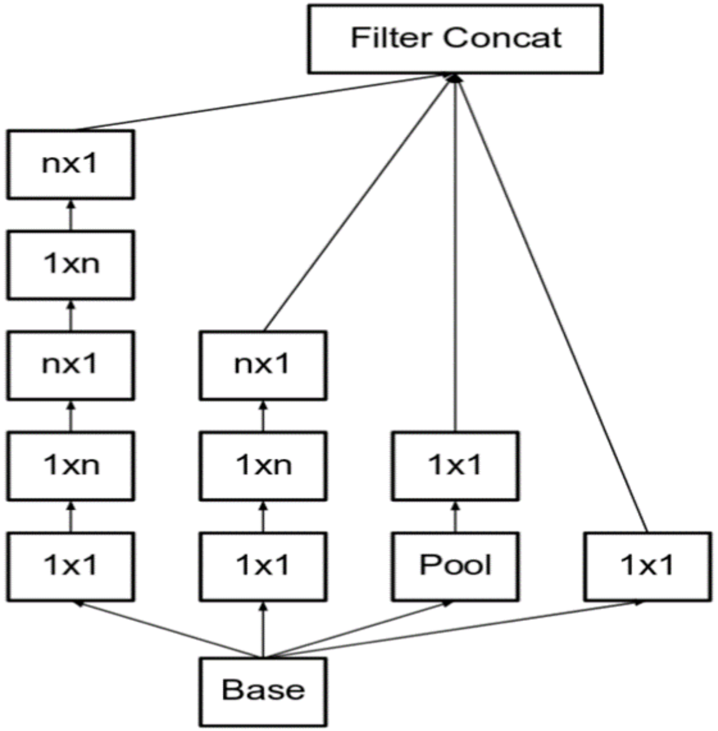
One 3×1 convolution followed by one 1×3 convolution replaces one 3×3 convolution

Inception v2 and V3

https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

- **Factorization Into Asymmetric Convolutions**
- With this technique, one of the new Inception modules (I call it Inception Module A here) becomes:

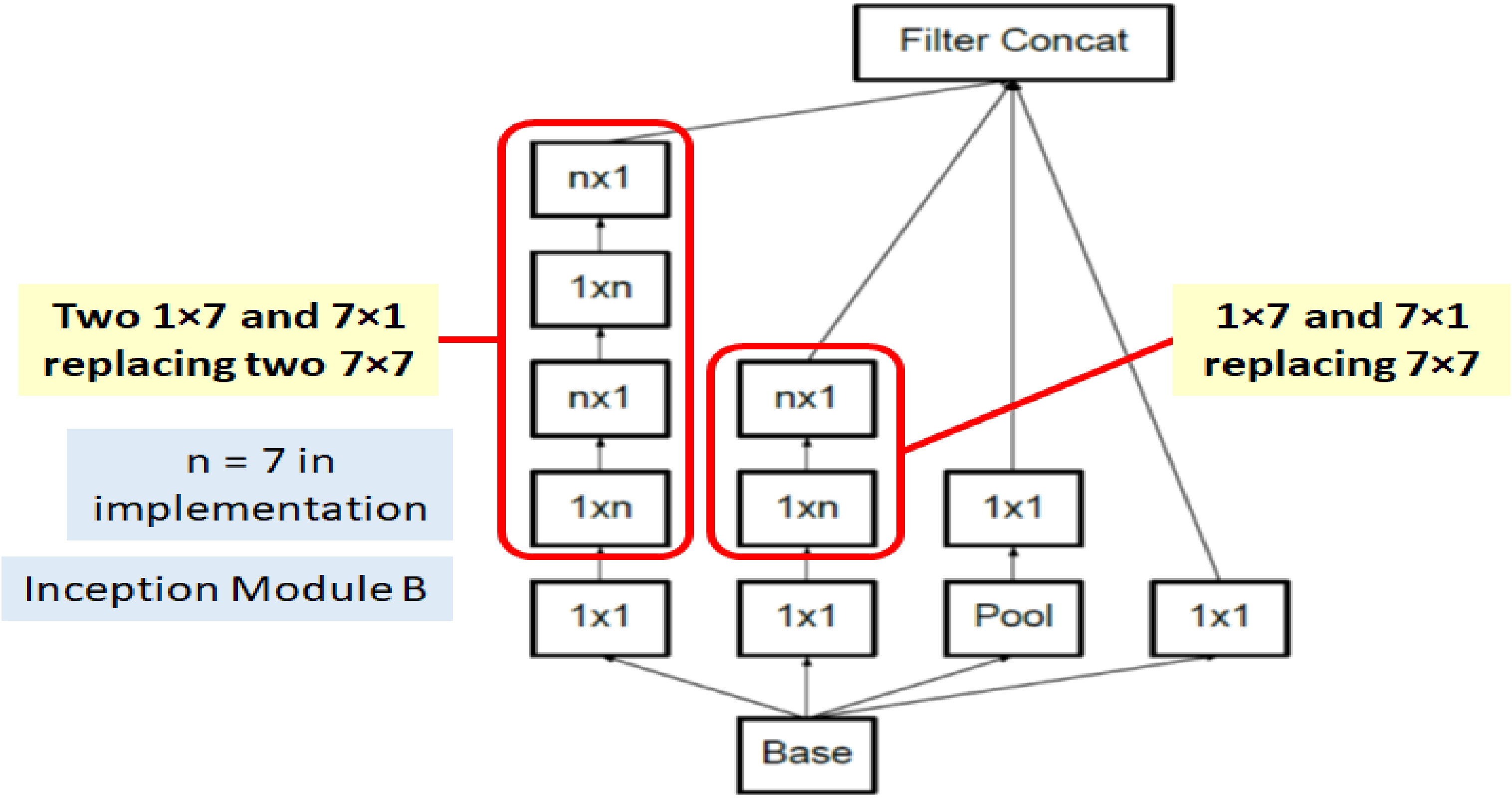




Here, put n=3 to obtain the equivalent of the previous image. The left-most 5x5 convolution can be represented as two 3x3 convolutions, which in turn are represented as 1x3 and 3x1 in series. (Source: Incpetion v2)

Inception v2 and V3

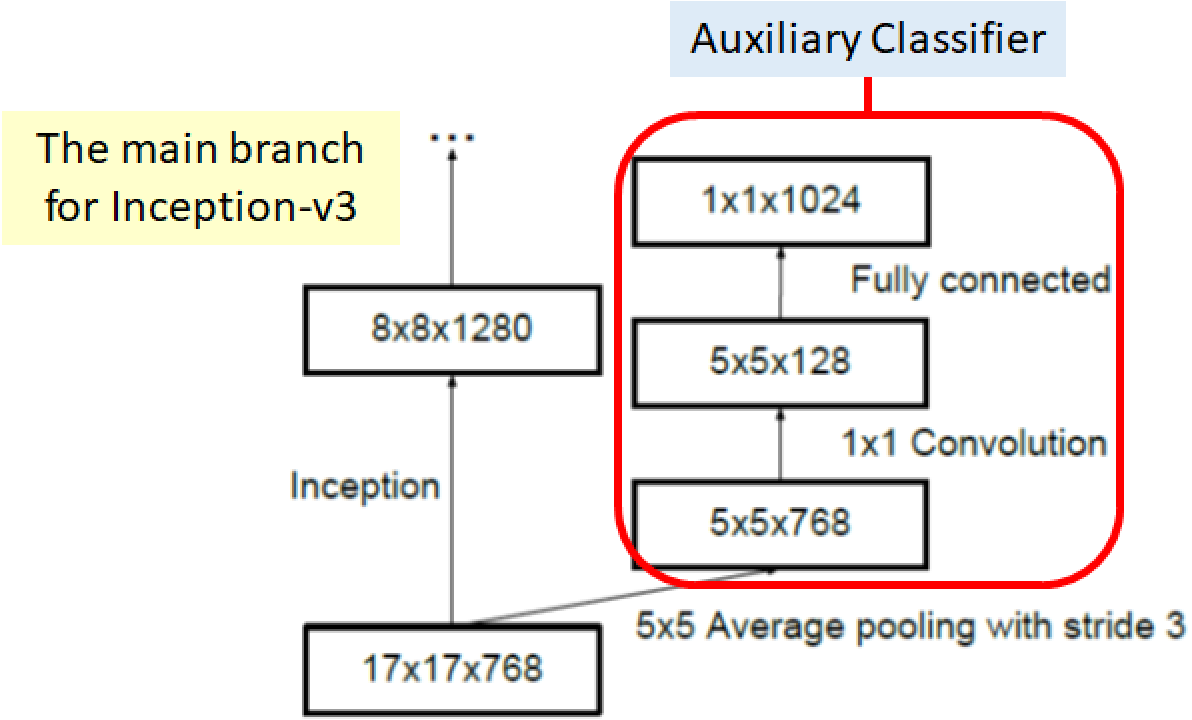
- **Factorization Into Asymmetric Convolutions**
- Factorization Into Asymmetric Convolutions
- With this technique, one of the new Inception modules (I call it Inception Module B here) becomes:



Inception Module B using asymmetric factorization(Inception-V3)

Inception v2 and V3

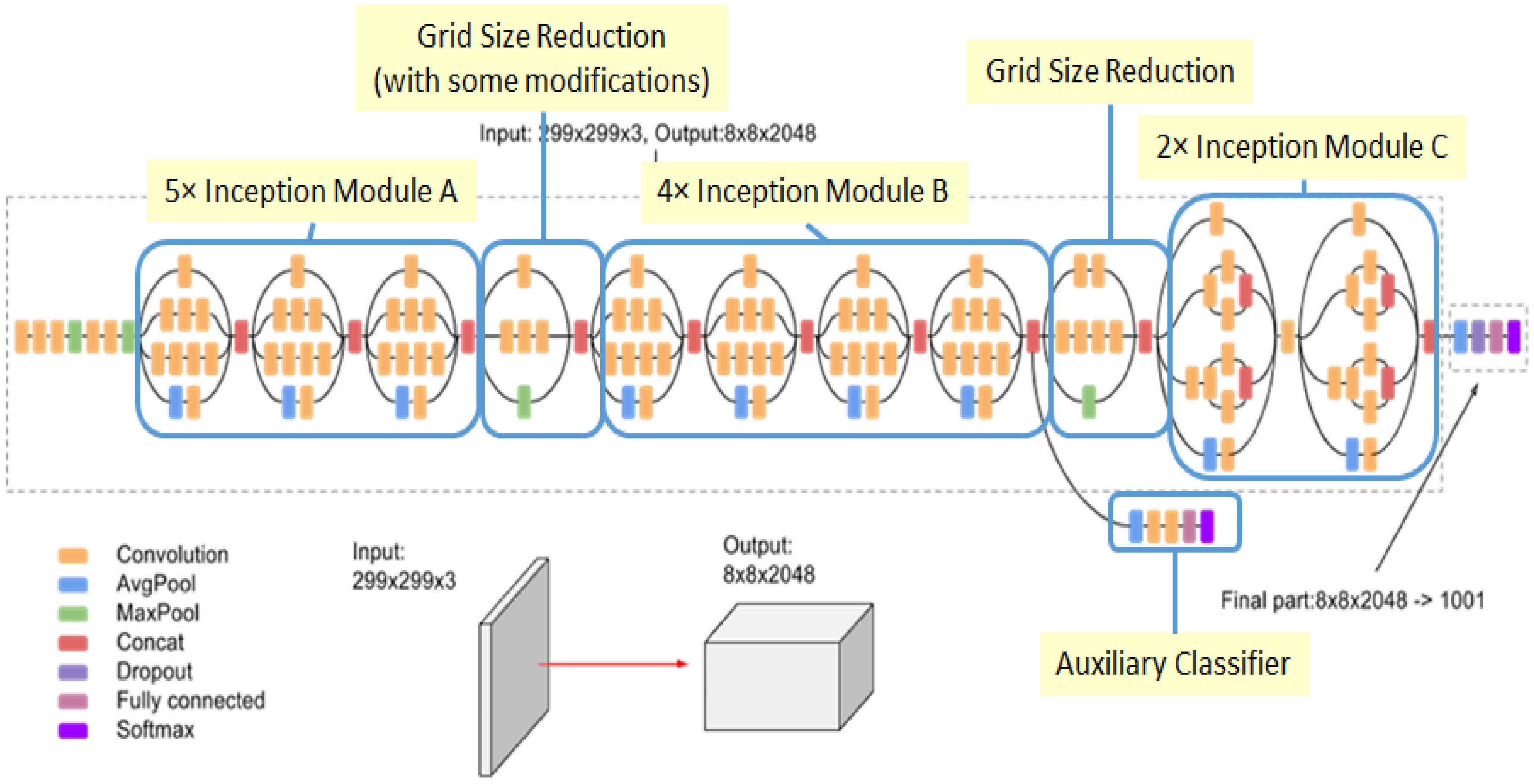
- **Inception v2 and V3**
- Auxiliary Classifiers were already suggested in GoogLeNet / Inception-v1 [4]. There are some modifications in Inception-v3.
- Only 1 auxiliary classifier is used on the top of the last 17×17 layer, instead of using 2 auxiliary classifiers. (The overall architecture would be shown later.)



The purpose is also different. In GoogLeNet / Inception-v1 [4], auxiliary classifiers are used for having deeper network. In Inception-v3, auxiliary classifier is used as regularizer. So, actually, in deep learning, the modules are still quite intuitive.

InceptionV3

- **InceptionV3**



With **42** layers deep, the computation cost is only about 2.5 higher than that of GoogLeNet [4], and much more efficient than that of VGGNet [3].

Inception v2

- **Inception v2**

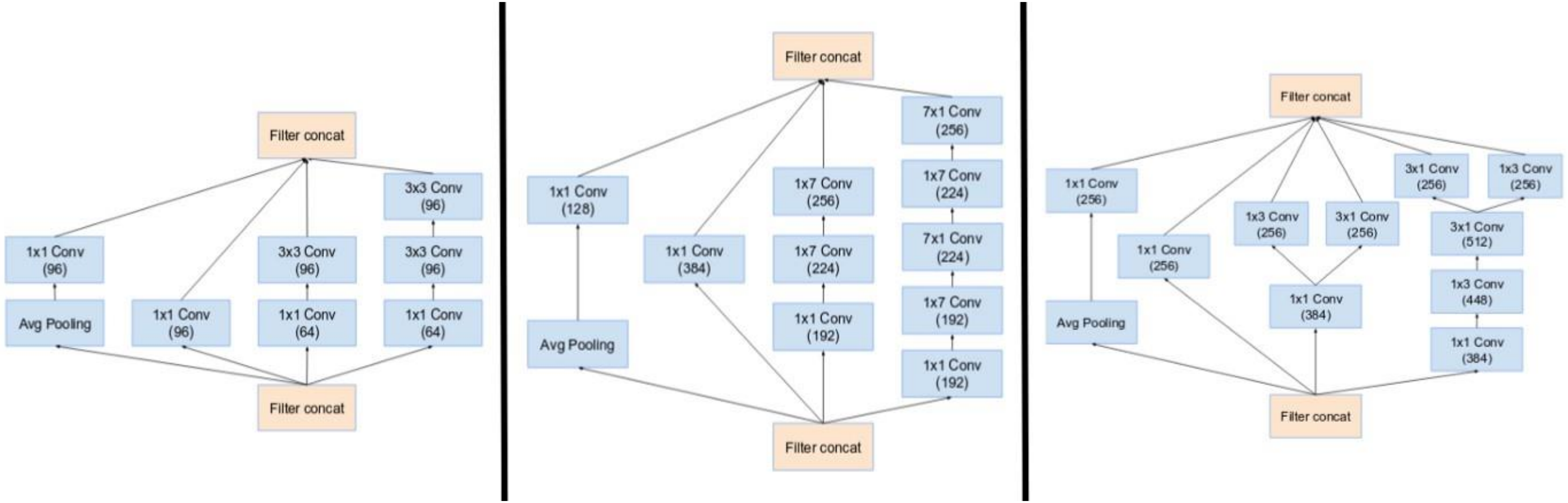
type	patch size/stride or remarks	input size
conv	$3 \times 3/2$	$299 \times 299 \times 3$
conv	$3 \times 3/1$	$149 \times 149 \times 32$
conv padded	$3 \times 3/1$	$147 \times 147 \times 32$
pool	$3 \times 3/2$	$147 \times 147 \times 64$
conv	$3 \times 3/1$	$73 \times 73 \times 64$
conv	$3 \times 3/2$	$71 \times 71 \times 80$
conv	$3 \times 3/1$	$35 \times 35 \times 192$
$3 \times$ Inception	As in figure $\boxed{5}$	$35 \times 35 \times 288$
$5 \times$ Inception	As in figure $\boxed{6}$	$17 \times 17 \times 768$
$2 \times$ Inception	As in figure $\boxed{7}$	$8 \times 8 \times 1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

Here, "figure 5" is module A, "figure 6" is module B and "figure 7" is module C. (Source: Incpetion v2)

Inception-V4

- **Inception-v4**

https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

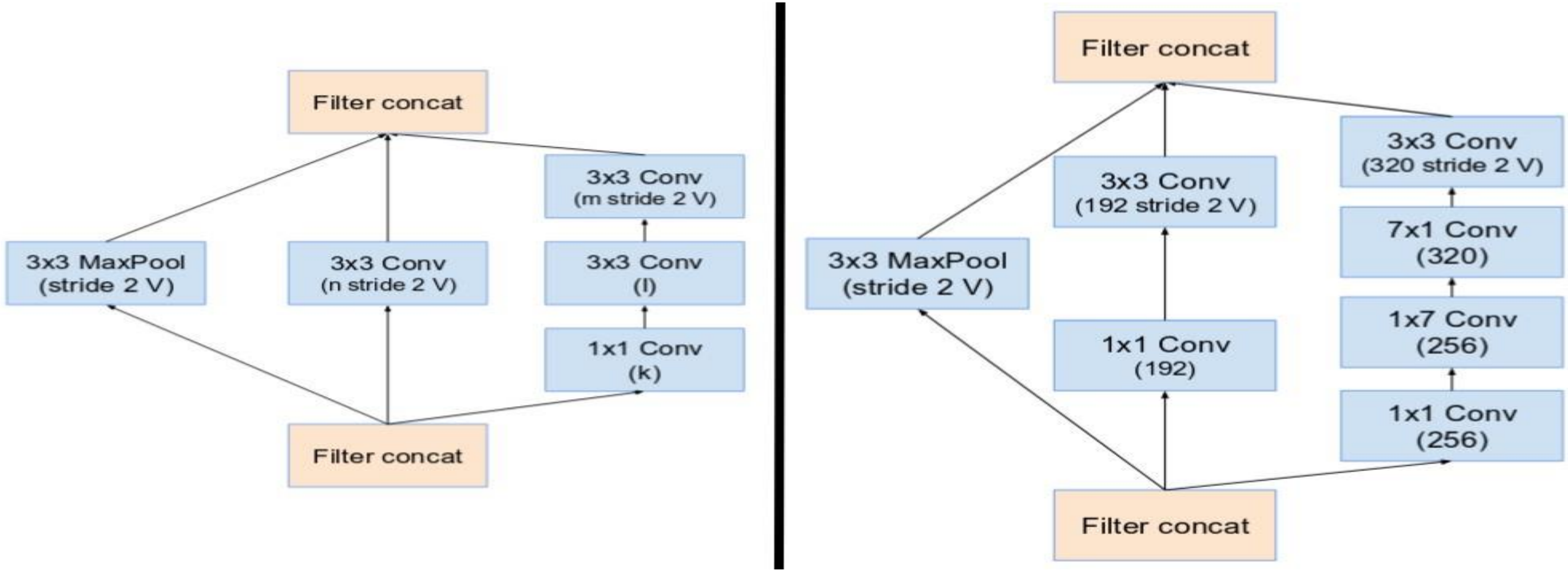


(From left) Inception modules A,B,C used in Inception v4. Note how similar they are to the Inception v2 (or v3) modules. (Source: Inception v4)

Inception-V4

https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

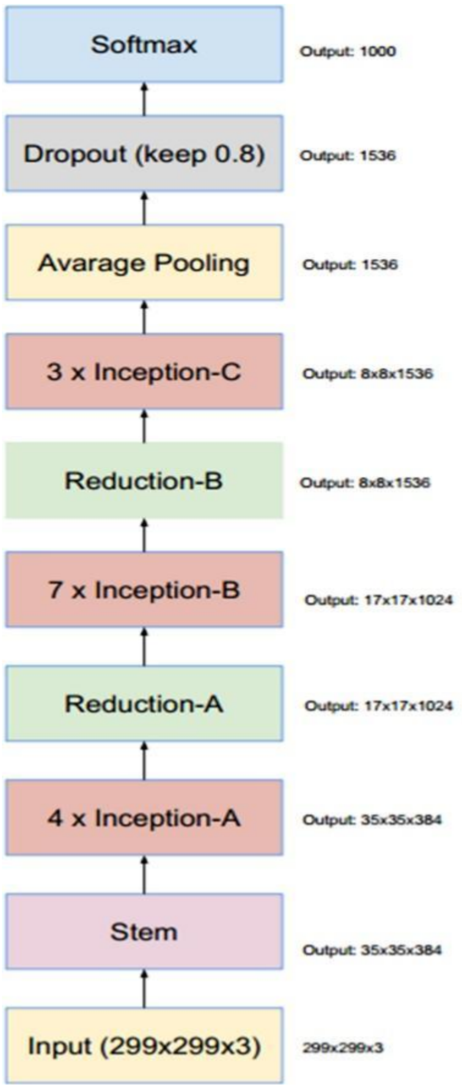
- **Inception-v4**



From Left) Reduction Block A (35x35 to 17x17 size reduction) and Reduction Block B (17x17 to 8x8 size reduction). Refer to the paper for the exact hyper-parameter setting (V,l,k). (Source: Inception v4)

Inception-V4

- **Inception-v4**



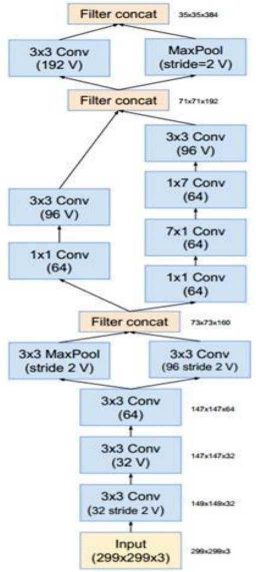


Figure 3. The schema for stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. Cf. Figures 9 and 15

Inception-V4

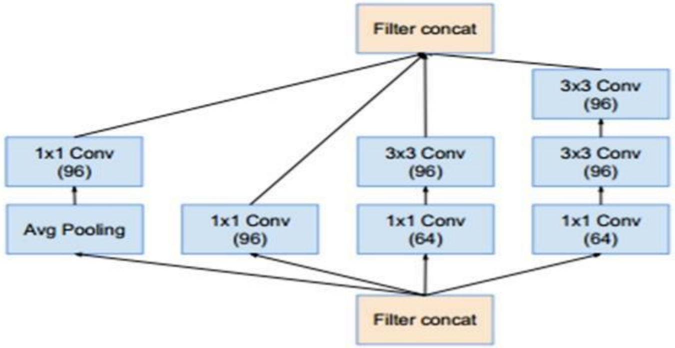


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

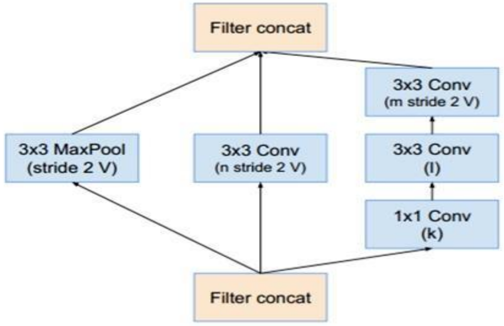


Figure 7. The schema for 35 × 35 to 17 × 17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

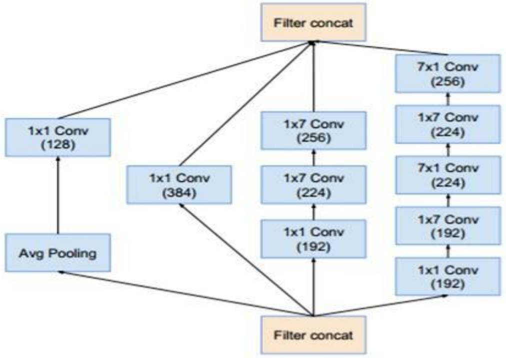


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

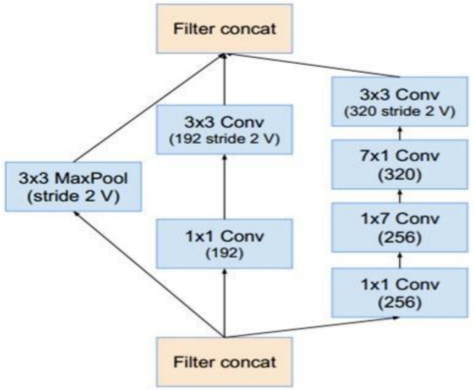


Figure 8. The schema for 17 × 17 to 8 × 8 grid-reduction module. This is the reduction module used by the pure Inception-v4 network in Figure 9.

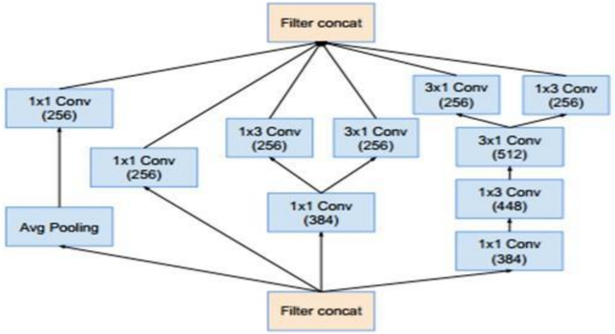
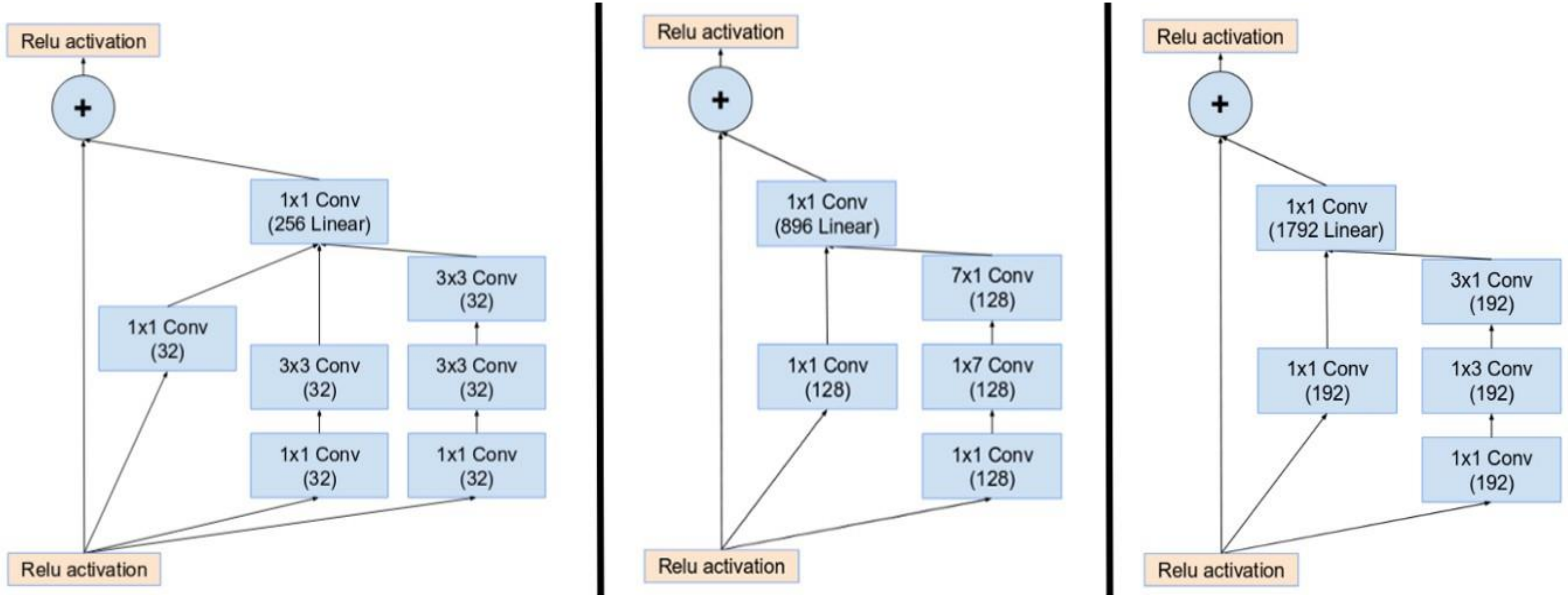


Figure 6. The schema for 8×8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.

Inception-ResNet-V1

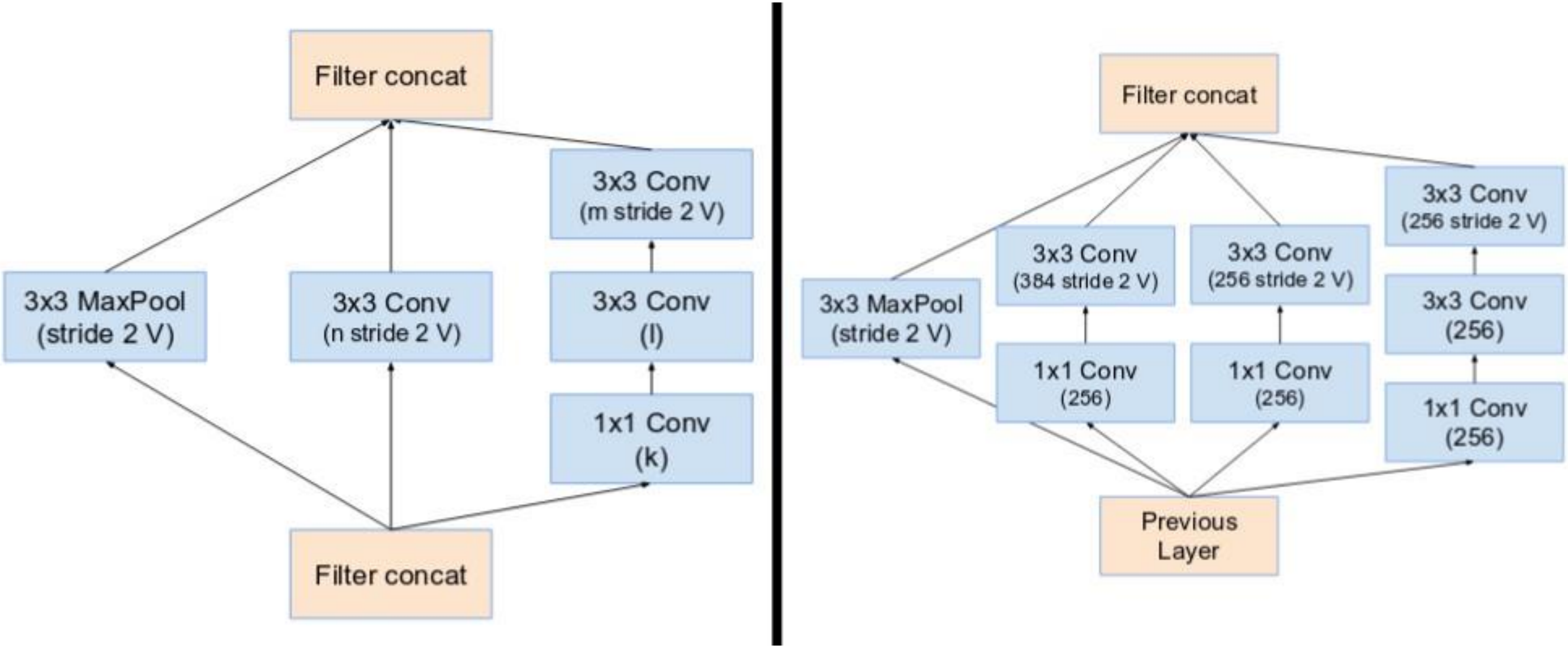
- **Inception-ResNet-V1**



(From left) Inception modules A,B,C in an Inception ResNet. Note how the pooling layer was replaced by the residual connection, and also the additional 1x1 convolution before addition. (Source: Inception v4)

Inception-ResNet-V1

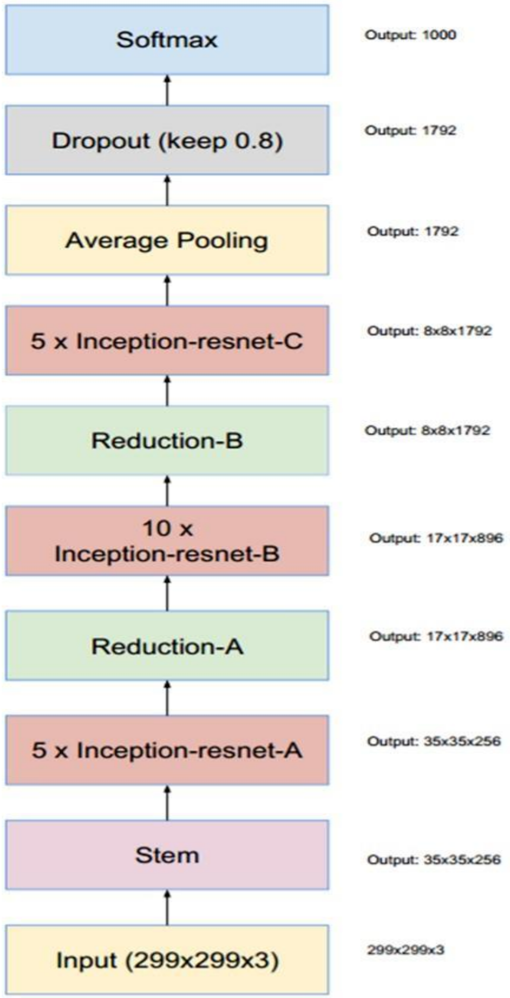
- **Inception-ResNet-V1**



(From Left) Reduction Block A (35x35 to 17x17 size reduction) and Reduction Block B (17x17 to 8x8 size reduction). Refer to the paper for the exact hyper-parameter setting (V,l,k). (Source: Inception v4)

Inception-ResNet-V1

- **Inception-ResNet-V1**



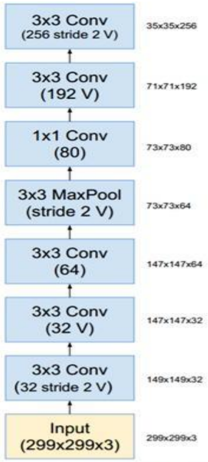


Figure 14. The stem of the Inception-ResNet-v1 network.

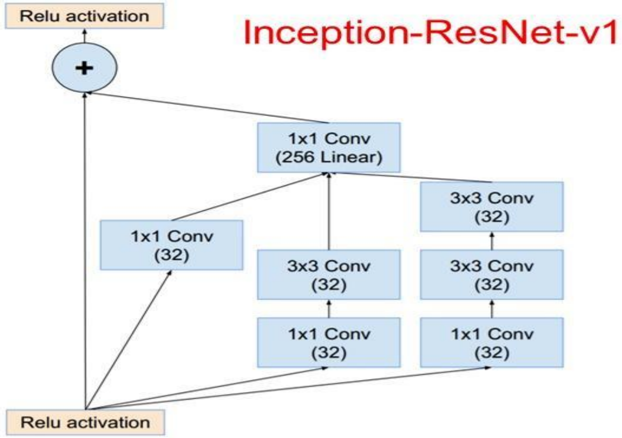


Figure 10. The schema for 35 × 35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.

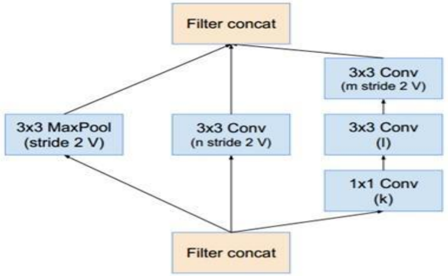


Figure 7. The schema for 35 × 35 to 17 × 17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

Network	k	l	m	n
Inception-v4	192	224	256	384
Inception-ResNet-v1	192	192	256	384
Inception-ResNet-v2	256	256	384	384

Table 1. The number of filters of the Reduction-A module for the three Inception variants presented in this paper. The four numbers in the colums of the paper parametrize the four convolutions of Figure 7.

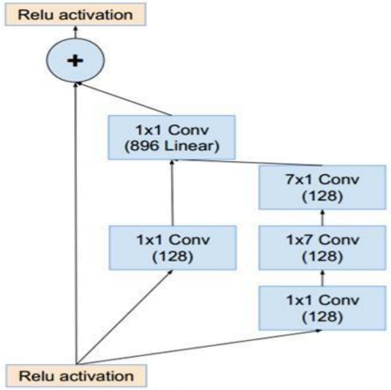


Figure 11. The schema for 17 × 17 grid (Inception-ResNet-B) module of Inception-ResNet-v1 network.

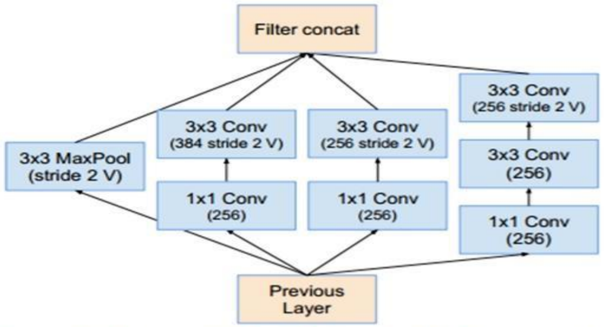


Figure 12. "Reduction-B" 17 × 17 to 8 × 8 grid-reduction module. This module used by the smaller Inception-ResNet-v1 network in Figure 15.

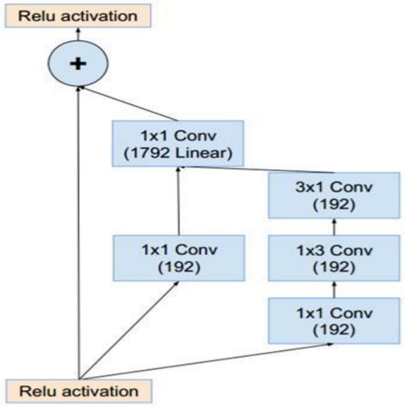
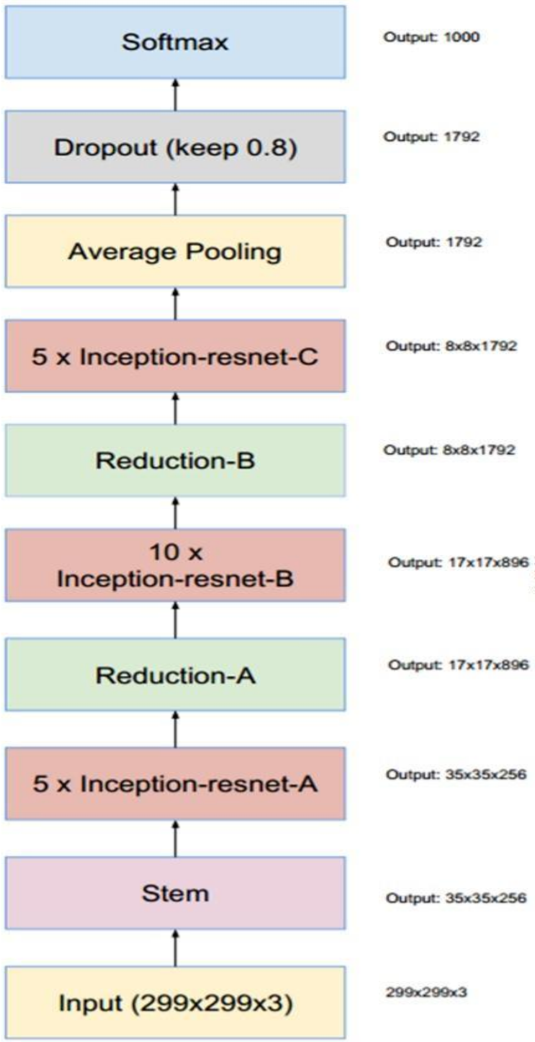


Figure 13. The schema for 8×8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network.

Inception-ResNet-V2

- **Inception-ResNet-V2**



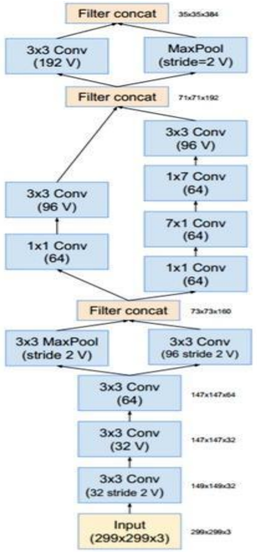


Figure 3. The schema for stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. Cf. Figures 9 and 15

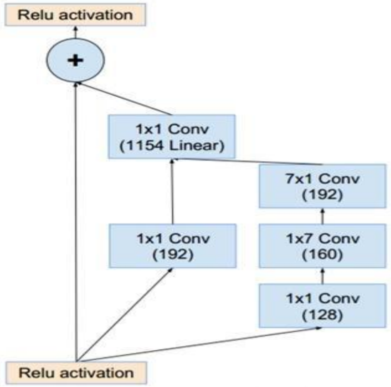


Figure 17. The schema for 17 × 17 grid (Inception-ResNet-B) module of the Inception-ResNet-v2 network.

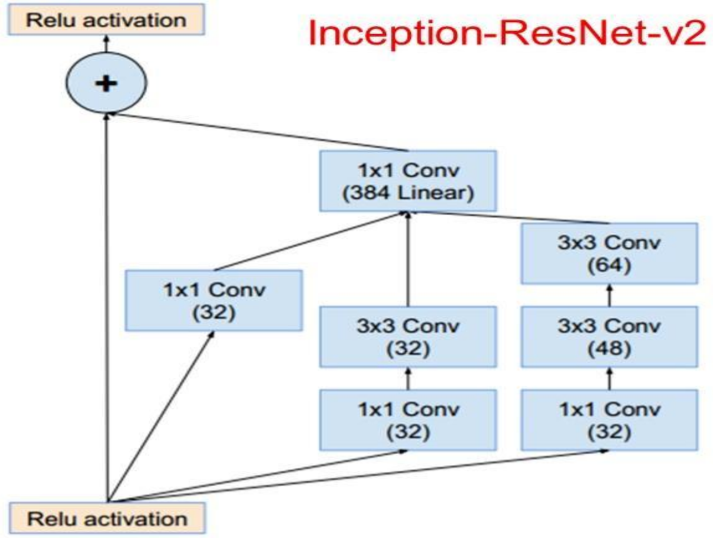


Figure 16. The schema for 35 × 35 grid (Inception-ResNet-A) module of the Inception-ResNet-v2 network.

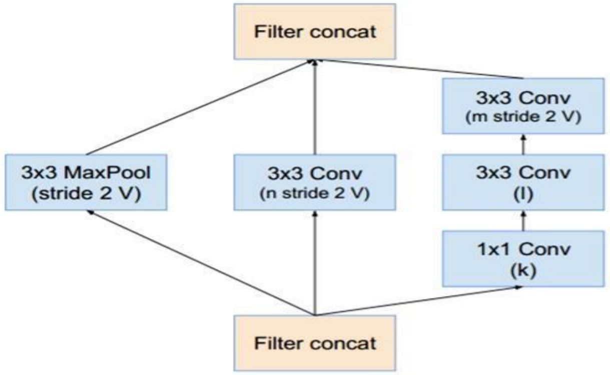


Figure 7. The schema for 35 × 35 to 17 × 17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

Network	k	l	m	n
Inception-v4	192	224	256	384
Inception-ResNet-v1	192	192	256	384
Inception-ResNet-v2	256	256	384	384

Table 1. The number of filters of the Reduction-A module for the three Inception variants presented in this paper. The four numbers in the colums of the paper parametrize the four convolutions of Figure 7

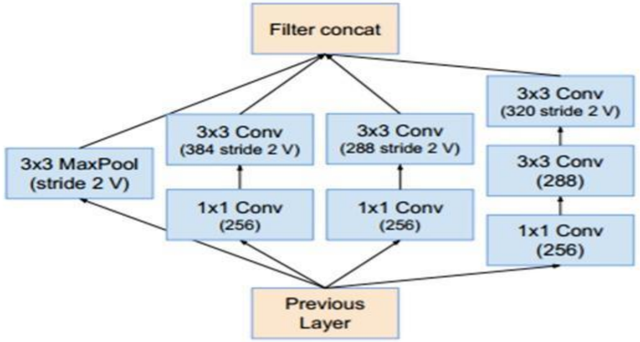


Figure 18. The schema for 17 × 17 to 8 × 8 grid-reduction module. Reduction-B module used by the wider Inception-ResNet-v1 network in Figure 15.

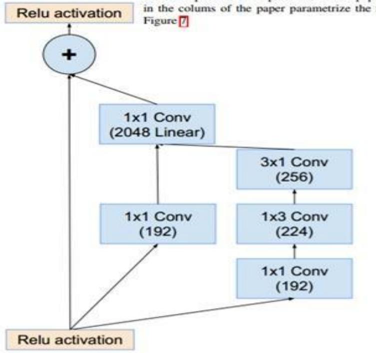
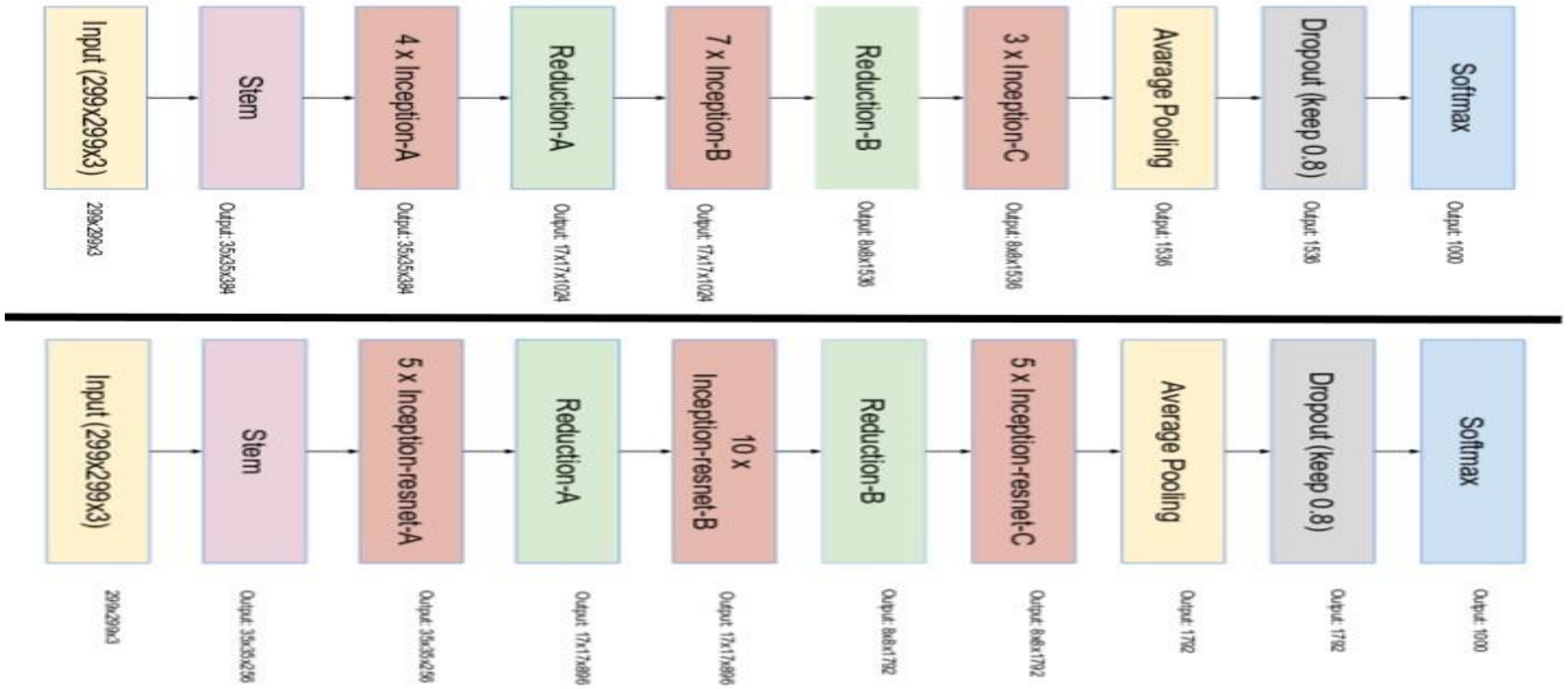


Figure 19. The schema for 8×8 grid (Inception-ResNet-C) module of the Inception-ResNet-v2 network.

Inception-ResNet-V1and V2

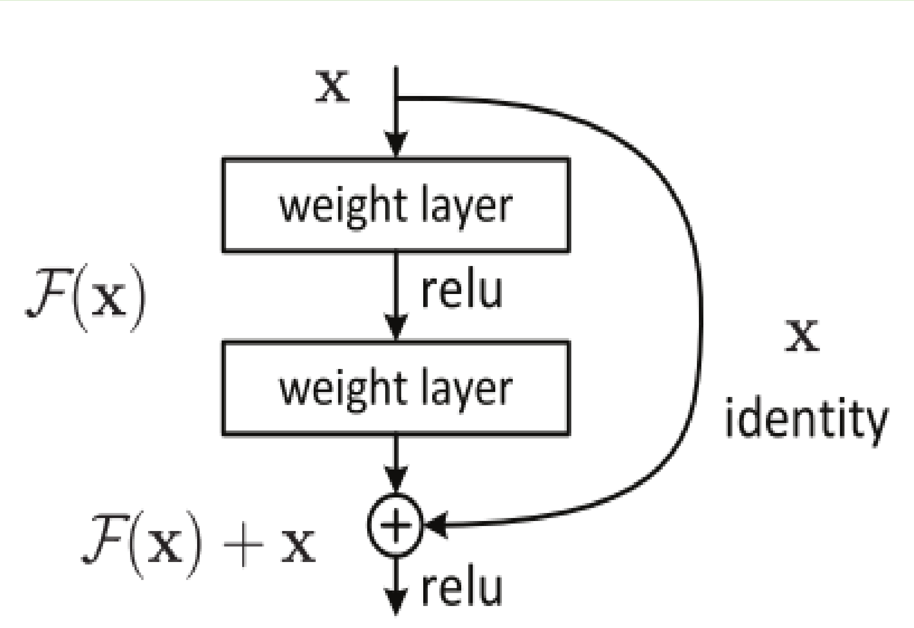
- **Inception-ResNet-V1and V2**



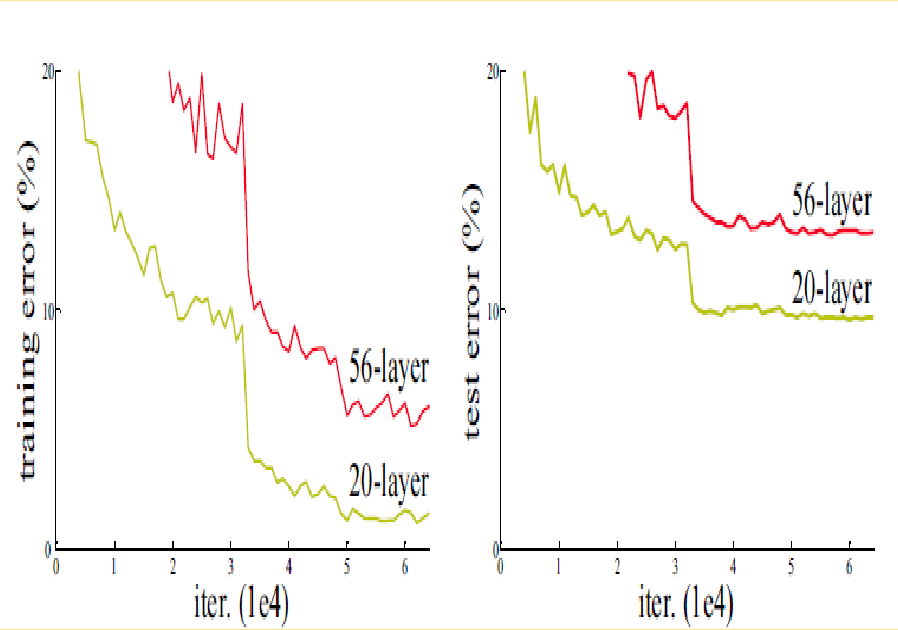
The top image is the layout of Inception v4. The bottom image is the layout of Inception-ResNet. (Source: Inception v4)

ResNet in Computer Vision

Residual Module

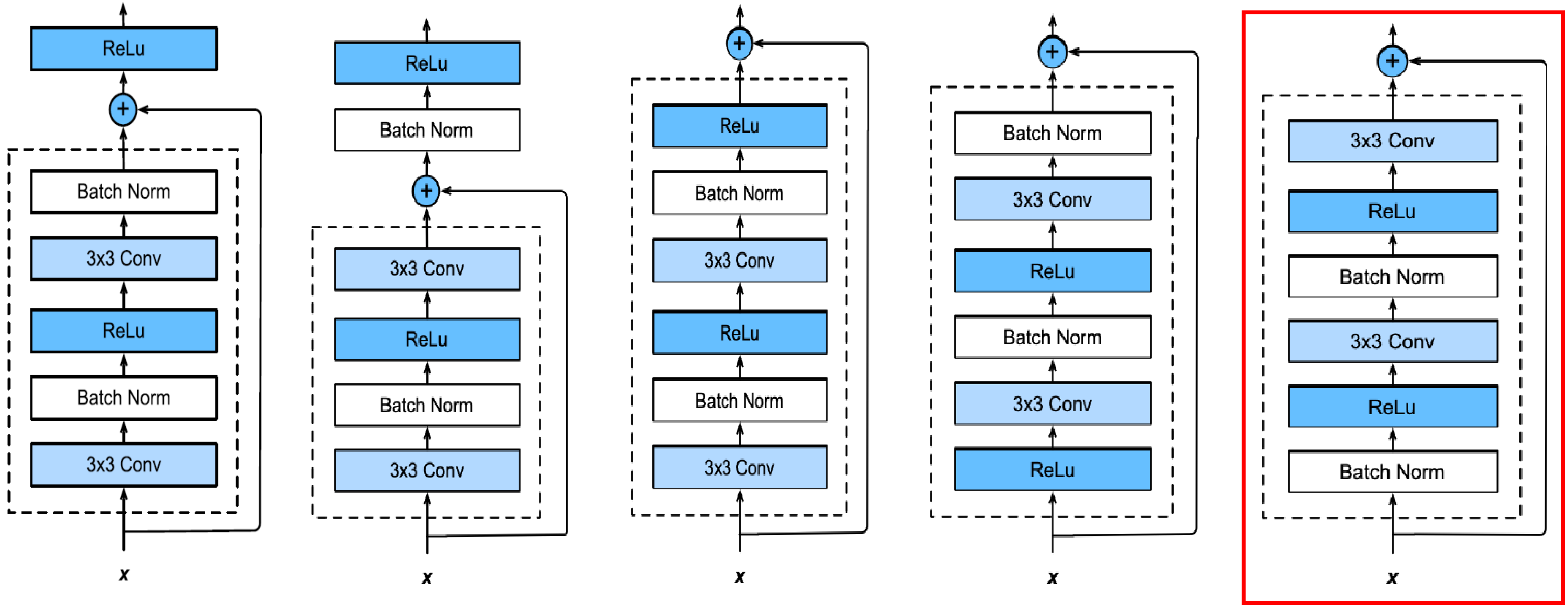


Comparison of Deeper Layers



https://arxiv.org/pdf/1512.03385.pdf

Design Configuration



Original | Batch norm after addition | ReLu last | Batch Norm last | Pre activation

https://arxiv.org/pdf/1603.05027.pdf

ResNet Model

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

<u>ResNet 18, 34</u>

- Two 3x3 Conv layers
- Multiple repeated Residual Blocks
- **4** Major Layers

<u>ResNet50,101,152</u>

- 3 Conv Layers
- Multiple repeated Residual Blocks
- **4** Major Layers

https://arxiv.org/pdf/1512.03385.pdf

ResNet 34

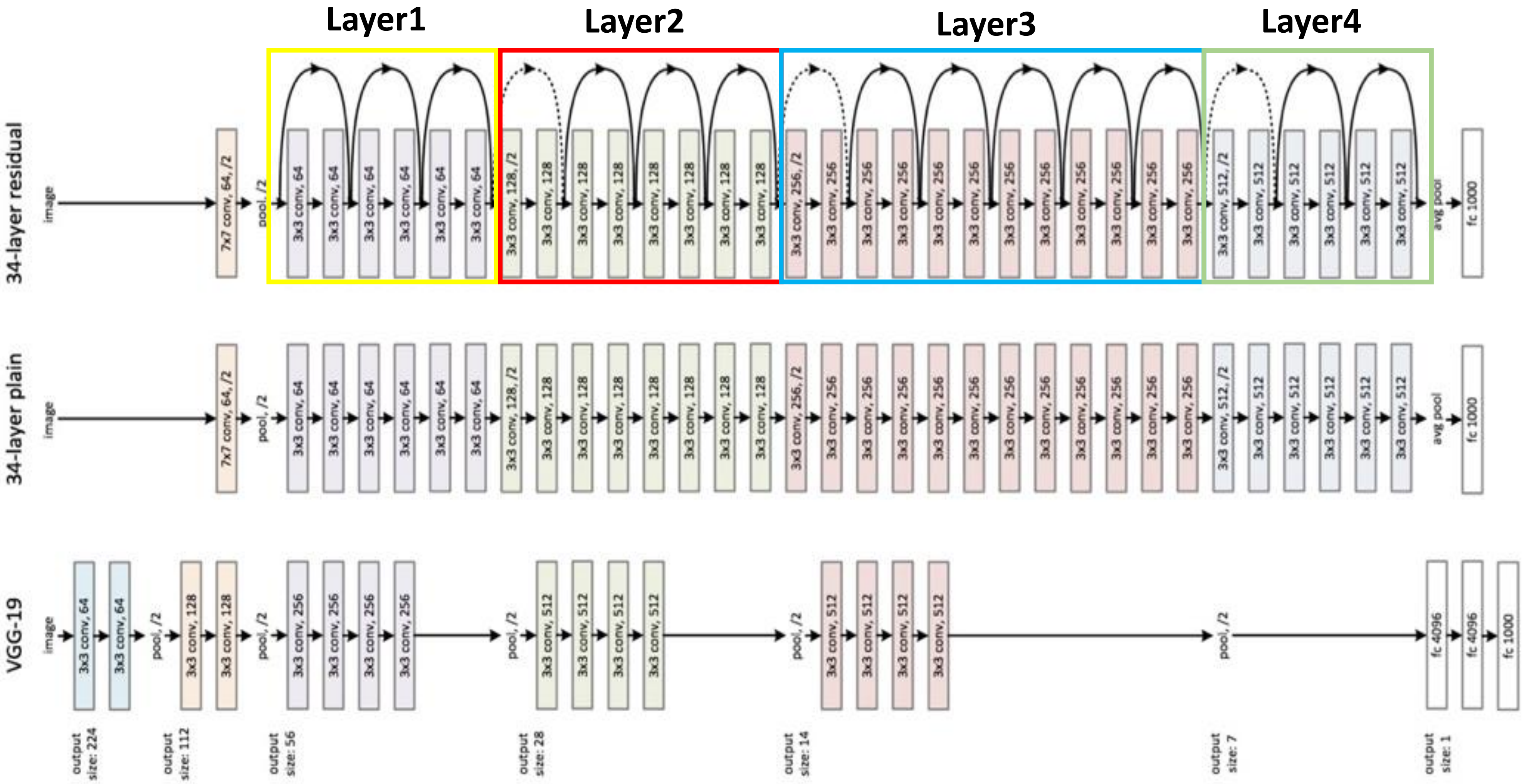
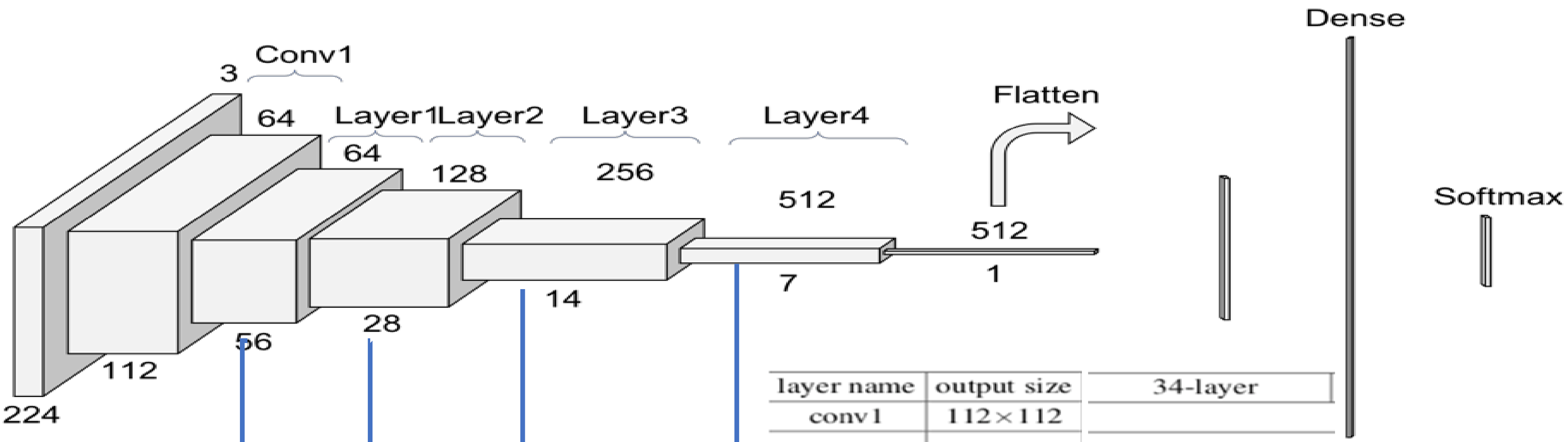


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

https://arxiv.org/pdf/1512.03385.pdf

ResNet 34



layer name	output size	34-layer
conv1	112×112	
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$
	1×1	ave
FLOPs		3.6×10^9

Layer1(**3** Residual Blocks)

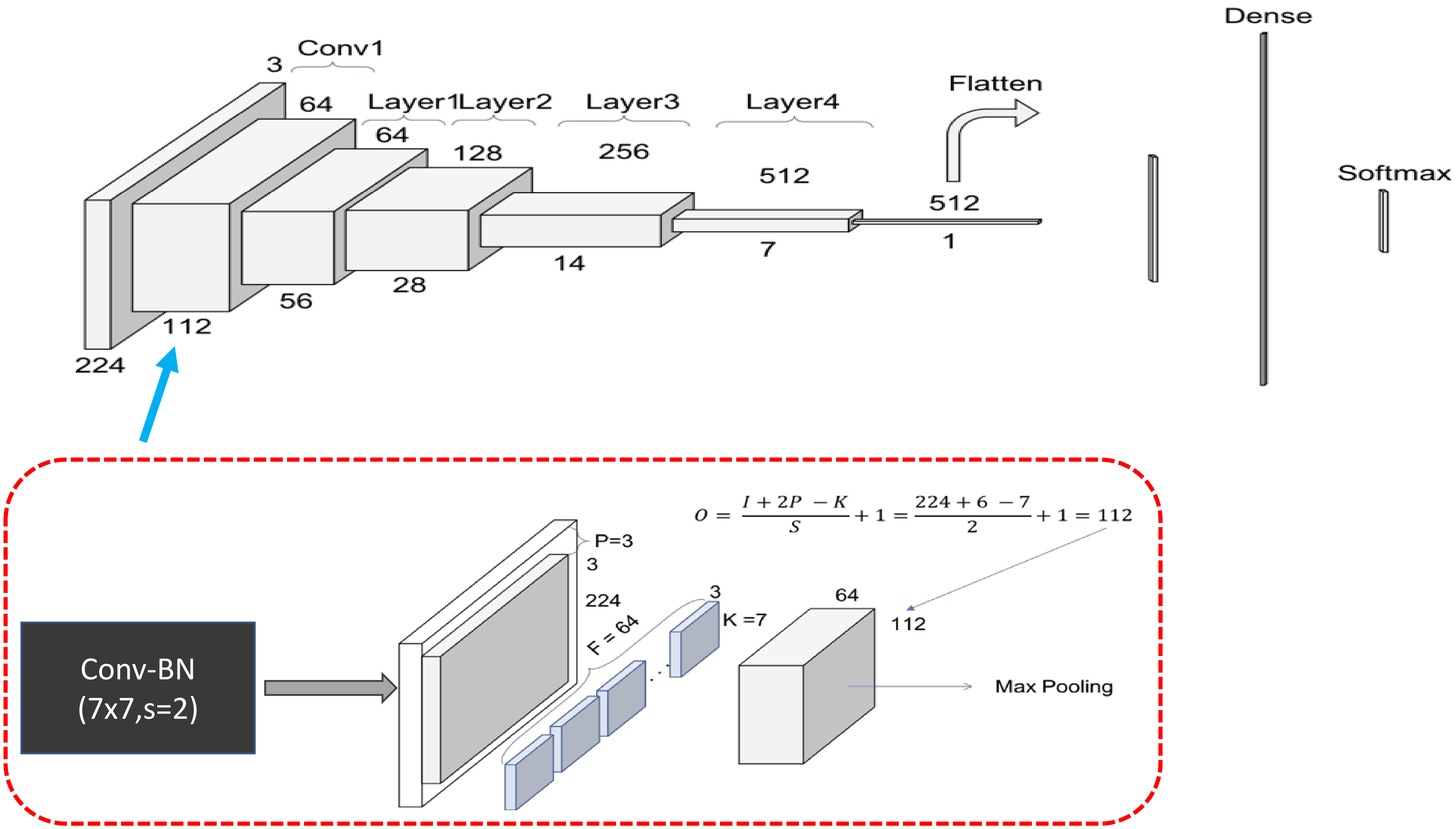
Layer2 (**4** Residual Blocks)

Layer3 (**6** Residual Blocks)

Layer4 (**3** Residual Blocks)

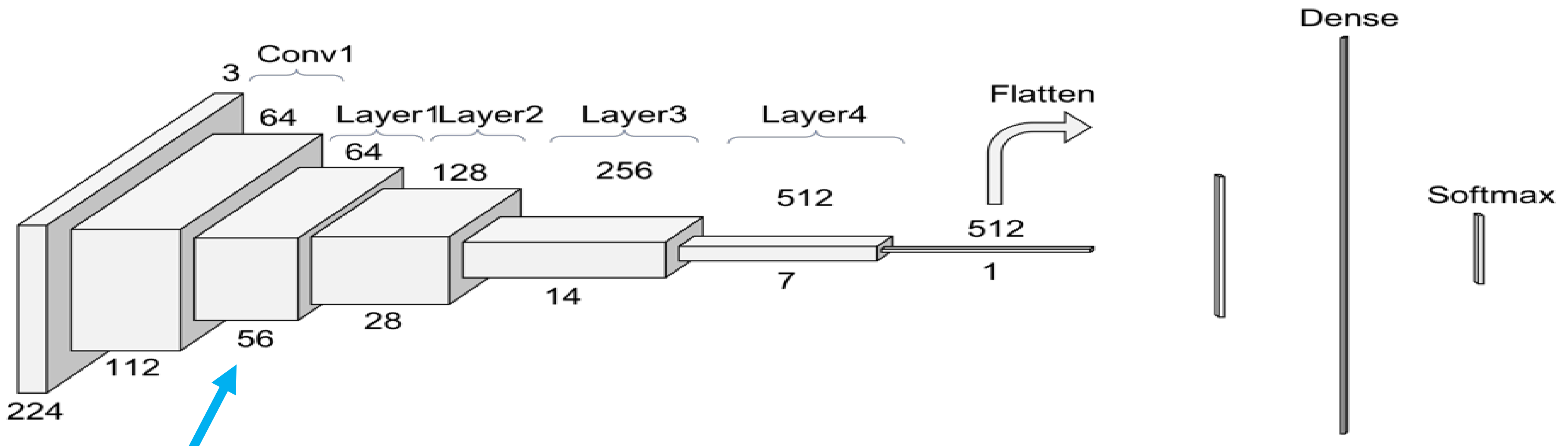
Every Residual Block consisted of 2*(3x3 Conv Layers using skip connection) with different depth or feature maps (**64, 128, 256, 512**)

ResNet 34

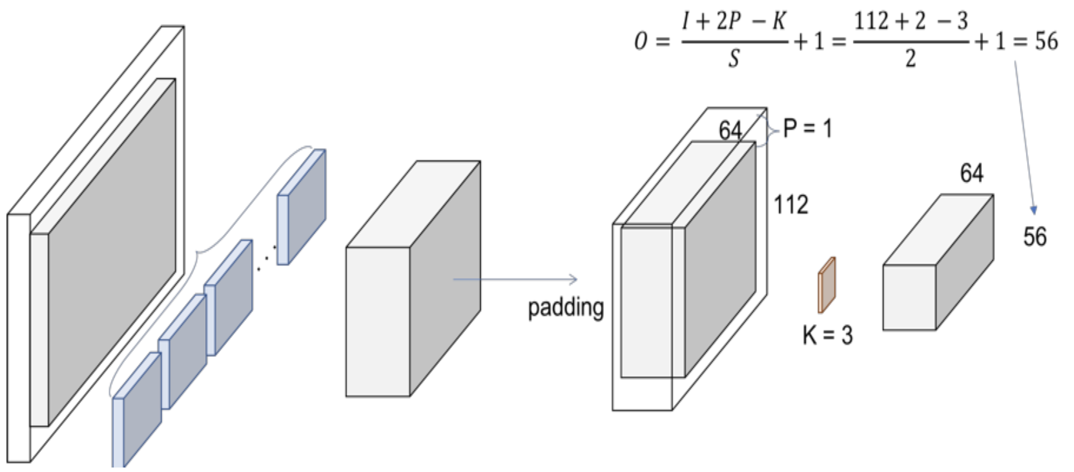


Conv1
3
64
Layer1
Layer2
Layer3
Layer4
64
128
256
512
Flatten
512
1
7
14
28
56
112
224
Dense
Softmax
Conv-BN
(7x7,s=2)
P=3
3
224
F = 64
3
K =7
64
112
$O = \frac{I + 2P - K}{S} + 1 = \frac{224 + 6 - 7}{2} + 1 = 112$
Max Pooling

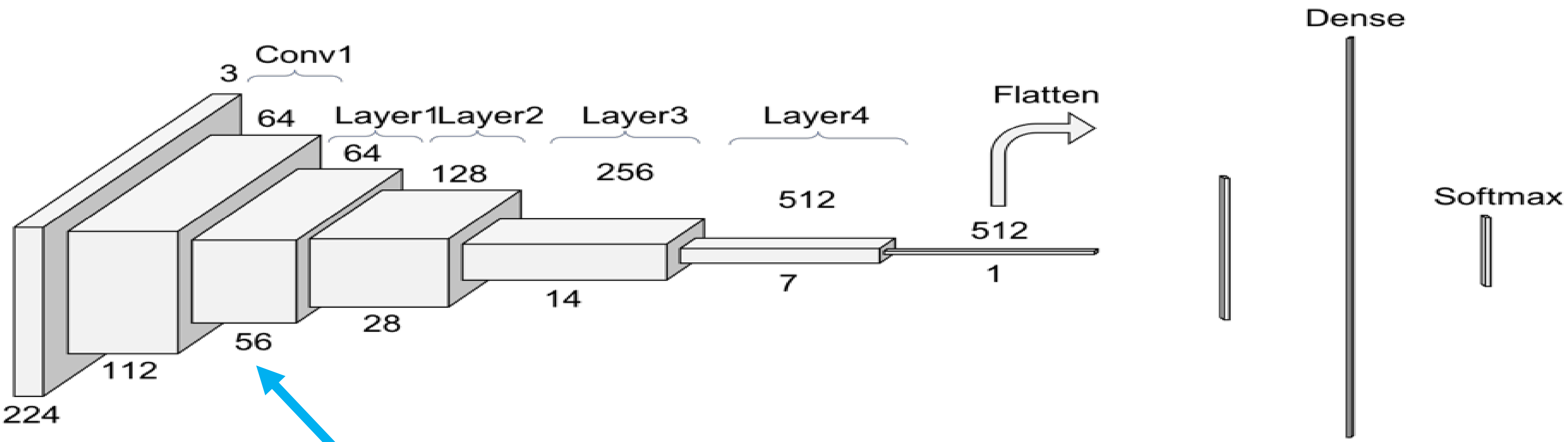
ResNet 34

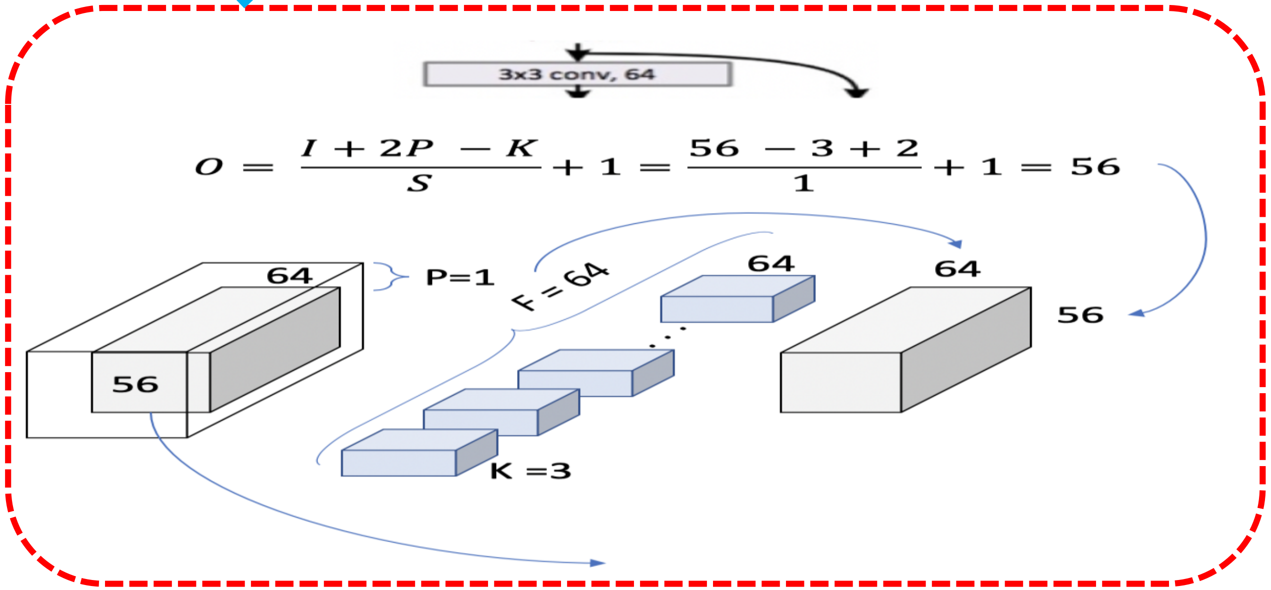


Max-Pool(3x3,s=2)



ResNet 34

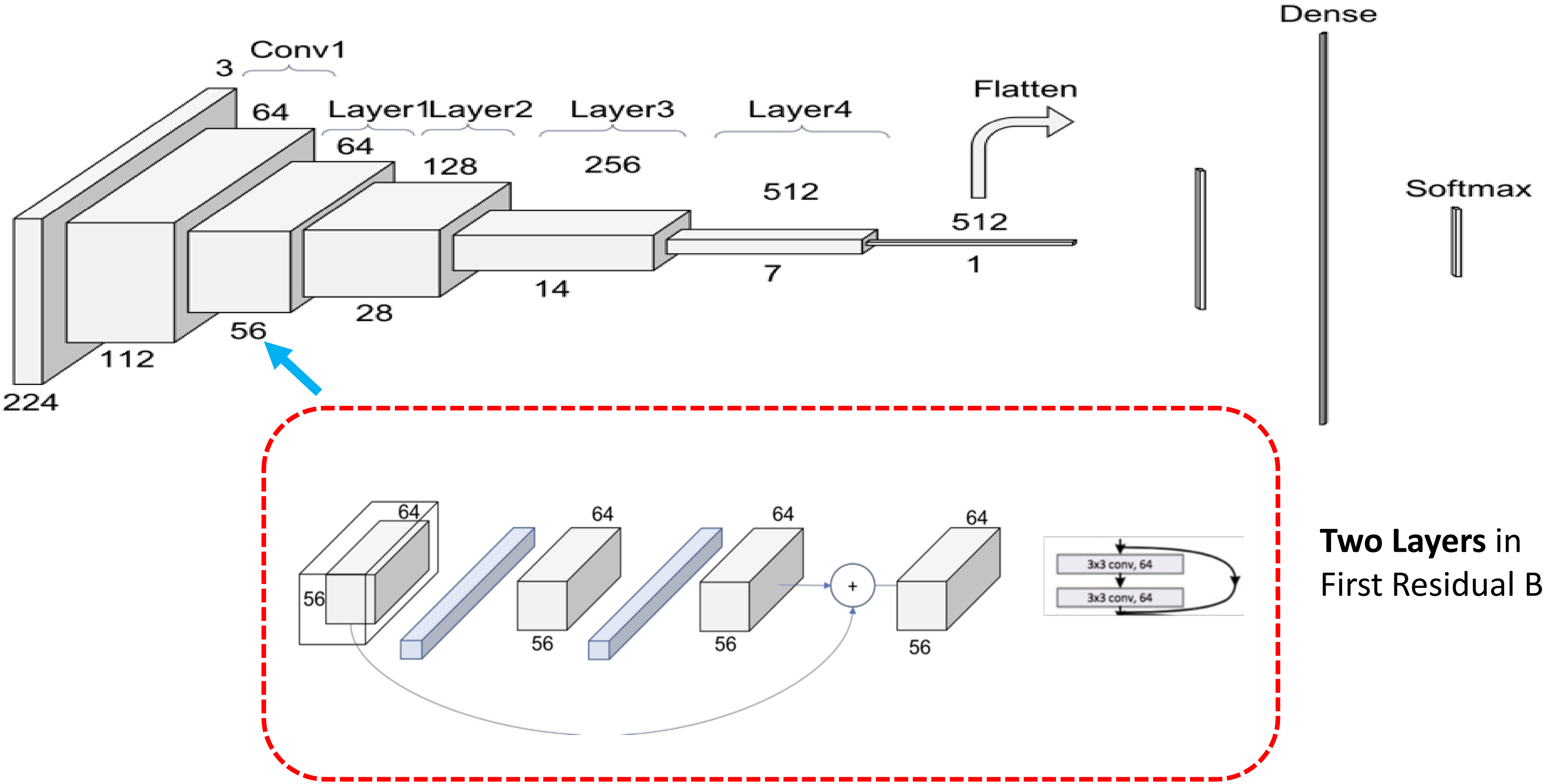




$$O = \frac{I + 2P - K}{S} + 1 = \frac{56 - 3 + 2}{1} + 1 = 56$$

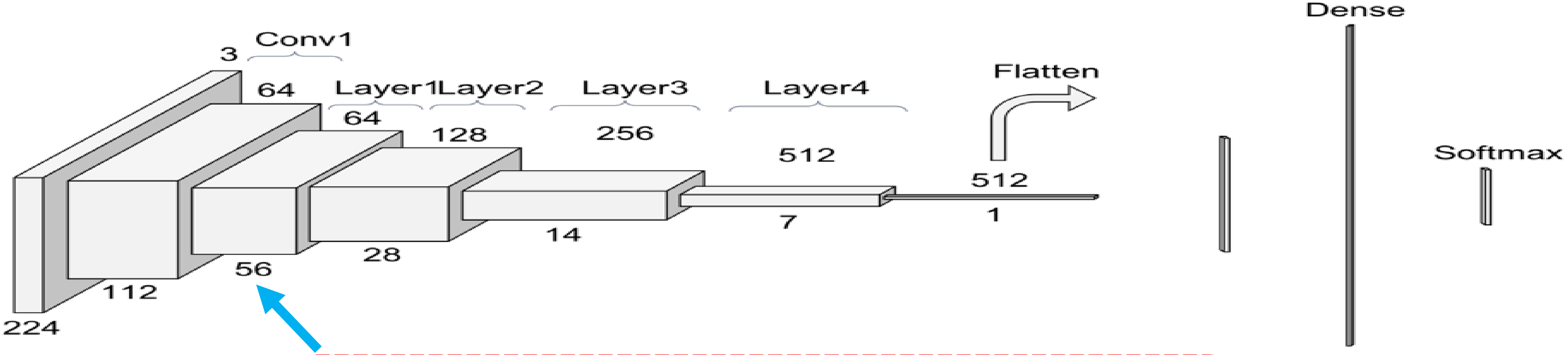
First Layer of residual Block in **Layer1**

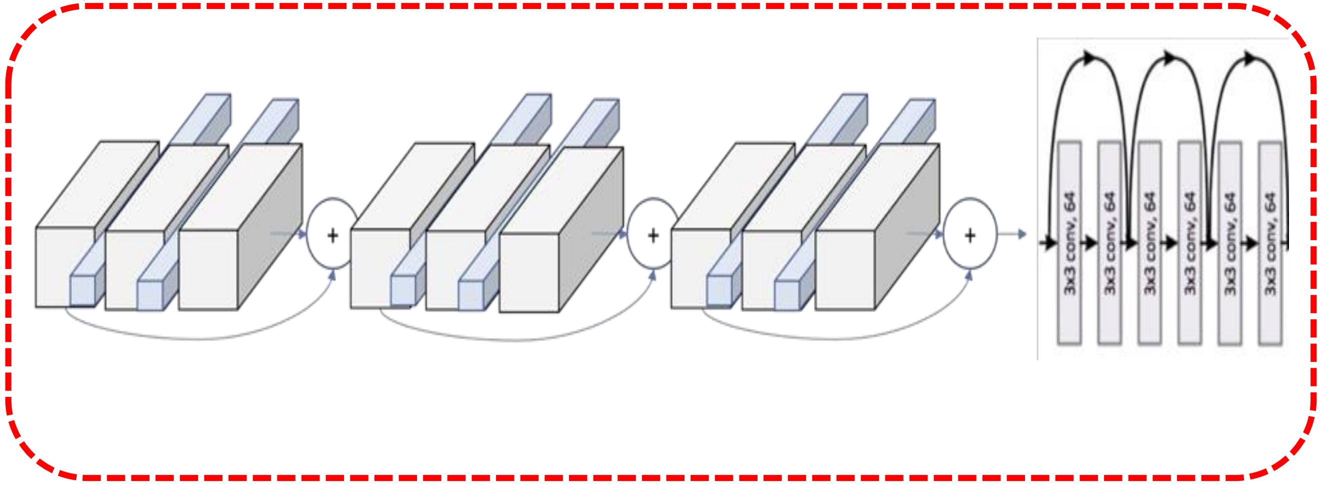
ResNet 34



Two Layers in
First Residual Block in **Layer1**

ResNet 34

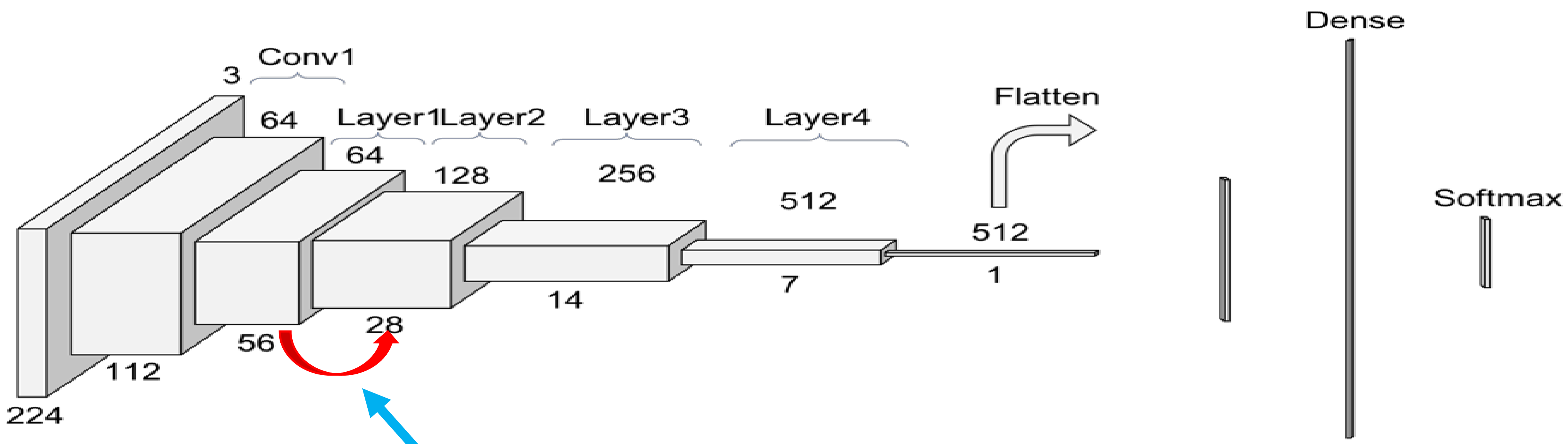




layer name	output size	34-layer
conv1	112×112	
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$
	1×1	ave
FLOPs		3.6×10^9

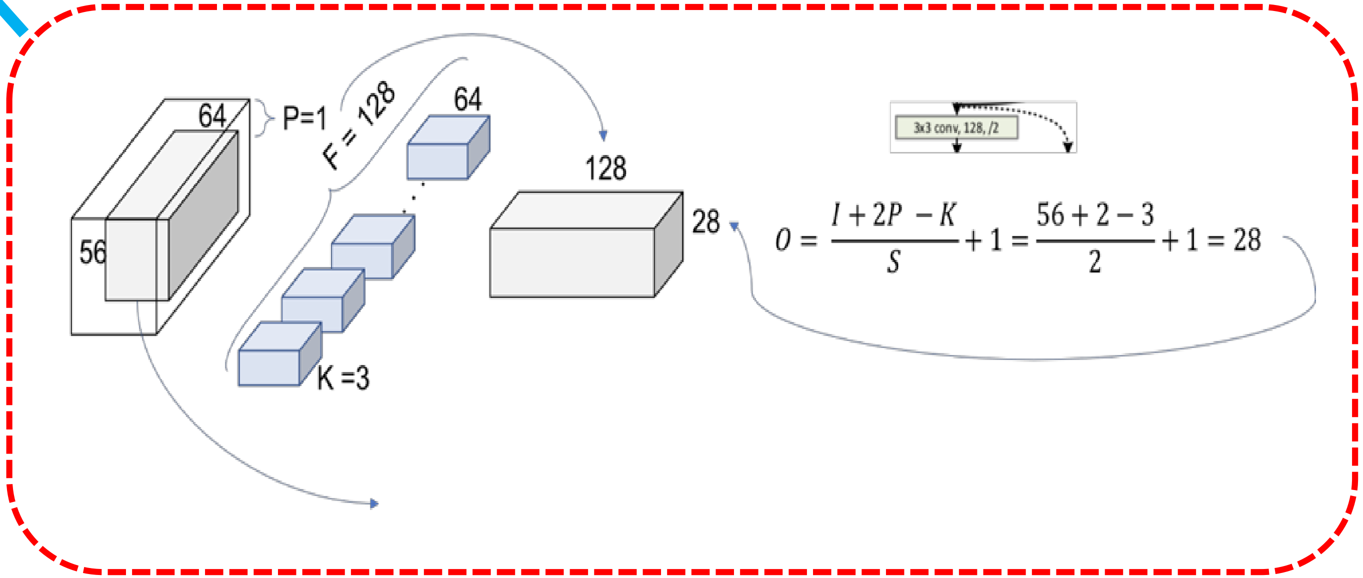
First Layer (Layer1)
3 residual blocks (every Residual block consisted of two 3x3 Conv layer with skip connection)

ResNet 34

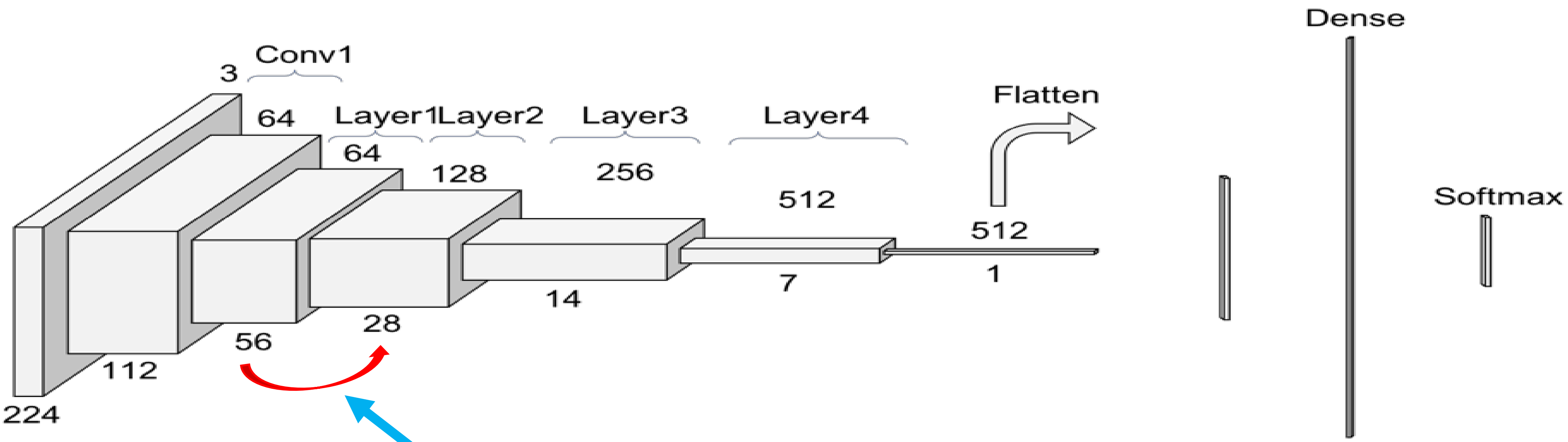


<u>Stride Conv Layer</u>

- 3x3 Conv layer with stride=2, P=1
- Down-sample the spatial dimension From 56-28,
- Depth from 64-128

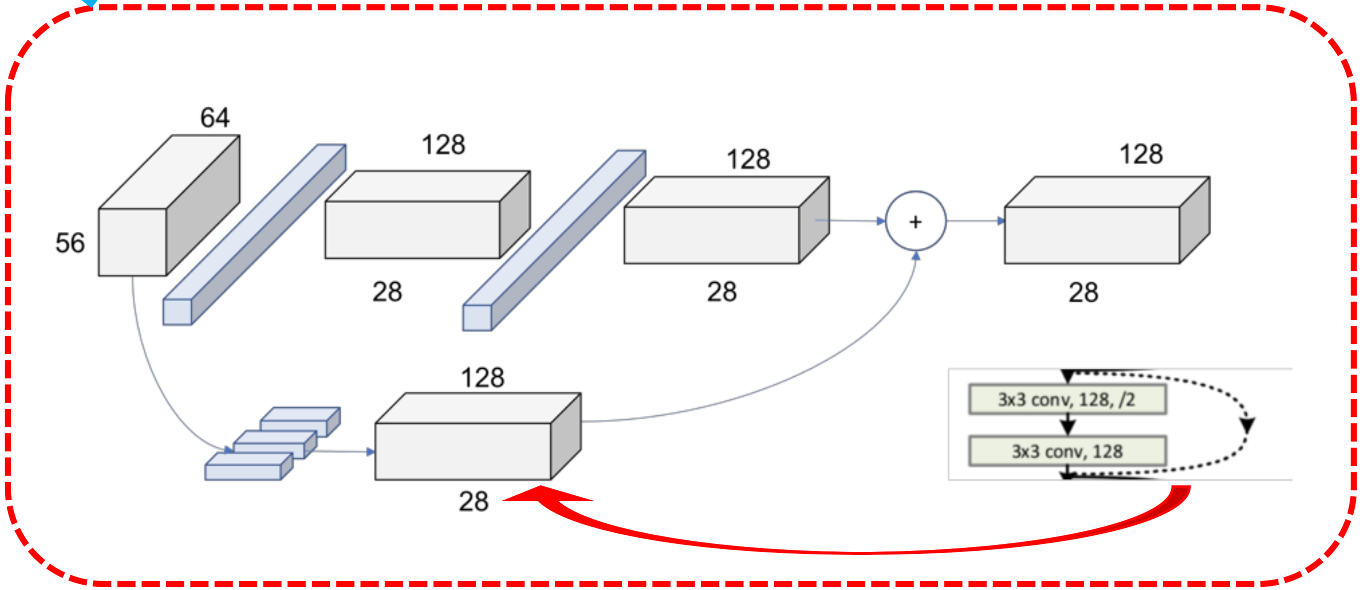


ResNet 34

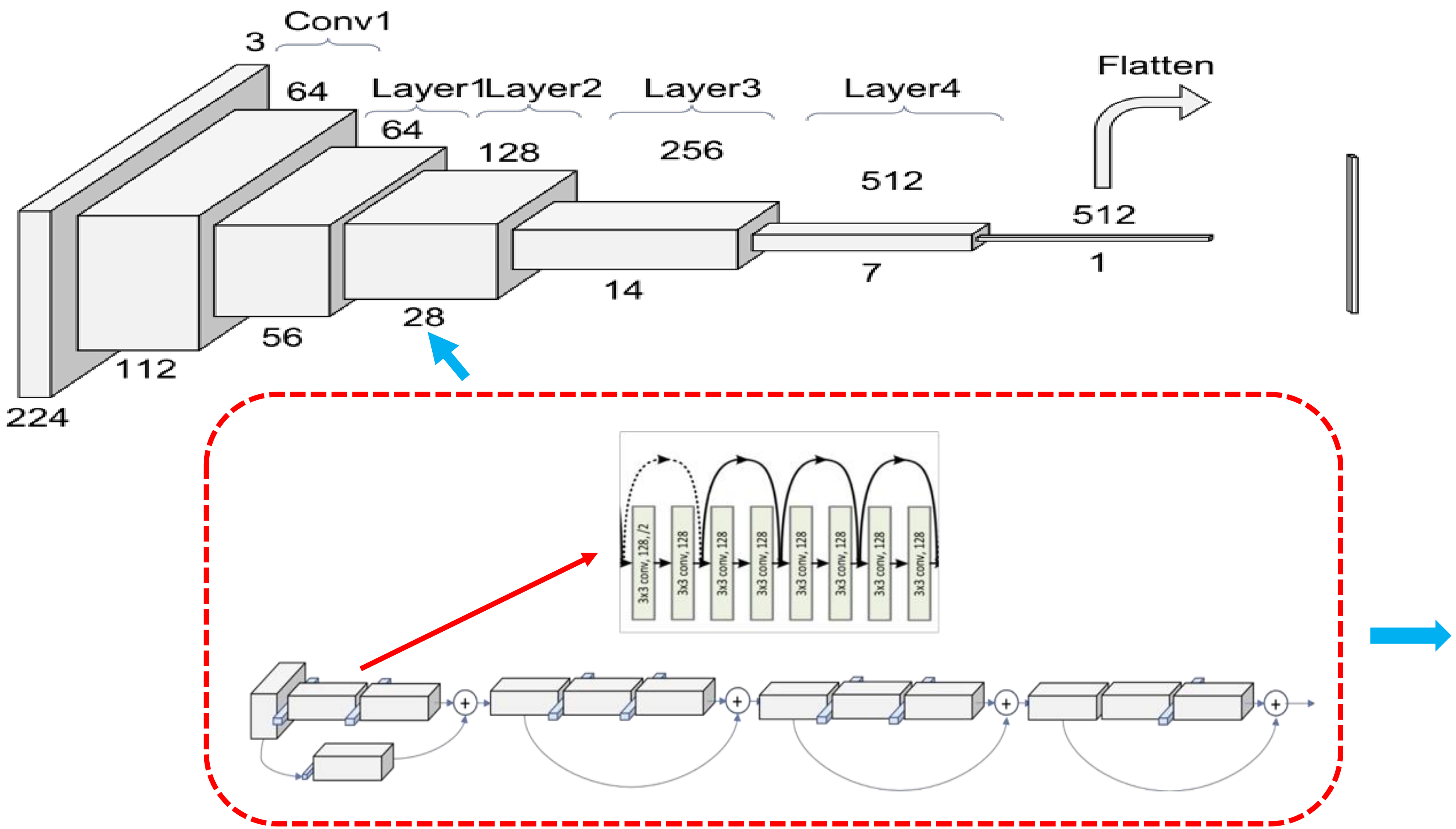


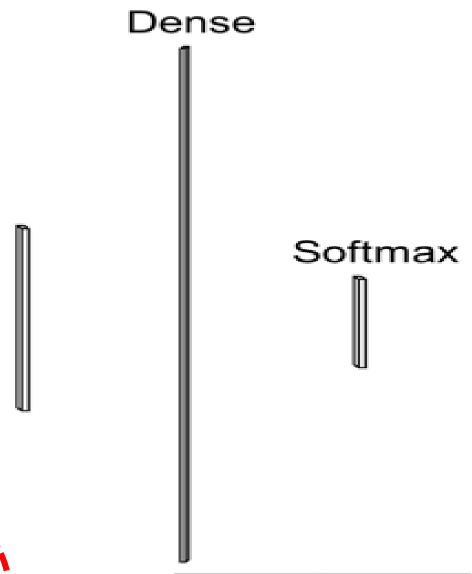
<u>Transition Residual Block</u>

Introduced 3x3 stride Conv layer (56-28 and 64-128) in Identity branch



ResNet 34

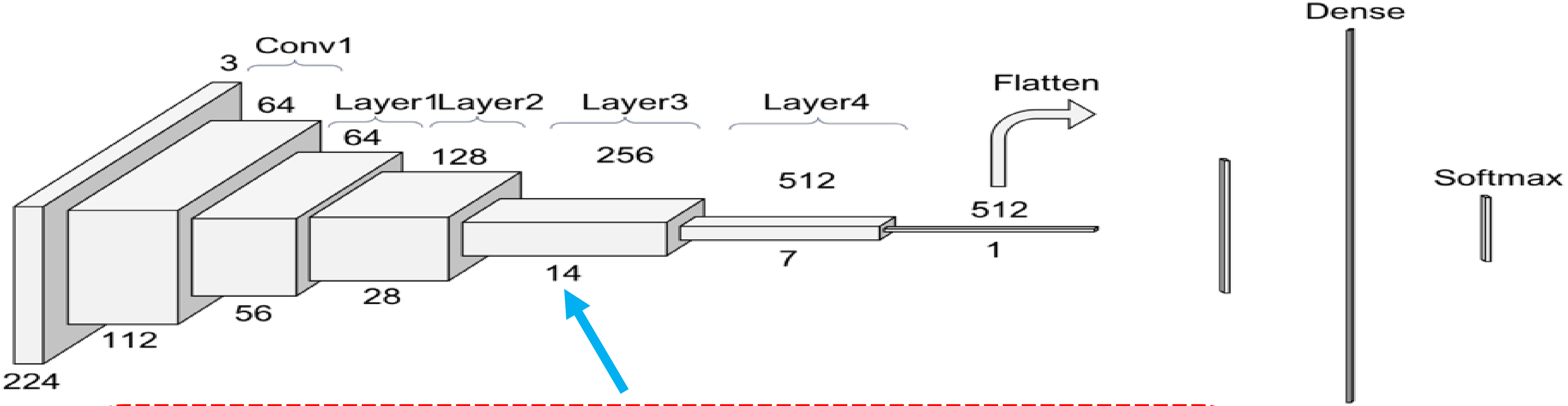


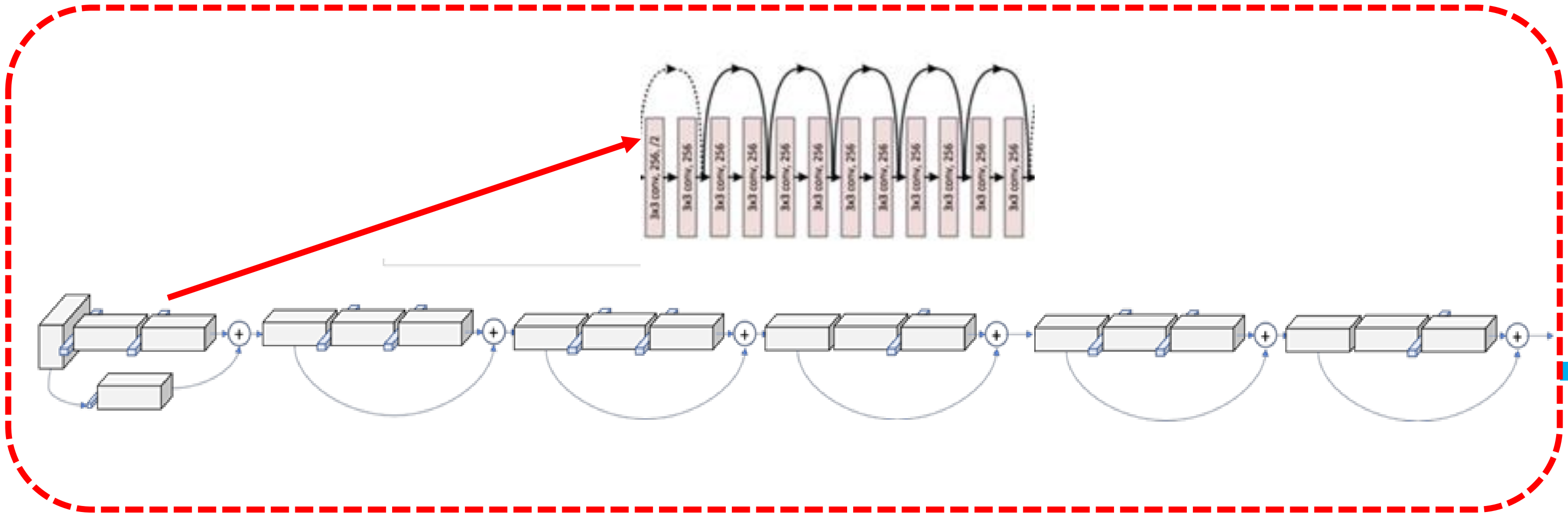


layer name	output size	34-layer
conv1	112×112	
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$
	1×1	ave
FLOPs		3.6×10^9

Layer2
(1) Transition Residual block+(**3**)* Residual Blocks

ResNet 34



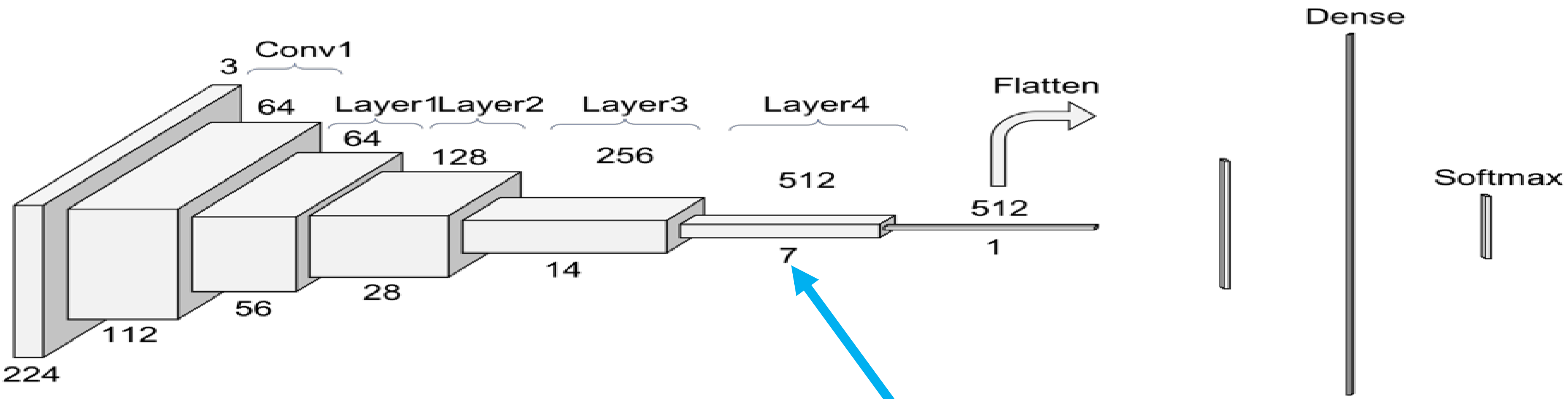


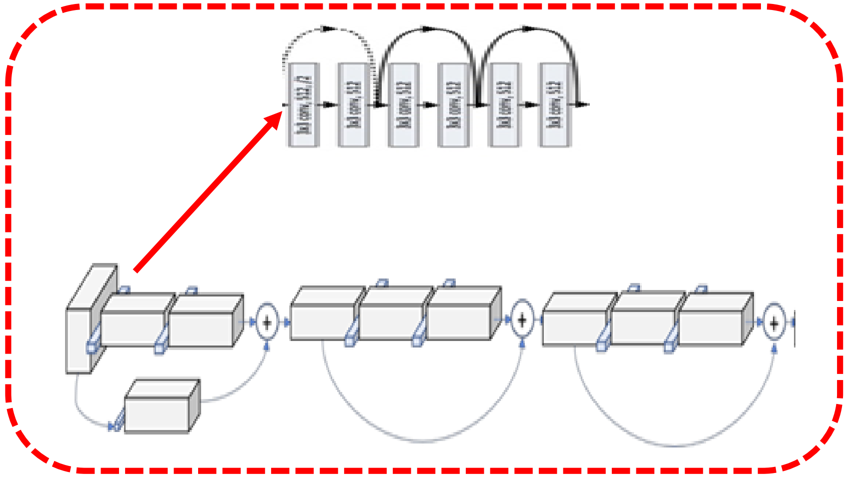
layer name	output size	34-layer
conv1	112×112	
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$
	1×1	ave
FLOPs		3.6×10^9

Layer3

(**1**)Transition Residual block+(**5**)* Residual Blocks

ResNet 34



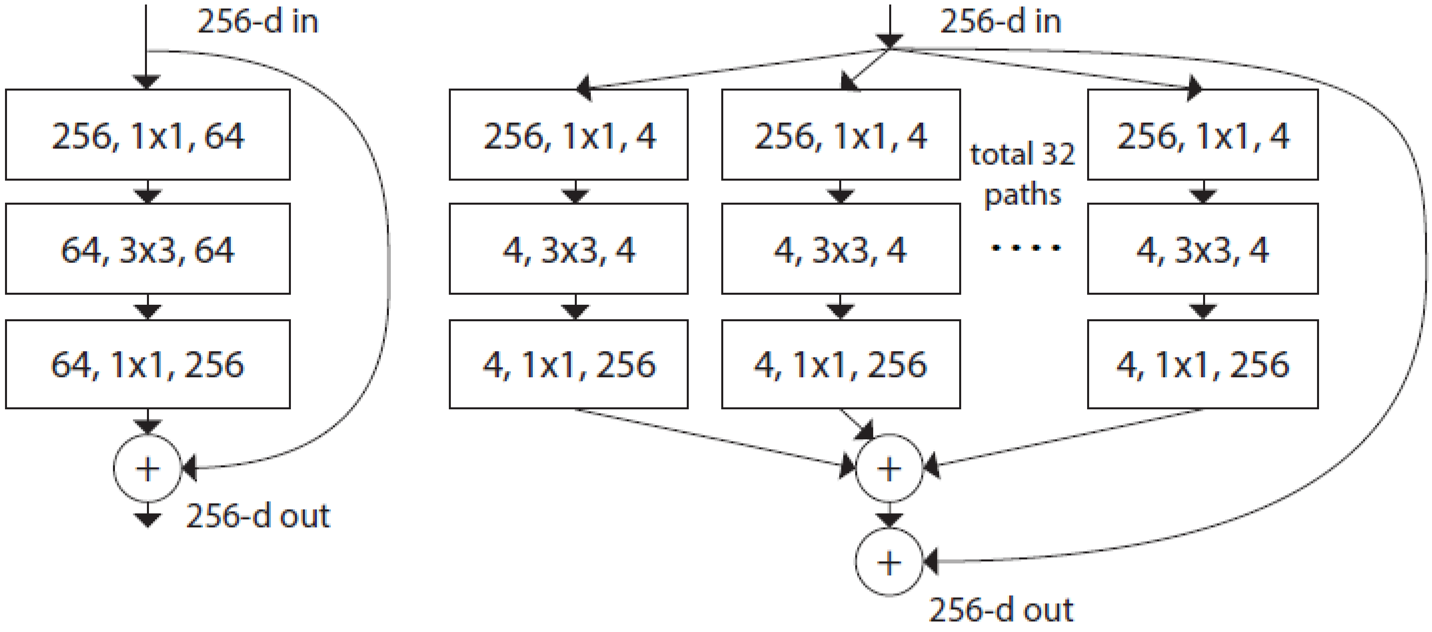


layer name	output size	34-layer
conv1	112×112	
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$
	1×1	ave
FLOPs		3.6×10^9

<u>Layer4</u>
(1)*Transition Residual block+**(2)*** Residual Blocks

ResNeXt

- **ResNeXt**
- ResNeXt — 1st Runner Up in ILSVRC 2016 (Image Classification)



Residual Block in ResNet (Left), A Block of ResNeXt with Cardinality = 32 (Right)

ResNeXt

https://towardsdatascience.com/review-resnext-1st-runner-up-of-ilsvrc-2016-image-classification-15d7f17b42ac

- **ResNeXt**
- ResNeXt — 1st Runner Up in ILSVRC 2016 (Image Classification)

Detailed Architecture of ResNet-50 and ResNeXt-50 (32x4d)

stage	output	ResNet-50	ResNeXt-50 (32×4d)
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2 [1×1, 64; 3×3, 64; 1×1, 256] ×3	3×3 max pool, stride 2 [1×1, 128; 3×3, 128, C=32; 1×1, 256] ×3
conv3	28×28	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 256; 3×3, 256, C=32; 1×1, 512] ×4
conv4	14×14	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 512; 3×3, 512, C=32; 1×1, 1024] ×6
conv5	7×7	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 1024; 3×3, 1024, C=32; 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**$\times 10^6$	**25.0**$\times 10^6$
FLOPs		**4.1**$\times 10^9$	**4.2**$\times 10^9$

Number of Parameters (Proportional to FLOPs)

$$C \cdot (256 \cdot d + 3 \cdot 3 \cdot d \cdot d + d \cdot 256)$$

Different settings to maintain similar complexity

cardinality C	1	2	4	8	32
width of bottleneck d	64	40	24	14	4
width of group conv.	64	80	96	112	128

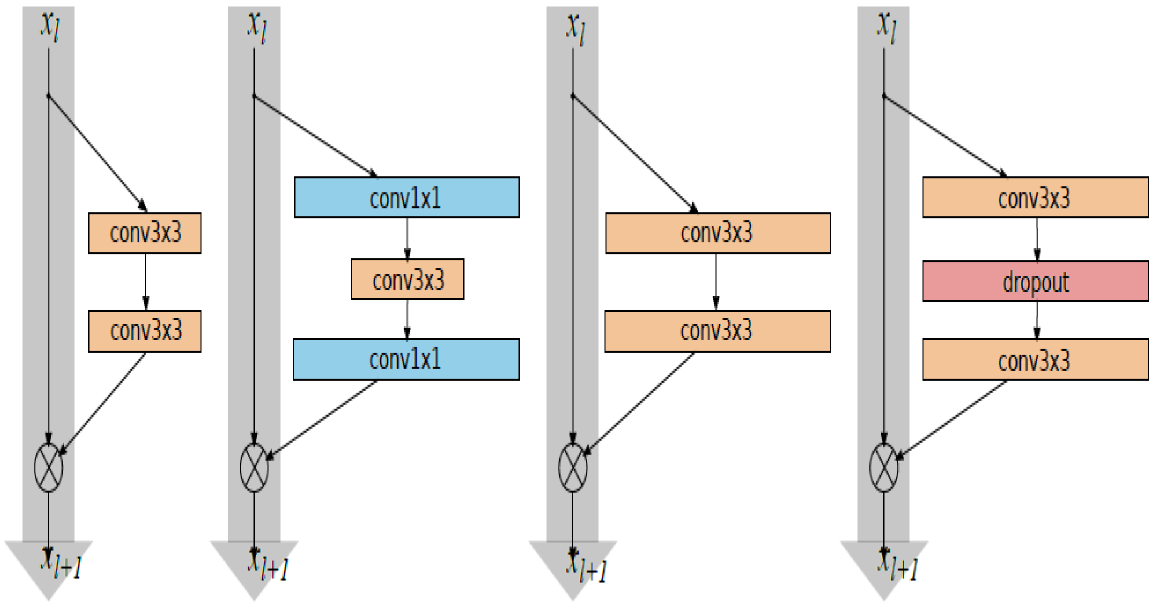
Comparisons under similar complexity

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Detailed Architecture (Left), Number of Parameters for Each Block (Top Right), Different Settings to Maintain Similar Complexity (Middle Right), Ablation Study for Different Settings Under Similar Complexity (Bottom Right)

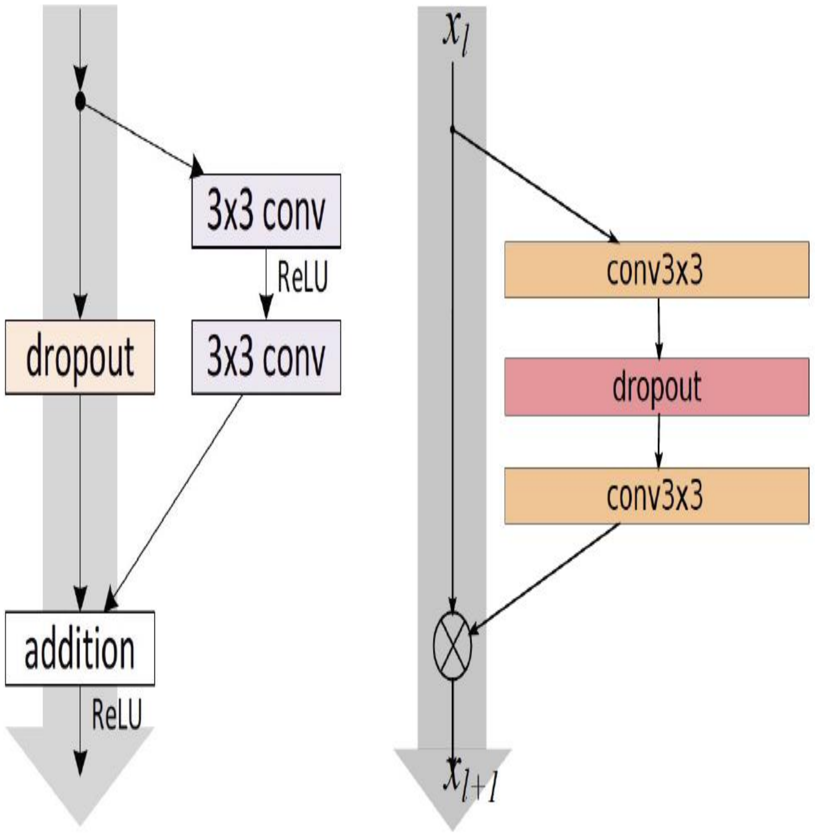
Wide Residual Networks

- **WRNs — Wide Residual Networks (Image Classification)**



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

Various ResNet Blocks

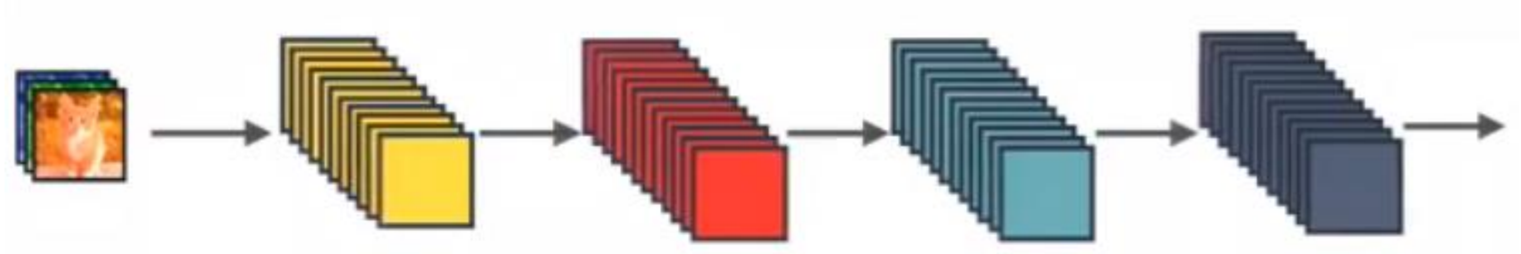


Dropout in Original ResNet (Left) and Dropout in WRNs (Right)

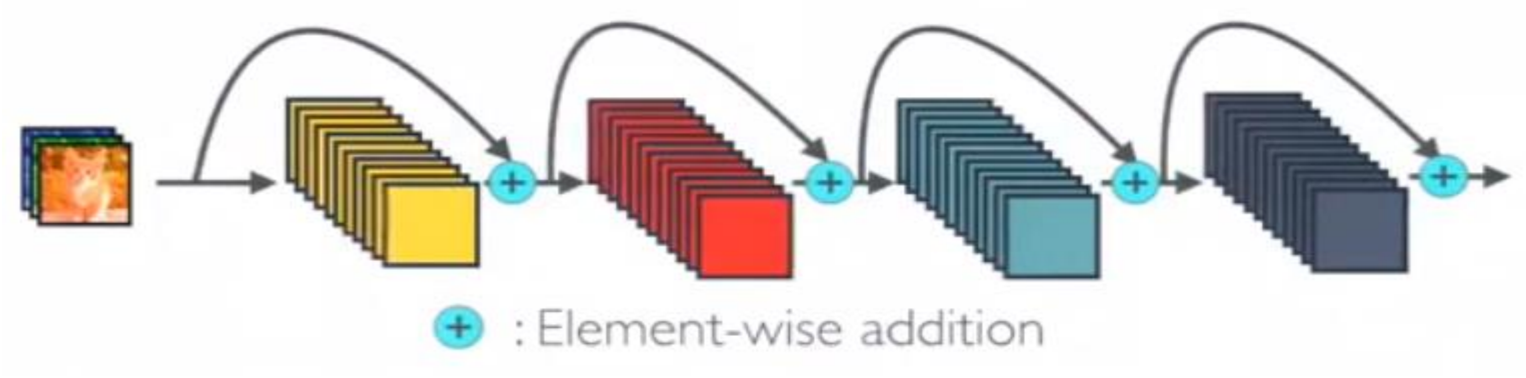
DenseNet

https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

- **DenseNet —Dense Convolutional Network**



In Standard ConvNet, input image goes through multiple convolution and obtain high-level features.

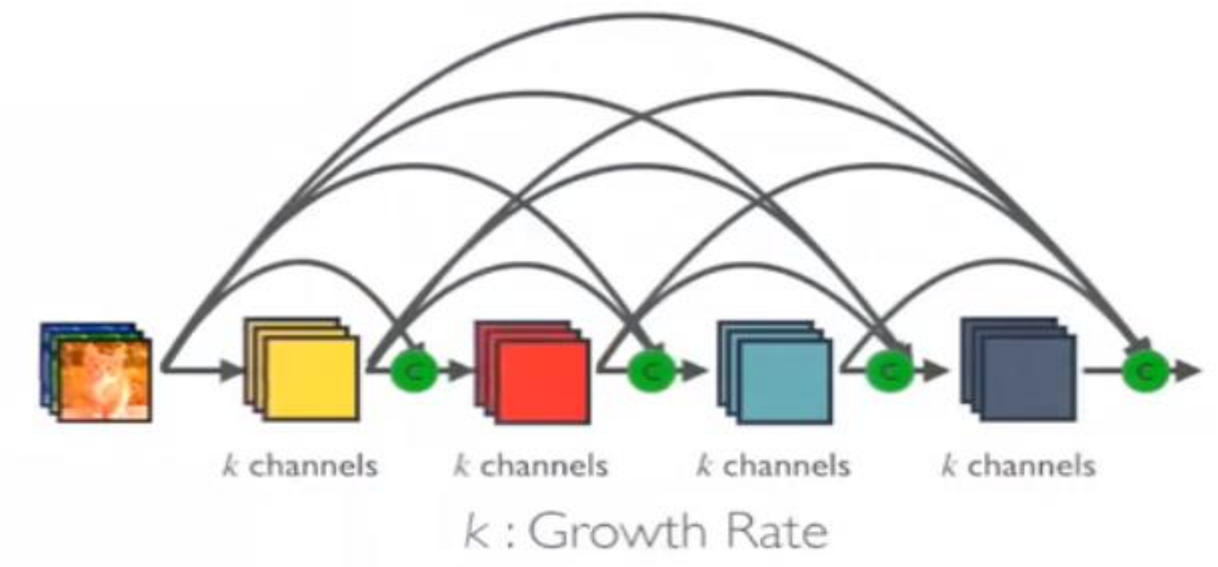


ResNet Concept

In ResNet, identity mapping is proposed to promote the gradient propagation. Element-wise addition is used. It can be viewed as algorithms with a state passed from one ResNet module to another one.

DenseNet

- **DenseNet —Dense Convolutional Network**



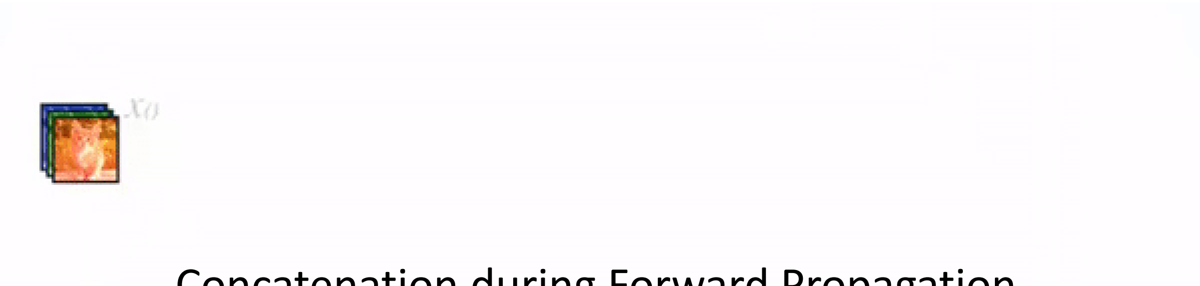
Since each layer receives feature maps from all preceding layers, network can be thinner and compact, i.e. number of channels can be fewer. The growth rate k is the additional number of channels for each layer.

Dense Block in DenseNet with Growth Rate k

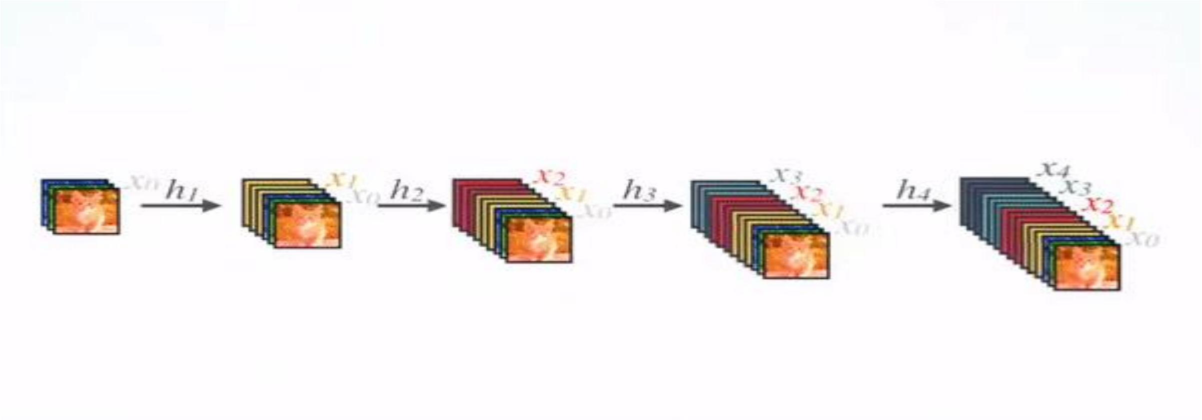
In DenseNet, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Concatenation is used. Each layer is receiving a “collective knowledge” from all preceding layers.

DenseNet

- **DenseNet —Dense Convolutional Network**



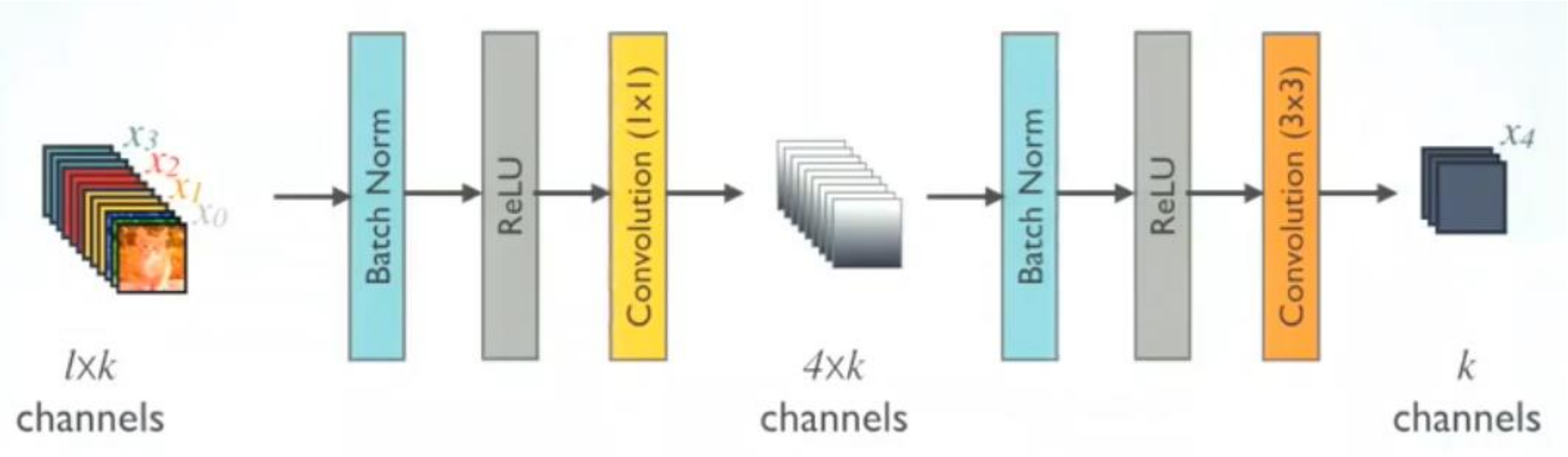
Concatenation during Forward Propagation



Composition Layer

DenseNet

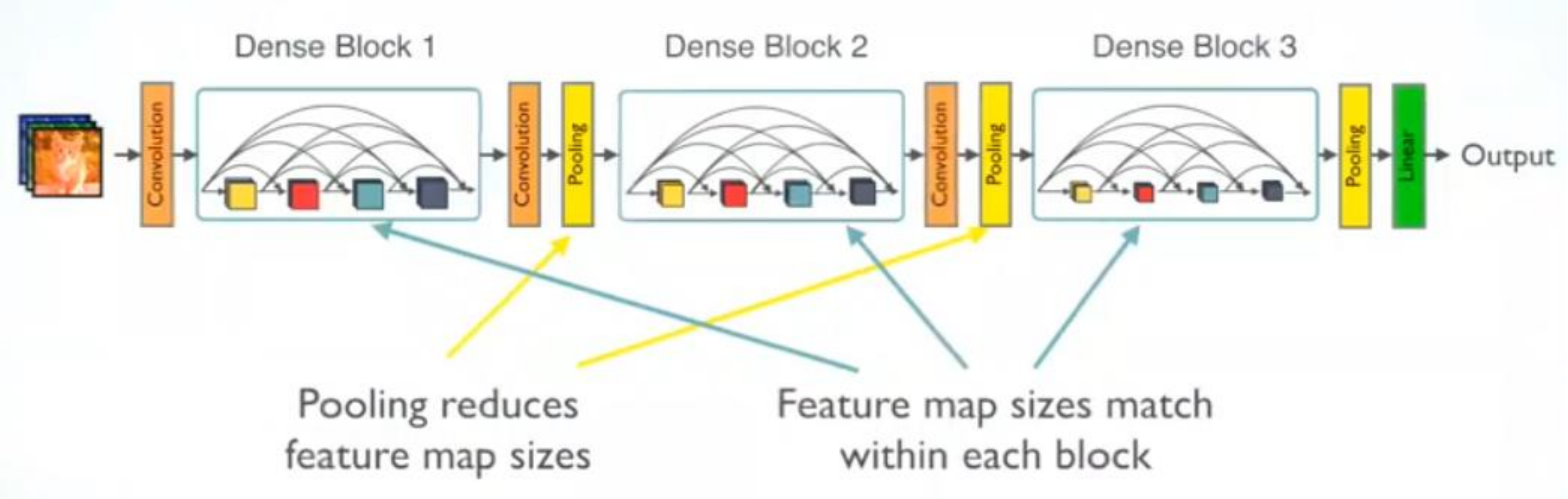
- **DenseNet —Dense Convolutional Network**



To reduce the model complexity and size, BN-ReLU-1×1 Conv is done before BN-ReLU-3×3 Conv.

DenseNet

- **DenseNet —Dense Convolutional Network**



1×1 Conv followed by 2×2 average pooling are used as the transition layers between two contiguous dense blocks.
Feature map sizes are the same within the dense block so that they can be concatenated together easily.
At the end of the last dense block, a global average pooling is performed and then a softmax classifier is attached.

DenseNet

https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

- **DenseNet —Dense Convolutional Network**

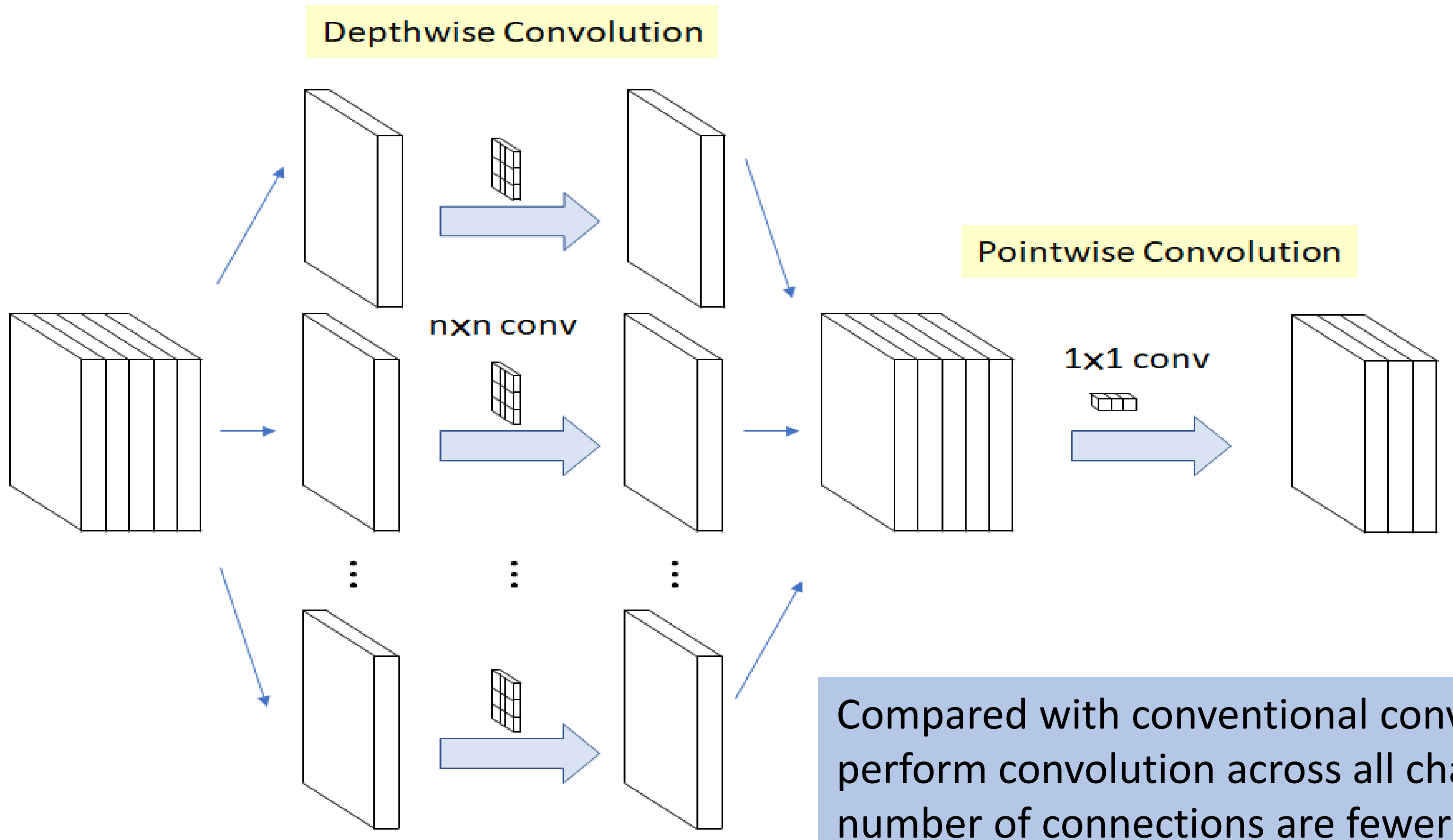
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Xception

https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568

- **Xception — With Depthwise Separable.**
- Original Depthwise Separable Convolution

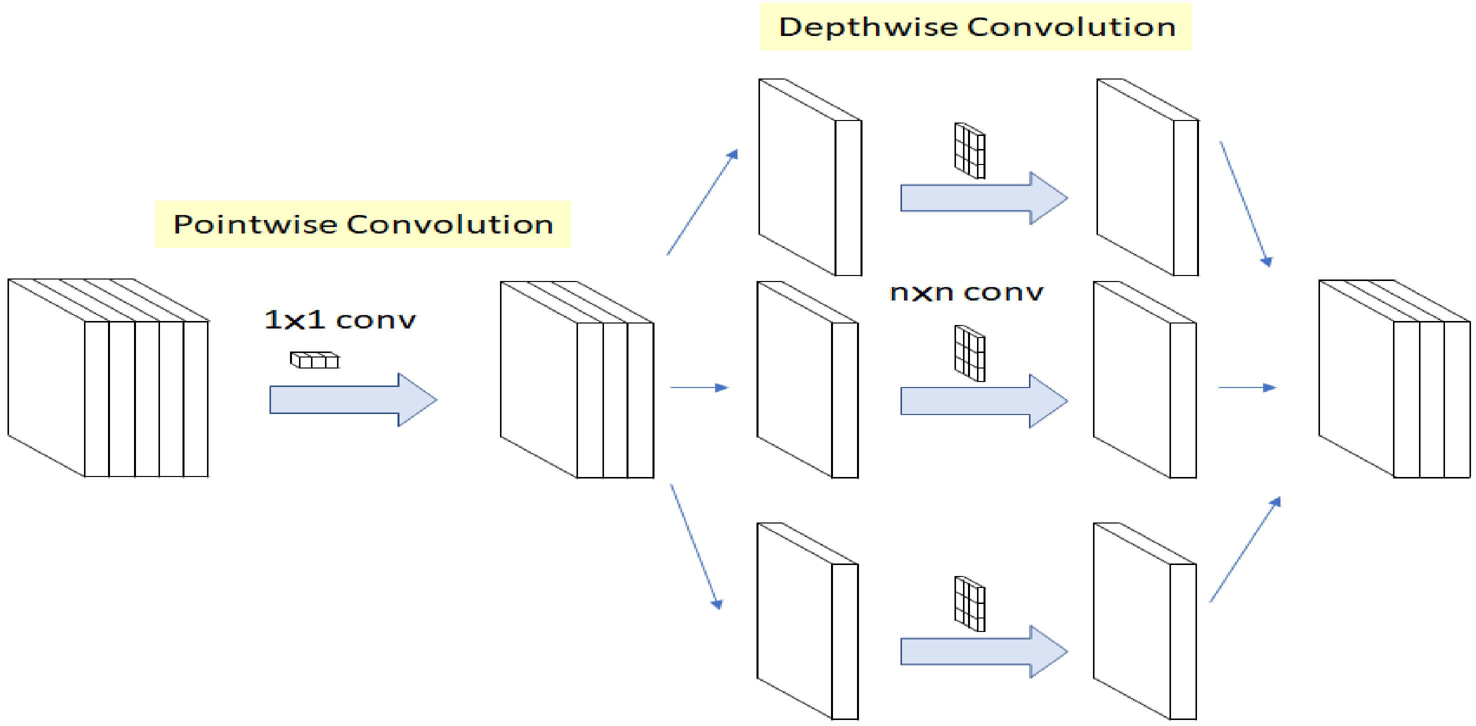


The original depthwise separable convolution is the depthwise convolution followed by a pointwise convolution.
Depthwise convolution is the channel-wise n×n spatial convolution.
Pointwise convolution actually is the 1×1 convolution to change the dimension.

Compared with conventional convolution, we do not need to perform convolution across all channels. That means the number of connections are fewer and the model is lighter.

Xception

- **Xception — With Depthwise Separable.**
- Modified Depthwise Separable Convolution in Xception



The modified depthwise separable convolution is the **pointwise convolution followed by a depthwise convolution.**

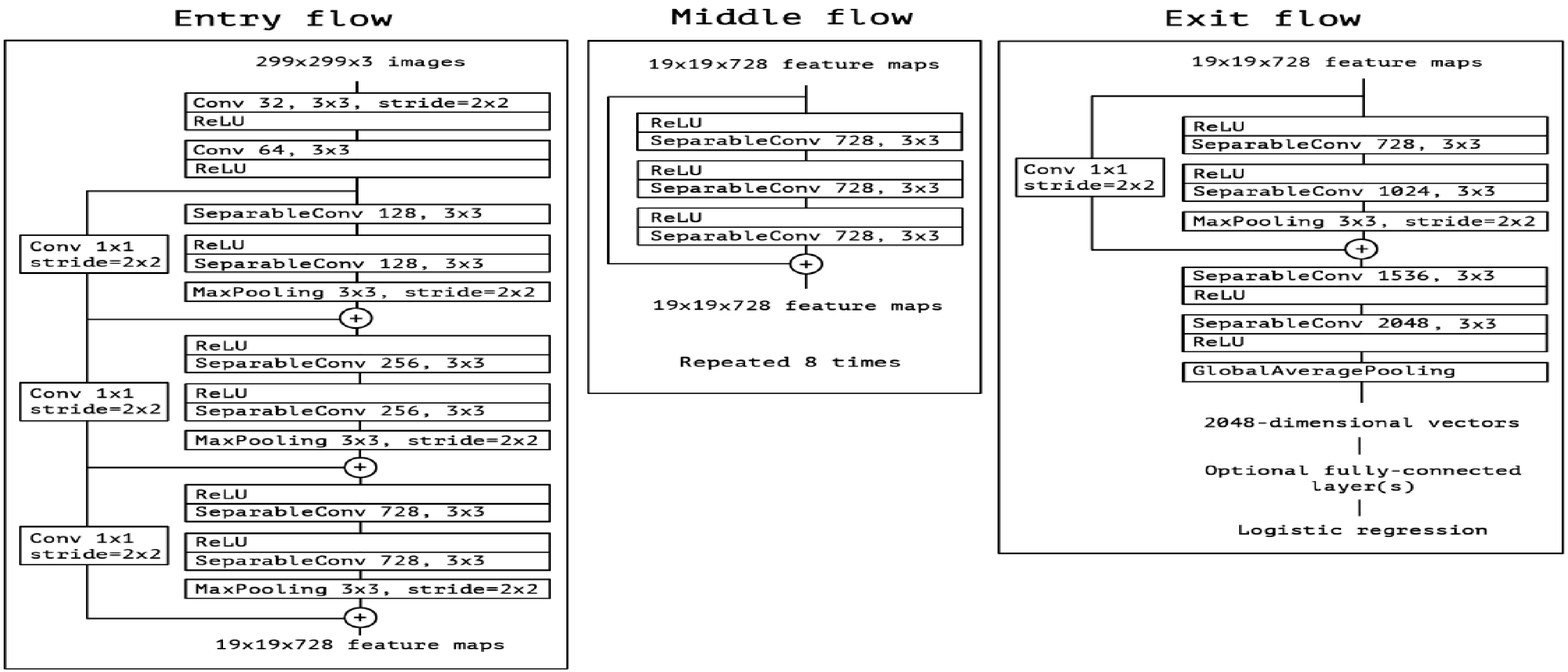
This is claimed to be unimportant because when it is used in stacked setting, there are only small differences appeared at the beginning and at the end of all the chained inception modules.

The Presence/Absence of Non-Linearity: In the original Inception Module, there is non-linearity after first operation. In Xception, the modified depthwise separable convolution, there is NO intermediate ReLU non-linearity.

The Modified Depthwise Separable Convolution used as an Inception Module in Xception, so called "extreme" version of Inception module (n=3 here).

Xception

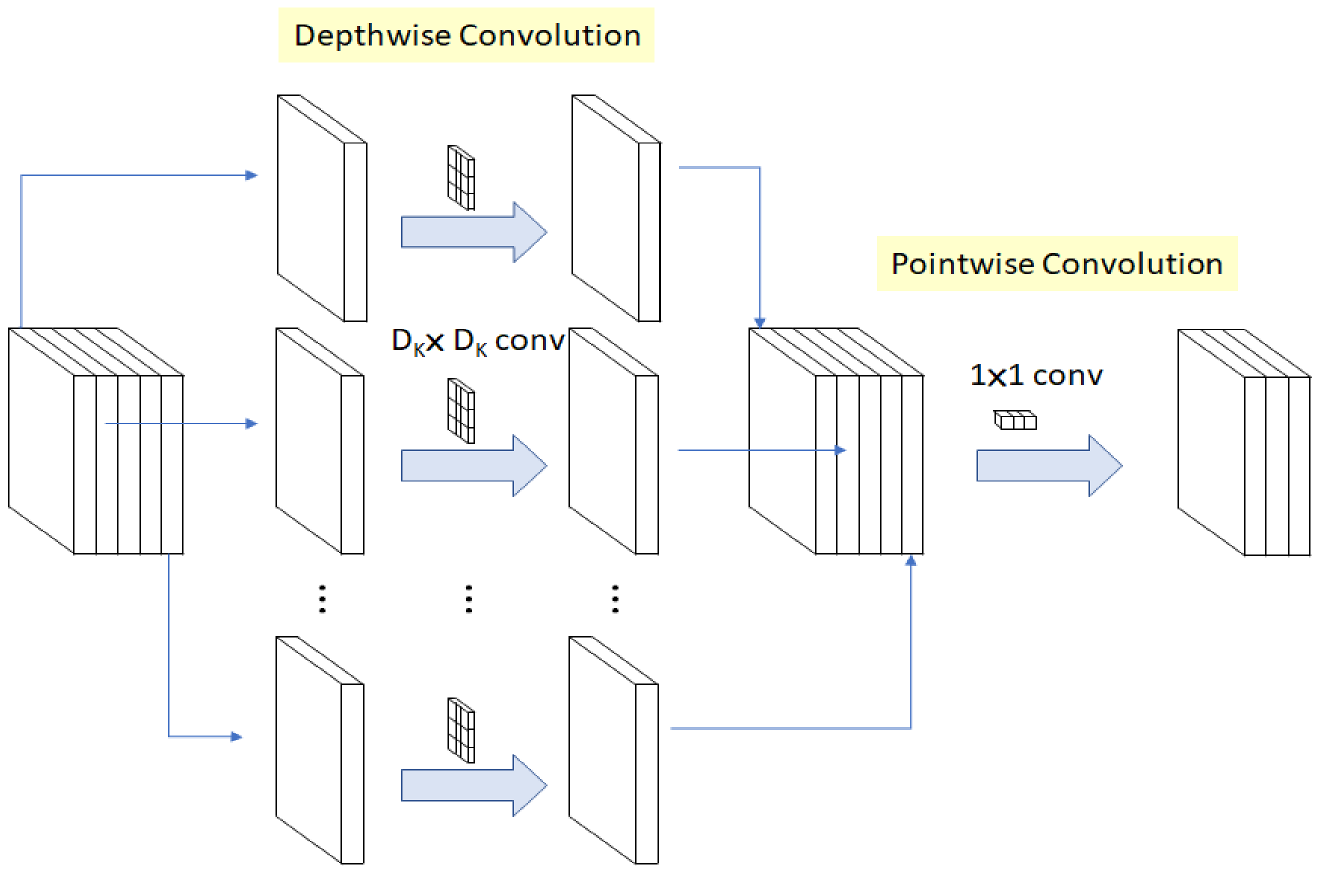
- **Xception — With Depthwise Separable.**
- Modified Depthwise Separable Convolution in Xception



Overall Architecture of Xception (Entry Flow > Middle Flow > Exit Flow)

MobileNetV1

- **MobileNetV1**
- MobileNetV1 — Depthwise Separable Convolution (Light Weight Model)



Depthwise convolution is the channel-wise DK×DK spatial convolution.
Suppose in the figure , we have 5 channels, then we will have 5 DK×DK spatial convolution.
Pointwise convolution actually is the 1×1 convolution to change the dimension.

MobileNetV1

- **MobileNetV1**
- MobileNetV1 — Depthwise Separable Convolution (Light Weight Model)

With above operation, the operation cost is: $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$

Depthwise Separable Convolution Cost: Depthwise Convolution Cost (Left), Pointwise Convolution Cost (Right) where M: Number of input channels, N: Number of output channels, DK: Kernel size, and DF: Feature map size. For standard convolution, it is: $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$

Standard Convolution Cost $D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$

Thus, the computation reduction is:

Depthwise Separable Convolution Cost / Standard Convolution Cost

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

When DK×DK is 3×3, 8 to 9 times less computation can be achieved, but with only small reduction in accuracy.

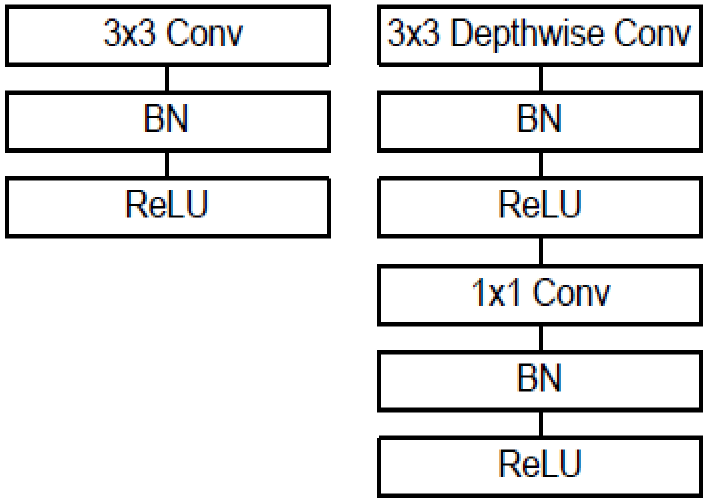
MobileNetV1

https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69

- **MobileNetV1**
- MobileNetV1 — Depthwise Separable Convolution (Light Weight Model)

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	$1 \times 1 \times 1000$

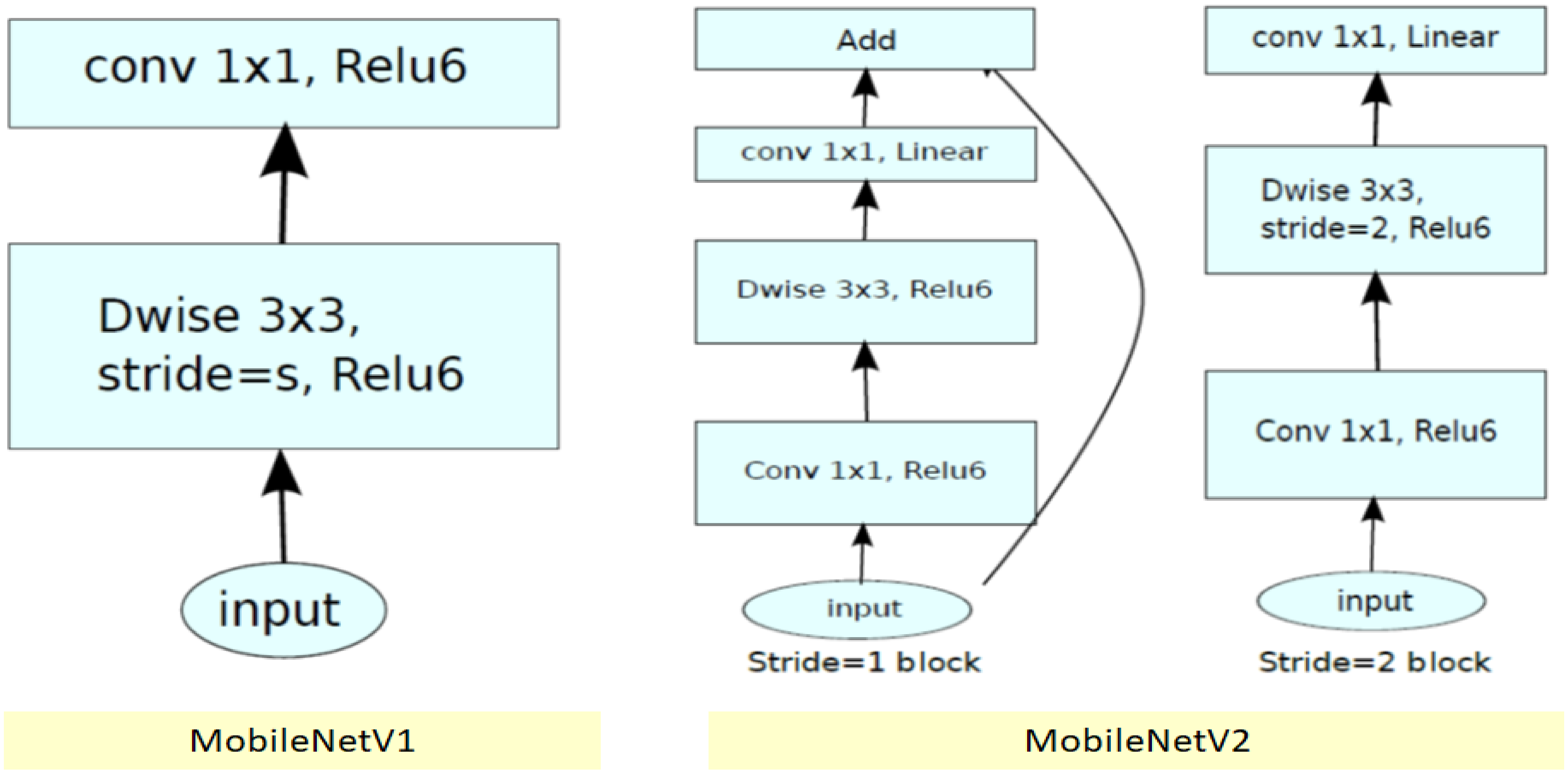


Standard Convolution (Left), Depthwise separable convolution (Right) With BN and ReLU

MobileNetV2

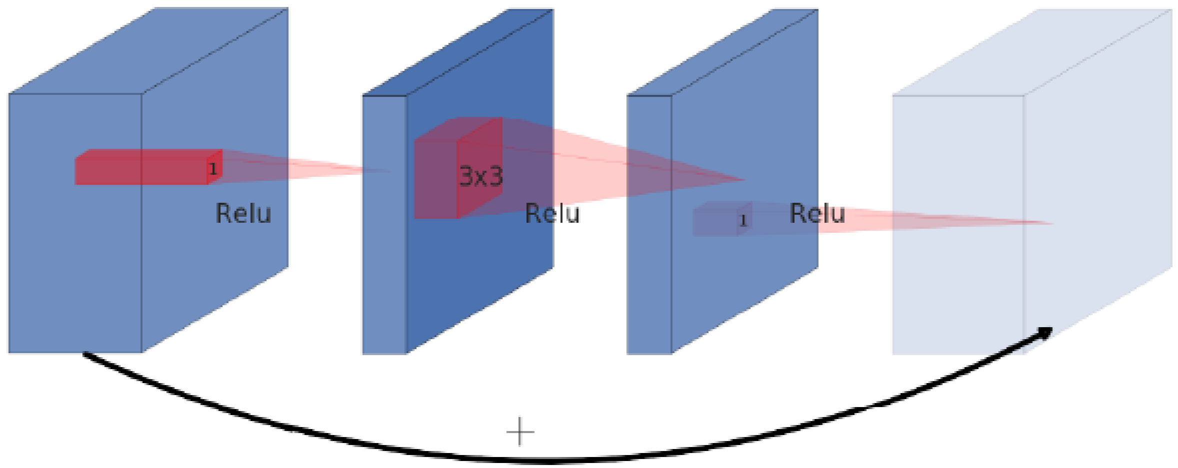
https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

- **MobileNetV2**
- **MobileNetV2 — Light Weight Model (Image Classification)**

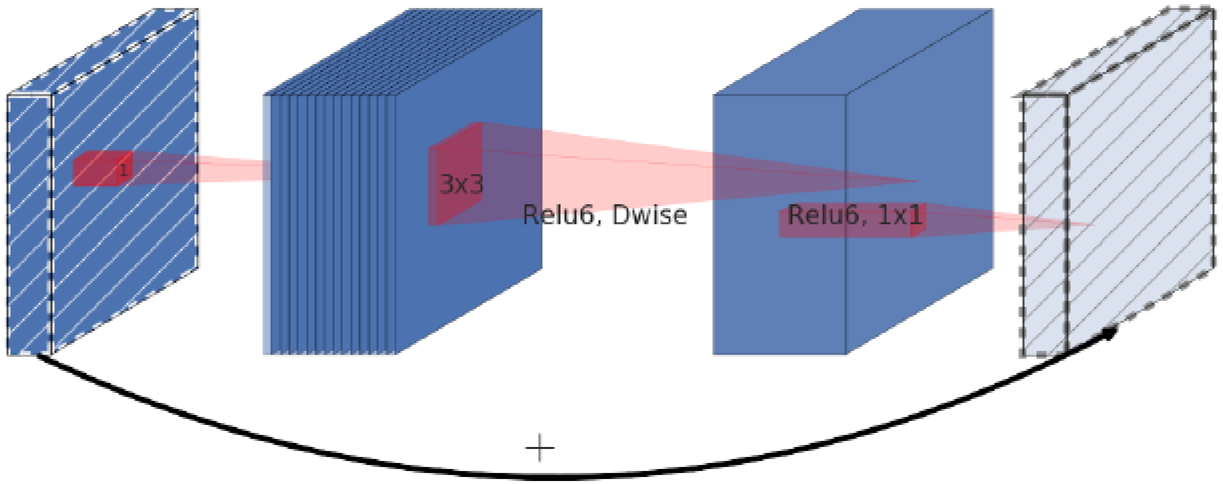


MobileNetV2

- **MobileNetV2**
- **MobileNetV2 — Light Weight Model (Image Classification)**



A residual block connects wide layers with a skip connection while layers in between are narrow



An inverted residual block connects narrow layers with a skip connection while layers in between are wide

MobileNetV2

- **MobileNetV2**
- **MobileNetV2 — Light Weight Model (Image Classification)**

In MobileNetV1, there are 2 layers.
The first layer is called a depthwise convolution, it performs lightweight filtering by applying a single convolutional filter per input channel.
The second layer is a 1×1 convolution, called a pointwise convolution, which is responsible for building new features through computing linear combinations of the input channels.

MobileNetV2
In MobileNetV2, there are two types of blocks. One is residual block with stride of 1.
Another one is block with stride of 2 for downsizing.
There are 3 layers for both types of blocks.
This time, the first layer is 1×1 convolution with ReLU6.
The second layer is the depthwise convolution.
The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.

MobileNetV2

- **MobileNetV2**
- **MobileNetV2 — Light Weight Model (Image Classification)**

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	-	

where t: expansion factor,
c: number of output channels,
n: repeating number,
s: stride. 3×3 kernels are used for spatial convolution.
In typical, the primary network (width multiplier 1, 224×224), has a computational cost of 300 million multiply-adds and uses 3.4 million parameters. (Width multiplier is introduced in MobileNetV1.)

The performance trade offs are further explored, for input resolutions from 96 to 224, and width multipliers of 0.35 to 1.4. The network computational cost up to 585M M Adds, while the model size vary between 1.7M and 6.9M parameters.

MobileNetV2

- **MobileNetV2**

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	-	

```
def bottleneck_block(x, expand=64, squeeze=16):
m = Conv2D(expand, (1,1))(x)
m = BatchNormalization()(m)
m = Activation('relu6')(m)
m = DepthwiseConv2D((3,3))(m)
m = BatchNormalization()(m)
m = Activation('relu6')(m)
m = Conv2D(squeeze, (1,1))(m)
m = BatchNormalization()(m)
return Add()([m, x])
```

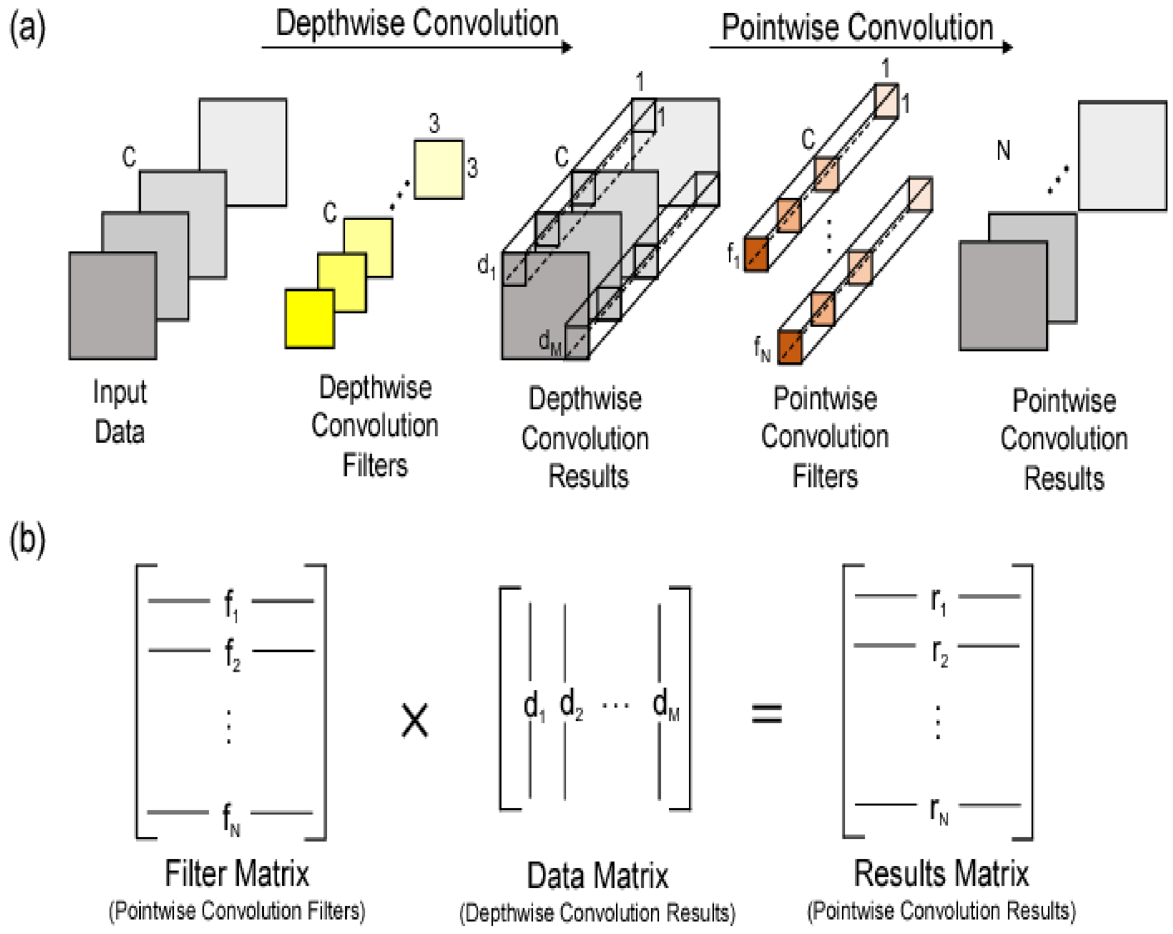
```
def inverted_residual_block(x, expand=64, squeeze=16):
  m = Conv2D(expand, (1,1), activation='relu')(x)
  m = DepthwiseConv2D((3,3), activation='relu')(m)
  m = Conv2D(squeeze, (1,1), activation='relu')(m)
  return Add()([m, x])
```

```
def residual_block(x, squeeze=16, expand=64):
m = Conv2D(squeeze, (1,1), activation='relu')(x)
m = Conv2D(squeeze, (3,3), activation='relu')(m)
m = Conv2D(expand, (1,1), activation='relu')(m)
return Add()([m, x])
```

EfficientNet Models

https://heartbeat.fritz.ai/reviewing-efficientnet-increasing-the-accuracy-and-robustness-of-cnns-6aaf411fc81d

- **Rethinking Model Scaling for Convolutional Neural Networks**



Depthwise Convolution + Pointwise Convolution: Divides the original convolution into two stages to significantly reduce the cost of calculation, with a minimum loss of accuracy.

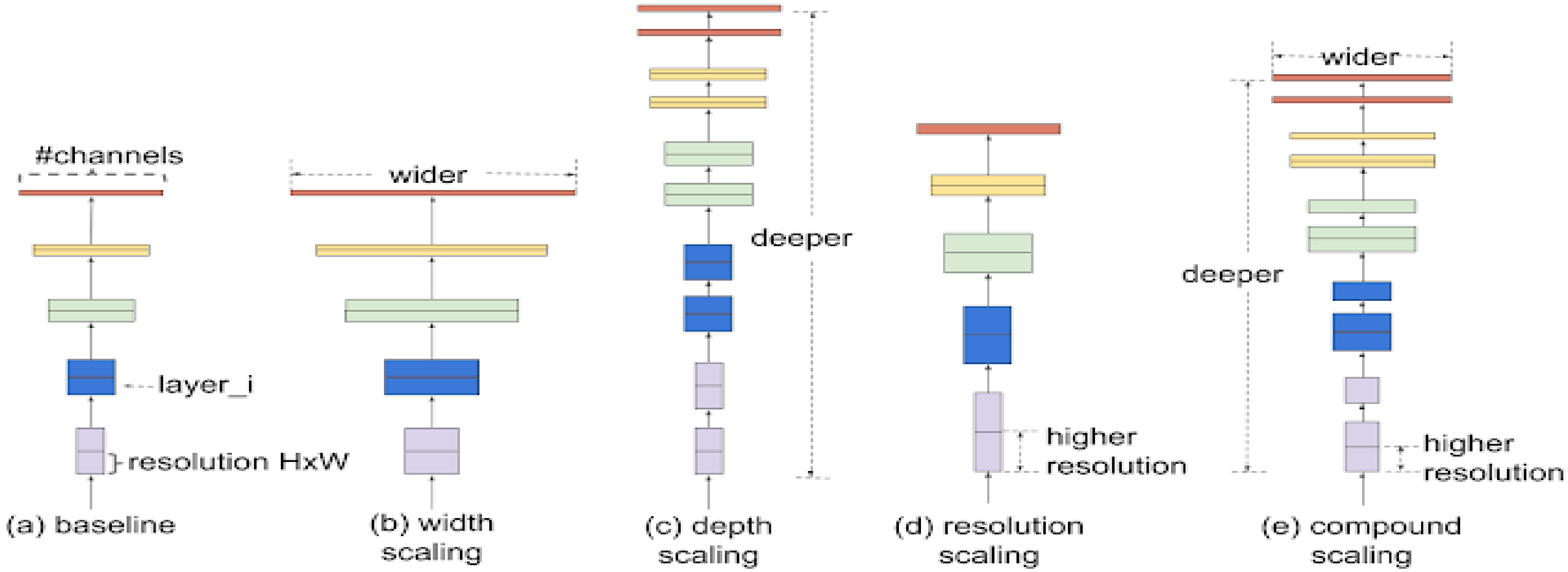
Inverse Res: The original ResNet blocks consist of a layer that squeezes the channels, then a layer that extends the channels. In this way, it links skip connections to rich channel layers. In **MBConv,** however, blocks consist of a layer that first extends channels and then compresses them, so that layers with fewer channels are skip connected.

Linear bottleneck: Uses linear activation in the last layer in each block to prevent loss of information from ReLU.

EfficientNet Models

https://heartbeat.fritz.ai/reviewing-efficientnet-increasing-the-accuracy-and-robustness-of-cnns-6aaf411fc81d

- **EfficientNet-B0-B7**



Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

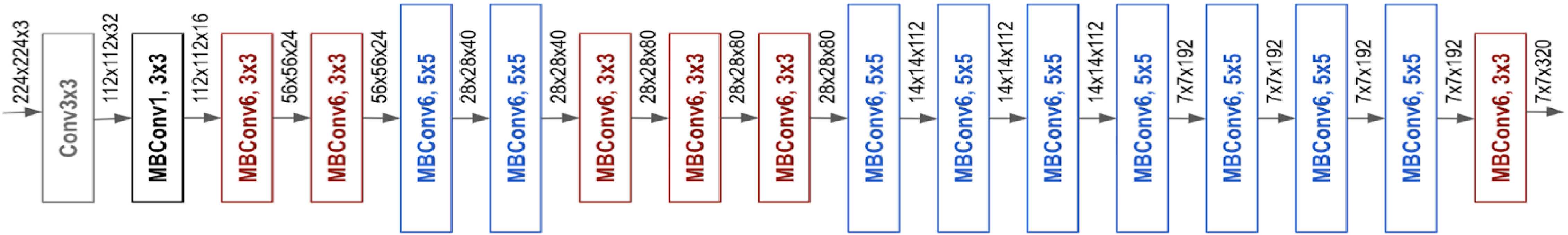
EfficientNet Models

- **EfficientNet-B0-B7**

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

Step 1:
Building a basic model is called EfficientNet-B0.
MBConv is used with MobileNet's inverted Res bottlenecks.

Basic network structure of EfficientNet-B0



A basic block representation of EfficientNet-B0

EfficientNet Models

- **EfficientNet-B0-B7**

Step 1:

Building a basic model is called EfficientNet-B0.

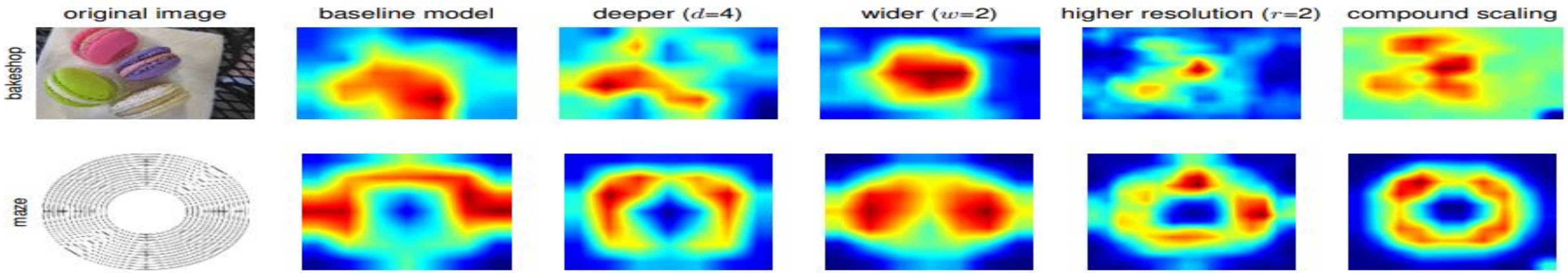
MBConv is used with MobileNet's inverted Res bottlenecks.

Step 2:

φ = 1 and grid search for α, β, and to scale from B0 to B1.

Step 3:

α,β,γ set. Thus, for scaling from B2 to B7, φ is selected between 2 ~ 7. Below, you can see class activation maps for models with different scaling methods.



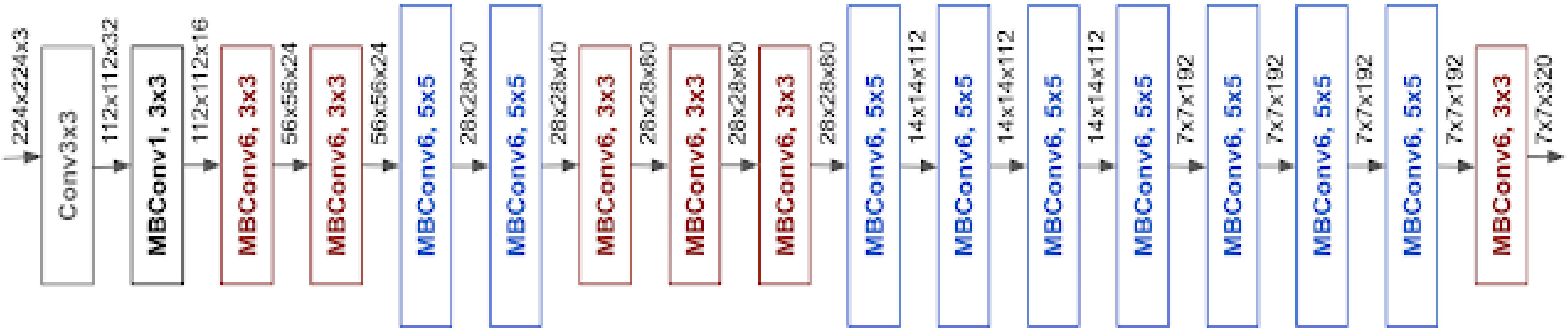
Class Activation Map (CAM) (Zhou et al., 2016) for Models with different scaling methods- Our compound scaling method allows the scaled model (last column) to focus on more relevant regions with more object details.

EfficientNet Models

- **EfficientNet-B0-B7**

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

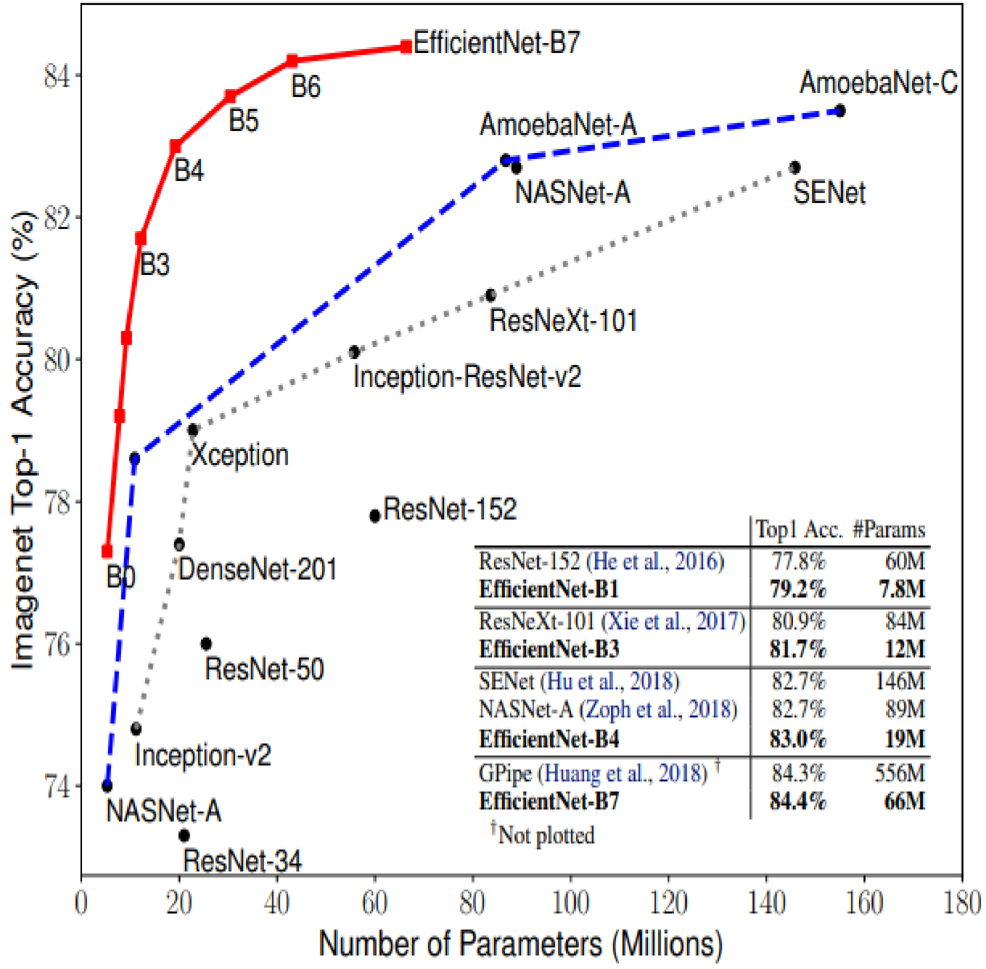


EfficientNet Models

https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

- **EfficientNet-B0-B7**

	Comparison to best public-available results						Comparison to best reported results					
	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)
CIFAR-10	NASNet-A	98.0%	85M	EfficientNet-B0	98.1%	4M (21x)	†Gpipe	**99.0%**	556M	EfficientNet-B7	98.9%	64M (8.7x)
CIFAR-100	NASNet-A	87.5%	85M	EfficientNet-B0	88.1%	4M (21x)	Gpipe	91.3%	556M	EfficientNet-B7	**91.7%**	64M (8.7x)
Birdsnap	Inception-v4	81.8%	41M	EfficientNet-B5	82.0%	28M (1.5x)	GPipe	83.6%	556M	EfficientNet-B7	**84.3%**	64M (8.7x)
Stanford Cars	Inception-v4	93.4%	41M	EfficientNet-B3	93.6%	10M (4.1x)	†DAT	**94.8%**	-	EfficientNet-B7	94.7%	-
Flowers	Inception-v4	98.5%	41M	EfficientNet-B5	98.5%	28M (1.5x)	DAT	97.7%	-	EfficientNet-B7	**98.8%**	-
FGVC Aircraft	Inception-v4	90.9%	41M	EfficientNet-B3	90.7%	10M (4.1x)	DAT	92.9%	-	EfficientNet-B7	**92.9%**	-
Oxford-IIIT Pets	ResNet-152	94.5%	58M	EfficientNet-B4	94.8%	17M (5.6x)	GPipe	**95.9%**	556M	EfficientNet-B6	95.4%	41M (14x)
Food-101	Inception-v4	90.8%	41M	EfficientNet-B4	91.5%	17M (2.4x)	GPipe	93.0%	556M	EfficientNet-B7	**93.0%**	64M (8.7x)
Geo-Mean						(4.7x)						(9.6x)



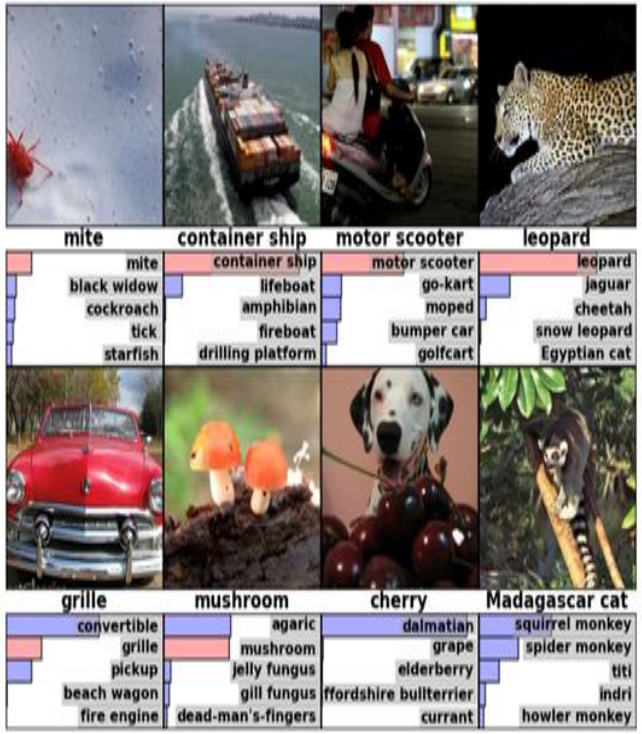
Transfer Learning DL models

- **Transfer Learning**

ImageNet Challenge

IMAGENET

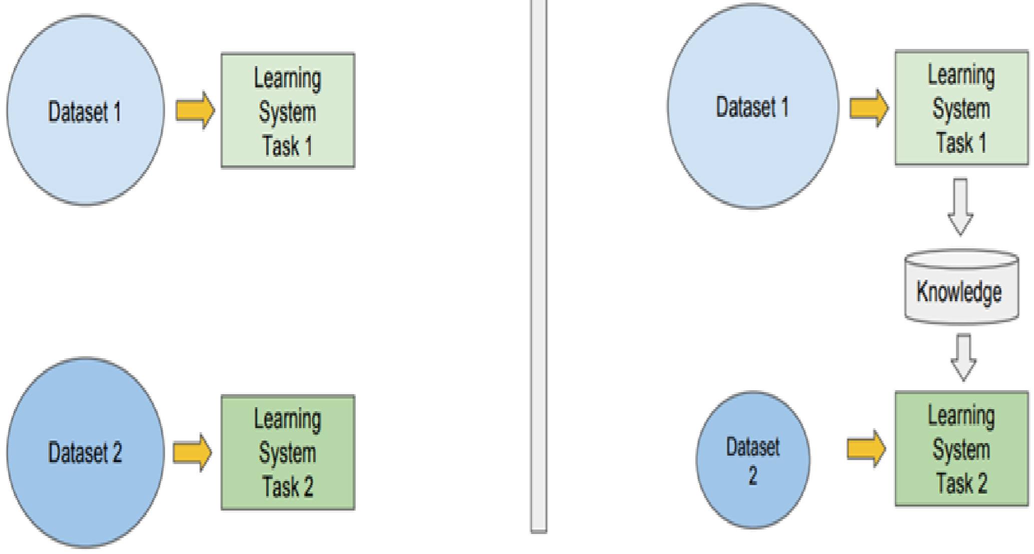
- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks
- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



Transfer Learning DL models

- **Transfer Learning**

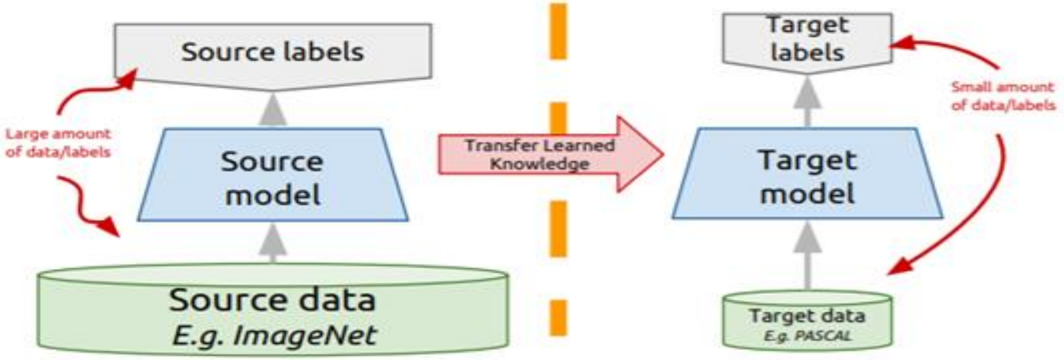
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

- Same domain, different task
- Different domain, same task



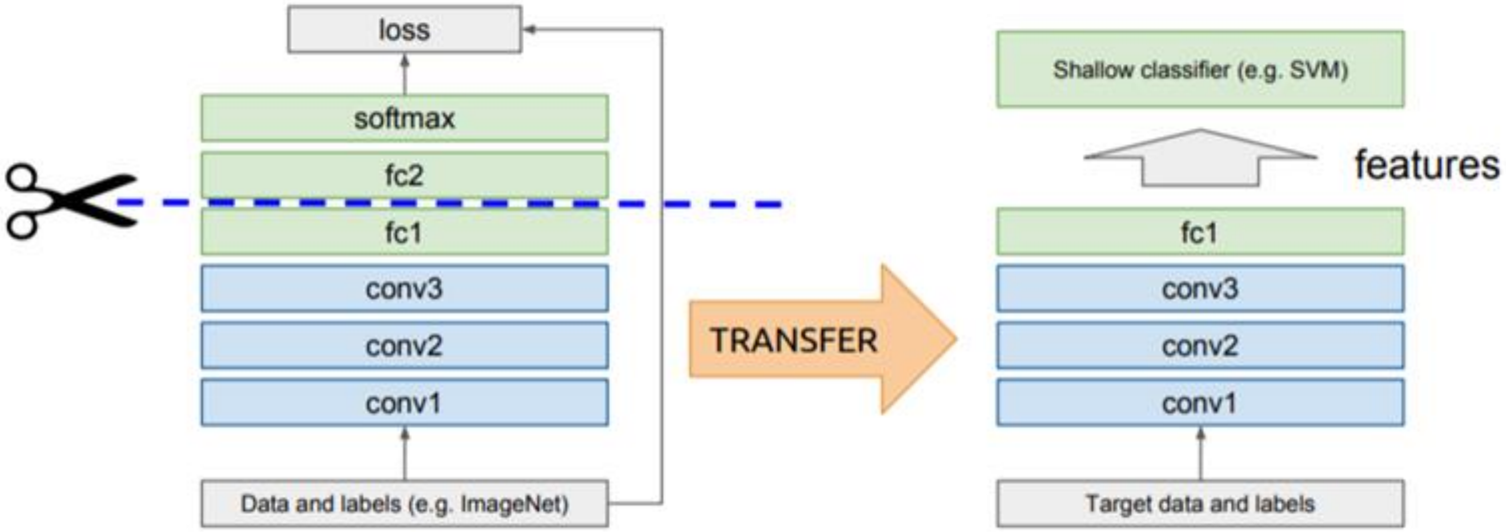
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning DL models

- **Off-the-shelf Pre-trained Models as Feature Extractors**

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

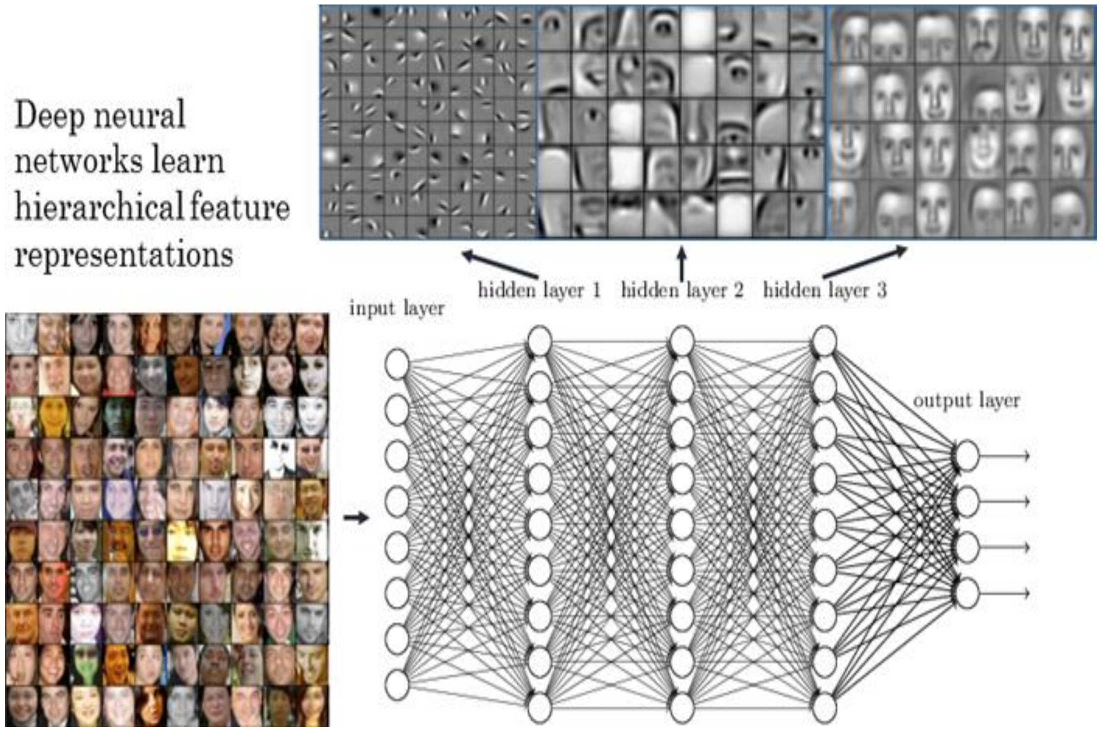
Assumes that $D_S = D_T$



https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning DL models

- **Fine Tuning Off-the-shelf Pre-trained Models**



Fine-tuning: supervised domain adaptation

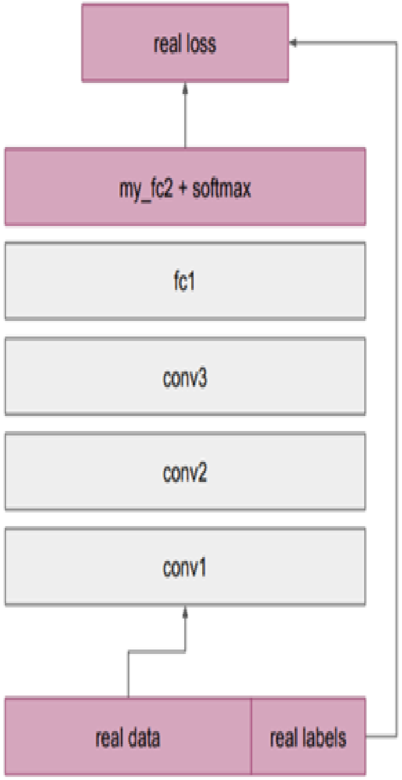
Train deep net on "nearby" task for which it is easy to get labels using standard backprop

- E.g. ImageNet classification
- Pseudo classes from augmented data
- Slow feature learning, ego-motion

Cut off top layer(s) of network and replace with supervised objective for target domain

Fine-tune network using backprop with labels for target domain until validation loss starts to increase

Aligns D_S with D_T



https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning DL models

- **Fine Tuning Off-the-shelf Pre-trained Models**

Freeze or fine-tune?

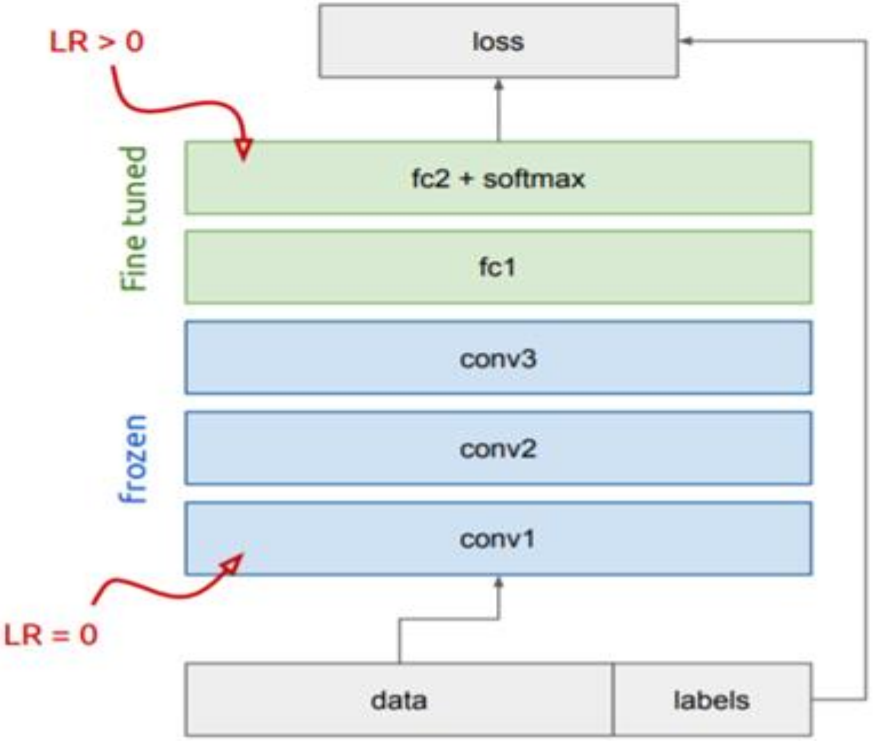
Bottom *n* layers can be frozen or fine tuned.

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

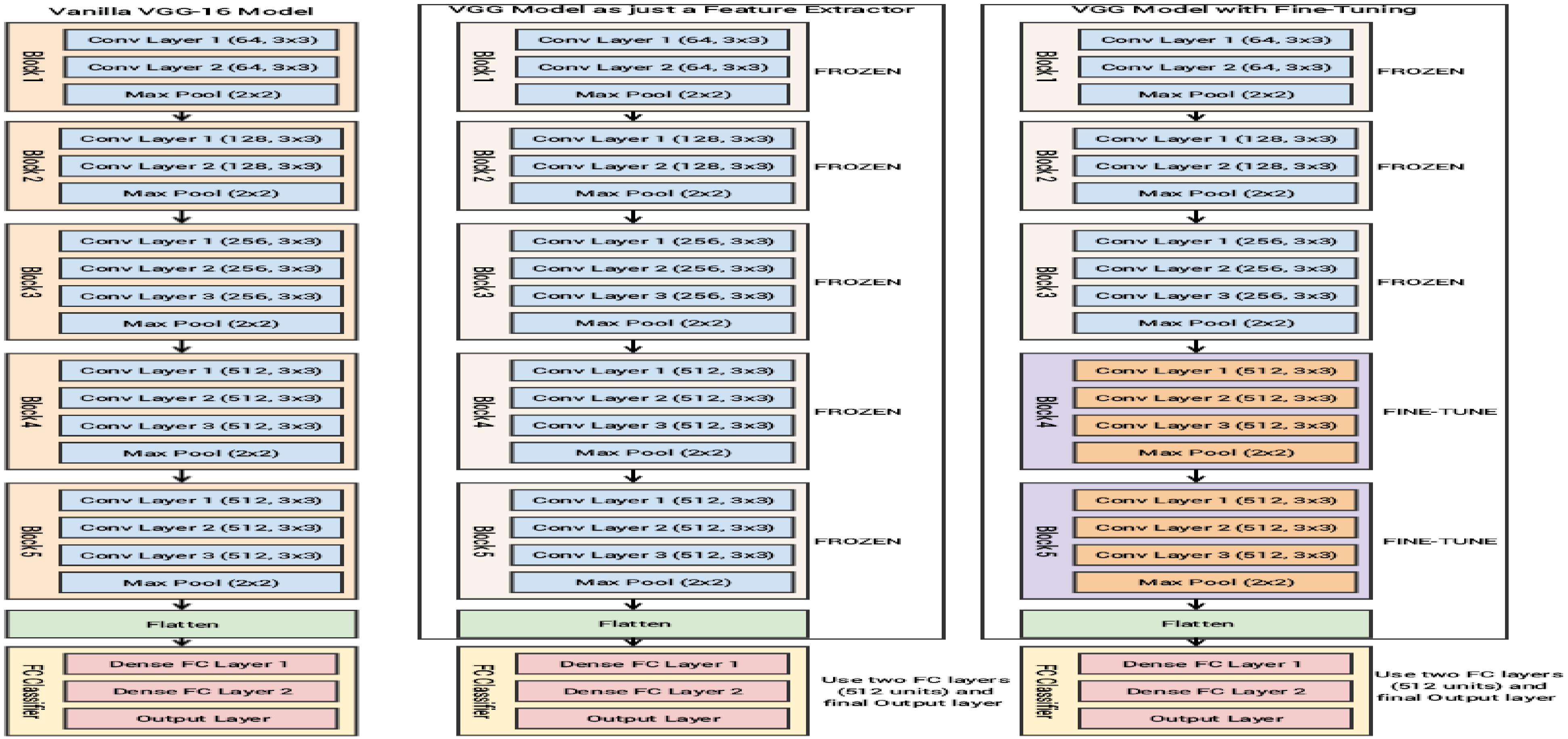
In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning



https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning DL models

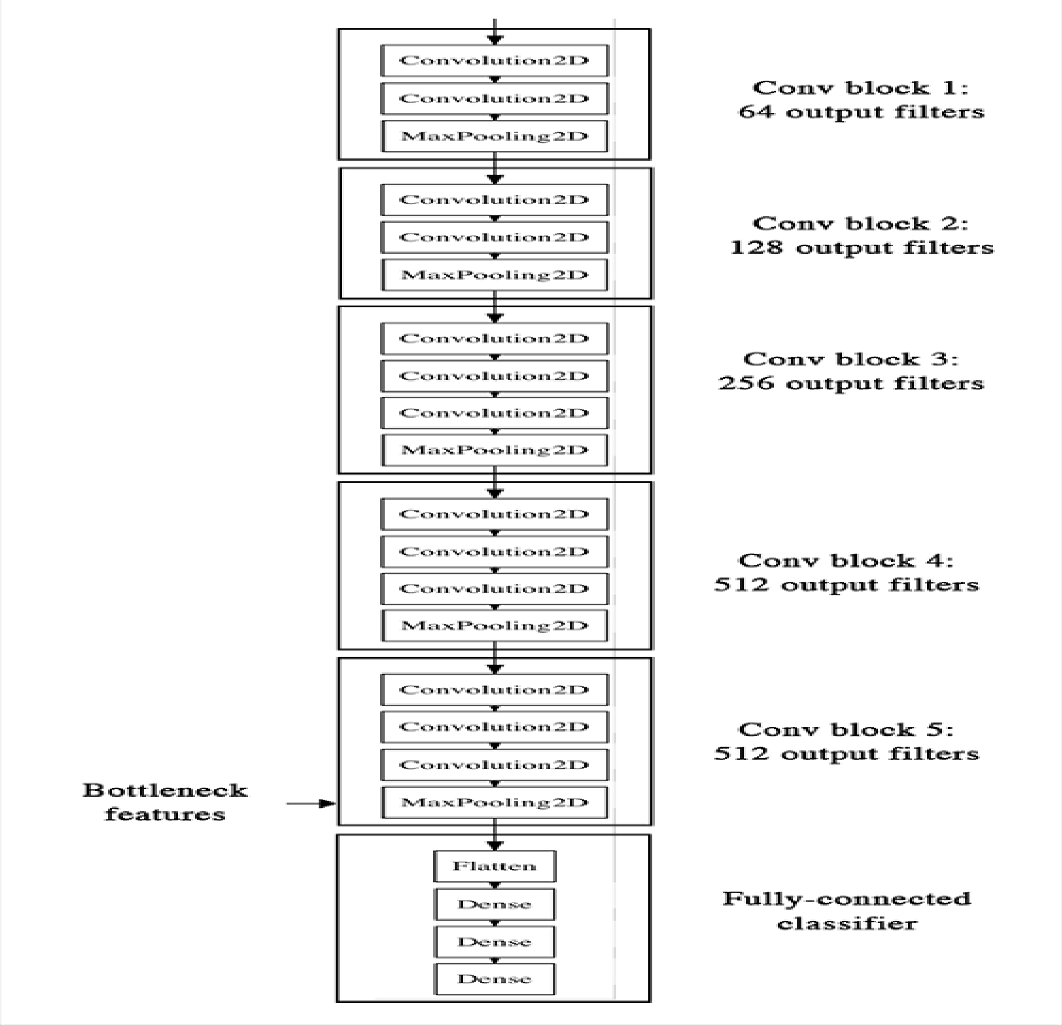
- **Fine Tuning Off-the-shelf Pre-trained Models**

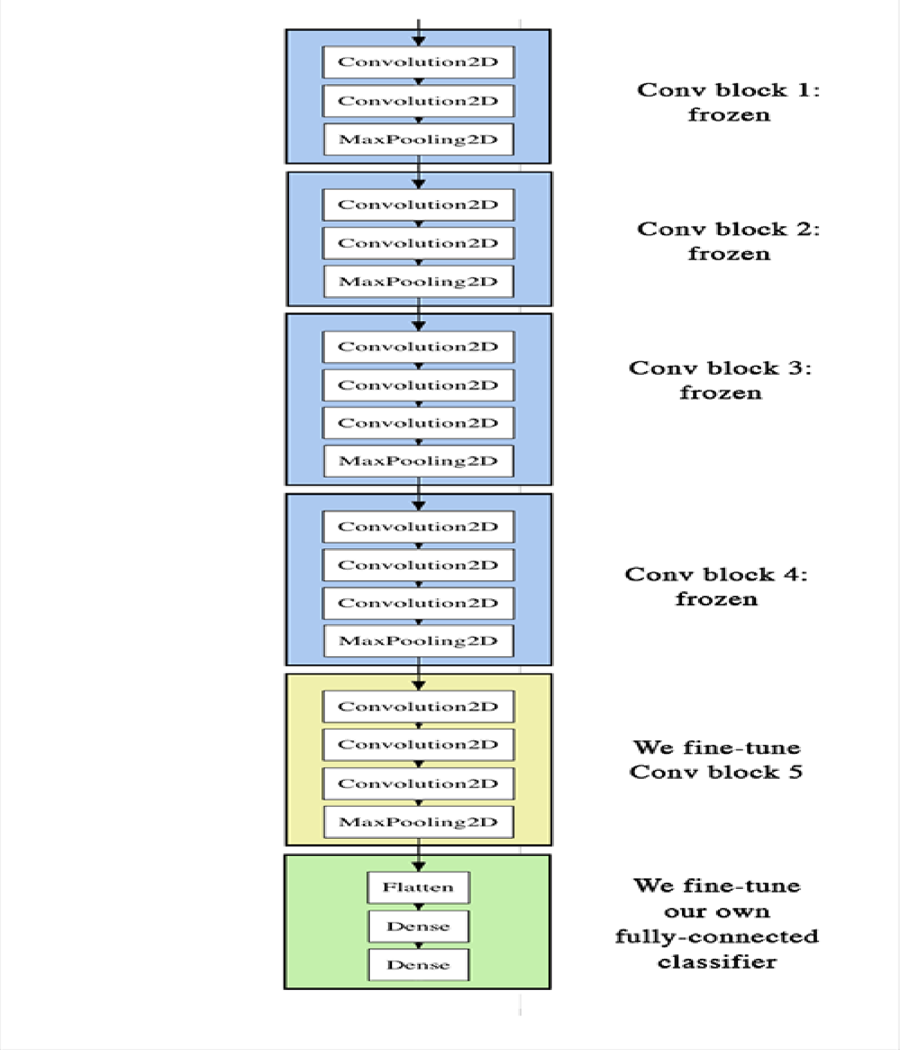


https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Transfer Learning DL models

- **Fine Tuning Off-the-shelf Pre-trained Models**





https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Object Detection DL models

- **Deep learning approach for object detection and classification**

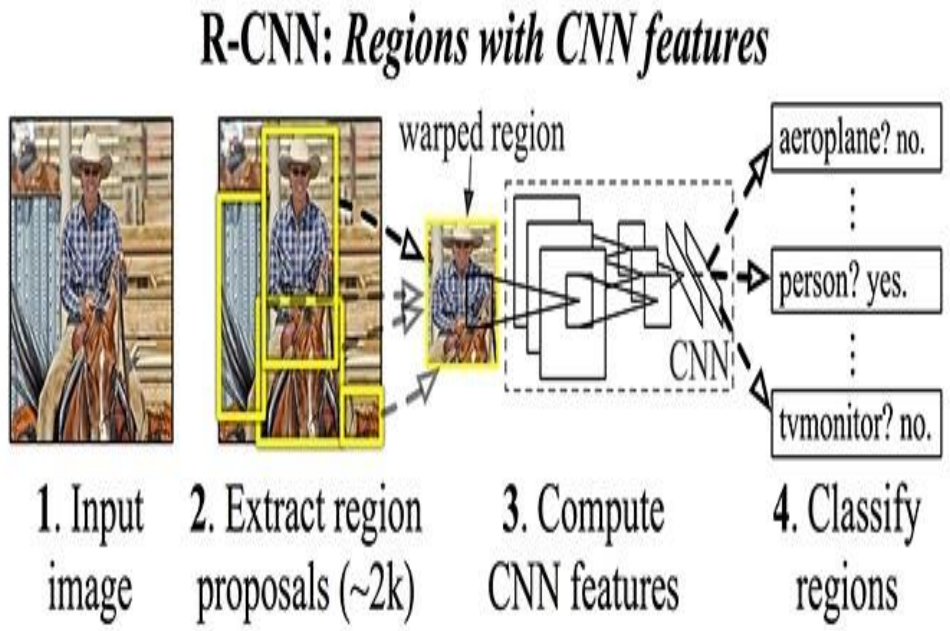
OverFeat

One of the first advances in using deep learning for object detection was OverFeat from NYU published in 2013. They proposed a multi-scale sliding window algorithm using Convolutional Neural Networks (CNNs).

R-CNN

Quickly after OverFeat, Regions with CNN features or R-CNN from Ross Girshick, et al. at the UC Berkeley was published which boasted an almost 50% improvement on the object detection challenge. What they proposed was a three stage approach:

Extract possible objects using a region proposal method (the most popular one being Selective Search).
Extract features from each region using a CNN.
Classify each region with SVMs.



https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/

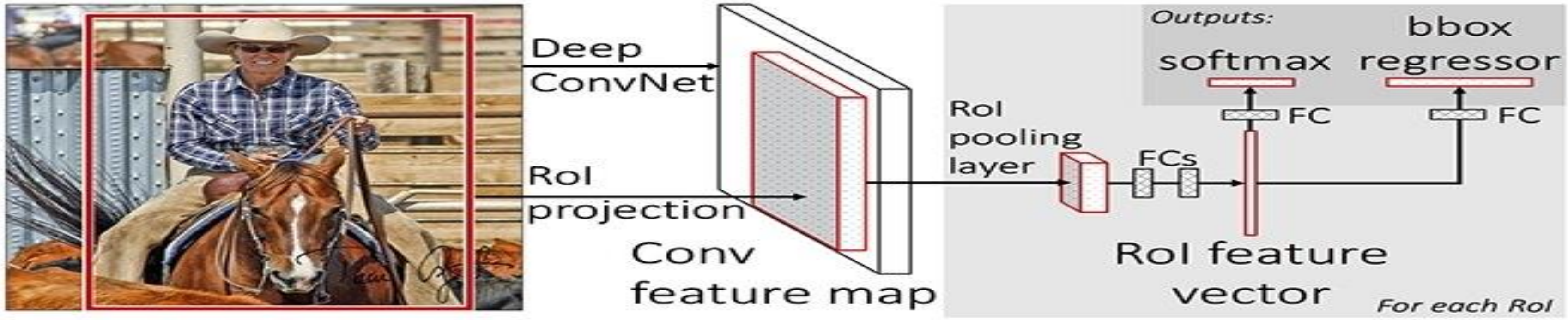
Object Detection DL models

- **Deep learning approach for object detection and classification**

Fast R-CNN

This approach quickly evolved into a purer deep learning one, when a year later Ross Girshick (now at Microsoft Research) published Fast R-CNN. **Similar to R-CNN, it used Selective Search to generate object proposals, but instead of extracting all of them independently and using SVM classifiers, it applied the CNN on the complete image and then used both Region of Interest (RoI) Pooling on the feature map with a final feed forward network for classification and regression.**

Not only was this approach faster, but having the RoI Pooling layer and the fully connected layers allowed the model to be end-to-end differentiable and easier to train. The biggest downside was that the model still relied on **Selective Search (or any other region proposal algorithm),** which became the bottleneck when using it for inference.



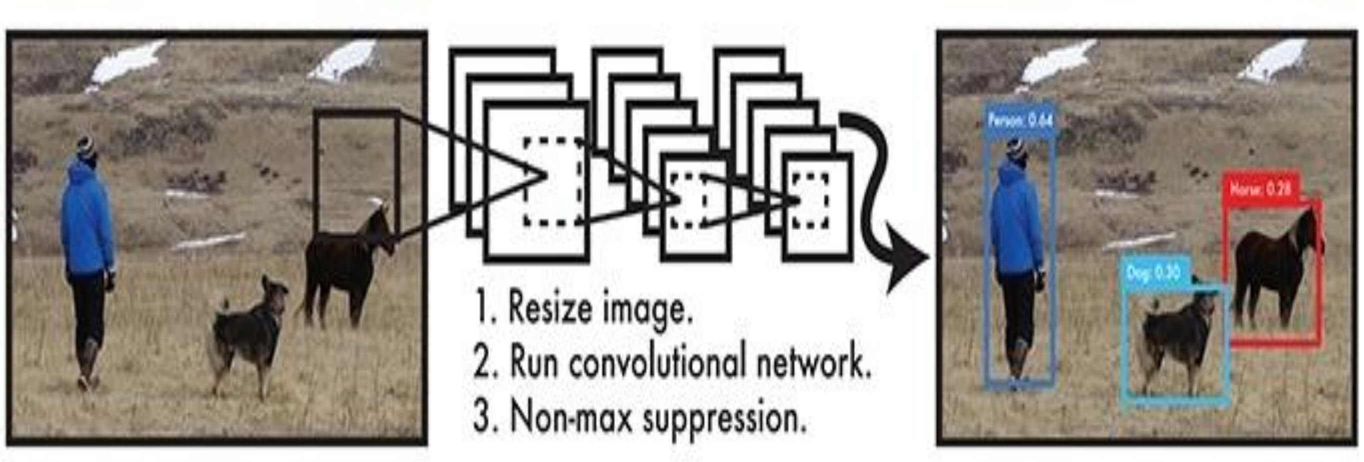
https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/

Object Detection DL models

- **Deep learning approach for object detection and classification**

YOLO

Shortly after that, [You Only Look Once: Unified, Real-Time Object Detection] (YOLO) paper published by Joseph Redmon (with Girshick appearing as one of the co-authors). YOLO proposed a simple convolutional neural network approach which has both great results and high speed, allowing for the first time real time object detection.



YOLO Architecture.
Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." 2016.

https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/

Object Detection DL models

- **Deep learning approach for object detection and classification**

Faster R-CNN

Subsequently, Faster R-CNN authored by Shaoqing Ren (also co-authored by Girshick, now at Facebook Research)**, the third iteration of the R-CNN series.**

Faster R-CNN added what they called a Region Proposal Network (RPN), in an attempt to get rid of the Selective Search algorithm and make the model completely trainable end-to-end.

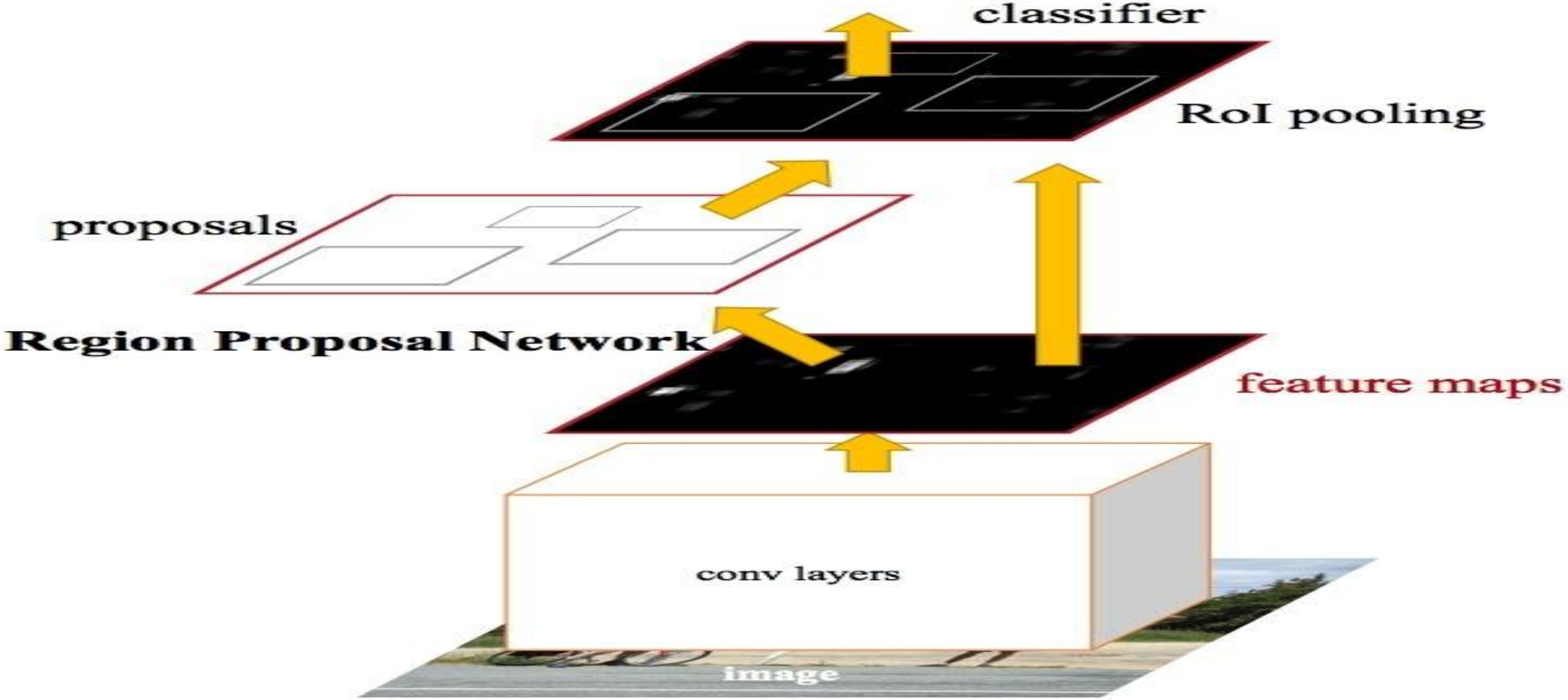
It has the task to output objects based **on an “objectness” score.**

These objects are used by the RoI Pooling and fully connected layers for classification.

https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/

Object Detection DL models

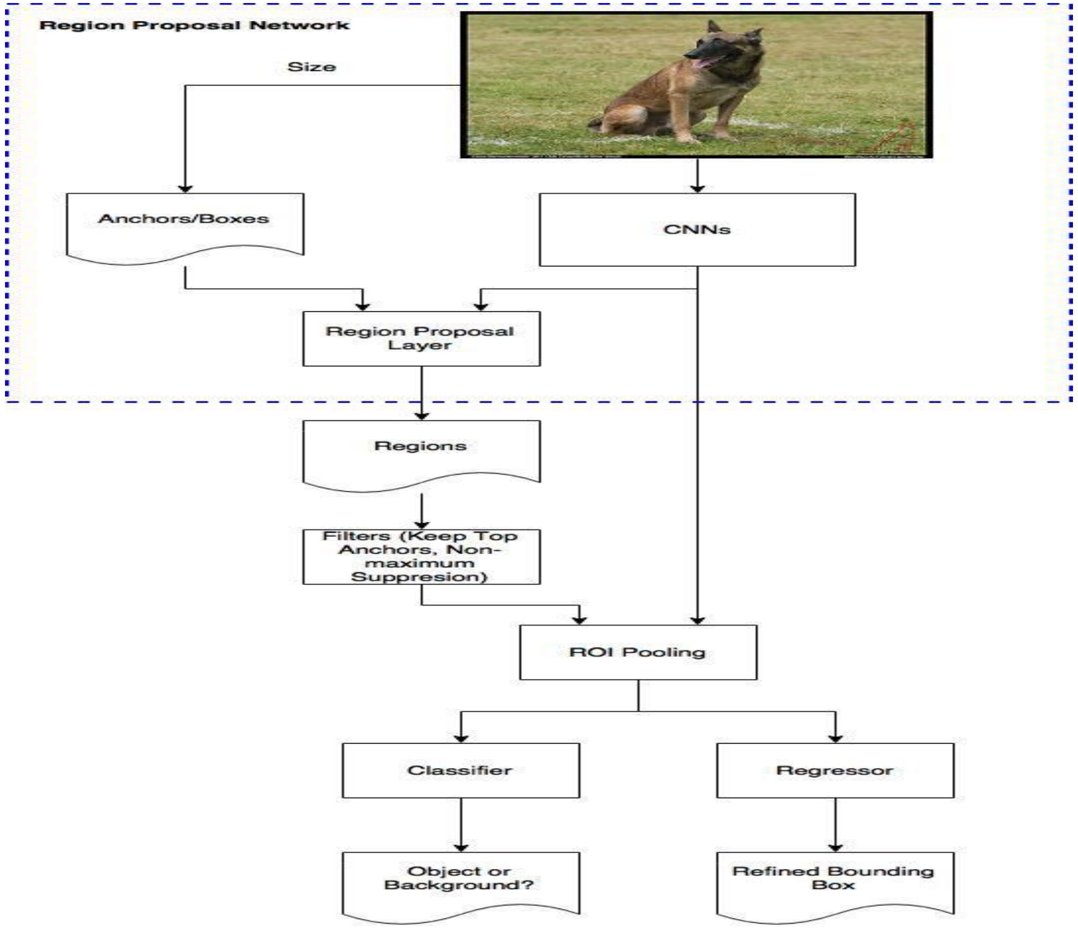
- **Deep learning approach for object detection and classification**

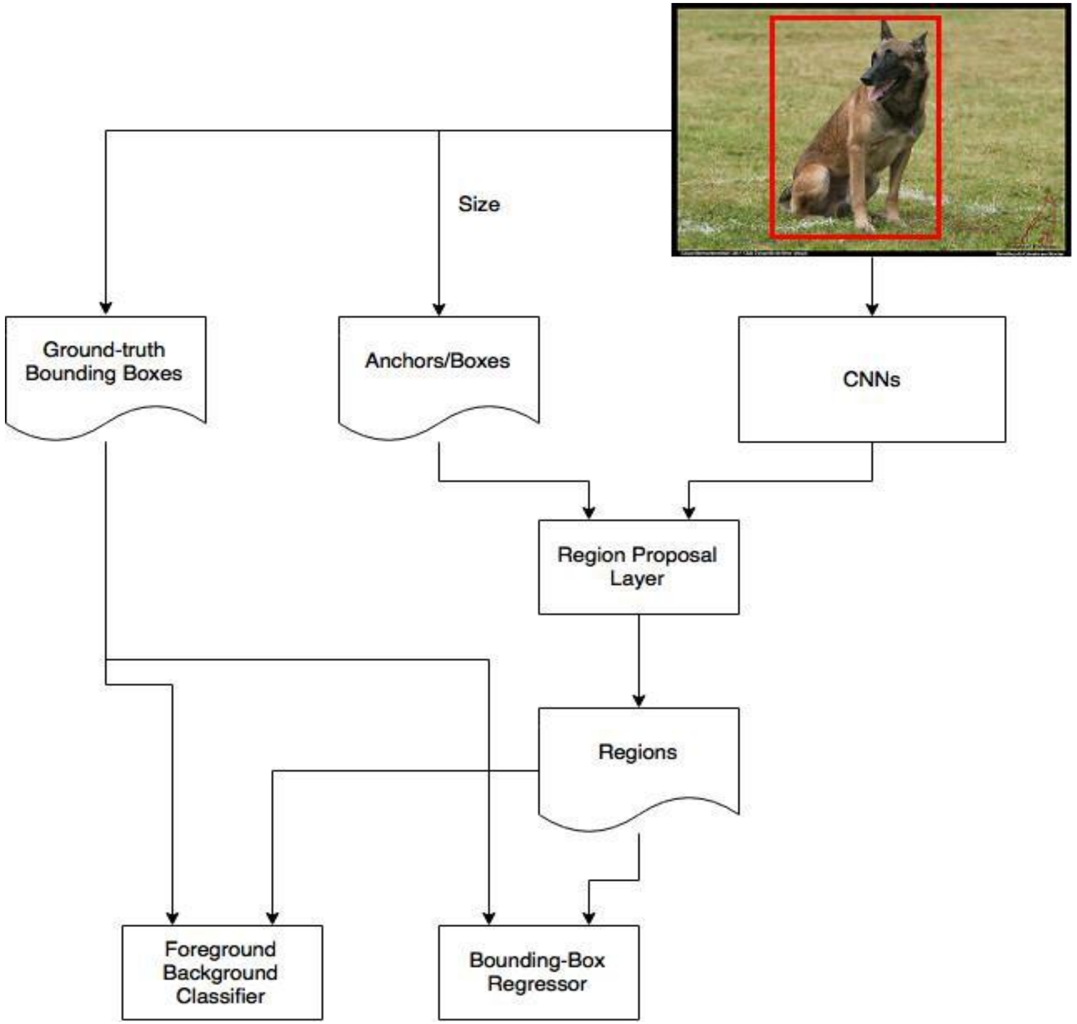


https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/

Object Detection DL models

- **Deep learning approach for object detection and classification**

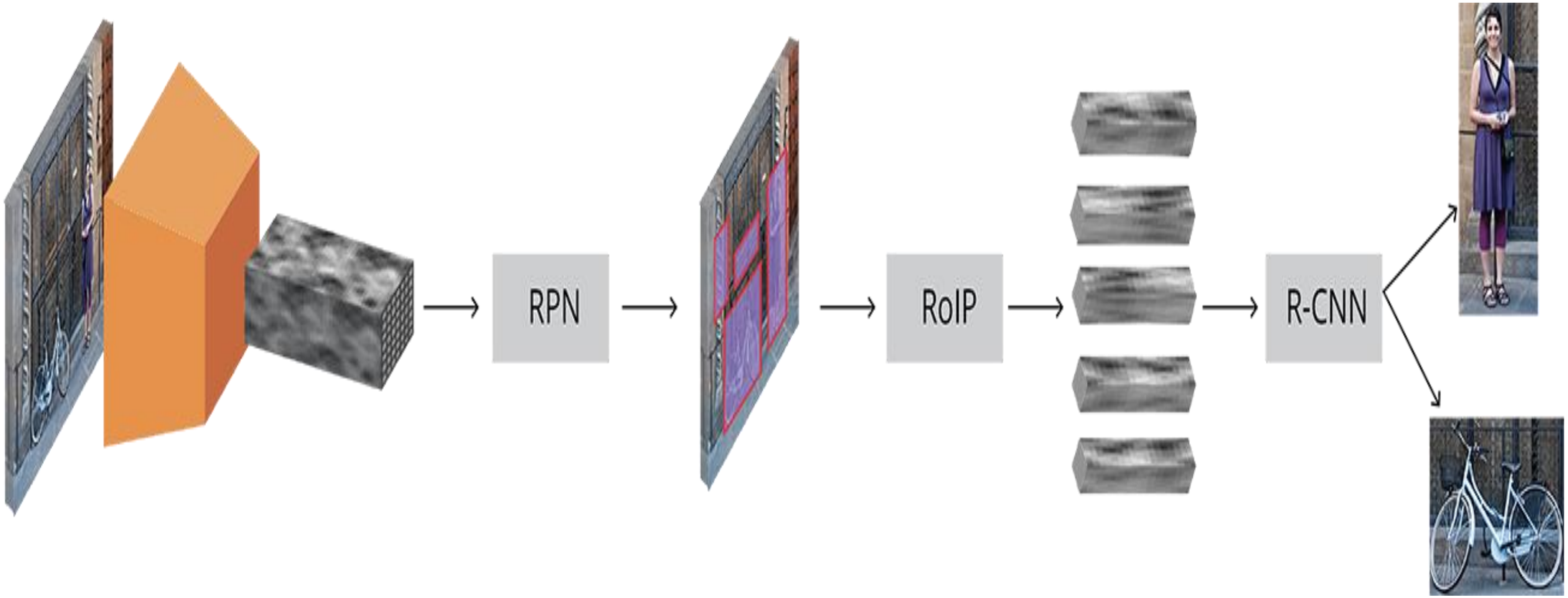




https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

Object Detection DL models

- **Deep learning approach for object detection and classification**



Complete Faster R-CNN architecture

https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Object Detection DL models

- **Deep learning approach for object detection and classification**

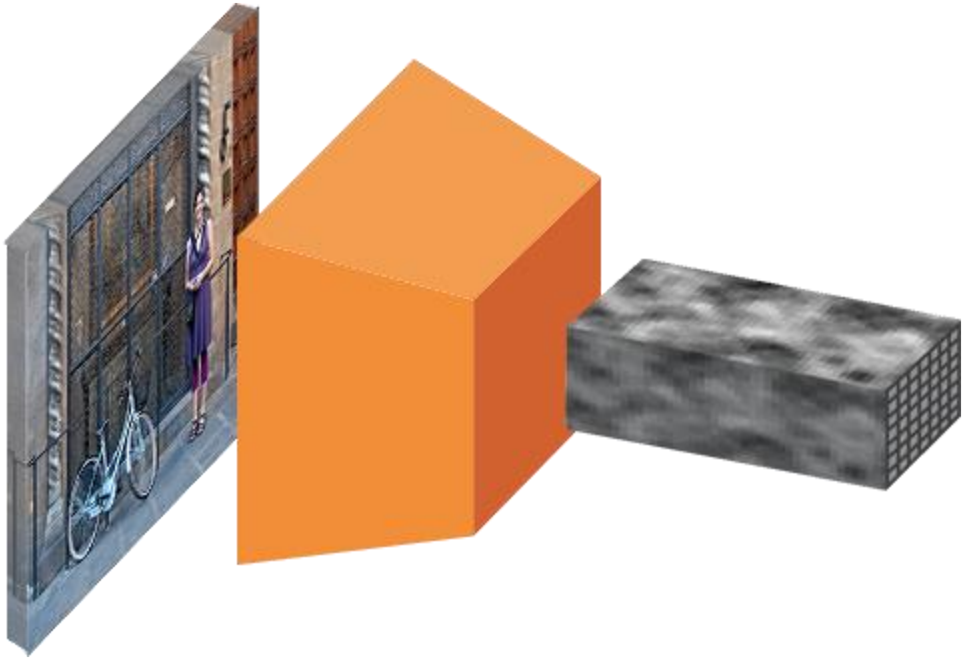
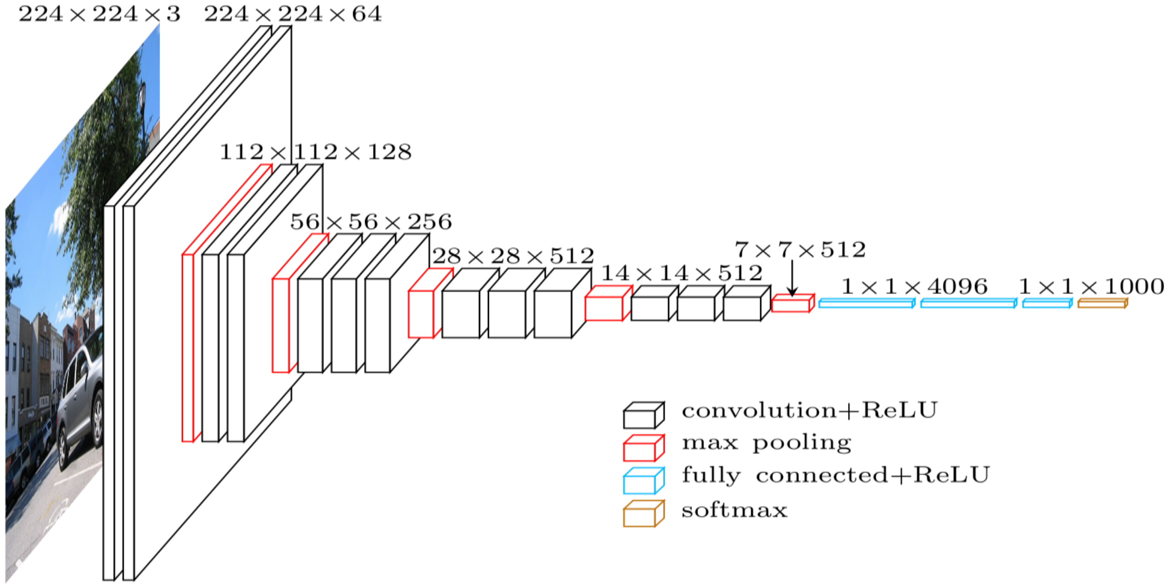


Image to convolutional feature map

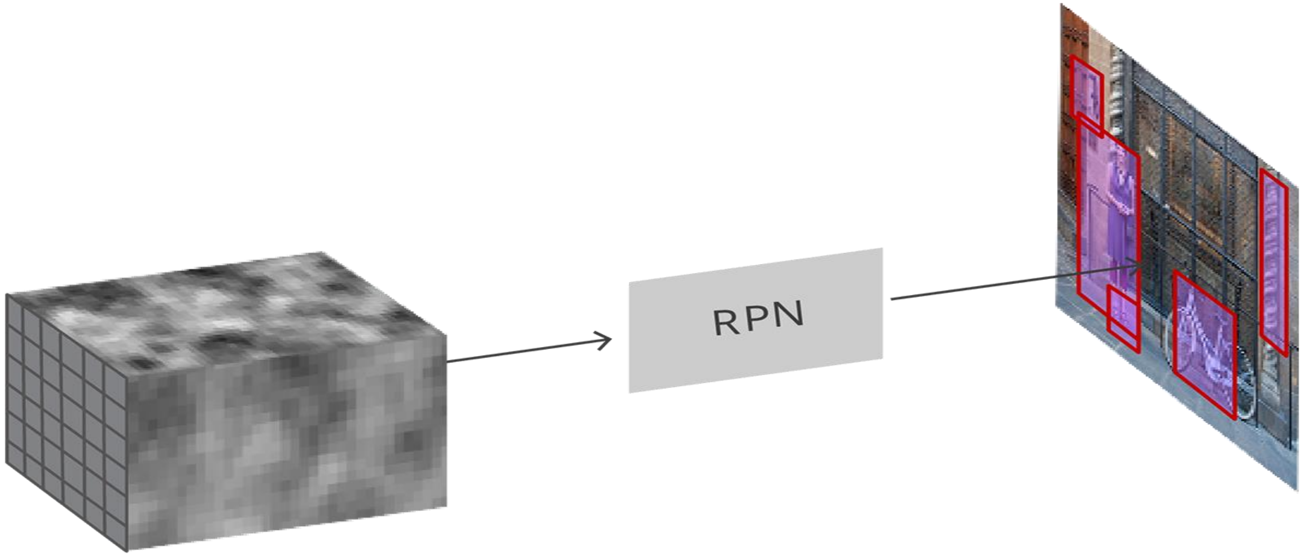


VGG architecture

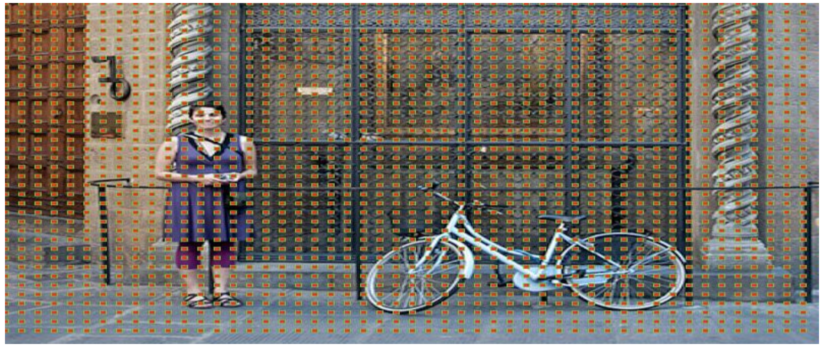
https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Object Detection DL models

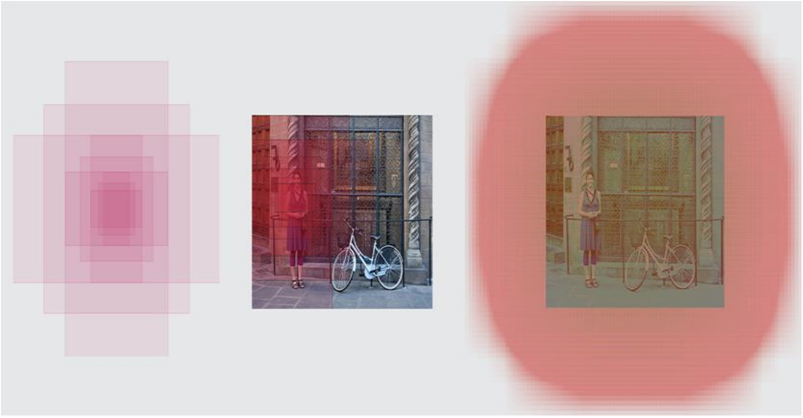
- **Deep learning approach for object detection and classification**



The RPN takes the convolutional feature map and generates proposals over the image



Anchor centers throught the original image



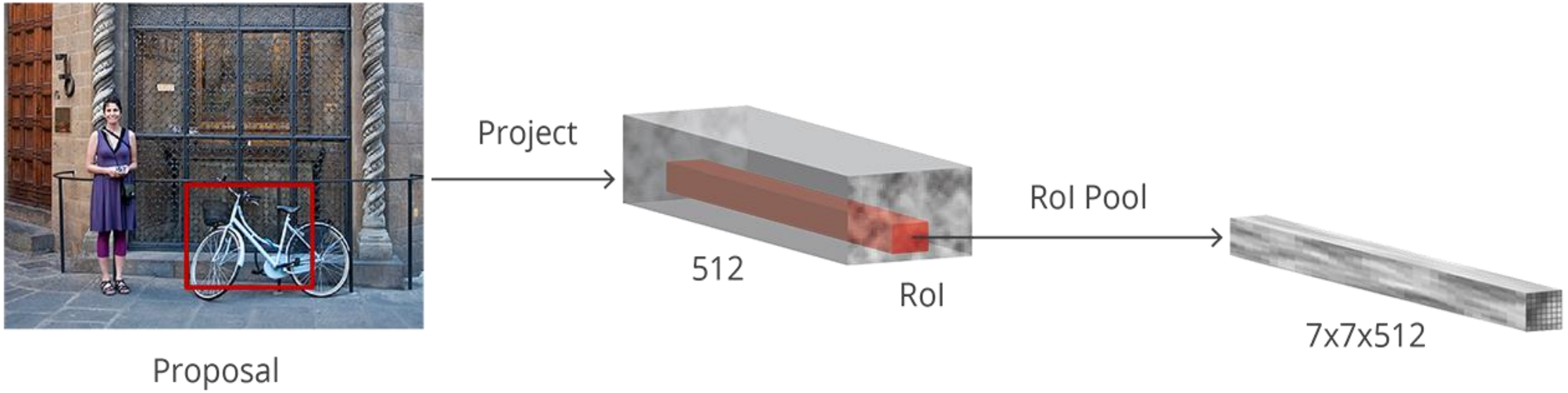
Left: Anchors, Center: Anchor for a single point, Right: All anchors

https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Object Detection DL models

- **Deep learning approach for object detection and classification**

Region of Interest Pooling



Region of Interest Pooling

https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Object Detection DL models

- **Deep learning approach for object detection and classification**

Region-based Convolutional Neural Network

Region-based convolutional neural network (R-CNN) is the final step in Faster R-CNN's pipeline. After getting a convolutional feature map from the image, using it to get object proposals with the RPN and finally extracting features for each of those proposals (via RoI Pooling),
we finally need to use these features for classification.
R-CNN tries to mimic the final stages of classification CNNs where a fully-connected layer is used to output a score for each possible object class.

R-CNN has two different goals:

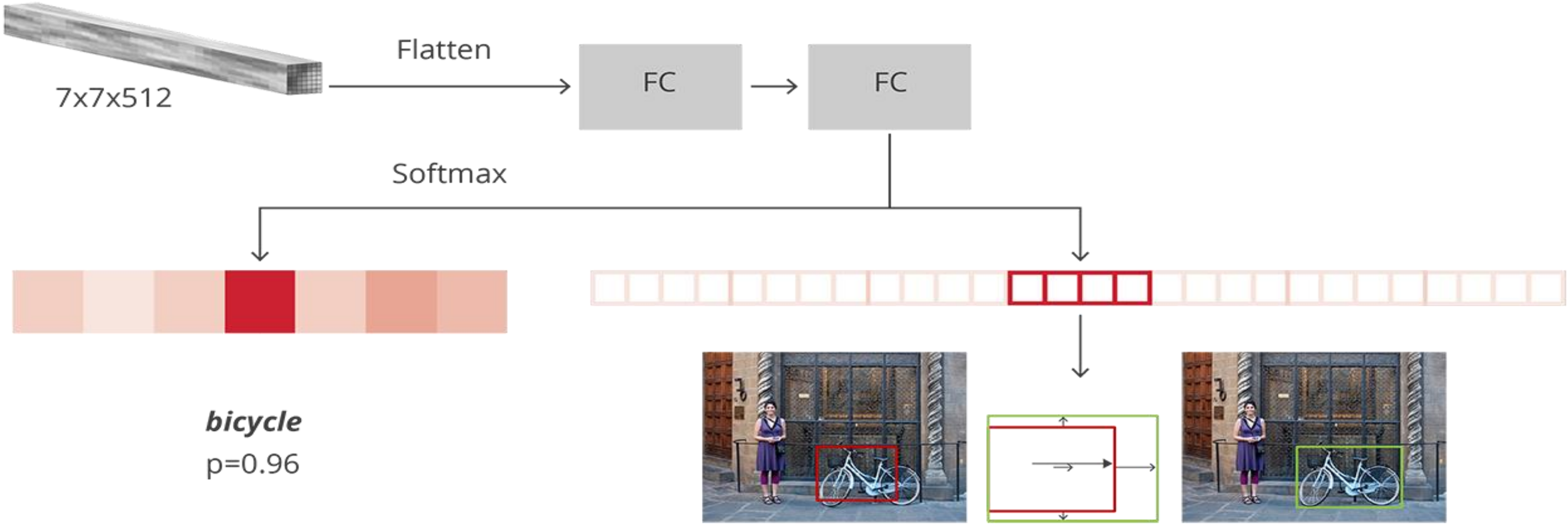
Classify proposals into one of the classes, plus a background class (for removing bad proposals).
Better adjust the bounding box for the proposal according to the predicted class.

https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Object Detection DL models

- **Deep learning approach for object detection and classification**

Region-based Convolutional Neural Network



https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Object Detection DL models

- **Deep learning approach for object detection and classification**

Region-based Convolutional Neural Network

In the original paper, Faster R-CNN was trained using a multi-step approach, training parts independently and merging the trained weights before a final full training approach. Since then, it has been found that doing end-to-end, joint training leads to better results.

After putting the complete model together we end up with 4 different losses, two for the RPN and two for R-CNN. We have the trainable layers in RPN and R-CNN, and we also have the base network which we can train (fine-tune) or not.

https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Object Detection DL models

- **Deep learning approach for object detection and classification**

SSD and R-FCN

Finally, there are two notable papers, **Single Shot Detector** (SSD) which takes on YOLO by using multiple sized convolutional feature maps achieving better results and speed, and **Region-based Fully Convolutional Networks** (R-FCN) which takes the architecture of Faster R-CNN but with only convolutional networks.

https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/


Q&A