AI VIETNAM
All-in-One Course

From Basic CNN to ConvNext

(*Performance Evaluation on CIFAR-10 Dataset*)

Vinh Dinh Nguyen
PhD in Computer Science

Year 2023

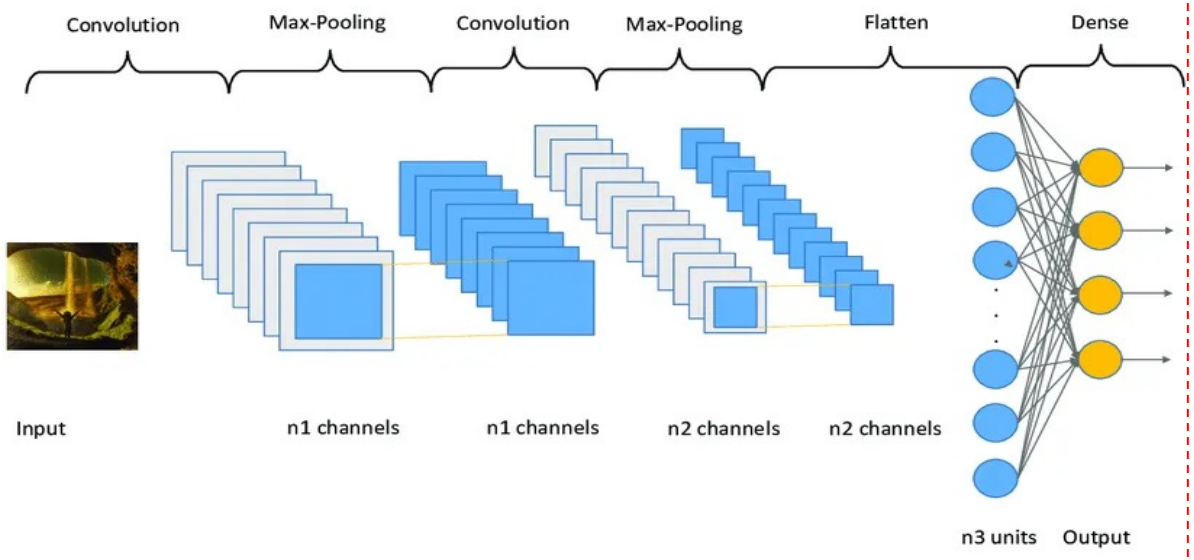
Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

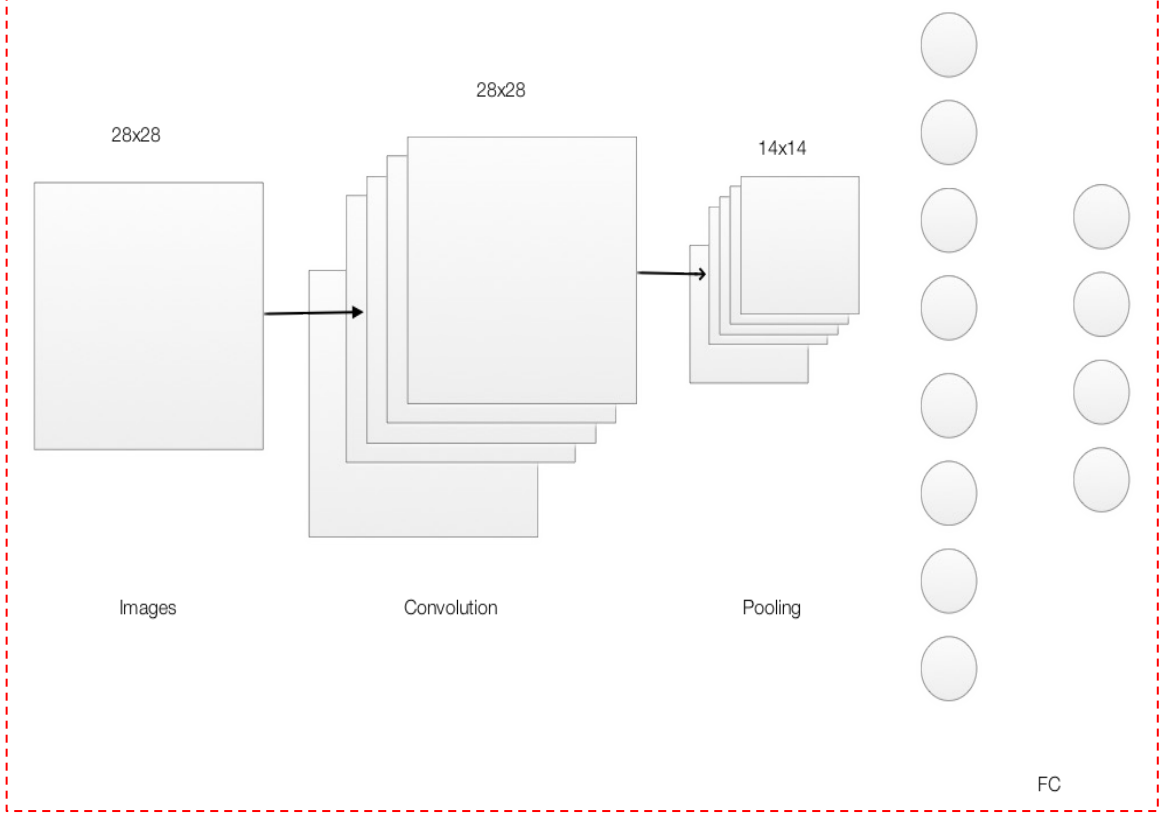
Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

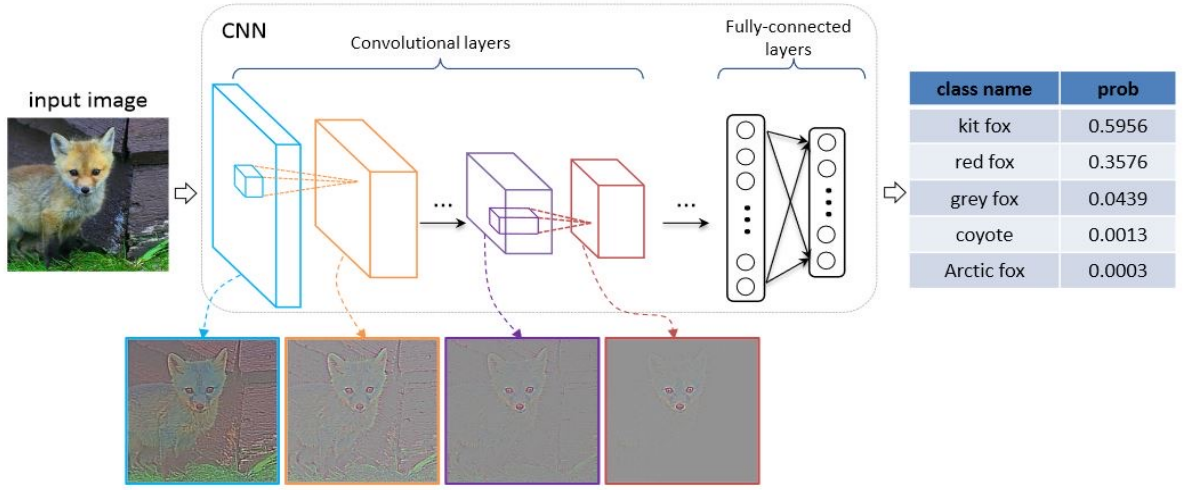
CNN: Basic Concepts



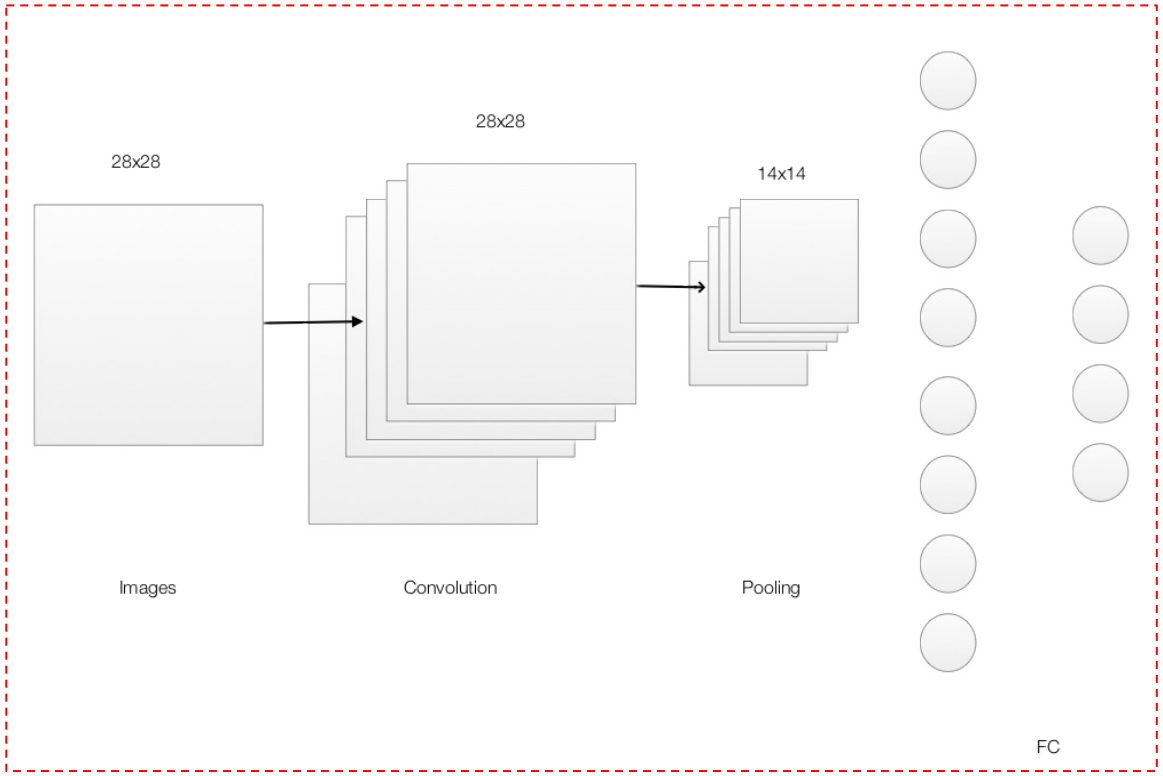
A convolutional neural network composes of convolution layers, pooling layers and fully connected layers(FC).



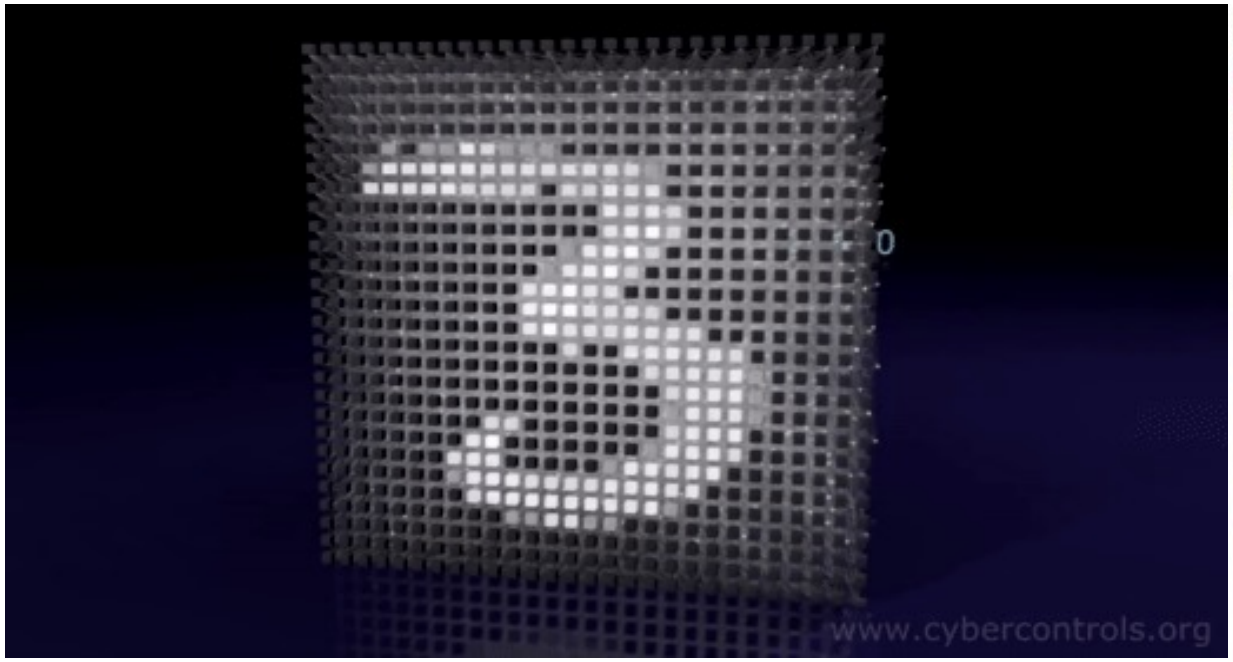
CNN: Basic Concepts

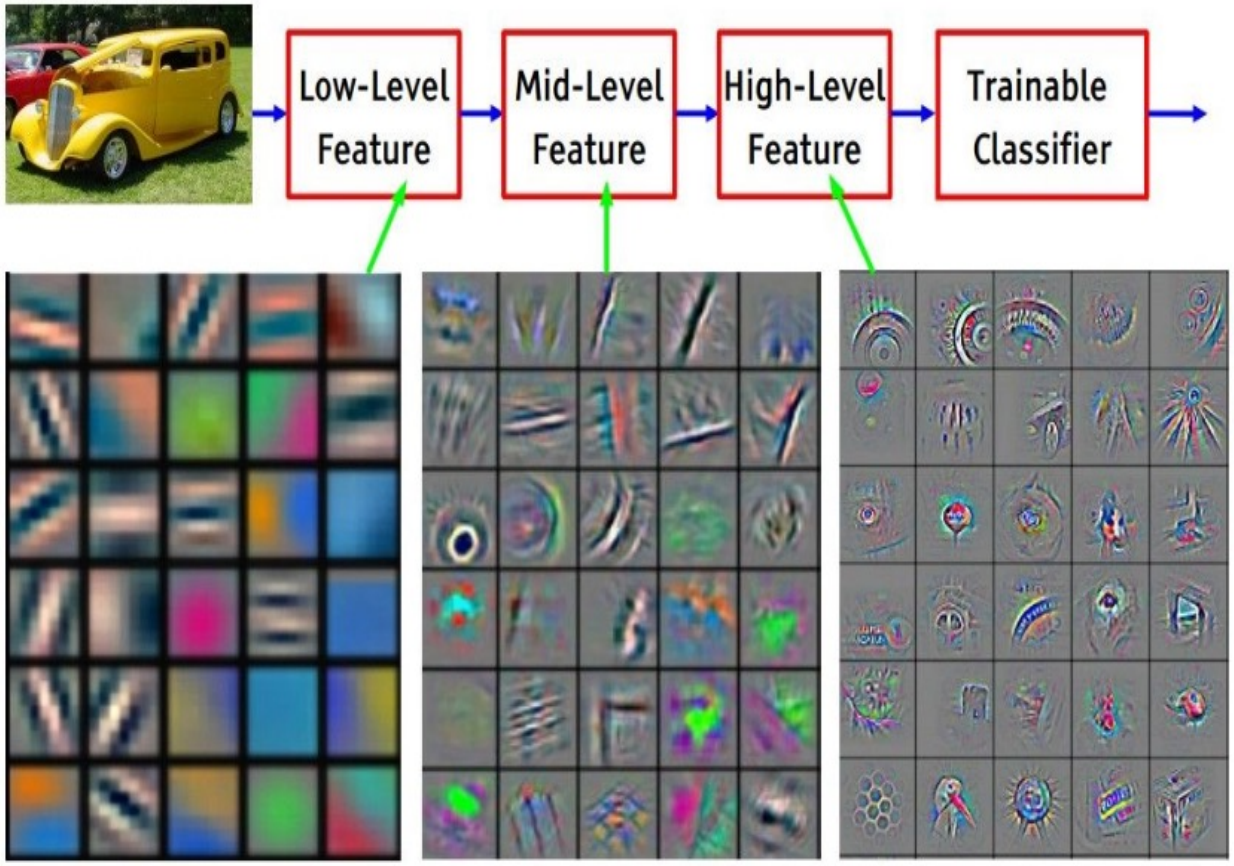


A convolutional neural network composes of convolution layers, pooling layers and fully connected layers(FC).



CNN: Basic Concepts



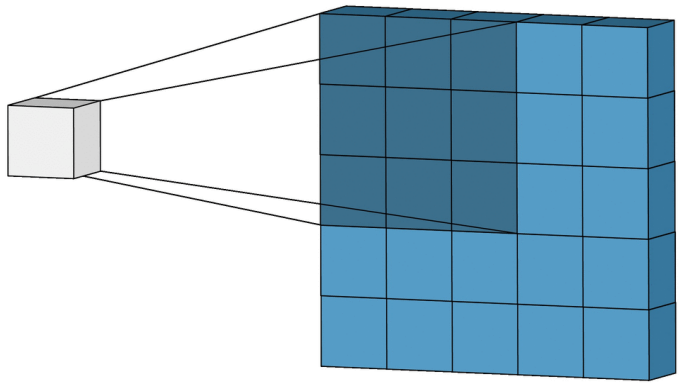


In the lower layers within a CNN, the units/neurons learn low-level features within the image such as lines, edges, contours etc. The higher layers learn more abstract features of the image such as shapes

Neural Network Simulation credit to Denis Dmitriev

CNN: Basic Concepts

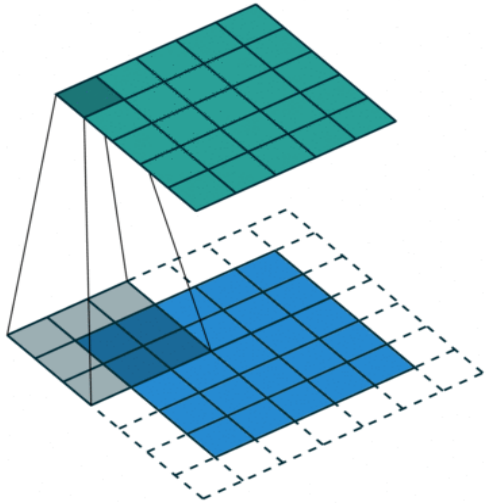
Simple Convolution



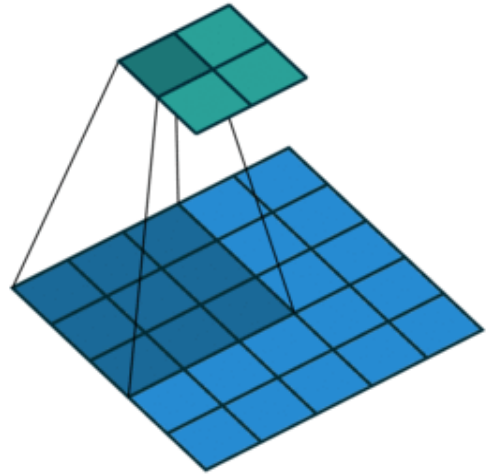
3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

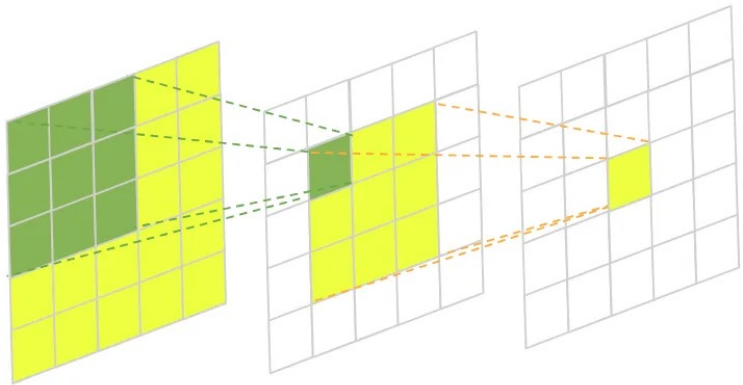
Padding Concept



Stride Concept

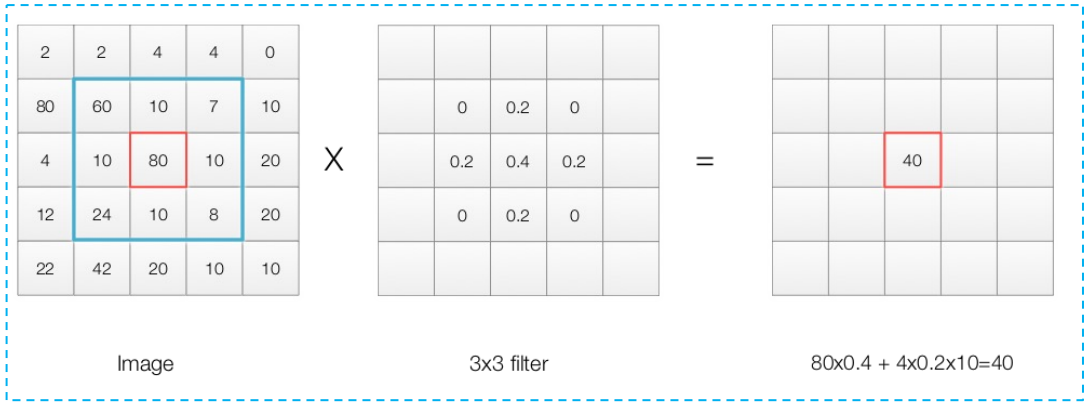


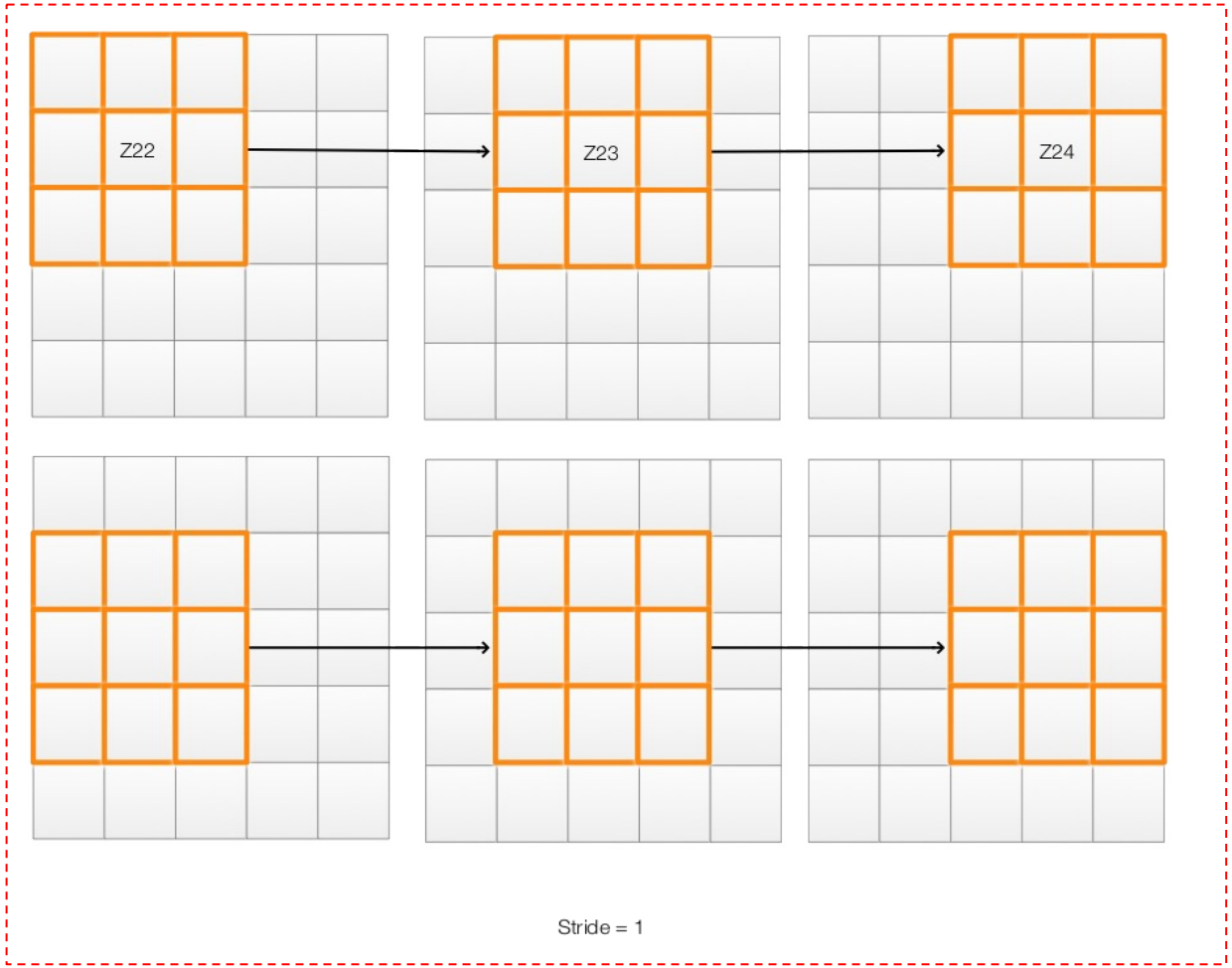
Reception Field



CNN: Basic Concepts

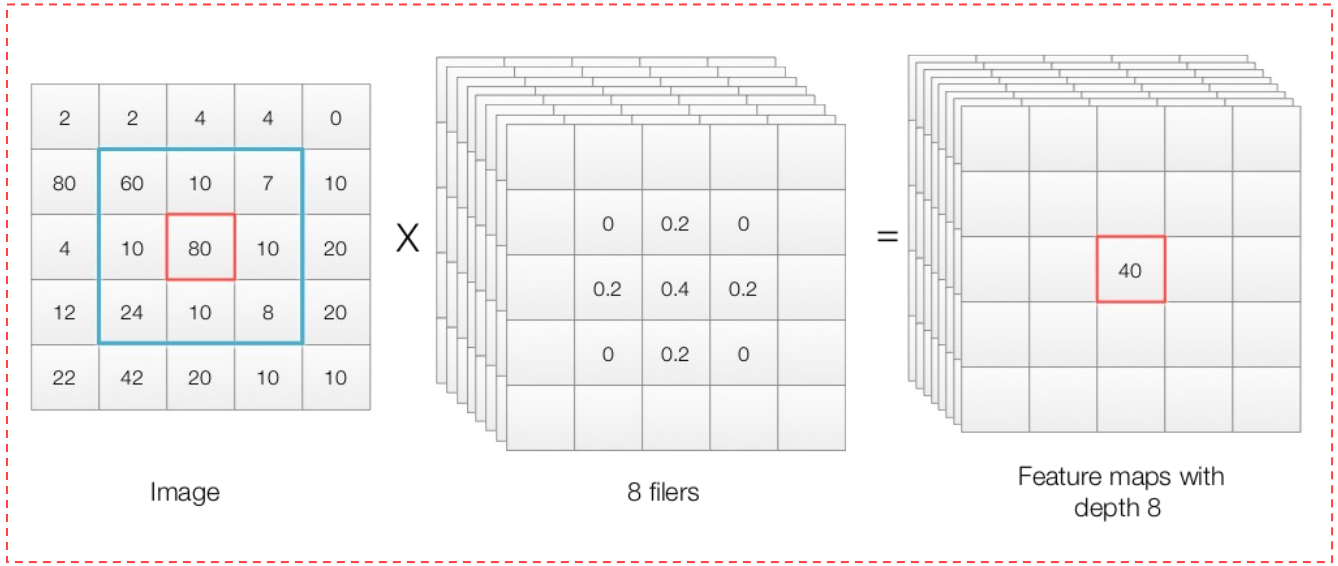
Filter in Image Processing

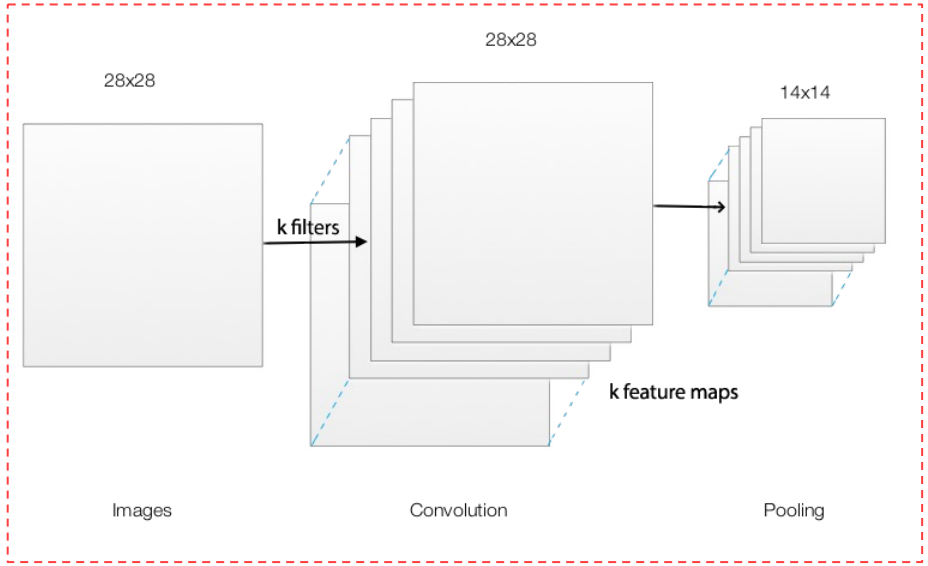




Move the filter 1 pixel at a time from left to right and top to bottom until we process every pixel.

CNN: Basic Concepts

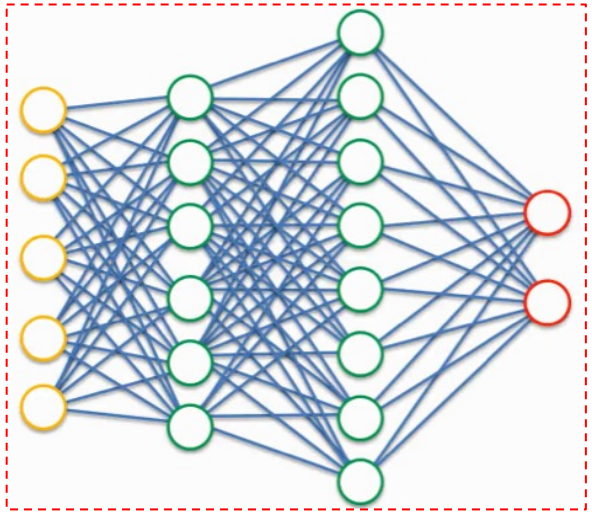




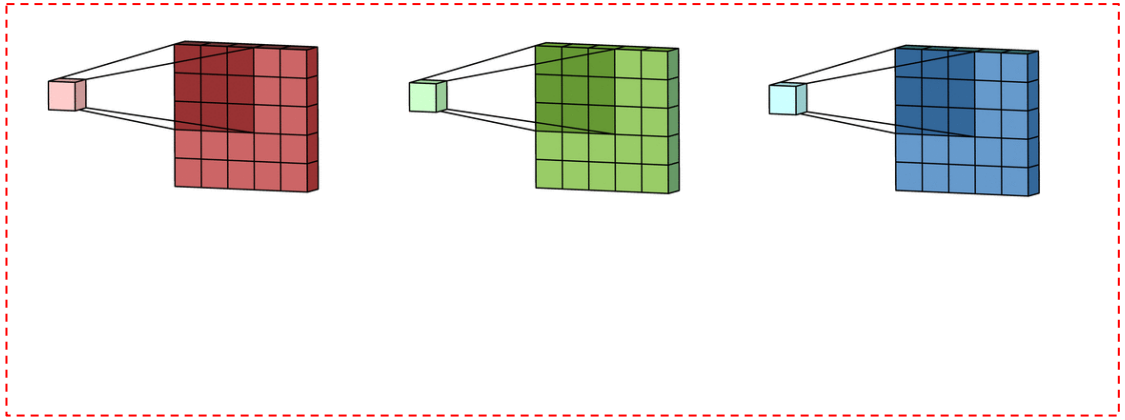
Apply filters which each generates an output that we call **feature map**. If k-features map is created, we have feature maps with depth k

CNN: Basic Concepts

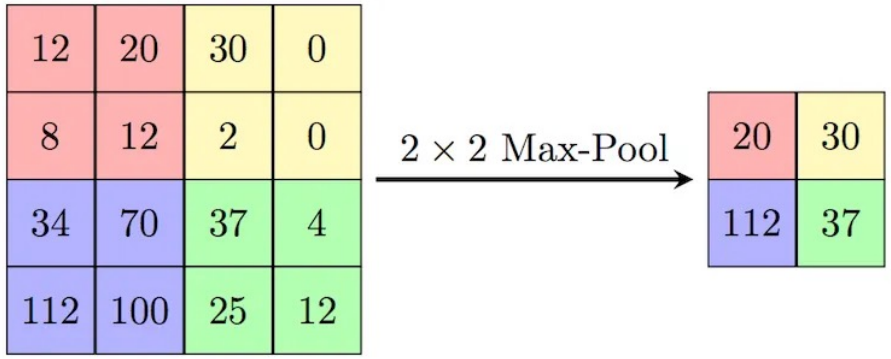
Fully connected



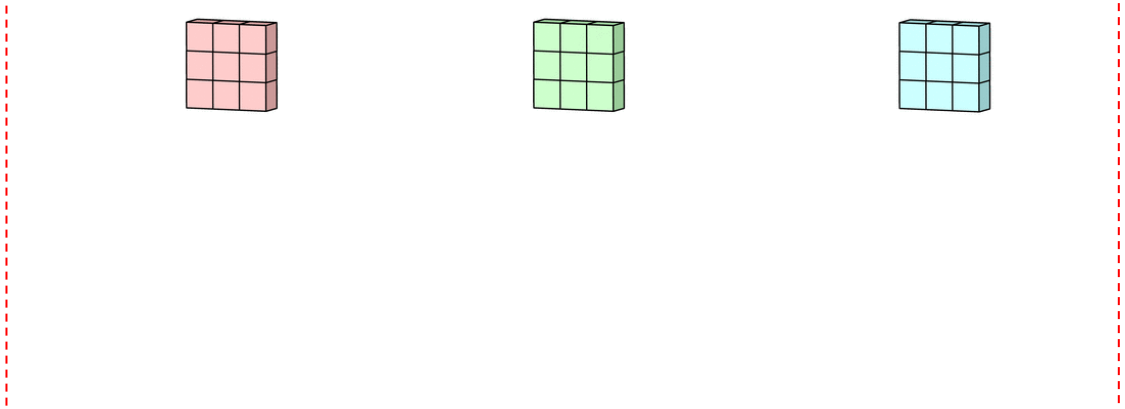
Feature Accumulation



Max Pooling Concept

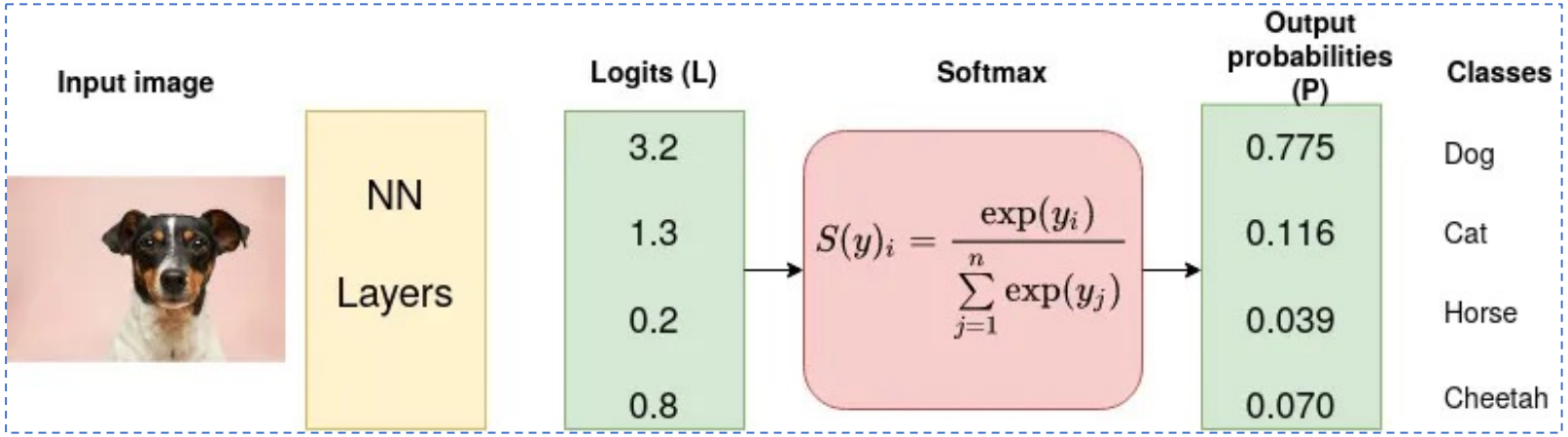


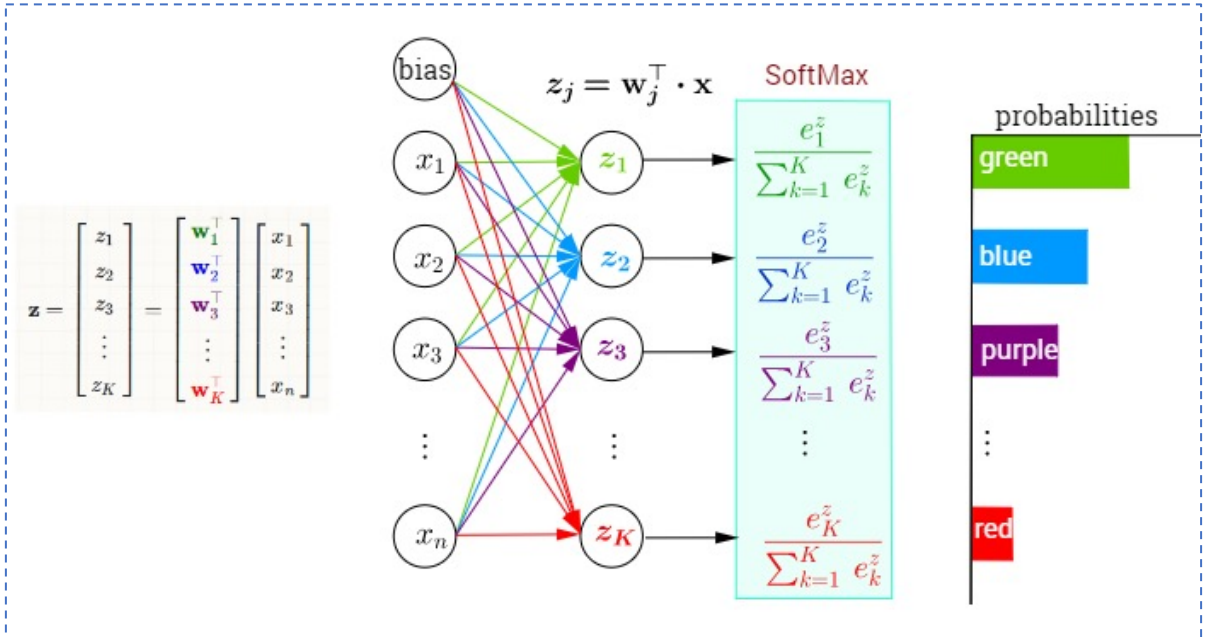
Feature Aggreation



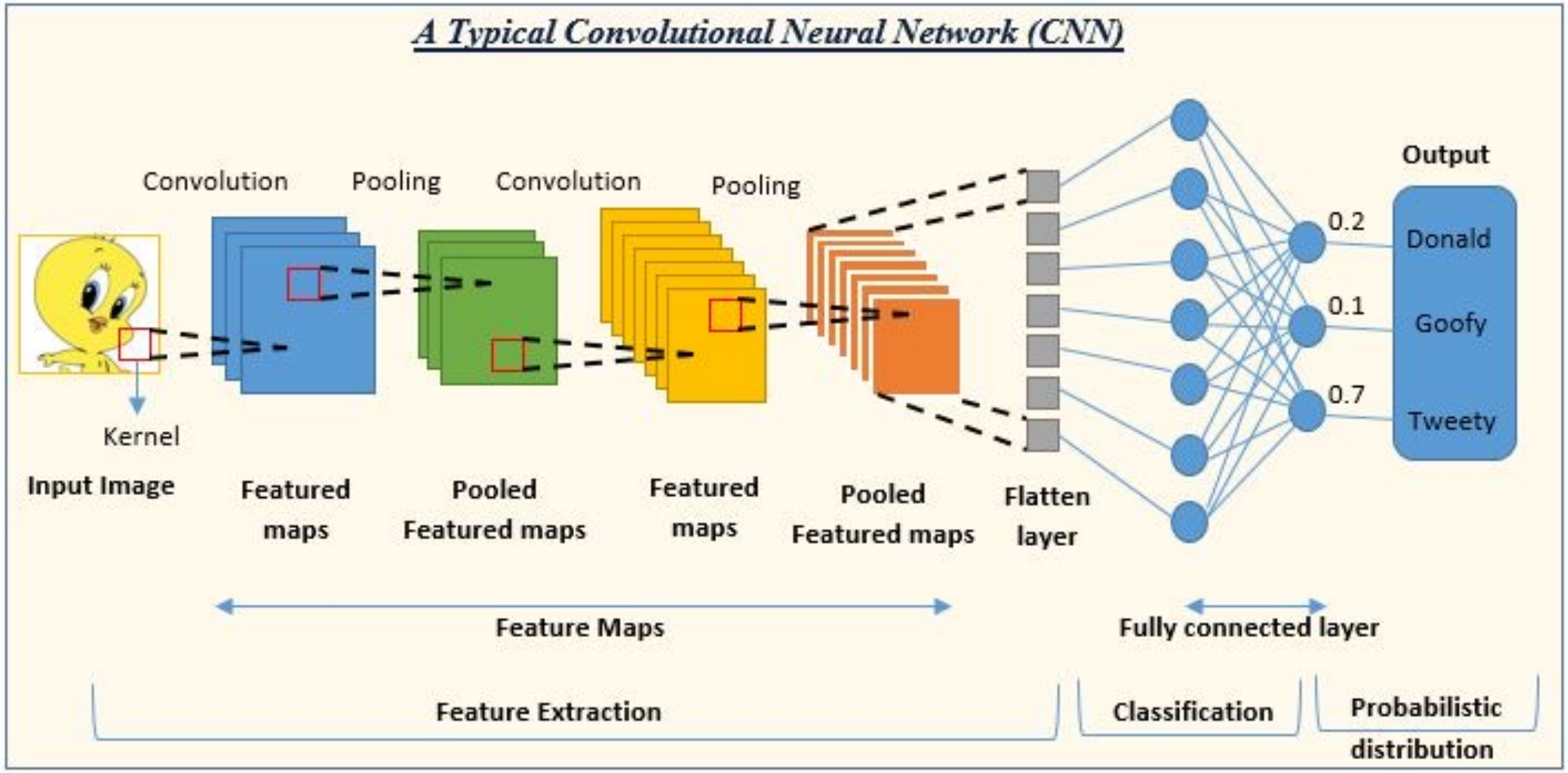
CNN: Basic Concepts

Softmax Activation Function





CNN: Basic Concepts



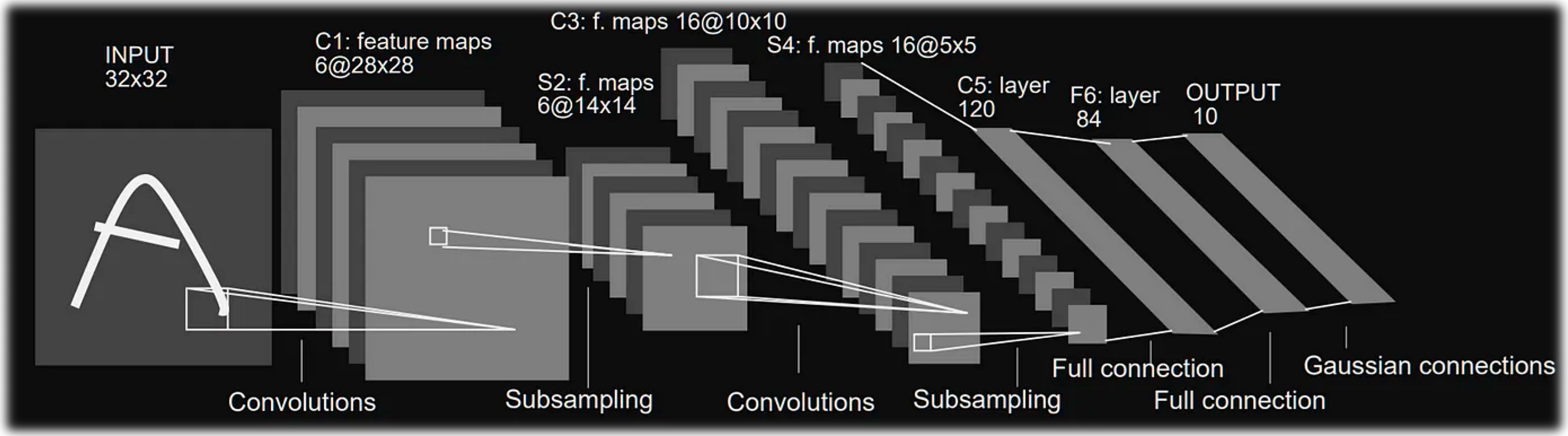
https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/

Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

LetNet

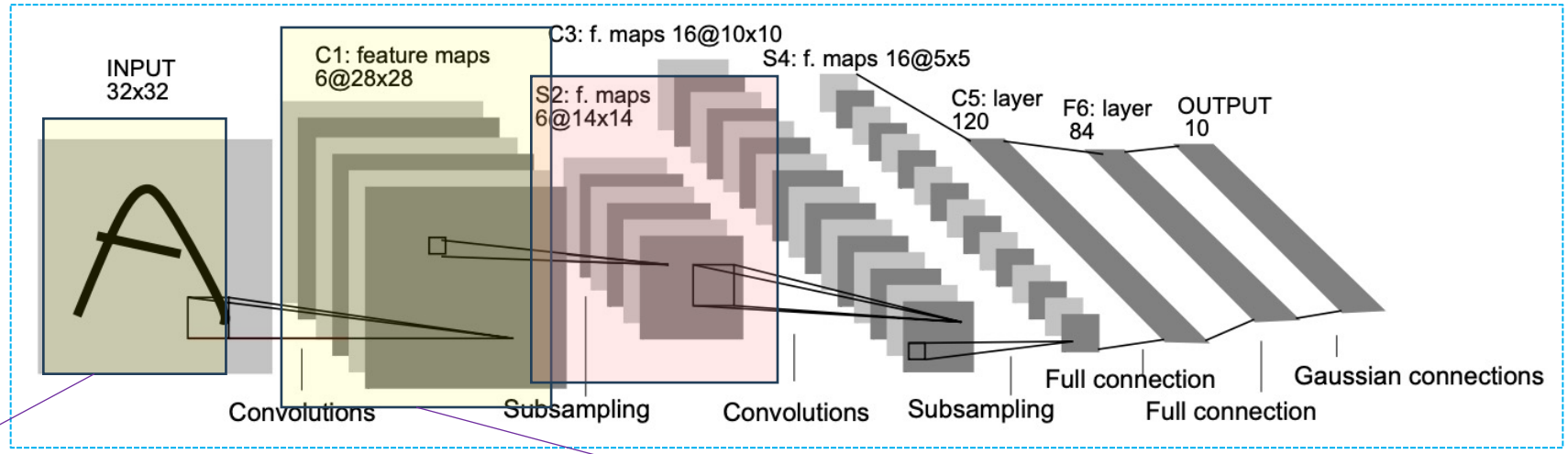
'Hello World' in the domain of Convolution Neural Networks



handwritten and machine-printed character recognition

It was introduced by Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner in 1998

LetNet



Input layer

LeNet takes as input grayscale images of size 32x32 pixels

32x32x1

C1: convolution layer

Number of filters (feature map): $n_c = 6$
Filter size: $F = 5$
Padding, $P = 0$
Stride, $S = 1$
Training parameters = Weight + Bias = 156
Number of connections = 122304

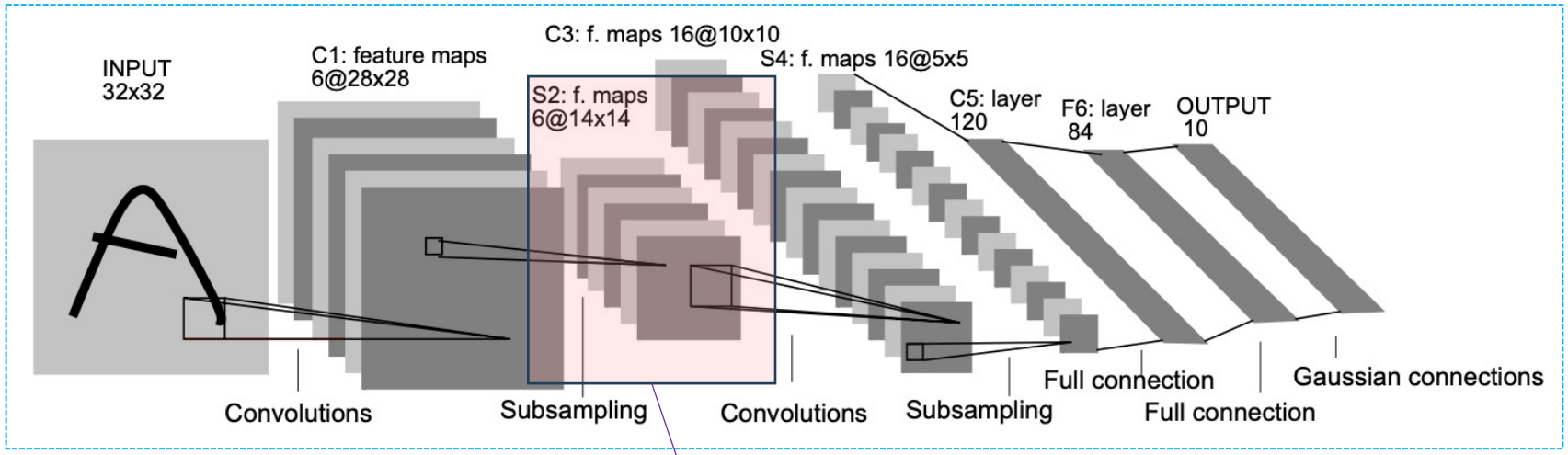
6

28x28x6

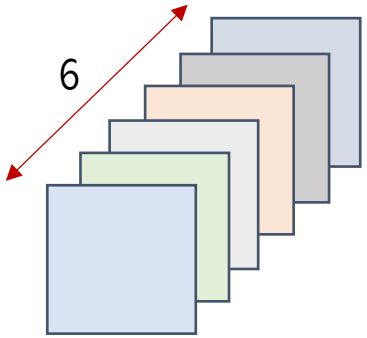
Why?

(shape of width of the filter * shape of height of the filter * number of filters in the previous layer+1)*number of filters)

LetNet



S2: Pooling layer



Number of filters (feature map): $n_c = 6$

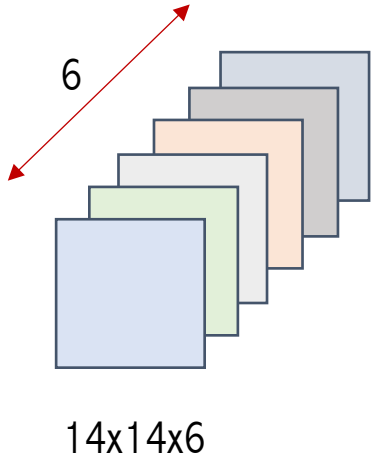
Filter size: $F = 2$

Padding, $P = 0$

Stride, $S = 1$

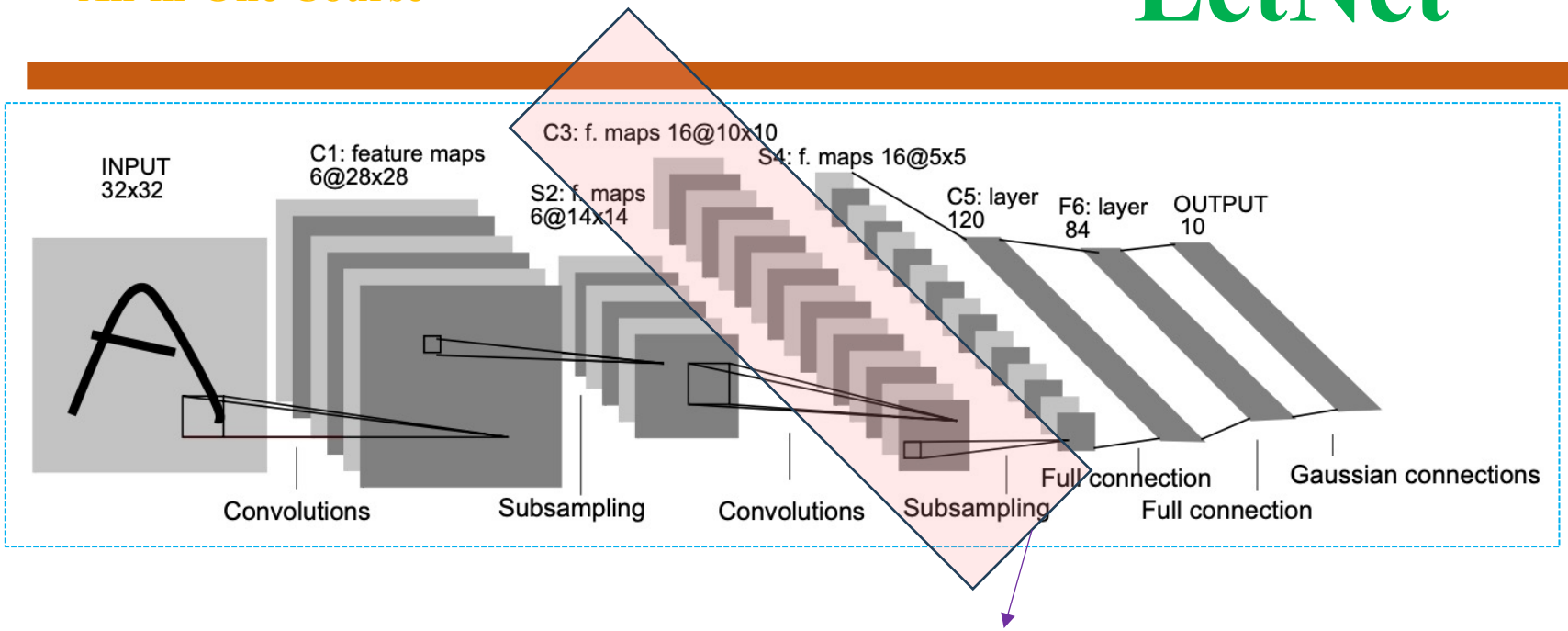
Training parameters = (cofficient + bias)x filters = 12

Number of connections = 5580



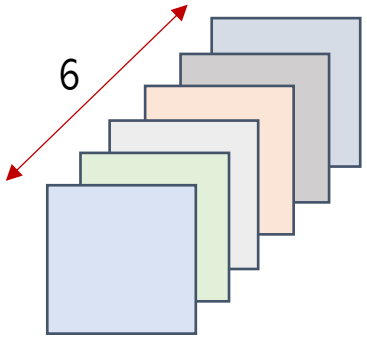
Why?

LetNet



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

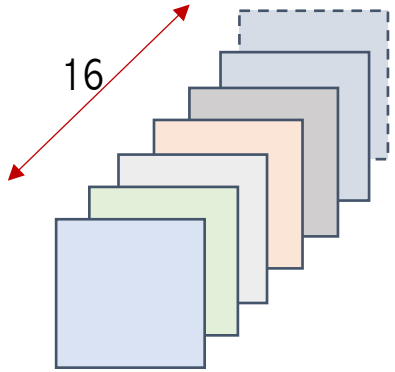
Each column indicates which feature map in S2 are combined by the units in a particular feature map of C3



14x14x6

C3: Convolutional layer

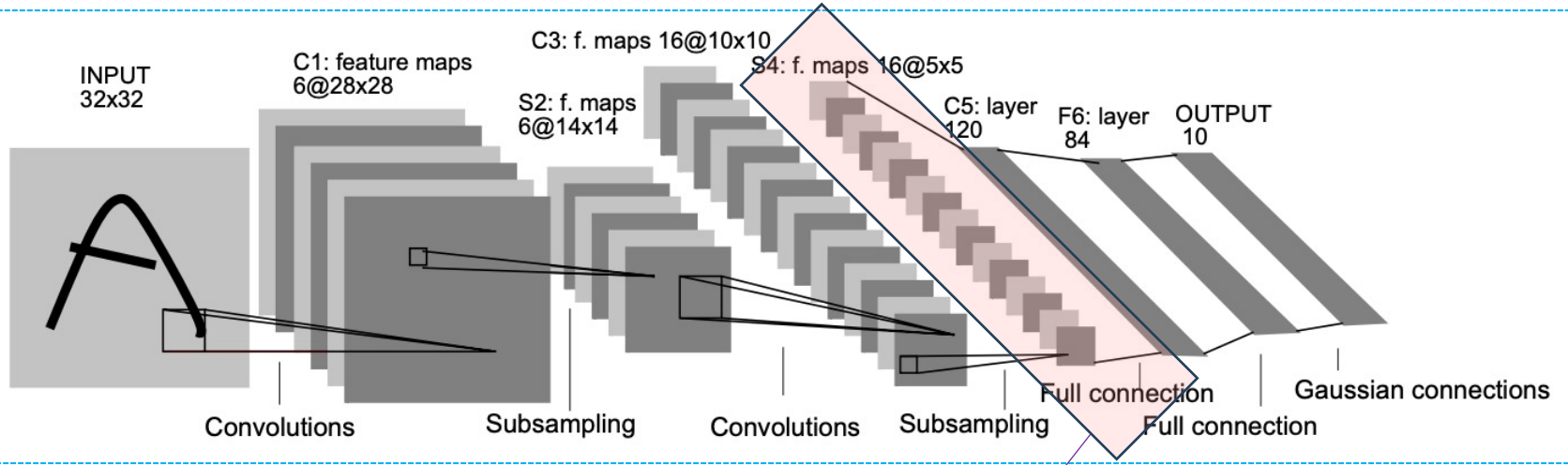
Number of filters (feature map): $n_c = 16$
Filter size: $F = 5$
Padding, $P = 0$
Stride, $S = 1$
Training parameters = (Weight + bias) = 1516
Number of connections = 151600



10x10x16

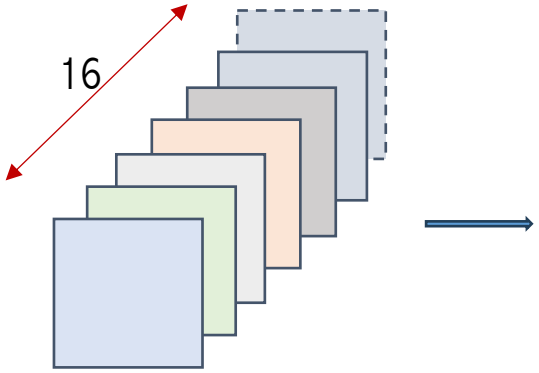
Why?

LetNet



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

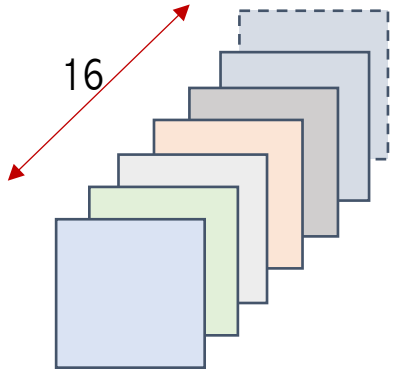
Each column indicates which feature map in S2 are combined by the units in a particular feature map of C3



10x10x16

S4: Convolutional layer

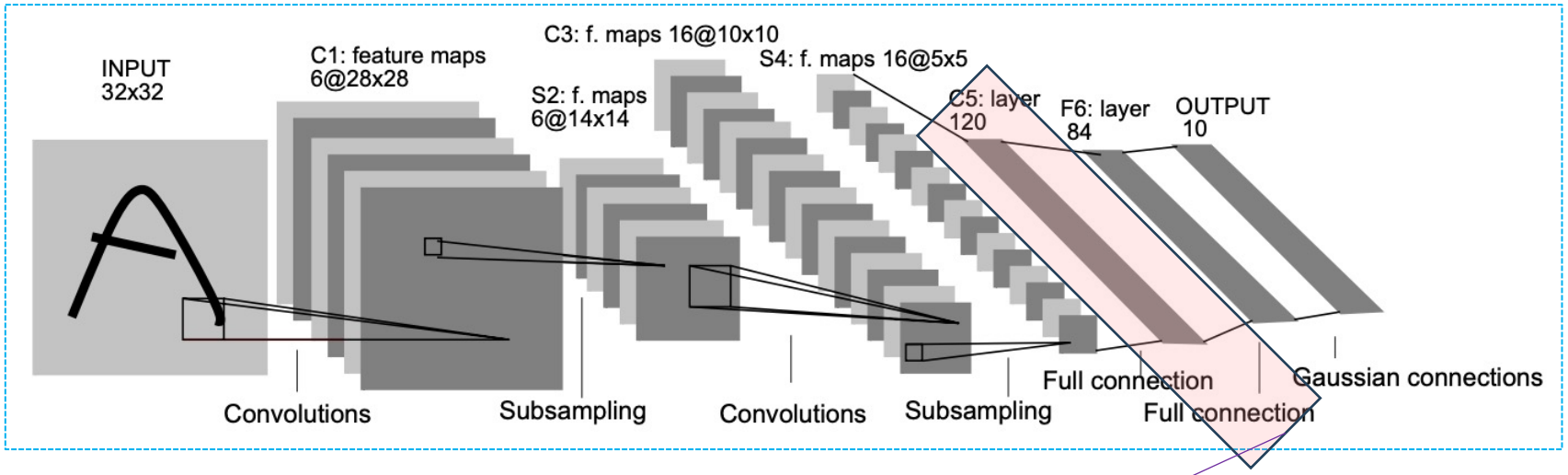
Number of filters (feature map): $n_c = 16$
Filter size: $F = 2$
Padding, $P = 0$
Stride, $S = 2$
Training parameters = 32
Number of connections = 2000

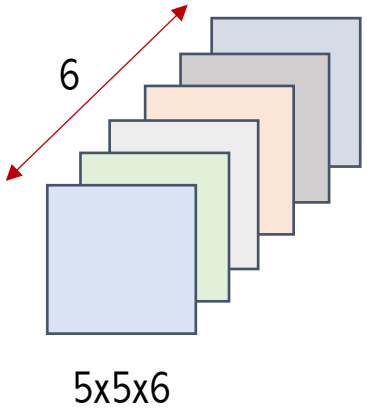


5x5x16

Why?

LetNet



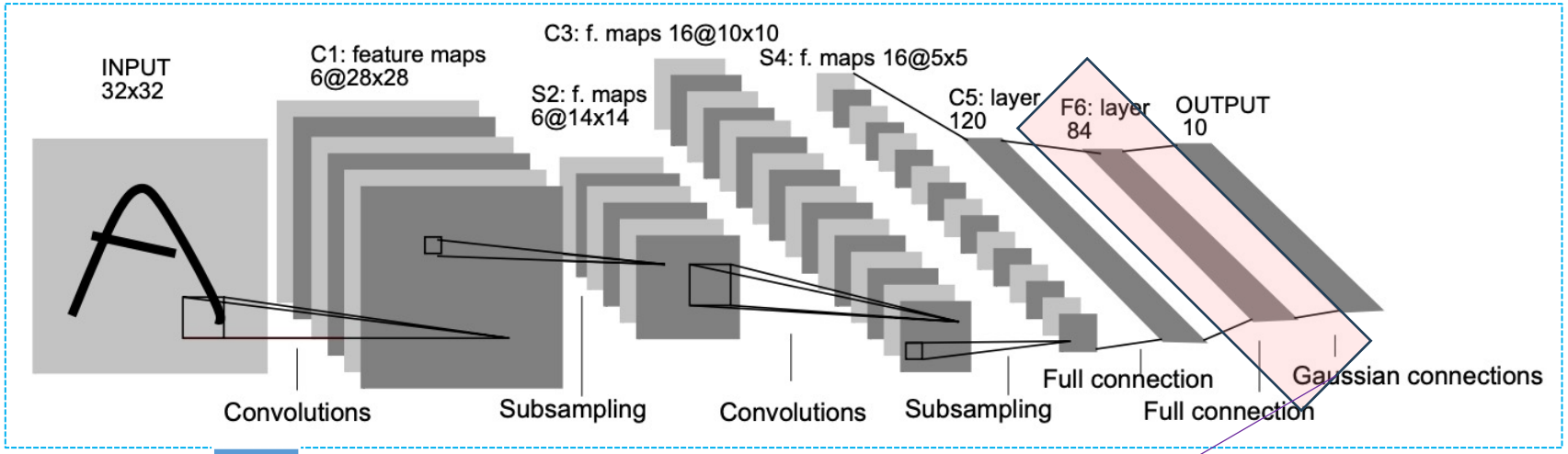


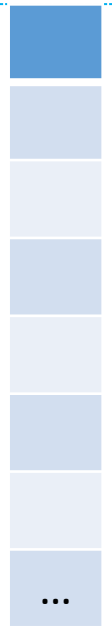
C5: Fully connected convolutional layer

Number of filters (feature map): N/A
Filter size: 5
Padding, N/A
Stride, N/A
Training parameters = (Weight + bias) = 48120

Why?

LetNet

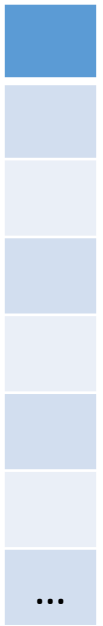




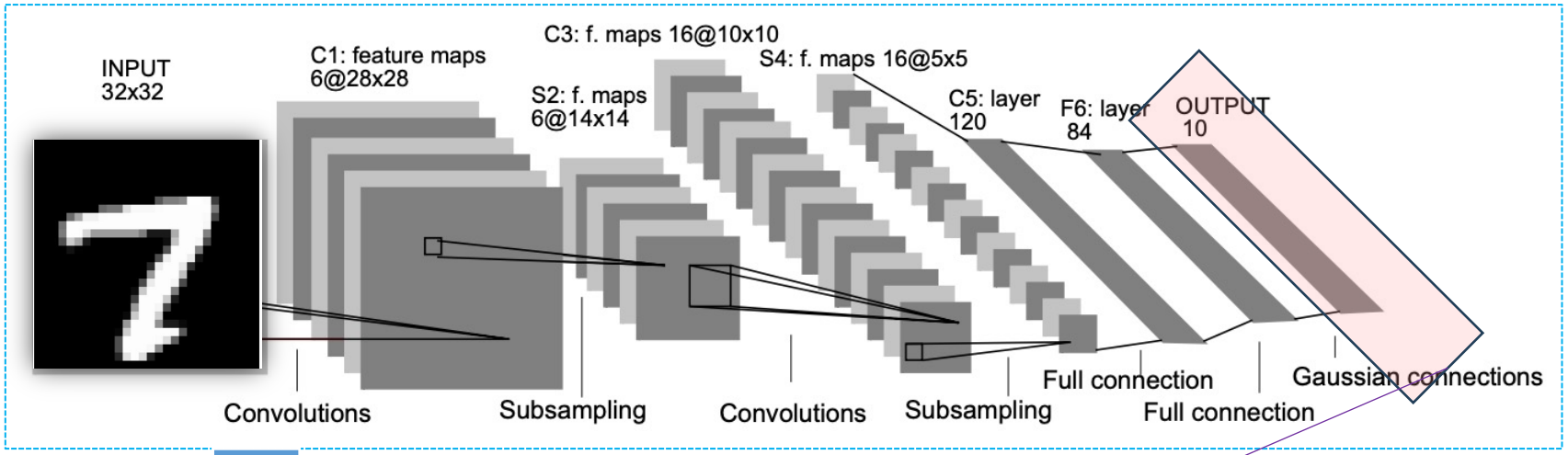
F6: Fully connected layer

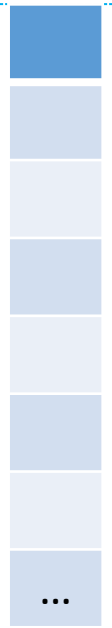
Number of filters (feature map): N/A
Filter size: N/A
Padding, N/A
Stride, N/A
Training parameters = (Weight + bias) = 10164

Why?



LetNet





84x1

Ouput: SoftMax output layer

Number of filters (feature map): N/A
Filter size: N/A
Padding, N/A
Stride, N/A
Training parameters = (Weight + bias) = 10164

Why?

10x1

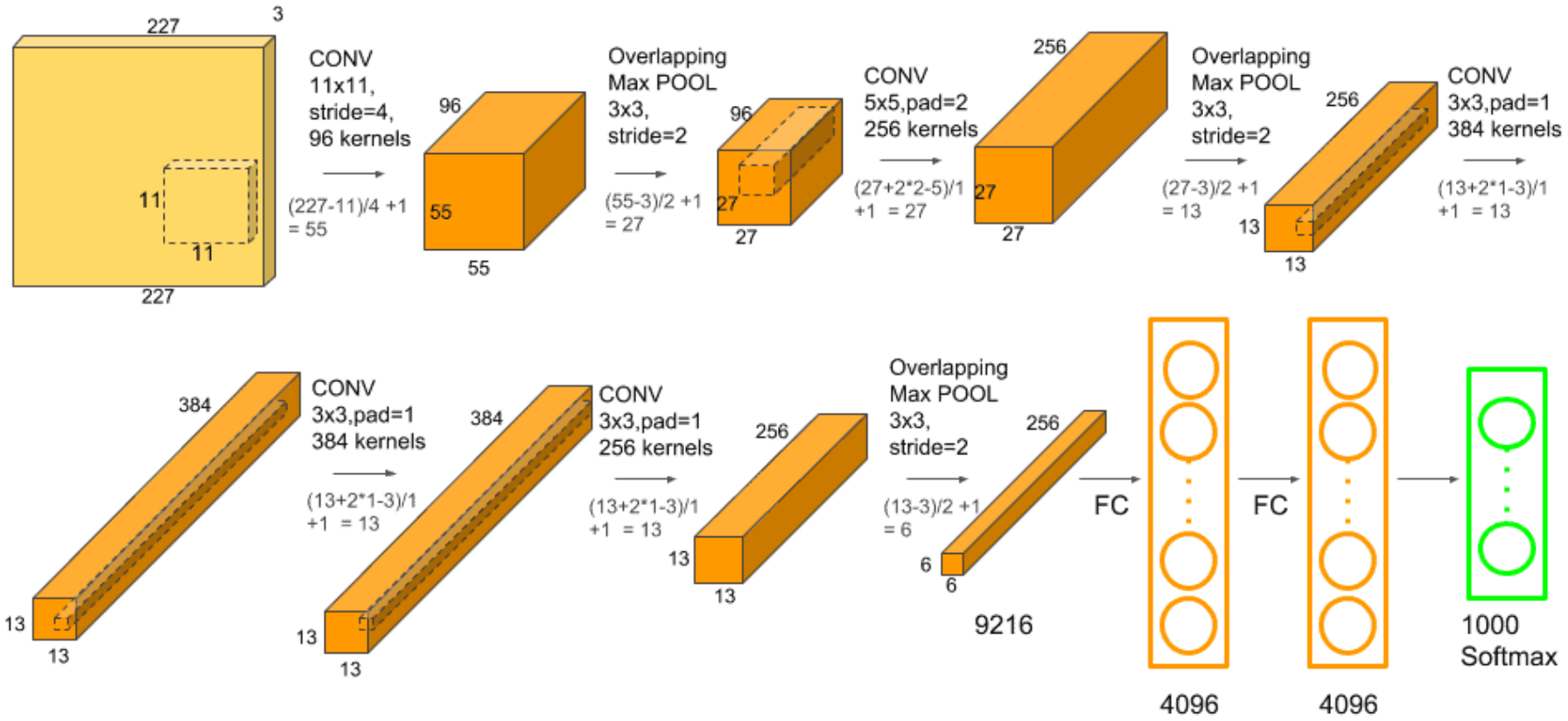
LetNet: Summary

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

AlexNet



AlexNet is the name of a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor at the University of Toronto.

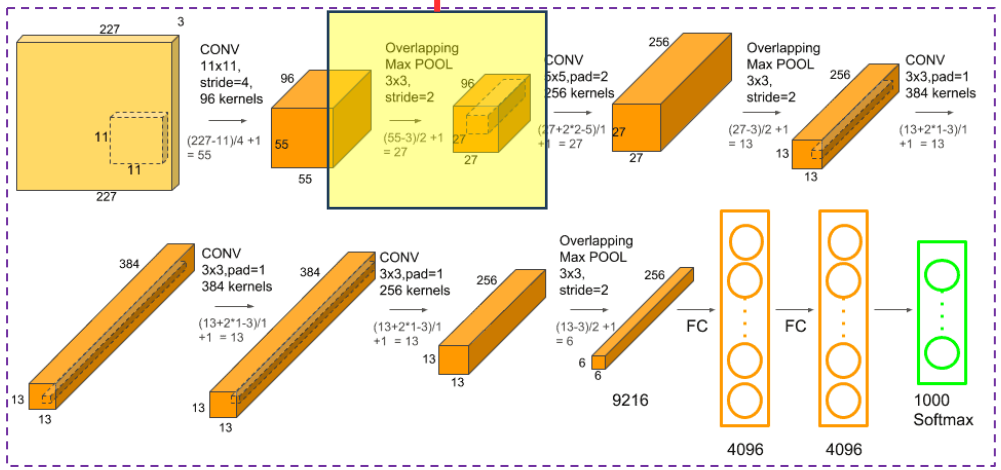
1. AlexNet consists of 5 Convolutional Layers and 3 Fully Connected Layers

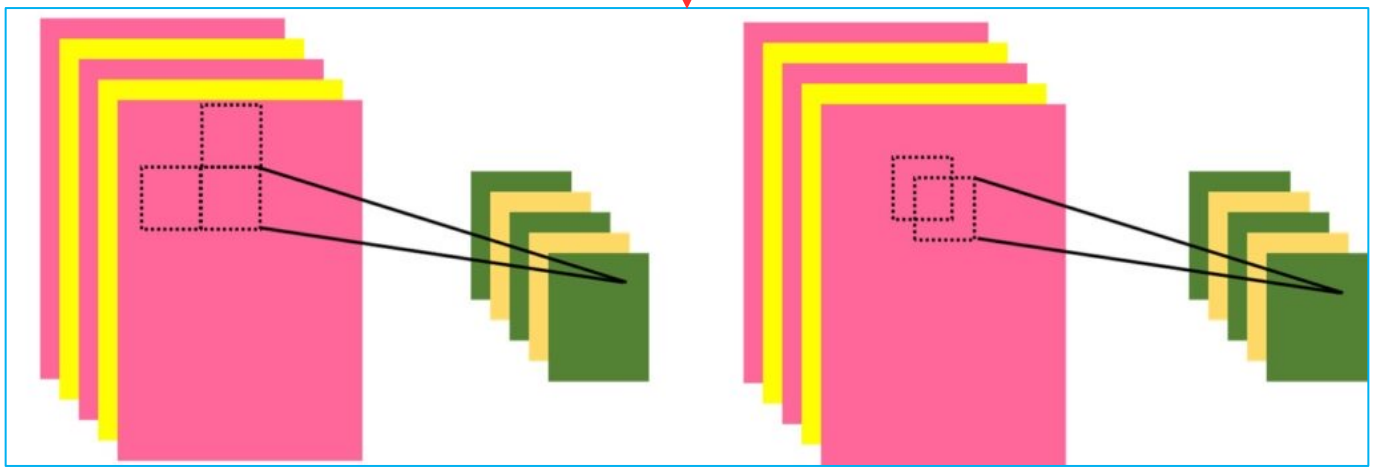
2. It has 60 million parameters and 650,000 neurons and took five to six days to train on two GTX 580 3GB GPUs (2012)

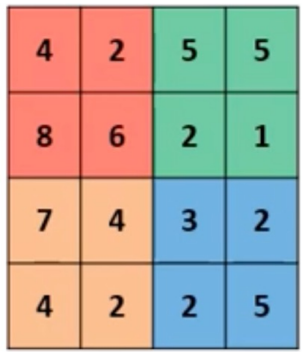
3. This was the first architecture that used GPU to boost the training performance

AlexNet

New: Overlapping Max Pooling



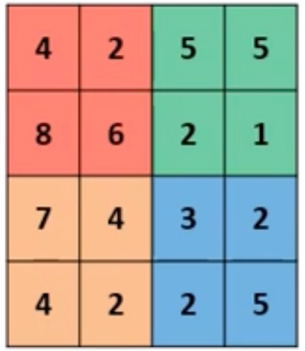




Max pooling

(2x2)
Stride 2

This overlapping nature of pooling helped reduce the top-1 error rate by 0.4% and top-5 error rate by 0.3% respectively when compared to using non-overlapping pooling windows of size 2×2 with a stride of 2 that would give same output dimensions.

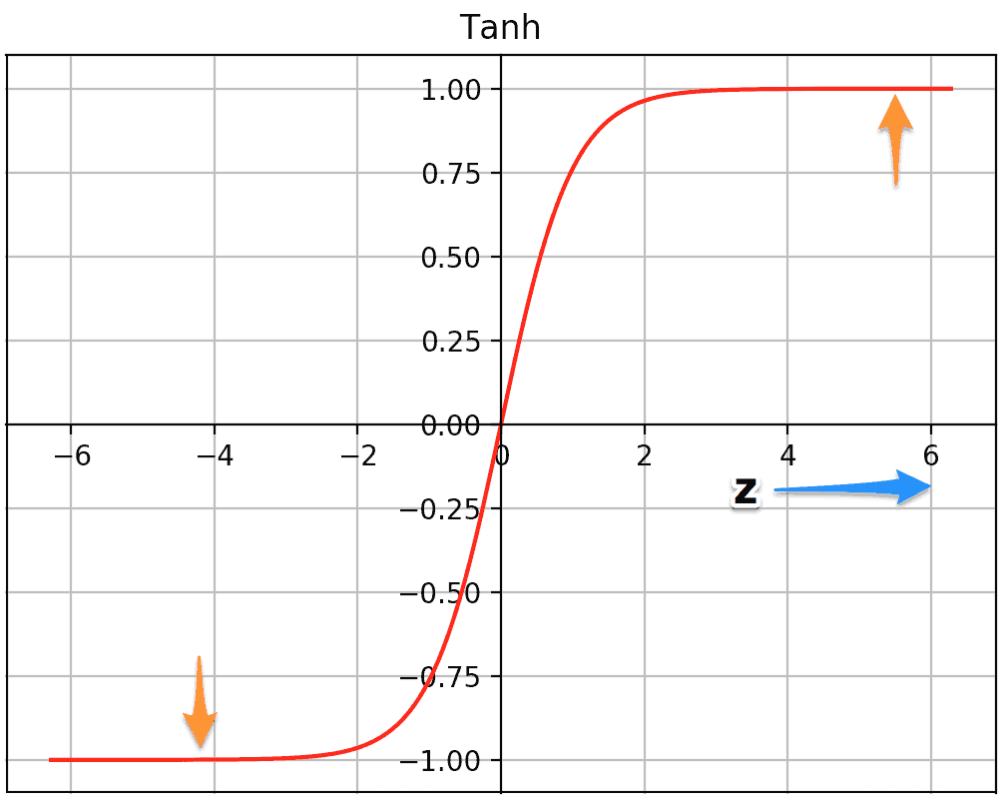


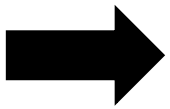
Overlap
max- pooling

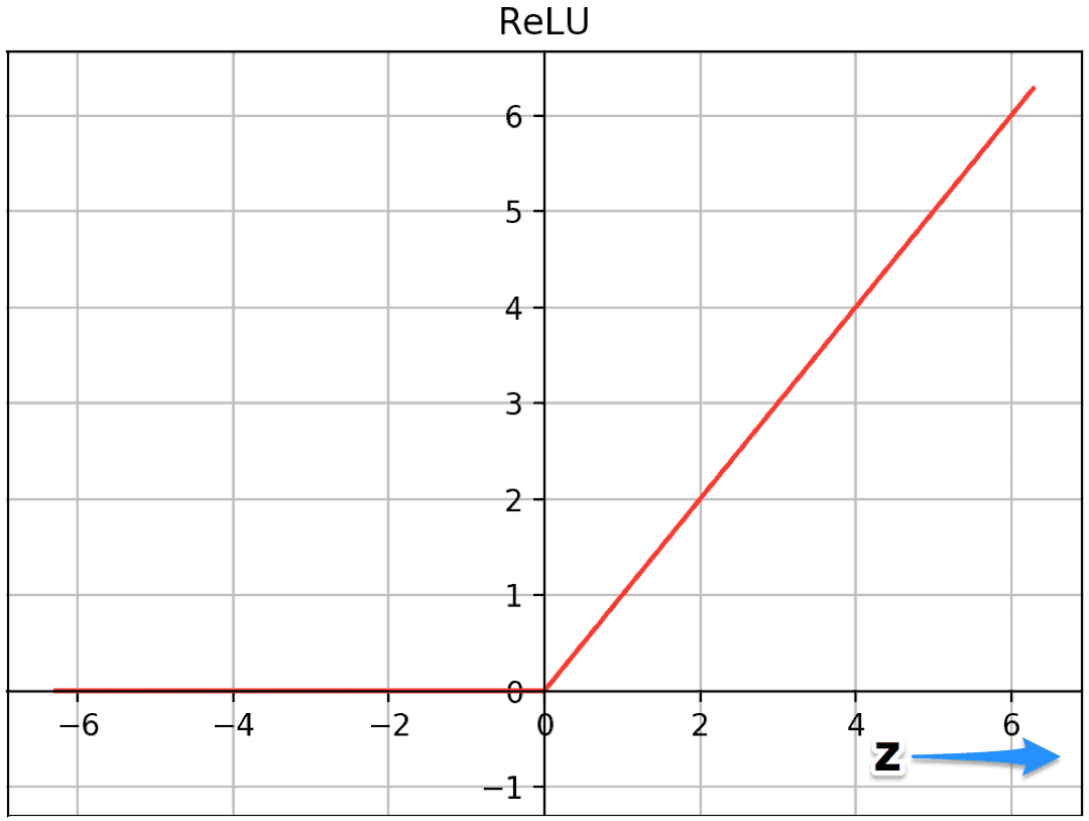
(2x2)
Stride 1

AlexNet

New: Activation Function





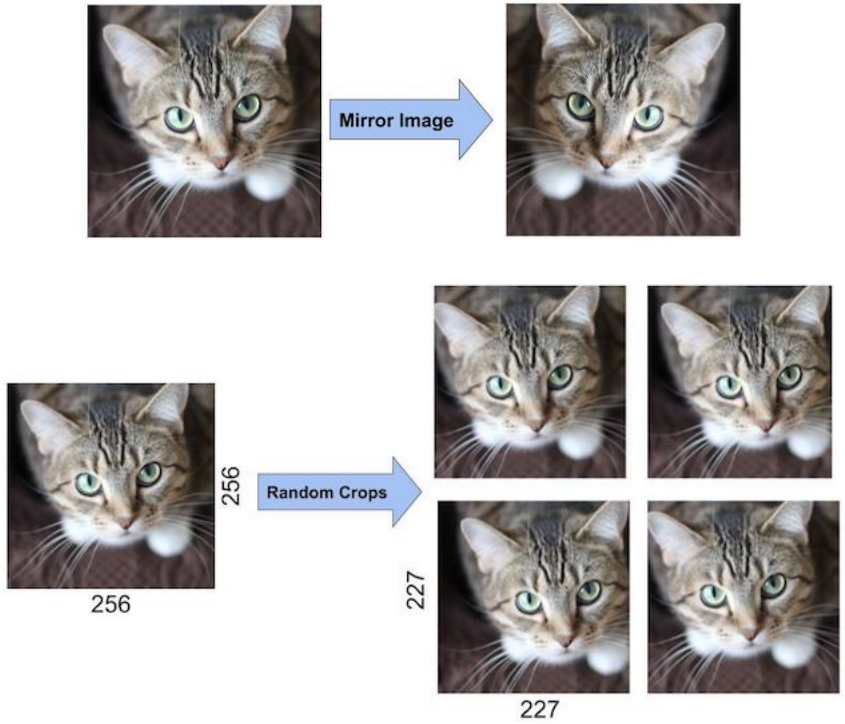


Using ReLU nonlinearity, deep CNNs could be trained much faster than using the saturating activation functions like tanh or sigmoid

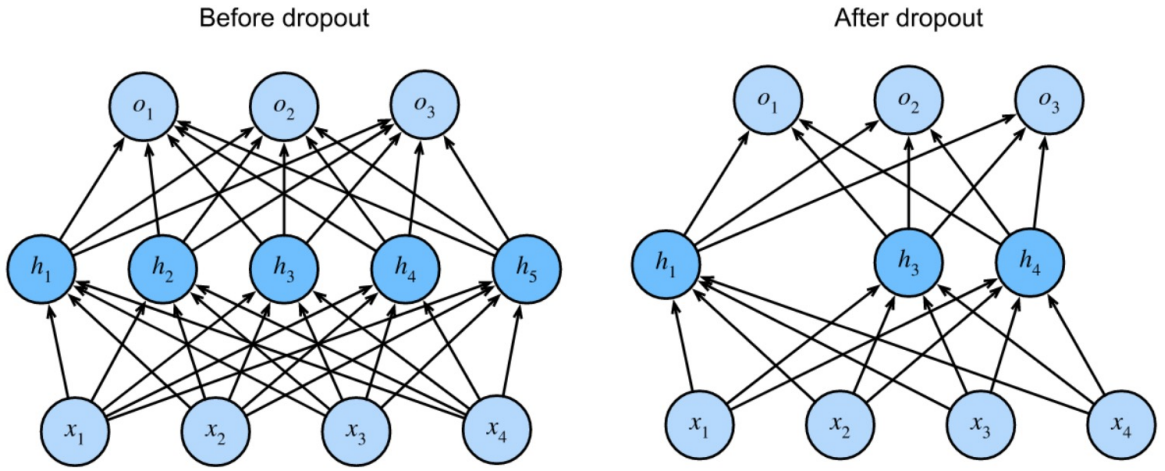
AlexNet

New: Reduce OverFitting

Data Augmentation



Drop Out



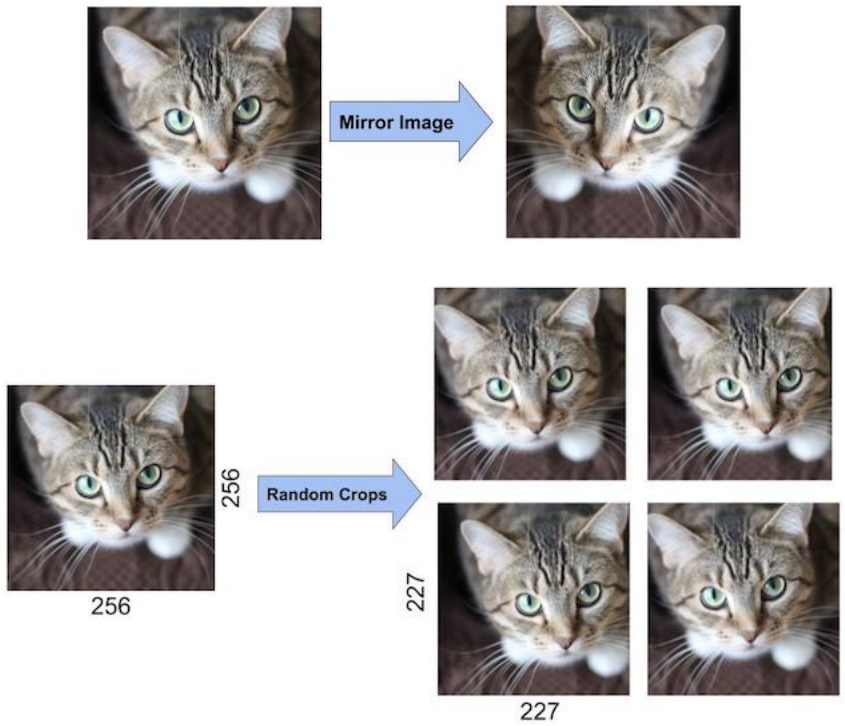
In dropout, a neuron is dropped from the network with a probability of 0.5.

Using ReLU nonlinearity, deep CNNs could be trained much faster than using the saturating activation functions like tanh or sigmoid

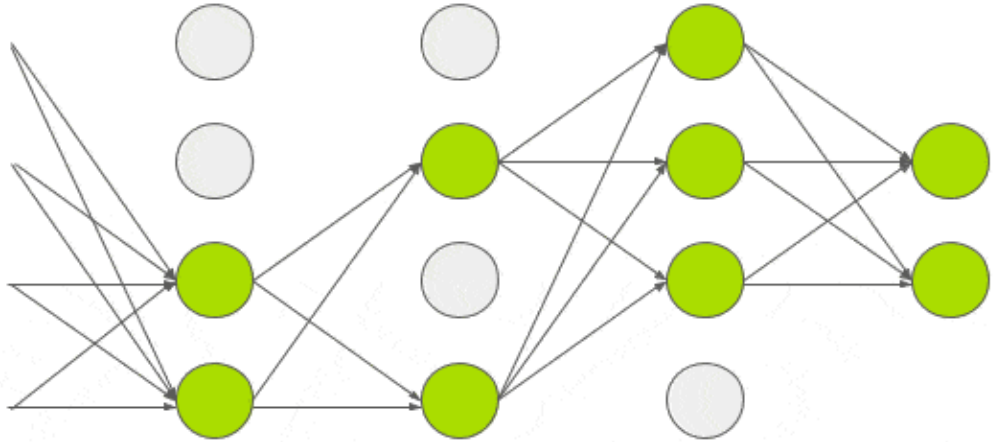
AlexNet

New: Reduce OverFitting

Data Augmentation



Drop Out



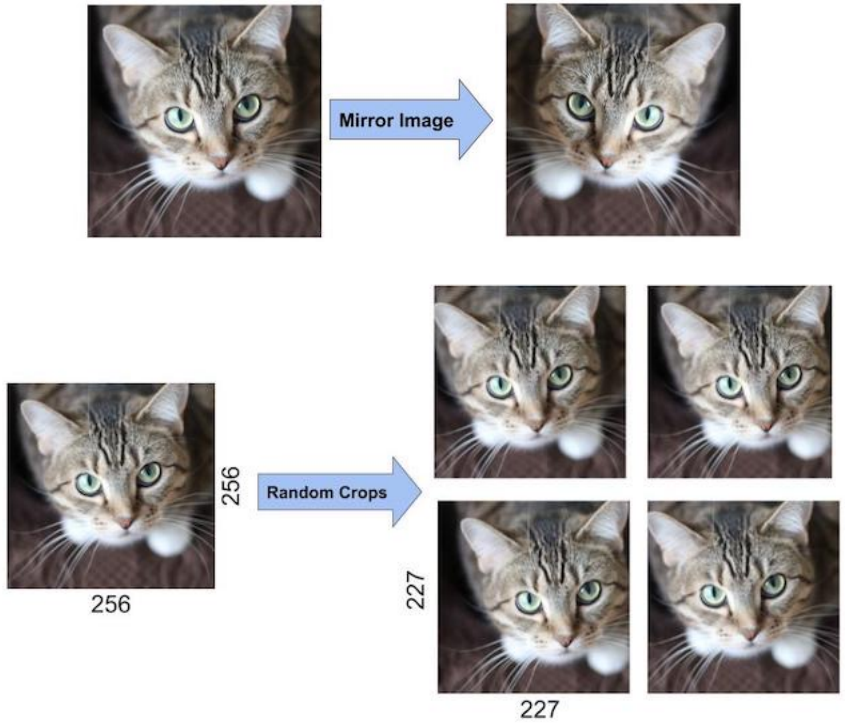
In dropout, a neuron is dropped from the network with a probability of 0.5.

Using ReLU nonlinearity, deep CNNs could be trained much faster than using the saturating activation functions like tanh or sigmoid

AlexNet

New: Reduce OverFitting

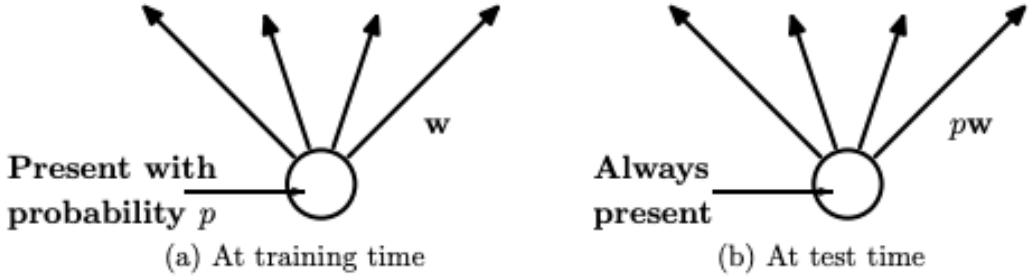
Data Augmentation



Drop Out

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$



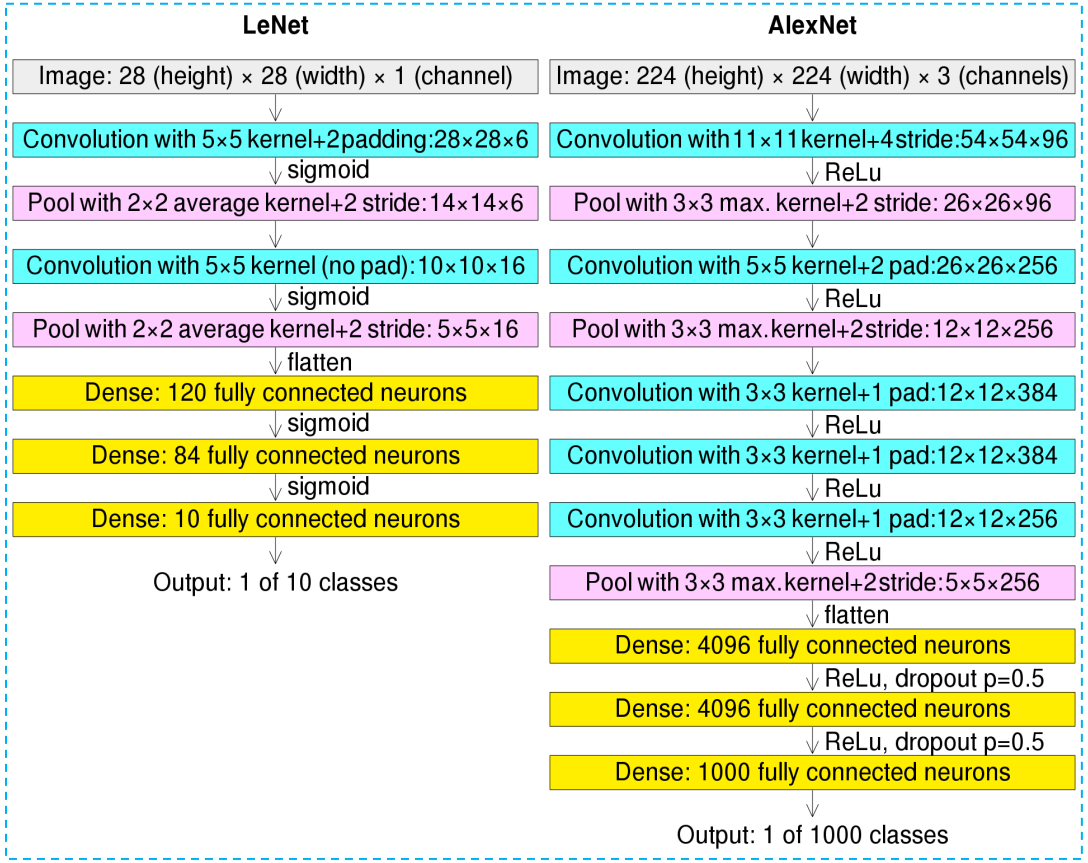
(a) At training time (b) At test time

if a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p during the prediction stage.

Using ReLU nonlinearity, deep CNNs could be trained much faster than using the saturating activation functions like tanh or sigmoid

AlexNet: Summary

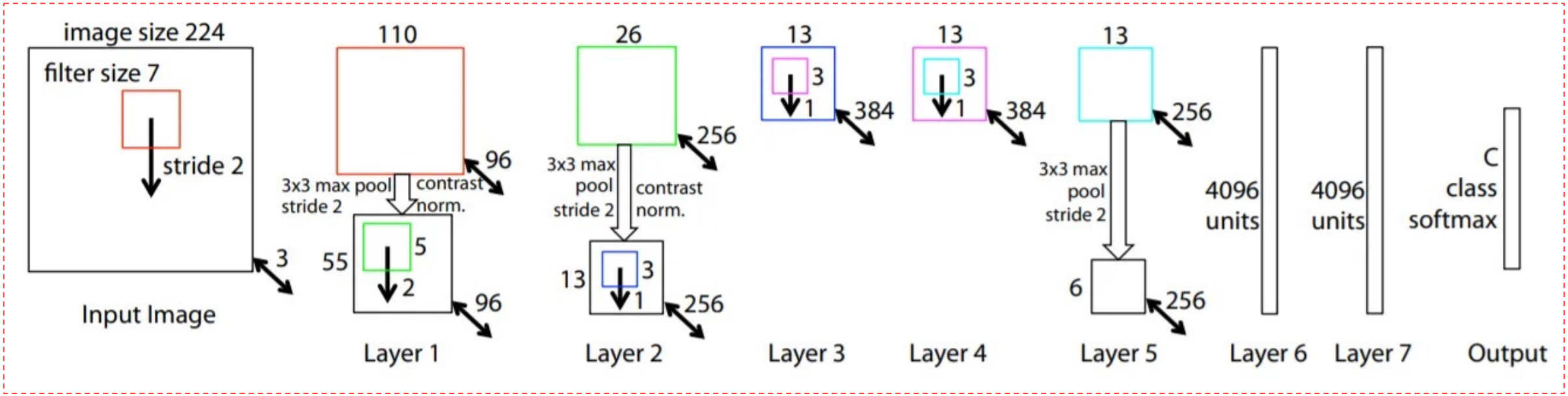
Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax



Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

ZFNet

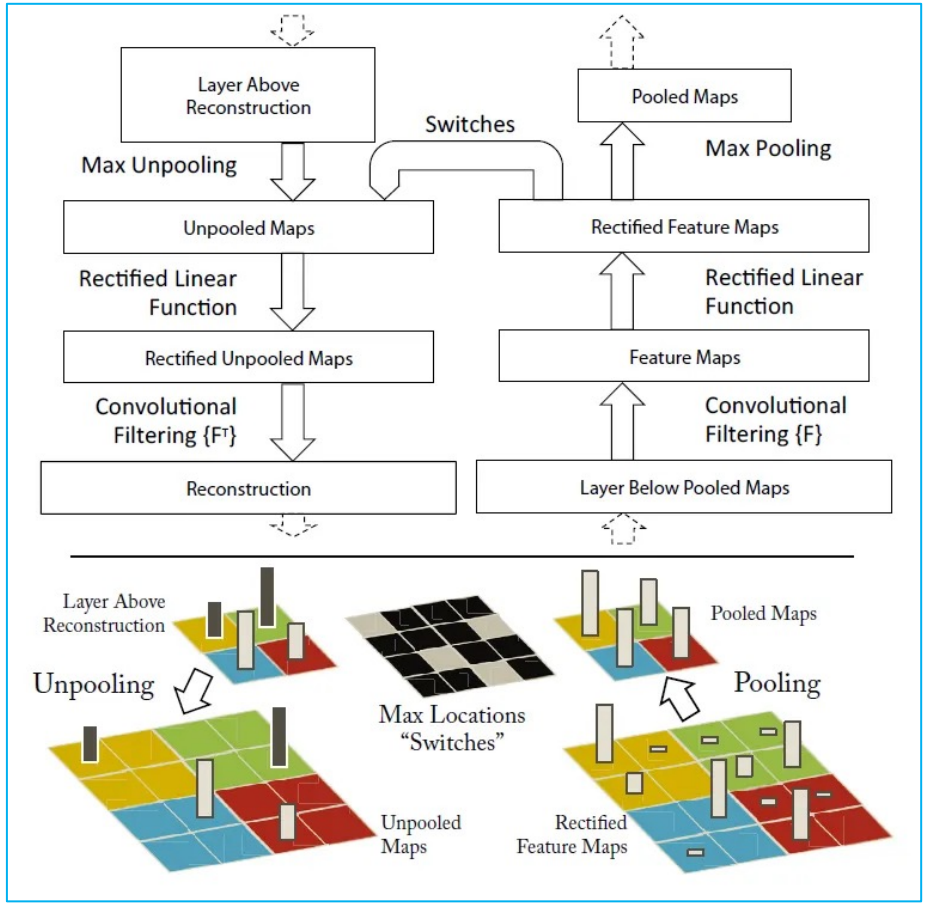


By visualizing the convolutional network, ZFNet become the Winner of ILSVLC 2013 in image classification by fine-tuning the AlexNet invented in 2012

ZFNet was invented by Rob Fergus and Matthew D. Zeiler

ZFNet

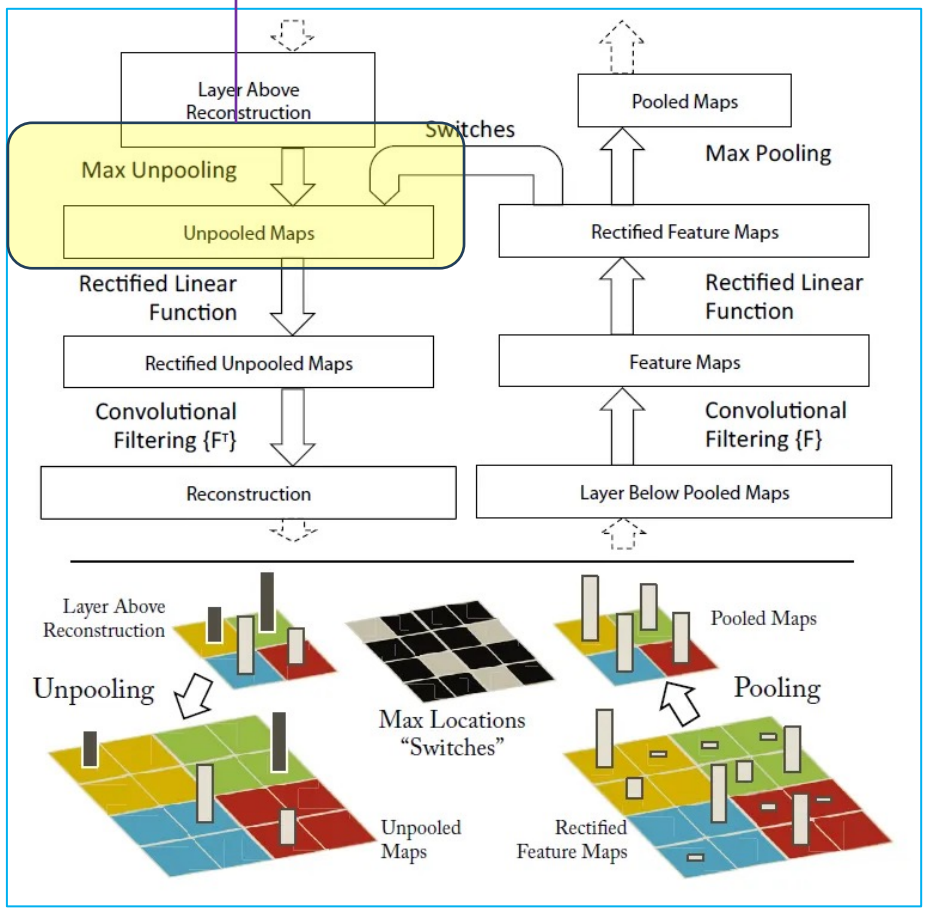
New: Deconvnet Techniques for Visualization

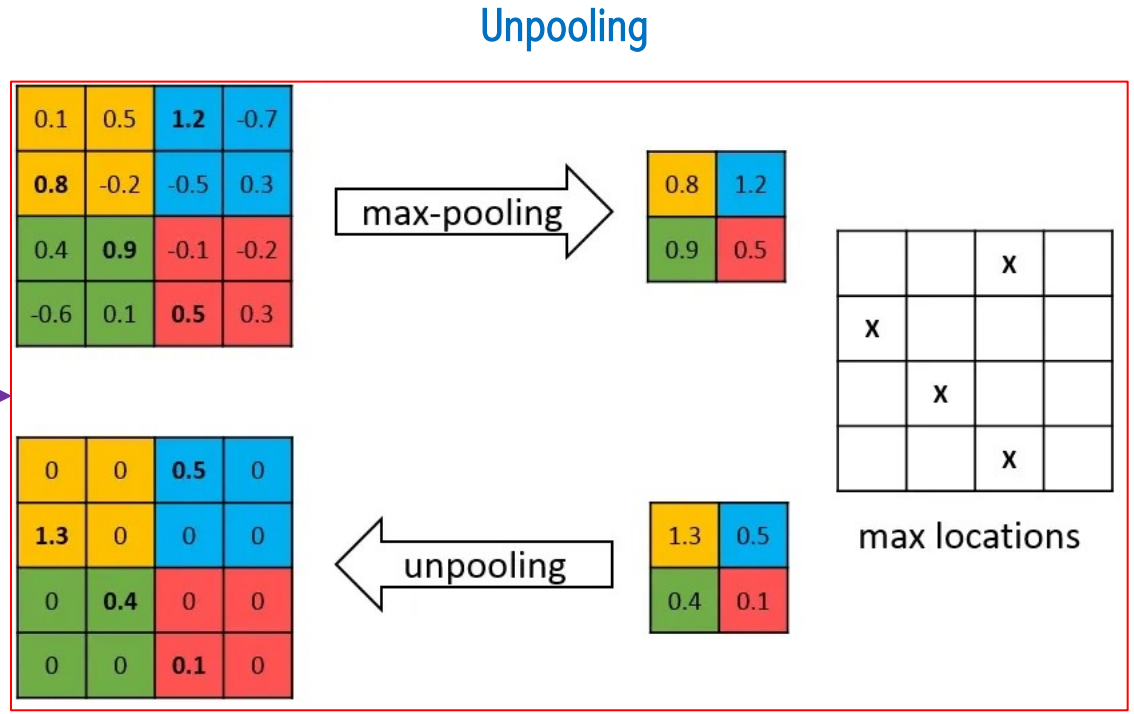


A standard step in deep learning framework is to have a series of **Conv > Rectification (Activation Function) > Pooling**. To visualize a deep layer feature, we need a set of decovnet techniques to reverse the above actions such that we can visualize the feature in pixel domain.

ZFNet

New: Deconvnet Techniques for Visualization

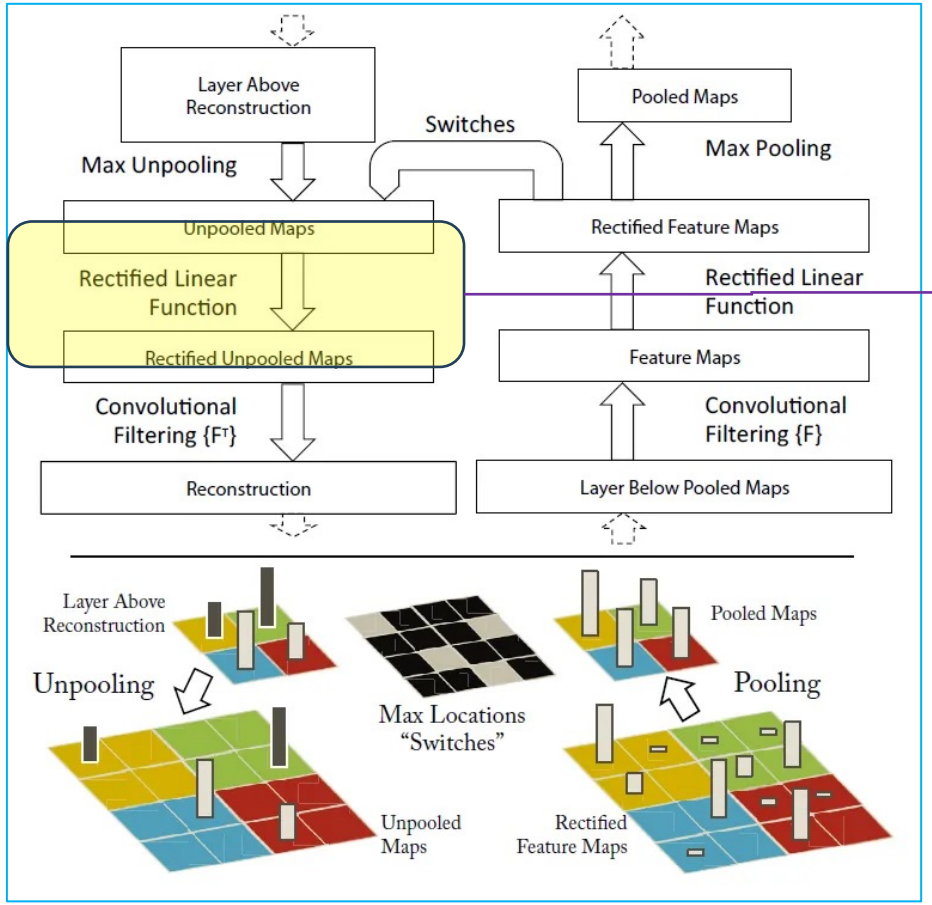




Max pooling operation is non-invertible, however we can obtain an approximate inverse by recording the locations of the maxima within each pooling region, as in the figure above.

ZFNet

New: Deconvnet Techniques for Visualization

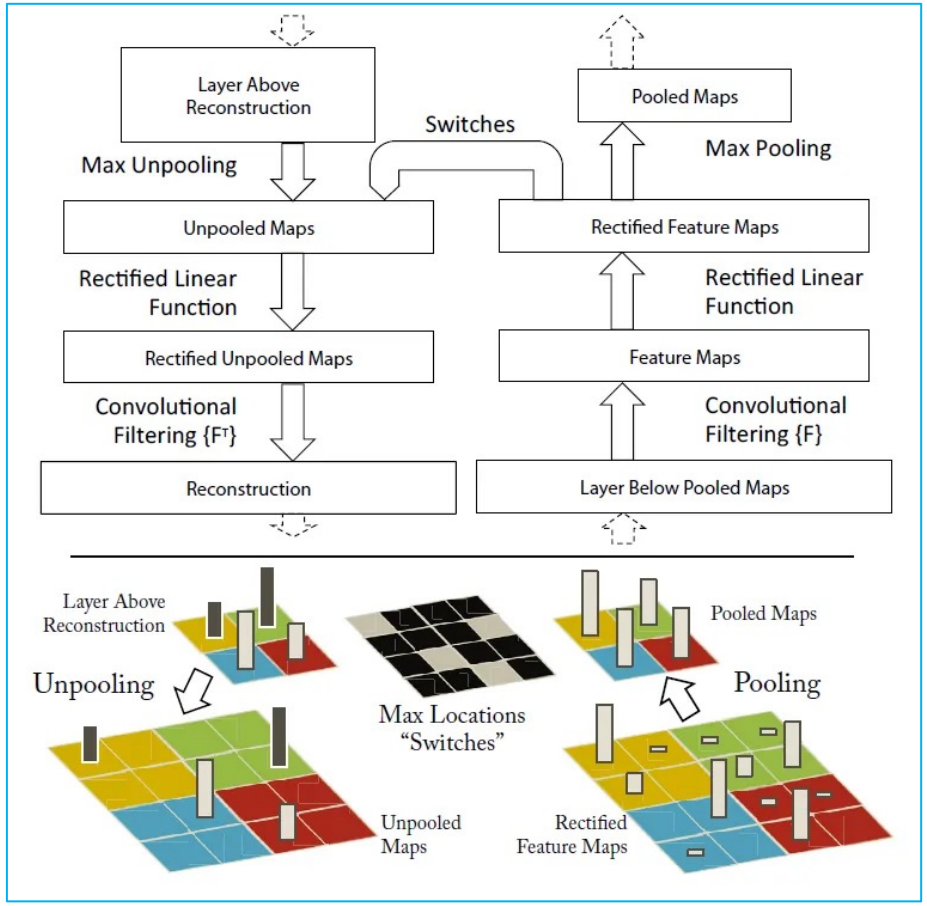


Rectification (Activation Function)

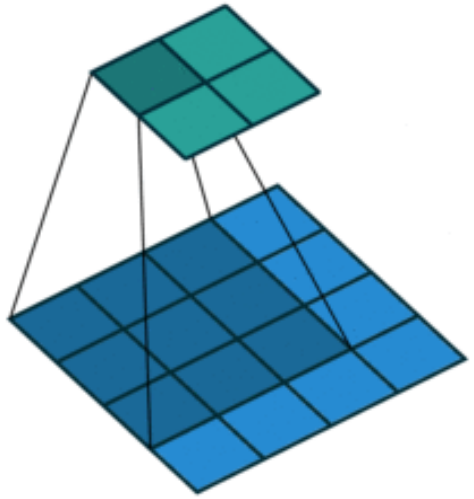
Since ReLU is used as the activation function, and ReLU is to keep all values positive while make negative values become zero. In the reverse operation, we just need to perform ReLU again.

ZFNet

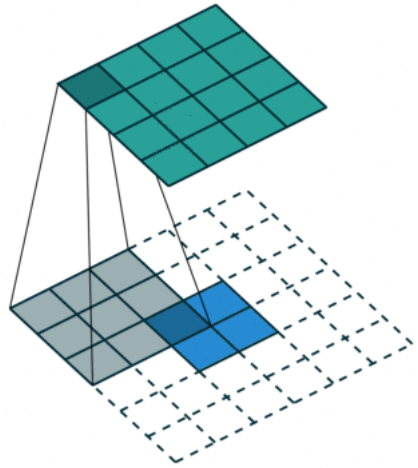
New: Deconvnet Techniques for Visualization



Convolution



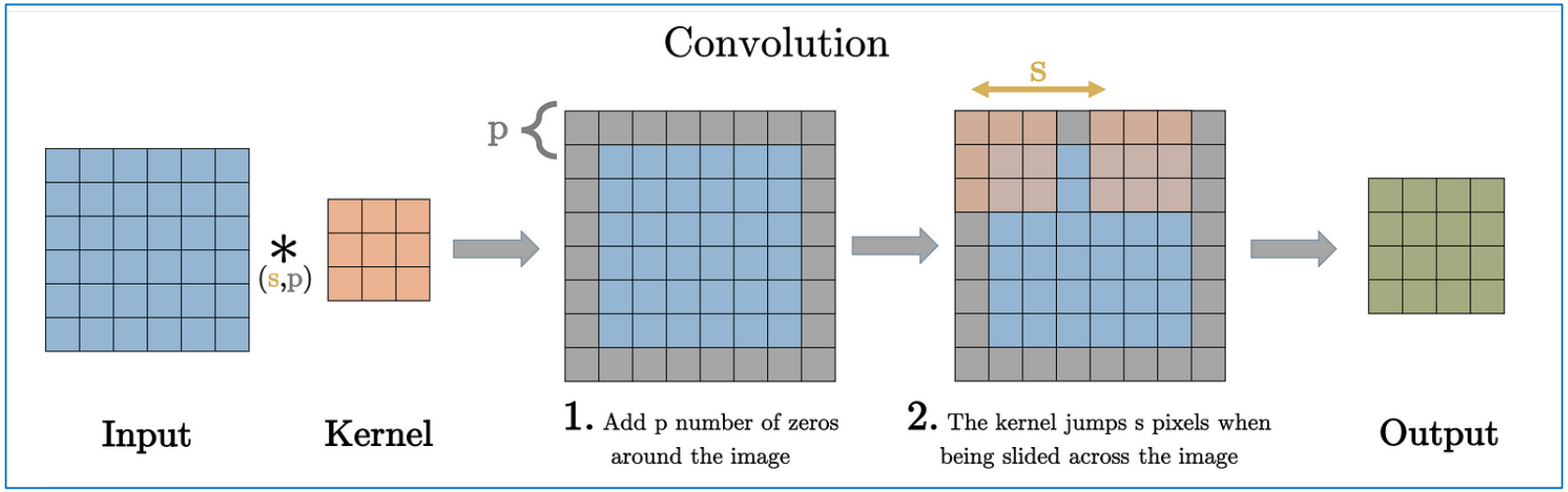
Convolution
(Blue is input, cyan is output)

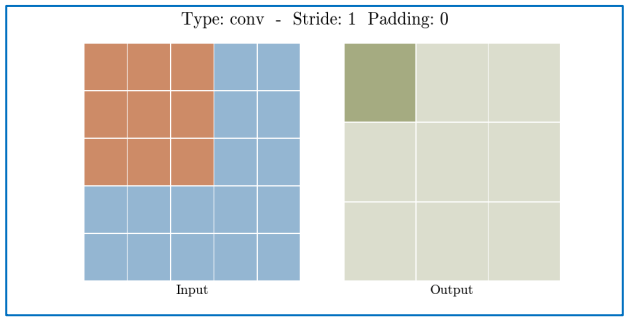


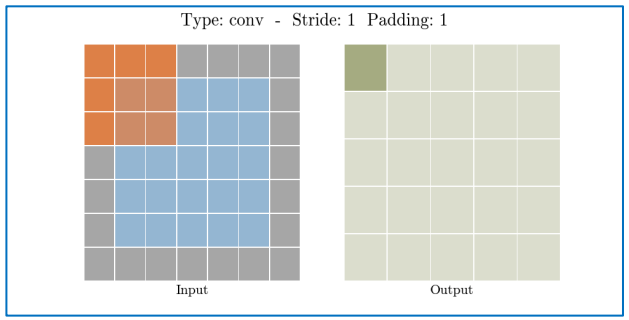
Deconvolution
(Blue is input, cyan is output)

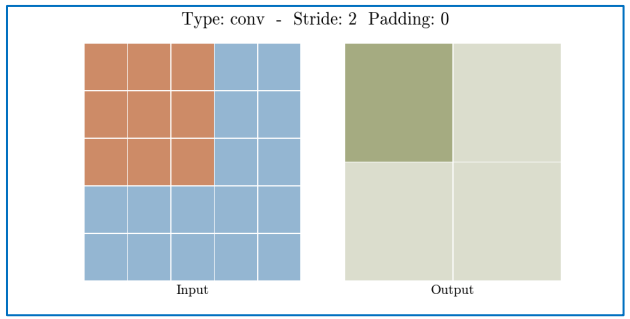
Convolution *adds up the information spread in various pixels to one pixel,* whereas ***DeConvolution*** *spreads the information present in one pixel to various pixels.*

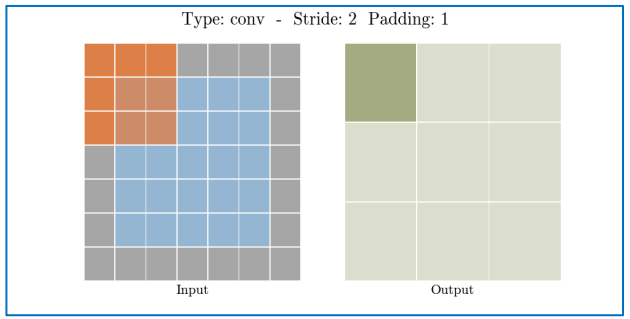
ZFNet


Convolution
p
s
*
(s,p)
Input
Kernel
1. Add p number of zeros around the image
2. The kernel jumps s pixels when being slided across the image
Output

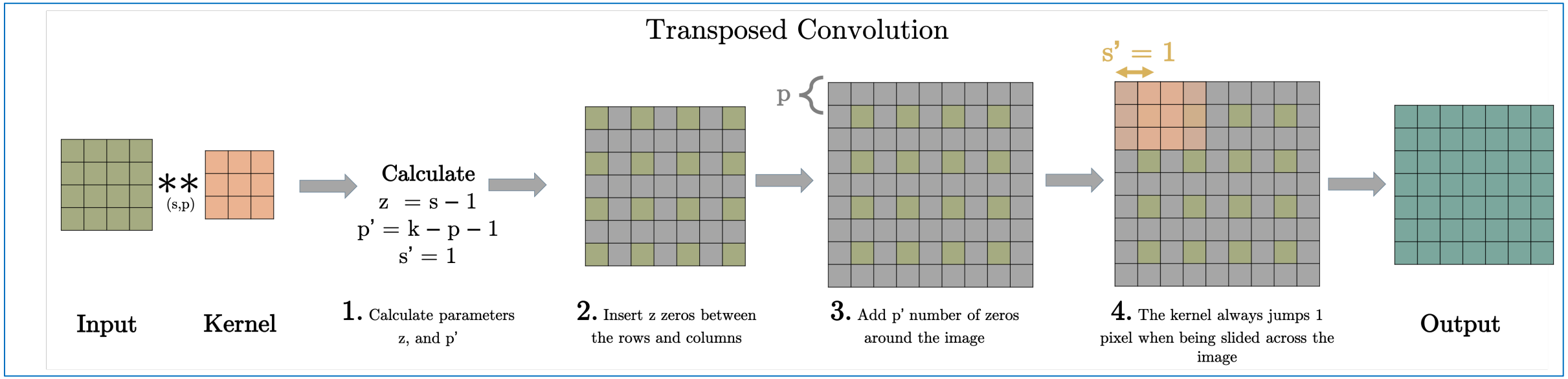

Type: conv - Stride: 1 Padding: 0
Input
Output

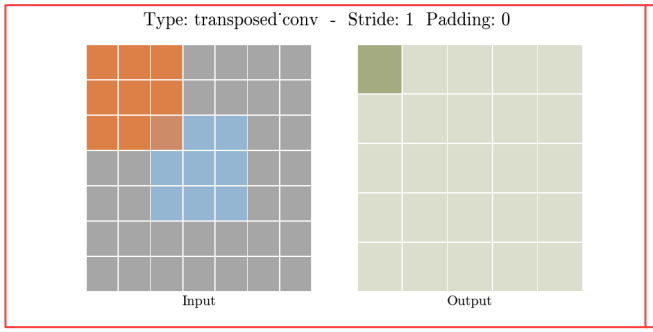

Type: conv - Stride: 1 Padding: 1
Input
Output

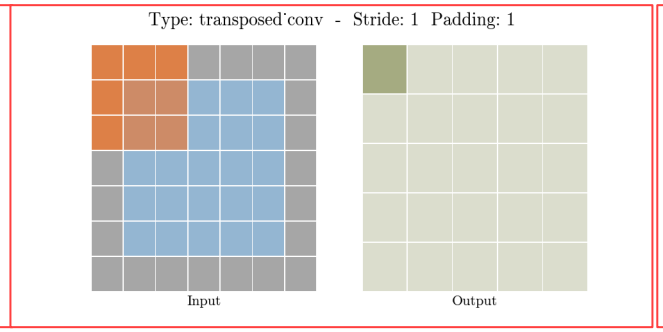

Type: conv - Stride: 2 Padding: 0
Input
Output

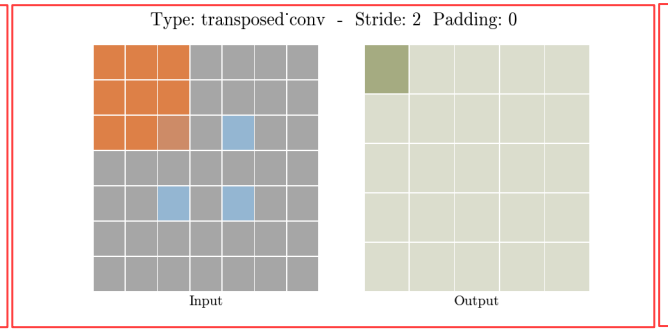

Type: conv - Stride: 2 Padding: 1
Input
Output

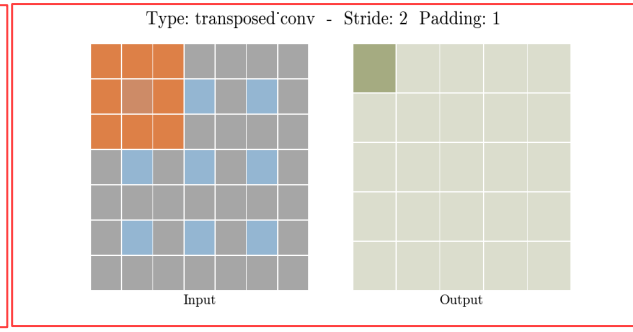
ZFNet


Transposed Convolution
(s,p)
Calculate
z = s − 1
p' = k − p − 1
s' = 1
p
s' = 1
Input
Kernel
1. Calculate parameters z, and p'
2. Insert z zeros between the rows and columns
3. Add p' number of zeros around the image
4. The kernel always jumps 1 pixel when being slided across the image
Output


Type: transposed conv - Stride: 1 Padding: 0
Input
Output

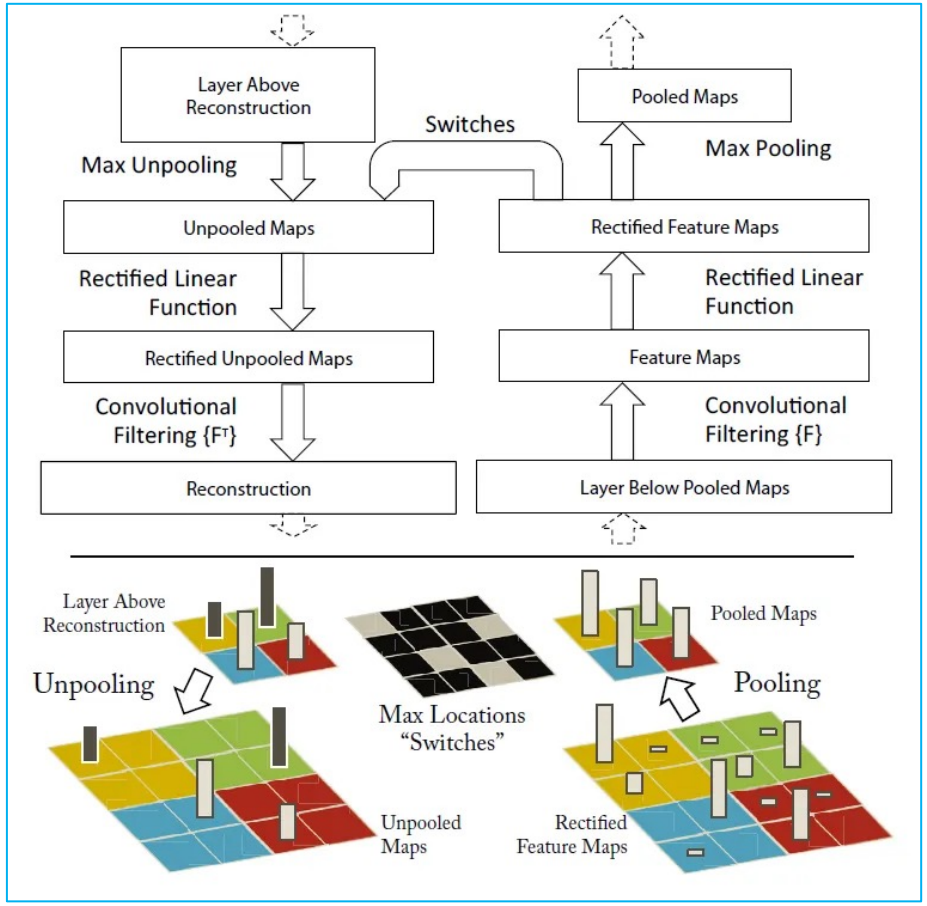

Type: transposed conv - Stride: 1 Padding: 1
Input
Output


Type: transposed conv - Stride: 2 Padding: 0
Input
Output

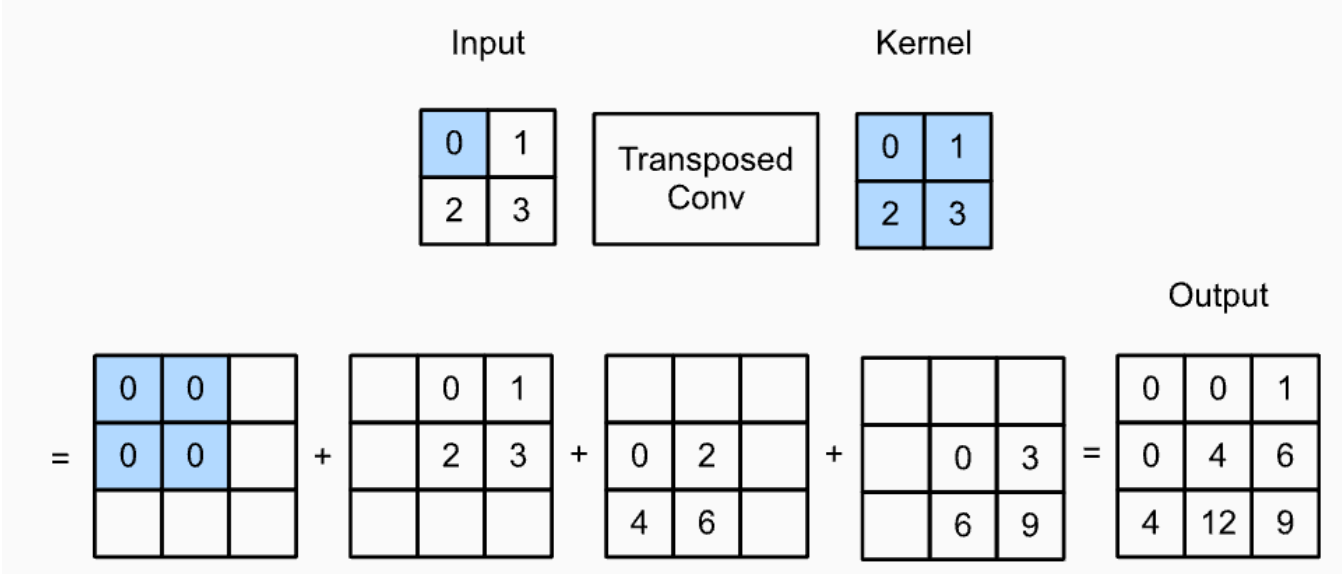

Type: transposed conv - Stride: 2 Padding: 1
Input
Output

ZFNet

New: Deconvnet Techniques for Visualization



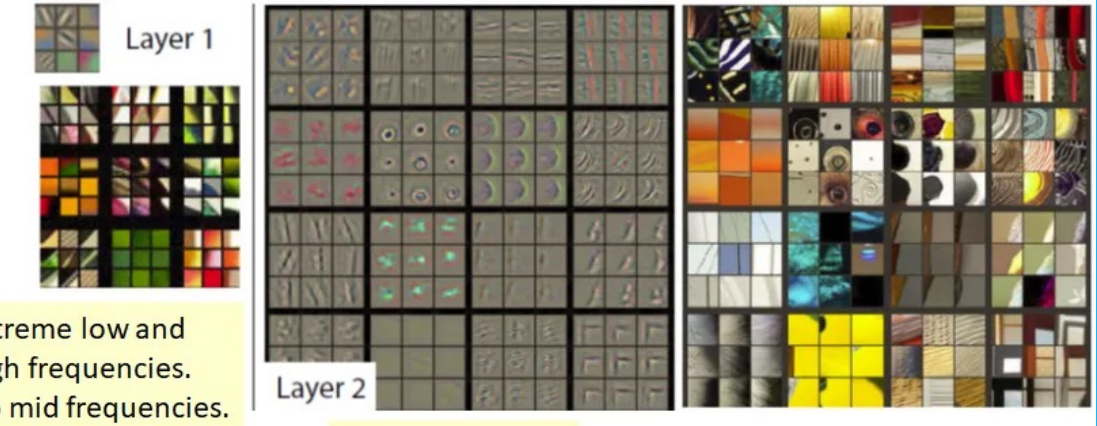
Deconvolution



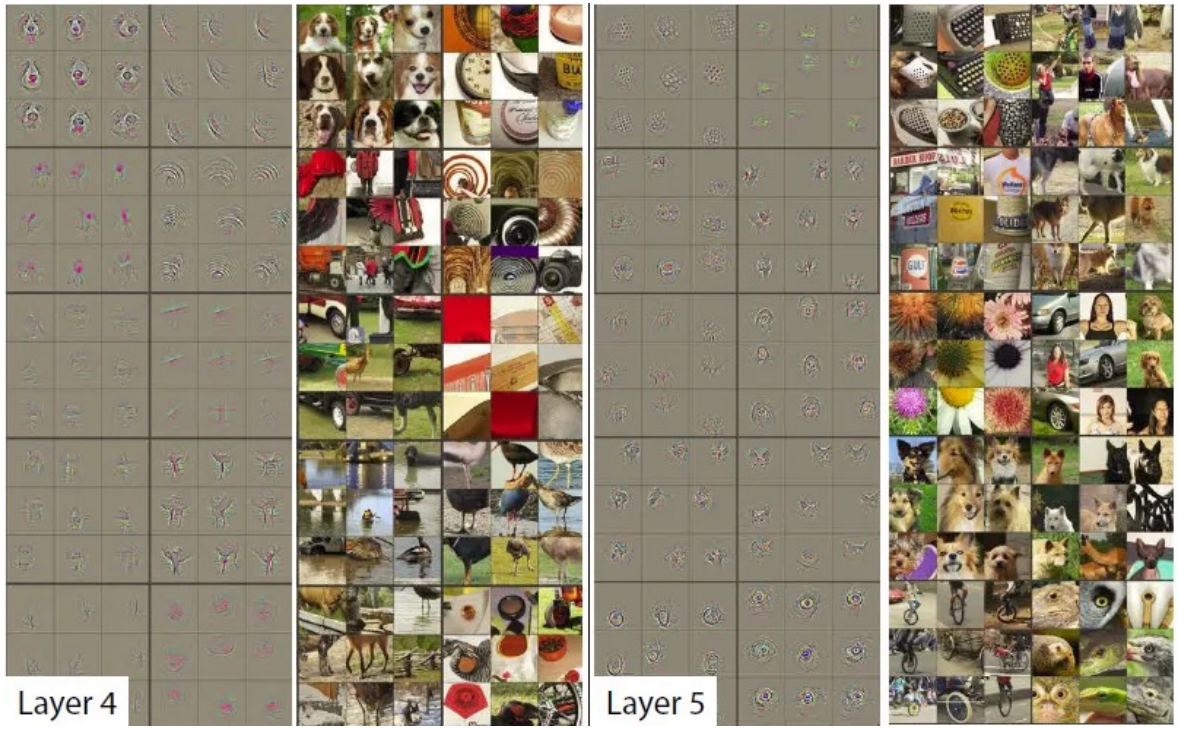
Convolution *adds up the information spread in various pixels to one pixel,* whereas ***DeConvolution*** *spreads the information present in one pixel to various pixels.*

ZFNet

New: Visualization for Each Layer

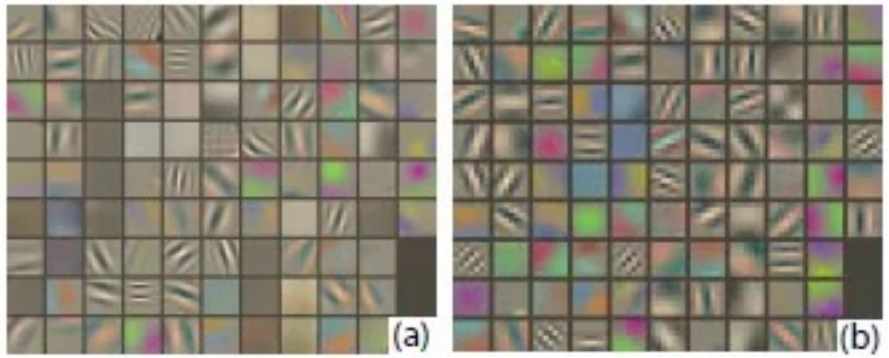


Layer 3 starts to learn some general patterns



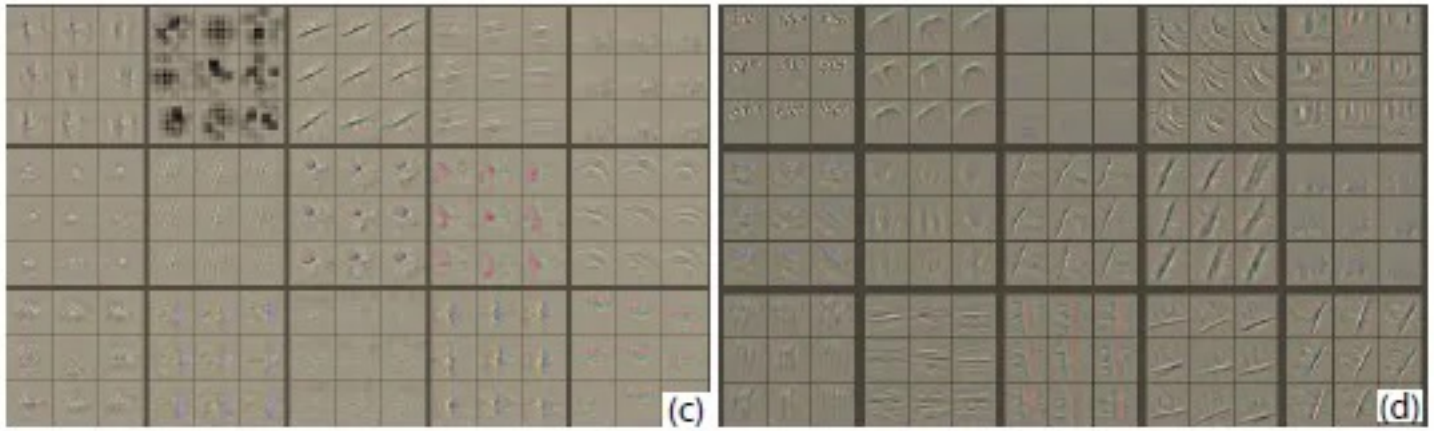
Layer 4 shows significant variation, and is more class-specific, such as dogs' faces and birds' legs.
Layer 5 shows entire objects with significant pose variation, such as keyboards and dogs.

ZFNet



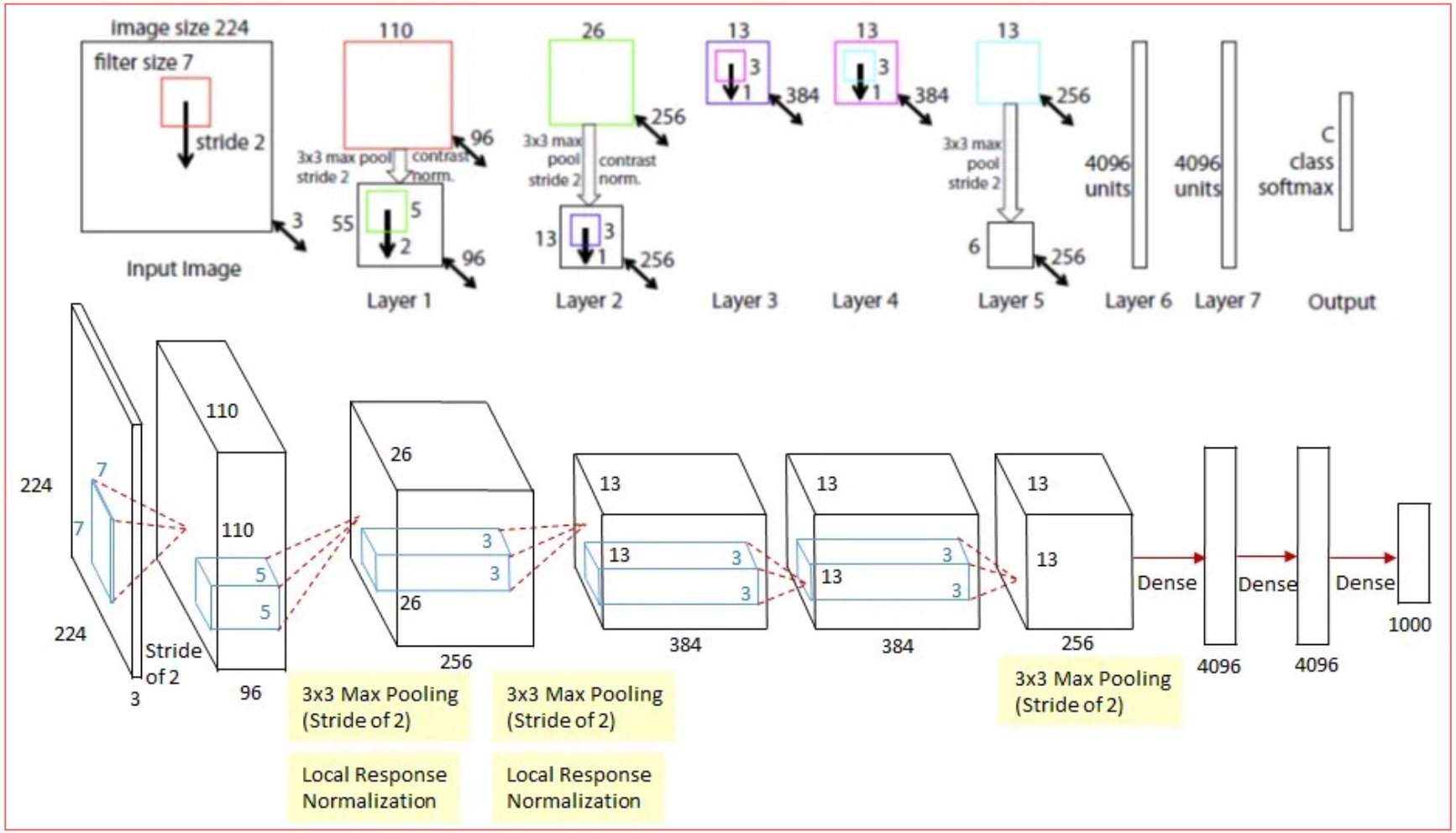
Layer 1: (a) More mid-frequencies in ZFNet, (b) Extremely low and high frequencies in AlexNet

Layer 2: (c) Aliasing artifacts in AlexNet and (d) much cleaner features in ZFNet



ZFNet

New: Modifications of AlexNet Based on Visualization Results



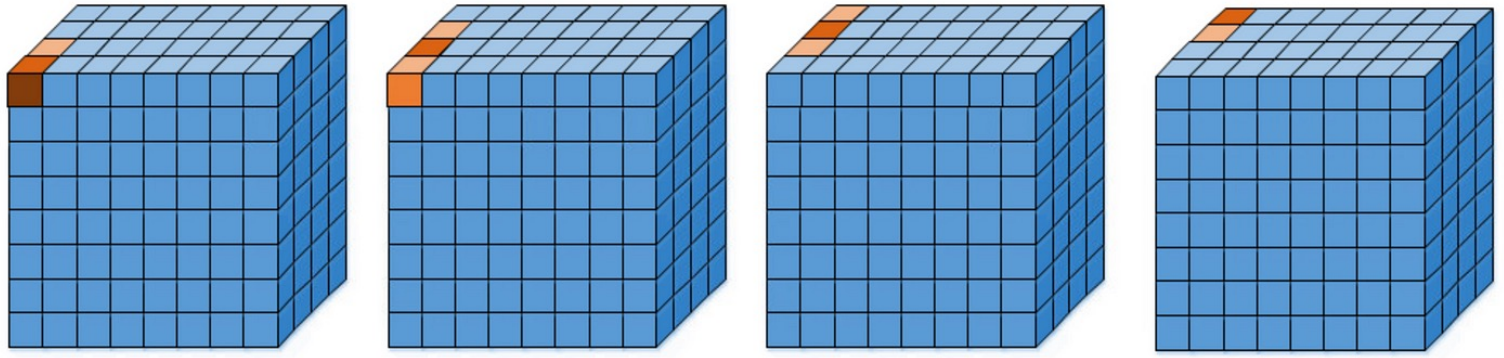
ZFNet is redrawn as the same style of AlexNet for the ease of comparison. To solve the two problems observed in layer 1 and layer 2, ZFNet makes two changes.

(i) Reduced the 1st layer filter size from 11x11 to 7x7.

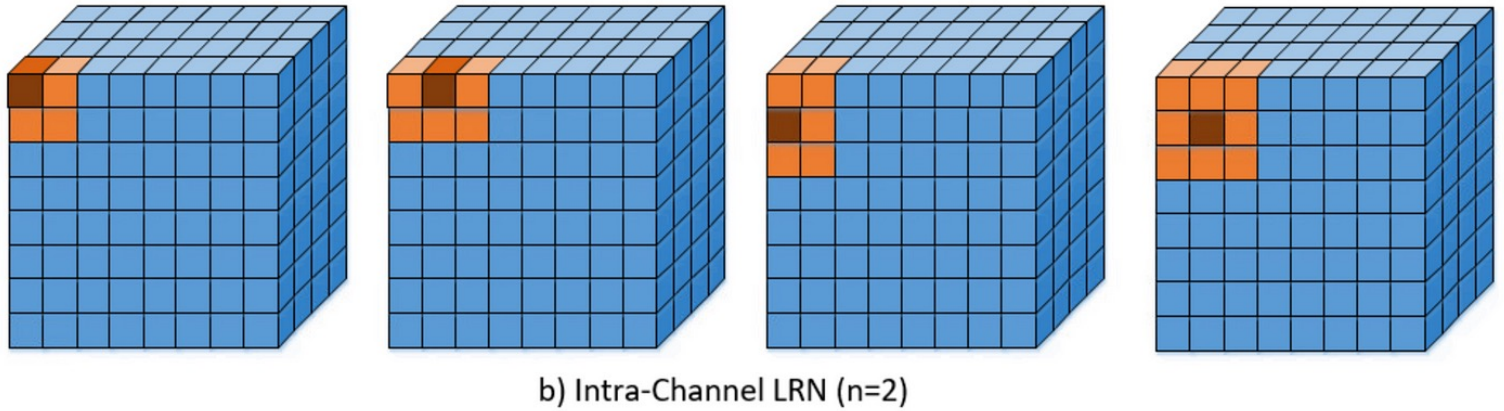
(ii) Made the 1st layer stride of the convolution 2, rather than 4.

ZFNet

Local Response Normalization



a) Inter-Channel LRN (n=2)



b) Intra-Channel LRN (n=2)

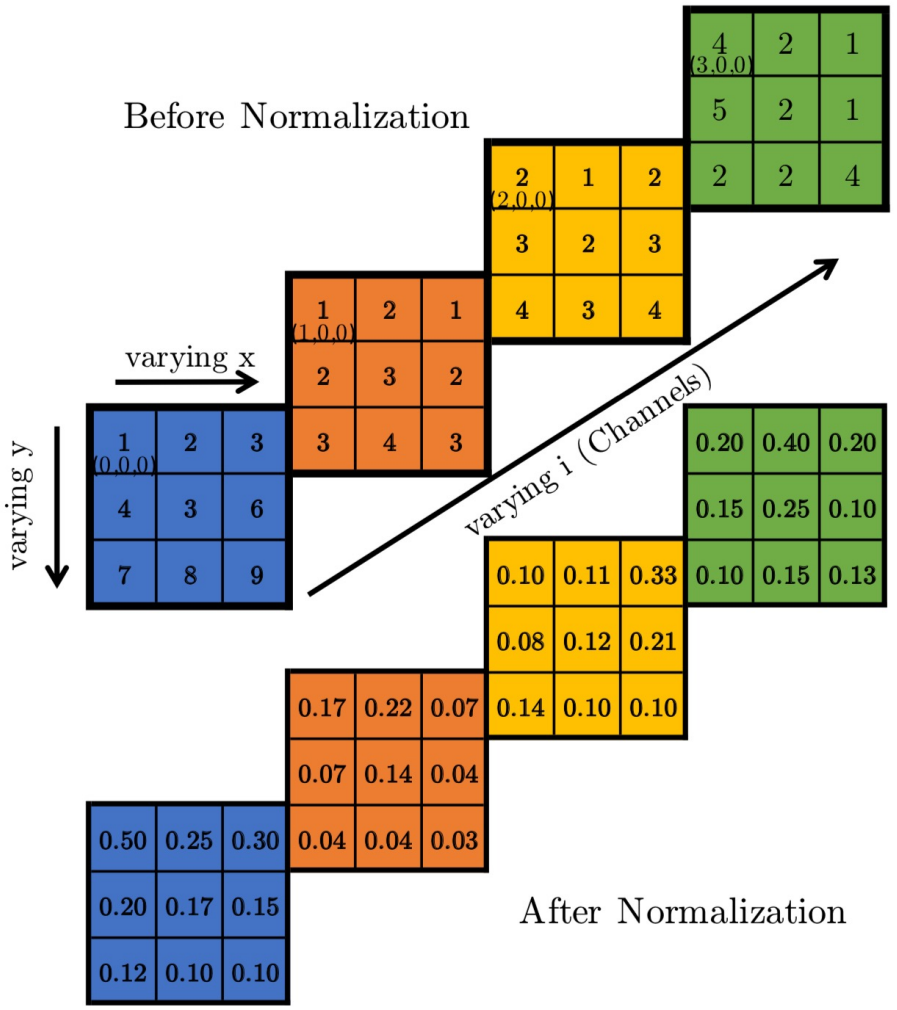
Normalization has become important for deep neural networks that compensate for the unbounded nature of certain activation functions such as ReLU, ELU, etc.

LRN is a non-trainable layer that square-normalizes the pixel values in a feature map within a local neighborhood

ZFNet

Inter-Channel LRN

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

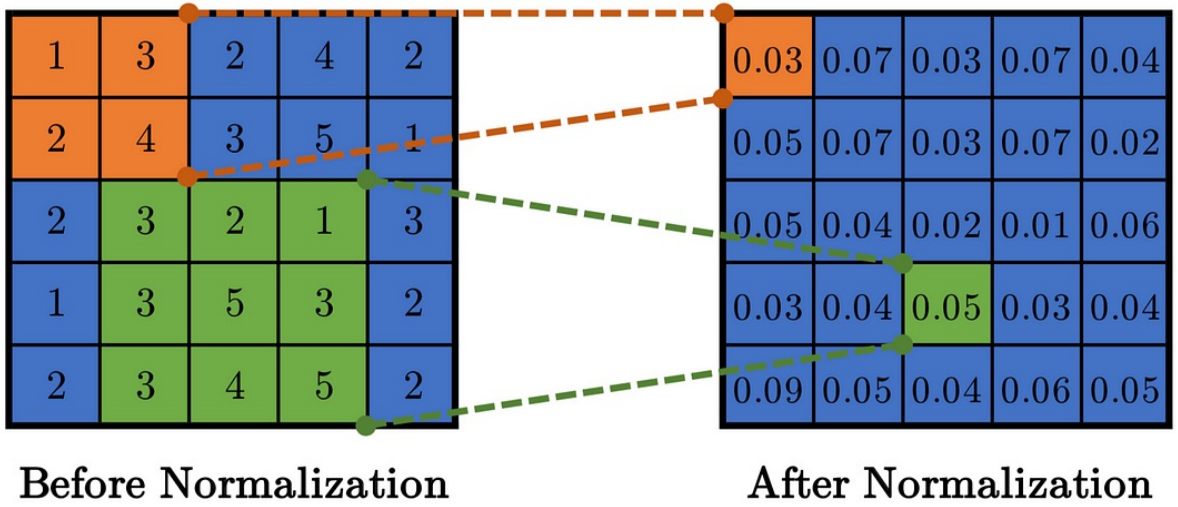


Lets take the hyper-parameters to be (*k,α, β, n)=(0,1,1,2).* The value of *n=2* means that while calculating the normalized value at position *(i,x,y)*, we consider the values at the same position for the previous and next filter i.e *(i-1, x, y)* and *(i+1, x, y)*. For *(i,x,y)=(0,0,0)* we have *value(i,x,y)=1*, *value(i-1,x,y)* doesn't exist and *value(i+,x,y)=1*. Hence *normalized_value(i,x,y) = 1/(¹²+¹²) = 0.5*

ZFNet

Intra-Channel LRN

$$b^k_{x,y} = a^k_{x,y} / \left(k + \alpha \sum_{i=max(0,x-n/2)}^{min(W,x+n/2)} \sum_{j=max(0,y-n/2)}^{min(H,y+n/2)} (a^k_{i,j})^2 \right)^{\beta}$$



In Intra-channel LRN, the neighborhood is extended within the same channel only as can be seen in the figure above. The formula is given by

ZFNet

Comparison of the two normalization techniques in DNNs				
Norm Type	**Trainable**	**Training Parameters**	**Fouses on**	Regularization
LRN	No	0	Lateral Inhibition	No
BN	Yes	2 (Scale and Shift)	Internal Covariate Shift	Yes

Batch Normalization (BN) is a **trainable layer** normally used for addressing the issues of *Internal Covariate Shift (ICF)*

Roses vs No-roses classification. The feature map plotted on the right have different distributions for two different batch sampled from the dataset

This is called the ***Covariate shift*** of the input neurons

ZFNet

Comparison of the two normalization techniques in DNNs				
Norm Type	**Trainable**	**Training Parameters**	**Fouses on**	Regularization
LRN	No	0	Lateral Inhibition	No
BN	Yes	2 (Scale and Shift)	Internal Covariate Shift	Yes

Batch Normalization (BN) is a **trainable layer** normally used for addressing the issues of ***Internal Covariate Shift (ICF)***

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

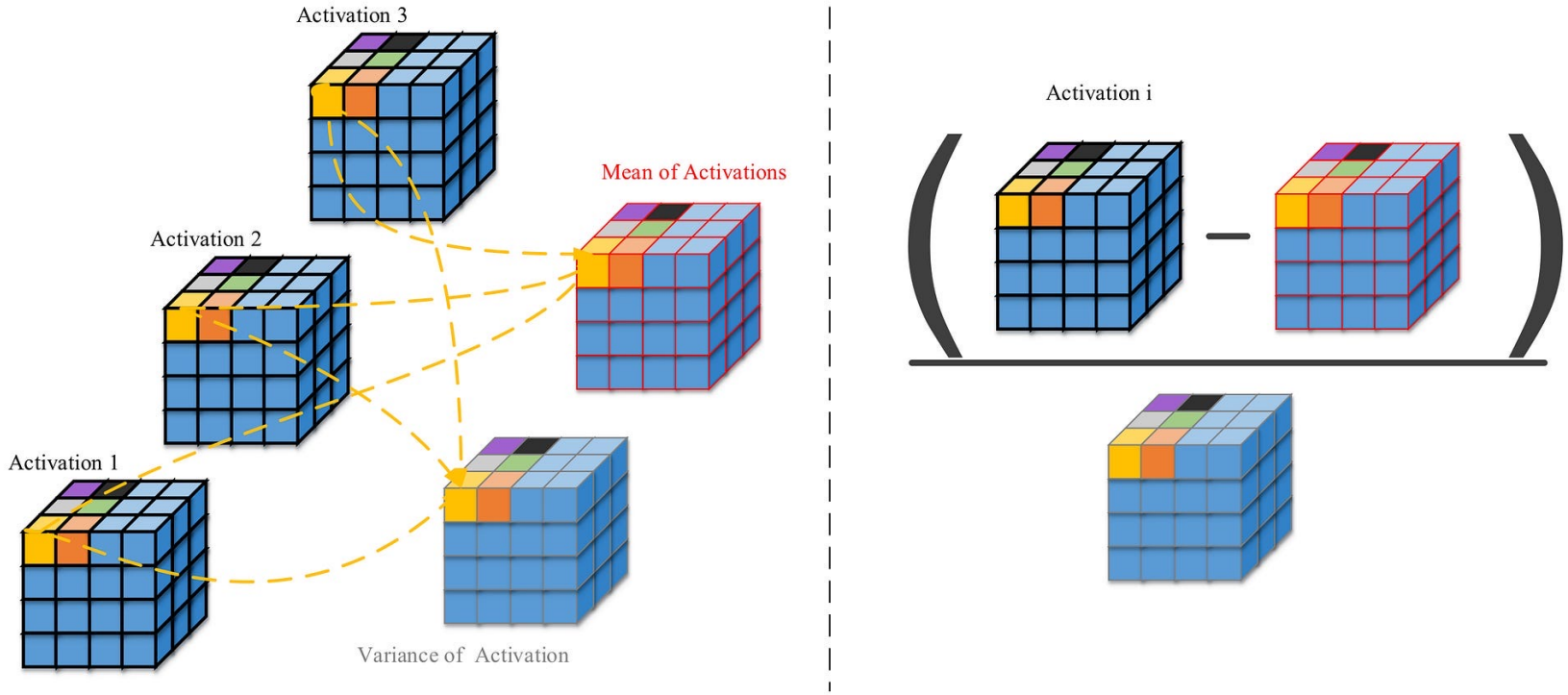
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

ZFNet

Comparison of the two normalization techniques in DNNs				
Norm Type	Trainable	Training Parameters	Fouses on	Regularization
LRN	No	0	Lateral Inhibition	No
BN	Yes	2 (Scale and Shift)	Internal Covariate Shift	Yes

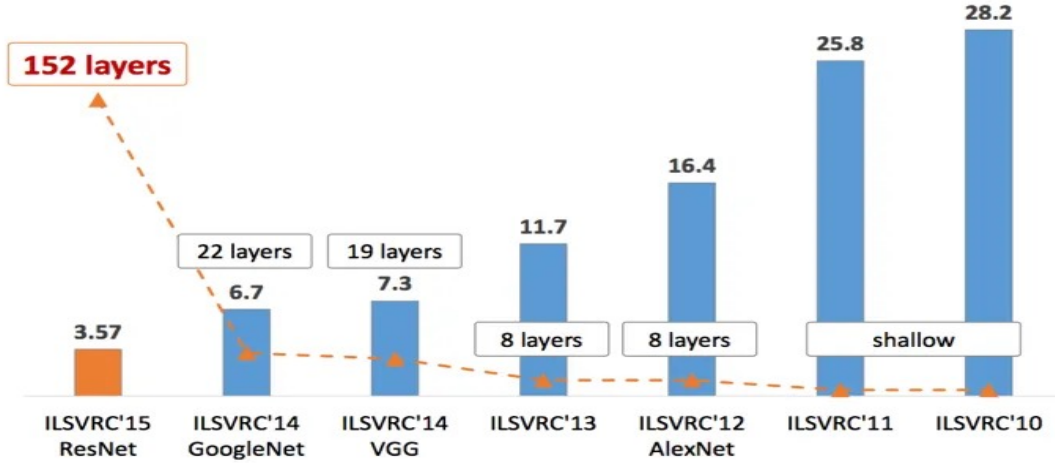


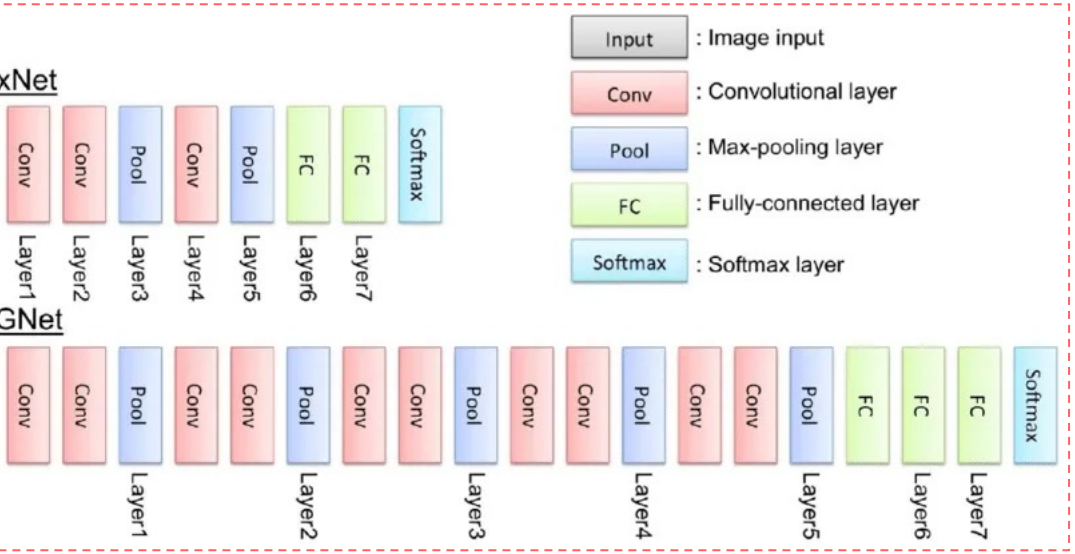
Batch Normalization (BN) is a **trainable layer** normally used for addressing the issues of *Internal Covariate Shift (ICF)*

Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

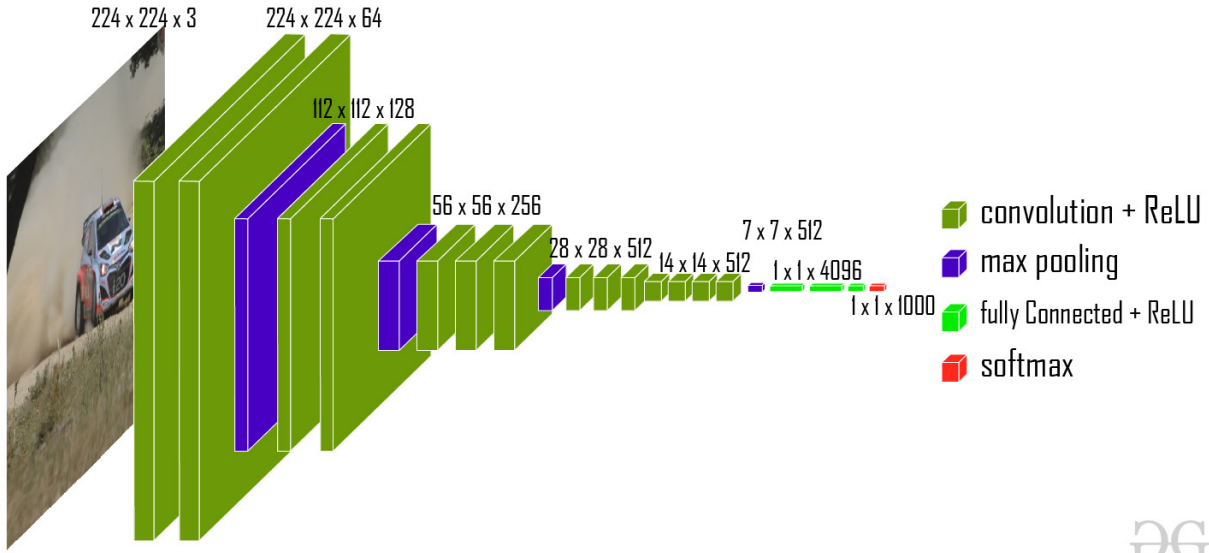
VGGNet



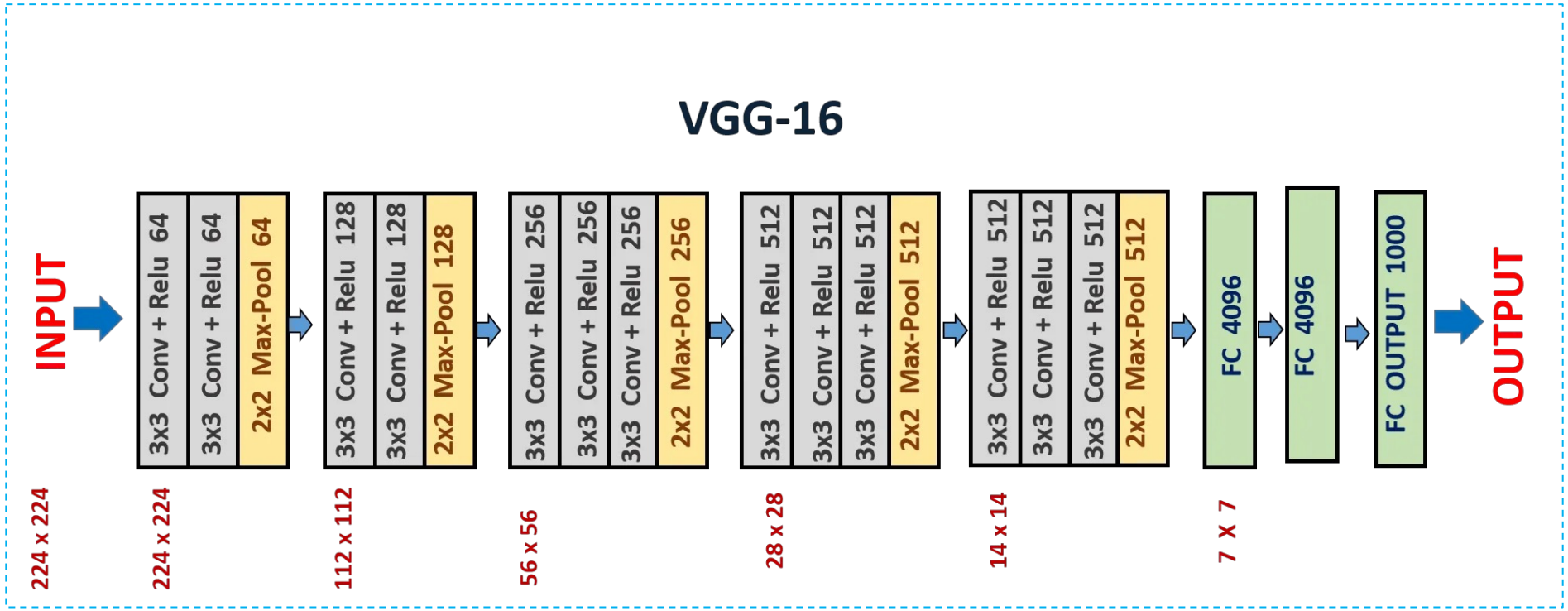


The full name of VGG is the **Visual Geometry Group**, which belongs to the Department of Science and Engineering of Oxford University.

The original purpose of VGG's research on the depth of convolutional networks is to understand how the depth of convolutional networks affects the accuracy and accuracy of large-scale image classification and recognition.



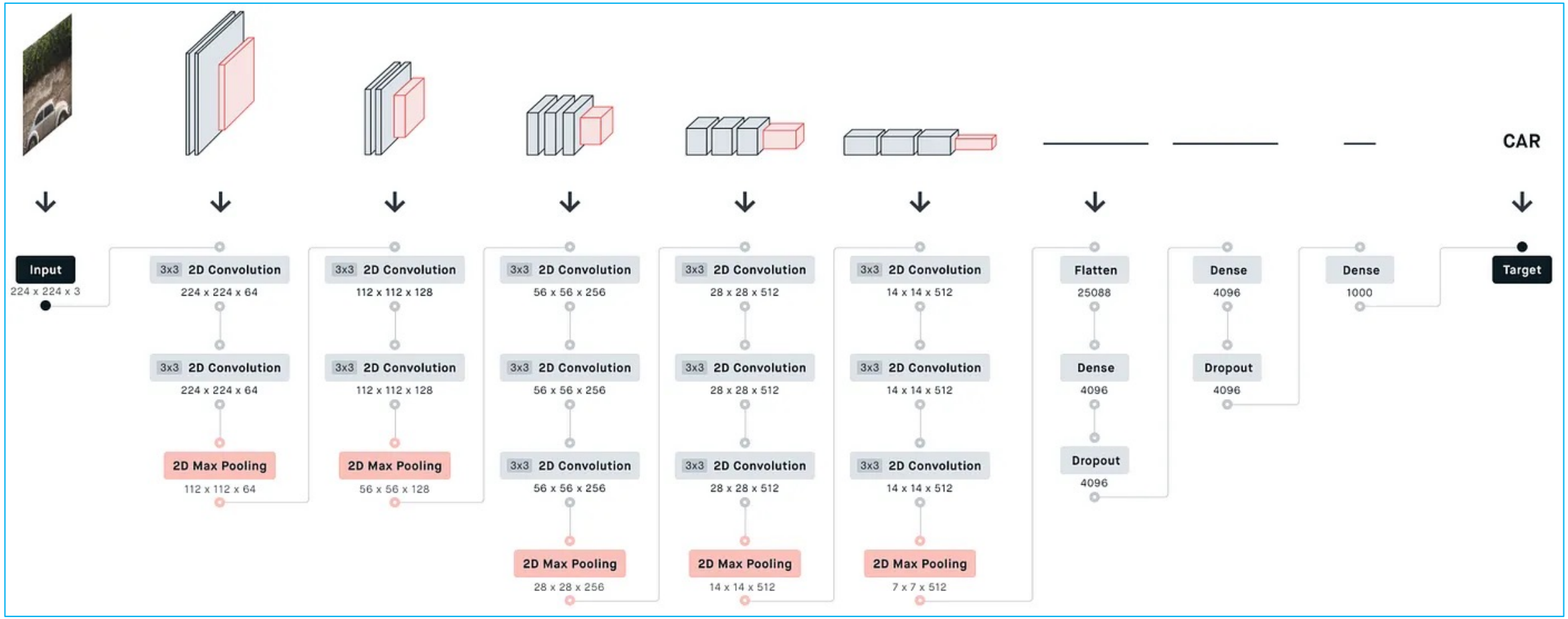
VGG16Net



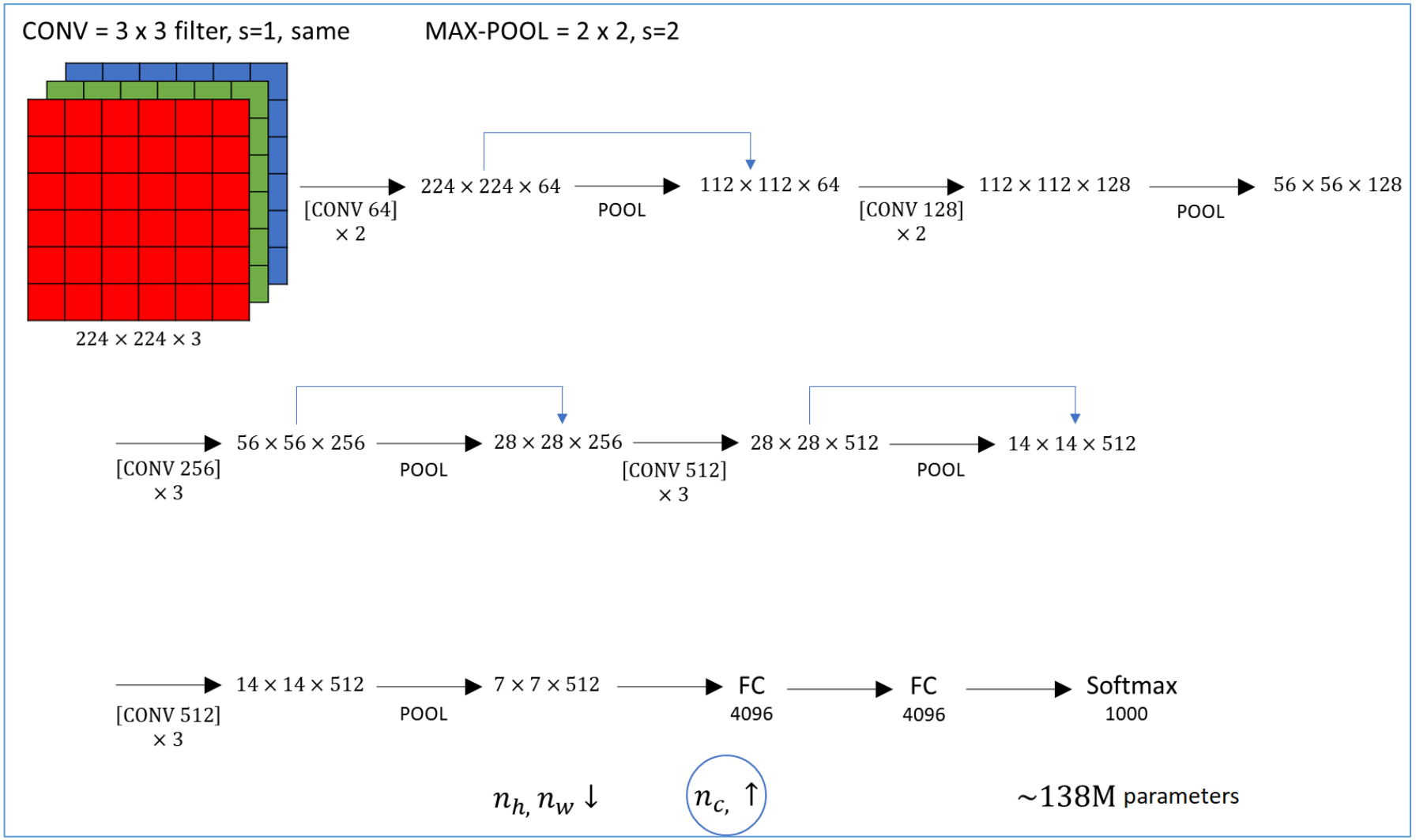
AlexNet came out in 2012 and was a revolutionary advancement; it improved on traditional Convolutional Neural Networks (CNNs) and became one of the best models for image classification… until VGG came out.

Use of a very small 3 x 3 receptive field (filters) throughout the entire network with the stride of 1 pixel. Please note that the receptive field in the first layer in AlexNet was 11 x 11 with stride 4, and the same was 7 x 7 in ZFNet with stride 2.

VGG16Net

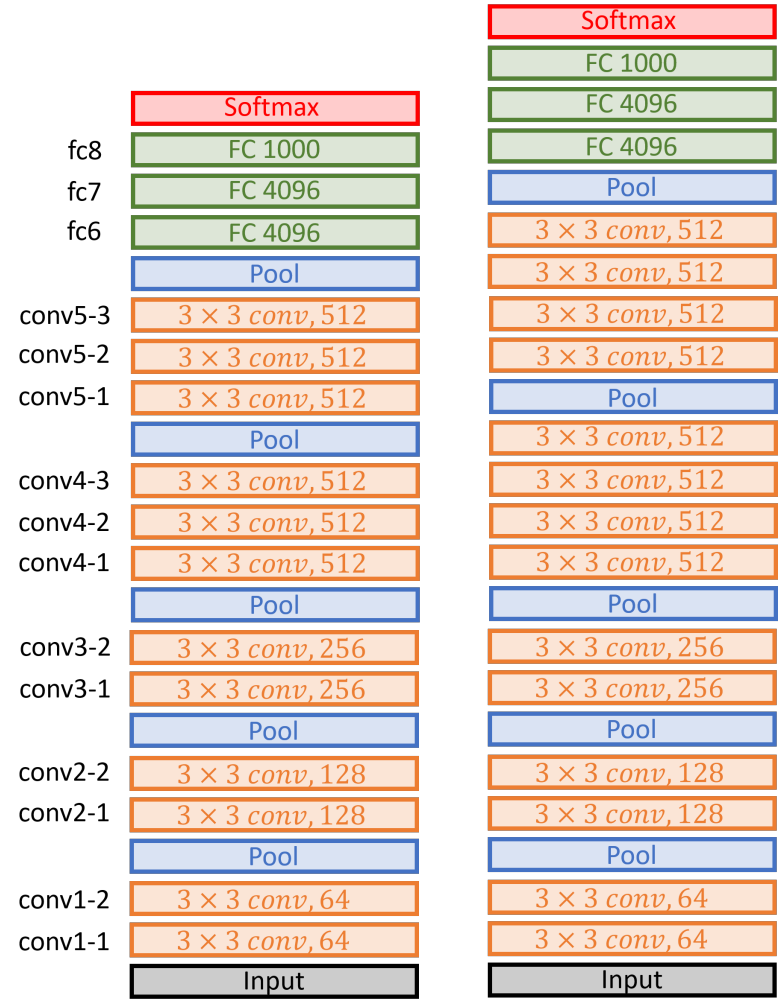

Input
224 x 224 x 3
3x3 2D Convolution
224 x 224 x 64
3x3 2D Convolution
224 x 224 x 64
2D Max Pooling
112 x 112 x 64
3x3 2D Convolution
112 x 112 x 128
3x3 2D Convolution
112 x 112 x 128
2D Max Pooling
56 x 56 x 128
3x3 2D Convolution
56 x 56 x 256
3x3 2D Convolution
56 x 56 x 256
3x3 2D Convolution
56 x 56 x 256
2D Max Pooling
28 x 28 x 256
3x3 2D Convolution
28 x 28 x 512
3x3 2D Convolution
28 x 28 x 512
3x3 2D Convolution
28 x 28 x 512
2D Max Pooling
14 x 14 x 512
3x3 2D Convolution
14 x 14 x 512
3x3 2D Convolution
14 x 14 x 512
3x3 2D Convolution
14 x 14 x 512
2D Max Pooling
7 x 7 x 512
Flatten
25088
Dense
4096
Dropout
4096
Dense
4096
Dropout
4096
Dense
1000
CAR
Target

VGG16Net


CONV = 3 x 3 filter, s=1, same
MAX-POOL = 2 x 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
[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
$n_h, n_w \downarrow$
$n_c \uparrow$
~138M parameters

VGG16Net

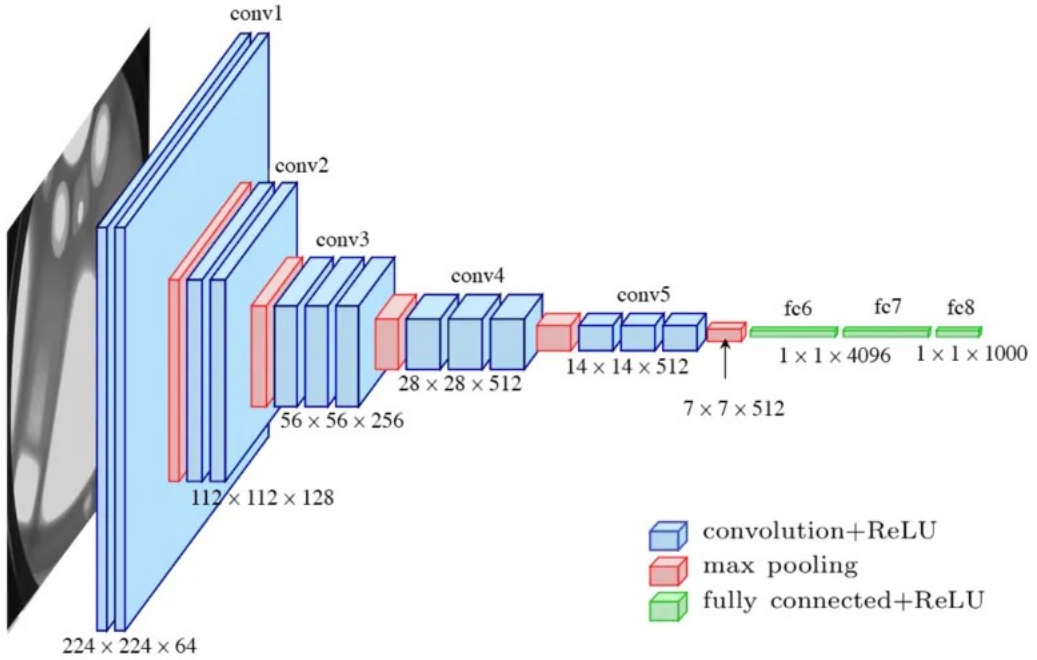
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax



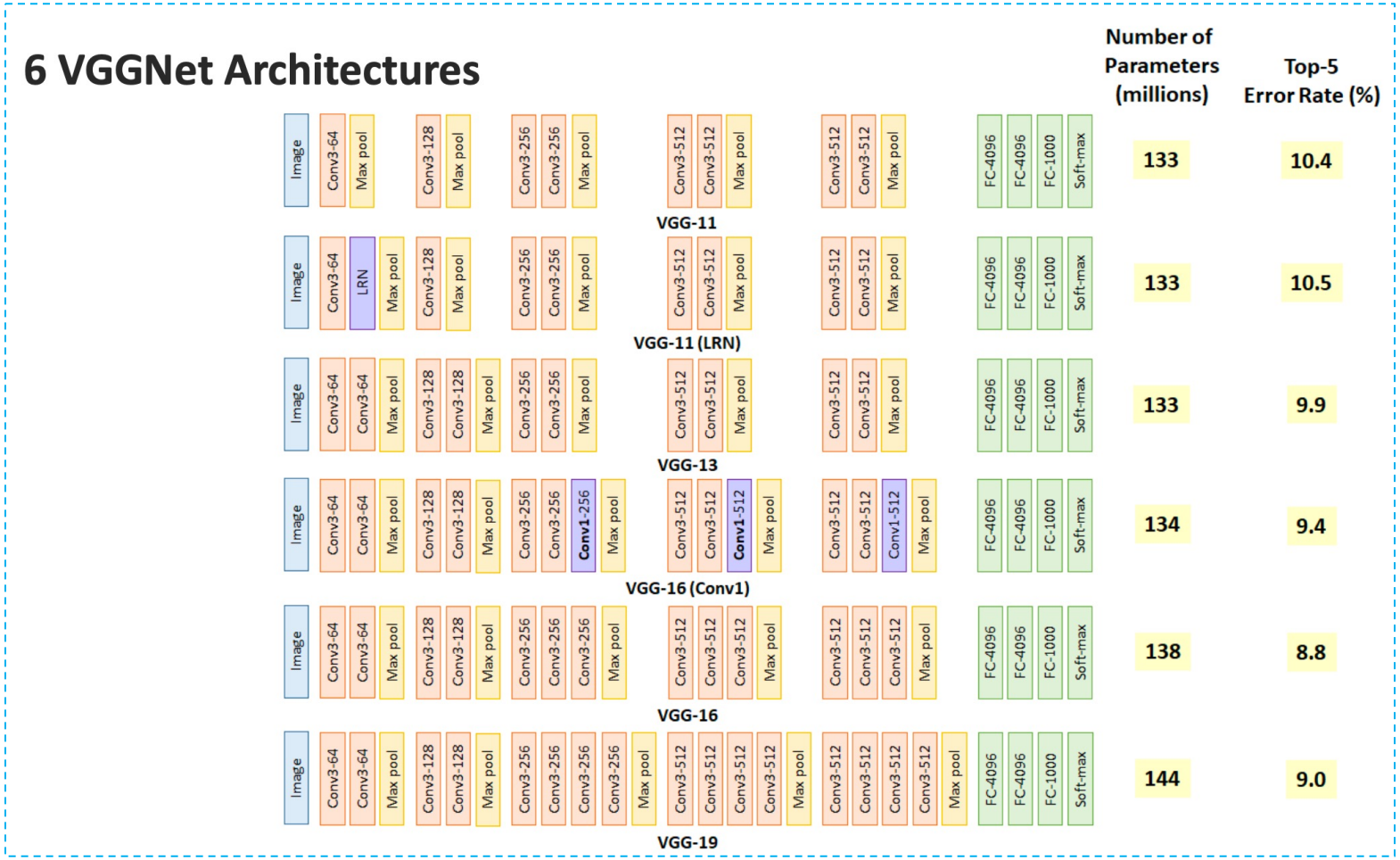
VGGNet: Parameters

The VGG network has five configurations named A to E.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



VGGNet: Parameters

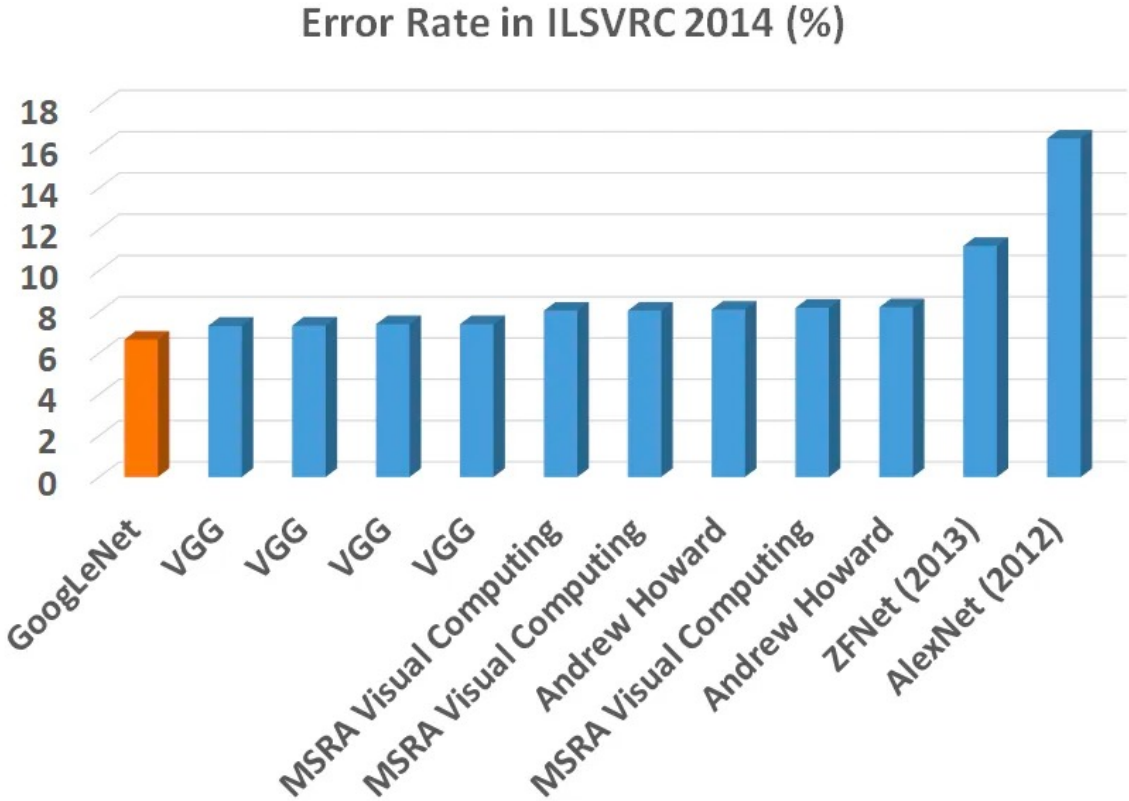


Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

GoogleLeNet: Motivation



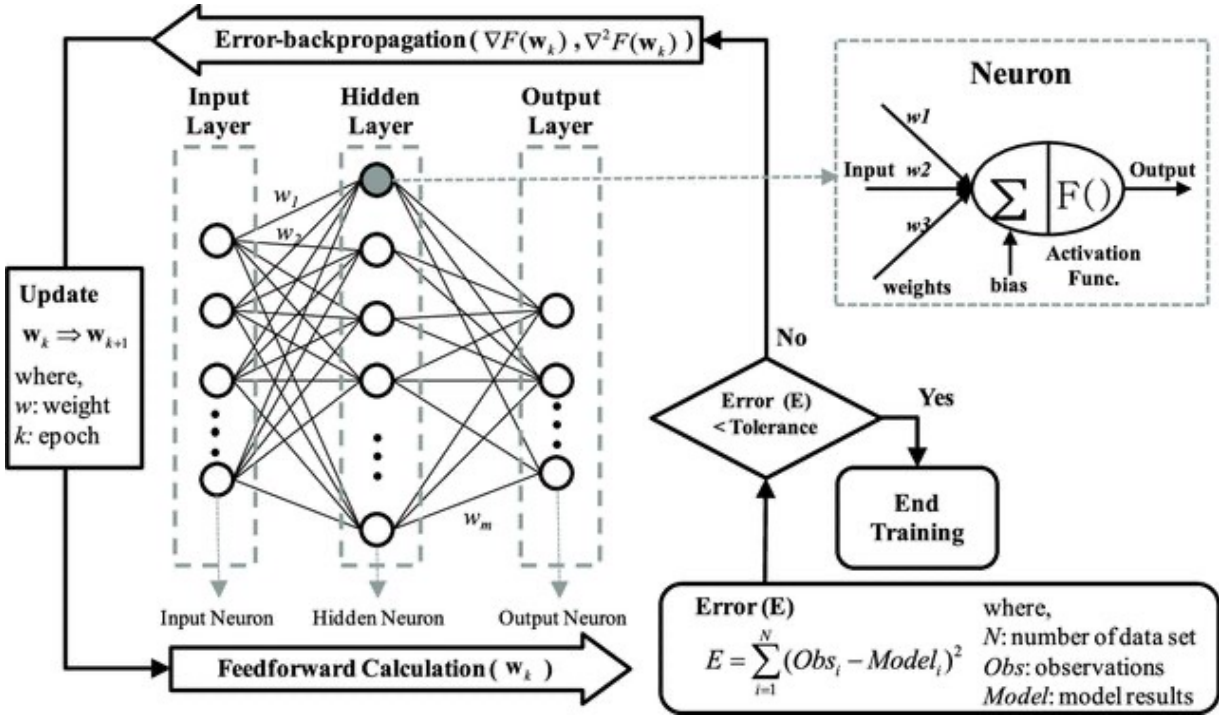
•GoogleLeNet introduced by google in 2014 ,which is the winner of ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14)

Standard CNN structure up until 2014 was stacked convolutional layers, maybe max-pooling, then one or more fully-connected layers
This has limits-

- Large memory footprint
- Large computation demand
- Prone to overfitting
- Vanishing and exploding gradients

GoogleLeNet: Motivation

A glimpse of the Backpropagation Algorithm



Signs of vanishing gradient	Signs of Exploding gradient
The weights of the model may become zero while training	The weights of the model may become NaN while training
The weights near the output layer show significant changes, while the weights near the output layer may show no change at all	The weights of the model grow exponentially
The model learns very slowly	The model may show huge changes in the loss
The model may stop learning just after a few epochs	The model is not able to learn from the training data and performs poorly

https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/

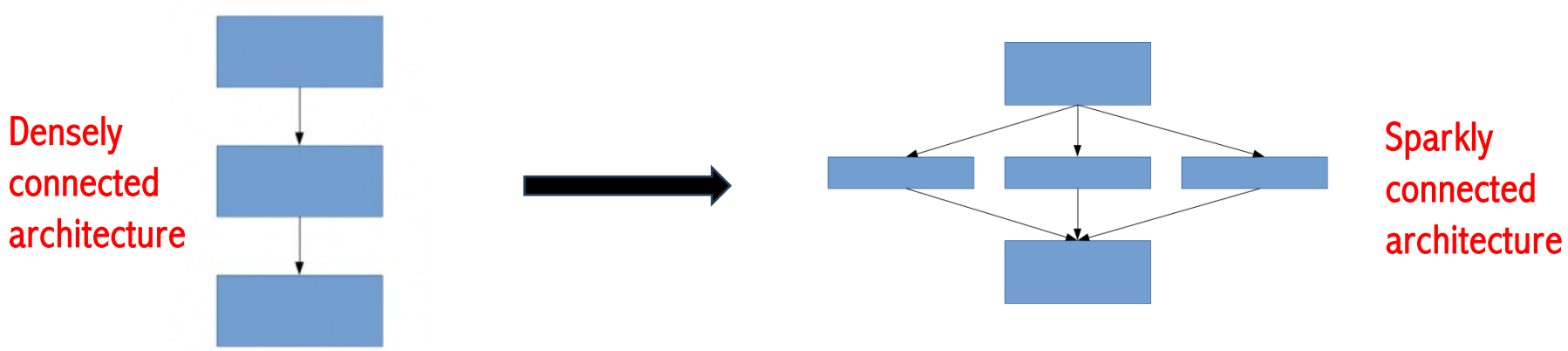
GoogleLeNet: Motivation

Problems

- Bigger size typically means a larger number of parameters, which makes the enlarged network more prone to overfitting, especially if the number of labeled examples in the training set is limited.
- The other drawback of uniformly increased network size is the dramatically increased use of computational resources.

Solution

Introduce sparsity and replace the fully connected layers by the sparse ones, even inside the convolutions.

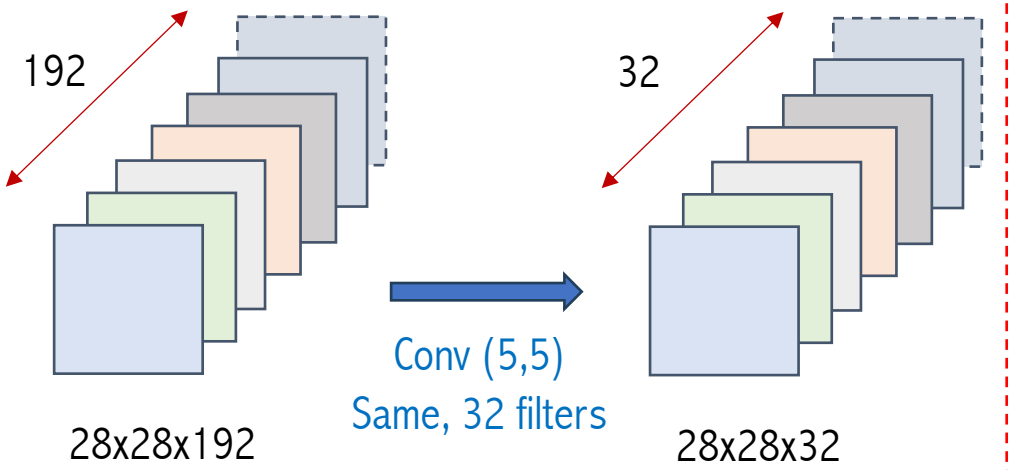


https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/

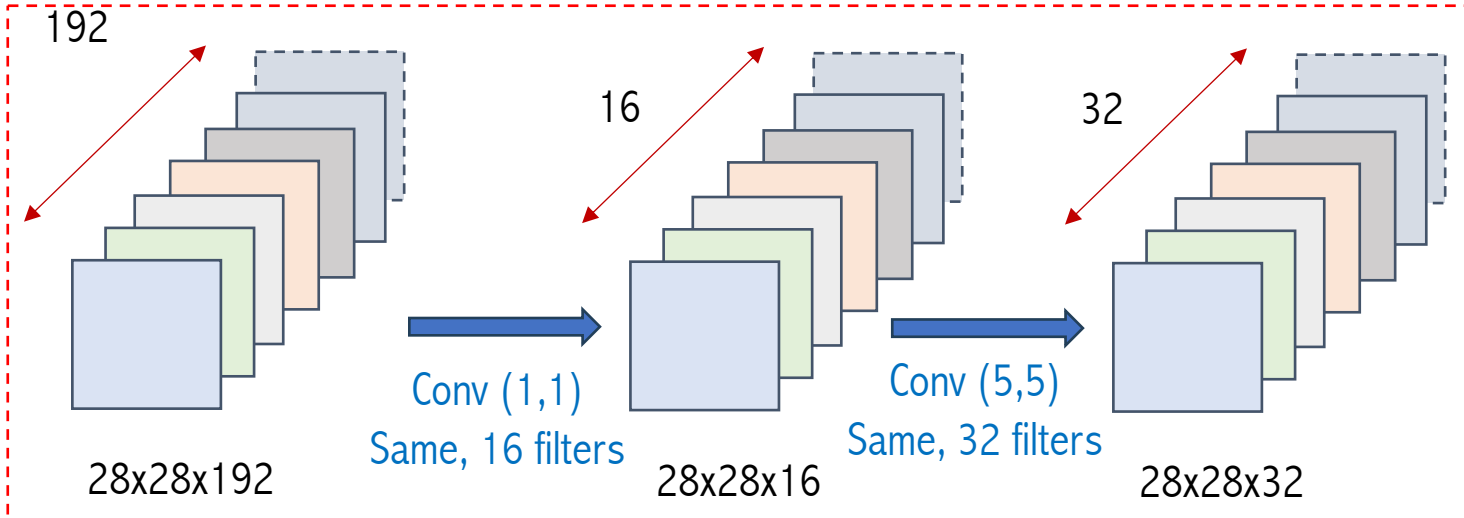
GoogleLeNet : Inception Net

New: 1×1 convolution is used as a dimension reduction module to reduce the computation

5x5 convolution without using 1x1 bottleneck convolution



5x5 convolution using 1x1 bottleneck convolution



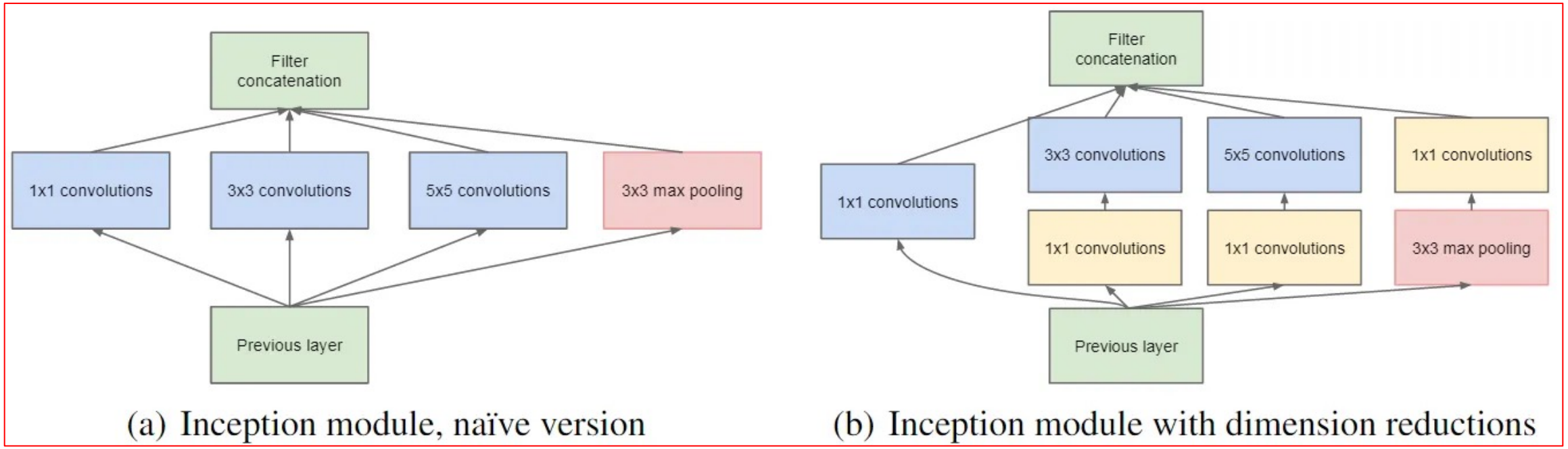
Number of parameters =28x28 (size of the input image)x 5x5(size of filter) x 192 (channels) x 32(no of filters) = 120,422,400 operations

Number of parameters =28 x 28 x 16 x 192 + 28 x 28 x 32 x 5 x 5 x 16 ~ 12.4M

*As the authors said that the idea of the name "**Inception**" comes from famous internet meme below: **WE NEED TO GO DEEPER***

GoogLeNet: Inception Net

New: Inception Module

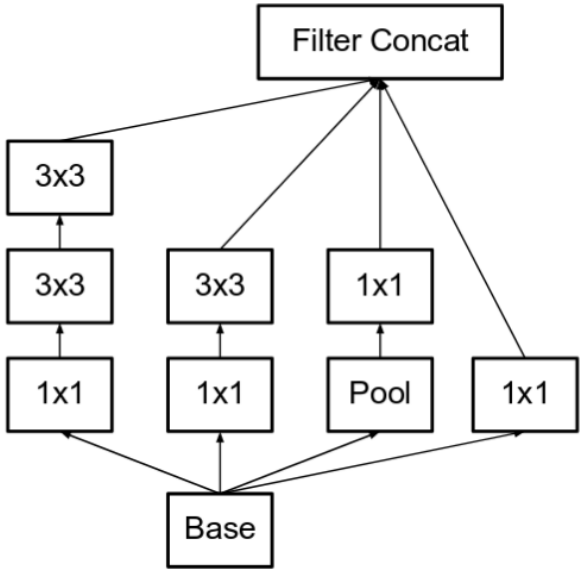


Inception V1 Module

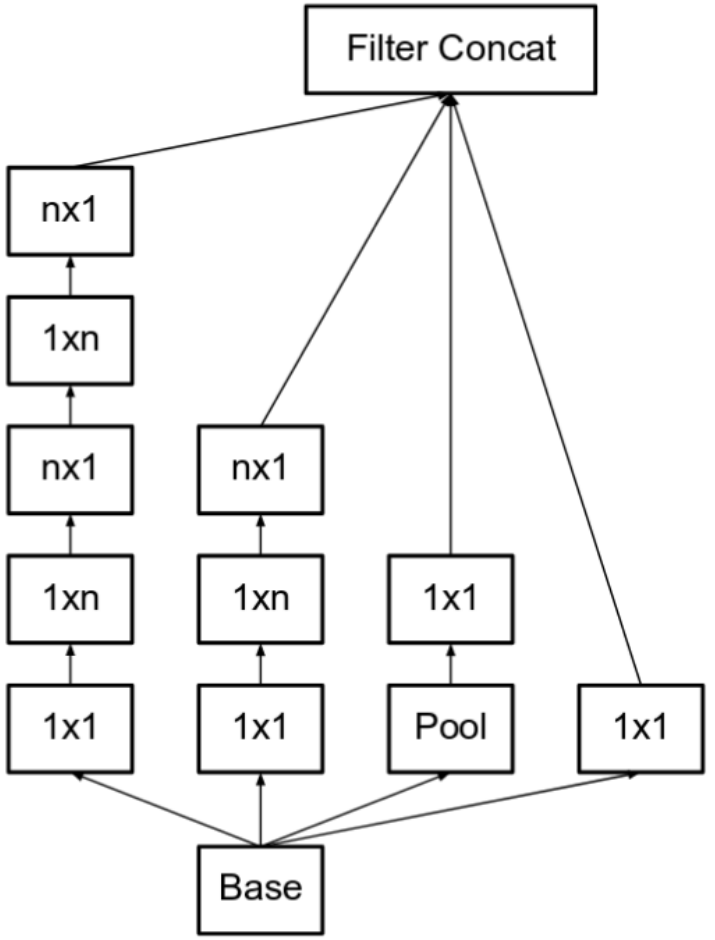
Inception V1 Optimizated

GoogLeNet: Inception Net

New: Inception Module

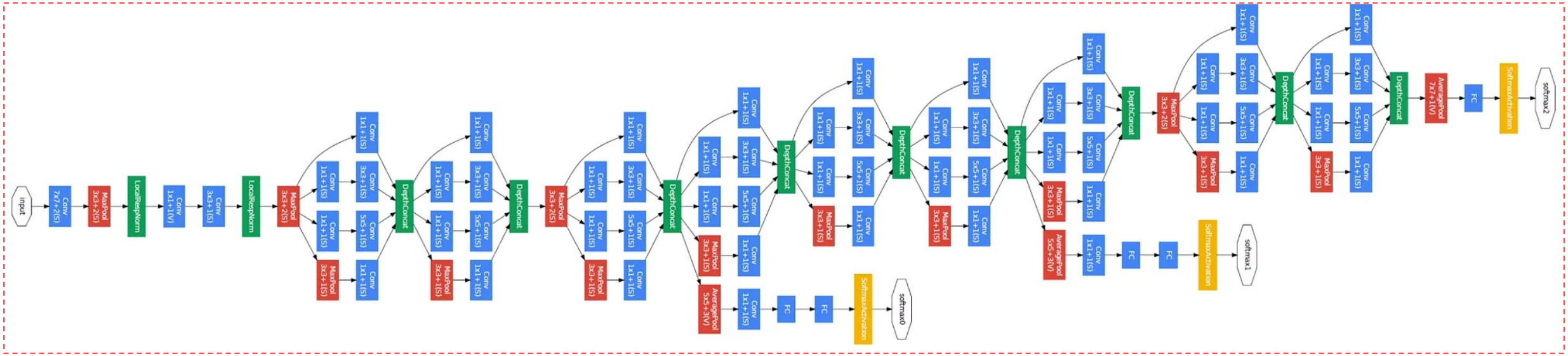


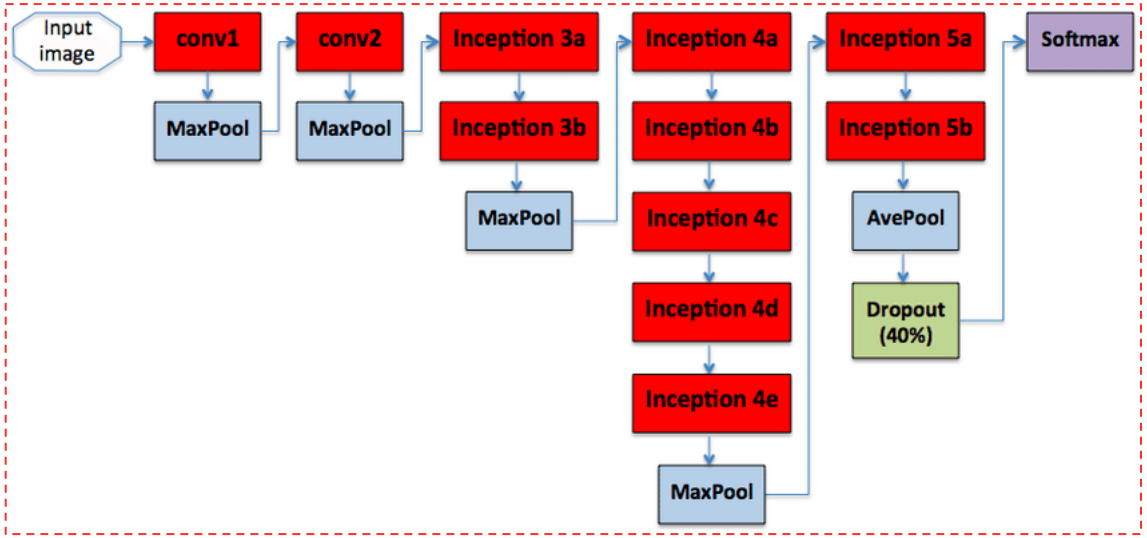
Inception V2 Module



Inception V2 Module

GoogleLeNet



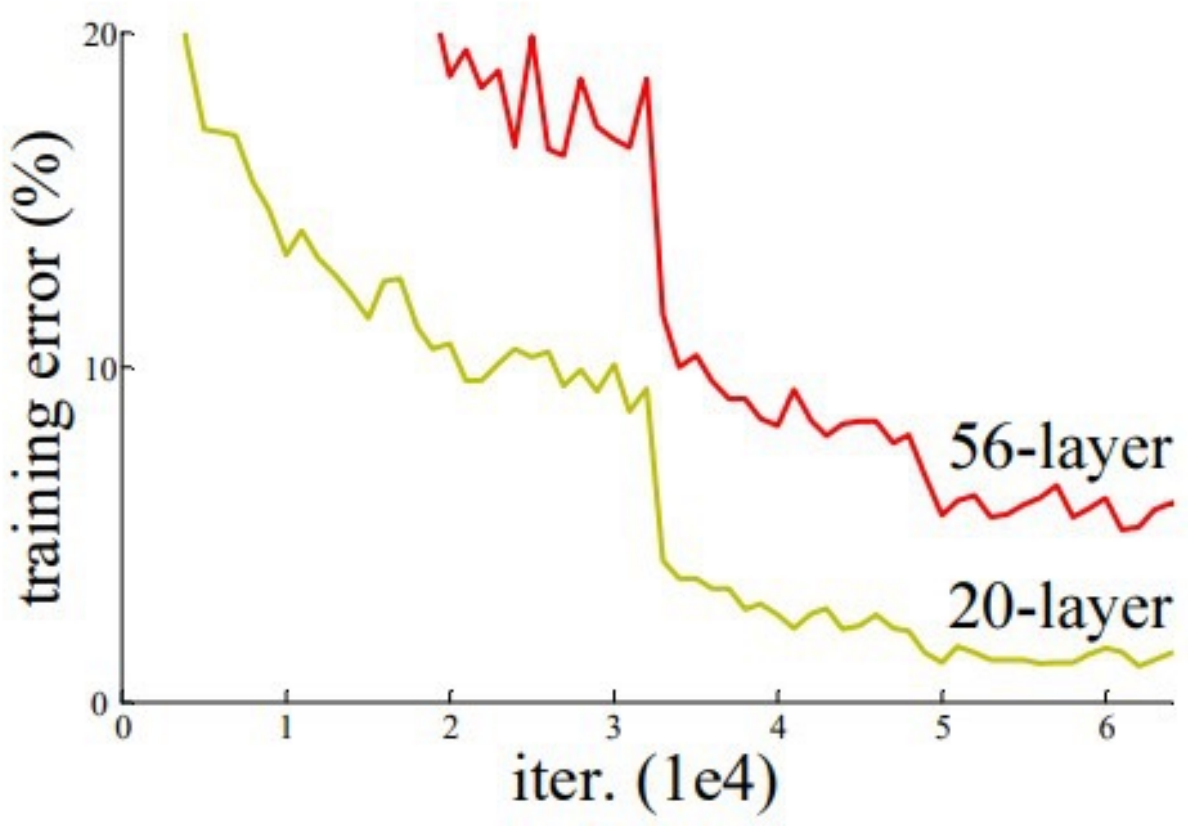


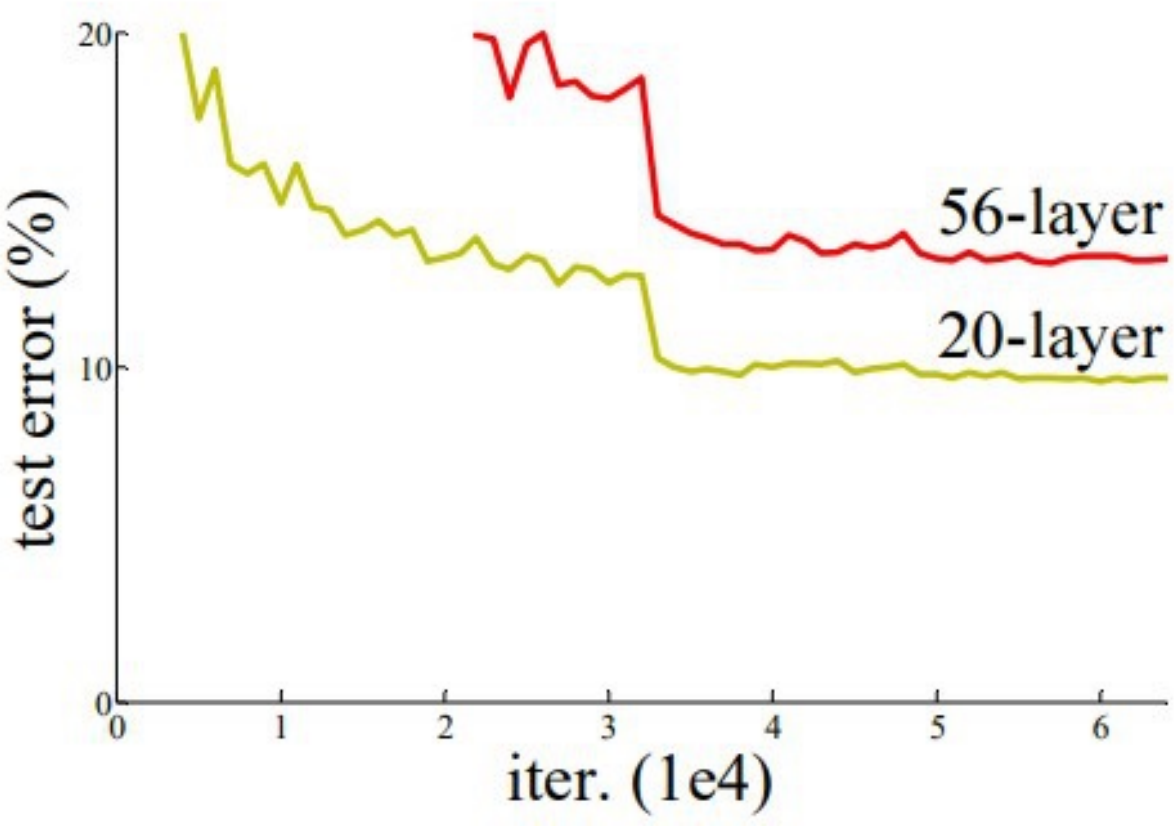
There are a total of 22 layers, It is already a very deep model compared with previous AlexNet, ZFNet, and VGGNet

Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

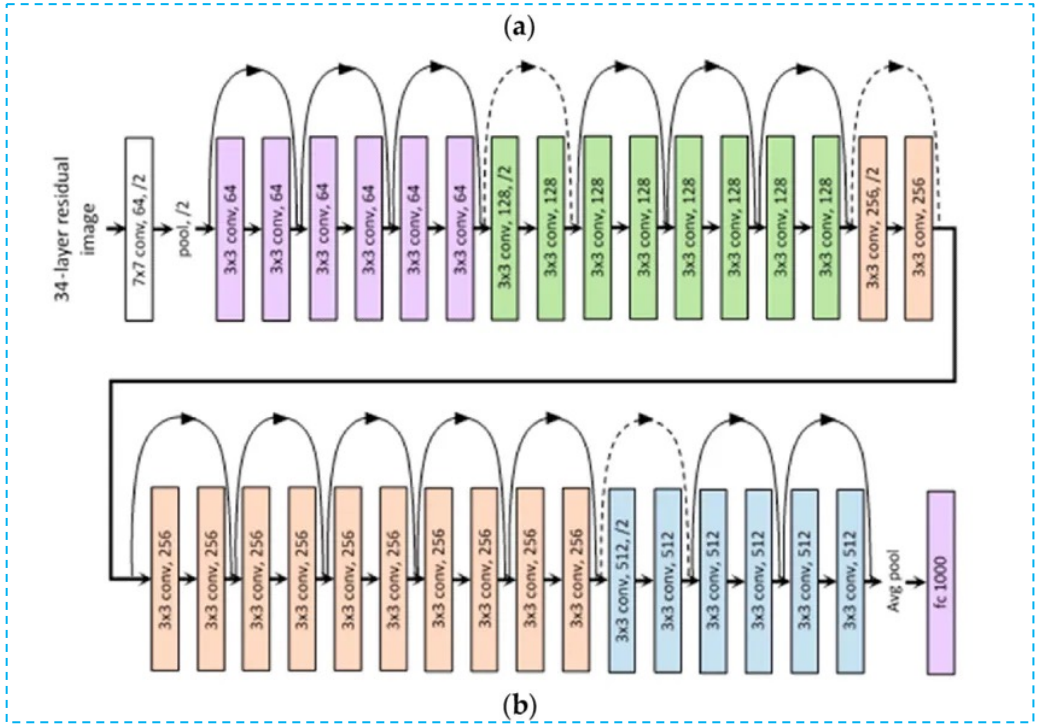
Residual Network (ResNet)

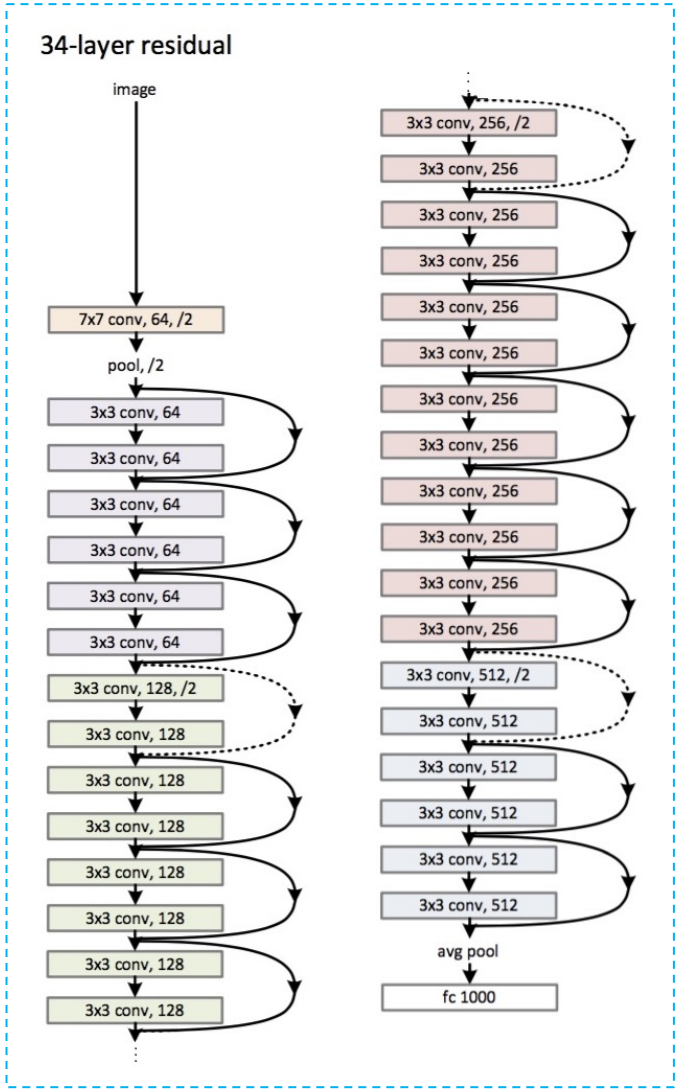




The Residual Blocks idea was created by this design to address the issue of the vanishing/exploding gradient

Residual Network (ResNet)

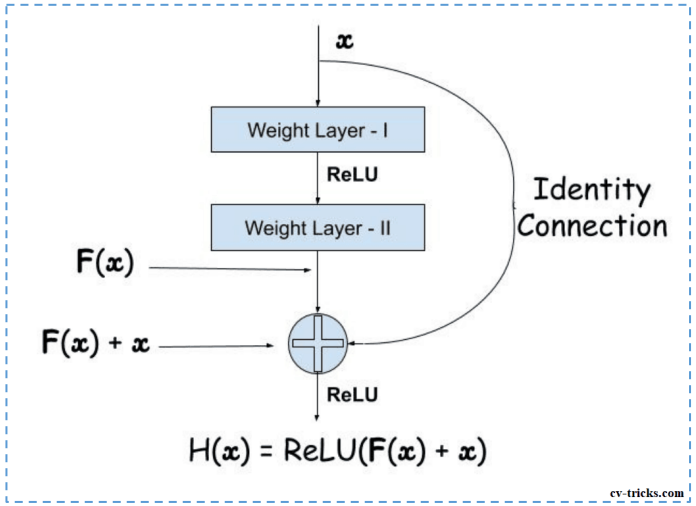




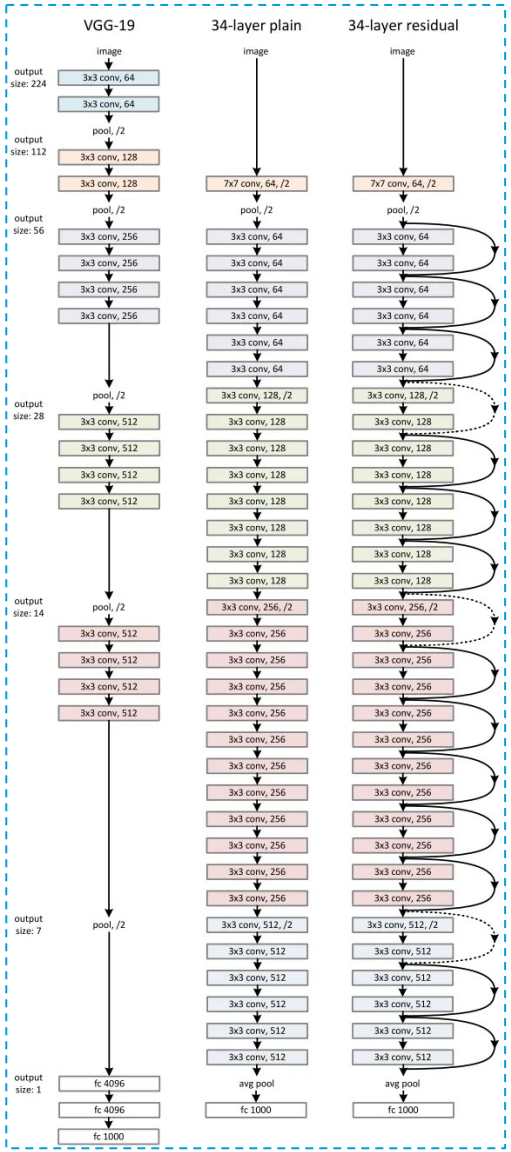
A novel architecture called **Residual Network** was launched by Microsoft Research experts in 2015 with the proposal of **ResNet.**

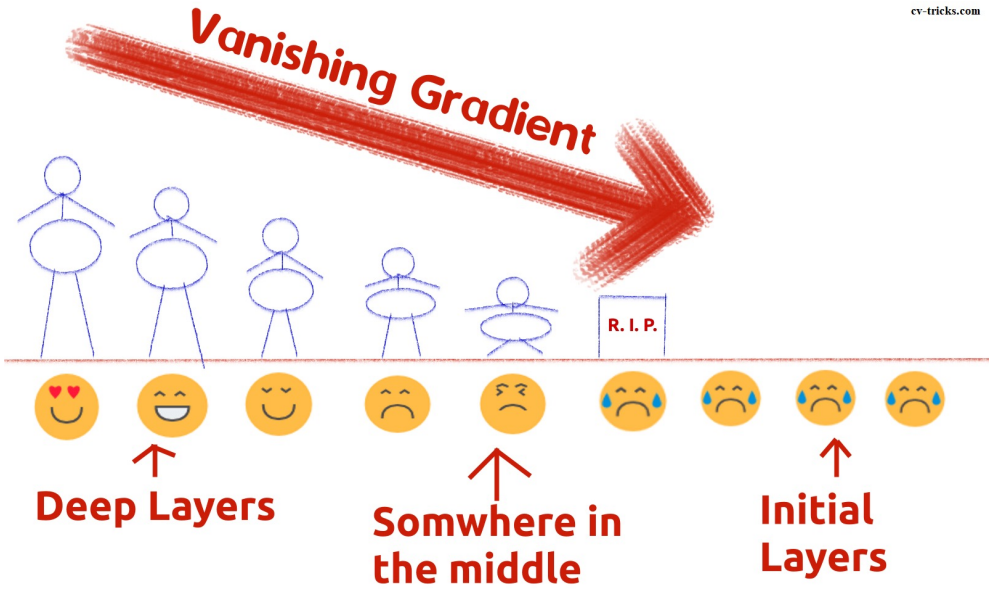
Residual Network (ResNet34)

New: Skip Connection



The intuition is that learning f(x) = 0 has to be easy for the network.





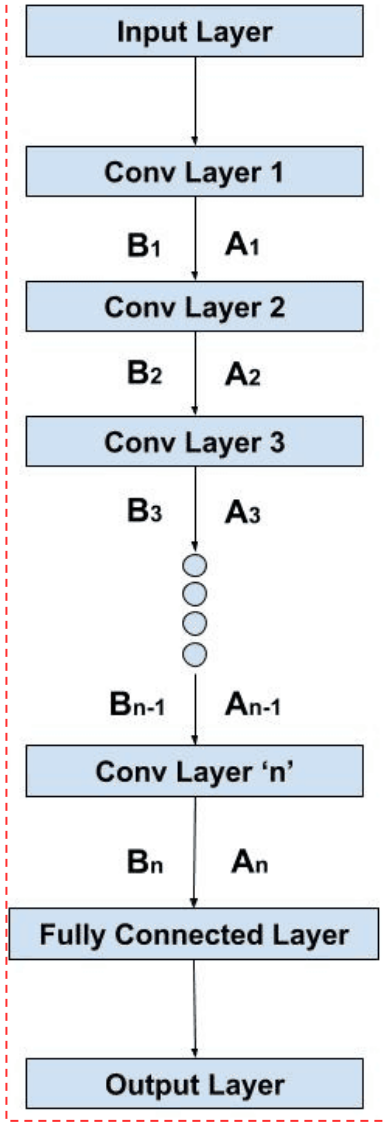
The VGG-19-inspired 34-layer plain network architecture used by ResNet is followed by the addition of the shortcut connection

Residual Network (ResNet34)

New: Skip Connection

Assumed Deep Convolutional Neural Network

Traditional networks such as AlexNet, VGG-16, etc try to learn the A_1, A_2, A_3, …. A_n directly as shown in the diagram of the simple deep network. In the forward pass, the input (image) is passed to the network to get the output. The error is calculated and gradients are determined and backpropagation helps the network to approximate the functions A_1, A_2, A_3, …. A_n in the form of B_1, B_2, B_3, …. B_n



Simple Deep Network

1- Note: Pooling and Activation functions are not shown for brevity.

2- A_1, A_2, A_3, … A_n are the actual / ideal functions that are required by the network to achieve highest performance.

3- B_1, B_2, B_3, … B_n are the functions which will be an asymptotic (nearly) approximate of the ideal functions.

cv-tricks.com

The creators of ResNet thought that:

IF THE MULTIPLE NON-LINEAR LAYERS CAN ASYMPTOTICALLY APPROXIMATE COMPLICATED FUNCTIONS, THEN IT IS EQUIVALENT TO HYPOTHESIZE THAT THEY CAN ASYMPTOTICALLY APPROXIMATE THE RESIDUAL FUNCTION

Residual Network (ResNet34)

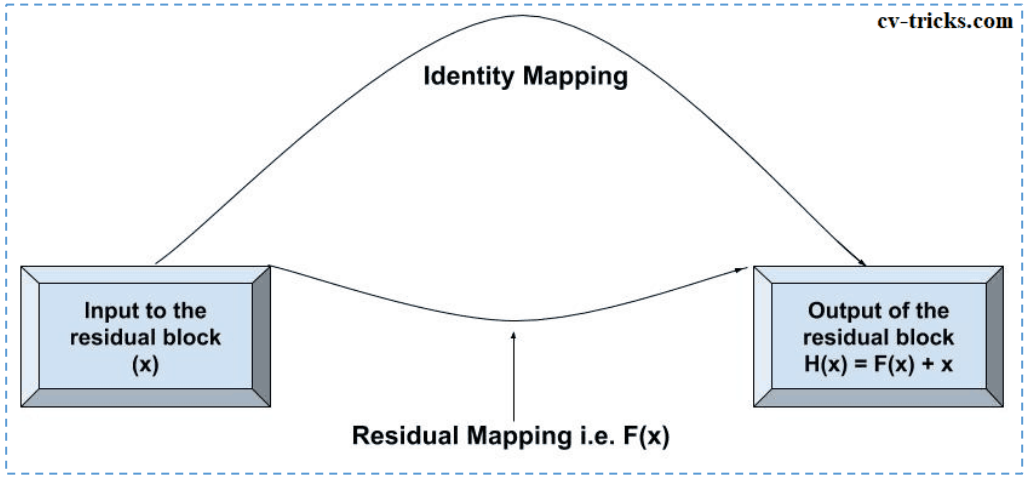
New: Skip Connection

What is a Residual Function?

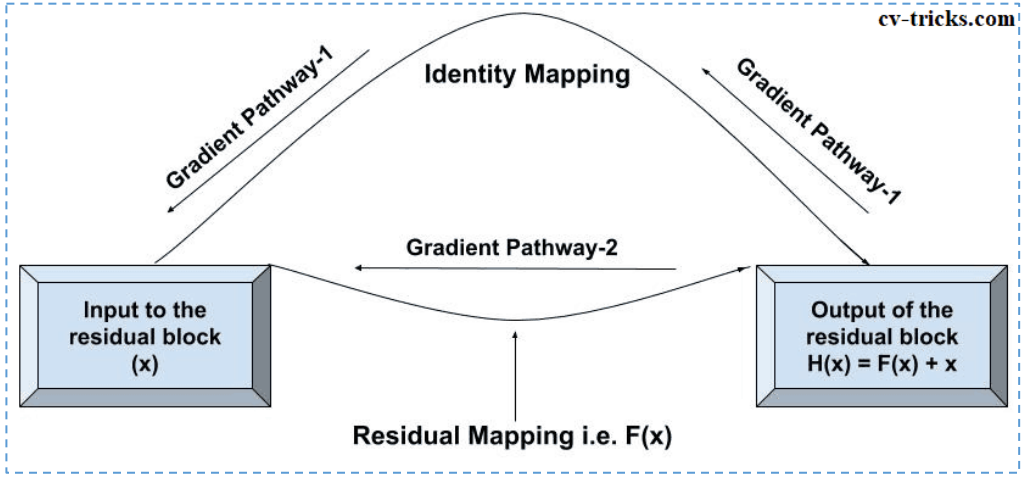
RATHER THAN EXPECTING STACKED LAYERS TO LEARN TO APPROXIMATE ***H(x),*** THE AUTHORS ARE LETTING THE LAYERS TO APPROXIMATE A RESIDUAL FUNCTION i.e. ***F(x) = H(x) – x.***

Gradients can flow directly through the skip connections backwards from later layers to initial filters.

Intuitive representation of Residual Function



Why identity mapping will work?

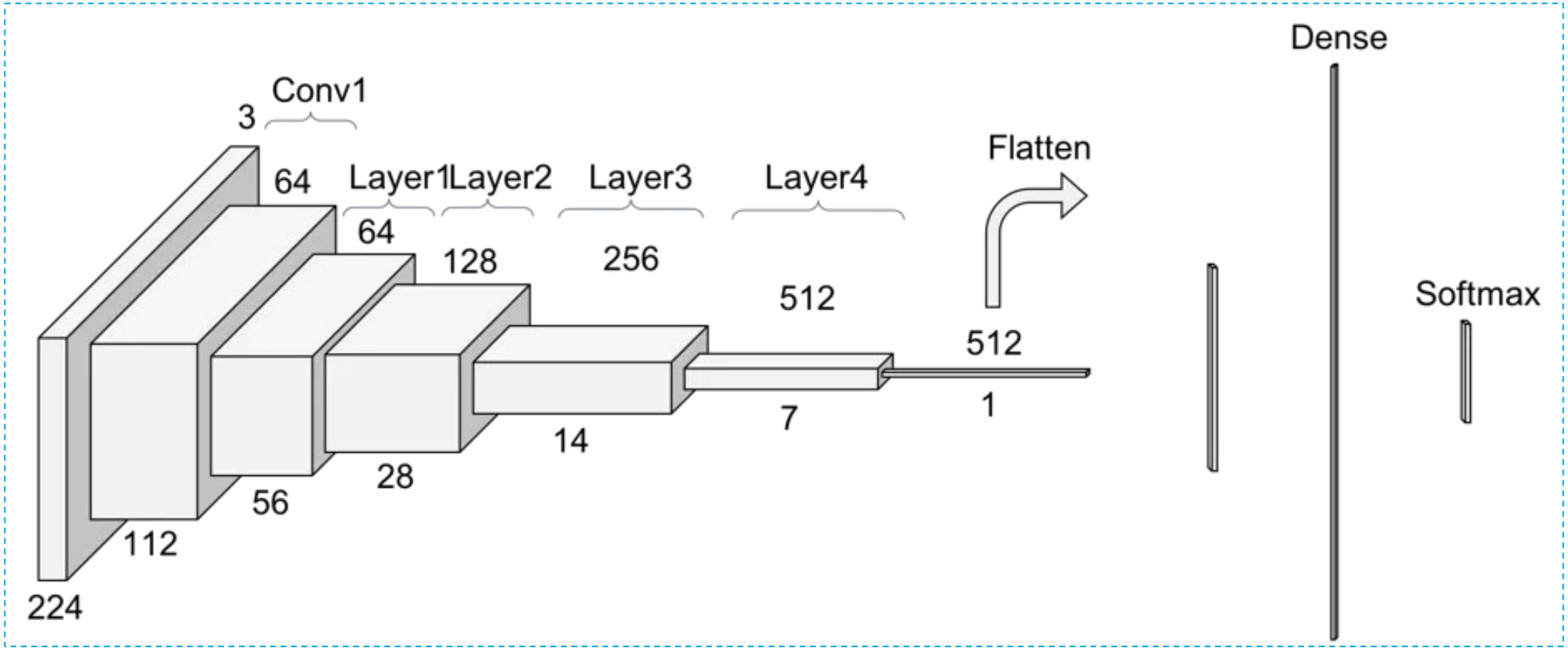


Gradient Pathways in ResNet

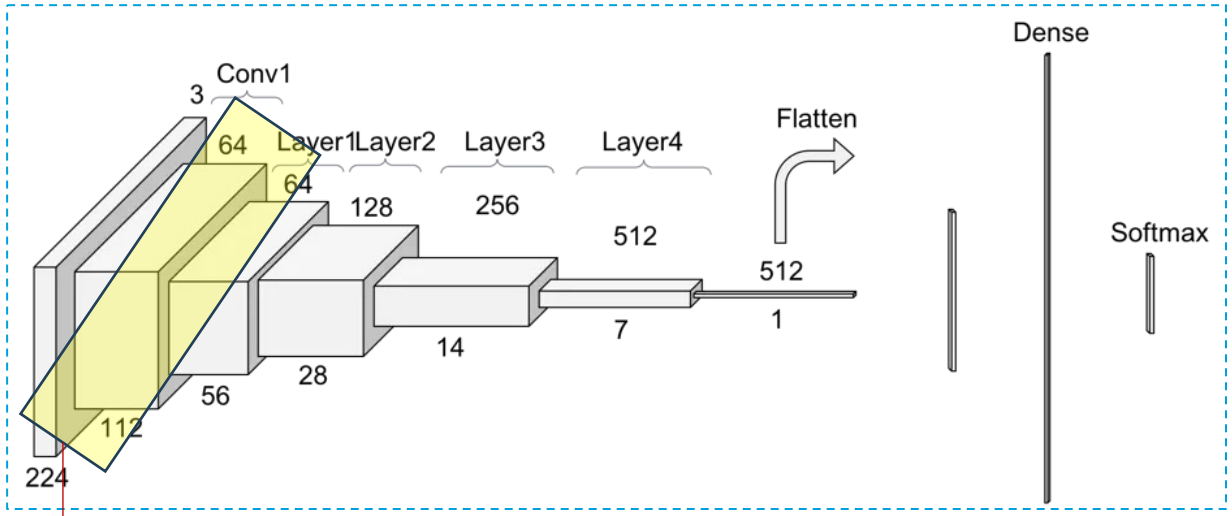
Residual Network

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

ResNet 34: Another look

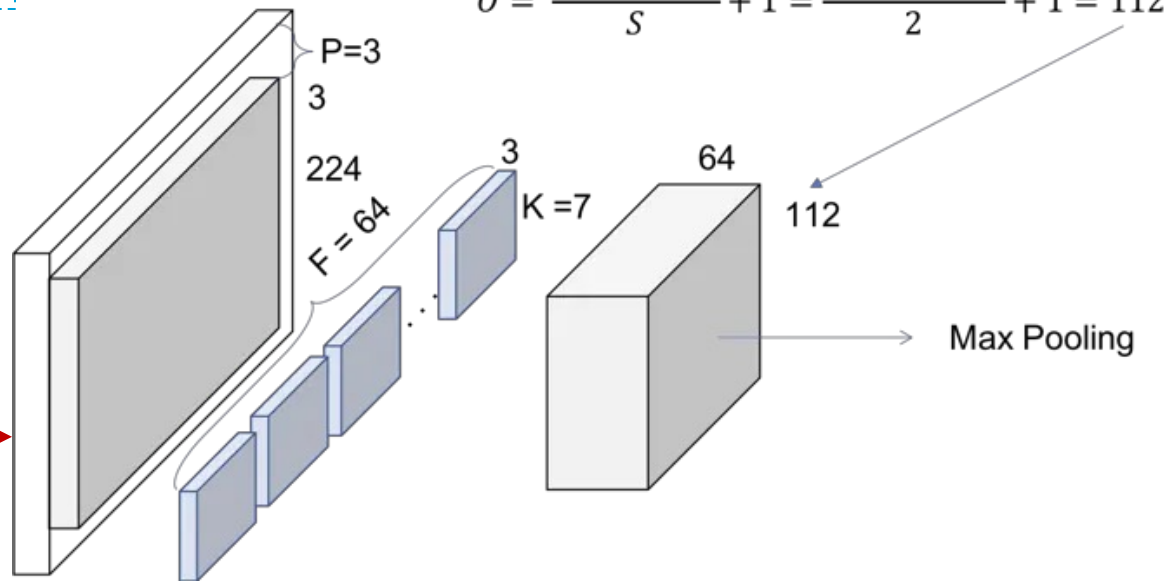


ResNet 34: Another look

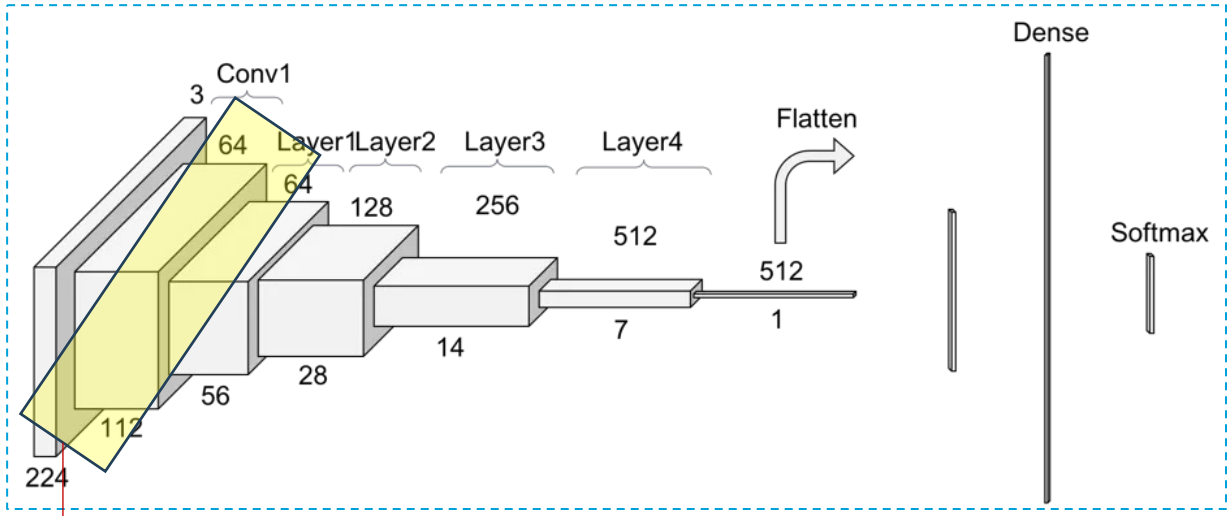


Consisting on a convolution + batch normalization + max pooling operation

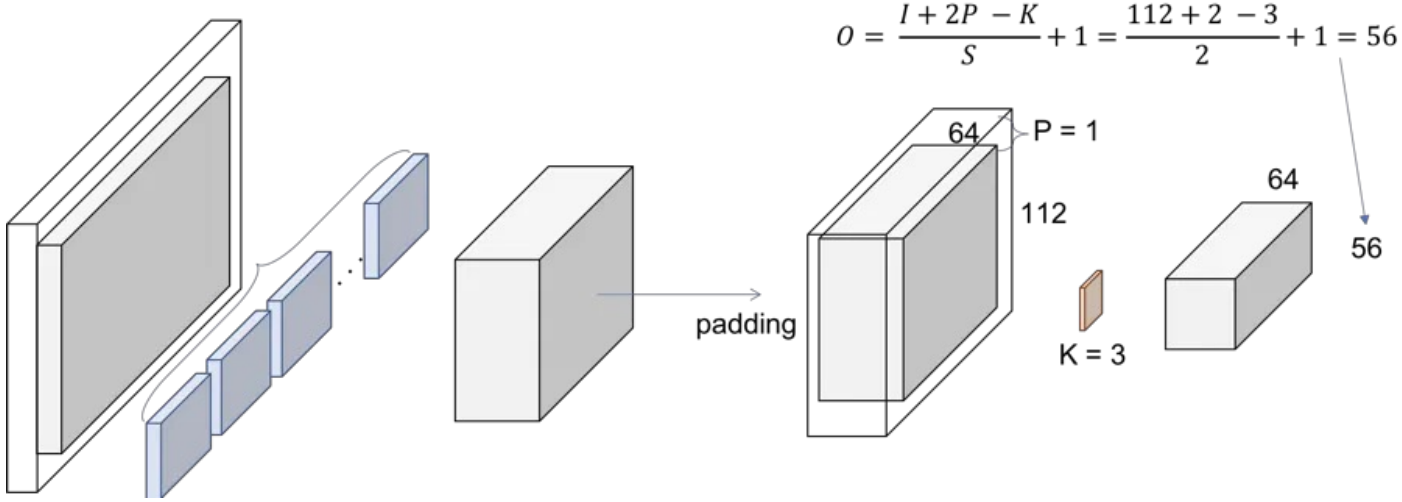
$$O = \frac{I + 2P - K}{S} + 1 = \frac{224 + 6 - 7}{2} + 1 = 112$$



ResNet 34: Another look



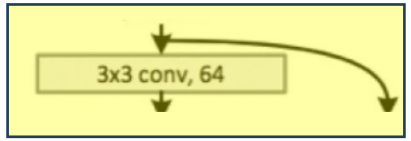
Consisting on a convolution + batch normalization + max pooling operation



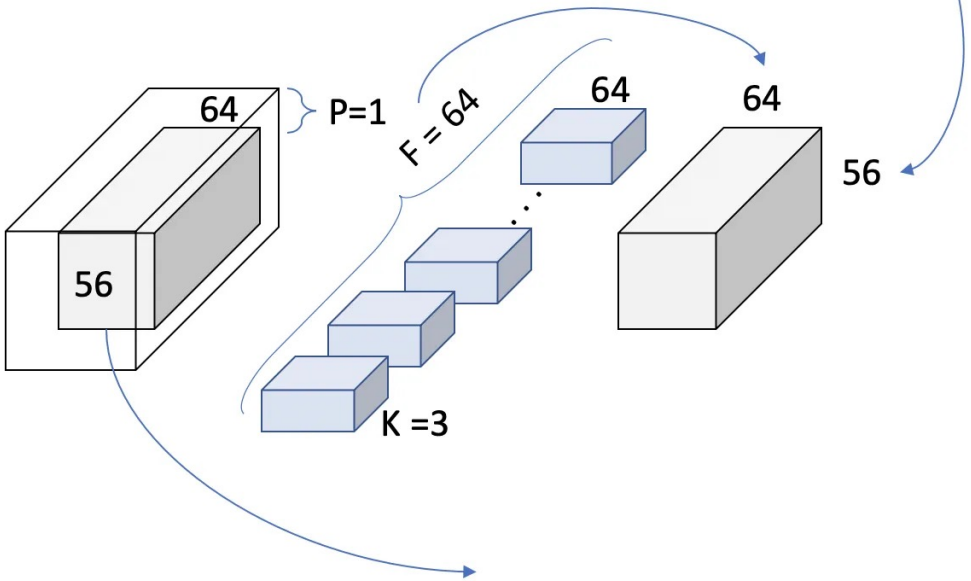
Resnet 34: Another Look

Every layer of a ResNet is composed of several blocks

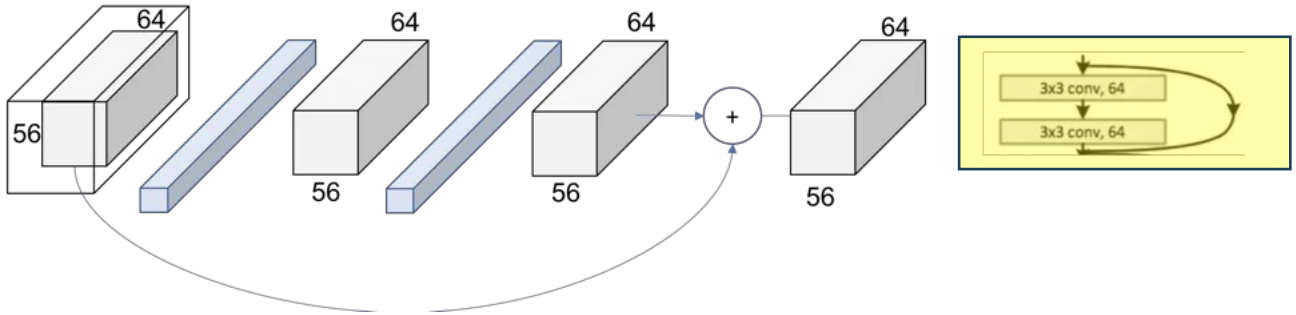
Layer 1, block 1, operation 1



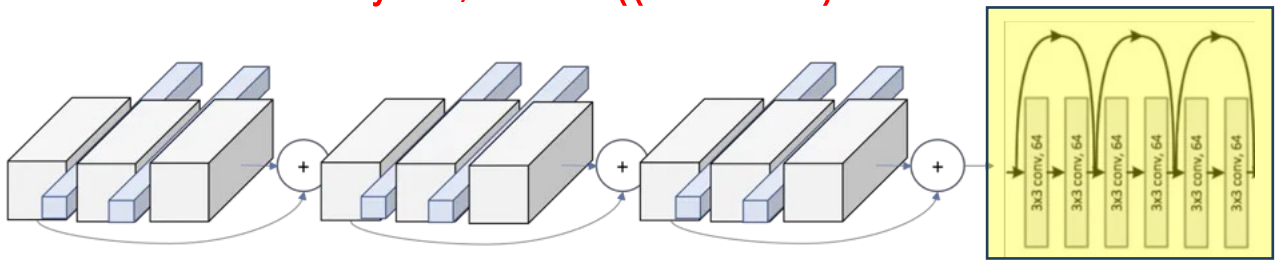
$$O = \frac{I + 2P - K}{S} + 1 = \frac{56 - 3 + 2}{1} + 1 = 56$$



Layer 1, block 1 (ResNet 18)



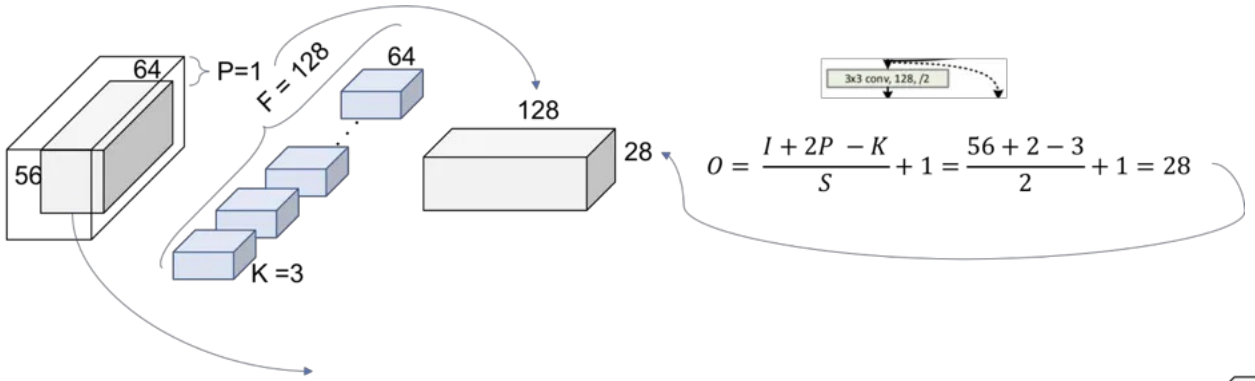
Layer 1, block 1 ((ResNet 18)

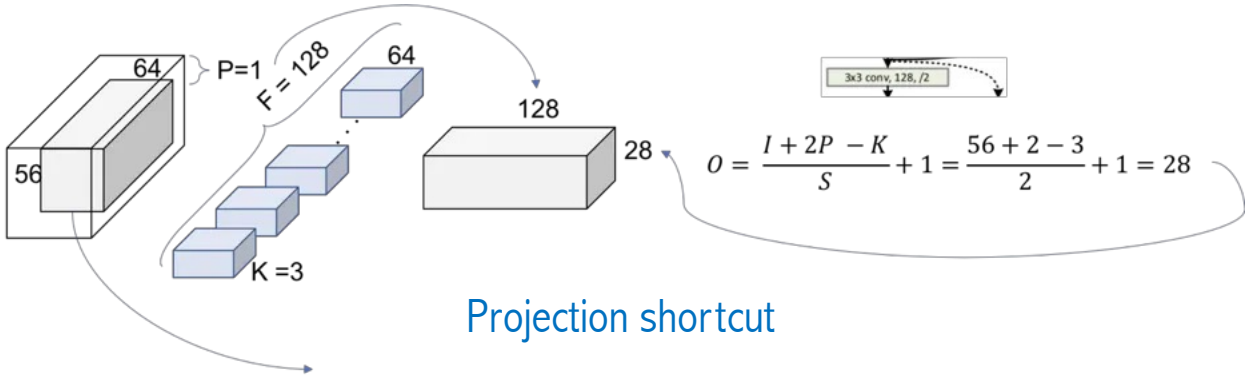


ResNet

Identity Shortcut and *Projection Shortcut*

Down sampling performed by increasing the stride to 2

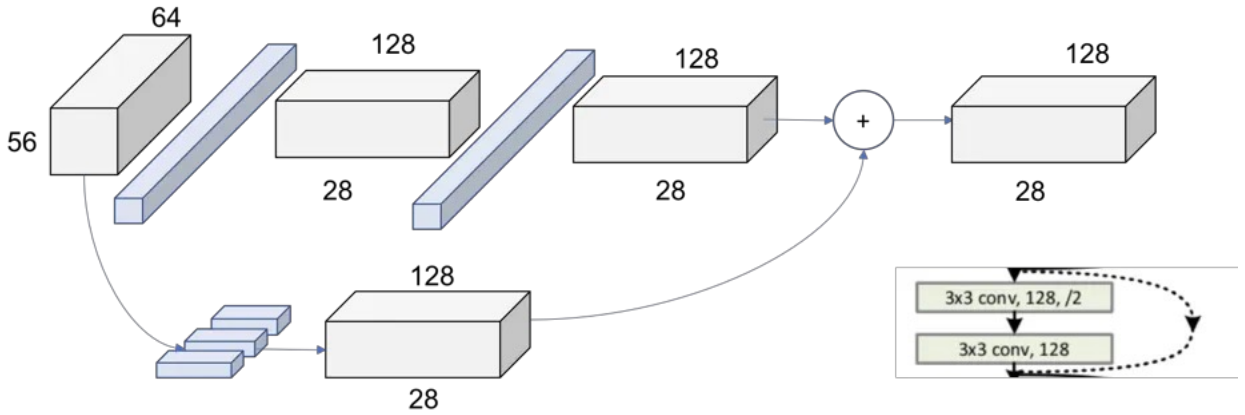




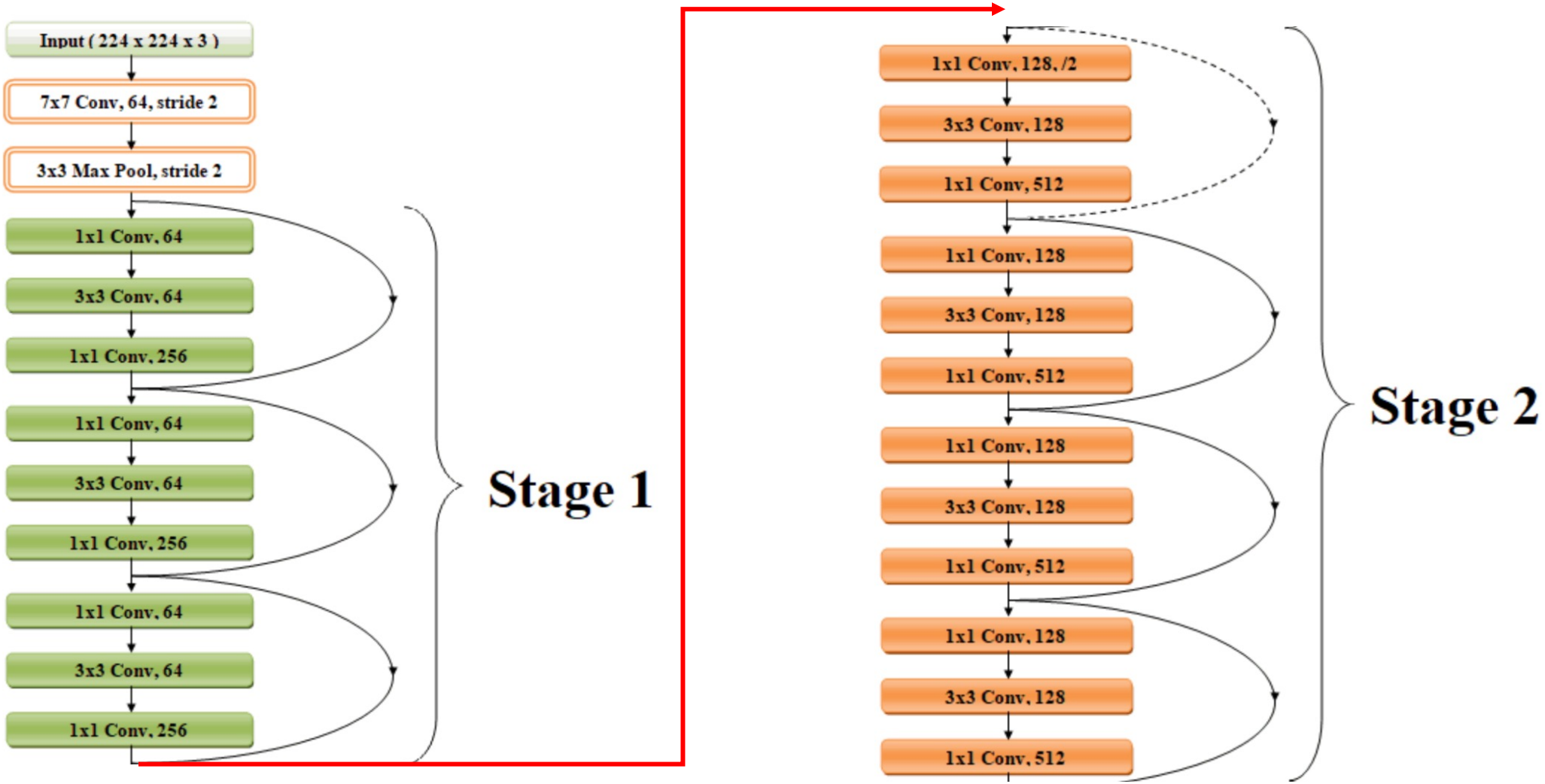
Projection shortcut

Number of Layers	Number of Parameters
ResNet 18	11.174M
ResNet 34	21.282M
ResNet 50	23.521M
ResNet 101	42.513M
ResNet 152	58.157M

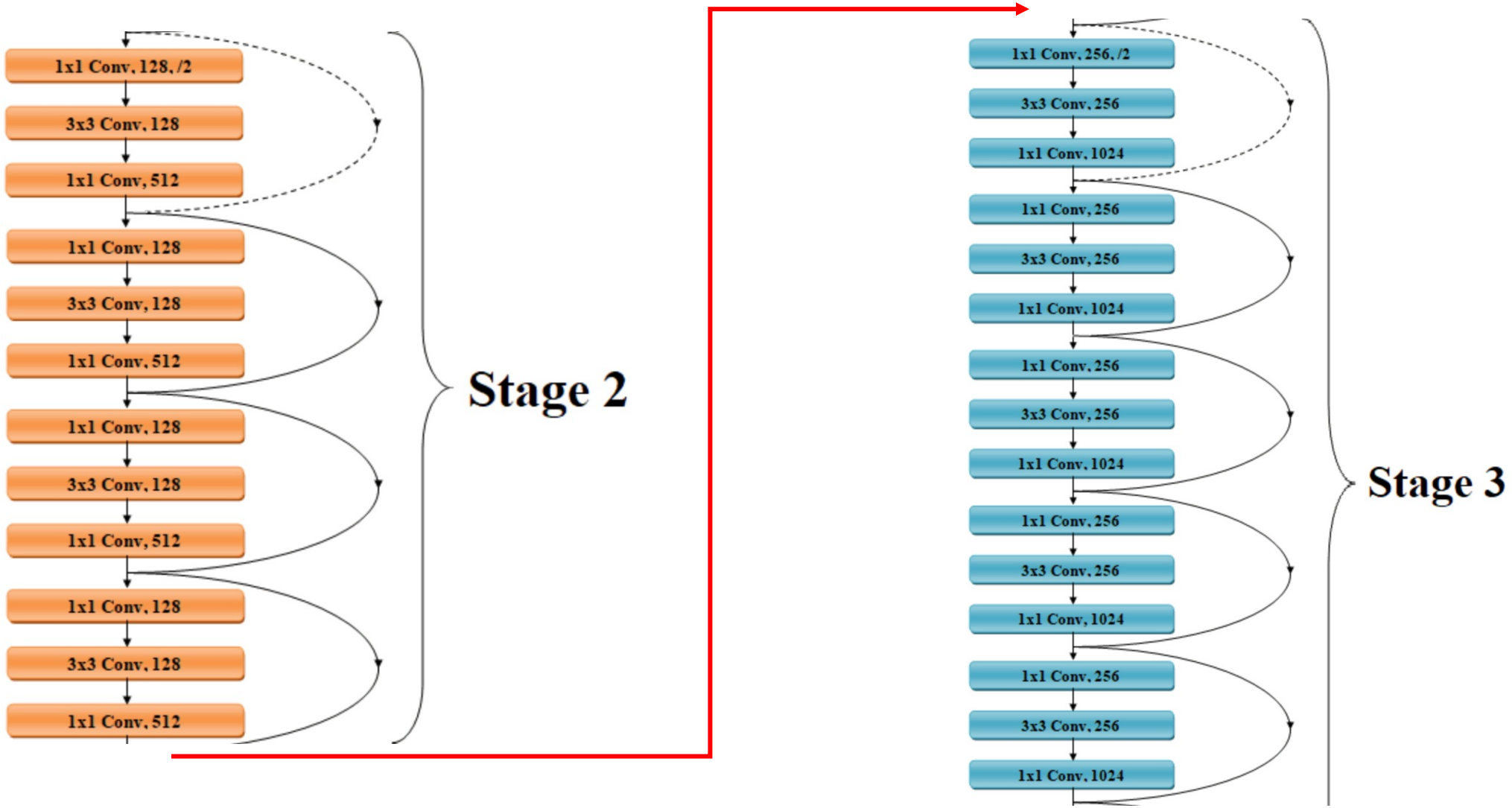
2 output volumes of each thread has the same size and can be added



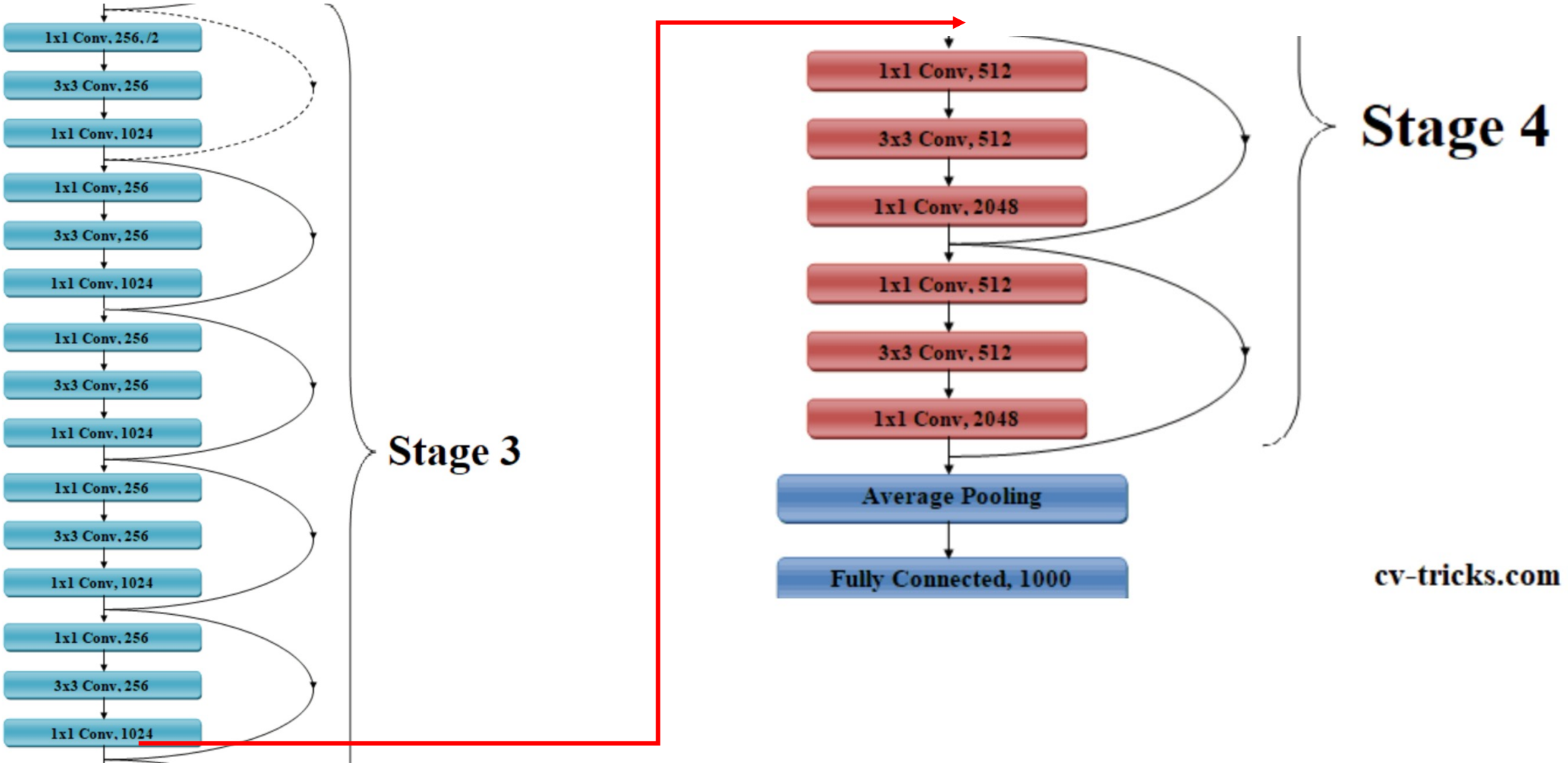
ResNet50 Architecture



ResNet50 Architecture



ResNet50 Architecture



Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

MobileNet

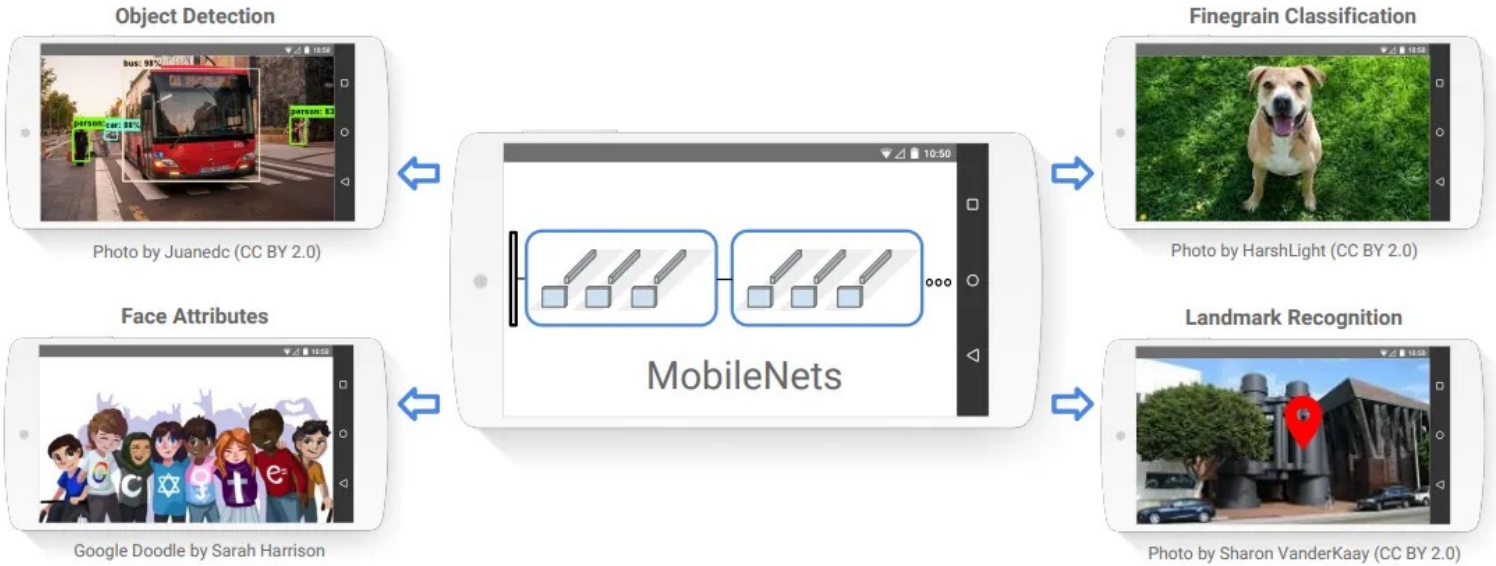
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

In 2017, Google introduced MobileNets, a family of computer vision models based on TensorFlow

The smaller MobileNets have as few as 1.3 million parameters. Also, the size of the models is significantly less. While a VGG16 model can take up to 500 MB of disk space, MobileNet just needs 16–18MB. This makes it ideal to be loaded on mobile devices.

News

1.Using *Depthwise Separable Convolutions*
2.Using two *Shrinking Hyperparameters*:



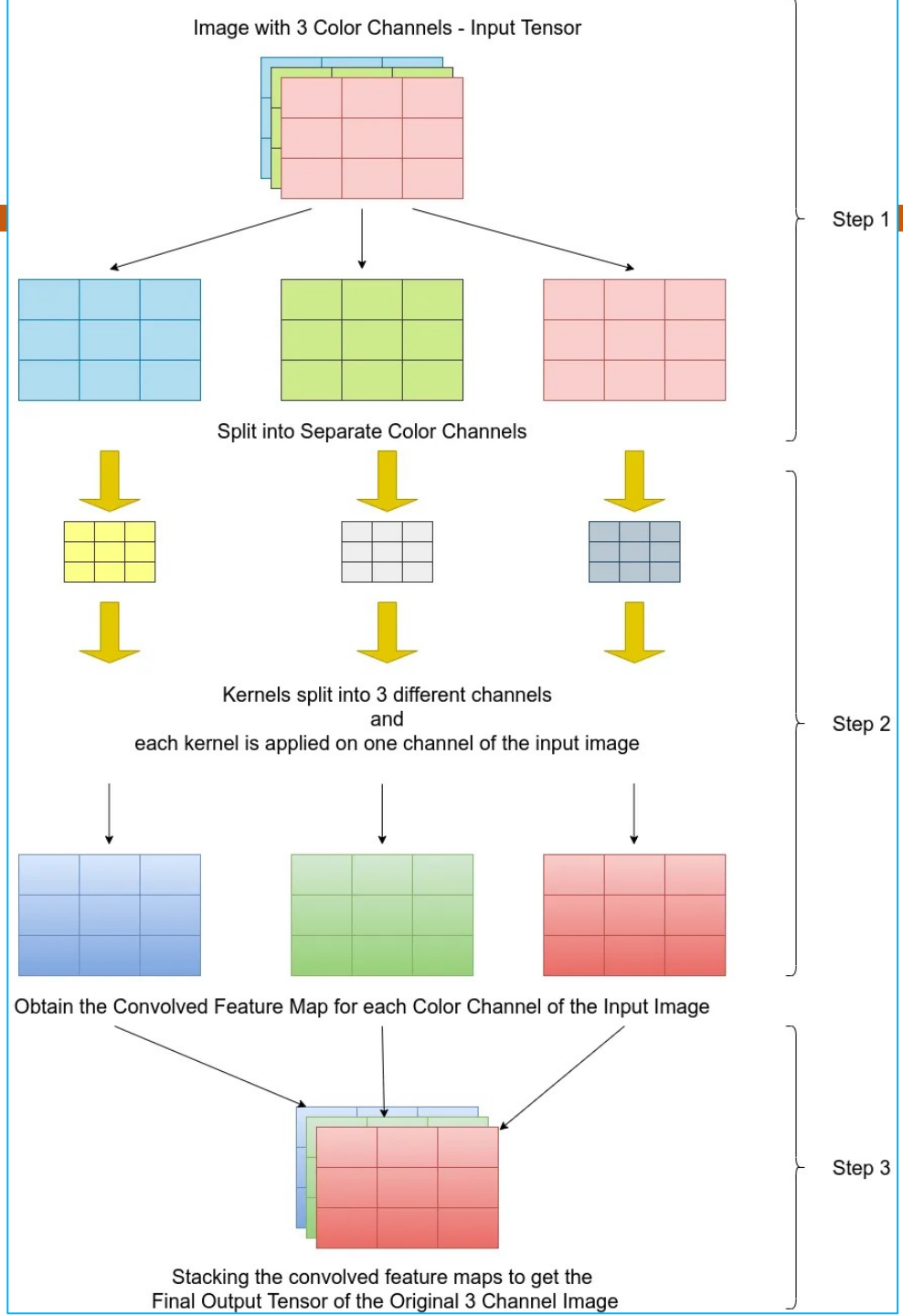
MobileNet

New: Depthwise convolutions

The main difference between 2D convolutions and Depthwise Convolution is that 2D convolutions are performed over all/multiple input channels, whereas in Depthwise convolution, each channel is kept separate.

1. Input tensor of 3 dimensions is split into separate channels
2. For each channel, the input is convolved with a filter (2D)
3. The output of each channel is then stacked together to get the output on the entire 3D tensor

The depth-wise convolutions are used to apply a single filter into each input channel.

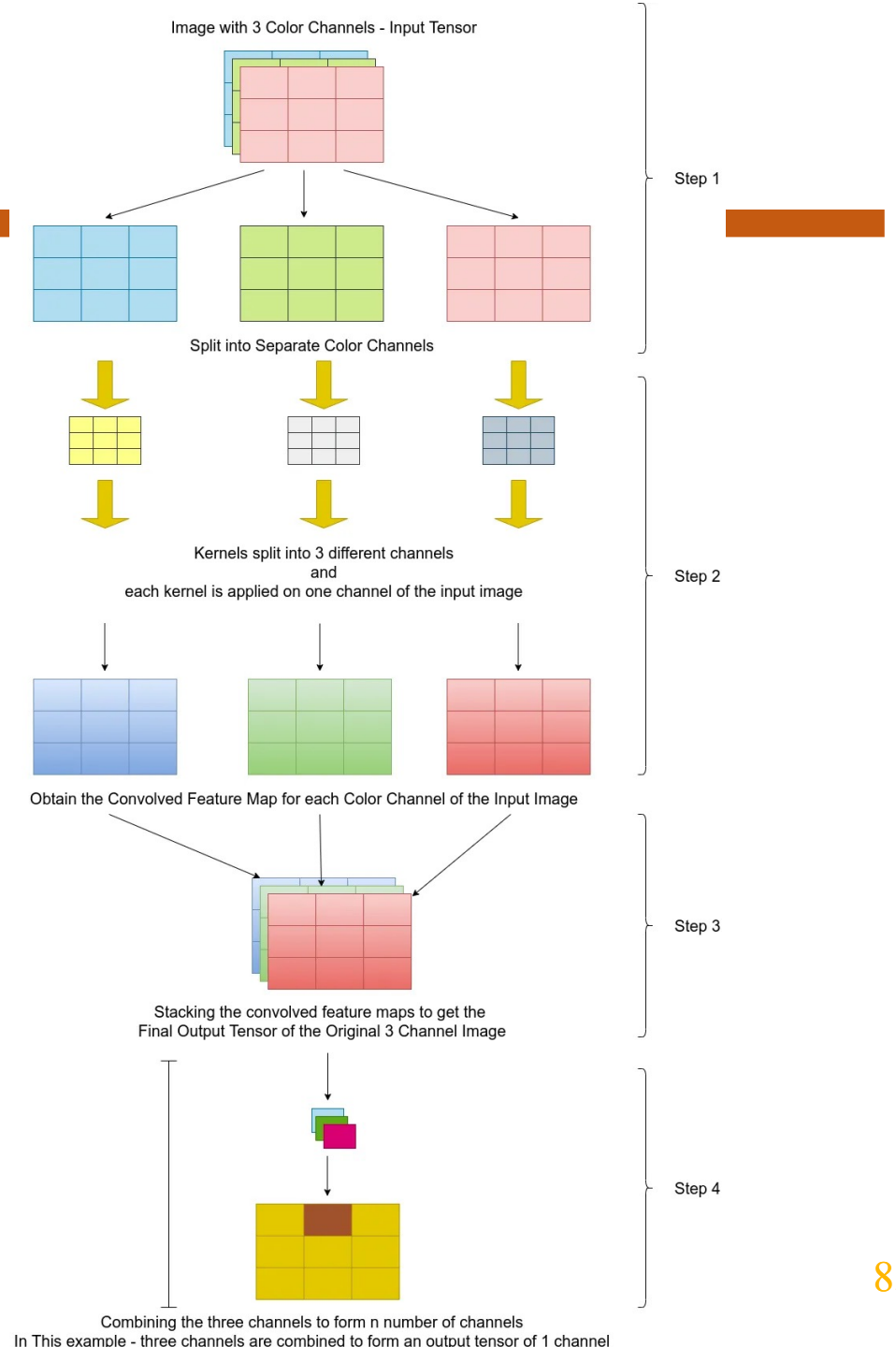


MobileNet

New: Depthwise Separable Convolutions:

Depthwise convolutions are generally applied in combination with another step — Depthwise Separable Convolution. This contains two parts

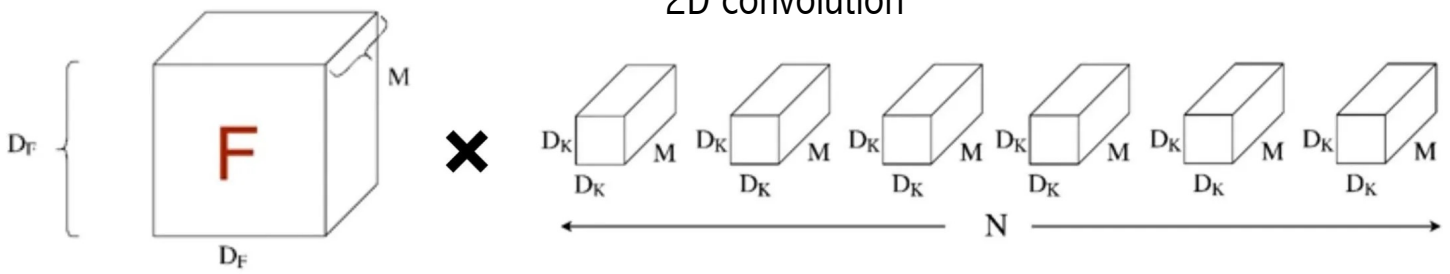
1. Filtering (all the previous steps) and
2. Combining (combining the 3 color channels to form 'n' number of channels, as desired — in the example below we see how the 3 channel can be combined to form a 1 channel output).



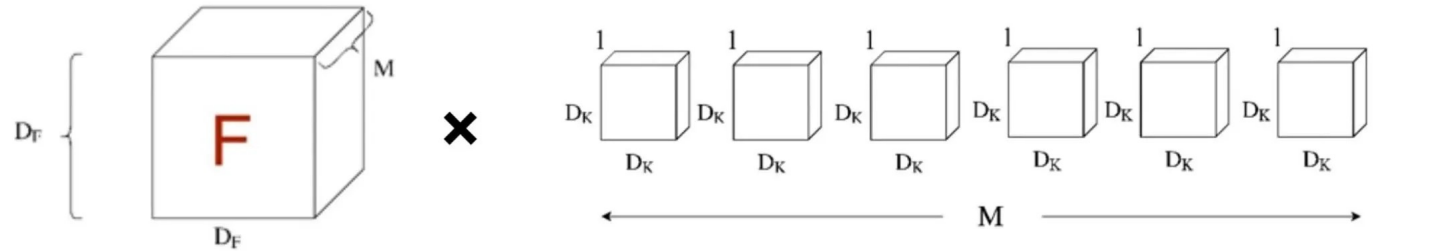
MobileNet

Why is Depthwise Separable Convolution so efficient?

2D convolution



Depthwise convolution cost



Standard Convolution Cost

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

Where DF is the special dimensions of the input feature map and DK is the size of the convolution kernel. Here M and N are the number of input and output channels respectively.

Depth-Wise Convolution cost

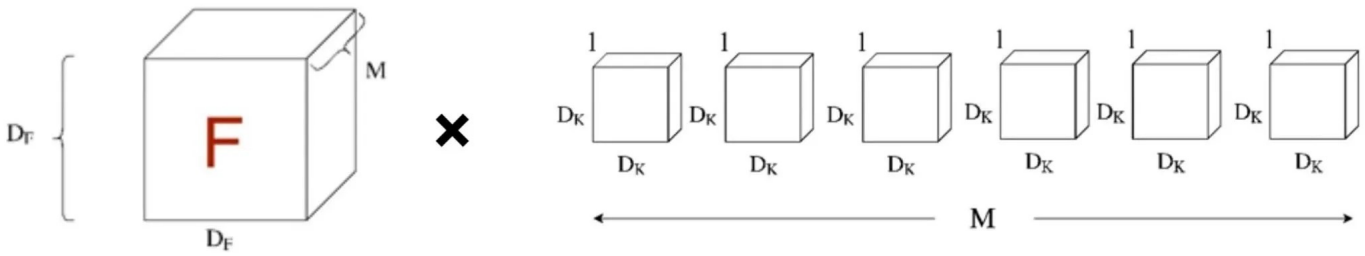
$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$$

Contains an input feature map of dimension DF*DF and M number of kernels of channel size 1.

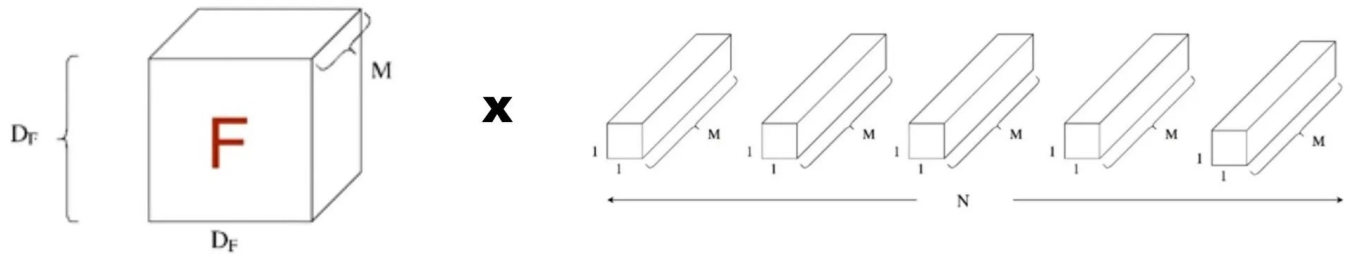
MobileNet

Why is Depthwise Separable Convolution so efficient?

Depthwise convolution cost



Point-wise Convolution



Depthwise separable convolutions cost

Depth-Wise Convolution cost

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$$

+

Pointwise convolution cost

$$M \cdot N \cdot D_F \cdot D_F$$

Since the depthwise convolution is only used to filter the input channel, it does not combine them to produce new features. So an additional layer called pointwise convolution layer is made, which computes a linear combination of the output of depthwise convolution using a 1×1 convolution.

MobileNet

Why is Depthwise Separable Convolution so efficient?

Depth-Wise Convolution cost

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$$

+

Pointwise convolution cost

$$M \cdot N \cdot D_F \cdot D_F$$

Standard Convolution Cost

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

Ratio: Depthwise separable convolutions cost with standard convolution cost

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}$$

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

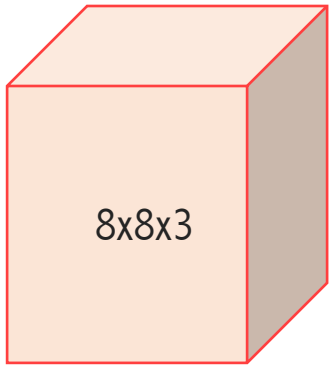
MobileNet

Why is Depthwise Separable Convolution so efficient?

Let's assume that we have an input tensor of size — 8x8x3, And the desired output tensor is of size — 8x8x256

In 2D Convolutions

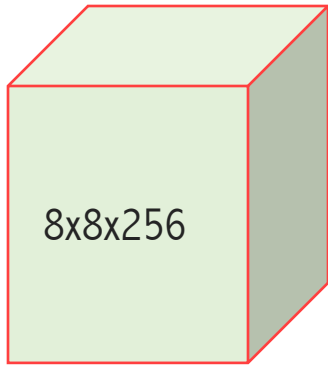
Number of multiplications required —
(8x8) x (5x5x3) x (256) = *1,228,800*



In Depthwise Separable Convolutions

Filtering — Split into single channels, so 5x5x1 filter is required in place of 5x5x3, and since there are three channels, so the total number of 5x5x1 filters required is 3: **(8x8) x (5x5x1) x (3) = 3,800**

Combining — Total number of channels required is 256, so, (8x8) x (1x1x3) x (256) = 49,152
Total number of multiplications = 3,800+49,152 = 53,952



MobileNet

New: *Shrinking Hyperparameters*

Width multiplier which adjusts the number of channels: Thinner Models

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

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

Computational Cost: Depthwise separable convolution with width multiplier

Here α will vary from 0 to 1, with typical values of [1, 0.75, 0.5 and 0.25]. When $\alpha = 1$, called as baseline MobileNet and $\alpha < 1$, called as reduced MobileNet. Width Multiplier has the effect of reducing computational cost by α^2.

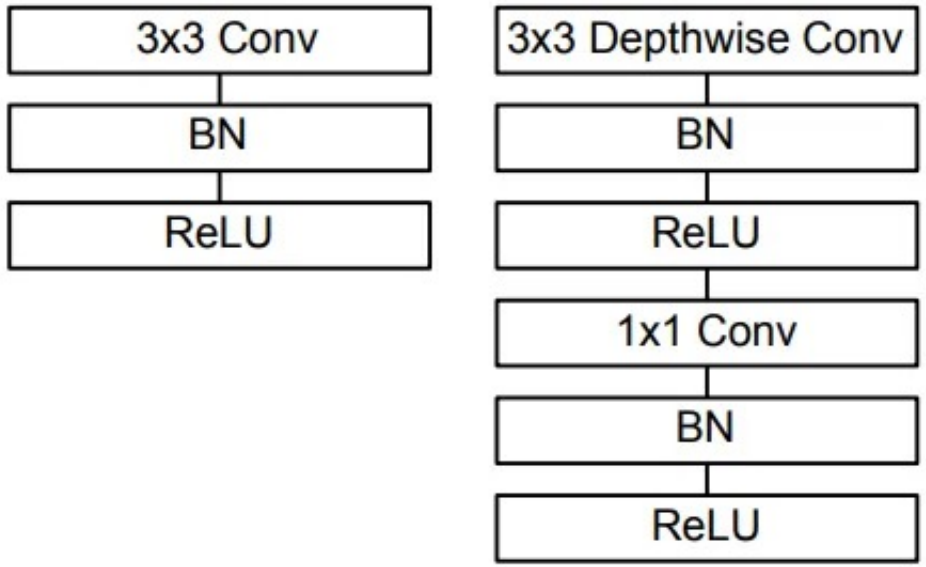
Resolution multiplier which adjusts the spatial dimensions of the feature maps and the input image: Reduced Representation

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

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F$$

Computational cost by applying **width multiplier** and **resolution multiplier**

Mobile Net



Standard Convolutional layer

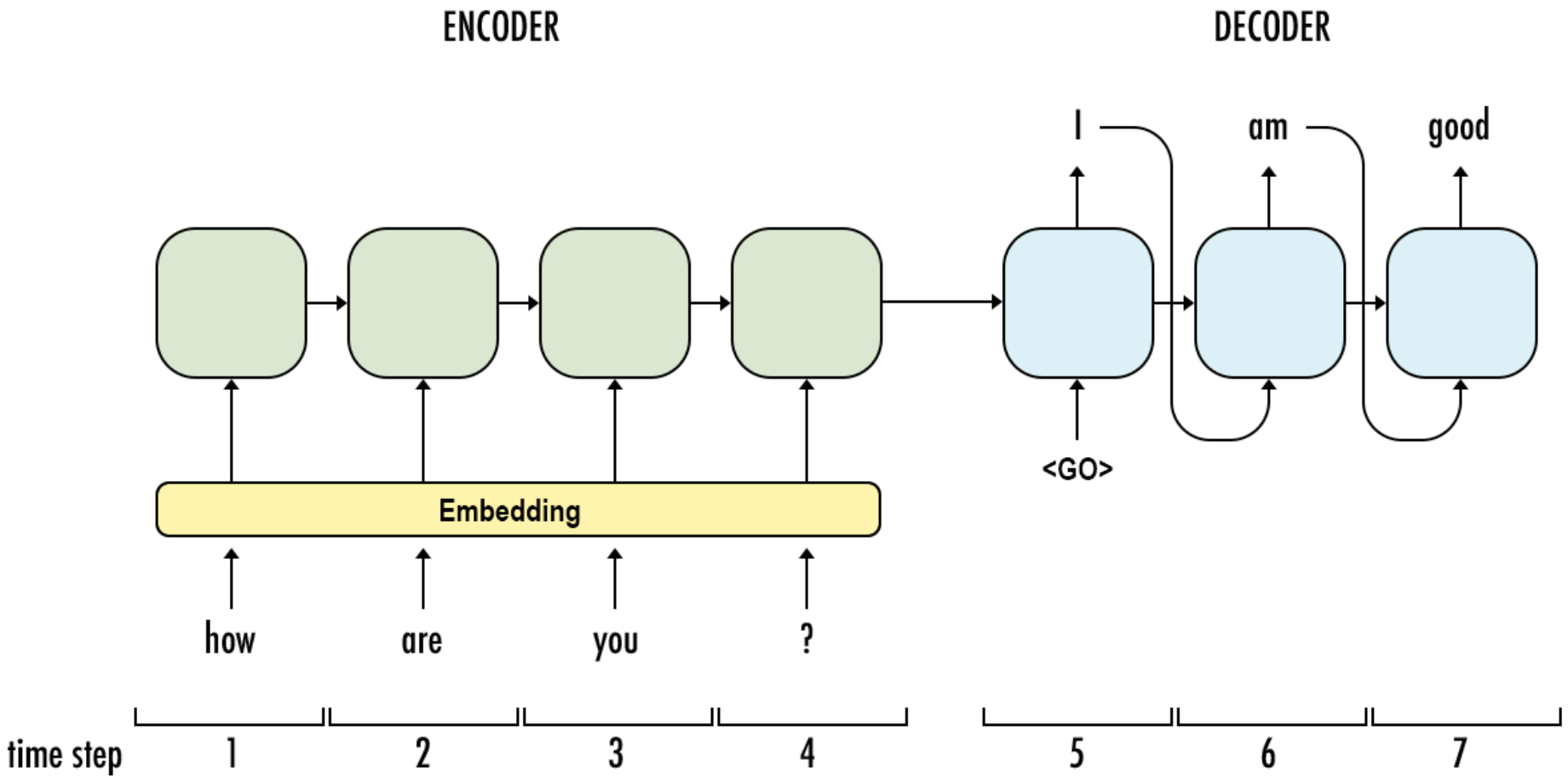
Depthwise Separable Convolutional layers

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	$1 \times 1 \times 1000$

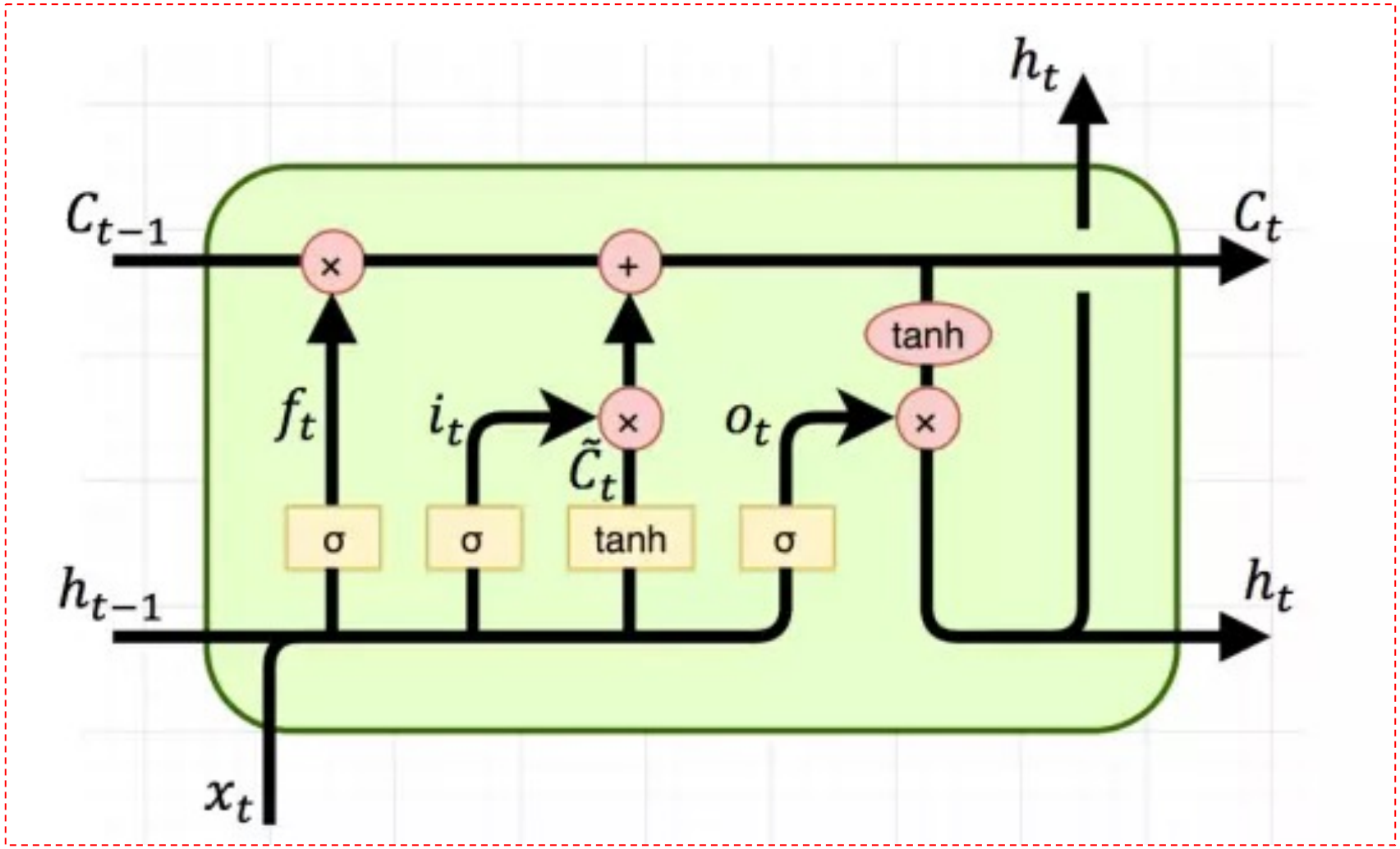
Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

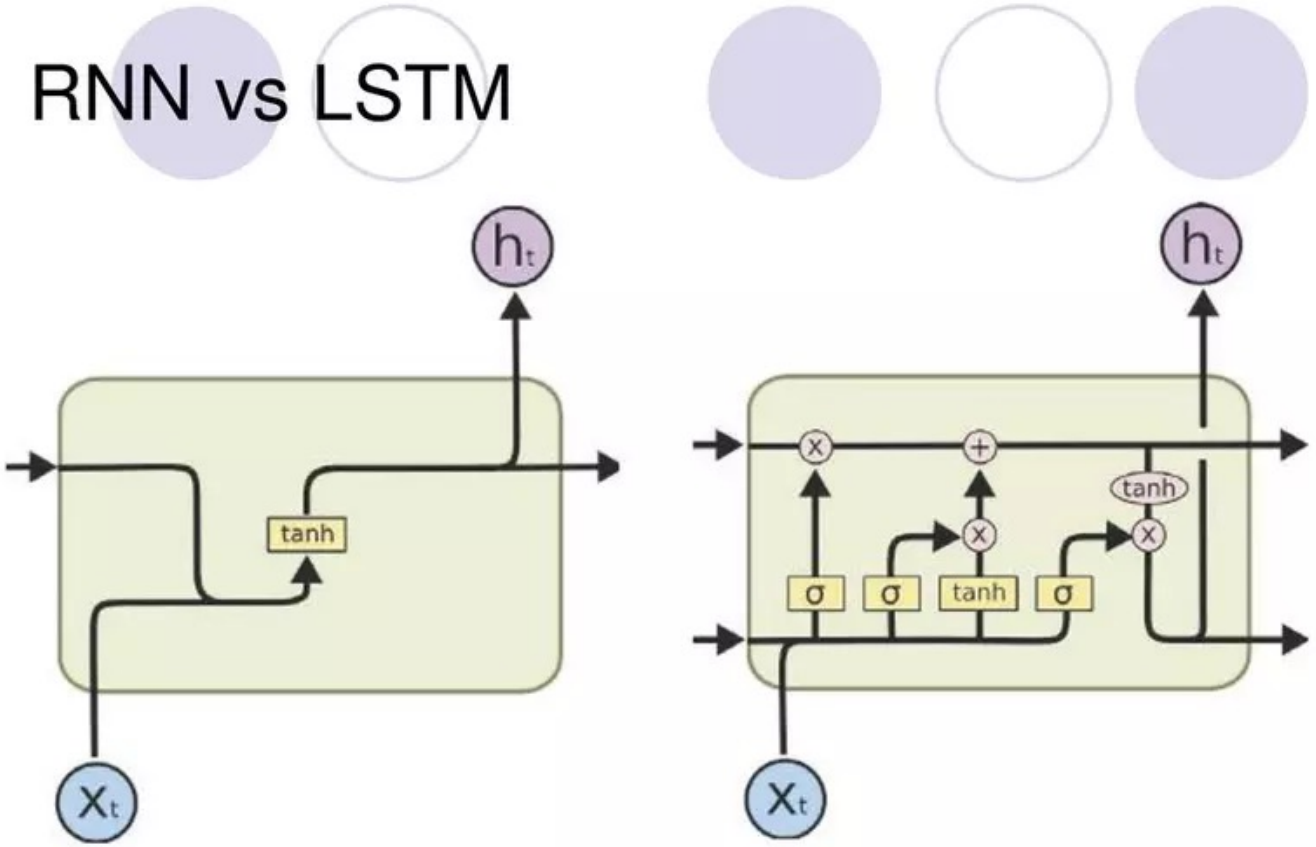
Long-short Term Memory (LSTM)



Sequence-to-Sequence using RNN

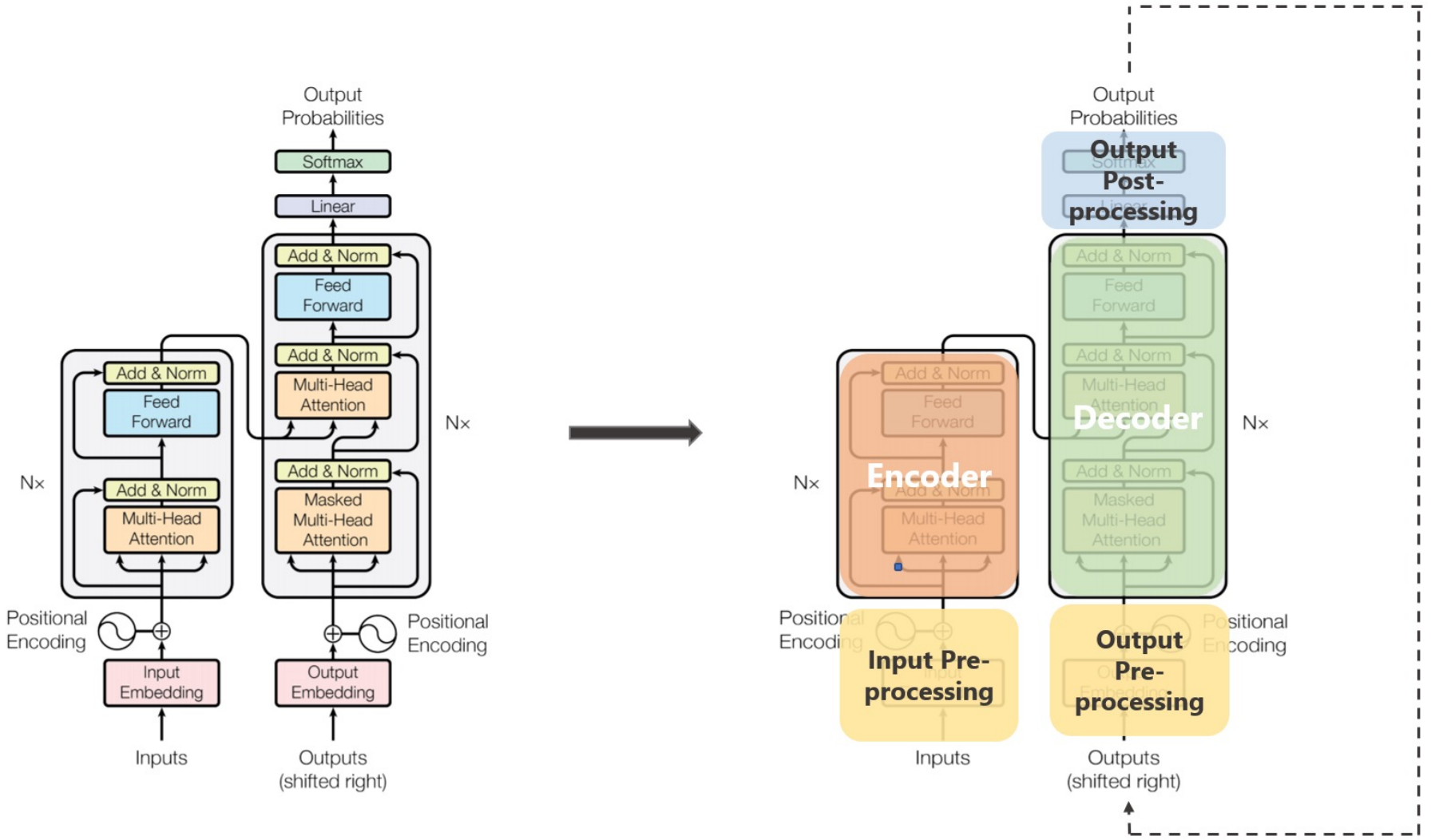


Sequence-to-Sequence using RNN



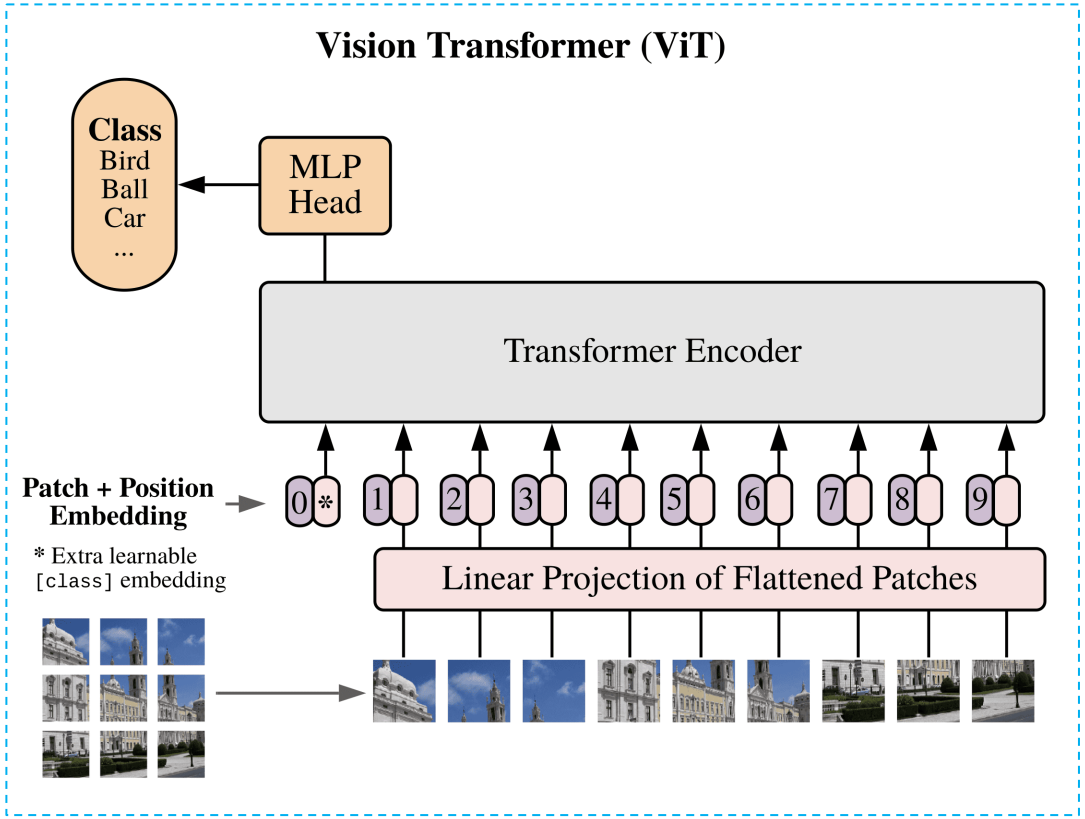
(a) RNN (b) LSTM

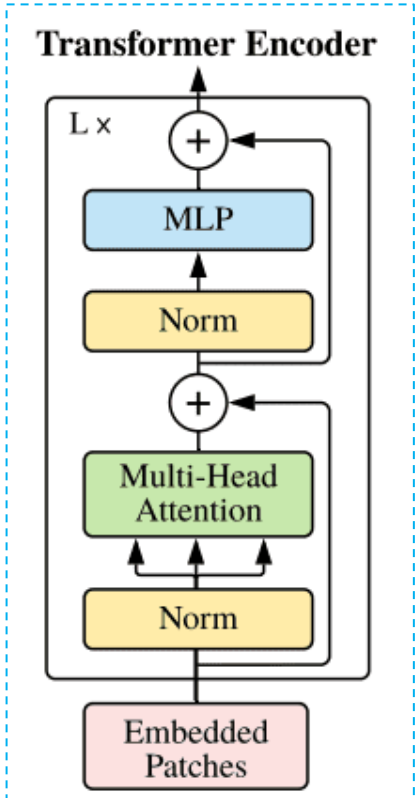
Transformer



ConvNext: The ConvNet for the Roaring 20s

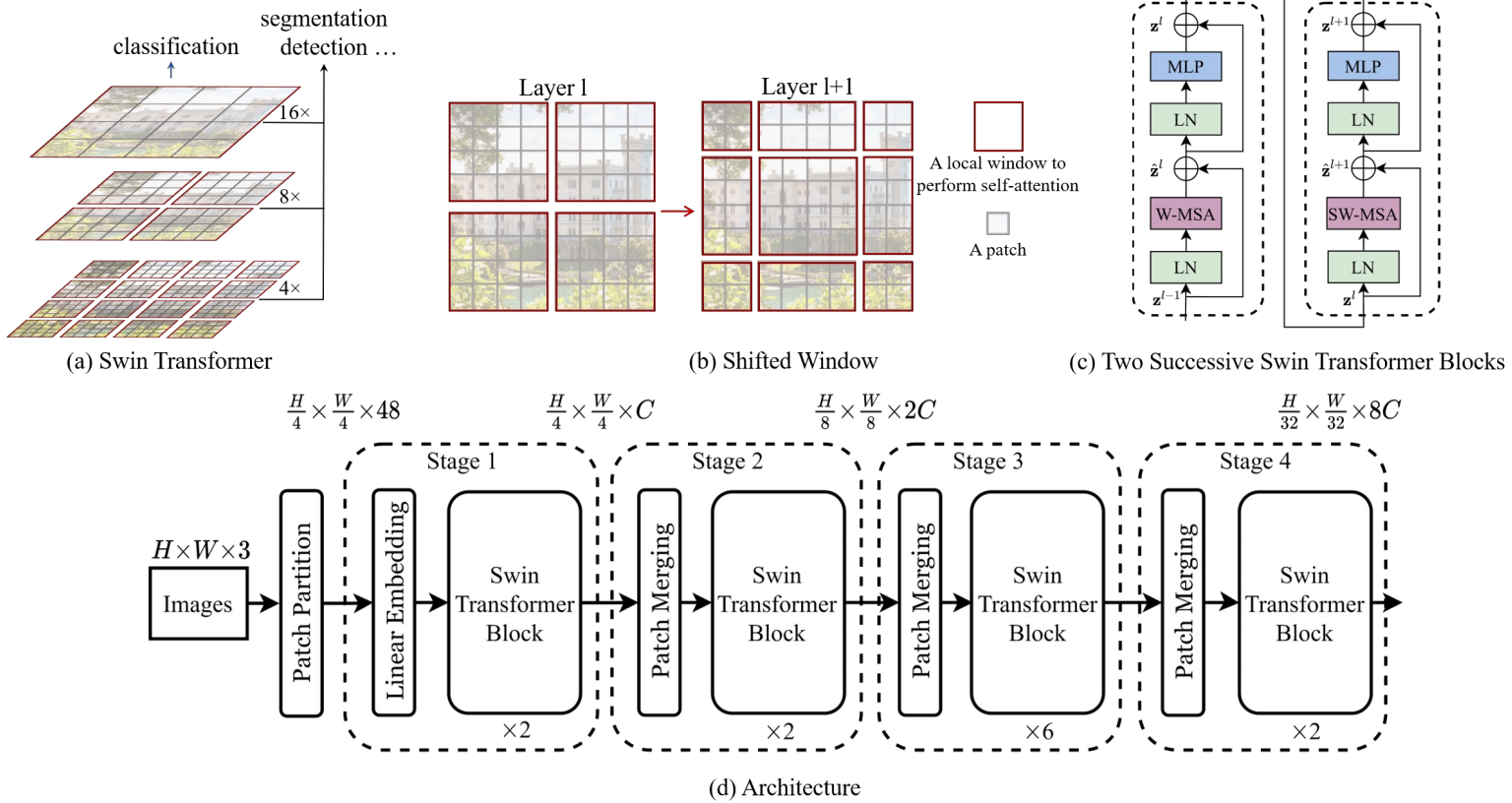
Vision Transformer





In NLP, Transformer models typically represent text as a sequence of words. However, images cannot be represented as a sequence of words. Instead, ViTs represent images as a sequence of patches. **A patch is a small rectangular region of an image. The size of the patch is typically 16x16 pixels.**

Swin Transformer



(a) Swin Transformer

(b) Shifted Window

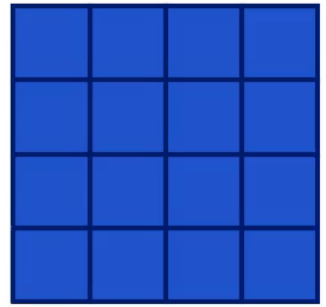
(c) Two Successive Swin Transformer Blocks

(d) Architecture

Patch Merging

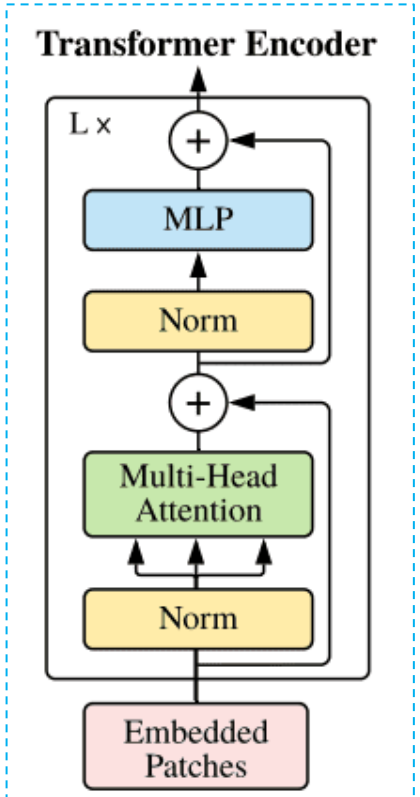
Assuming that n=2, and each group consists of 2x2 neighboring patches

Step 1: Split input image into groups of 2x2
Step 2: In each group, stack the patches depth-wise
Step 3: Combine the stacked groups



ConvNext: The ConvNet for the Roaring 20s

Vision Transformer

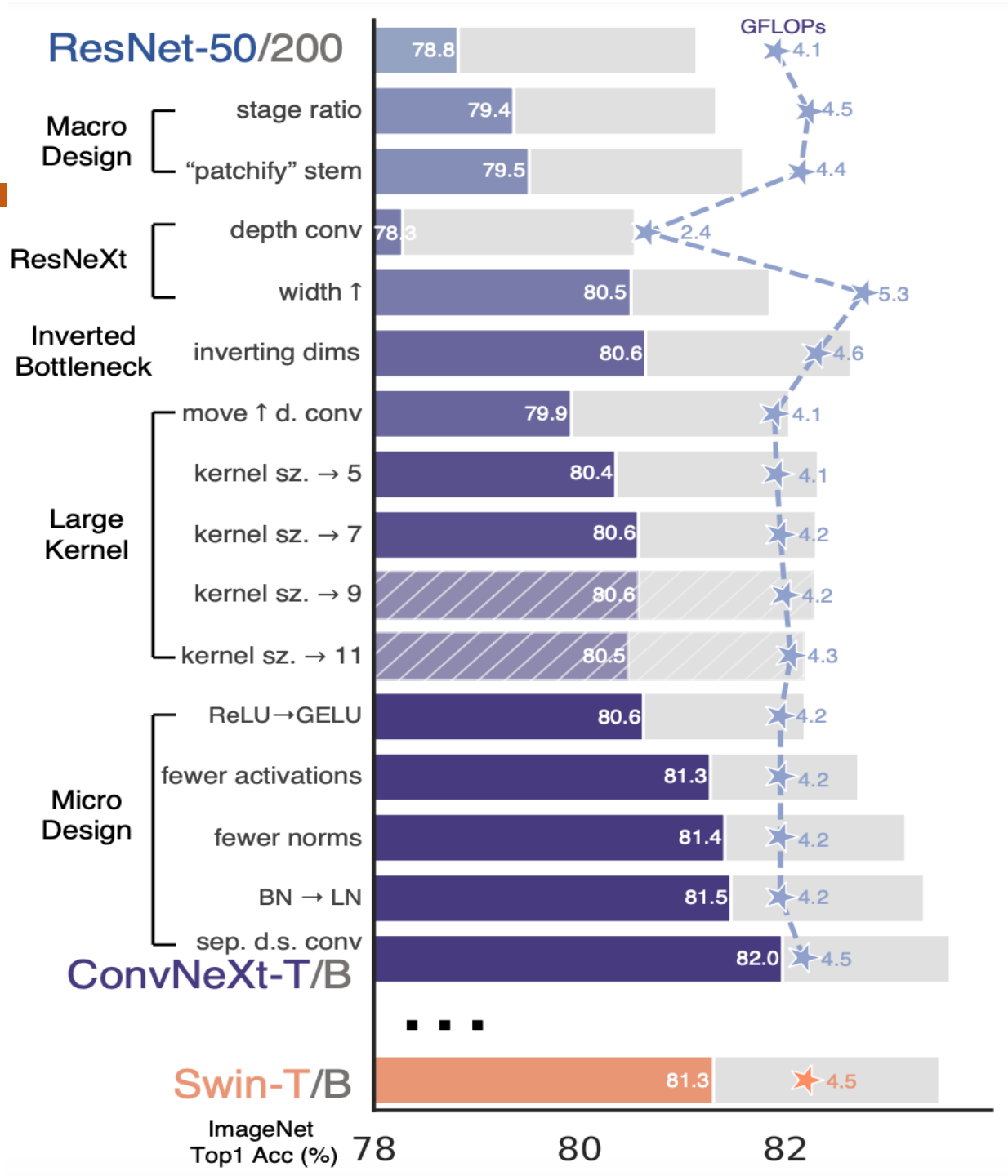


In NLP, Transformer models typically represent text as a sequence of words. However, images cannot be represented as a sequence of words. Instead, ViTs represent images as a sequence of patches. **A patch is a small rectangular region of an image. The size of the patch is typically 16x16 pixels.**

AI VIETNAM
All-in-One Course

ConvNext

Ever since their invention in 2017, transformers have been "the next big thing" in natural language processing (NLP). In recent years, vision transformers have joined their NLP counterparts and are now regularly used to solve computer vision issues. But does that mean that classic convolutional neural networks (CNN) are dead? Not yet. The authors of ConvNeXt did their best to revive ResNet, a popular CNN, by tweaking it to perform better and to be more similar to a transformer, without actually turning it into a transformer or adding attention into the mix.

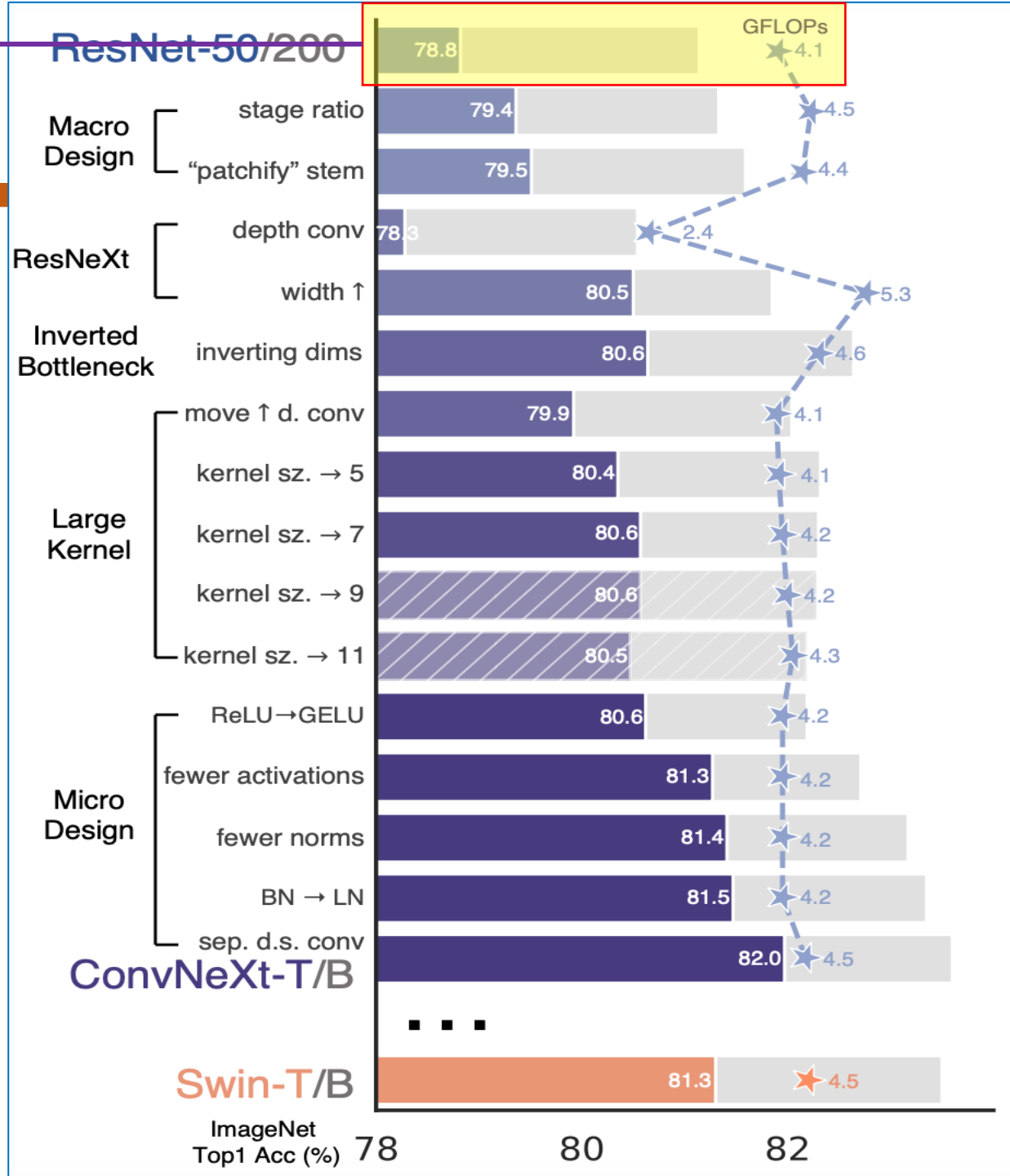


AI VIETNAM
All-in-One Course

ConvNext

Original, the perfomrance ResNet-50 is 76.1% on Image Net.
Improving the performance off ResNet-50 by:
- Using AdamW optimizer
- Extend epoches from 90 to 300
- Data augmentation: Mixup, cutmix, RandAument, Random Erasing
- Regularization schemes: Stochatis Depth, Label smoothing

Accuracy increased by + 2.7%

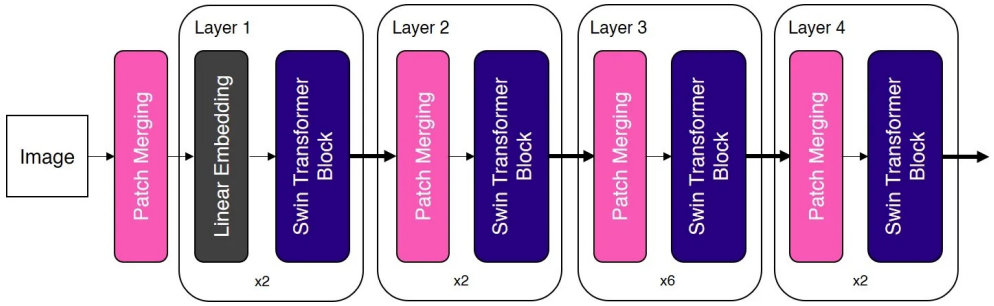


AI VIETNAM
All-in-One Course

ConvNext

Changing stage compute ratio

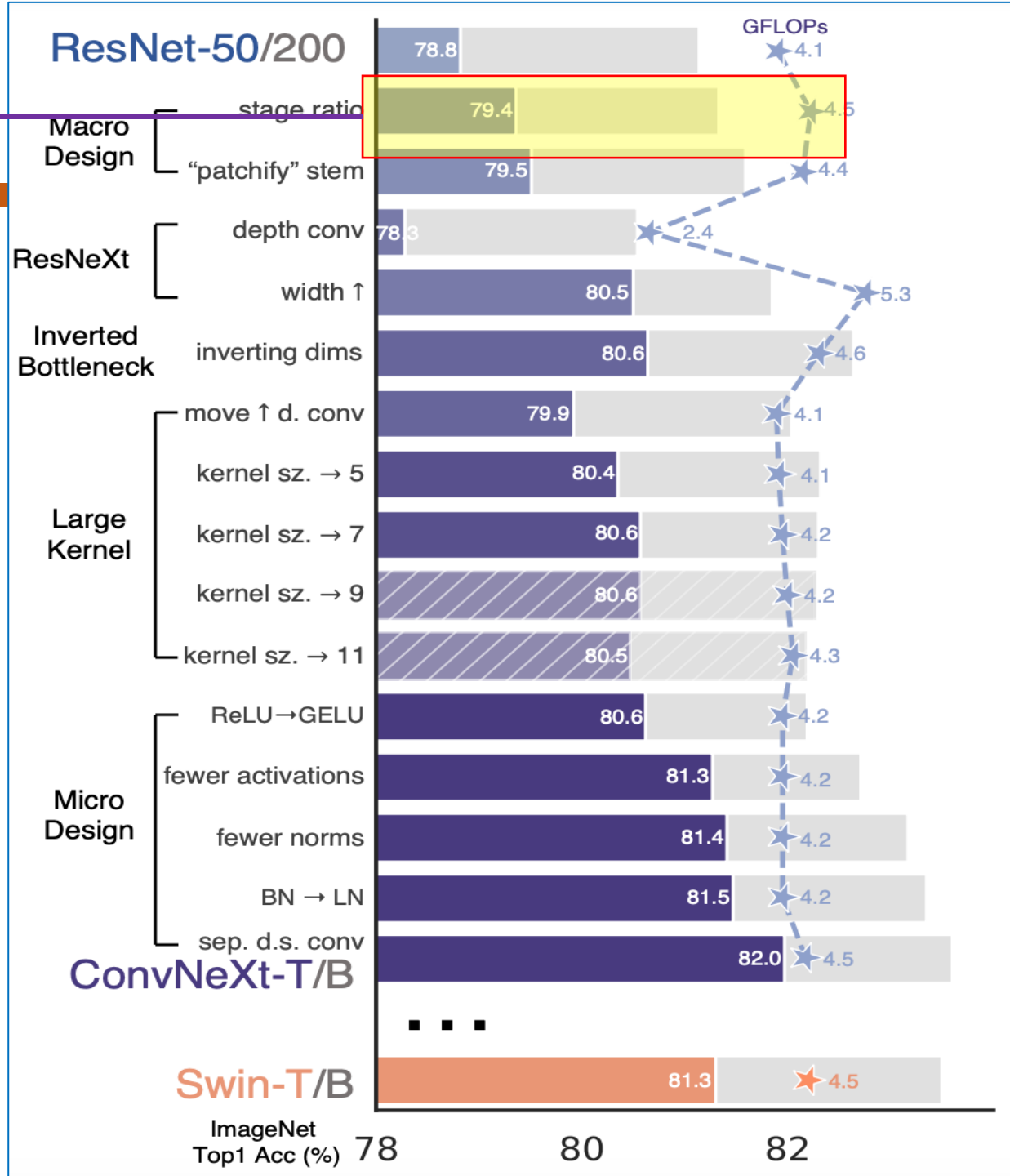
Swin tranformer



Compute ratio: 1:1:3:1

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	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

ResNet50 ratio: 3:4:6:3 => 3:3:9:3



AI VIETNAM
All-in-One Course

ConvNext

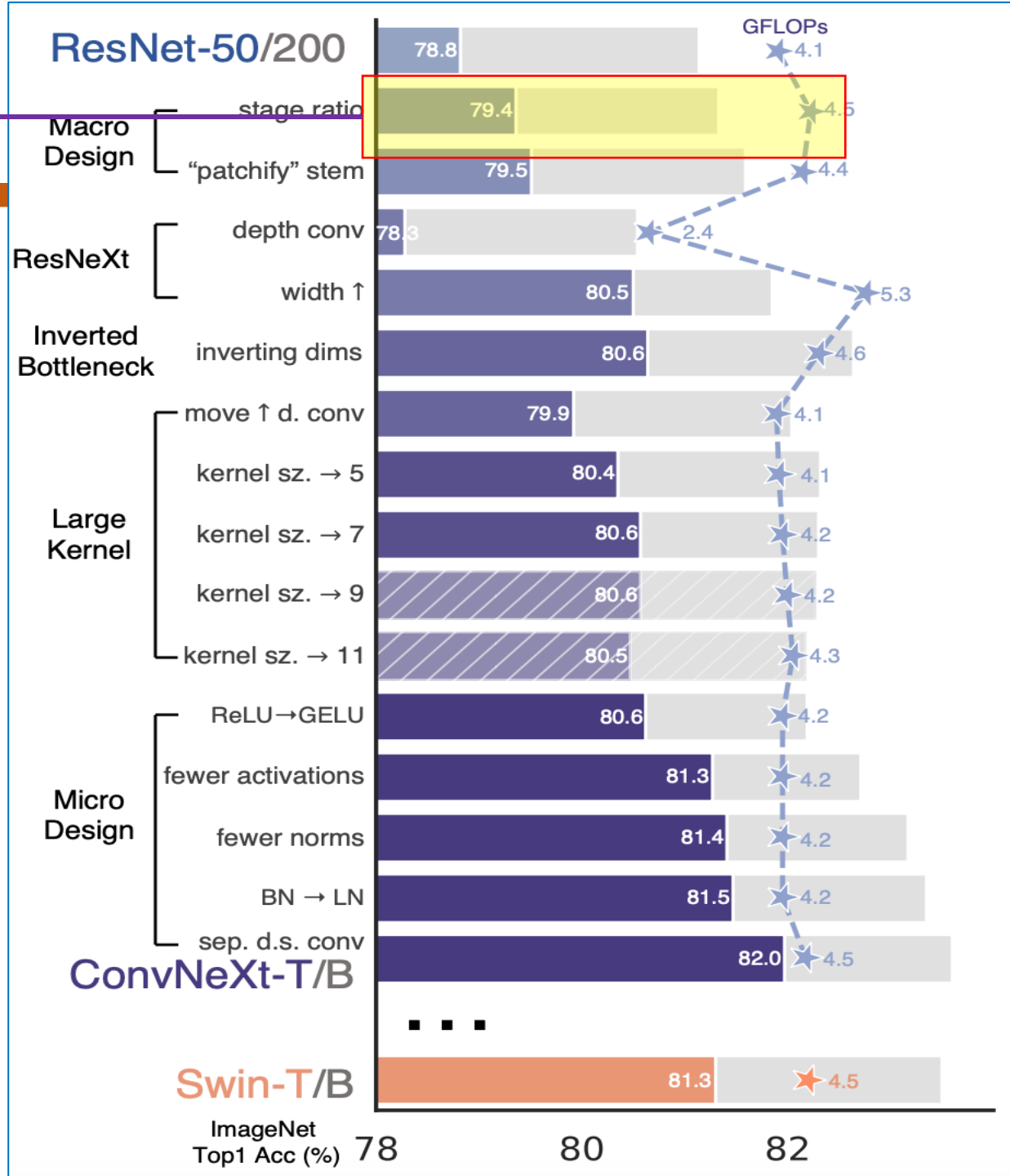
	downsp. rate (output size)	Swin-T	Swin-S	Swin-B	Swin-L
stage 1	4× (56×56)	concat 4×4, 96-d, LN	concat 4×4, 96-d, LN	concat 4×4, 128-d, LN	concat 4×4, 192-d, LN
		[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 128, head 4] × 2	[win. sz. 7×7, dim 192, head 6] × 2
stage 2	8× (28×28)	concat 2×2, 192-d , LN	concat 2×2, 192-d , LN	concat 2×2, 256-d , LN	concat 2×2, 384-d , LN
		[win. sz. 7×7, dim 192, head 6] × 2	[win. sz. 7×7, dim 192, head 6] × 2	[win. sz. 7×7, dim 256, head 8] × 2	[win. sz. 7×7, dim 384, head 12] × 2
stage 3	16× (14×14)	concat 2×2, 384-d , LN	concat 2×2, 384-d , LN	concat 2×2, 512-d , LN	concat 2×2, 768-d , LN
		[win. sz. 7×7, dim 384, head 12] × 6	[win. sz. 7×7, dim 384, head 12] × 18	[win. sz. 7×7, dim 512, head 16] × 18	[win. sz. 7×7, dim 768, head 24] × 18
stage 4	32× (7×7)	concat 2×2, 768-d , LN	concat 2×2, 768-d , LN	concat 2×2, 1024-d , LN	concat 2×2, 1536-d , LN
		[win. sz. 7×7, dim 768, head 24] × 2	[win. sz. 7×7, dim 768, head 24] × 2	[win. sz. 7×7, dim 1024, head 32] × 2	[win. sz. 7×7, dim 1536, head 48] × 2

Swin tranformer *Compute ratio: 1:1:3:1*

Changing stage compute ratio

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	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

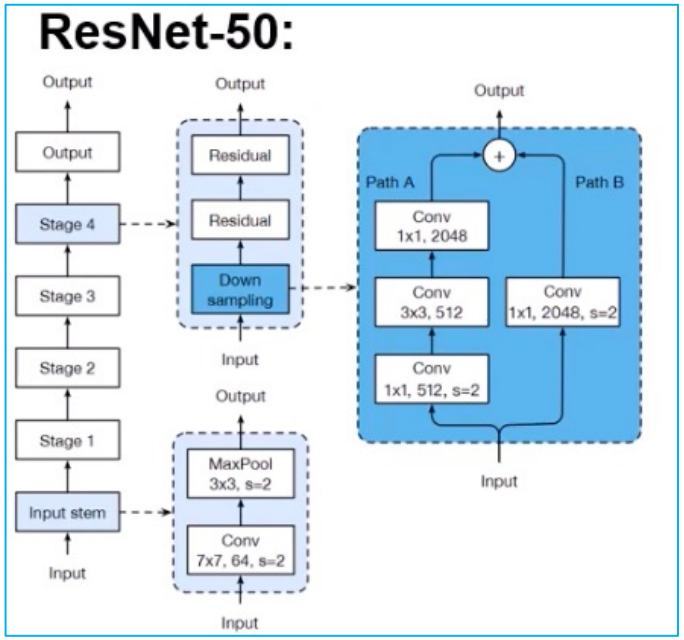
ResNet50 ratio: 3:4:6:3 => 3:3:9:3



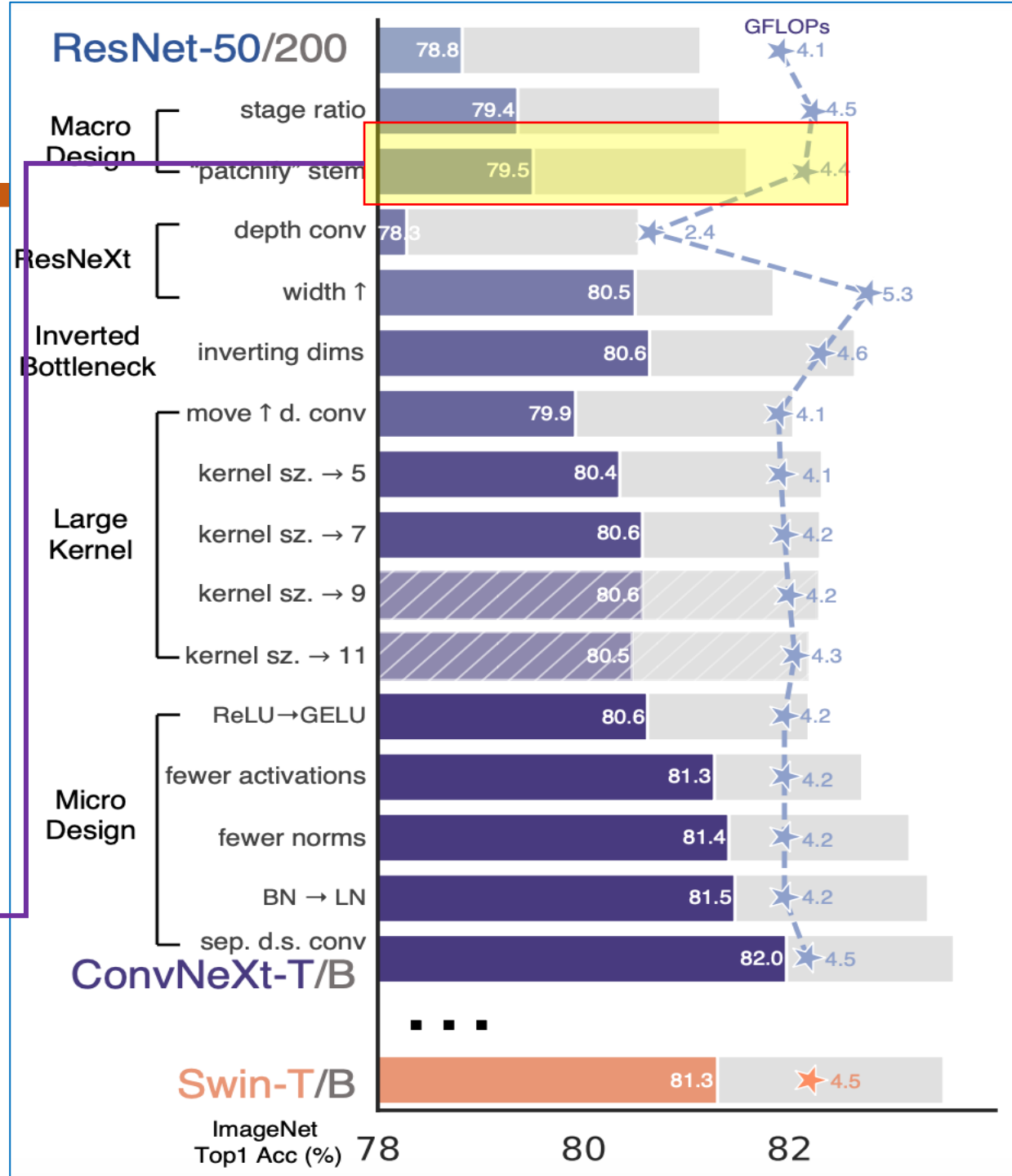
AI VIETNAM
All-in-One Course

ConvNext

Changing stem to "Patchify"



- 7x7 conv stride 2 + MaxPool => 4x downsampling
- Swin Transformer uses 4x4 patch size instead
- Using 4x4 conv with stride of 4 on stem layer



ResNet50 ratio: 3:4:6:3 => 3:3:9:3

AI VIETNAM
All-in-One Course

ConvNext

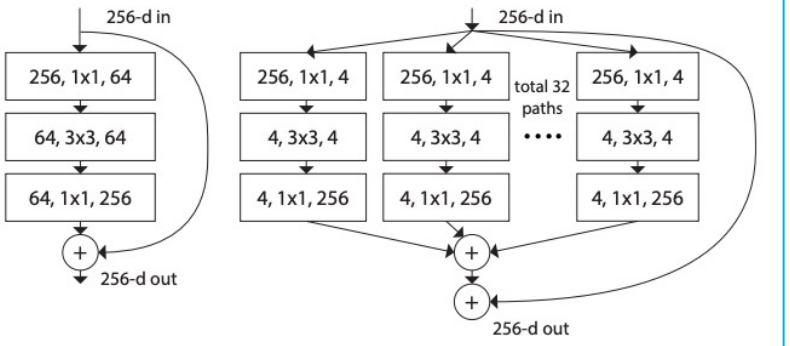
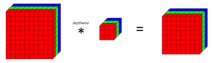


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

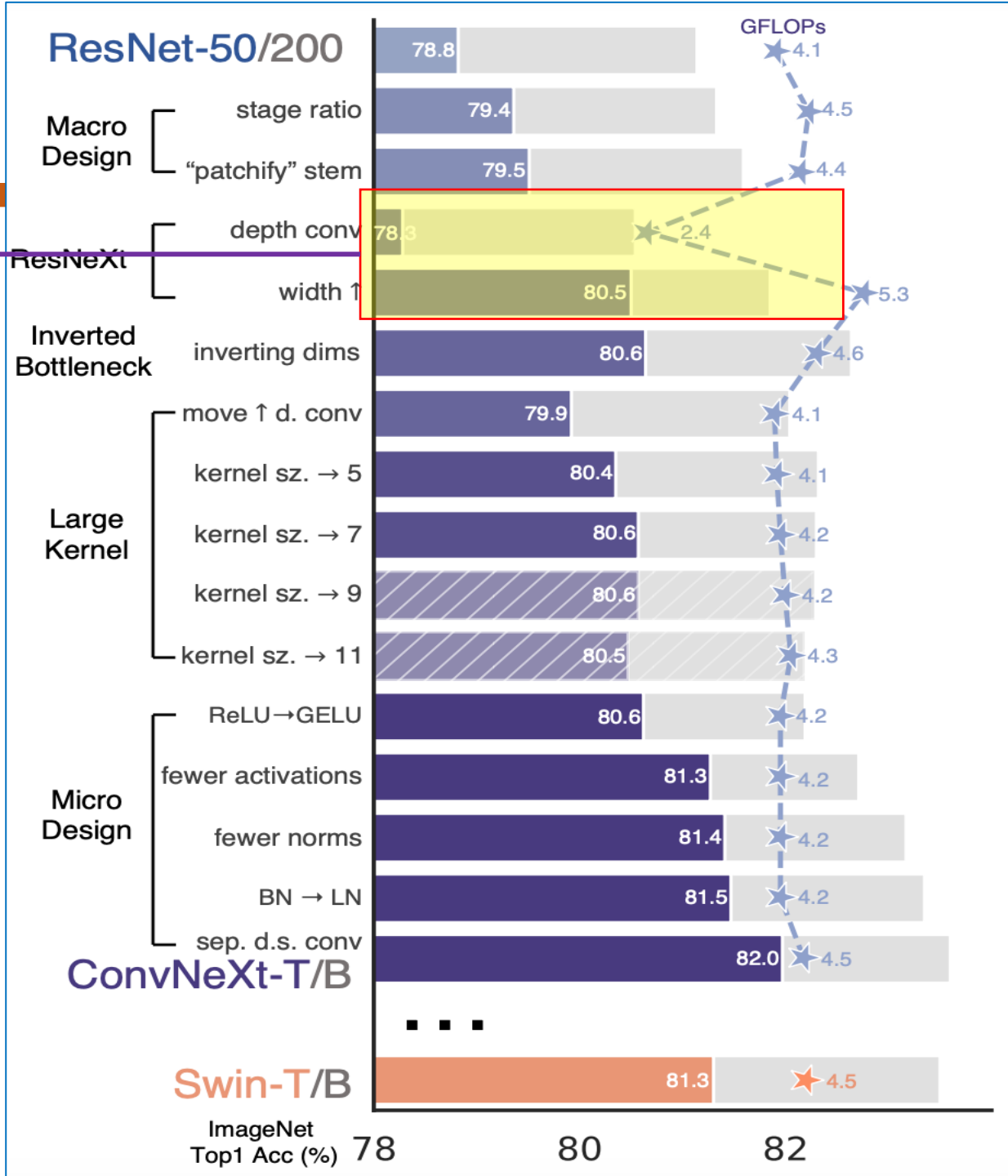
stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2 [1×1, 64 3×3, 64 1×1, 256] ×3	3×3 max pool, stride 2 [1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 256 3×3, 256, C=32 1×1, 512] ×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**×10^6	**25.0**×10^6
FLOPs		**4.1**×10^9	**4.2**×10^9

ResNeXt-ify

Depthwise convolution is a special case of grouped convolution

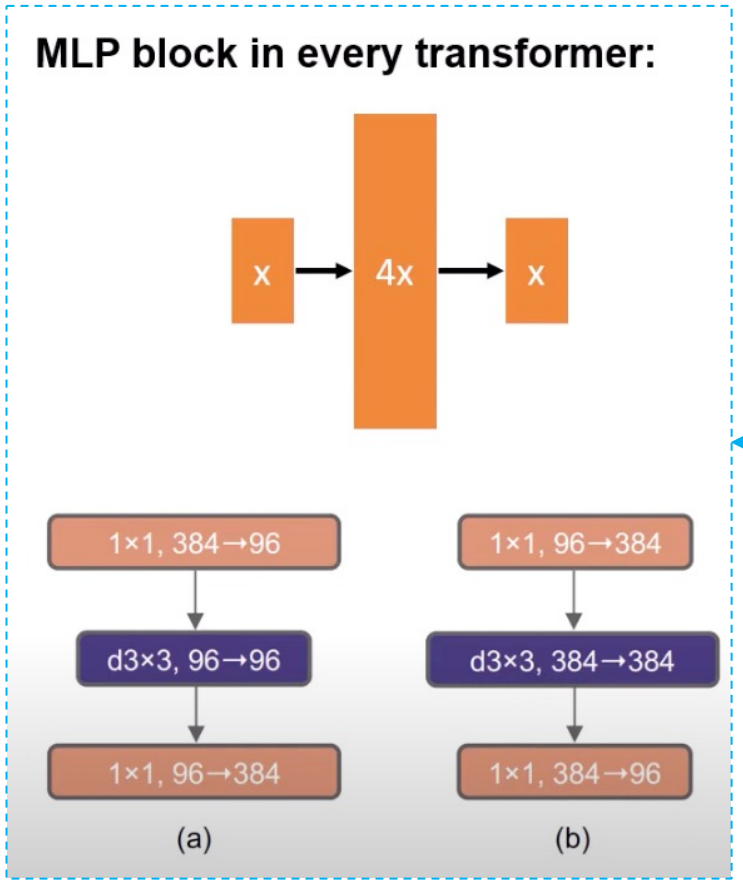


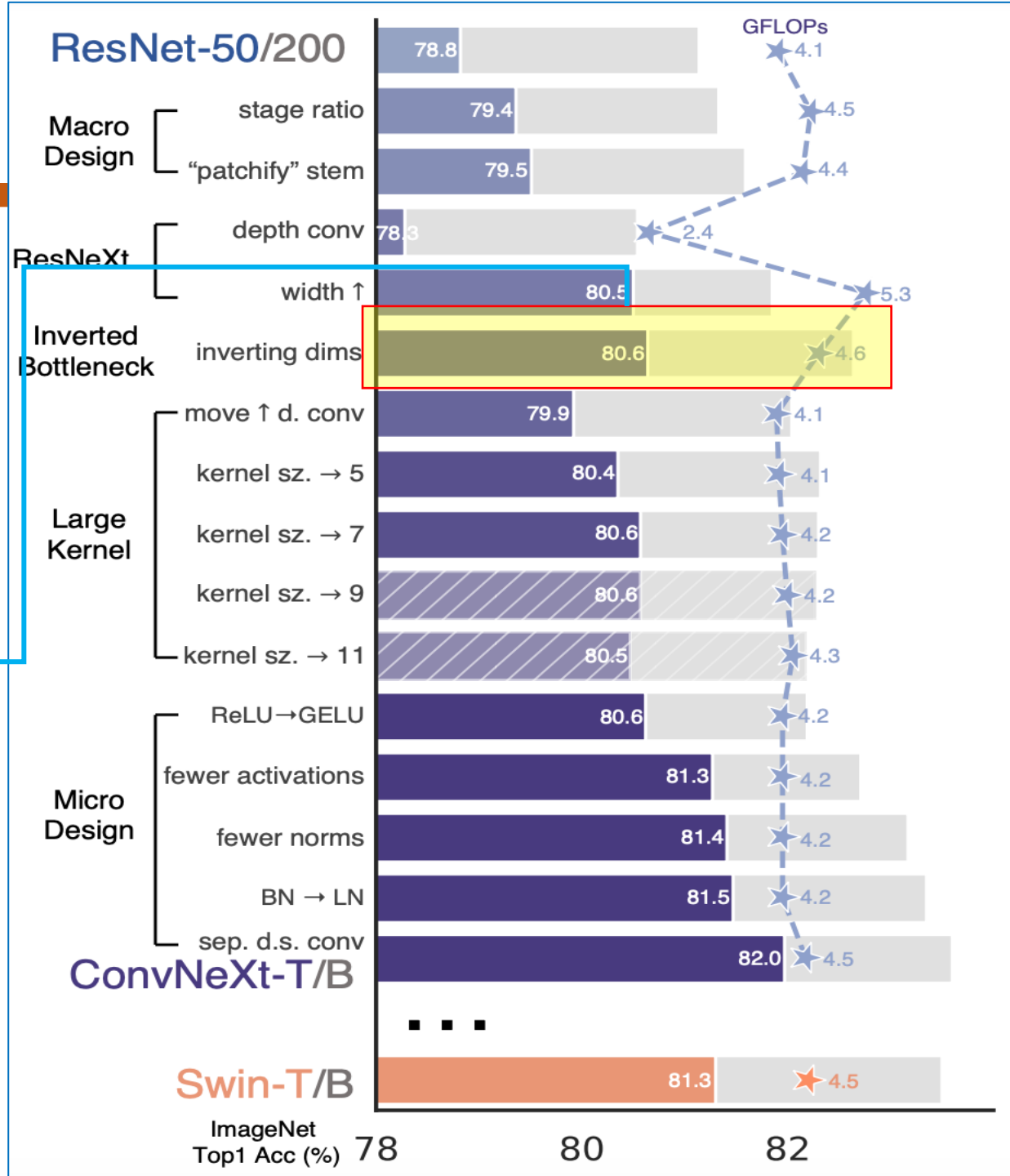
Width increased from 64 to 96



ConvNext

Inverted Bottle Neck

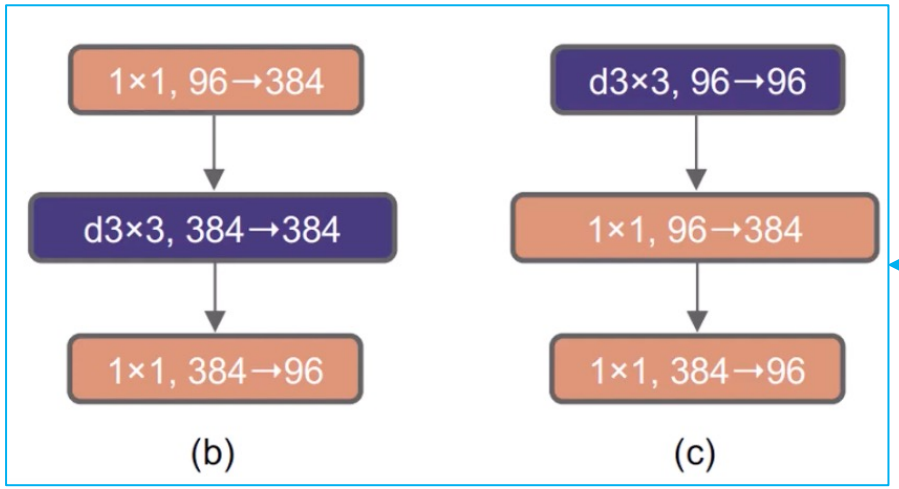


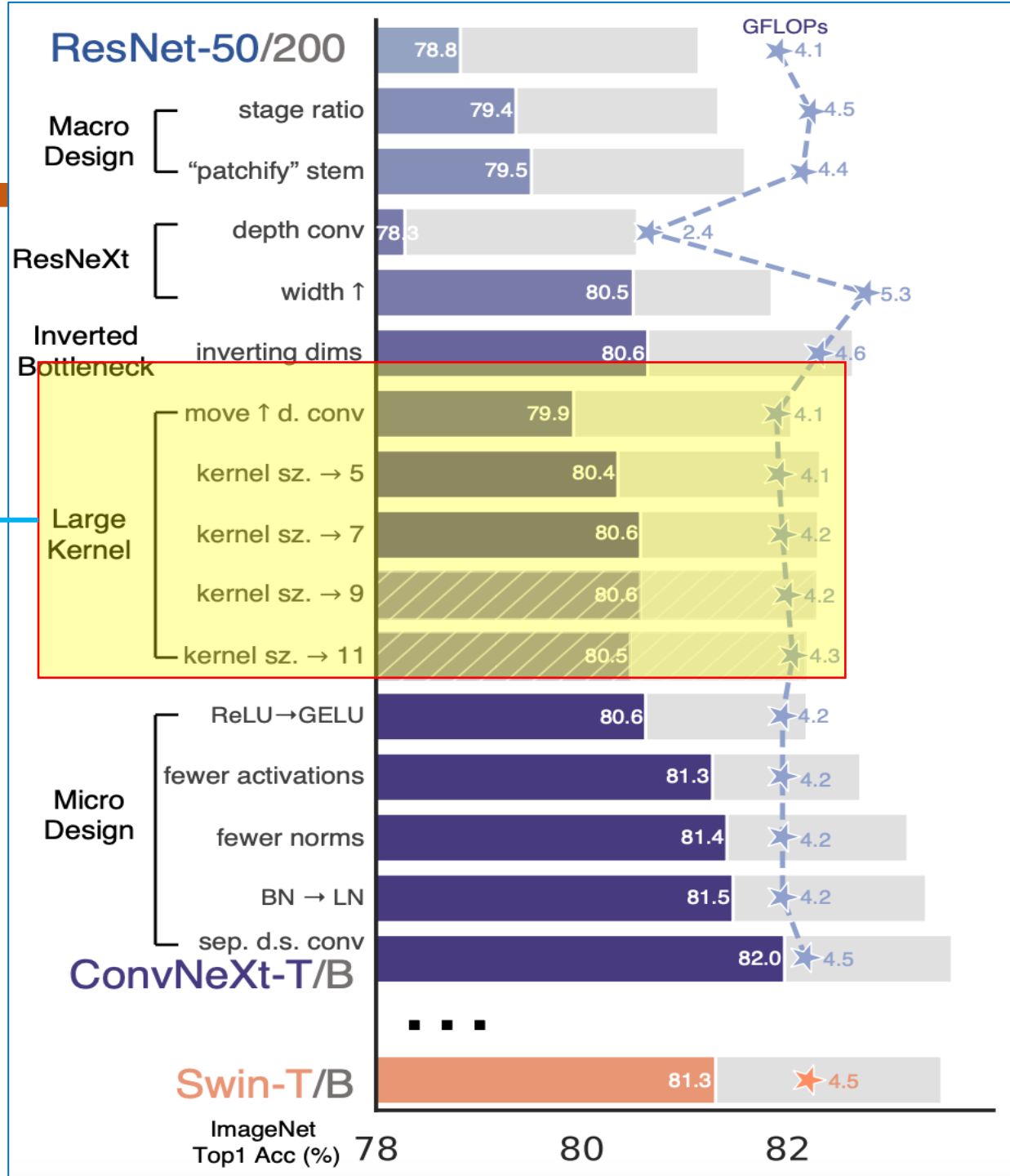


AI VIETNAM
All-in-One Course

ConvNext

Large Kernel Size



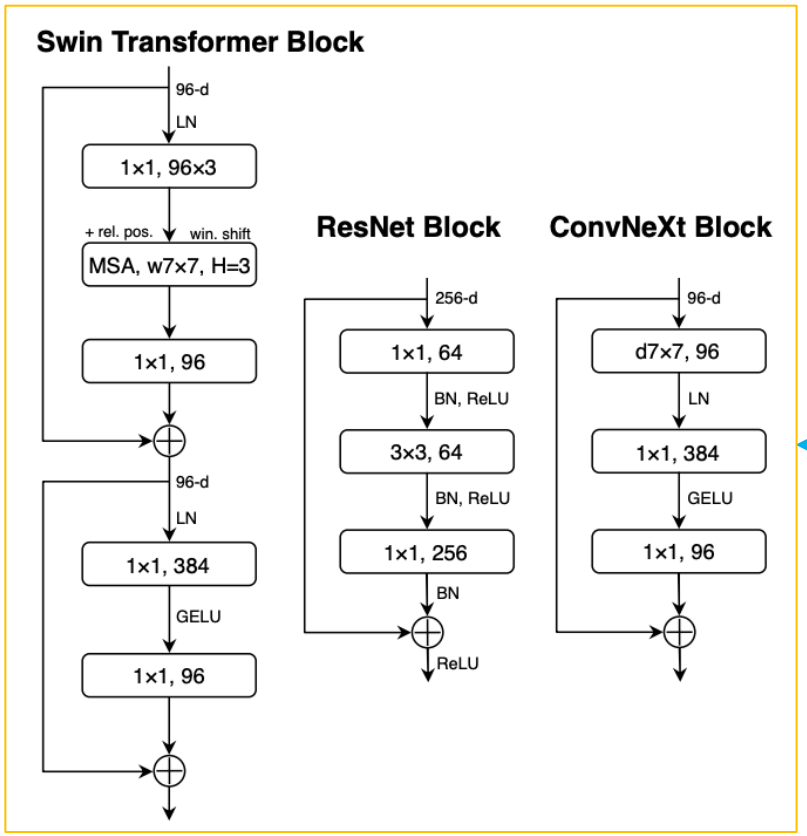


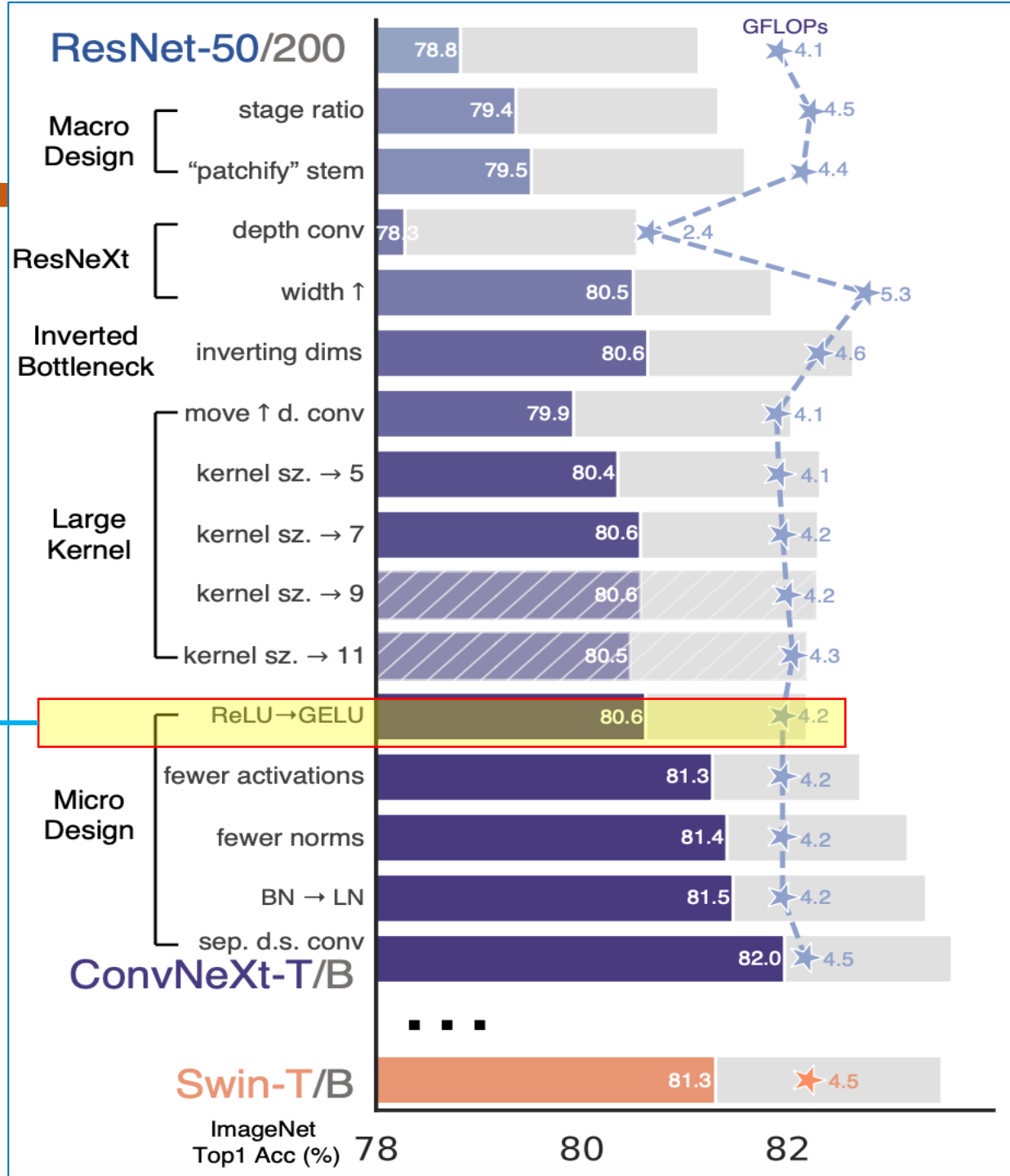
AI VIETNAM
All-in-One Course

ConvNext

Mirco Design

Replacing ReLU with GELU





ResNet50 ratio: 3:4:6:3 => 3:3:9:3

AI VIETNAM
All-in-One Course

ConvNext

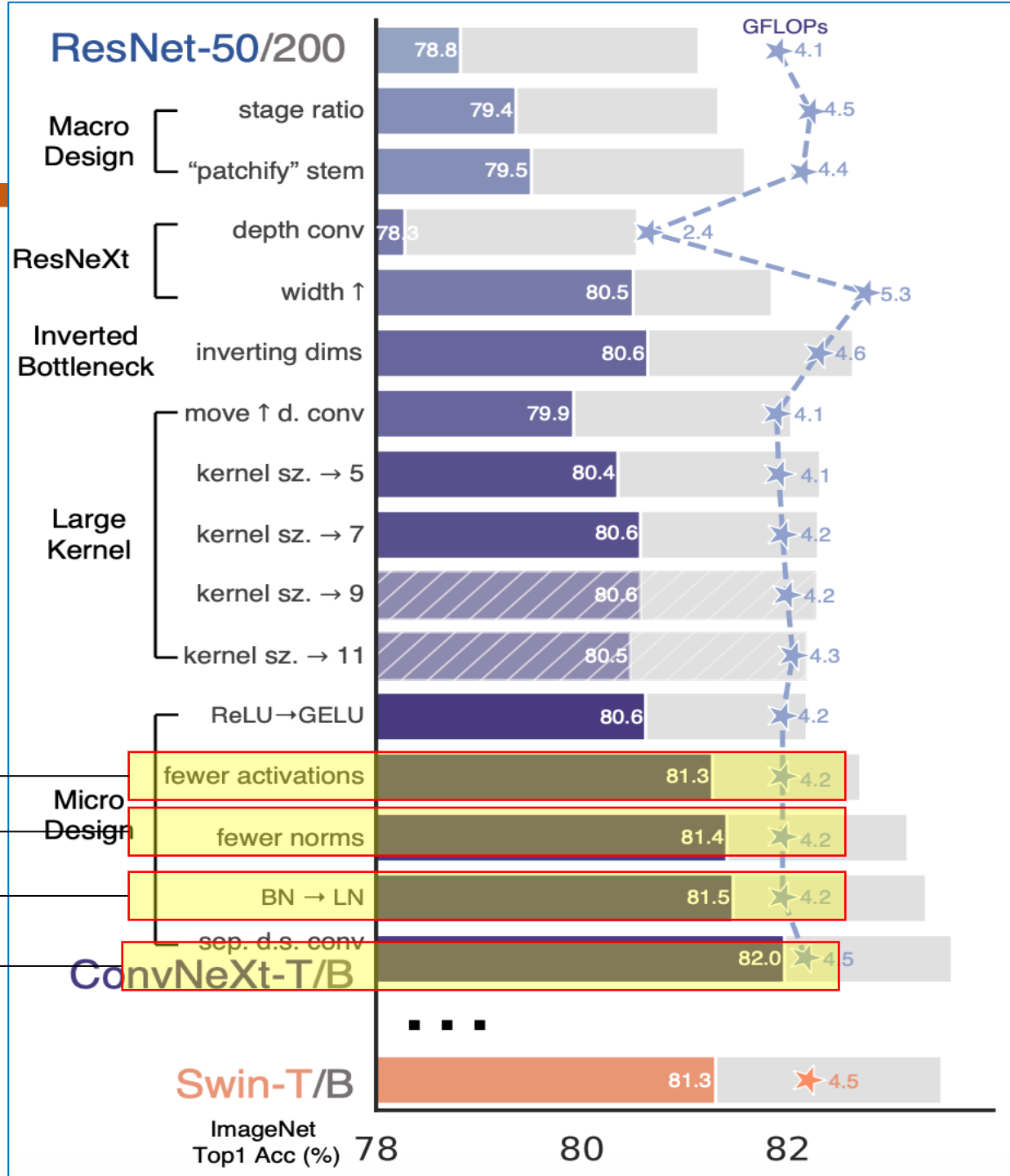
Mirco Design

One Activation function per block to mimic Transformer

Use only One Norm layer instead of 3

Substitute LayerNorm for BatchNorm

Adding separate down sampling layer + Norm layers when spatial resolution changes helps stabilize training

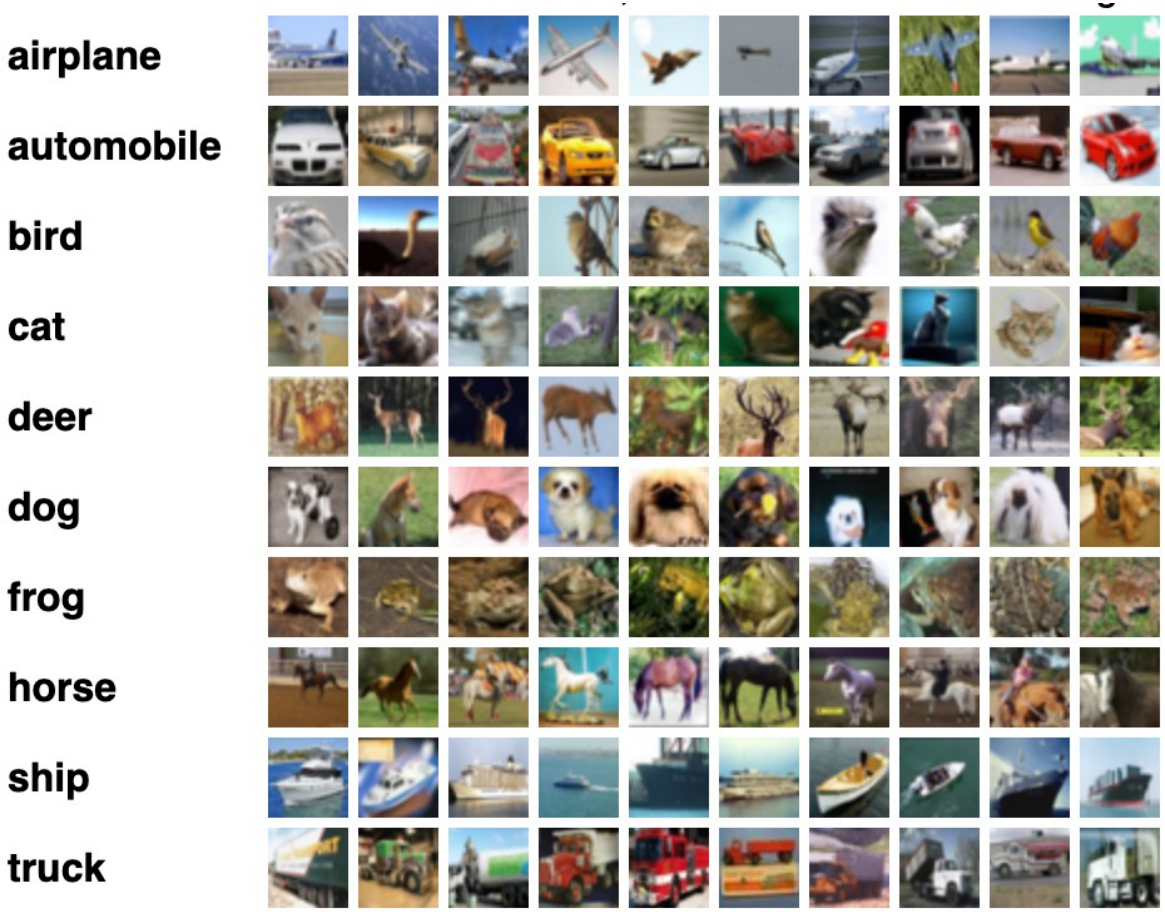


ResNet50 ratio: 3:4:6:3 => 3:3:9:3

Outline

- CNN: Basic Concepts
- LeNet
- AlexNet
- ZFNet
- VGGNet
- GoogleLeNet
- ResNet
- MobileNet
- ConvNext
- Performance Evaluation on Cifar-10 Dataset

CIFAR-10 Dataset



The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Performance Evaluation

Image Size: 32 x 32

name	year	#params (M)	pretrained?	accuracy (%)
LeNet	1998	4.5	no	72.5
AlexNet	2012	57	no	69.5
ZFNet	2013	58.3	no	67.8
VGG11	2014	128.8	no	78
VGG16	2014	134.3	no	78.9
VGG19	2014	139.6	no	78.6
GoogLeNet	2014	5.6	no	80.5
ResNet18	2015	11.2	no	77.4
ResNet34	2015	21.3	no	77
ResNet50	2015	23.5	no	75.7
ResNet101	2015	41.4	no	75.4
MobileNetv1	2017	3.2	no	75.6
MobileNetv2	2018	2.2	no	67.4
ConvNeXT_tiny	2022	27.8	no	62.2
ConvNeXT_small	2022	49.4	no	62.5
ConvNeXT_base	2022	87.6	no	62.6

Image Size: 224 x 224

name	year	#params (M)	pretrained?	accuracy (%)
LeNet	1998	4.5	no	66.4
AlexNet	2012	57	no	72.2
ZFNet	2013	58.3	no	67.3
VGG11	2014	128.8	no	81
VGG16	2014	134.3	no	81.5
VGG19	2014	139.6	no	80.2
GoogLeNet	2014	5.6	no	86.4
ResNet18	2015	11.2	no	86.8
ResNet34	2015	21.3	no	85.9
ResNet50	2015	23.5	no	85.6
ResNet101	2015	41.4	no	86
MobileNetv1	2017	3.2	no	85.6
MobileNetv2	2018	2.2	no	87.5
ConvNeXT_tiny	2022	27.8	no	81.1
ConvNeXT_small	2022	49.4	no	82.2
ConvNeXT_base	2022	87.6	no	81

Done by TA Bui Minh Duc

Summary

Model	Year	Type	Features	#params	#layers	Model description	Note
LeNet	1998		a pioneering CNN arch. for handwritten digit recognition	4.5M	4	2 conv + 2 fc layers	
AlexNet	2012		deep architecture with multiple conv layers	57M	8	5 conv + 3 fc layers	
ZFNet	2013		a refinement of AlexNet larger filter sizes	58.3M	8	5 conv + 3 fc layers	
GoogLeNet	2014		inception modules	5.6M	22	21 conv + 1 fc layers	
VGGNet	2014	VGG-16	very deep architecture with multiple conv layers stack of 3x3 conv layers 1x1 conv layer at the end of each stack	134.3M	16	13 conv + 3 fc layers	
		VGG-19			19	16 conv + 3 fc layers	
ResNet	2015	ResNet-18	residual connections address the vanishing gradient problem	11.2M	18	17 conv + 1 fc layers	
		ResNet-34		21.3M	50	49 conv + 1 fc layers	different hidden dim
		ResNet-50		23.5M	50	49 conv + 1 fc layers	
		ResNet-101		41.4M	101	100 conv + 1 fc layers	
MobileNet	2017	MobileNetv1	factorize a standard convolution into: - a depthwise convolution - a 1x1 conv (pointwise conv)	3.2M	28	27 conv + 1 fc layers	
	2018	MobileNetv2	inverted residual blocks pointwise linear bottleneck layer	2.2M	53	52 conv + 1 fc layers	
	2019	MobileNetv3_small	mNASNet search space squeeze-and-expand block	1.5M	54	52 conv + 2 fc layers	
		MobileNetv3_large		4.2M	64	62 conv + 2 fc layers	
ConvNeXT	2022	ConvNeXt-T	depthwise separable conv layer-wise adaptive activation global response normalization	28M	81	80 conv + 1 fc layers	
		ConvNeXt-S		50M	153	152 conv + 1 fc layers	different hidden dim
		ConvNeXt-B		89M	153	152 conv + 1 fc layers	
		ConvNeXt-L		198M	153	152 conv + 1 fc layers	
		ConvNeXt-XL		350M	153	152 conv + 1 fc layers	

Done by TA Bui Minh Duc


Thank you