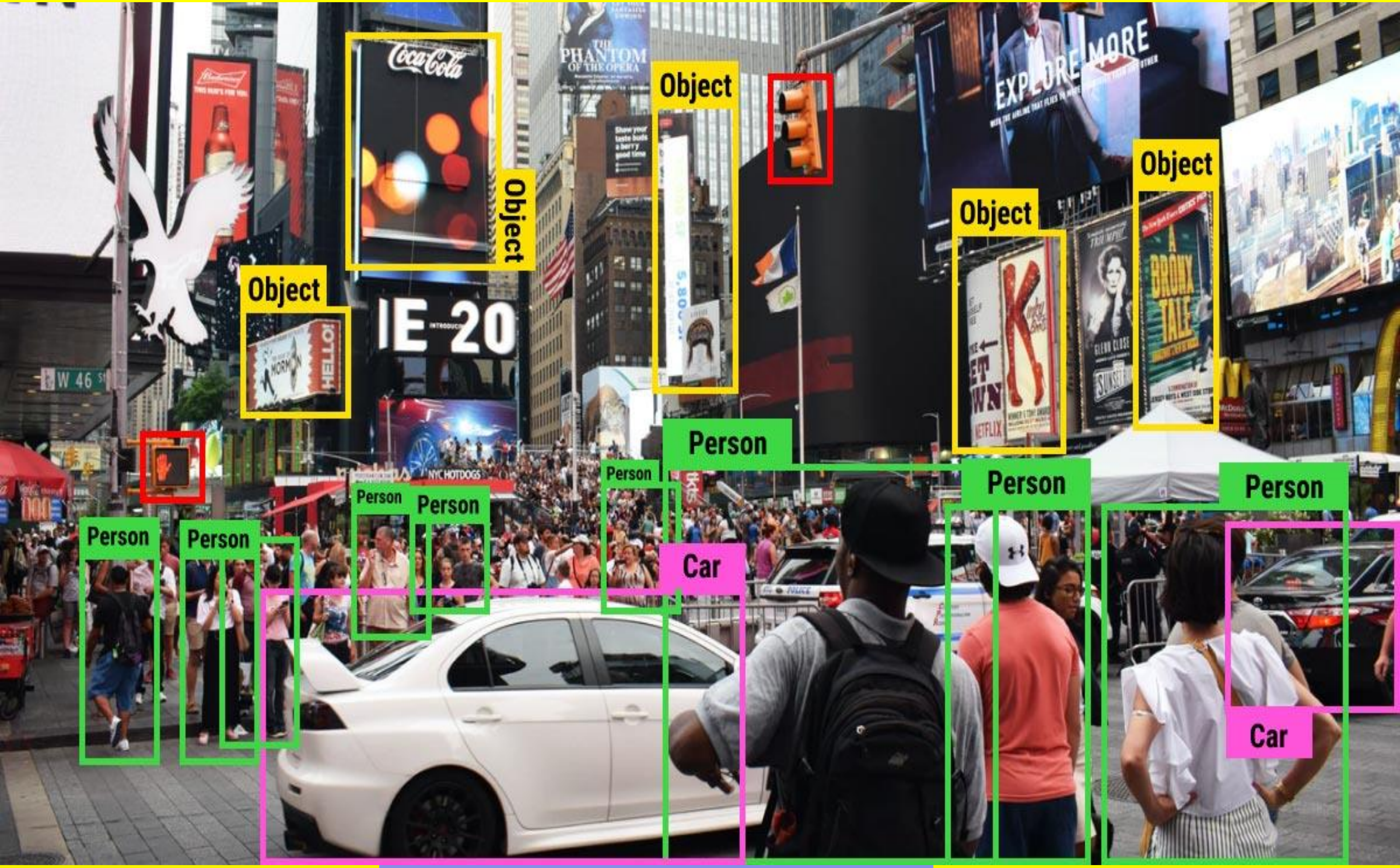
Lecture-12-13
Course: Data Science

Convolutional Neural Network and Transfer Learning

By
Dr. Sibarama Panigrahi
Senior Member, IEEE
Assistant Professor, Department of Computer Sc. & Engineering
National Institute of Technology, Rourkela, Odisha, 769008, India
Mobile No.: +91-7377302566
Email: panigrahis[at]nitrkl[dot]ac[dot]in
panigrahi[dot]sibarama[at]gmail[dot]com

MOTIVATION

MOTIVATION

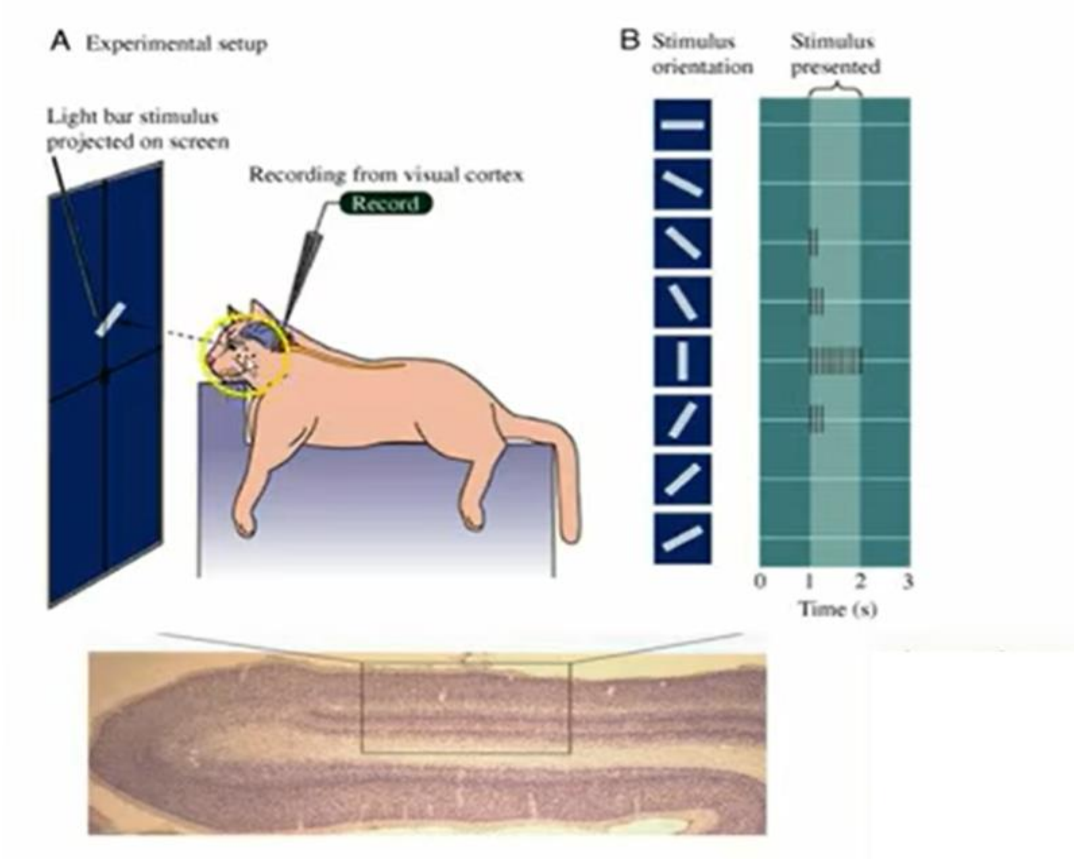

Object
Object
Object
Object
Object
Person
Person
Person
Person
Person
Person
Person
Person
Person
Car
Car

MOTIVATION


Object Tracking
Pose Estimation
Object Detection
Action Recognition
Autonomous Navigation
3D Reconstruction
Crowd Understanding
Urban Scene Understanding
Indoor Scene Understanding
Multi-agent Collaboration
Human Training
Aerial Surveying
Image
Image Label
Depth/Multi-View
User Input
Video
Physics
Segmentation/Bounding Box
Camera Localization

CONVOLUTIONAL NEURAL NETWORKS

- Popularly called **as CNN or Convnets.**
- https://en.wikipedia.org/wiki/David_H._Hubel#Research
- https://www.youtube.com/watch?v=v20-E_2bT2c
- **Hubel and Wiesel** received the Nobel Prize.



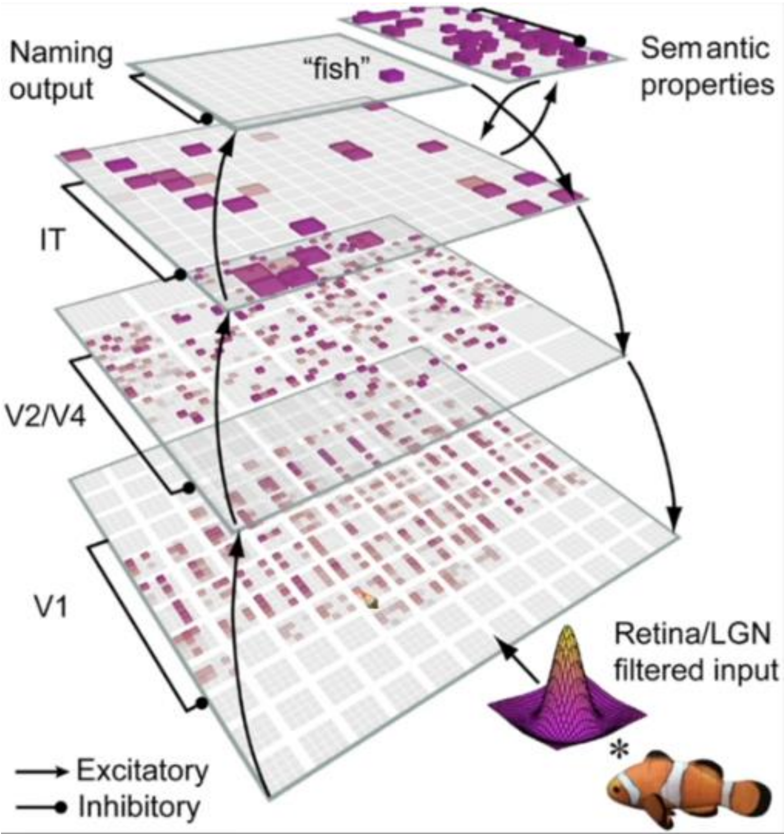
KEY FINDINGS

- Some neurons in the visual cortex fires when lines at specific angle is presented.
- There is a special region called ***primary visual cortex that detects edges***.
- There are some more complex neurons that detect motion, depth, color, shapes, complex edges like faces.

Functional specialization

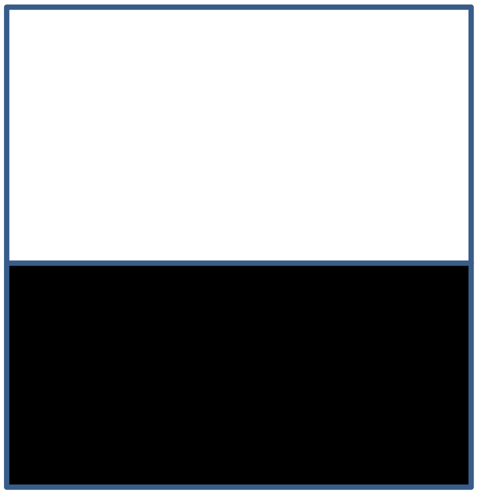
Match each visual area to its corresponding function:

V1	Motion / **Edges**
V2	Stereo
V3	Color
V3a	Texture segregation
V3b	Segmentation, grouping
V4	Recognition
V7	Face recognition
MT	Attention
MST	Working memory/mental imagery
etc	Etc.



CONVOLUTION: EDGE DETECTION

We detect edges by applying Convolution operator on the i/p image.



Gray Scale Image
(6X6)

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

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

Sobel Horizontal Edge Detector/ Kernel

Convolution

Kernel / Filter / Mask / Operator

CONVOLUTION: EDGE DETECTION

A

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

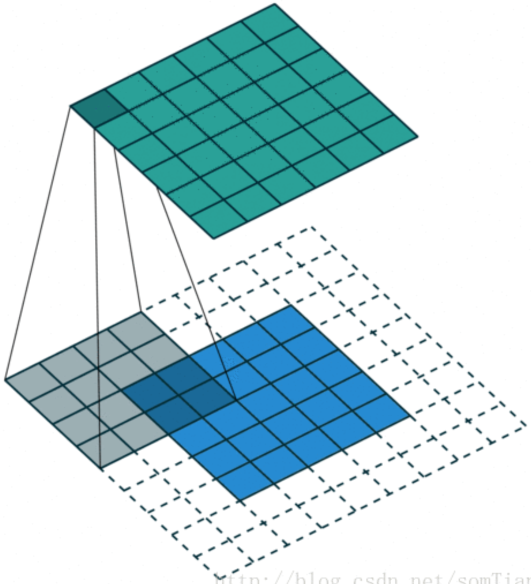
K

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

=

R

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0



0*1 + 0*2 + 0*1 +
0*0 + 0*0 + 0*0 +
-1*0 + -2*0 + -1*0 +
= 0

CONVOLUTION: EDGE DETECTION

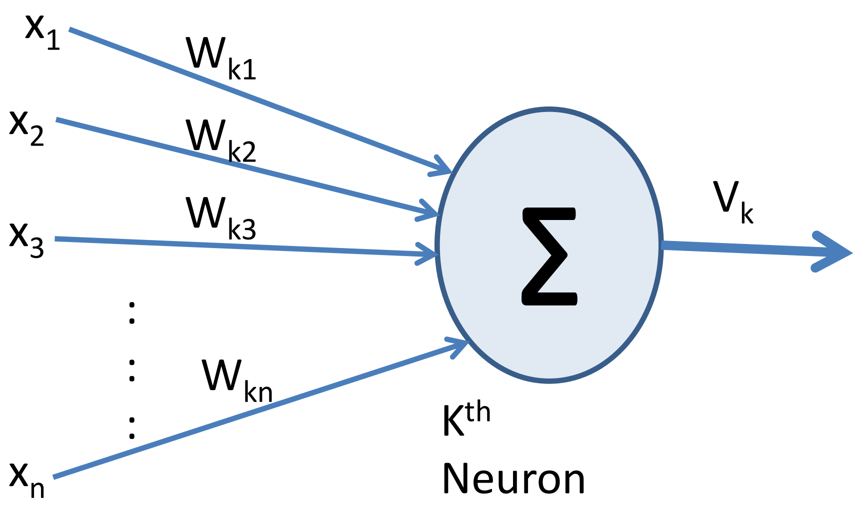
0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

Normalization

255	255	255	255
0	0	0	0
0	0	0	0
255	255	255	255

CONVOLUTION: EDGE DETECTION

- Convolution is a generalization of a dot product.
- We can achieve it through the operation of a neuron in the neural network.



$$V_k = W_{k1}*x_1 + W_{k2}*x_2 + W_{k3}*x_3 + \ldots + W_{kn}*x_n$$

CONVOLUTION: EDGE DETECTION

- Sobel Kernel
  - Horizontal can detect Horizontal Edge
  - Vertical can detect Vertical Edge

- Kernels can be of any size Typically a square matrix.

Sobel Vertical Edge Detector/ Kernel

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

CONVOLUTION: EDGE DETECTION

Formulation [edit]

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. If we define **A** as the source image, and $\mathbf{G}_x$ and $\mathbf{G}_y$ are two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations are as follows:[2]

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

where $*$ here denotes the 2-dimensional signal processing convolution operation.

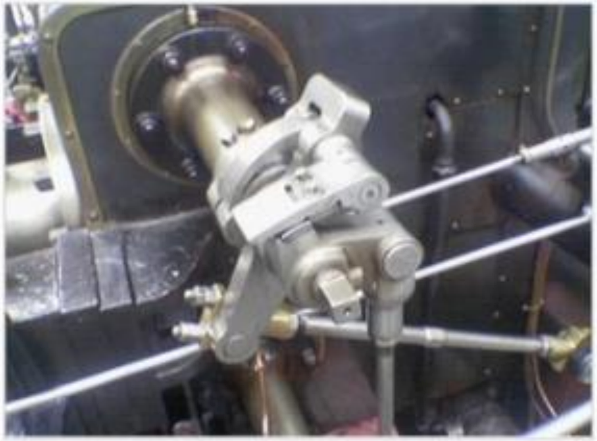
Since the Sobel kernels can be decomposed as the products of an averaging and a differentiation kernel, they compute the gradient with smoothing. For example, $\mathbf{G_x}$ can be written as

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

The *x*-coordinate is defined here as increasing in the "right"-direction, and the *y*-coordinate is defined as increasing in the "down"-direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

Using this information, we can also calculate the gradient's direction:



A color picture of a steam engine

The Sobel operator applied to that image

MOTIVATION FOR PADDING

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

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

=

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

- By Performing Convolution operation using 3 X 3 matrix on an input image of 6X6, We have got a 4X4 result.
- By Performing Convolution operation using **K** X **K** matrix (filter) on an input image of **N** X **N**, We have got a **N-K+1** X **N-K+1** result.
- This results in reduction in Dimension.
- How not to reduce the dimension of the original Image.

PADDING

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

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

=

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

0	0	0	0	0	0
0	0	0	0	0	0
0	-1020	-1020	-1020	-1020	0
0	-1020	-1020	-1020	-1020	0
0	0	0	0	0	0
0	0	0	0	0	0

– This can be achieved by padding 1 row and 1 column of zeros around the resultant matrix. **This is called padding by 1.**

PADDING

- If you add Zero in padding it is called Zero Padding. (Extensively Used because of Simplicity)

- You can do same value padding.

- Padding ***m*** results in ***n+2m*** size

0	0	0	0	0	0
0	0	0	0	0	0
0	-1020	-1020	-1020	-1020	0
0	-1020	-1020	-1020	-1020	0
0	0	0	0	0	0
0	0	0	0	0	0

0	0	0	0	0	0
0	0	0	0	0	0
1020	-1020	-1020	-1020	-1020	1020
1020	-1020	-1020	-1020	-1020	1020
0	0	0	0	0	0
0	0	0	0	0	0

STRIDE

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

A

*

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

K

=

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

R

Stride 1= Shift by 1 column/row

STRIDE

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

A

*

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

K

=

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

R

Stride 2= Shift by 2 column/row

STRIDE

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

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

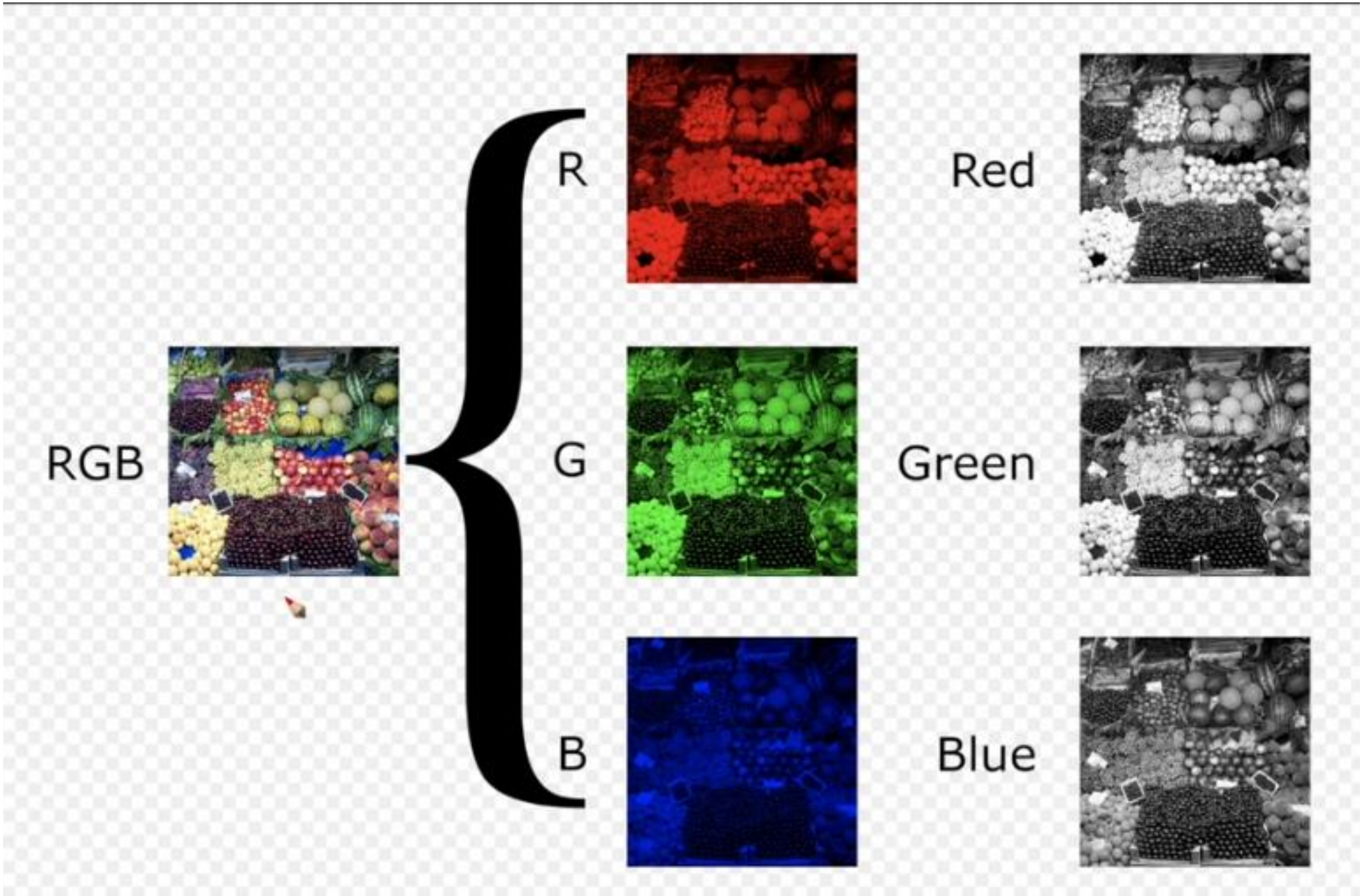
=

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

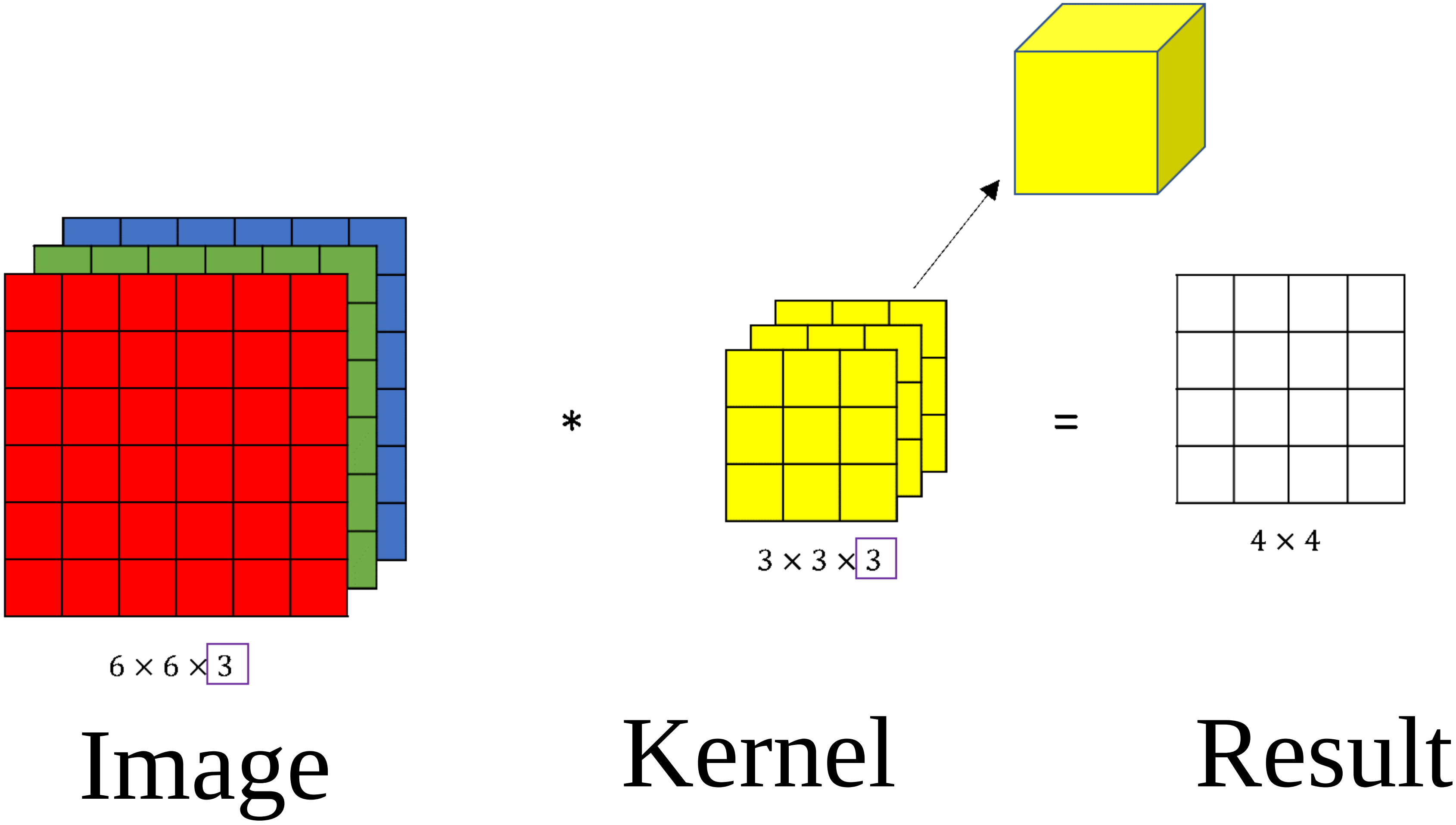
A K R

n X n → Stride=s and Kernel= kX k → $\left(\left\lfloor\frac{n-k}{s}\right\rfloor + 1\right) \text{ X } \left(\left\lfloor\frac{n-k}{s}\right\rfloor + 1\right)$

CONVOLUTION IN AN RGB IMAGE



CONVOLUTION IN AN RGB IMAGE



CONVOLUTION IN AN RGB IMAGE

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$158 \quad + \quad -14 \quad + \quad 653 \quad + 1 = 798$$

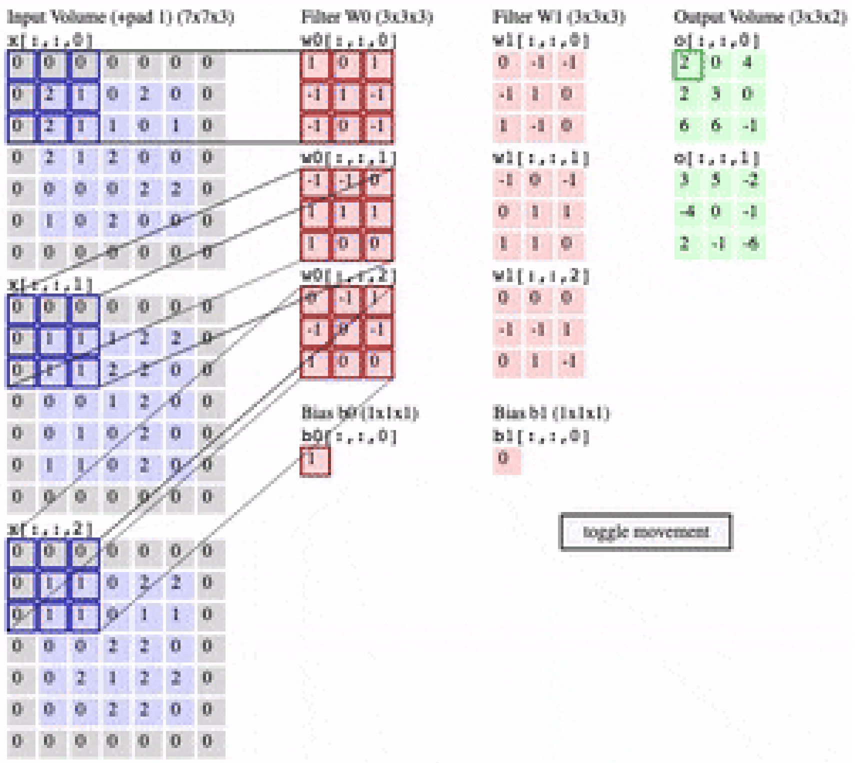
Bias = 1

Output

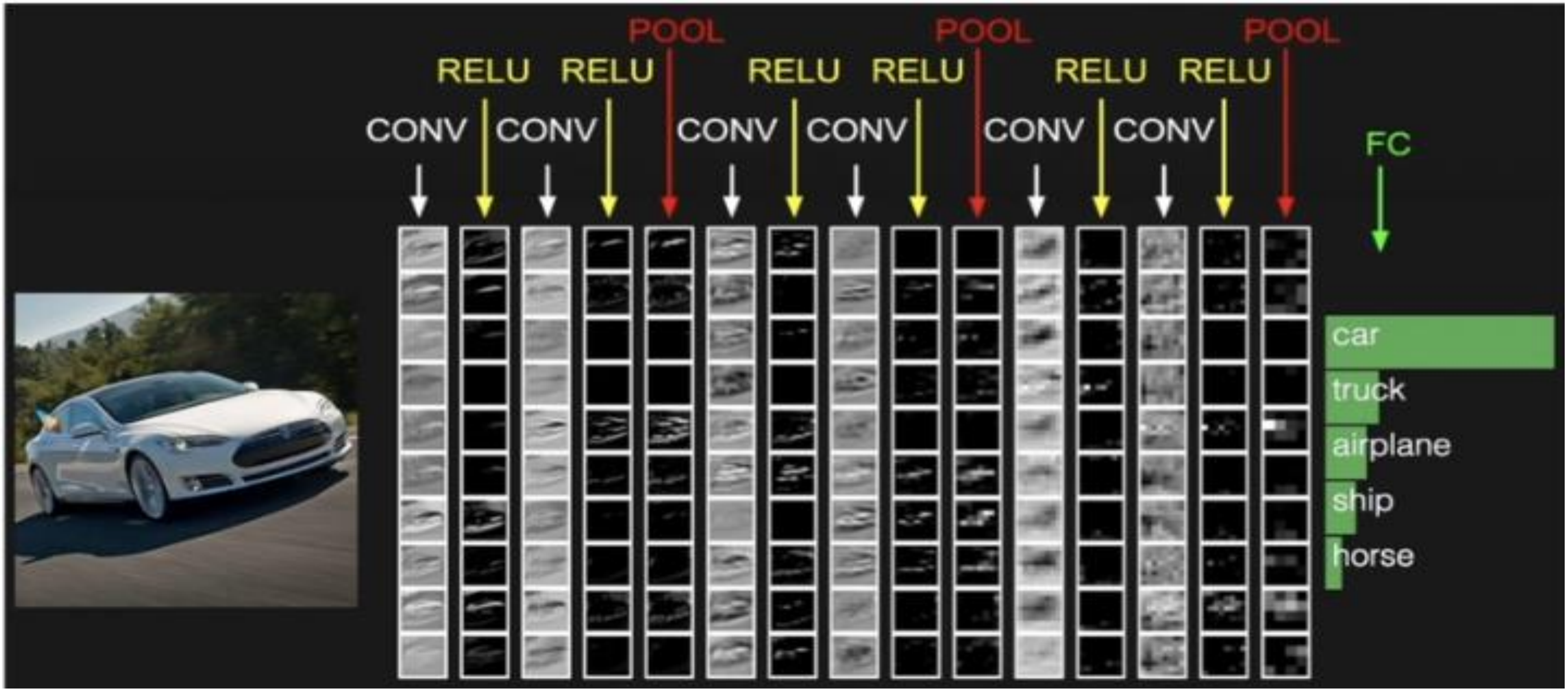
-25	466	466	475	...
295	787	798		...
				...
				...
...	...	...	...	...

CONVOLUTION LAYER IN CNN

- Biologically Inspired
- Multiple Edge Detectors: Multiple Kernels
- In MLP, we learn the weights
- In CNN, We learn the kernel matrices

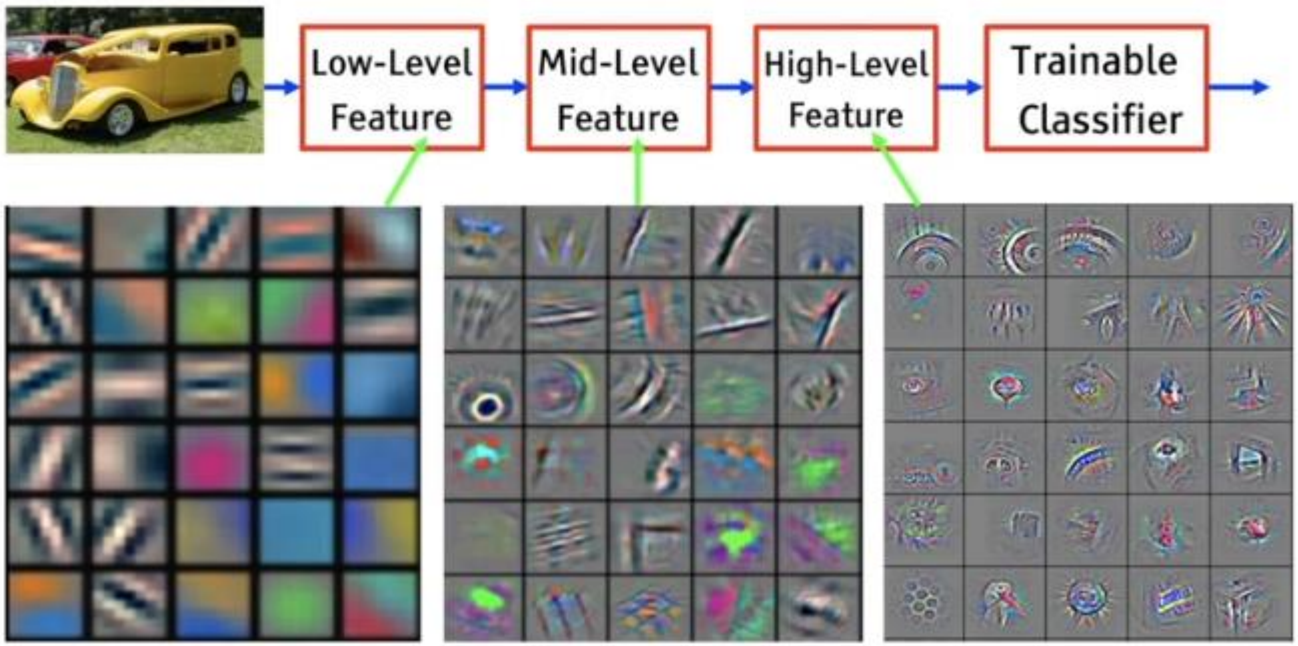


CONVOLUTION LAYER IN CNN



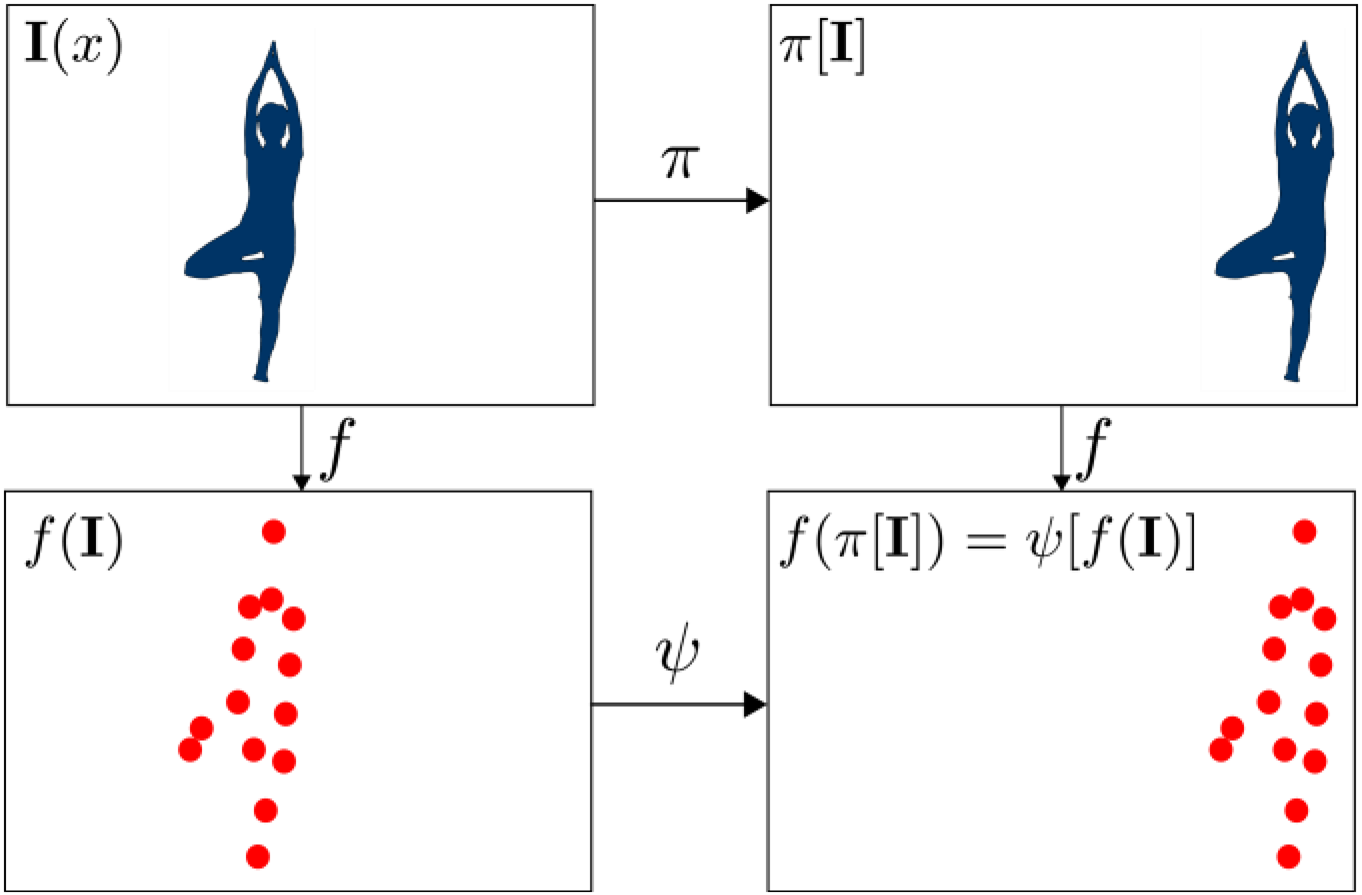
WHY MULTIPLE LAYERS?

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition: Pixel → edge → texton → motif → part → object
- Text: Character → word → word group → clause → sentence → story
- Speech: Sample → spectral band → sound → … → phone → phoneme → word



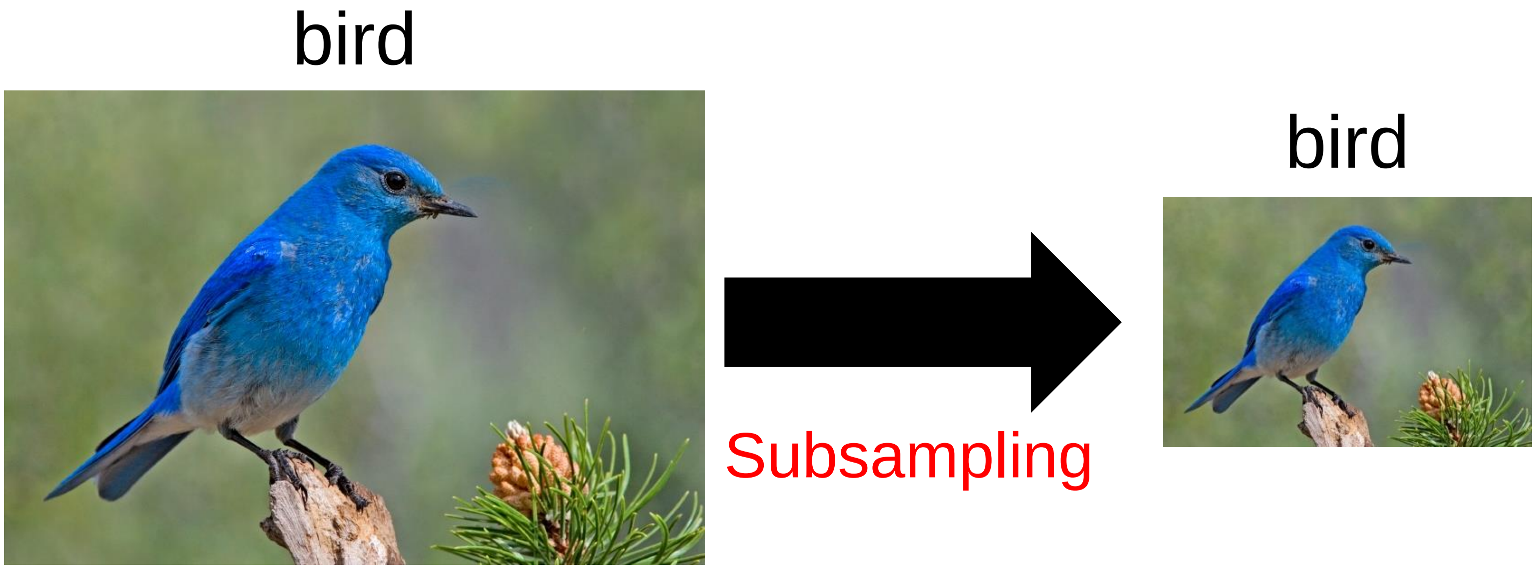
MOTIVATION FOR POOLING

- **Location Invariant:** Changing the location will not change the object.



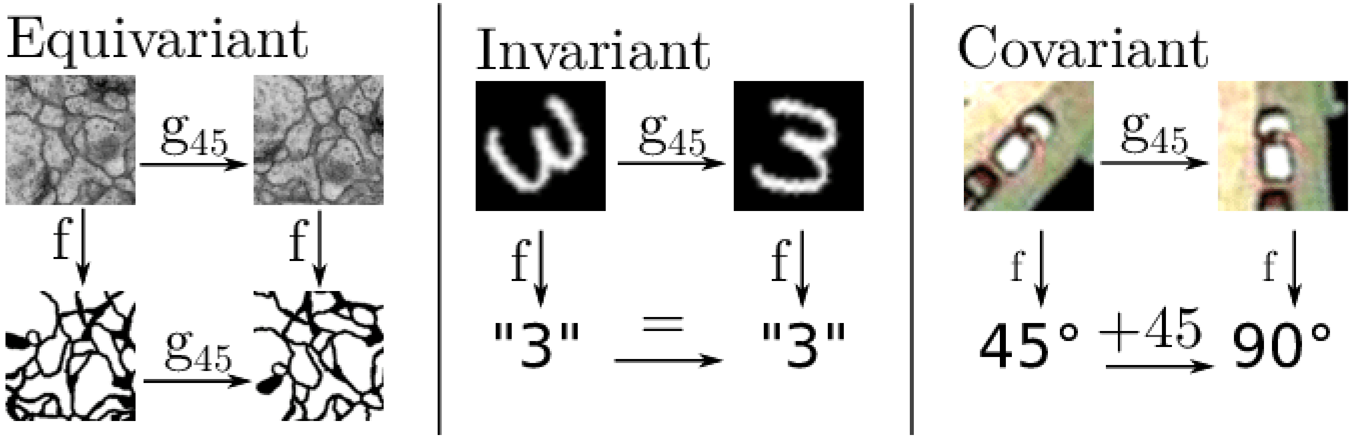
MOTIVATION FOR POOLING

- **Scale Invariant:** Subsampling Image will not change the Image



Motivation for Pooling

- **Rotation Invariant:** Rotating an image will not change the Object.



MOTIVATION FOR POOLING

- **Pooling is a concept** that makes the CNN models **invariant to *Location, Scale and Rotation.***

POOLING

- **E.g. Max Pooling**
  - Let's have a 4X4 image with kernel size 2X2 and Stride 2

- Popular method

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

How Derivative of Pooling works in CNN

Derivatives of Pooling

Pooling layer subsamples statistics to obtain summary statistics with any aggregate function (or filter) *g* whose input is vector, and output is scalar. Subsampling is an operation like convolution, however *g* is applied to disjoint (non-overlapping) regions.

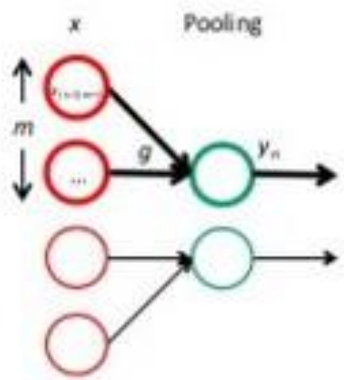
- Definition: *subsample* (or *downsample*)

  Let *m* be the size of pooling region, *x* be the input, and *y* be the output of the pooling layer. subsample(*f*, *g*)[*n*] denotes the *n*-th element of subsample(*f*, *g*).

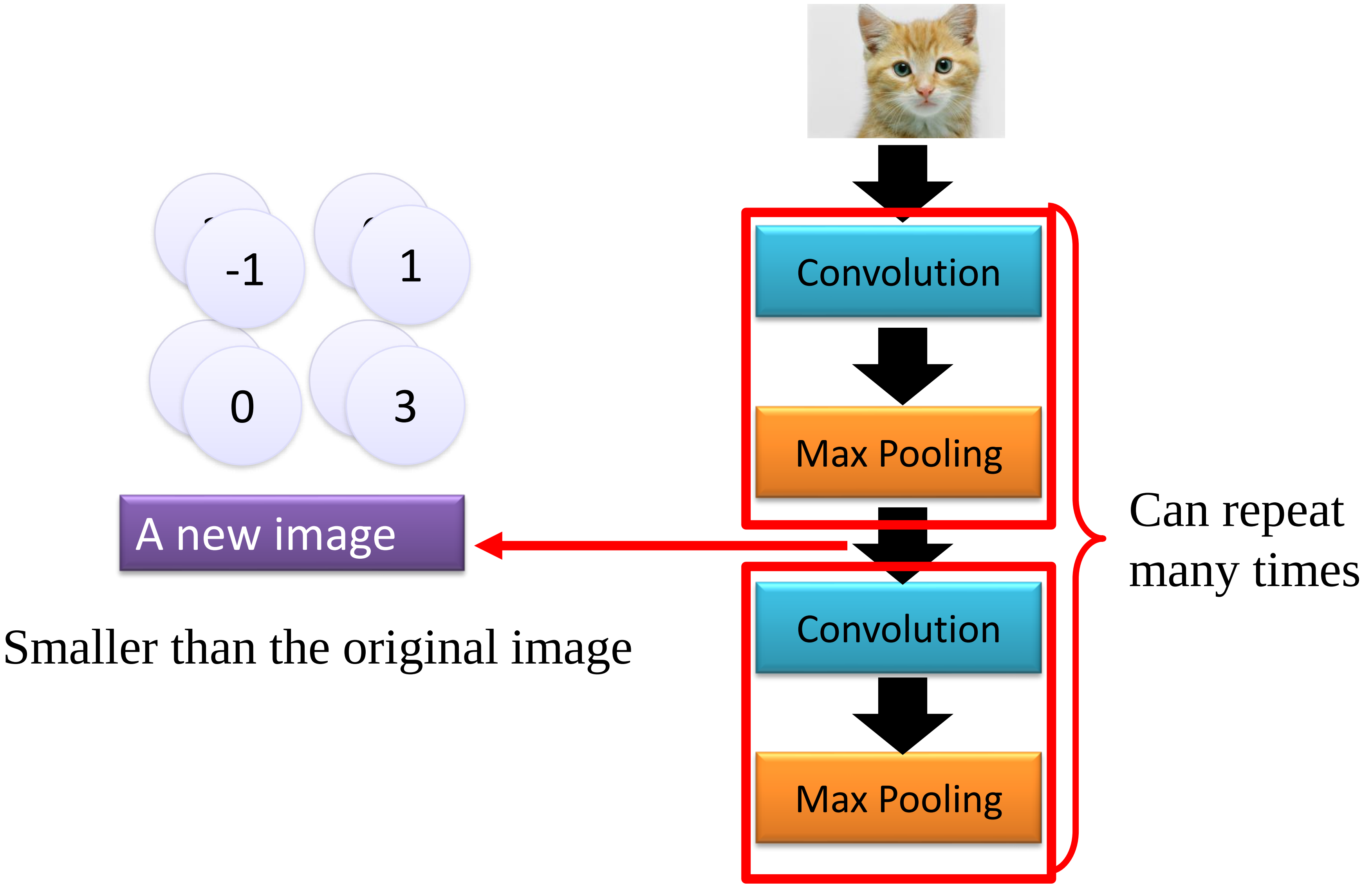
$$y_n = \text{subsample}(x, g)[n] = g\left(x_{(n-1)m+1:nm}\right)$$

$$y = \text{subsample}(x, g) = [y_n]$$

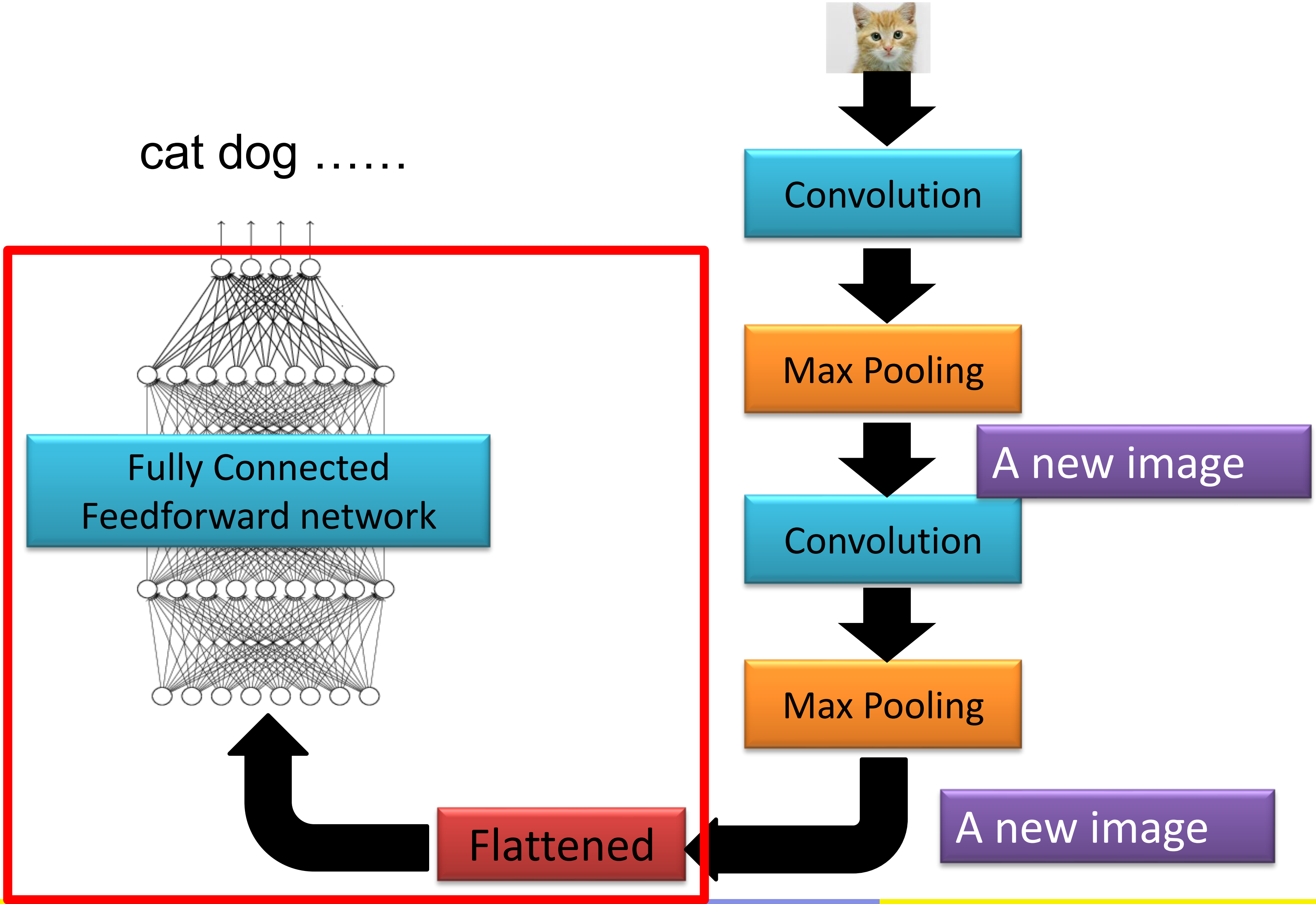
$$g(x) = \begin{cases} \dfrac{\sum_{k=1}^{m} x_k}{m}, \ \dfrac{\partial g}{\partial x} = \dfrac{1}{m} & \text{mean pooling} \\ \max(x), \ \dfrac{\partial g}{\partial x_i} = \begin{cases} 1 \text{ if } x_i = \max(x) \\ 0 \text{ otherwise} \end{cases} & \text{max pooling} \\ \|x\|_p = \left(\sum_{k=1}^{m} |x_k|^p\right)^{1/p}, \ \dfrac{\partial g}{\partial x_i} = \left(\sum_{k=1}^{m} |x_k|^p\right)^{1/p-1} |x_i|^{p-1} & L^p \text{ pooling} \\ \text{or any other differentiable } \mathbf{R}^m \to \mathbf{R} \text{ functions} & \end{cases}$$



CONVNETS


-1
1
0
3
A new image
Smaller than the original image
Convolution
Max Pooling
Convolution
Max Pooling
Can repeat many times

CONVNETS


cat dog ……
Convolution
Max Pooling
A new image
Convolution
Max Pooling
A new image
Fully Connected
Feedforward network
Flattened

POPULAR CNN MODELS

- LeNet
- AlexNet
- VGGNet
  - VGG16
  - VGG19
- ResNets
- GoogLeNet

LeNet

- https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

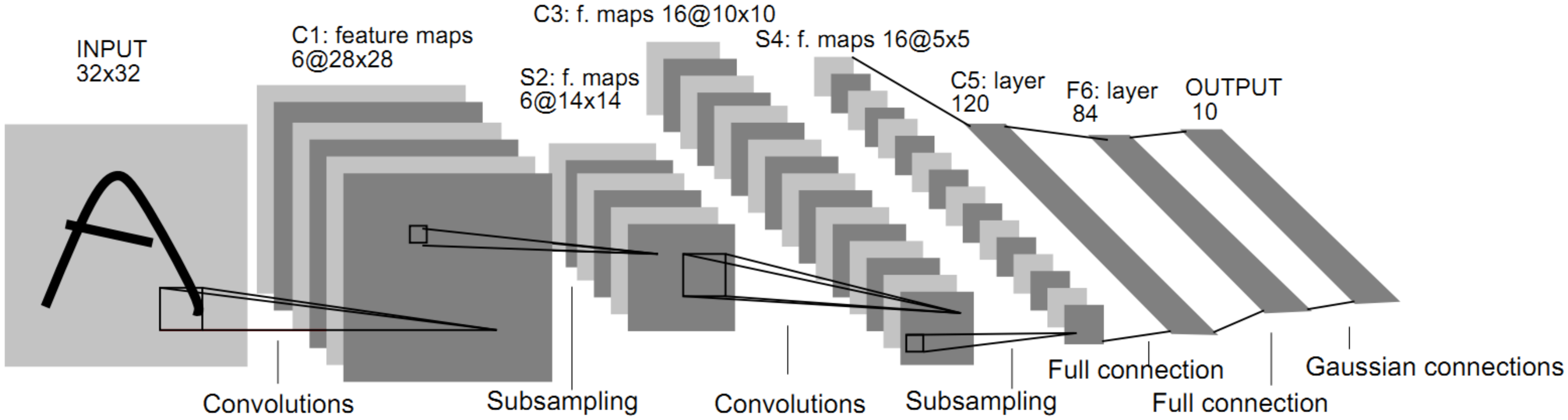


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.

LENET

https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

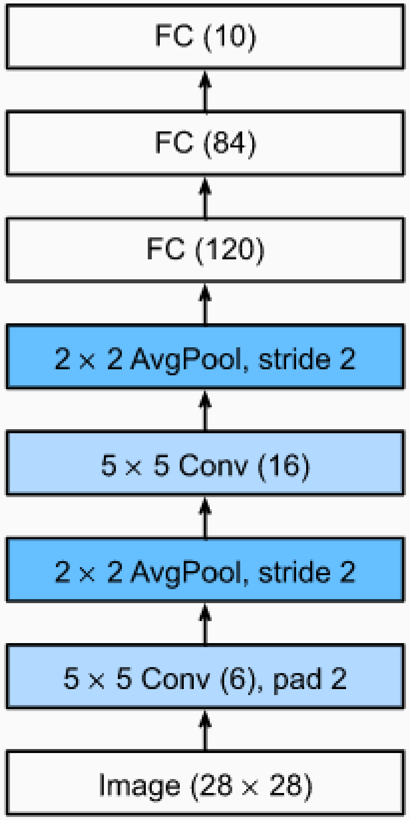
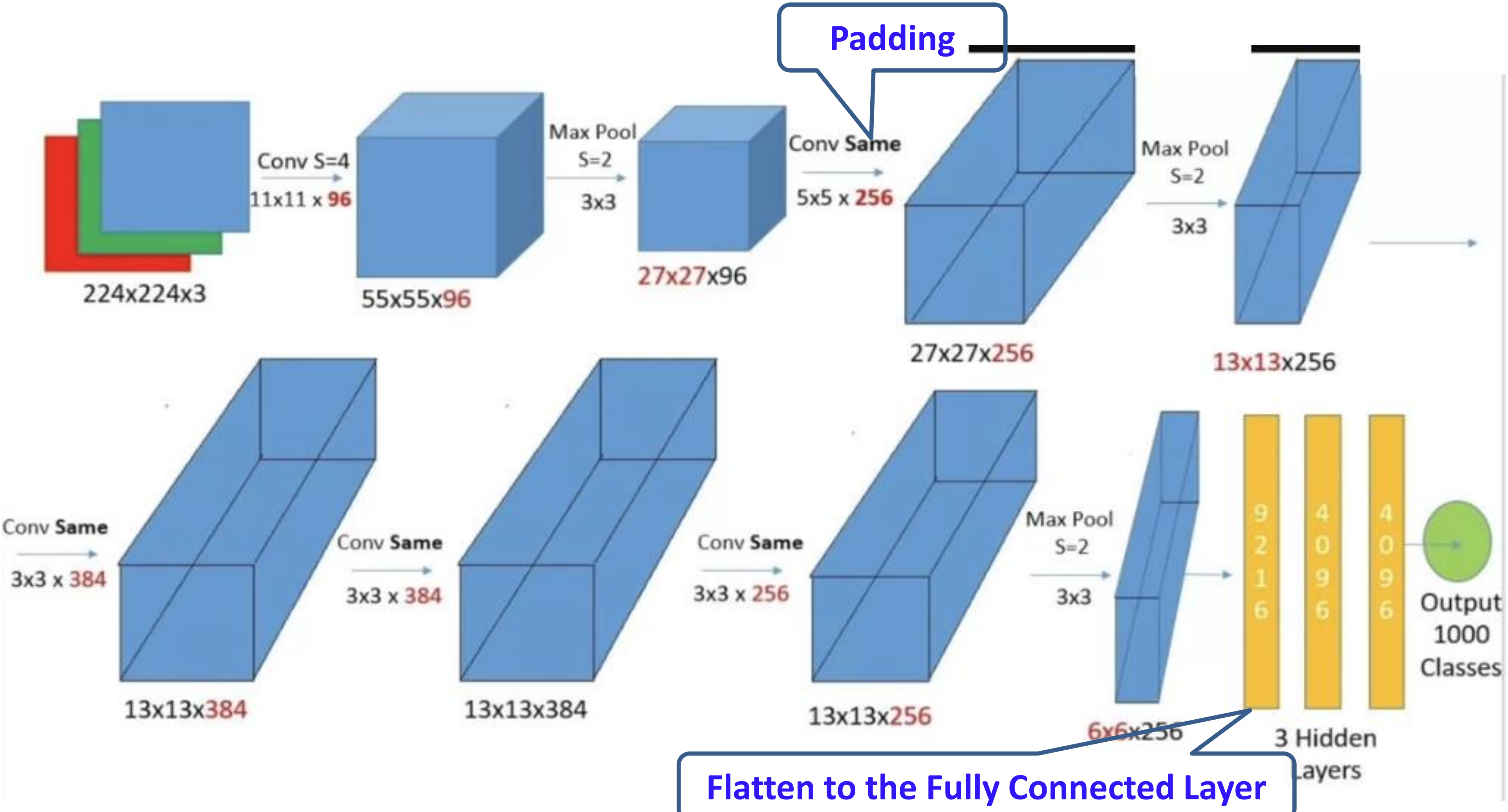


Fig. 6.6.2 Compressed notation for LeNet-5.

ALEXNET



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems, Lake Tahoe, Nevada, 3-6 December. 1: 1097-1105.

VGGNET- VGG16

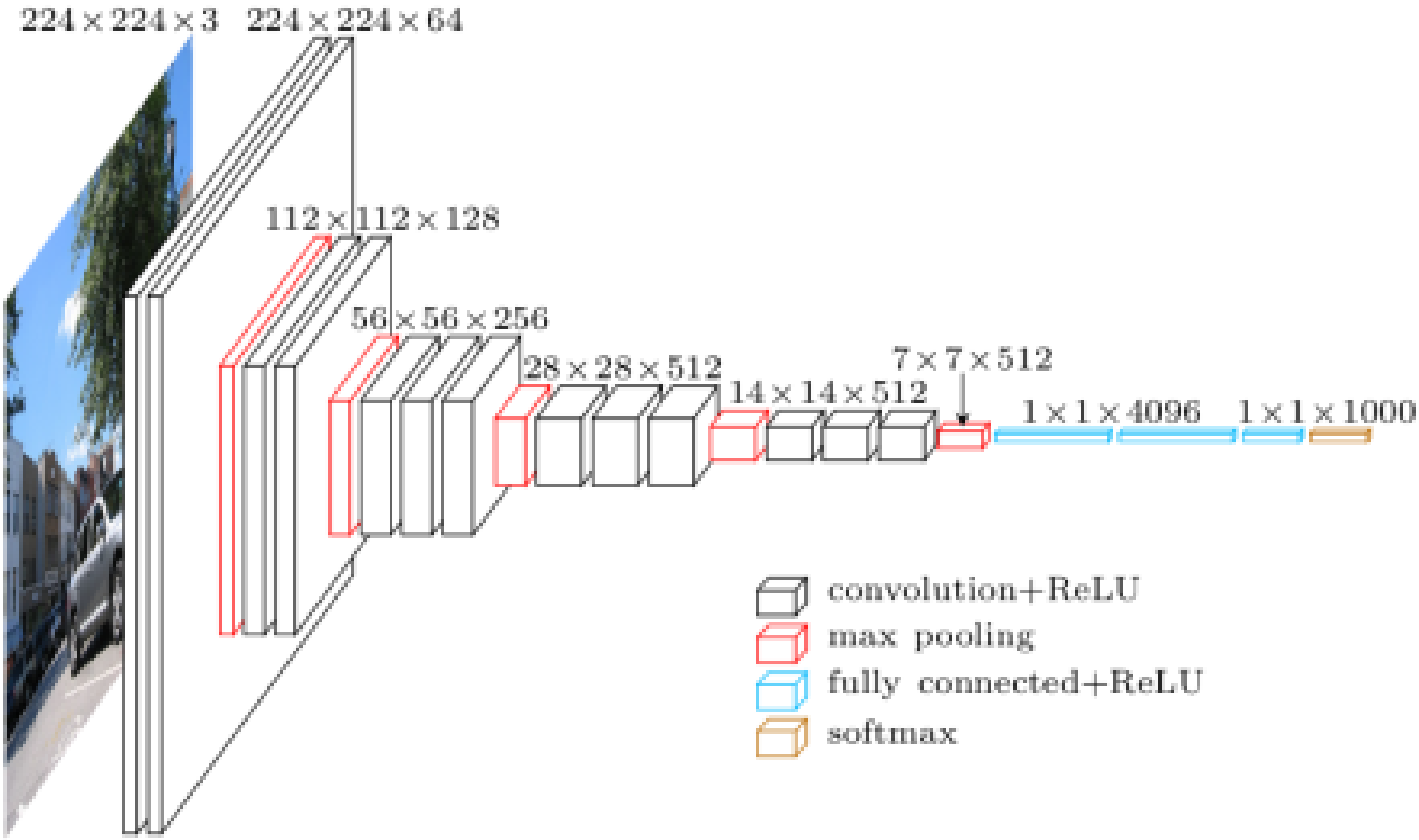
All Convolution (3X3 Kernel, Stride=1, Padding='Same')

All MaxPool (2X2, Stride=2)

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556.*

3D Visualization of VGG16 Architecture

VGGNET- VGG16

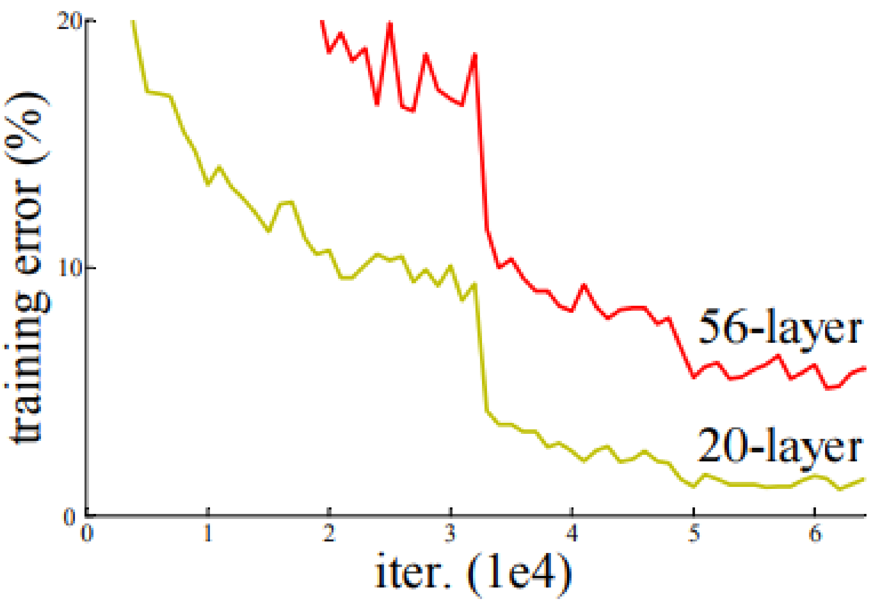


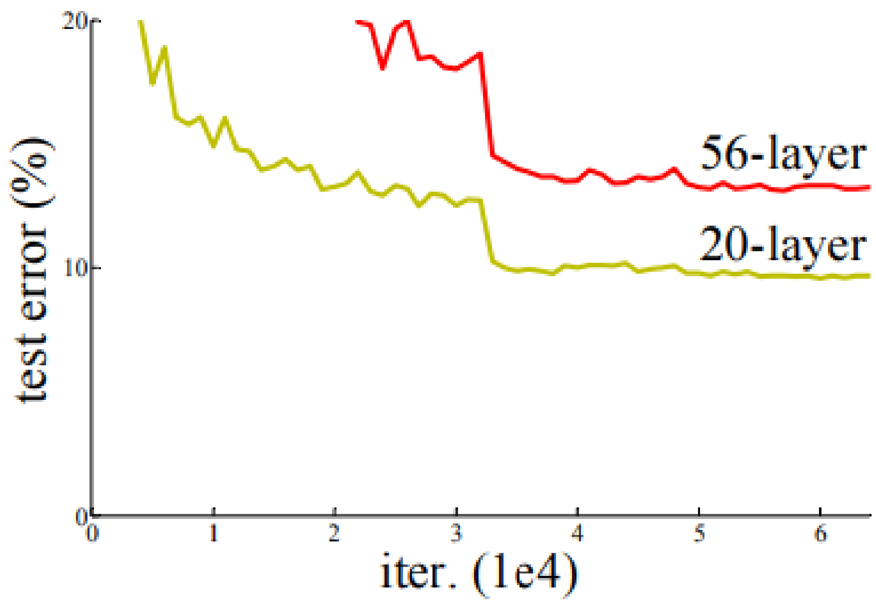
VGG-16

- Reference

https://github.com/keras-team/keras-applications/blob/master/keras_applications/vgg16.py

RESIDUAL NETWORKS: RESNETS





He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

RESIDUAL NETWORKS: RESNETS

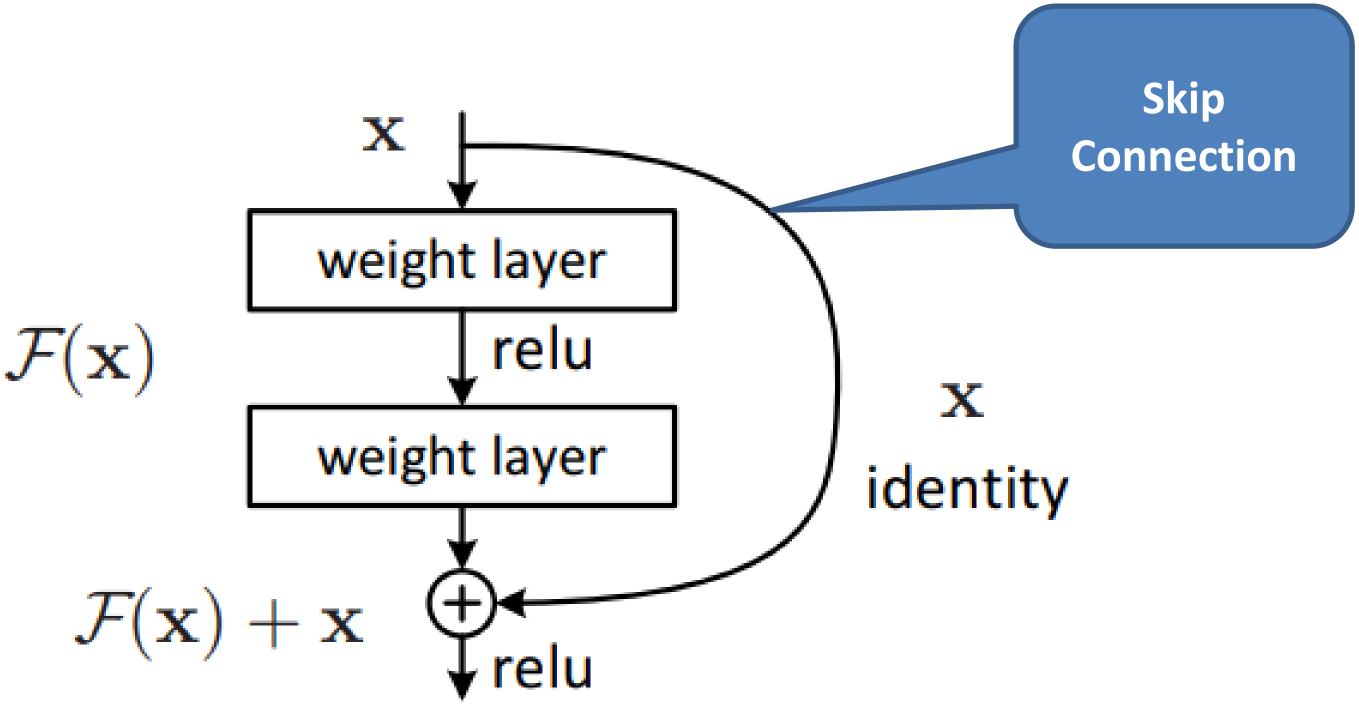
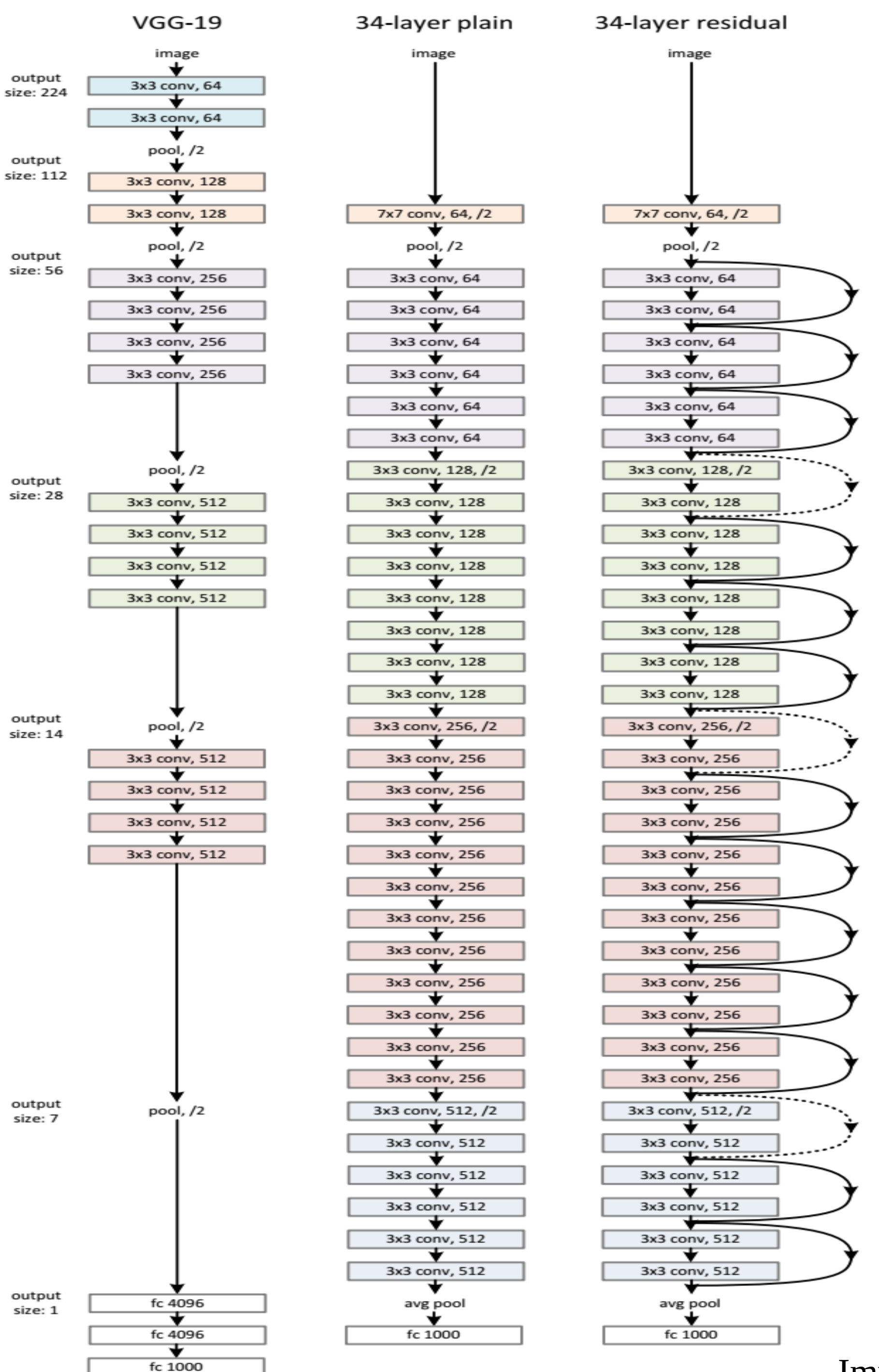


Figure 2. Residual learning: a building block.

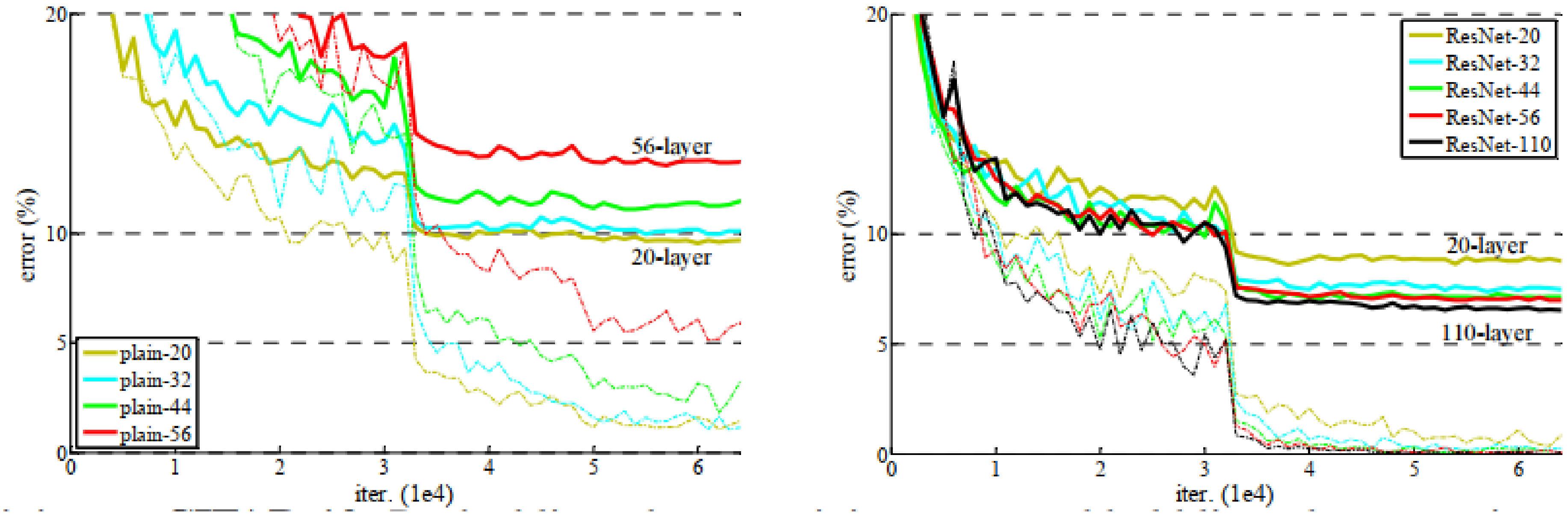
He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

RESIDU



Improvement in Performance due to Resudual

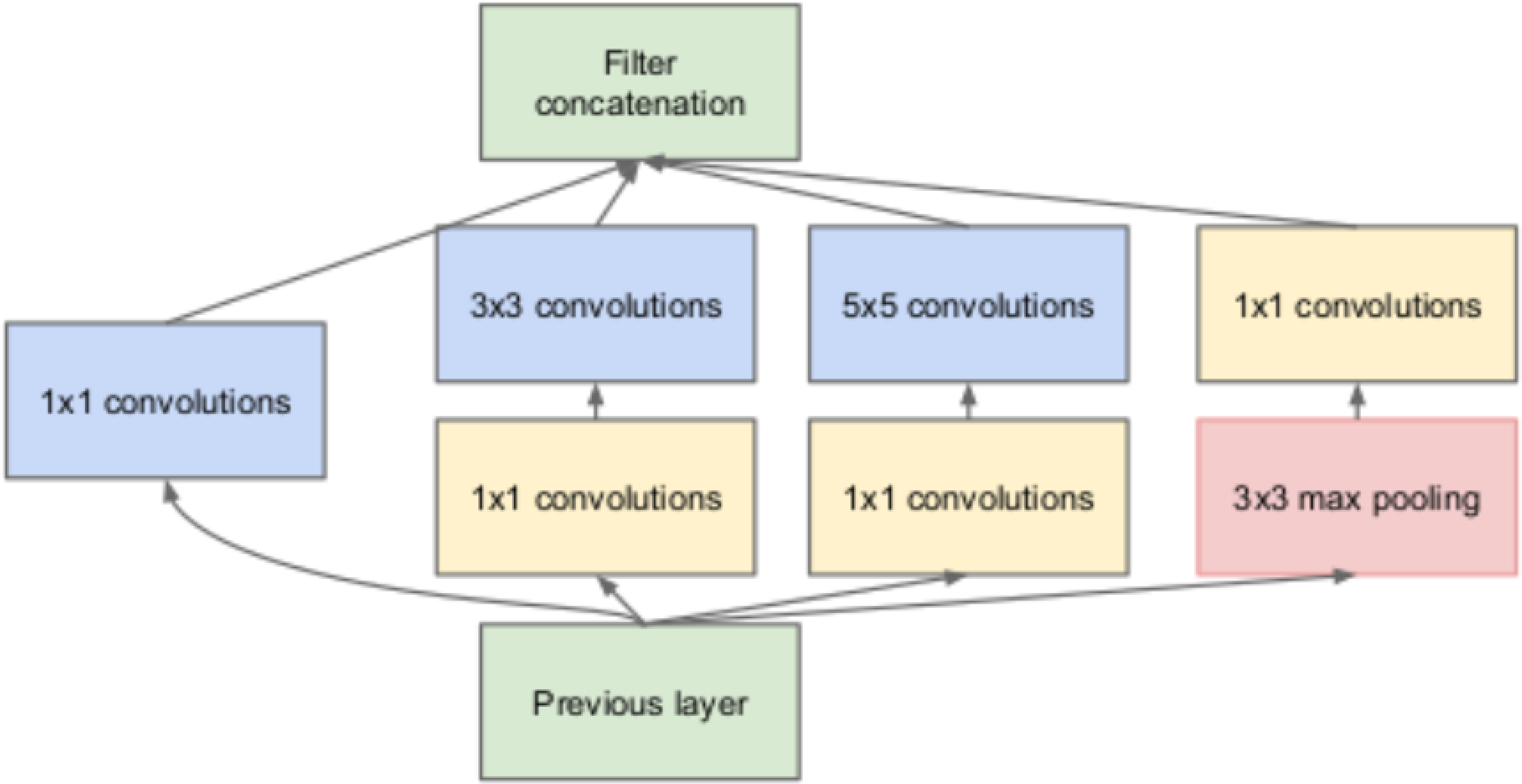
RESIDUAL NETWORKS: RESNETS



RESNETS-50

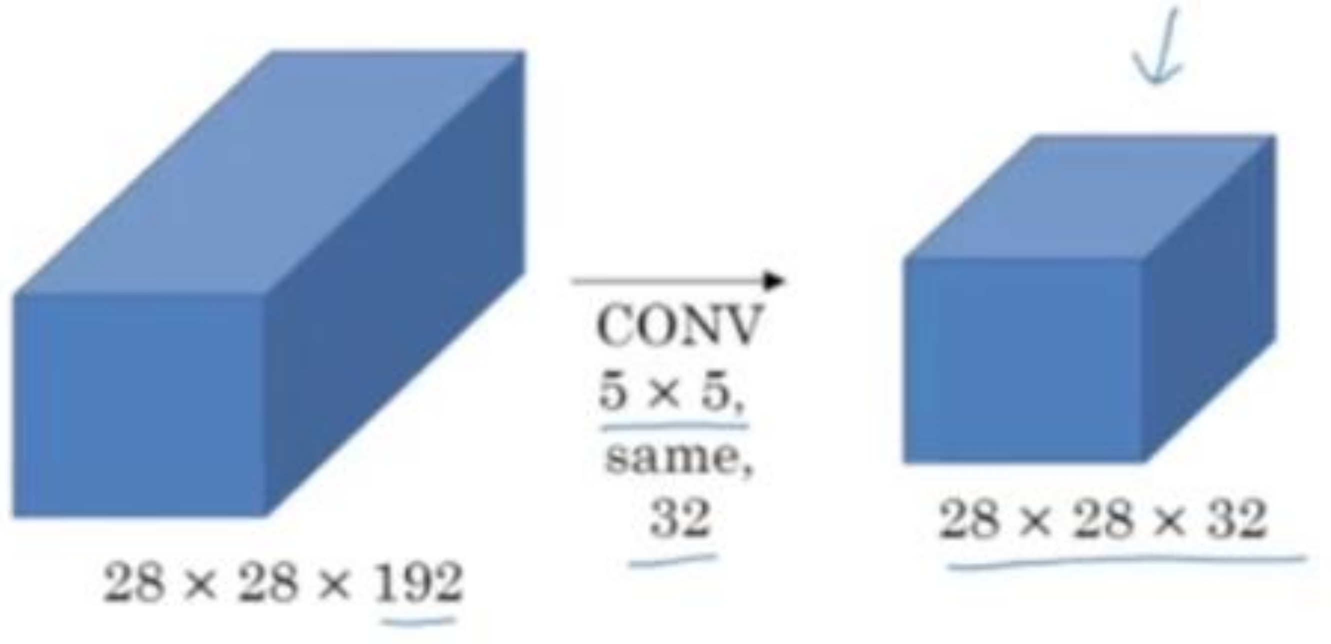
https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnet50.py

INCEPTION MODULE MOTIVATION



Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818-2826).

INCEPTION MODULE MOTIVATION



Input: 28x28x192

Filter: Conv 5x5x192, same, 32

Output: 28x28x32

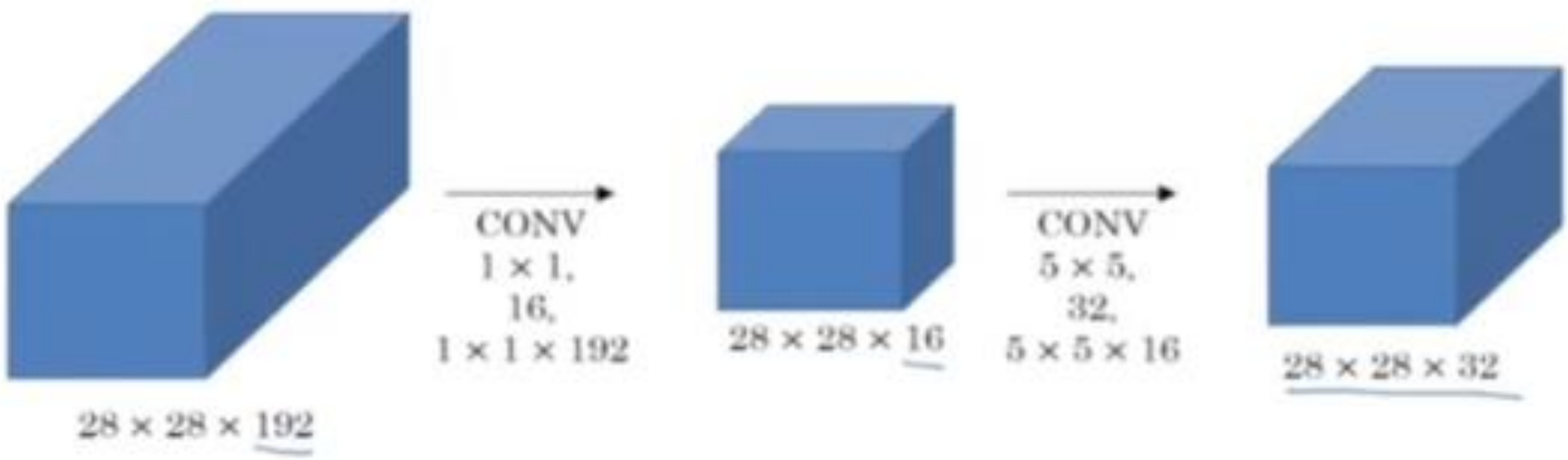
Total number of calculations = (28 * 28 * 32) * (5 * 5 * 192) = 120 Million !!

 Advantage of 1X1 kern before 5X5

INCEPTION MODULE MOTIVATION

Using 1x1 Convolution to reduce computation cost

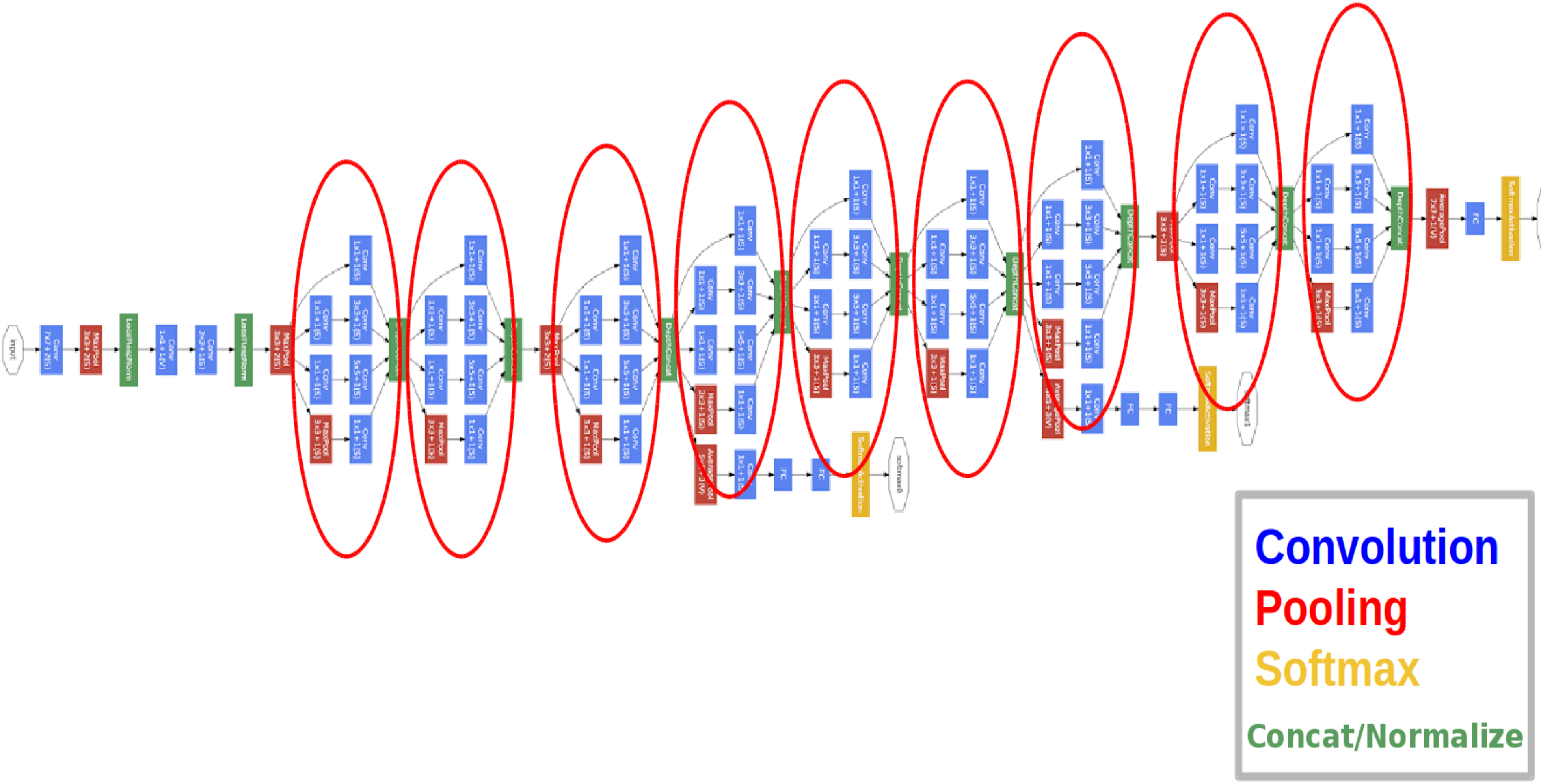
A 1x1 convolution is added before the 5x5 cvonvolution -= Also called a **bottleneck layer**



Total number of calculations = [(28 * 28 * 16) * (1 * 1 * 192)] + [(28 * 28 * 32) * (5 * 5 * 16)]

= 12.4 Million !! (earlier the cost was 120 Million)

GOOGLENET



LeNet-5, AlexNet, VGG-19, GoogLeNet for MNIST Dataset

http://euler.stat.yale.edu/~tba3/stat665/lectures/lec18/notebook18.html

TRANSFER LEARNING

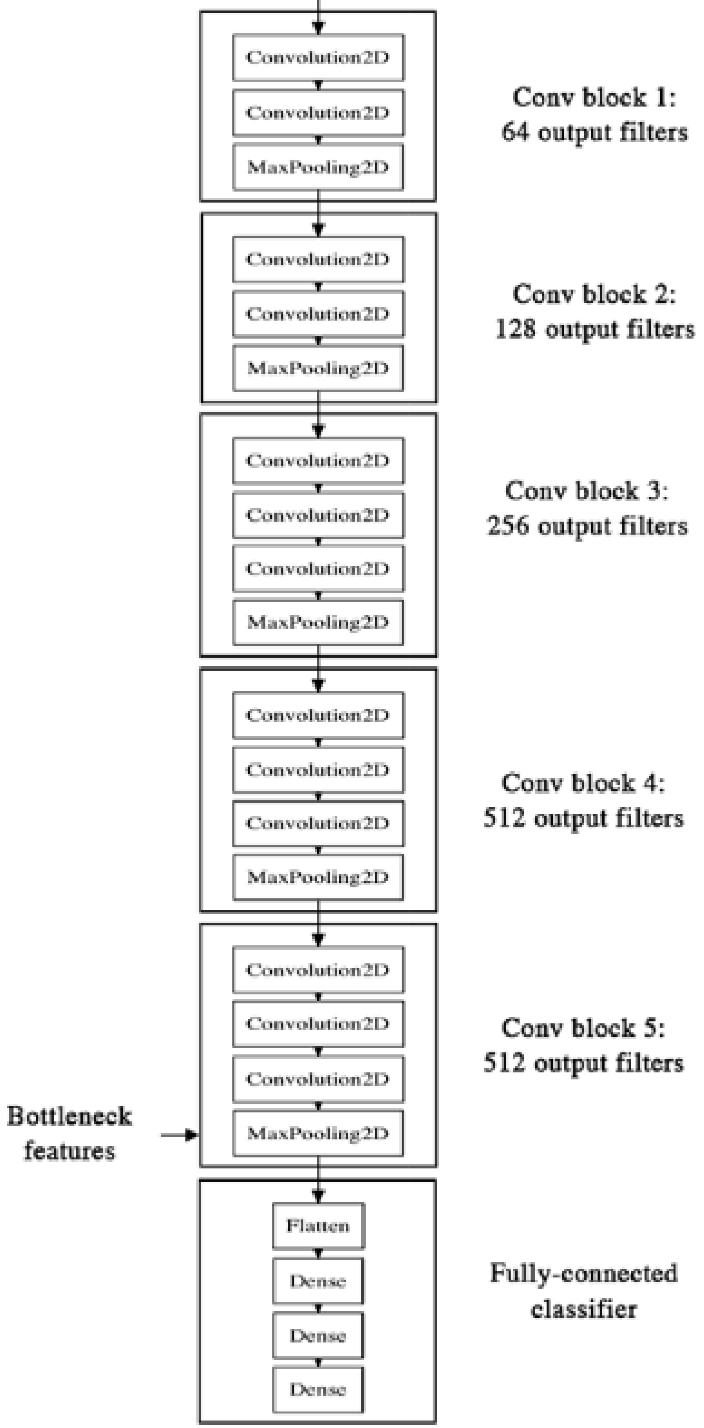
Transfer learning (TL) is **a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem**.

-Wikipedia

TRANSFER LEARNING

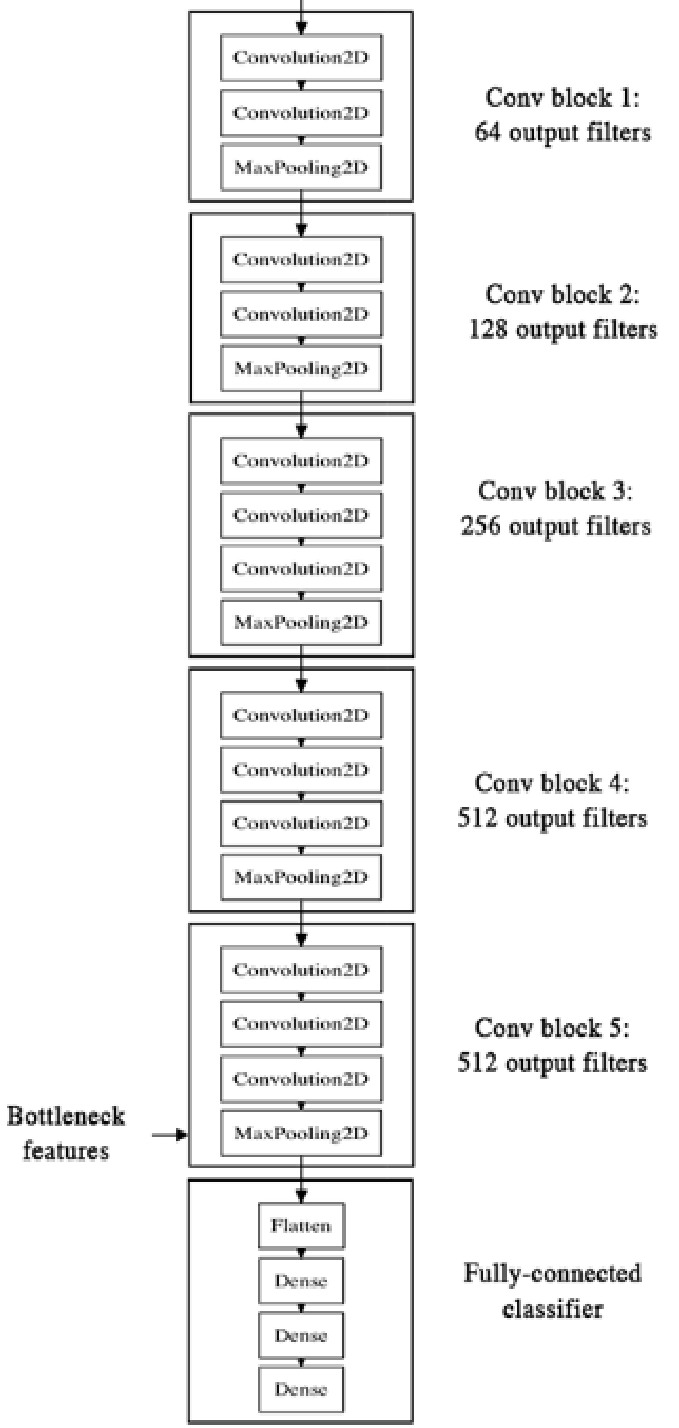
https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

- VGGNet Trained using IMAGENET



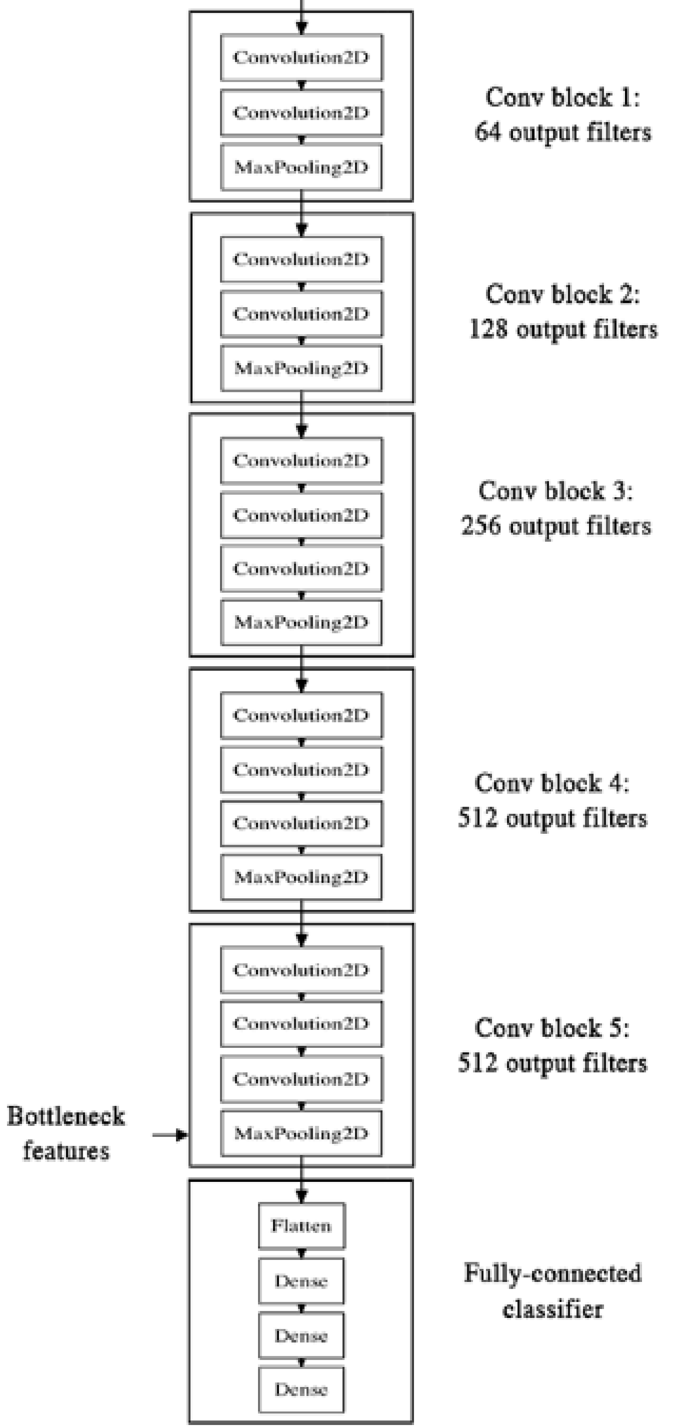
TRANSFER LEARNING(CASE-0)

- **Pre-Trained CNN Models**
  - **Use the Pre-trained CNN to predict the new dataset**



TRANSFER LEARNING(CASE-1)

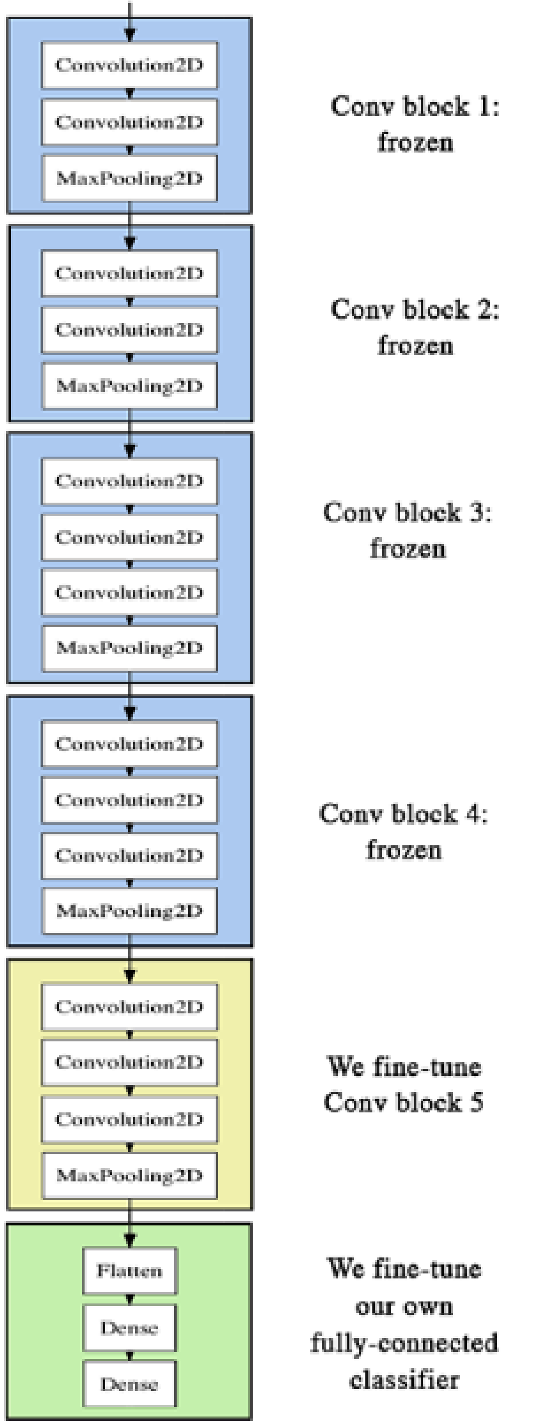
- Pre-Trained CNN Models + ML Classifiers
  – Remove the last Dense Connected Layer
  – Take Bottleneck features and use it on a Shallow ML Classifiers
  – Here Pre-Trained CNN is used for Feature Engineering



TRANSFER LEARNING (CASE-2)

- **Fine Tuning the last two layers of CNN**
  - **Take the Original Dataset and Pretrained CNN Models**
  - **Freeze the early layers (Don't Change)**
  - **Fine tune the last two layers using the new dataset**

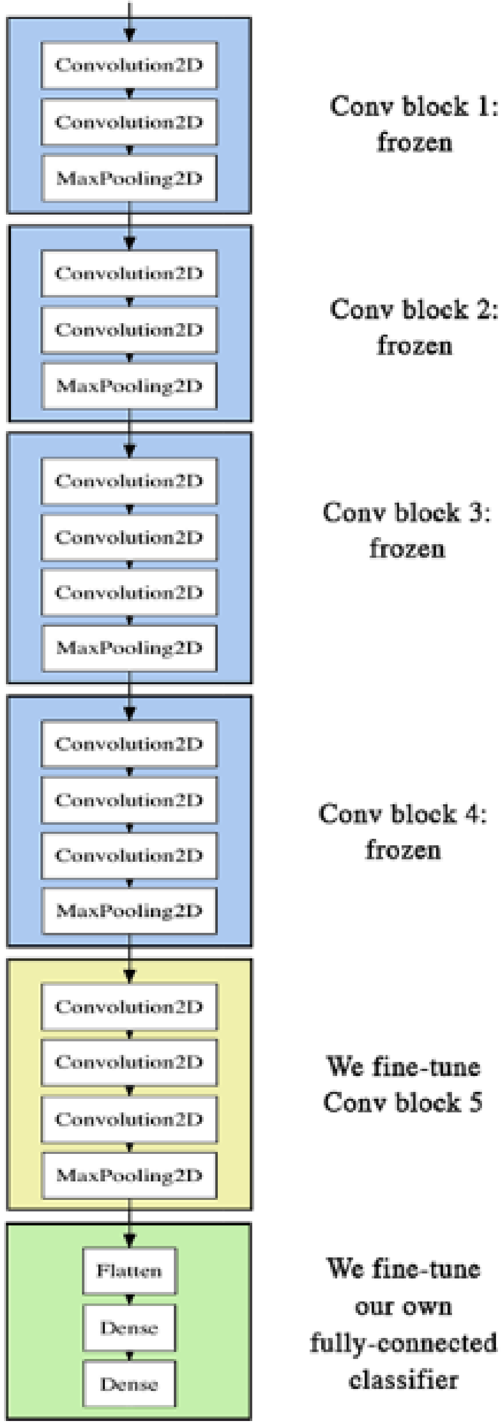
While fine tuning ensure small learning rate



TRANSFER LEARNING (CASE-3)

- Fine Tuning the Complete model taking the pretrained model as initial model
  - Take the Original Dataset and Pretrained CNN Models
  - Fine tune the complete model using the new dataset

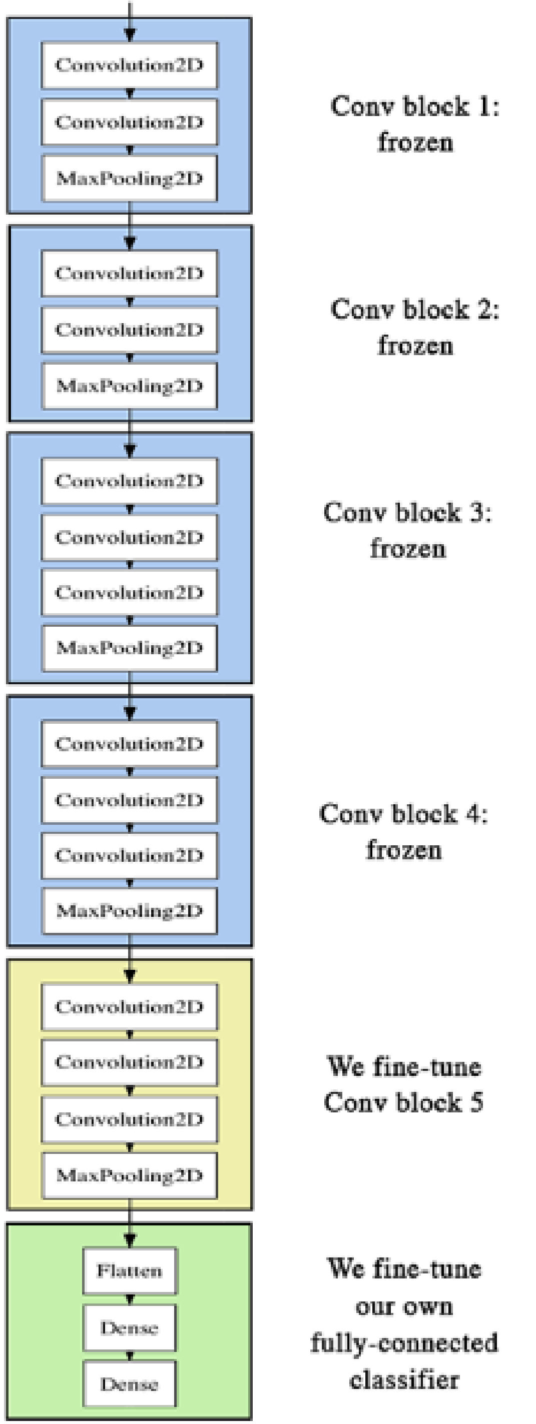
While fine tuning ensure small learning rate



TRANSFER LEARNING (CASE-4)

- **Dump everything & Retrain from Scratch**

Not widely used



How to Choose the Type of Transfer Learning

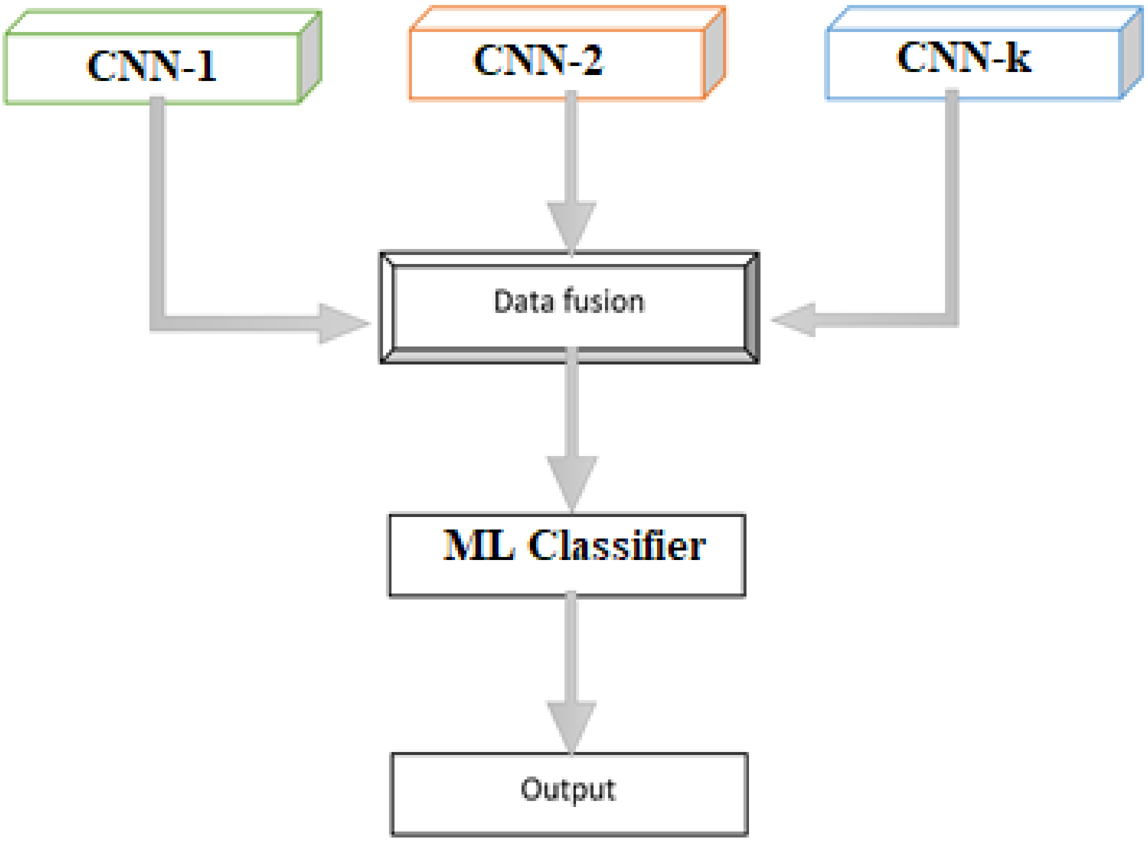
- **Based on:**
  - Size of the Dataset
  - Characteristic of the new dataset to the Imagenet Dataset

https://cs231n.github.io/transfer-learning/

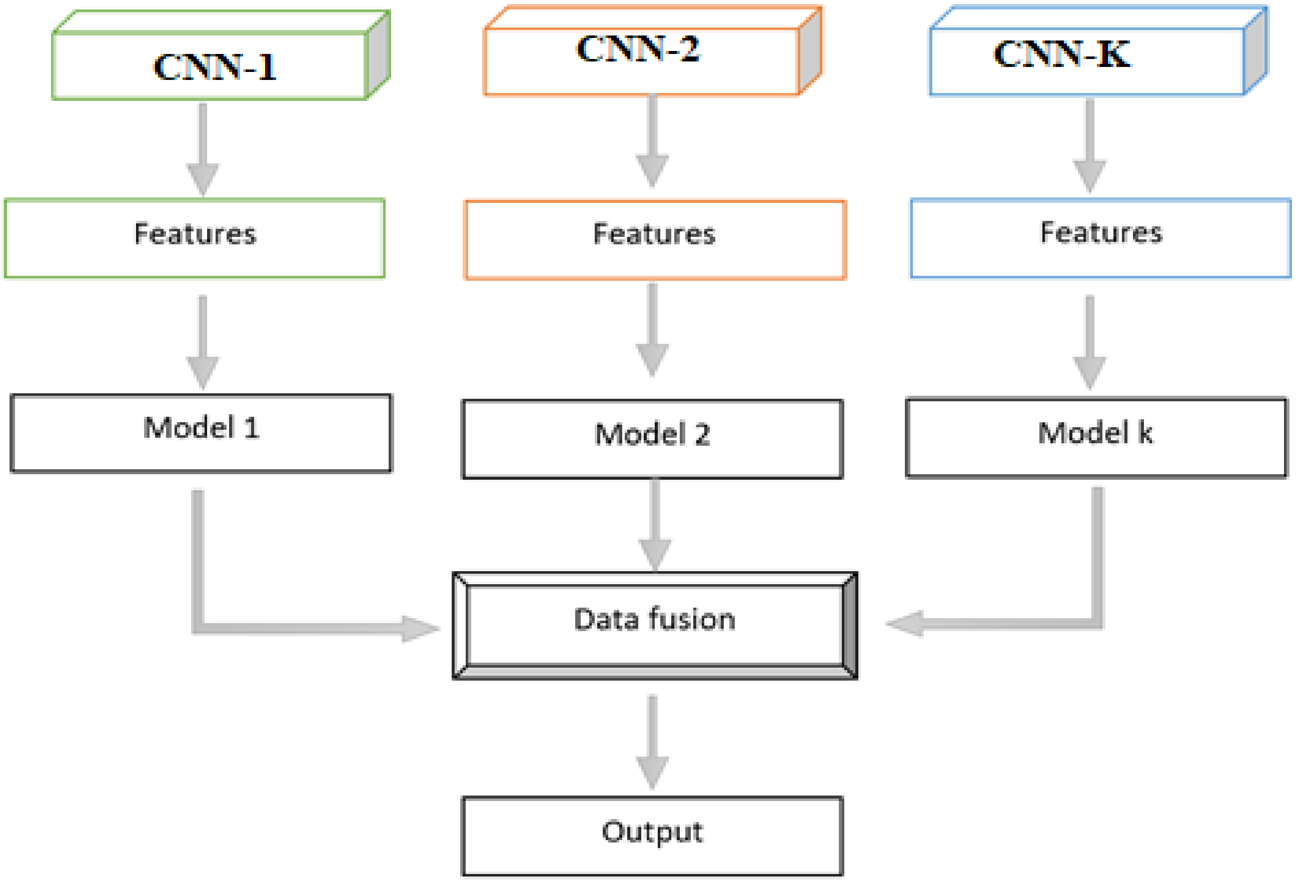
How to Choose the Type of Transfer Learning

- Case:1
  - If Size(Dataset)=Small and Similar to (IMAGENET)
    - Use Case-1 of Transfer Learning
- Case:2
  - If Size(Dataset)=Large and Similar to (IMAGENET)
    - Use Case-3 of Transfer Learning
- Case:3
  - If Size(Dataset)=Medium and Similar to (IMAGENET)
    - Use Case-2 of Transfer Learning
- Case:4
  - If Size(Dataset)=Small and DisSimilar to (IMAGENET)
    - Use Initial Layers and dump middle layers and use flatten and train a Shallow ML model.
- Case:5
  - If Size(Dataset)=Large and DisSimilar to (IMAGENET)
    - Initialize the Model using pretrained CNN and tune the whole model.

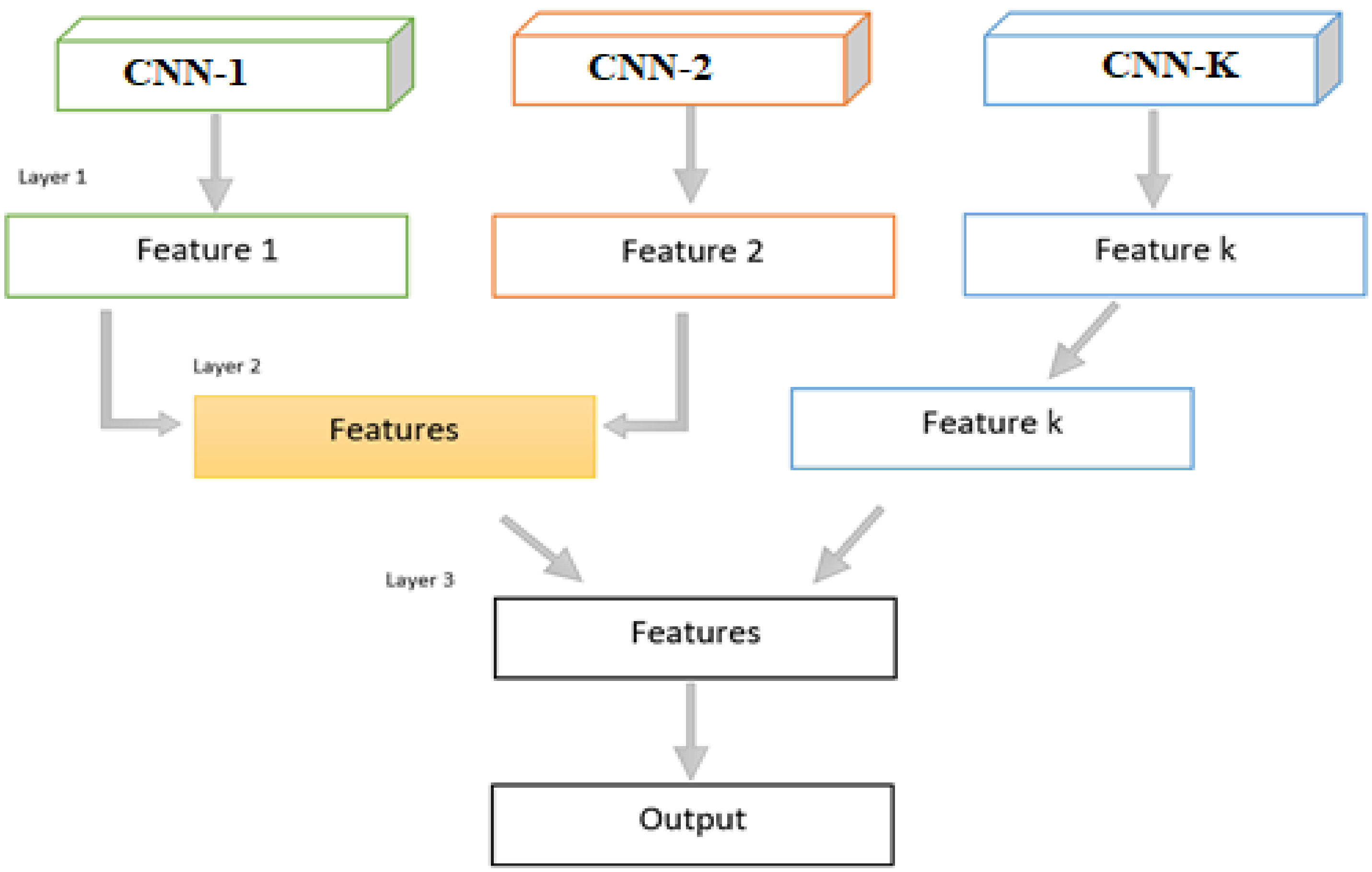
EARLY FUSION (OR FEATURE LEVEL FUSION)



LATE FUSION (OR DECISION LEVEL FUSION)



INTERMEDIATE FUSION



For Your Valuable Time.