Deep Learning with Convolutional Neural Networks

Anoop K

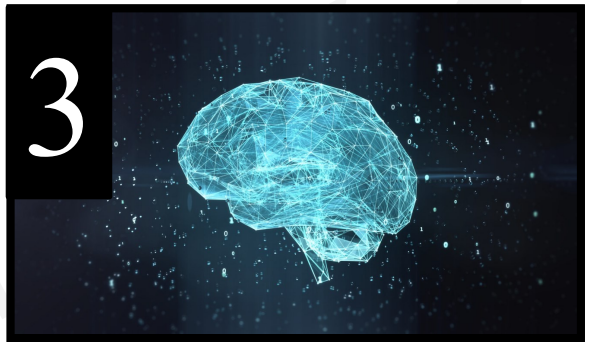
Research Scholars
Affective Computing & Machine Learning
Computational Intelligence & Data Analytics Labs
Department of Computer Science
University of Calicut, Kerala

anoopk_dcs@uoc.ac.in
https://dcs.uoc.ac.in/~anoop
https://github.com/anoopkdcs
YouTube: Notes2Learn

Plan for this Lecture

A brief history of
deep learning

Artificial Neural Network
Foundations



Convolutional
Neural Networks

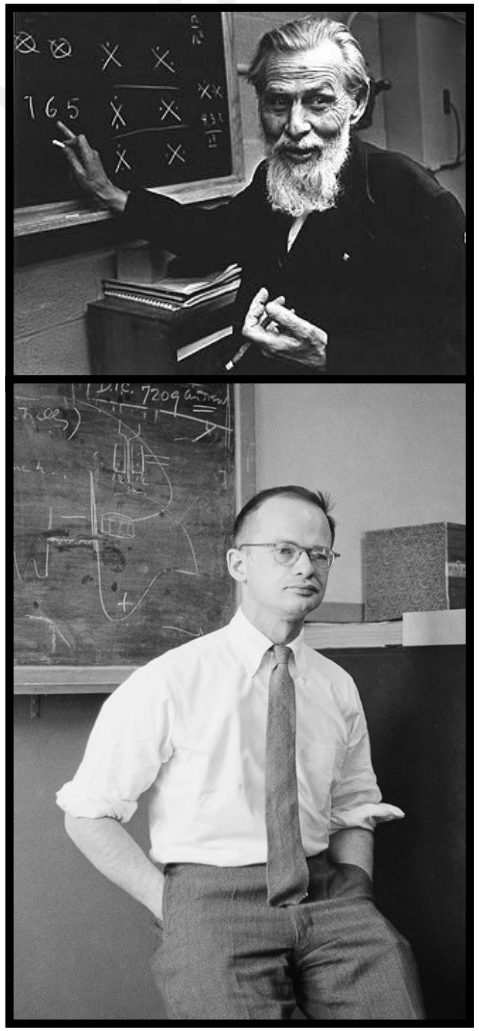
Bridging
Theory & Practice


How was I born?
Ok! Let us talk about Neural Networks first

A Brief History of Deep Learning

First Peak : 1940s

- Warren Sturgis **McCulloch** - American neurophysiologist and cybernetician
- Walter **Pitts** - Logician worked in the field of computational neuroscience
- ANN models with adjustable weights
- "*A Logical Calculus of Ideas Immanent in Nervous Activity*" in **1943**



Second peak : 1960s

- **Rosenblatt's**: Perceptron convergence theorem (*in Principles of Neurodynamics, 1962*)
- **Minsky & Papert's**:

  Perceptrons: An introduction to Computational Geometry, 1962

  Showing the limitations of a simple perceptron

  Dampened the enthusiasm of most researchers

  Especially those in the computer science community

A temporary interval of quiet or lack of activity in neural network research lasted almost **20 years**

Third Peak : 1980s

- **Hopfield**: energy approach in 1982
  - "*Neural Networks and Physical Systems with Emergent Collective Computational Abilities*"
- **Werbos**: Backpropagation learning algorithm for multilayer feedforward networks, 1974
  - "*Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*"
  - PhD thesis, Dept. of Applied Mathematics, Harvard University, Cambridge
- Rumelhart, **Hinton** & Williams: Renewed interest in backpropagation
  - Parallel Distributed Processing: Exploration in the Microstructure of Cognition, 1986

Until the mid 1980s

- Earliest days of pattern recognition, the aim of researchers has been to **replace hand engineered features** with trainable multilayer networks
- But despite its simplicity, the solution was not widely understood until the **mid 1980s**.

In the late 1990s

- Neural nets and backpropagation were **largely forsaken** (abandoned) by the machine-learning community and **ignored** by the computer-vision and speech-recognition communities.
- In particular, it was commonly thought that simple **gradient descent would get trapped in poor local minima**

Interest in deep feedforward networks was revived around **2006** by a group of researchers brought together by the Canadian Institute for Advanced Research (CIFAR).

- ✓ **Hinton, G**. E. "*What kind of graphical model is the brain?*", 2005
- ✓ **Hinton, G**. E., Osindero, S. & Teh, Y. W. A "*fast learning algorithm for deep belief nets*", 2006
- ✓ **Bengio, Y**., Lamblin, P., Popovici, D. & Larochelle, H. "*Greedy layer-wise training of deep networks*", 2006
- ✓ Ranzato, M., Poultney, C., Chopra, S. & **LeCun, Y. "***Efficient learning of sparse representations with an energy-based model*" 2006

- ✓ Researchers introduced unsupervised learning procedures that could create **layers of feature detectors**
- ✓ The objective in learning each layer of feature detectors was to be able to reconstruct or model the activities of feature detectors (or raw inputs) in the layer below

2018 Turing Award

Yoshua Bengio, **Geoffrey Hinton**, and **Yann LeCun** recipients of the **2018** ACM A.M. **Turing Award** for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing

- The first major application of this pre-training approach was in **speech recognition**, in 2009
- It was made possible by the advent of **GPUs** - train networks 10 or 20 times faster
- Record-breaking results on a standard speech recognition benchmark that used a **small vocabulary**
- Quickly developed to give record-breaking results on a **large vocabulary** task.

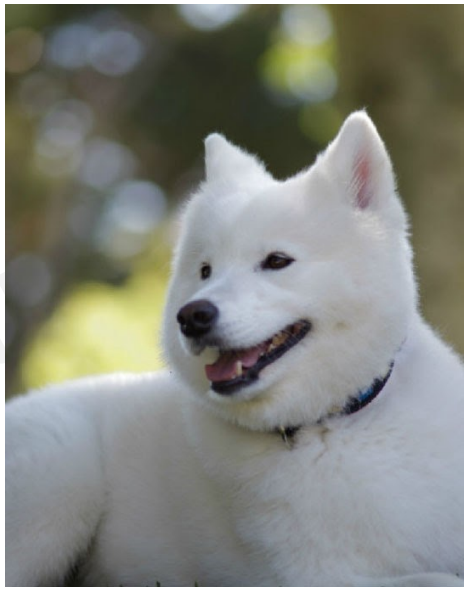
Conventional Machine Learning Vs. Representation Learning

Conventional Machine Learning

- Limited in their ability to process natural data in their raw form
- Required careful engineering and considerable domain expertise to design a feature extractor
- Transform raw data into a suitable internal representation or feature vector to learn patterns in the input

But problems such as image and speech recognition require the input–output function to be **insensitive to irrelevant variations of the input**, such as variations in position, orientation or illumination of an object, or variations in the pitch or accent of speech, while being very sensitive to particular minute variations


Two wolves in different poses and in different environments

A wolf and a samoyed in the same position and on similar backgrounds

A linear classifier, or any other ‘shallow’ classifier operating on raw pixels could not possibly distinguish the latter two, while putting the former two in the same category.

require a **good feature extractor** that solves the selectivity–invariance dilemma

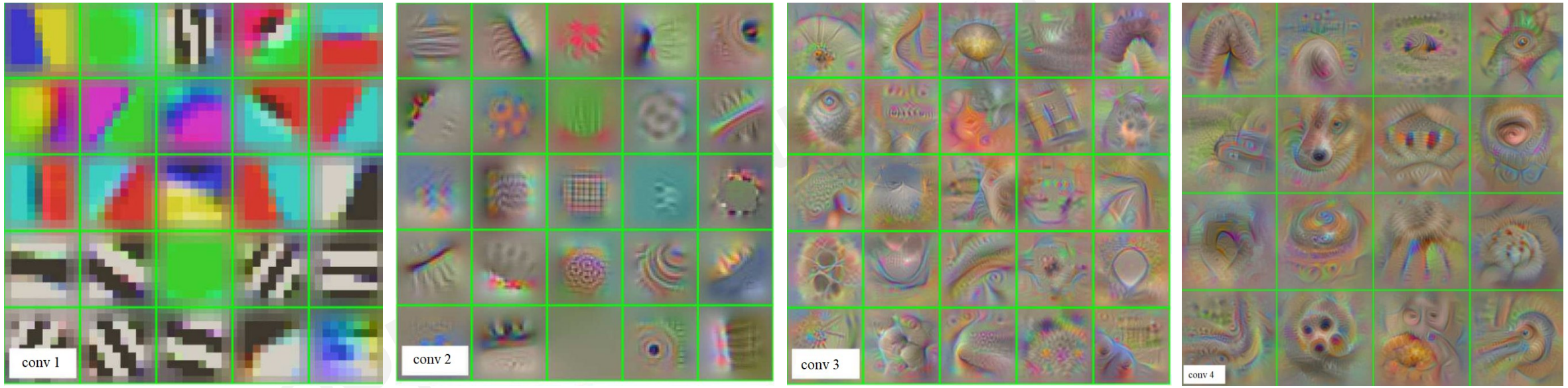
Representation Learning

- A set of methods that allows a machine to be fed with raw data
- Automatically discover the representations needed for detection or classification
- Deep-learning methods are representation-learning methods

Multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (*starting with the raw input*) into a representation at a *higher, slightly more abstract level*.

Array of pixel values like an Image:

- First layer of representation - presence or absence of edges at particular orientations and locations
- Second detects motifs (shape) by spotting particular arrangements of edges
- Third layer may assemble motifs into larger combinations that correspond to parts of familiar objects
- Subsequent layers would detect objects as combinations of these parts.



Activation maximization of the first filters of each convolutional layer in VGG-M

Mahendran, A., & Vedaldi, A. (2016).
Visualizing deep convo*nternat*lutional neural networks using natural pre-images. *lional Journal of Computer Vision*, *120*(3), 233-255.

Artificial Neural Network Foundations

Dog or Muffin ?

Classification problem

Feature Extraction

$x^{(1)}$

$x^{(2)}$

$x^{(3)}$

$x^{(4)}$

Feature Vector

$x \in \mathbb{R}^n$

Feature Mapping

$y^{(1)}$

$y^{(2)}$

$y^{(3)}$

$y^{(4)}$

Feature Label

$y \in (0,1)$

Muffin

Dog

Muffin

Dog

Feature Vector to Labels

Feature Extraction

$x^{(1)}$

$x^{(2)}$

$x^{(3)}$

$x^{(4)}$

Feature Vector

$x \in \mathbb{R}^n$

Feature Mapping

$\boldsymbol{w}, \boldsymbol{b}$

$y^{(1)}$

$y^{(2)}$

$y^{(3)}$

$y^{(4)}$

Feature Label

$y \in (0,1)$

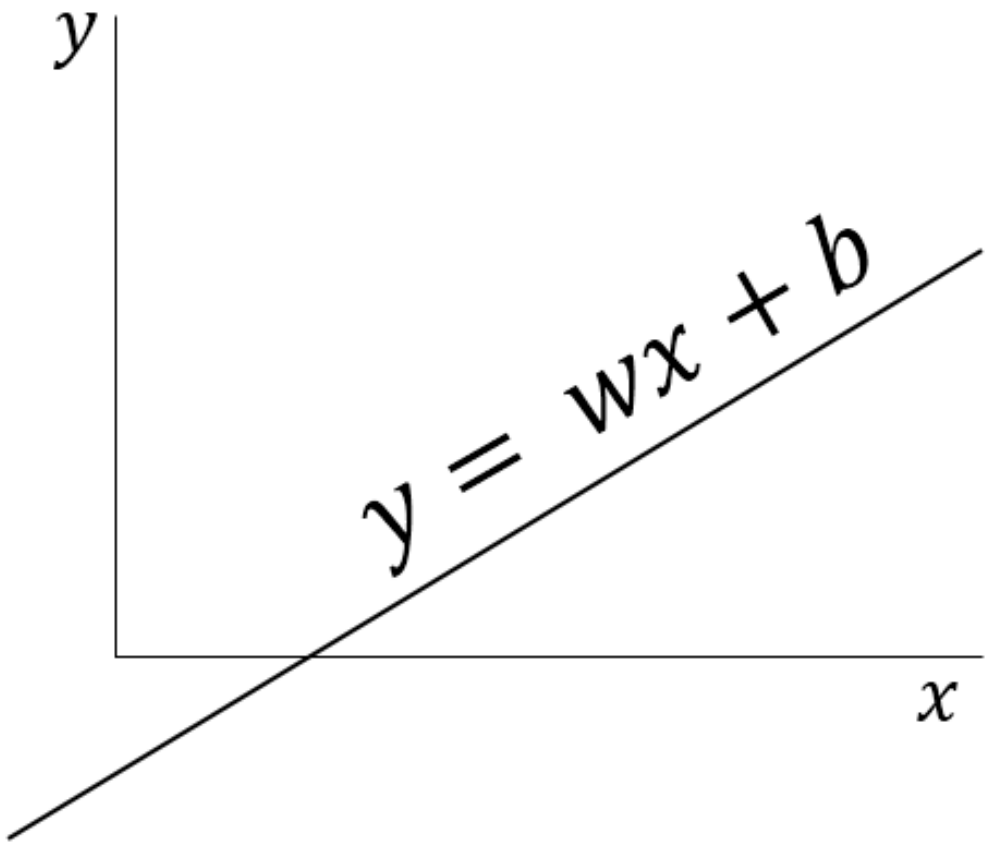
Muffin

Dog

Muffin

Dog

Linear Regression

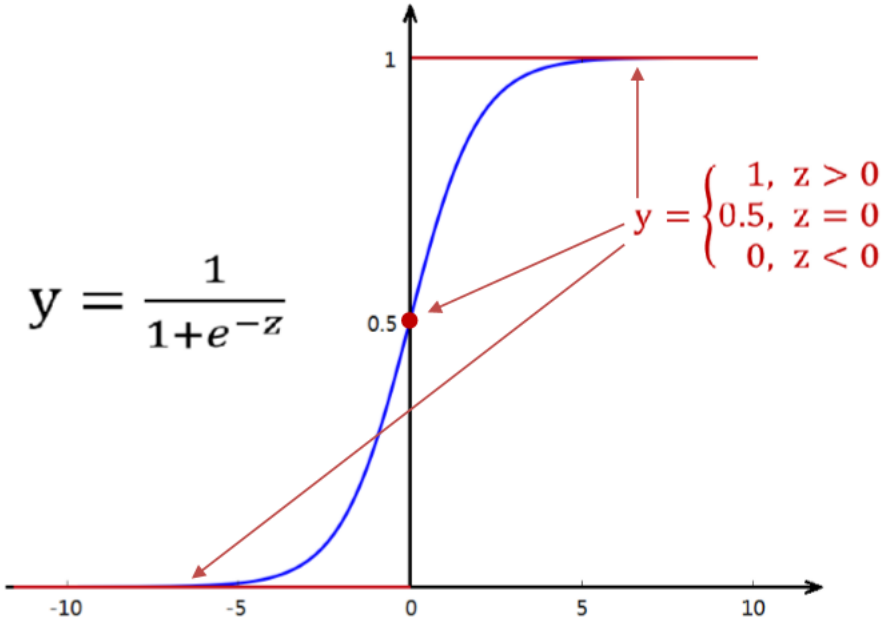


The output y can be negative, grater than zero

Logistic Regression

- $z = (wx + b)$
- $y = \sigma\,(wx + b)$
- $y = \sigma\,(z)$
- E.g. $\sigma(z) = \frac{1}{1+e^{-z}}$

Logistic Regression



If z is very large, i.e., $z \to \infty$,

then, $y = \sigma(z) = \frac{1}{1+e^{-\infty}} \to \frac{1}{1+0} \to 1$

If z is too small, i.e., $z \to -\infty$,

then, $y = \sigma(z) = \frac{1}{1+e^{--\infty}} \to \frac{1}{1+\infty} \to \frac{1}{\infty} \to 0$

$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$

z	y
Very large	1
Too small	0

Activation Functions

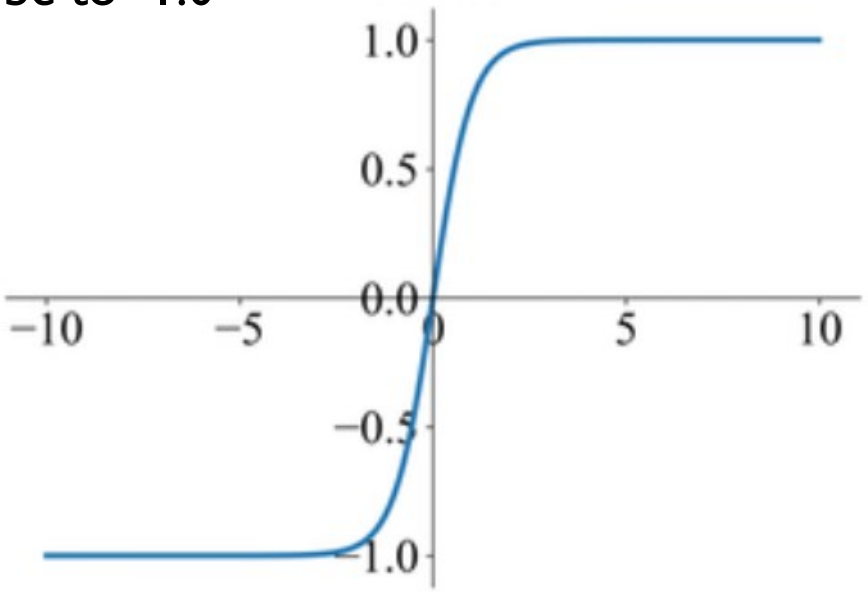
- Sigmoid : $\sigma(z) = \frac{1}{1+e^{-z}}$
- tanh : $\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
- ReLU : $\sigma(z) = \max(0, z)$
- leaky ReLU : $\sigma(z) = \max([0.01 * z], z)$
- Softmax: $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

k is number of classes in the multi-class classifier

Tanh Function (Hyperbolic Tangent)

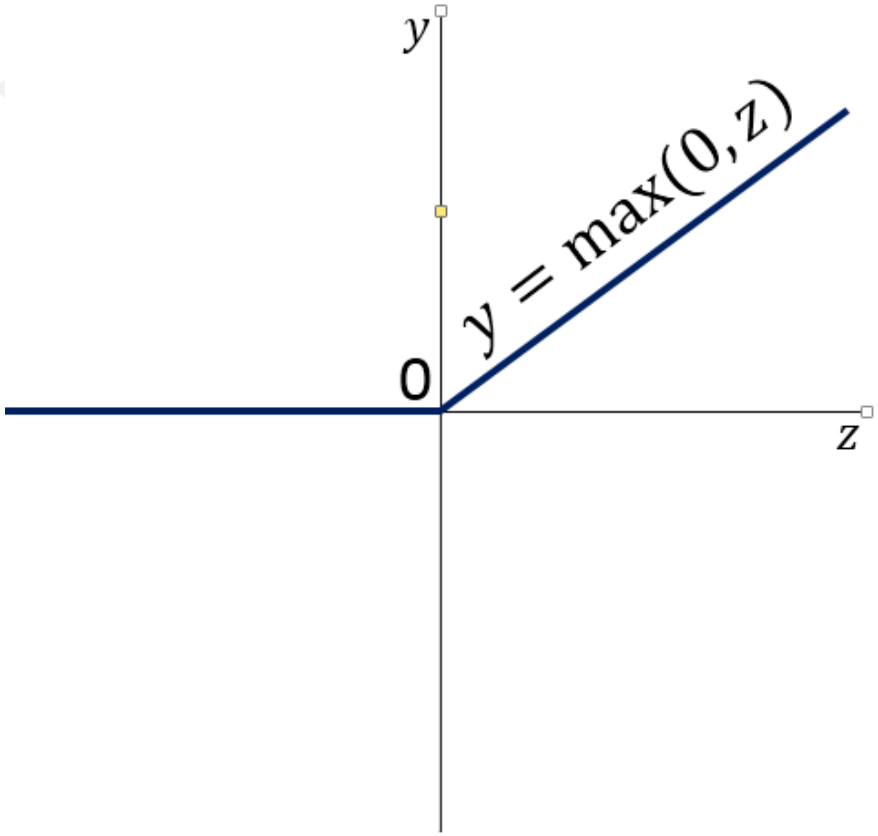
- Very similar to the sigmoid/logistic activation function
- Output range of -1 to 1.
- Larger the input (more positive), the closer the output value will be to 1.0
- Smaller the input (more negative), the closer the output will be to -1.0

$$\sigma = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Rectification Linear Unit (ReLU)

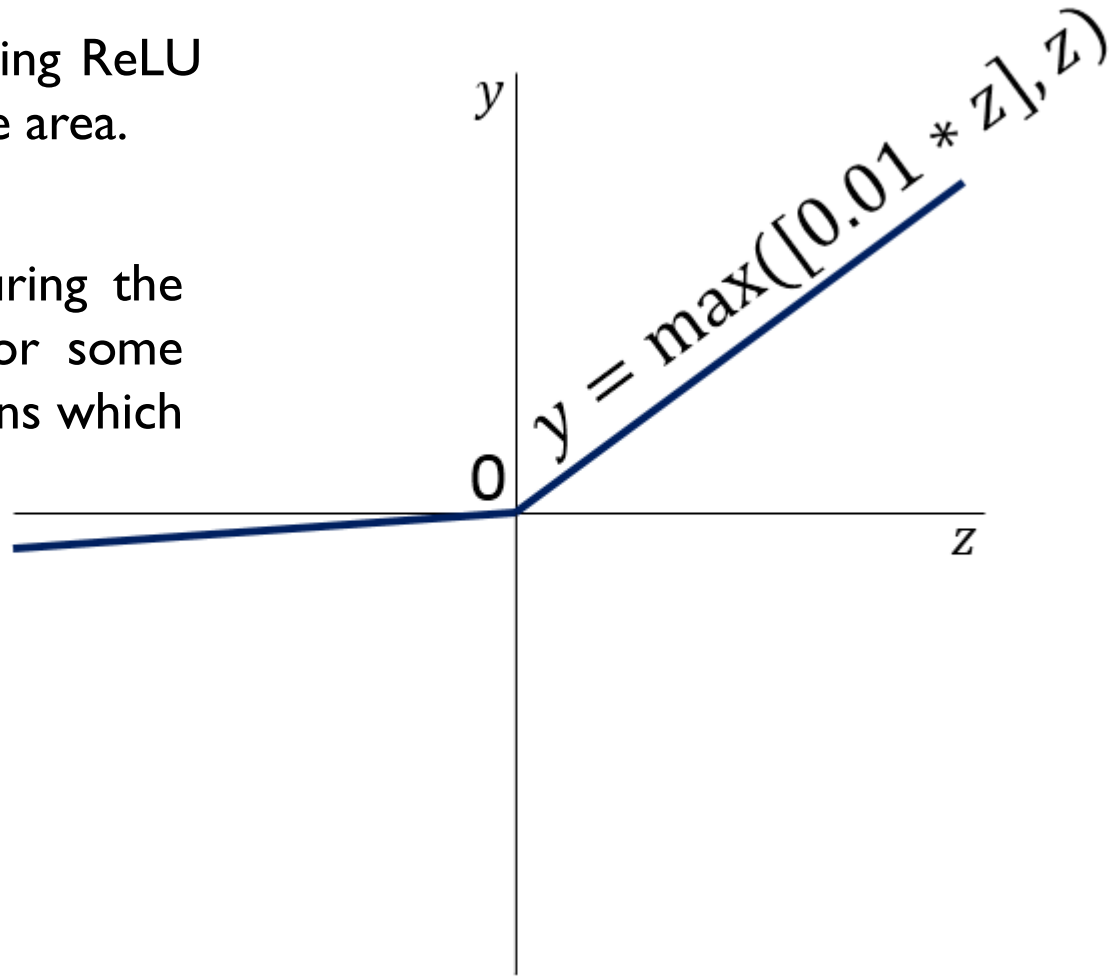
- The ReLU function does not activate all the neurons at the same time
- Neurons will only be deactivated if the output of the linear transformation is less than 0
- Computationally efficient when compared to the sigmoid and tanh functions



leaky ReLU

Improved version of ReLU function to solve the Dying ReLU problem as it has a small positive slope in the negative area.

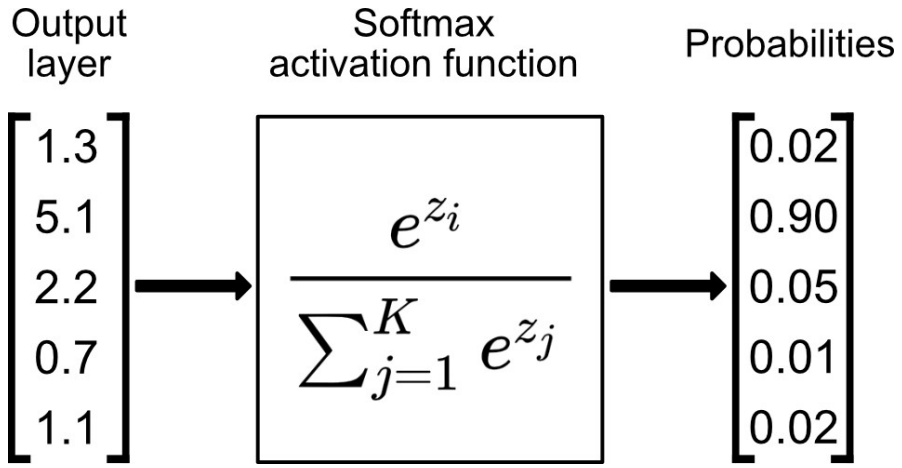
The gradient value is zero. Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated -- Dying ReLU problem



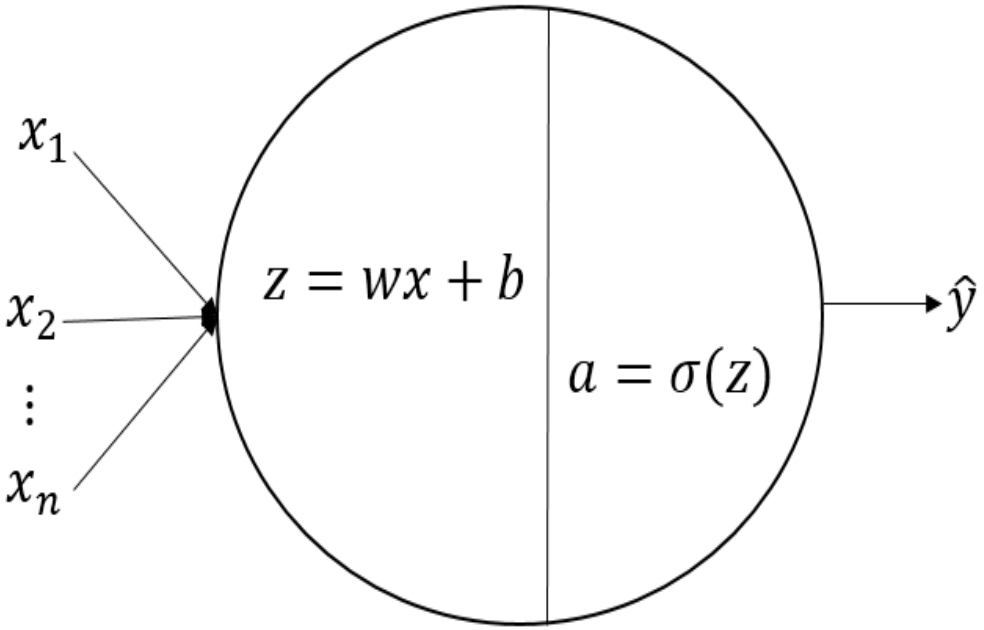
Softmax

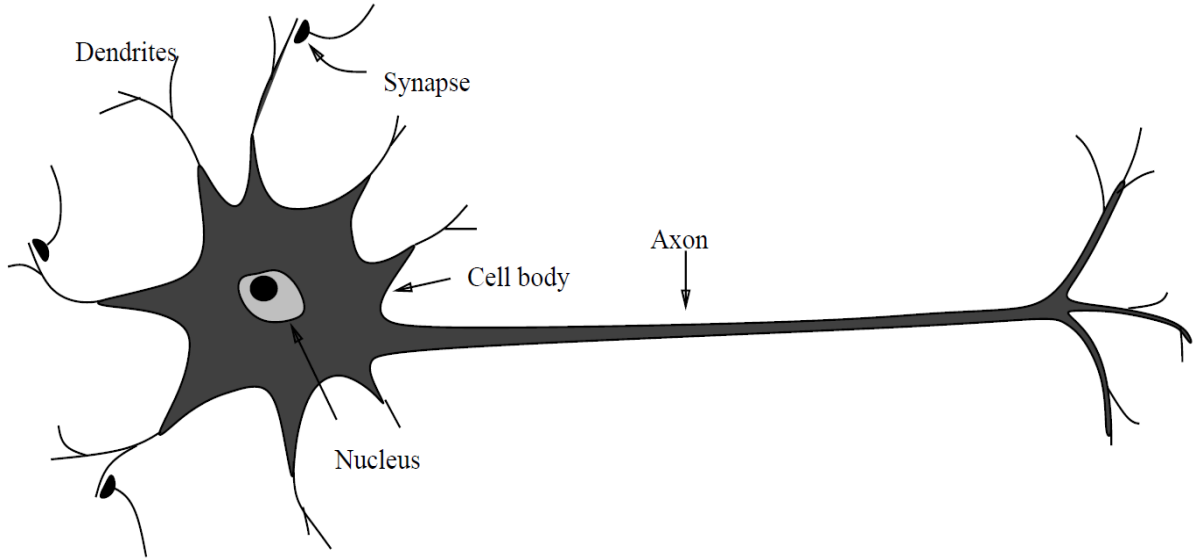
- Most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification
- Returns the probability of each class

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



Single Neuron





Label Prediction

Feature Extraction

$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$x^{(4)}$

Feature Vector
$x \in \mathbb{R}^n$

Feature Mapping

w, b

$y^{(1)}$
$y^{(2)}$
$y^{(3)}$
$y^{(4)}$

Feature Label
$y \in (0,1)$

1
1
0
1

Predicted Label
$\hat{y}$

0 : Muffin
1: Dog
0 : Muffin
1: Dog

Actual Label
y

Predicted vs. Actual Label

Feature Extraction

$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$x^{(4)}$

Feature Vector
$x \in \mathbb{R}^n$

Feature Mapping

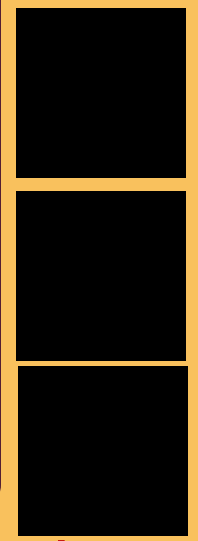
w, b

$y^{(1)}$
$y^{(2)}$
$y^{(3)}$
$y^{(4)}$

Feature Label
$y \in (0,1)$

1
1
0
1

Predicted Label
$\hat{y}$



0 : Muffin
1 : Dog
0 : Muffin
1 : Dog

Actual Label
y

Error

Feature Extraction

$x^{(1)}$

$x^{(2)}$

$x^{(3)}$

$x^{(4)}$

Feature Vector

$x \in \mathbb{R}^n$

Feature Mapping

w, b

$y^{(1)}$

$y^{(2)}$

$y^{(3)}$

$y^{(4)}$

Feature Label

$y \in (0,1)$

1

1

0

1

Predicted Label

$\hat{y}$

Loss function ↔ 0 : Muffin

Loss function ↔ 1 : Dog

Loss function ↔ 0 : Muffin

Loss function ↔ 1 : Dog

Actual Label

y

Error Calculation

- Actual label : y
- Predicted label : $\hat{y}$, where, $\hat{y} = \sigma(wx + b)$
- Good Model: $\hat{y} \approx y$
- Error → Loss function : $\mathcal{L}(y, \hat{y})$

Loss Function

- Squared Error : $\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$
- $\mathcal{L}(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$

Cost function

Feature Extraction

$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$x^{(4)}$

Feature Vector
$x \in \mathbb{R}^n$

Feature Mapping
w, b

$y^{(1)}$
$y^{(2)}$
$y^{(3)}$
$y^{(4)}$

Feature Label
$y \in (0,1)$

Predicted Label y

1
1
0
1

Loss function $\mathcal{L}(y, \hat{y})$

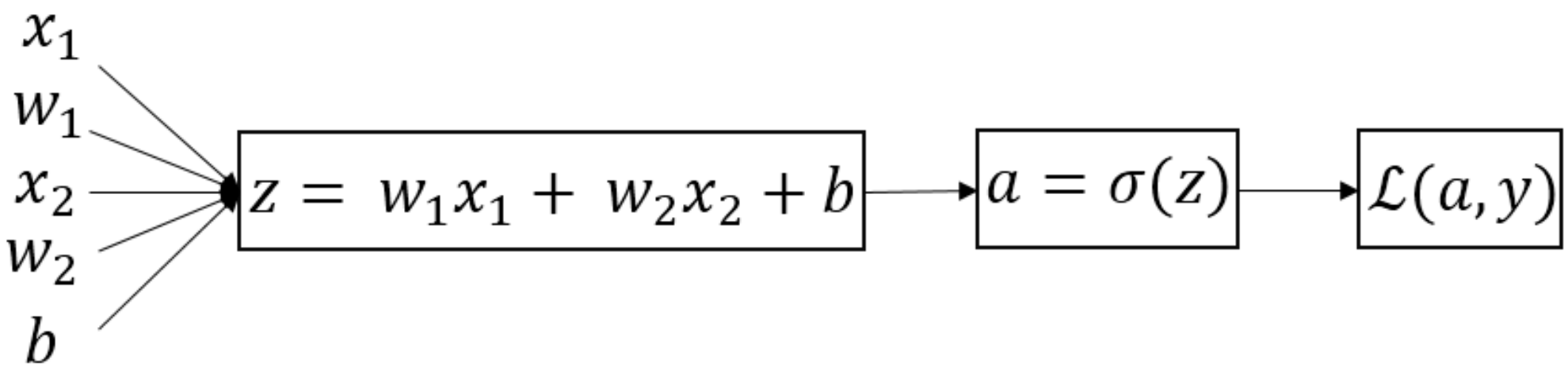
Actual Label $\hat{y}$

0
1
0
1

Cost function

$$\mathcal{J} = \frac{1}{4}\sum_{i=1}^{4} \mathcal{L}(y, \hat{y})$$

Forward Propagation



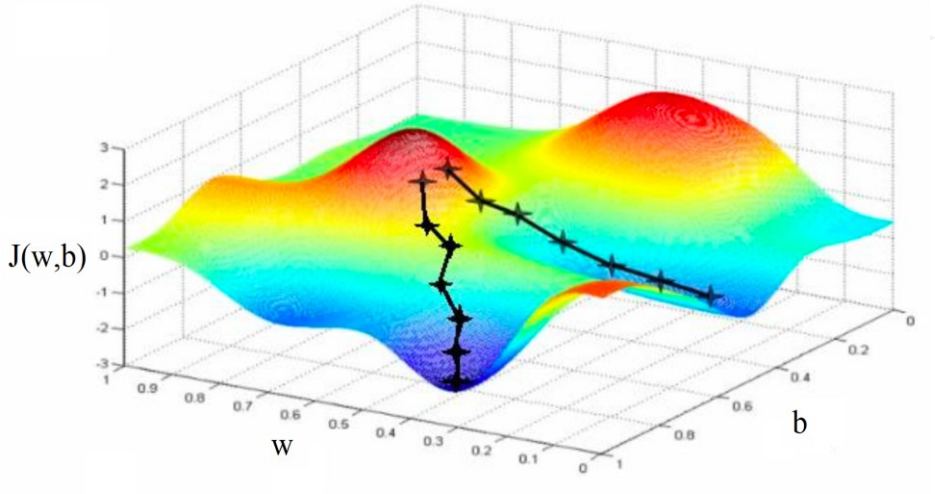
Gradient Descent

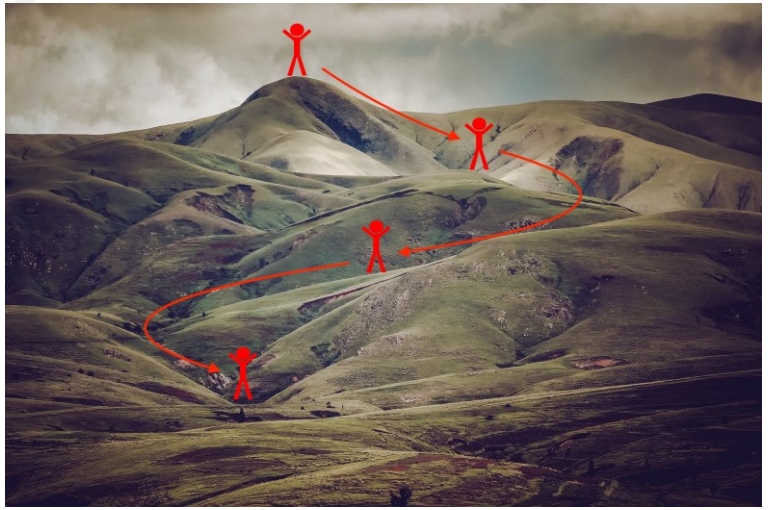
An optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

$\mathcal{L}$: loss function

$\mathcal{J}$: cost function, aggregate of all loss of y(i)

Aim: Find w, b that minimize (global minima) $\mathcal{J}(w, b)$

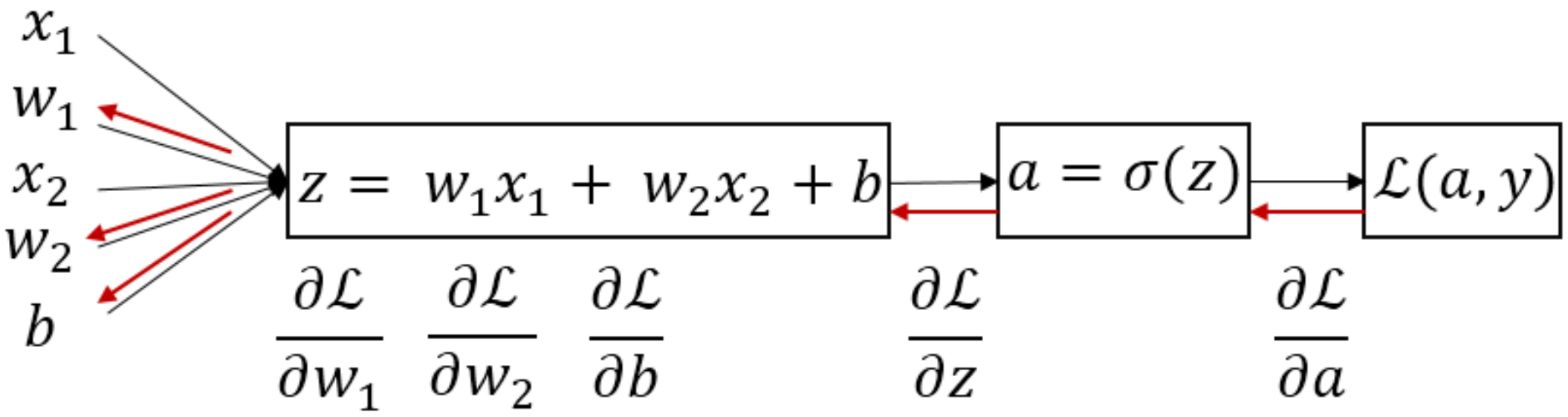


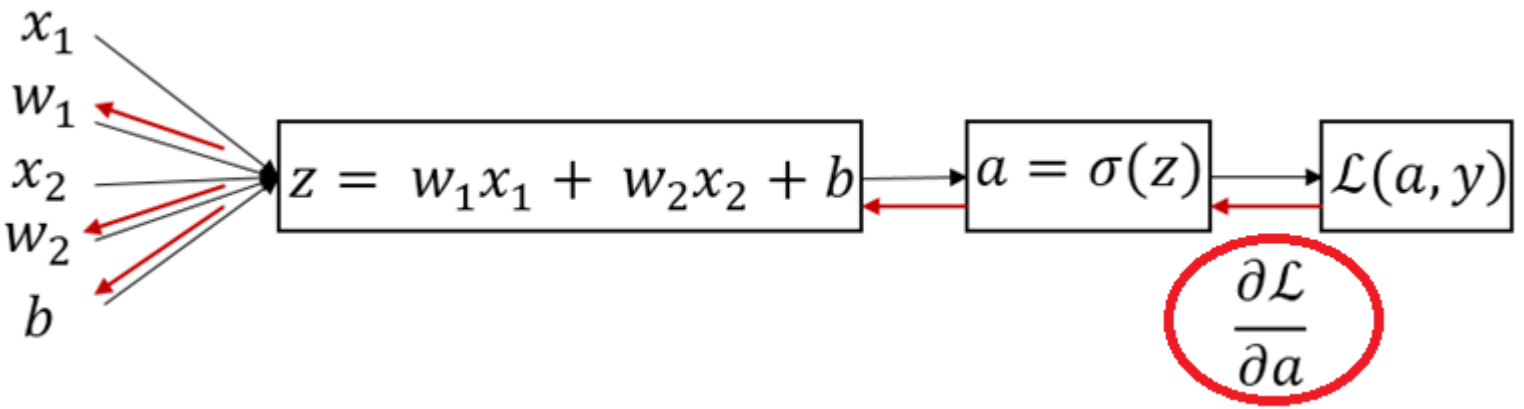


$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

Backward Propagation



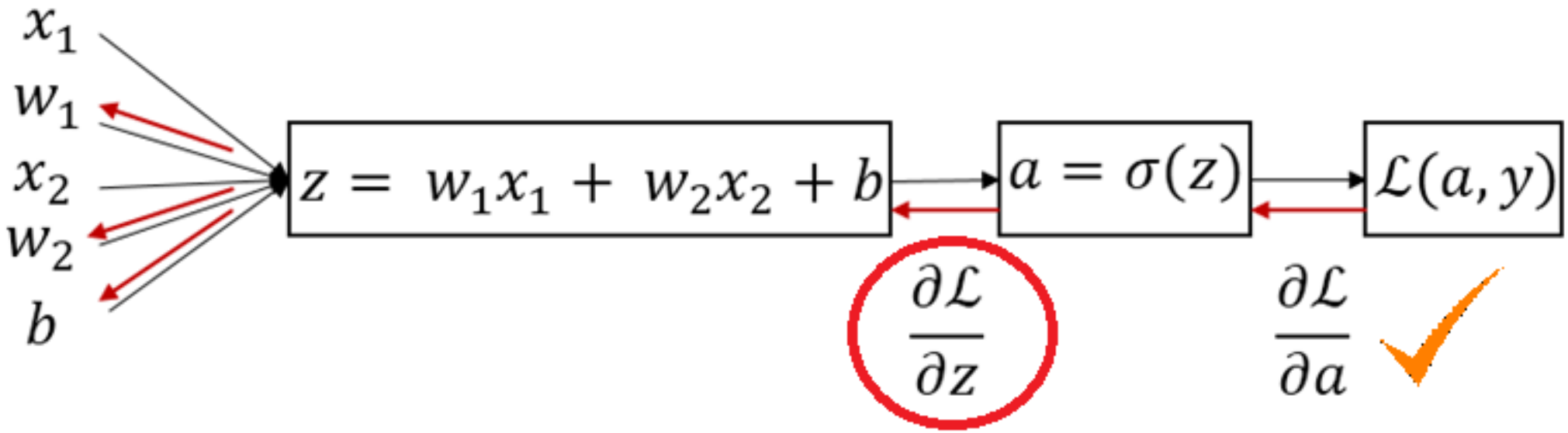


$$\mathcal{L}(a, y) = -(y \log a + (1 - y) \log(1 - a))$$

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}(a, y)}{\partial a} = \frac{\partial}{\partial a}\left(-(y \log a + (1 - y) \log(1 - a))\right)$$

$$= -\left(\frac{y}{a} + \frac{1-y}{-(1-a)}\right) = \frac{-y}{a} + \frac{1-y}{1-a}$$

Hint: $\frac{\partial}{\partial x} \log x = \frac{1}{x}$



$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} * \frac{\partial a}{\partial z}, \text{(Chain rule)}$$

$$a = \frac{1}{1 + e^{-z}}$$

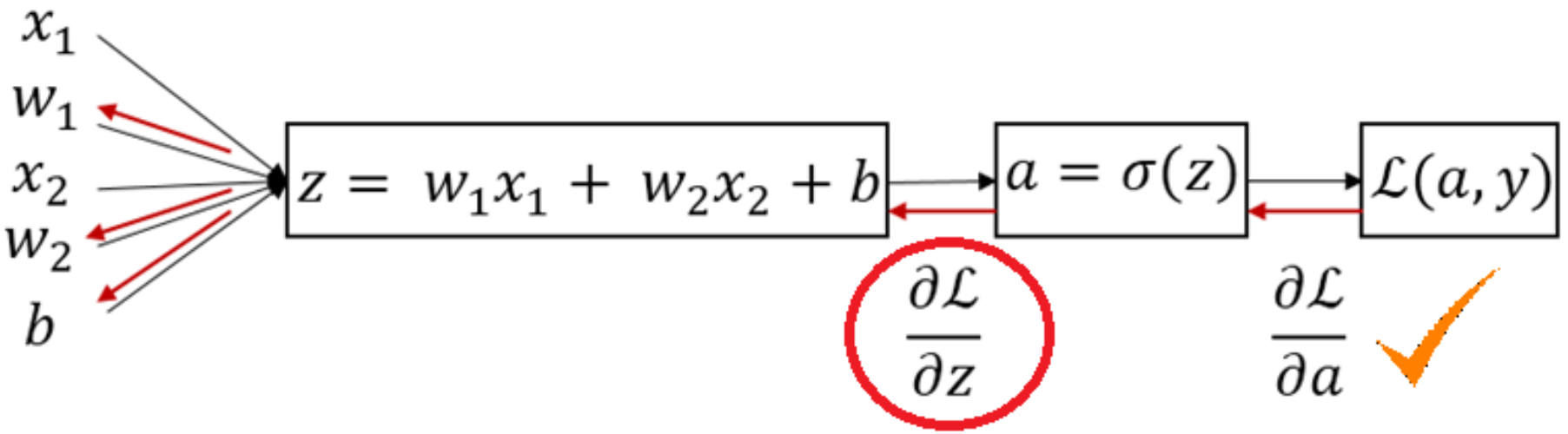
Hints: $\frac{\partial}{\partial x} e^x = e^x$

$\frac{\partial x^n}{\partial x} = nx^{n-1}$

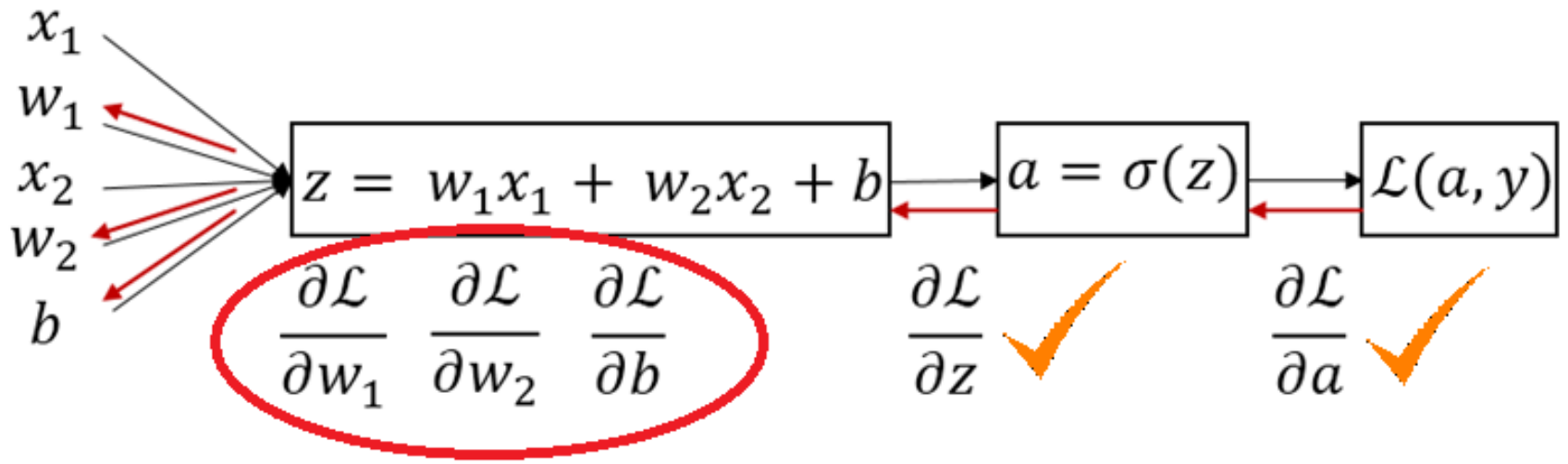
$$\frac{\partial a}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = \frac{\partial}{\partial z}(1 + e^{-z})^{-1}$$

$$= -1 * (1 + e^{-z})^{-2} * e^{-z} * -1$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} = \boldsymbol{a(1-a)} \qquad [\because a = \frac{1}{1+e^{-z}} \rightarrow e^{-z} = \frac{1-a}{a}]$$



$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} * \frac{\partial a}{\partial z} = \left(\frac{-y}{a} + \frac{1-y}{1-a}\right) * a(1-a) = \underline{a - y}$$



$$\frac{\partial \mathcal{L}}{\partial w_1} = \left(\frac{\partial \mathcal{L}}{\partial z} * \frac{\partial z}{\partial w_1}\right) = \left(\frac{\partial \mathcal{L}}{\partial a} * \frac{\partial a}{\partial z}\right) * \frac{\partial z}{\partial w_1} = (a - y) * \frac{\partial z}{\partial w_1}$$

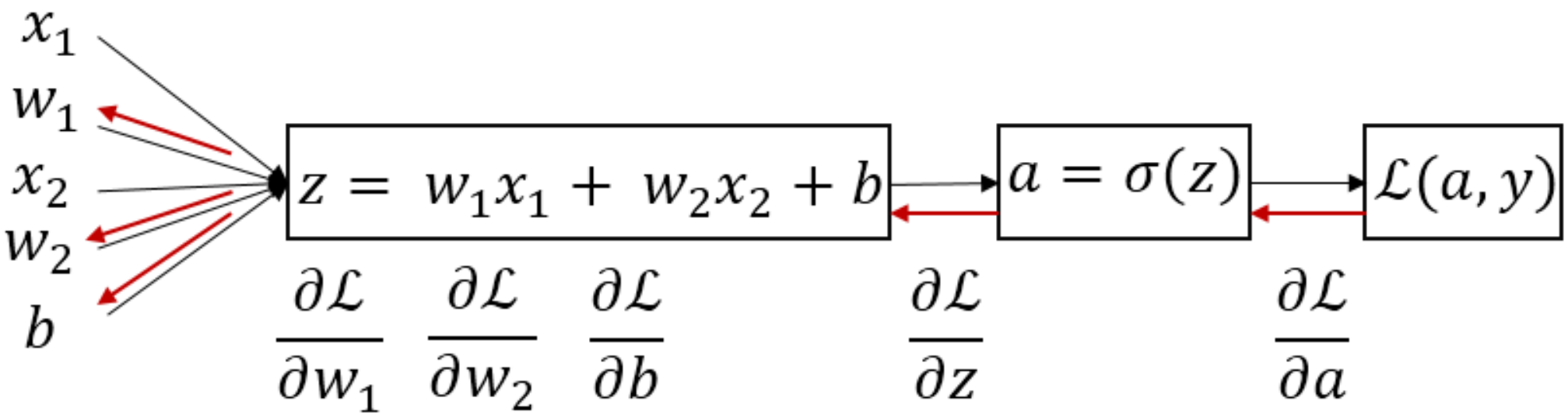
$$\frac{\partial \mathcal{L}}{\partial b} = \left(\frac{\partial \mathcal{L}}{\partial z} * \frac{\partial z}{\partial b}\right) = \left(\frac{\partial \mathcal{L}}{\partial a} * \frac{\partial a}{\partial z}\right) * \frac{\partial z}{\partial b} = (a - y) * \frac{\partial z}{\partial b}$$

$$\frac{\partial z}{\partial w_1} = \frac{\partial}{\partial w_1}(w_1 x_1 + w_2 x_2 + b) = x_1$$

$$\therefore \frac{\partial \mathcal{L}}{\partial w_1} = \left(\frac{\partial \mathcal{L}}{\partial a} * \frac{\partial a}{\partial z}\right) * \frac{\partial z}{\partial w_1} = \underline{(a - y) * x_1}$$

$$\frac{\partial z}{\partial b} = \frac{\partial}{\partial b}(w_1 x_1 + w_2 x_2 + b) = 1$$

$$\therefore \frac{\partial \mathcal{L}}{\partial b} = \left(\frac{\partial \mathcal{L}}{\partial a} * \frac{\partial a}{\partial z}\right) * \frac{\partial z}{\partial b} = \underline{(a - y)}$$

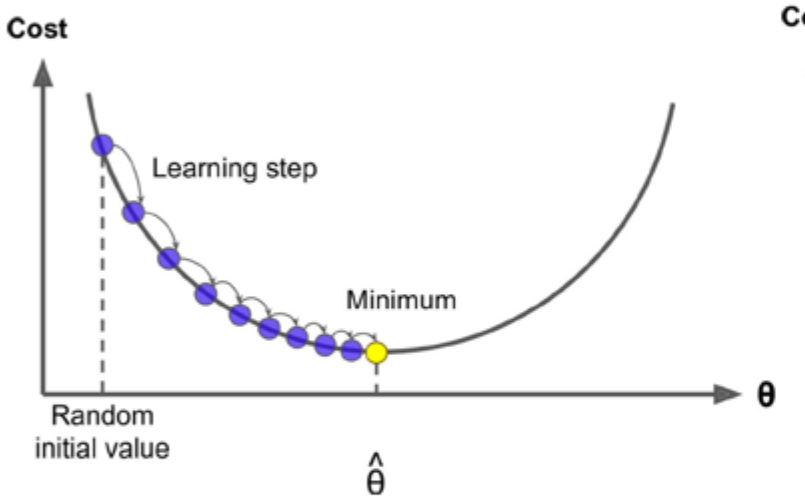


$$w_{new} := w_{old} - \alpha \frac{\partial \mathcal{L}}{\partial w_{old}}$$

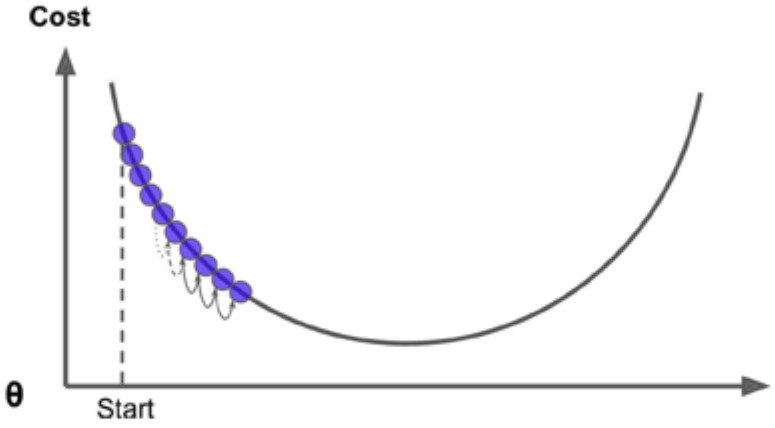
$$b_{new} := b_{old} - \alpha \frac{\partial \mathcal{L}}{\partial b_{old}}$$

Gradient Descent – Learning Rate

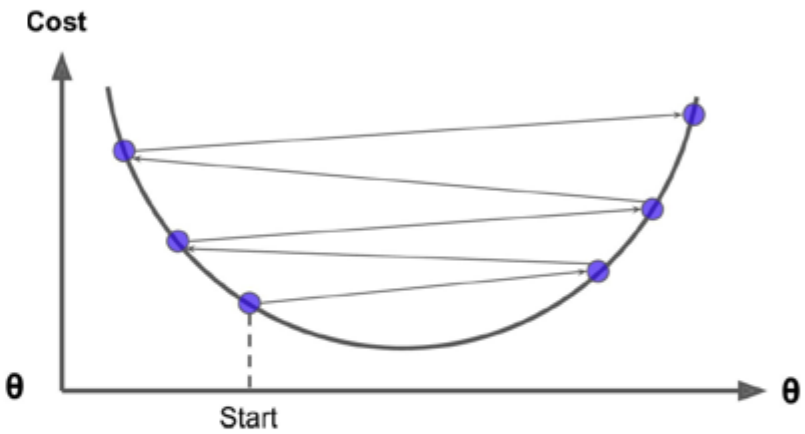
In training Regression models or Neural Networks, the key technique to arrive at optimal weights is the Gradient Descent algorithm. This algorithm relies on a hyperparameter called the ***learning rate*** which allows one to moderate the rate of weight change, such that the cost function is minimized.



Optimum learning rate : The model adjusts weights (θ) in subsequent training loops to arrive at cost minima.

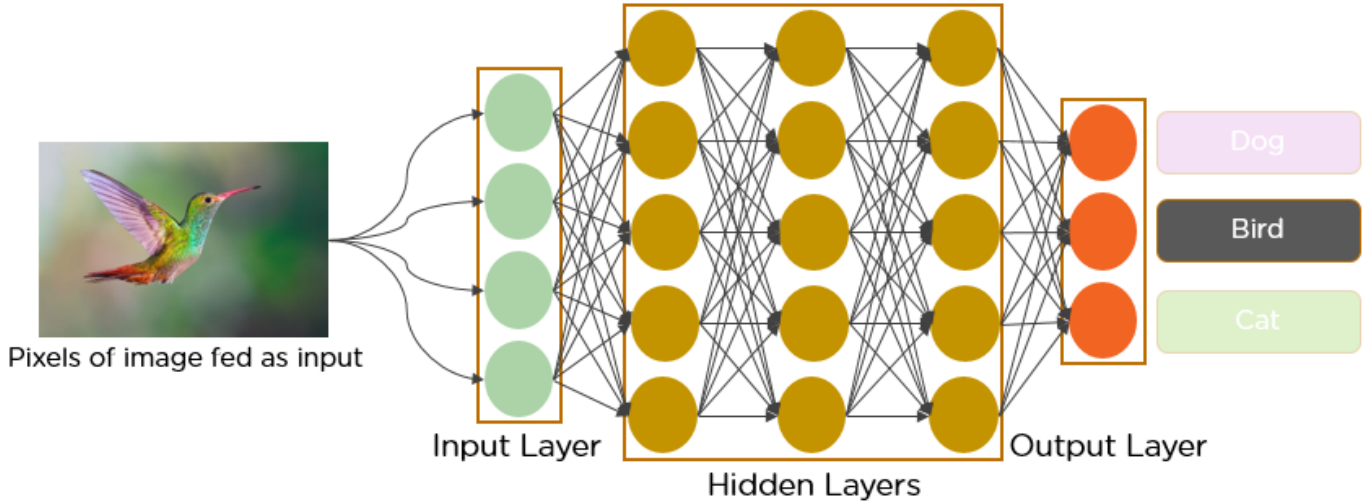


Slow learning rate : Converges to cost minima but very slowly.

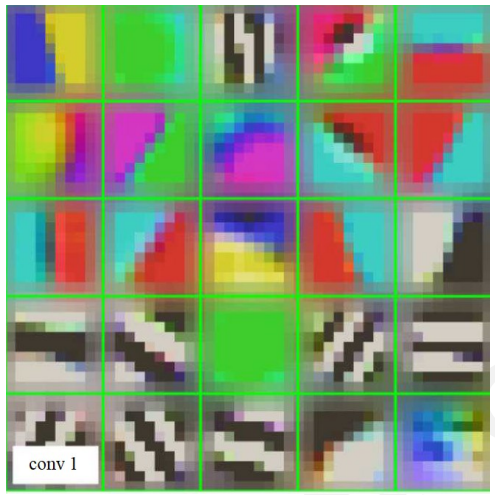


*Fast learning rate : **may** not converge to cost minima and the cost might keep increasing with further training loops.*

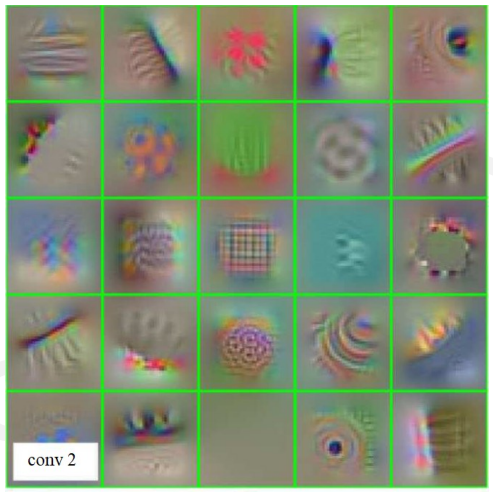
Image Credit : "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron



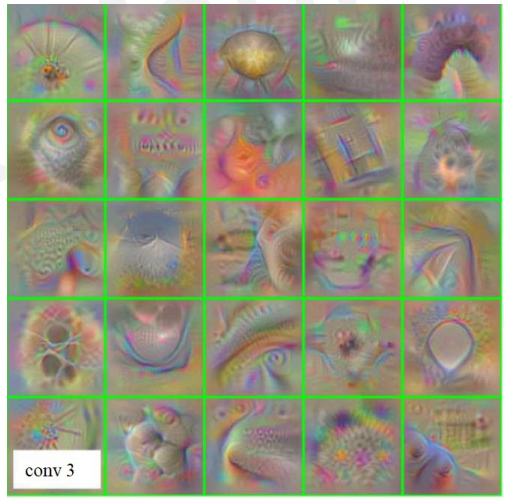
Edges

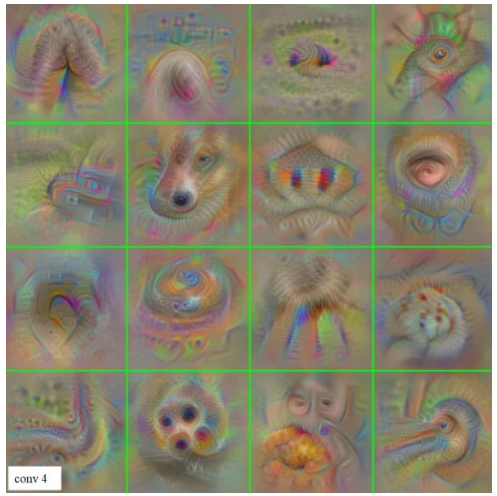


Patterns

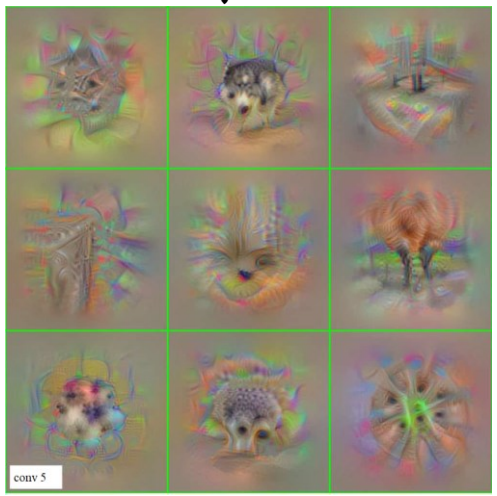


Parts like feather, eye, beak, nose, etc.

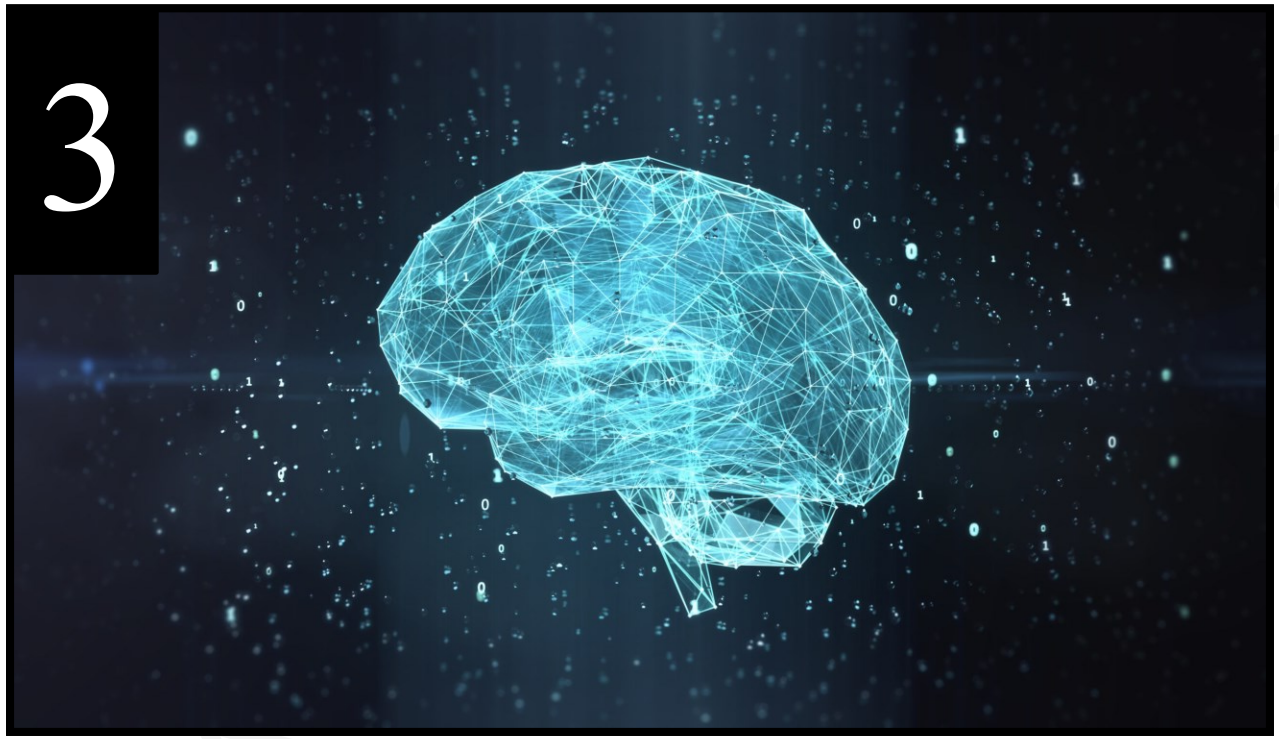




Objects

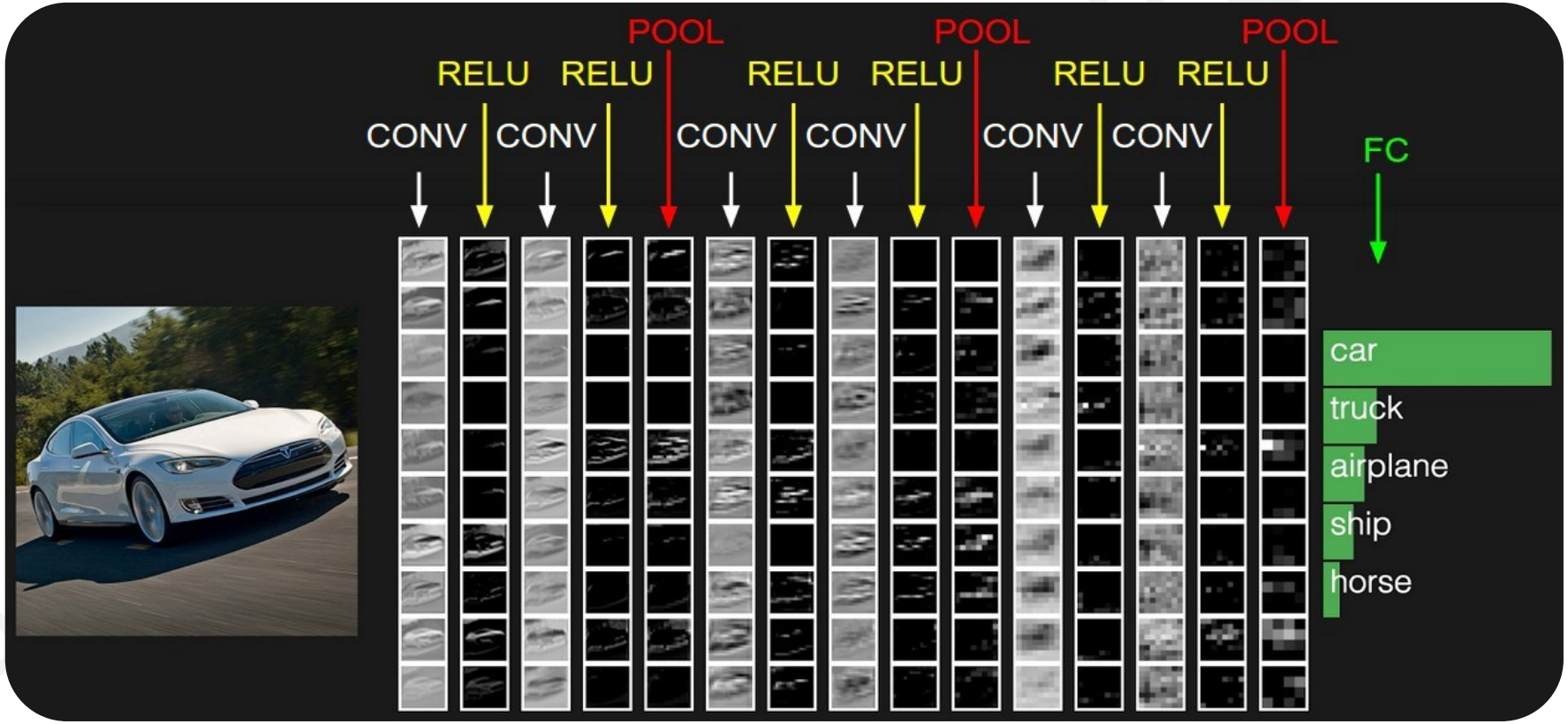


https://playground.tensorflow.org/



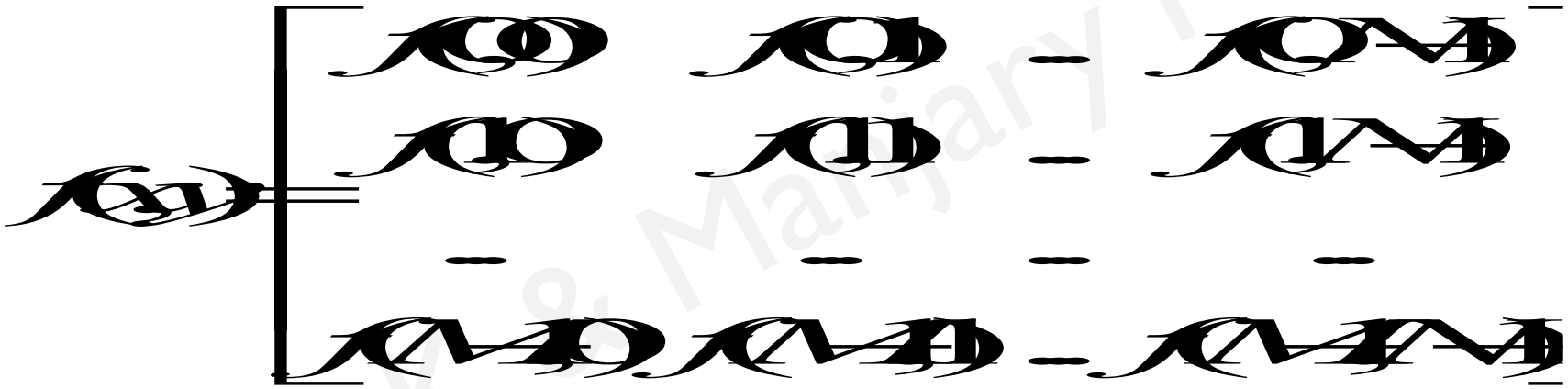
Convolutional Neural Networks

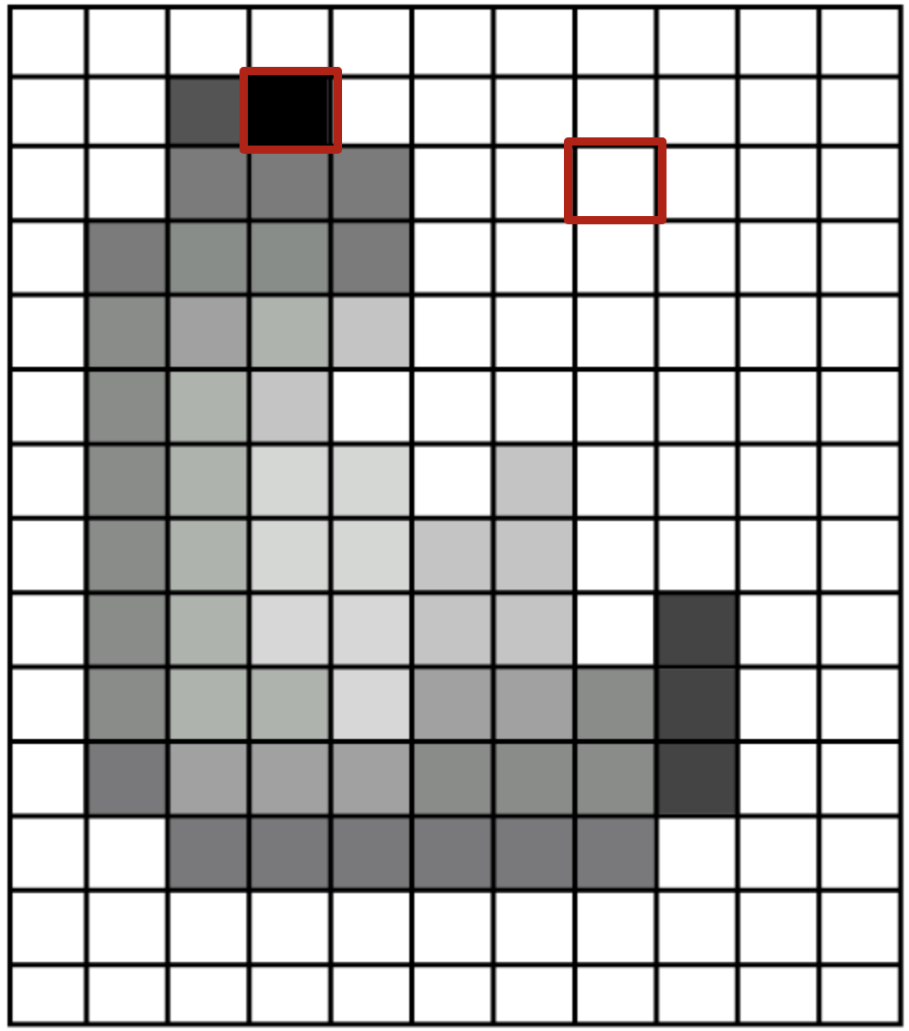
Convolutional Neural Networks



Representing Digital Images

M×N numerical array





=

255	255	255	255	255	255	255	255	255	255	255
255	255	20	0	255	255	255	255	255	255	255
255	255	75	75	255	255	255	255	255	255	255
255	75	95	95	75	255	255	255	255	255	255
255	96	127	145	175	255	255	255	255	255	255
255	127	145	175	175	175	255	255	255	255	255
255	127	145	200	200	175	175	95	255	255	255
255	127	145	200	200	175	175	95	47	255	255
255	127	145	145	175	127	127	95	47	255	255
255	74	127	127	127	95	95	95	47	255	255
255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255

0 = black; 255 = white

Convolution

Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

*

Filter

1	0	1
0	1	0
1	0	1

= Convolved Image

Convolution Cont.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

1	0	1
0	1	0
1	0	1

Filter / Kernel

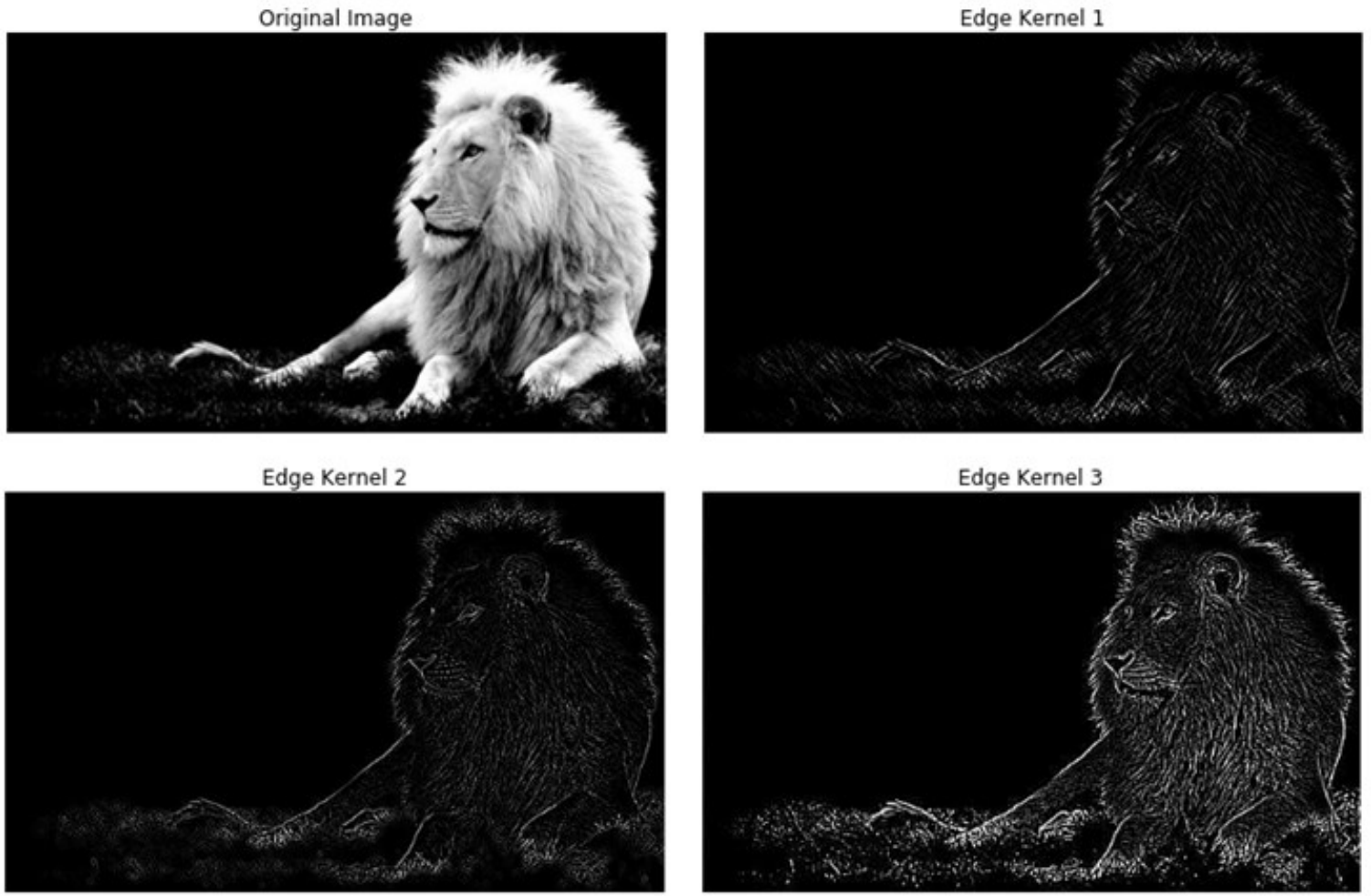
$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

Convolution Cont.



Dimension of Convolved Image

- Eg: Image (5×5) * Filter $(3 \times 3) =$ Convolved image (3×3)
- Eg: Image (10×10) * Filter $(3 \times 3) =$ Convolved image (8×8)
- Image $(n \times n)$ * Filter $(f \times f)$

  $=$ Convolved Image $(n - f + 1 \times n - f + 1)$

P for Padding

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

Padding Cont.

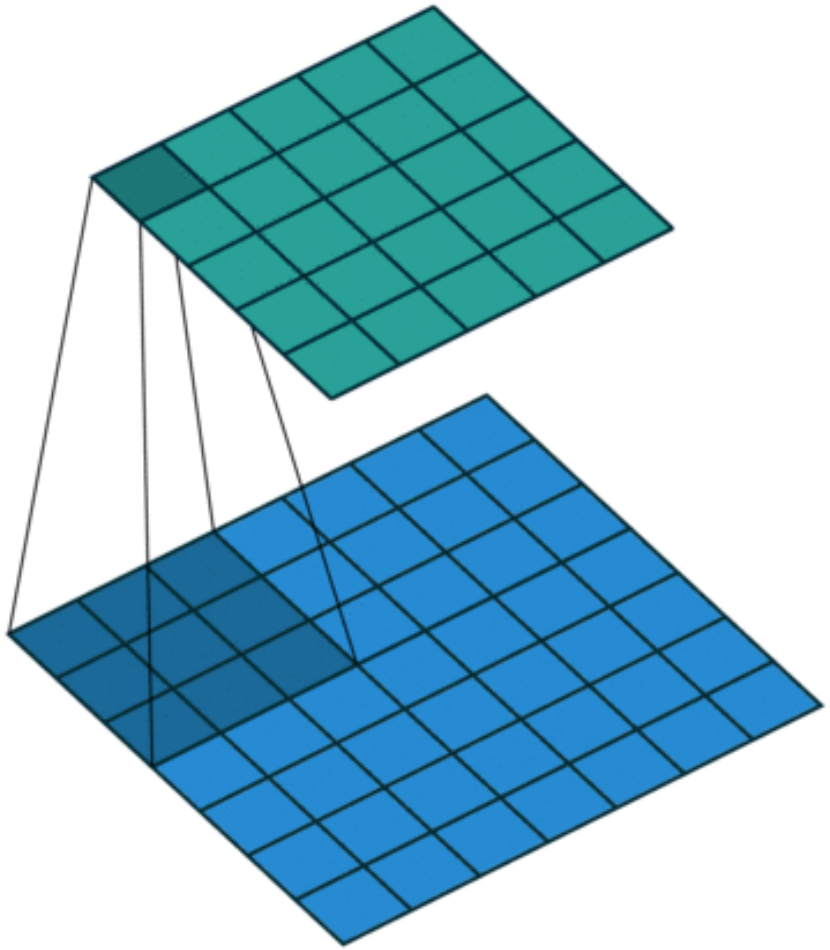
- Image $(n \times n)$ ➔ Pad with (p, p) ➔ Padded Image $(n + 2p \times n + 2p)$
- Padded Image $(n + 2p \times n + 2p)$) * filter $(f \times f)$

$$= (n + 2p - f + 1 \times n + 2p - f + 1)$$

S for Stride

- Number of steps to move the filter over the image

S = 1



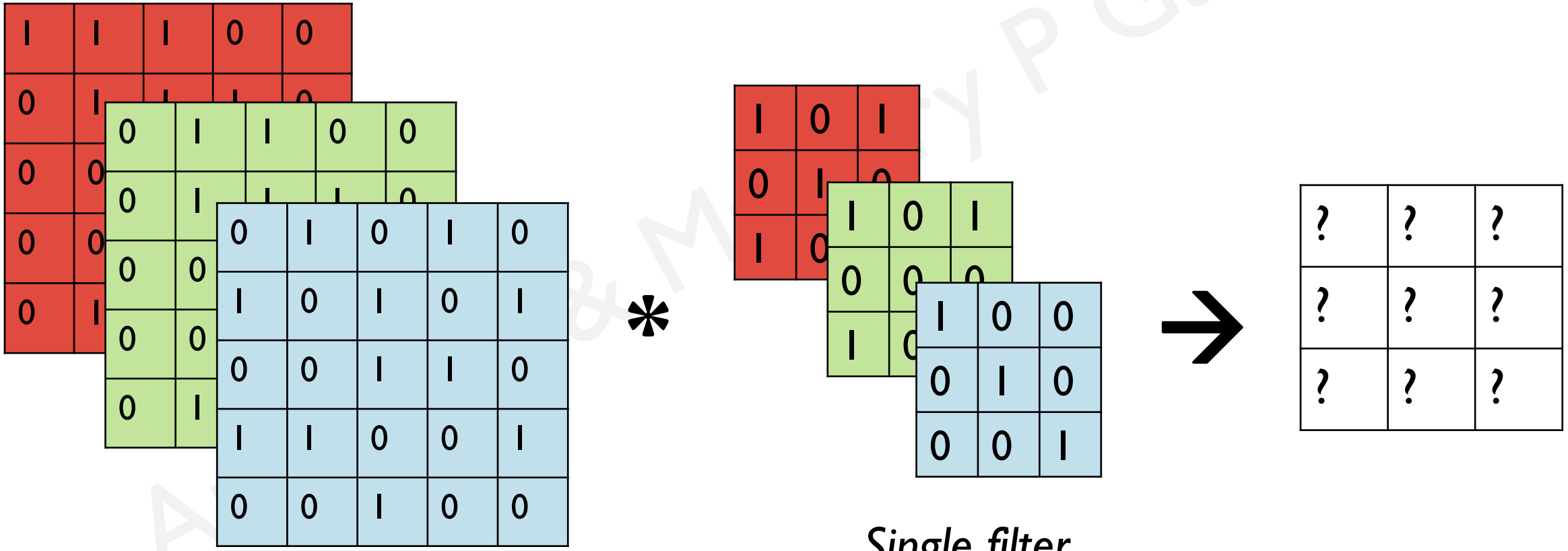
S = 2

1	6	5
7	10	9
7	10	8

Dimension of Convolved Image after Padding and Stride

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

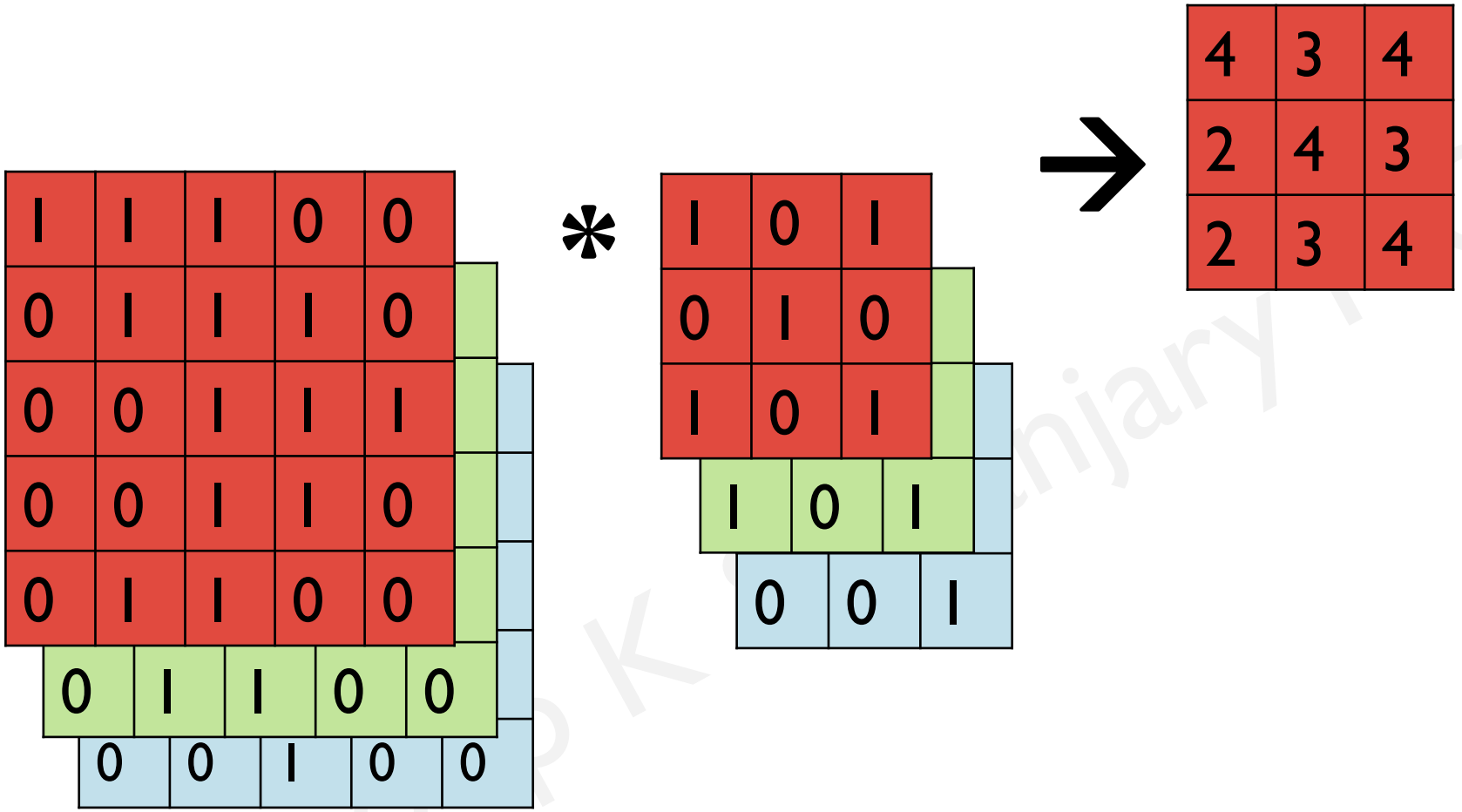
Convolution over Volumes

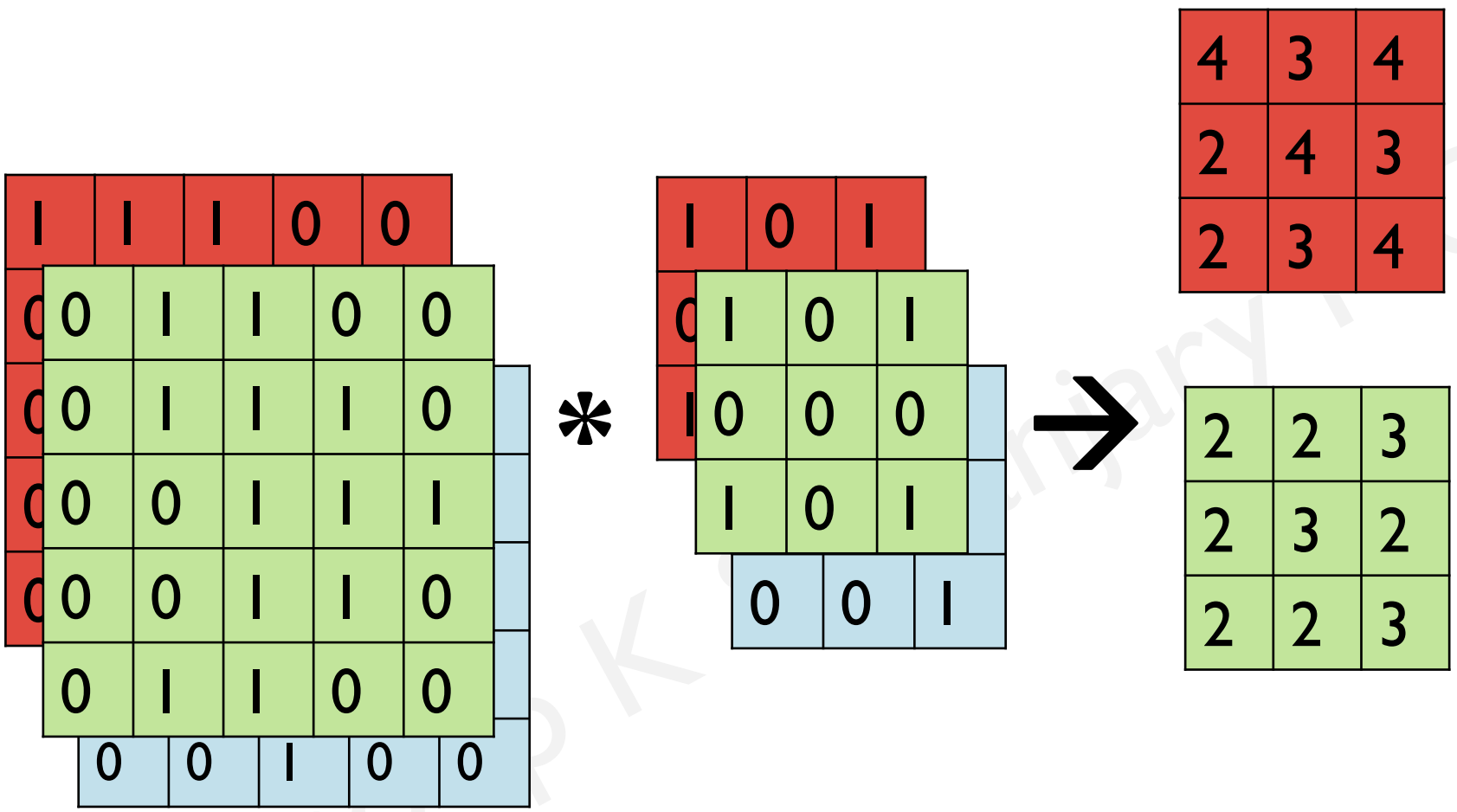


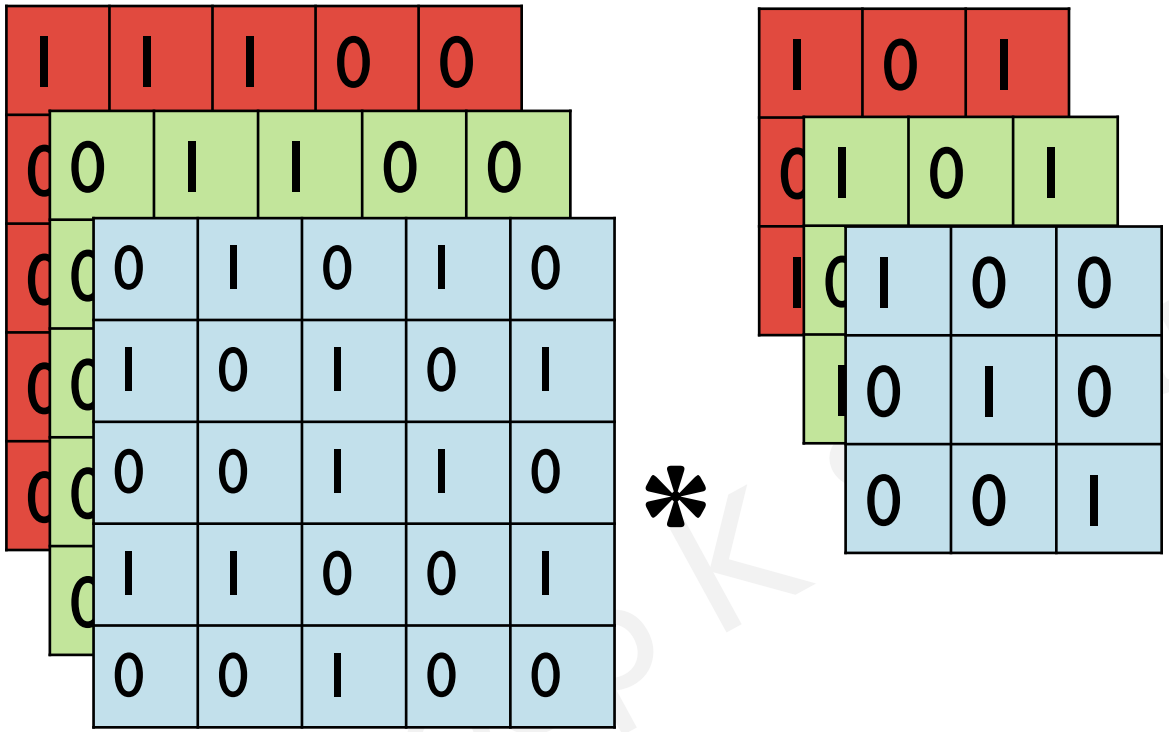
Single filter

 # Image channels : $n_c = 3$

 # Filter channels : $n_c = 3$





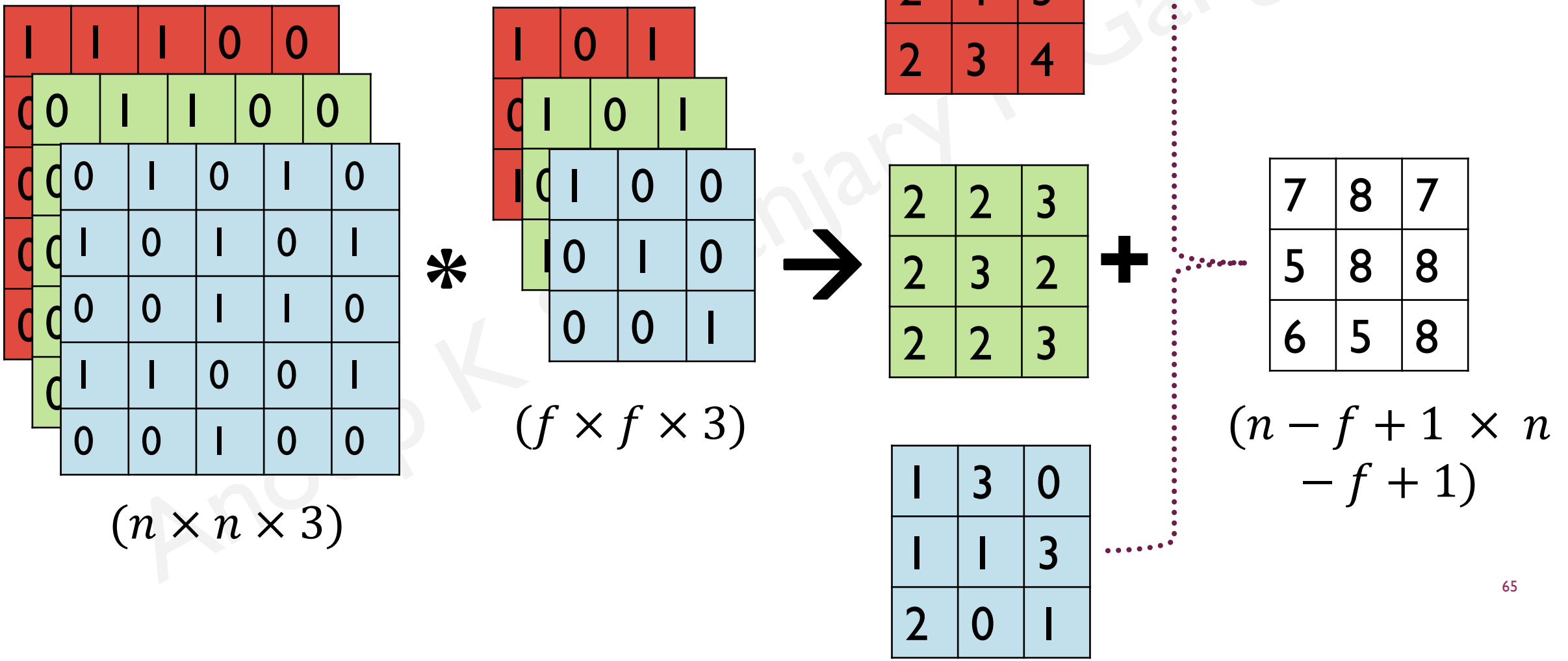


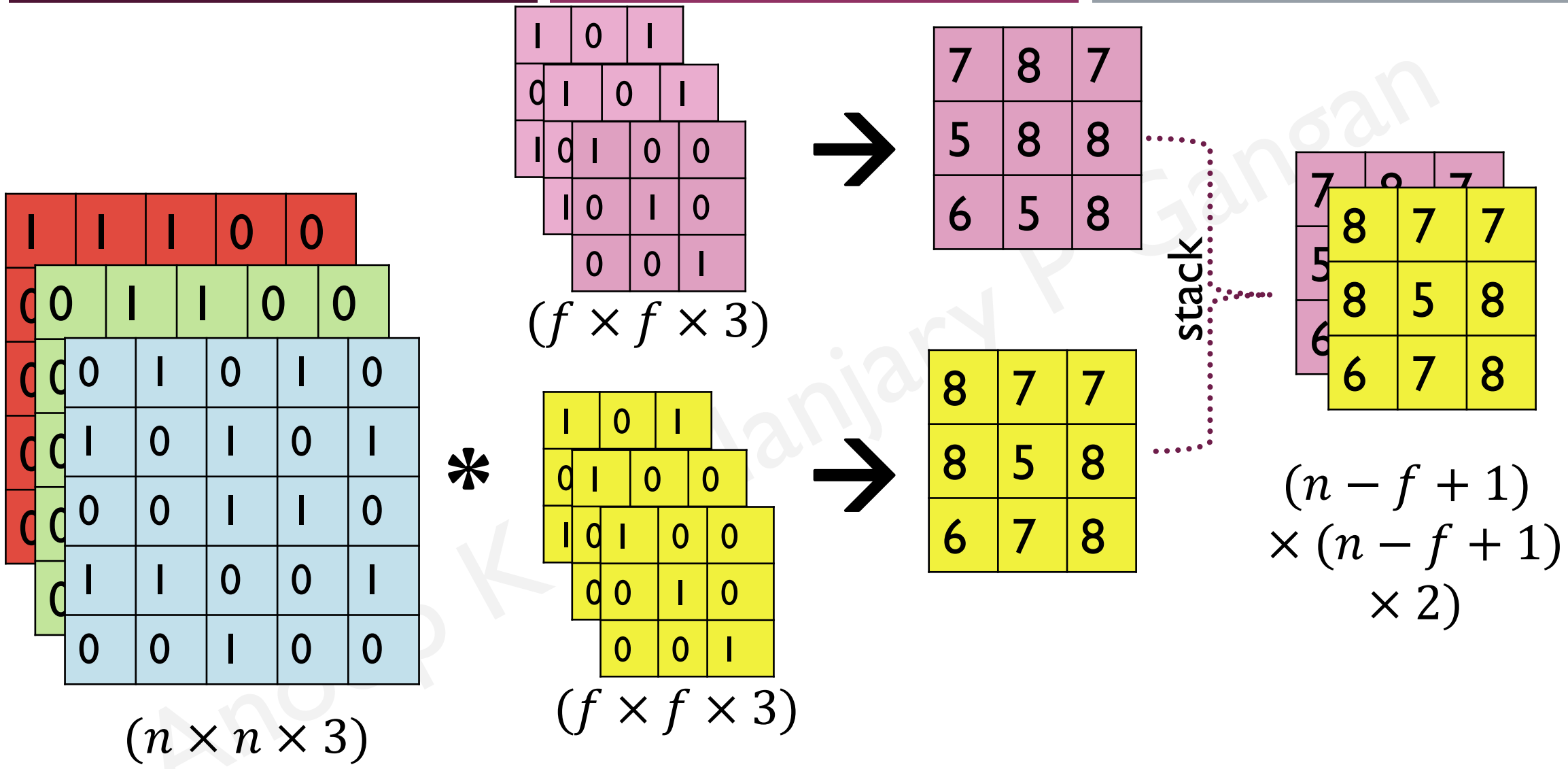
4	3	4
2	4	3
2	3	4

2	2	3
2	3	2
2	2	3

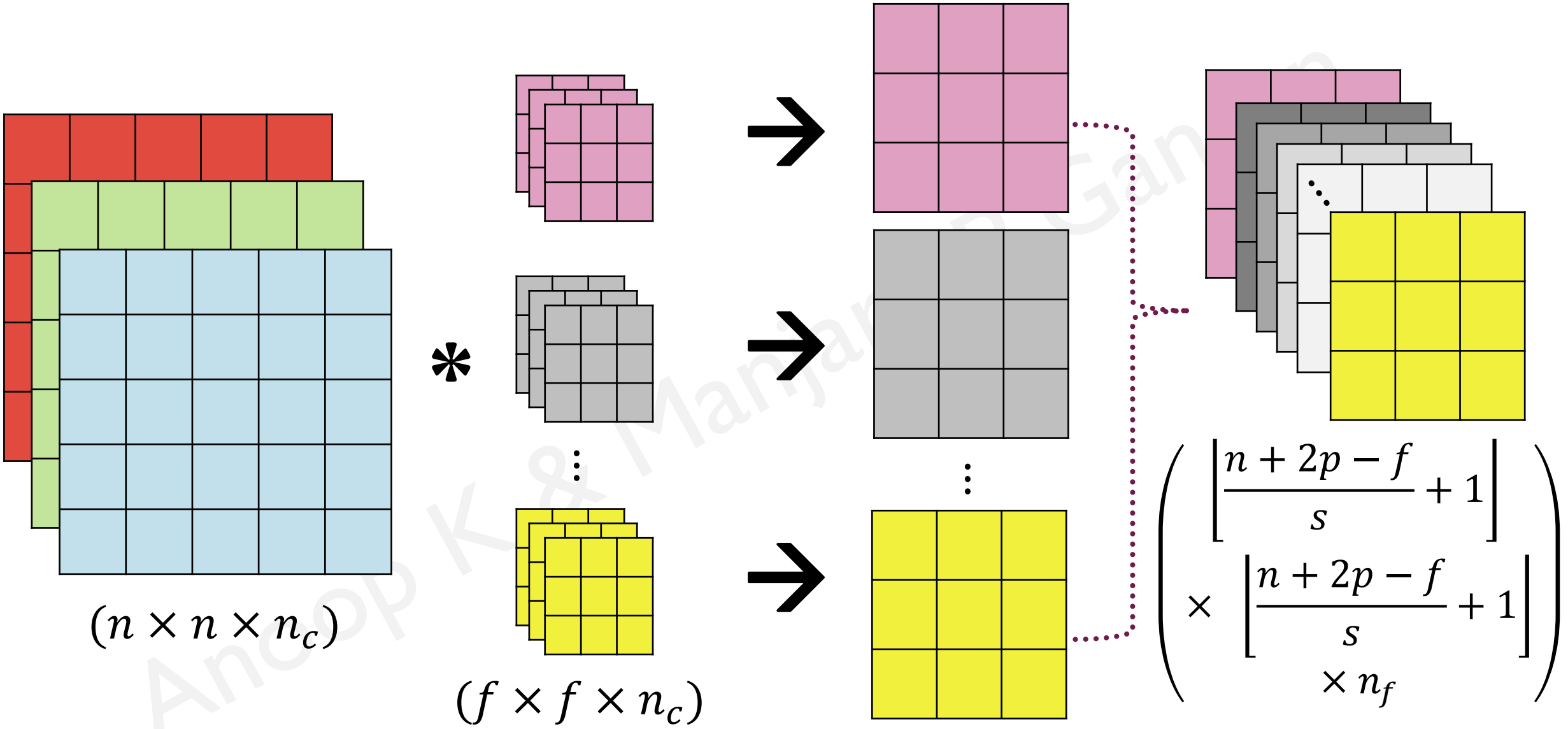
→

1	3	0
1	1	3
2	0	1





$$\#filters: n_f = 2$$



Pooling

Feature Map

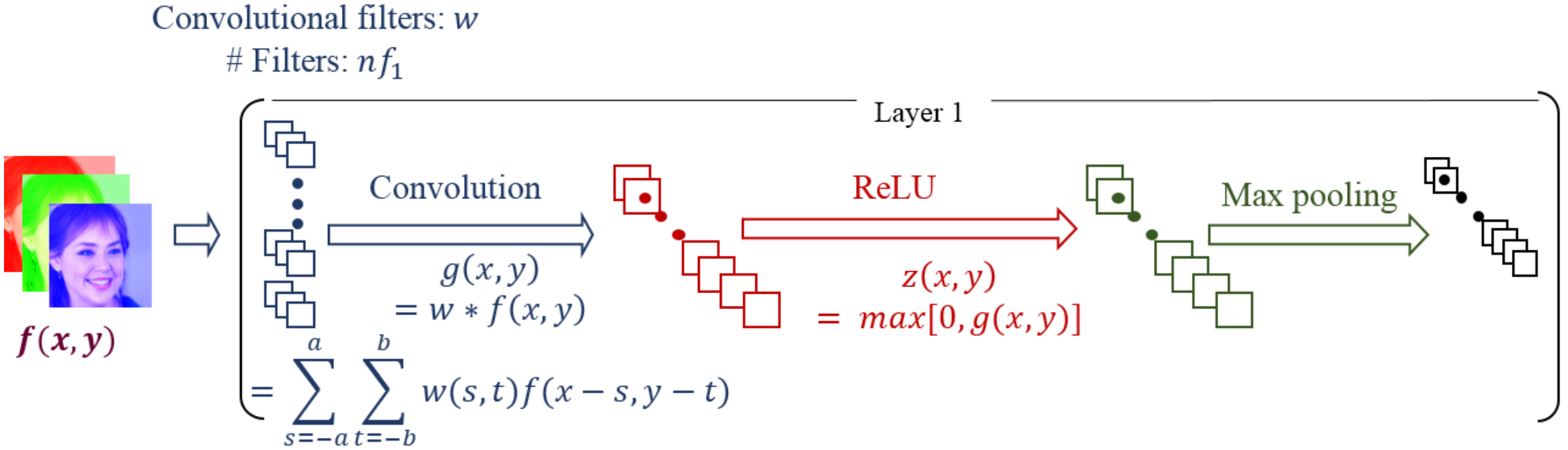
6	6	6	6
4	5	5	4
2	4	4	2
2	4	4	2

Max Pooling

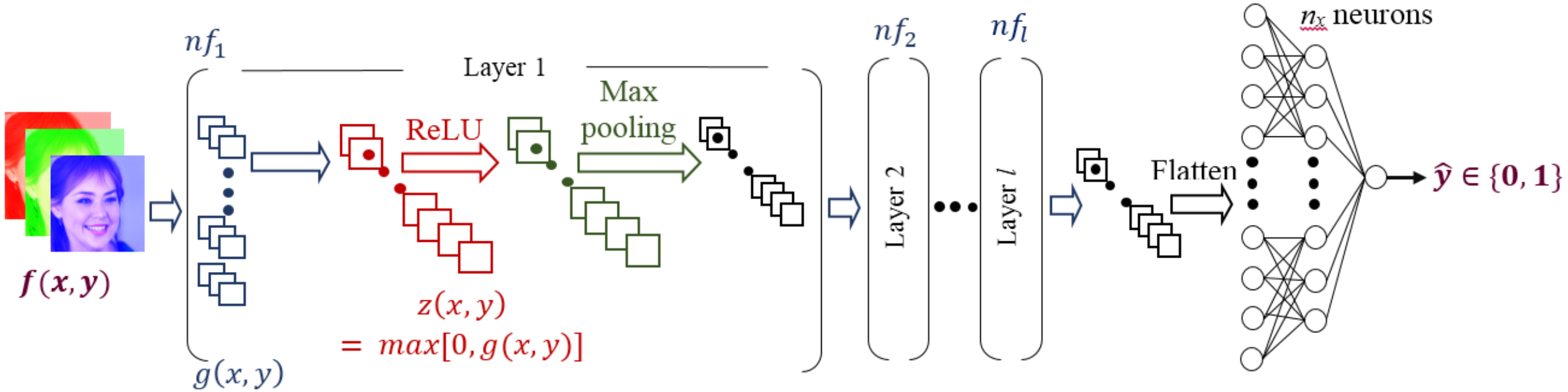
Average Pooling

Sum Pooling

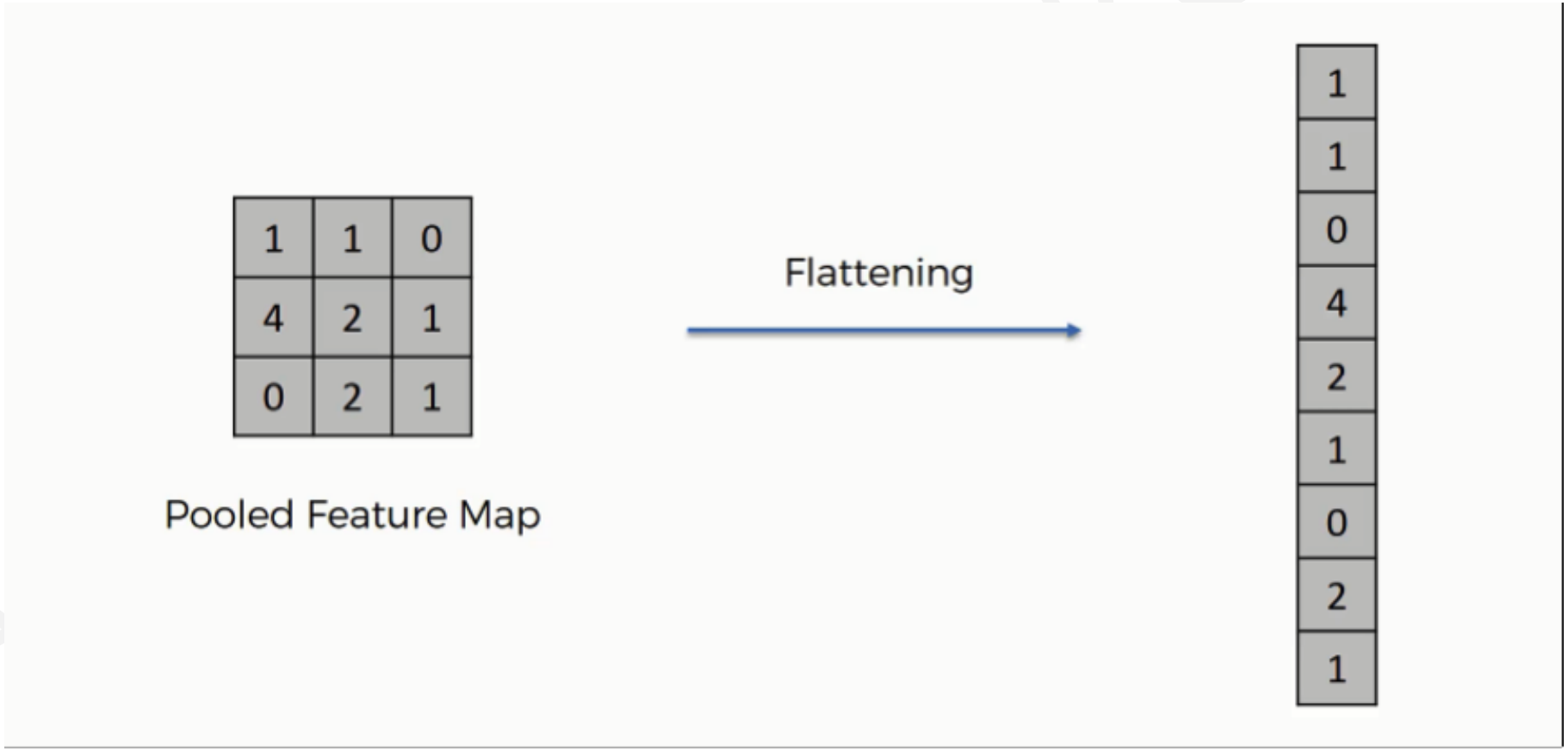
One Layer of Convolutional Neural Network



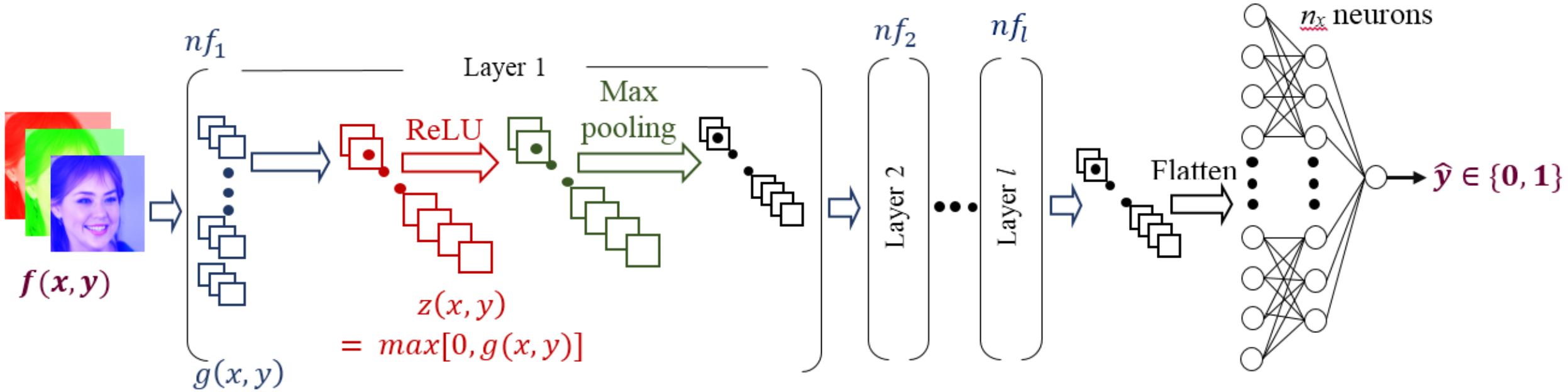
l – Layer CNN


nf_1
Layer 1
ReLU
Max pooling
nf_2
nf_l
Layer 2
Layer l
n_x neurons
Flatten
$\hat{y} \in \{0, 1\}$
$f(x, y)$
$z(x, y) = max[0, g(x, y)]$
$g(x, y) = w * f(x, y) =$
$\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t)$

Flatten



l – Layer CNN


nf_1
Layer 1
ReLU
Max pooling
nf_2
nf_l
Layer 2
Layer l
n_x neurons
Flatten
$\hat{y} \in \{0, 1\}$
$f(x, y)$
$z(x, y) = max[0, g(x, y)]$
$g(x, y) = w * f(x, y) =$
$\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t)$

Trainable Parameter Calculation

Layer	Shape	Size	# Trainable parameters
input	(32, 32, 3)	32*32*3=3072	0
conv1($f: 5 \times 5, s: 1$)	(28, 28, 8)	6272	(5*5*8) + 8 = 208
pool1	(14,14, 8)	1568	0
conv2($f: 5 \times 5, s: 1$)	(10, 10, 16)	1600	(5*5*16) + 16 = 416
pool2	(5, 5, 16)	400	0
fc3	(120, 1)	120	(400*120) + 1 = 48,001
fc4	(84, 1)	84	(120*84) + 1 = 1,00,081
softmax	(10, 1)	10	(84*10) + 1 = 841
Total trainable parameters			1,49,547

SOTA Datasets & Pre-trained Models for Image Classification

In Image Classification, there are some very popular datasets used across research, industry, and hackathons.

- **ImageNet**: https://image-net.org/download

  (training set 1,281,167, valiation set 50,000 and test set 100,000)

- **CIFAR**: https://www.cs.toronto.edu/~kriz/cifar.html

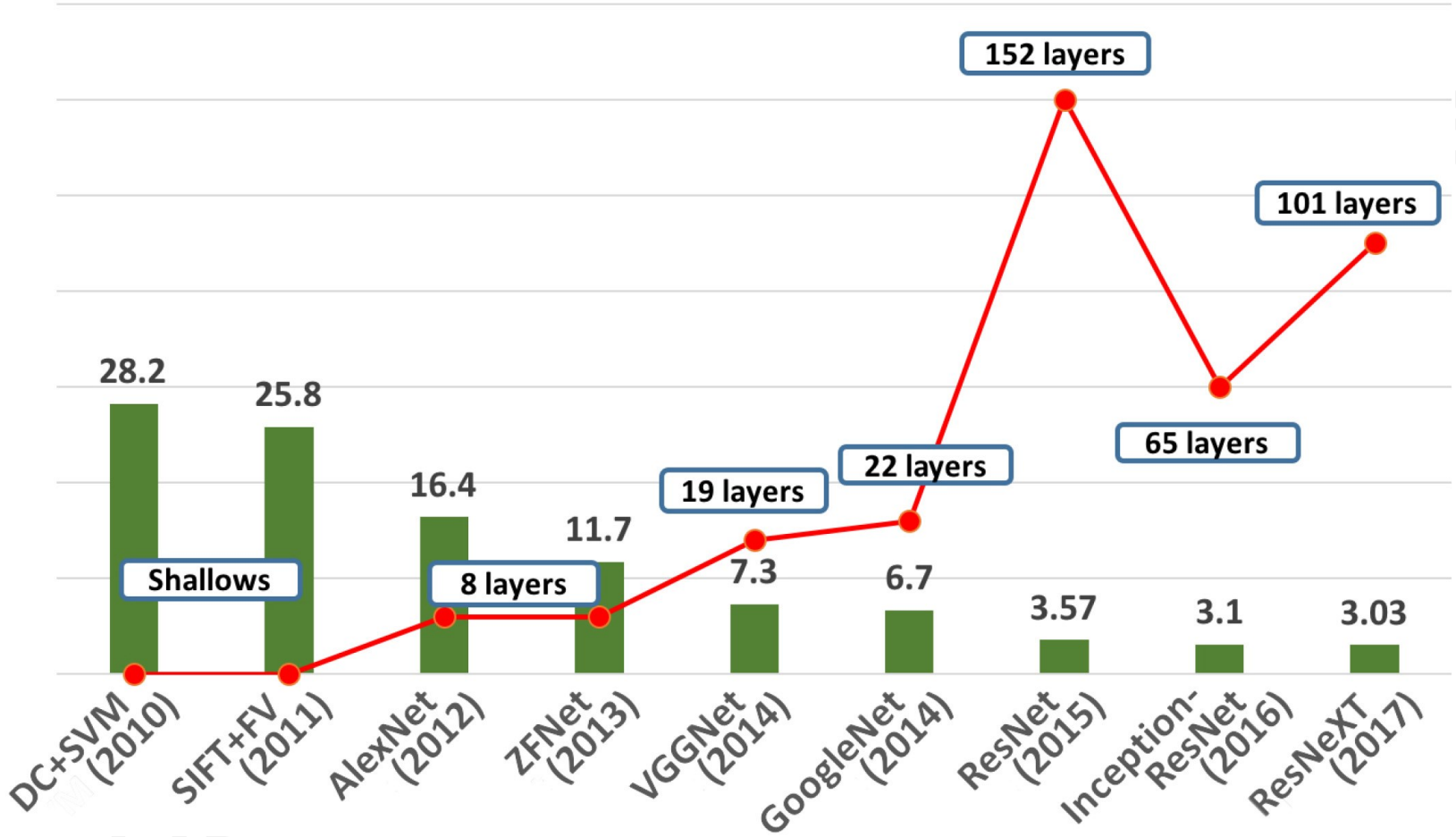
  (training set 50,000 and test set 10,000)

- **MNIST**: http://yann.lecun.com/exdb/mnist/

  (training set of 60,000 and test set 10,000)

https://keras.io/api/applications/

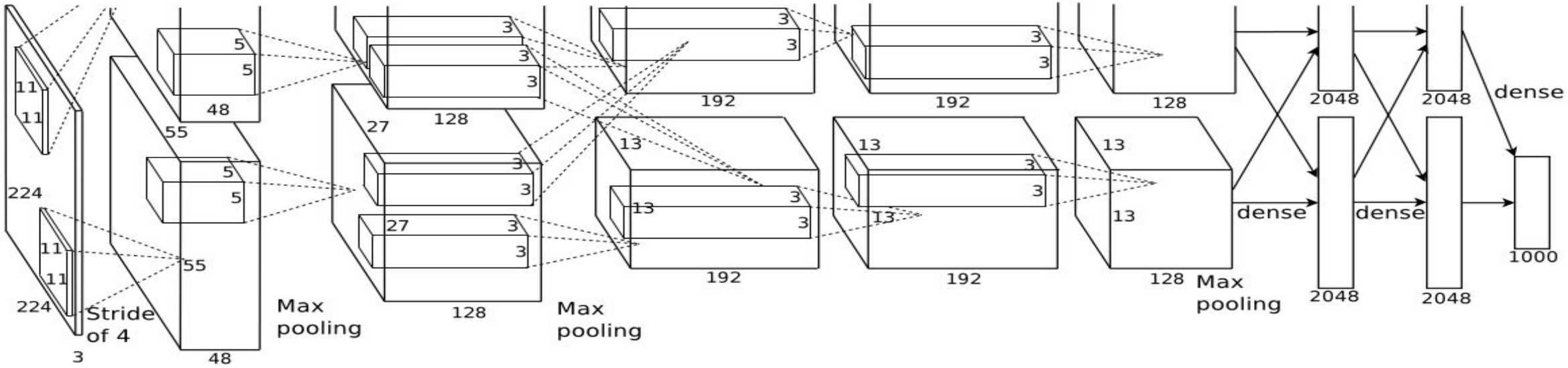
The network top five errors (%) and layers in the ImageNet classification over time.



Pouyanfar, Samira, et al. "A survey on deep learning: Algorithms, techniques, and applications."
ACM Computing Surveys (CSUR) 51.5 (2018): 1-36.

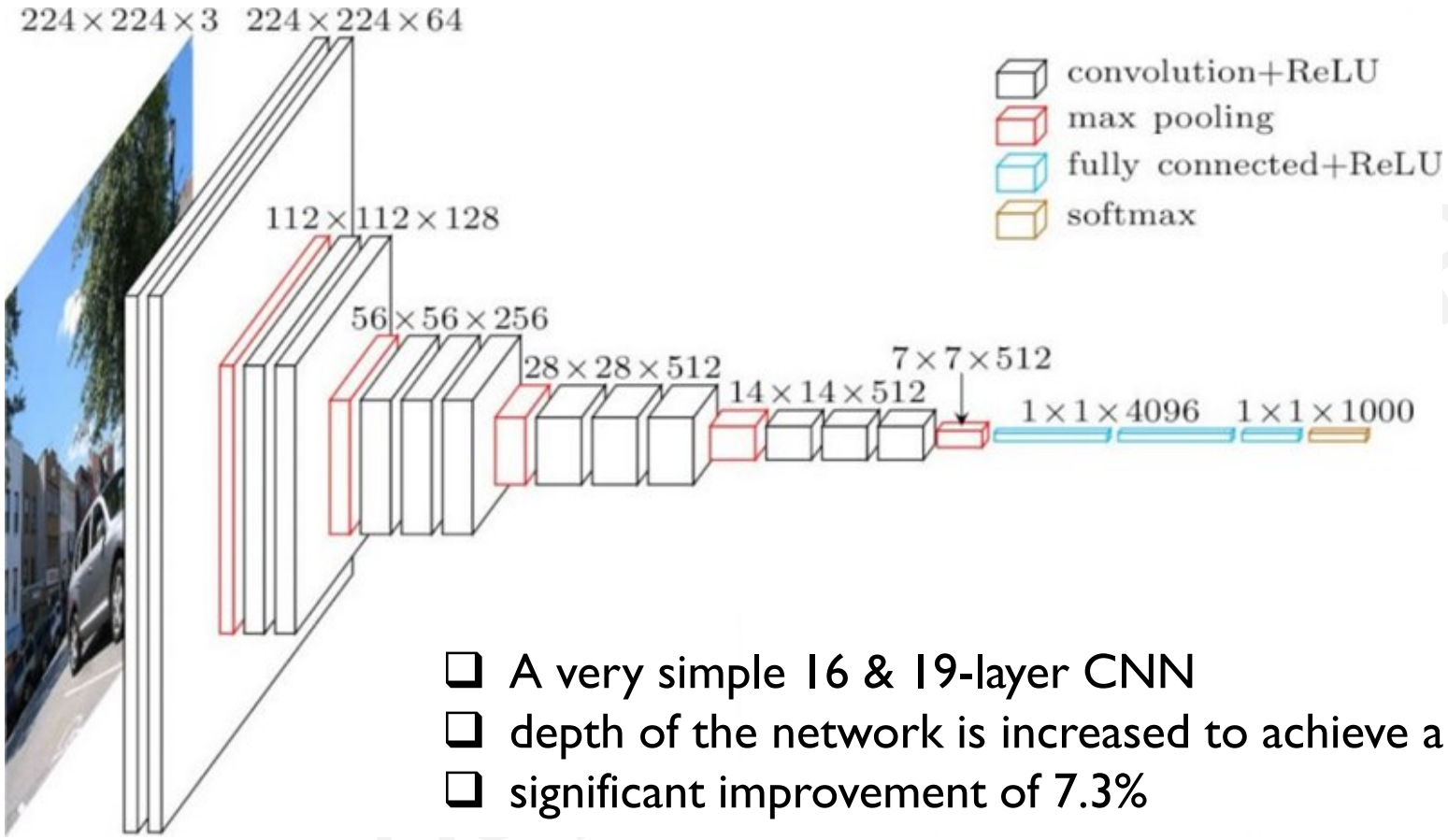
AlexNet (2012)

- First CNN model that substantially improved the image classification results on a very large dataset (e.g., ImageNet)
- Architecture: 8 layers (5 convolutional layers and 3 fully-connected layers), ReLU Nonlinearity
- Trained 6 Days on 2 GPU
- Overfitting is reduced using: Data Augmentation & Dropout
- Results: Improved on the best results from the previous years by almost 10% regarding the top five test errors



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012): 1097-1105.

VGG Net (2014)

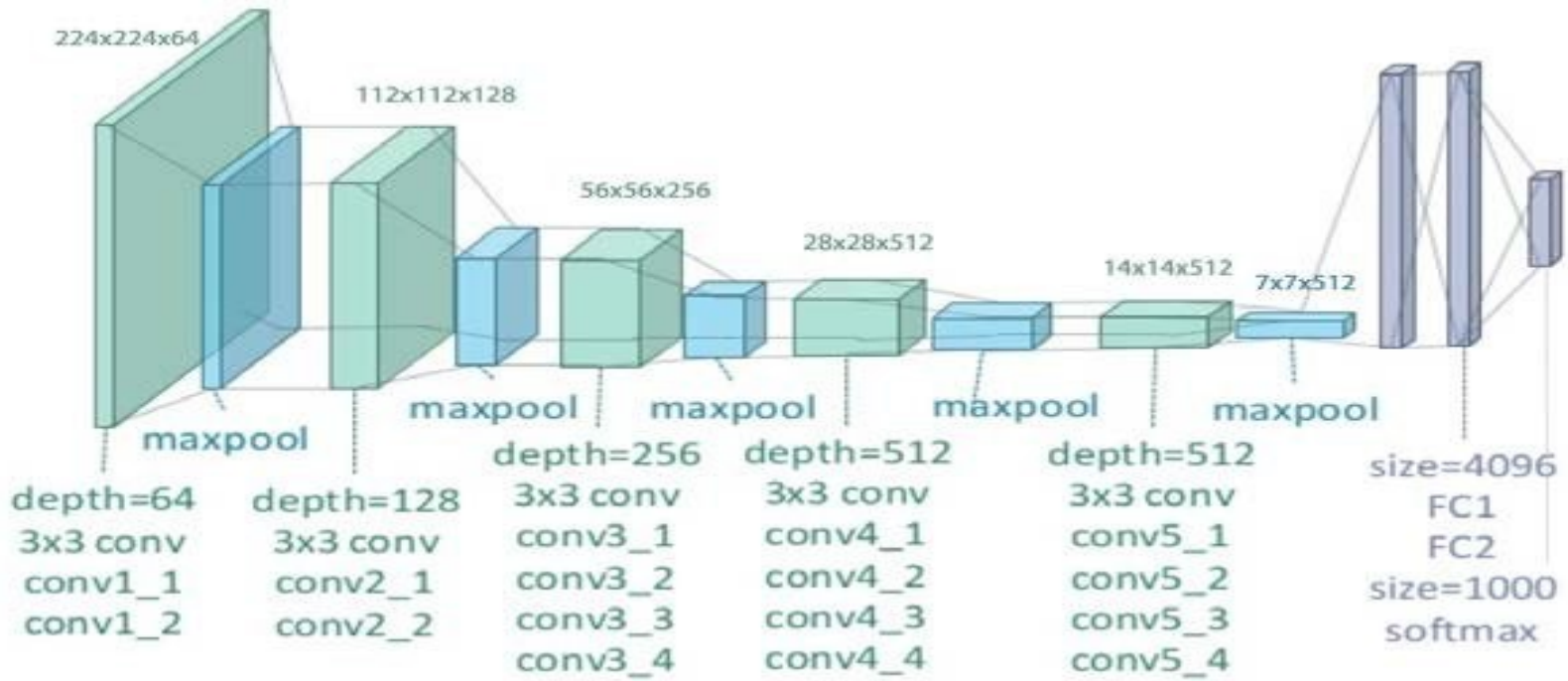


VGG 16

- A very simple 16 & 19-layer CNN
- depth of the network is increased to achieve a more effective and efficient model
- significant improvement of 7.3%

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

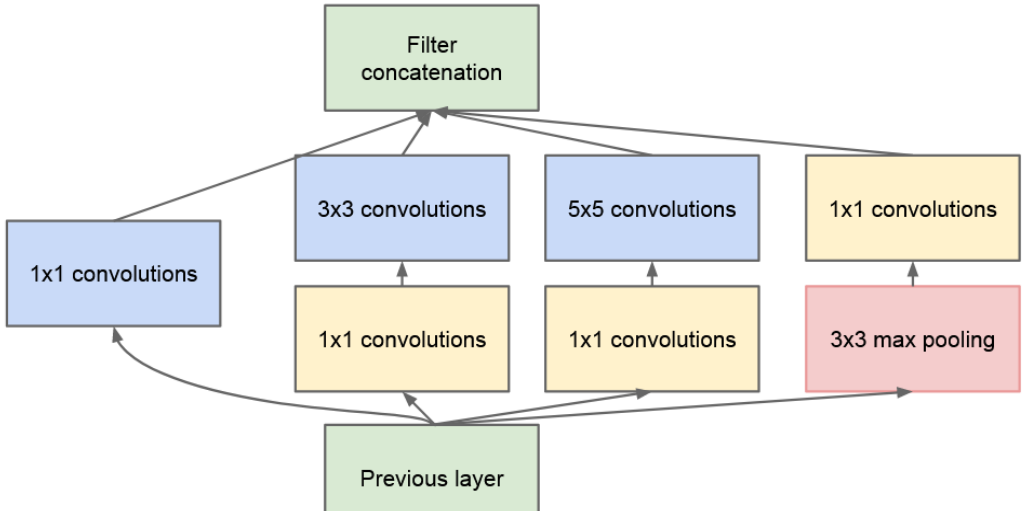
VGG 16 Performs better than 19

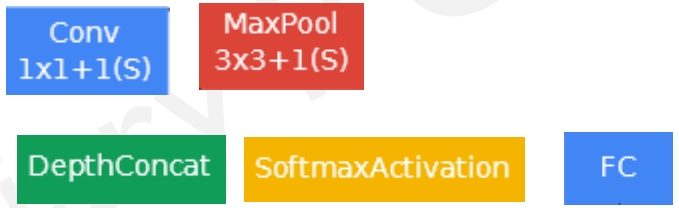


VGG 19

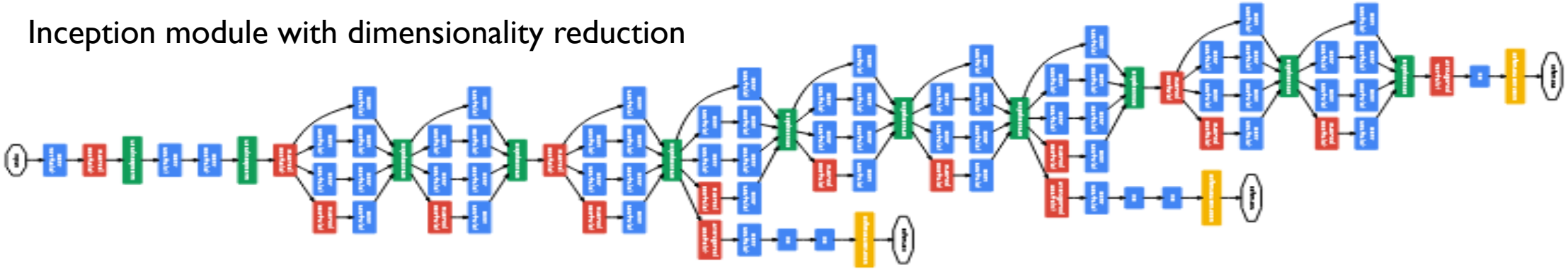
GoogLeNet / Inception V1, V2, & V3 (2014)

Inception Module just performs convolutions with different filter sizes on the input, performs Max Pooling, and concatenates the result for the next Inception module





Inception module with dimensionality reduction



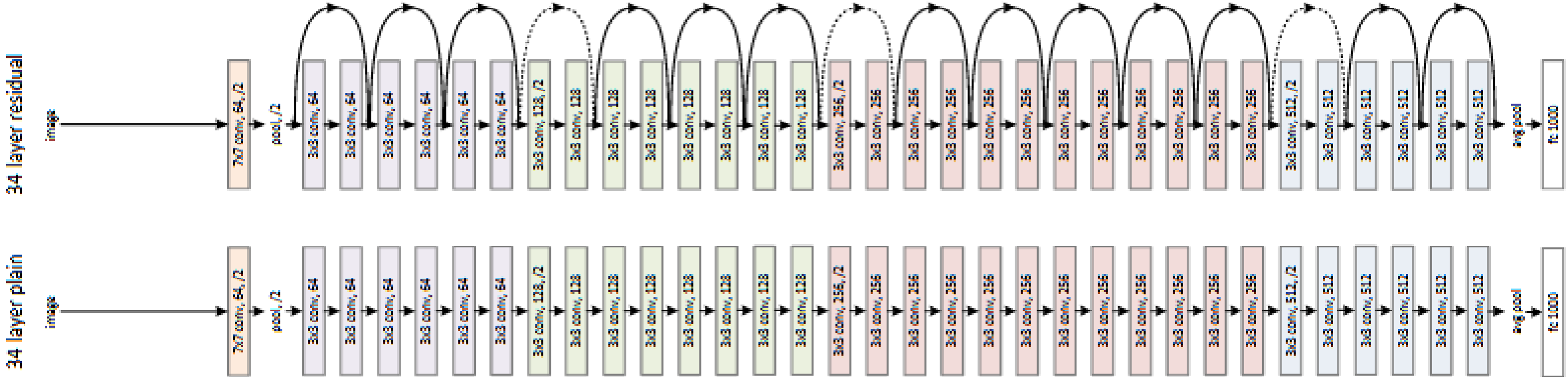
Inception v1: Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

Inceptionv1

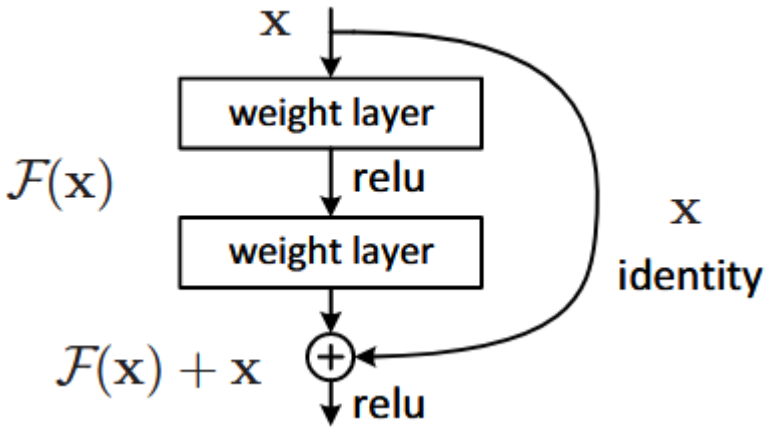
- Number of layers in Inceptionv1 is 22
- 9 inception modules stacked linearly. It is 22 layers deep (27, including the pooling layers)
- It uses global average pooling at the end of the last inception module
- Inceptionv3 is a convolutional neural network which has a depth of 50 layers

Inceptionv2&3: 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).

ResNet50 (2015)



- After every 2 convolutions, bypassing/skipping the layer in-between - This is the main concept behind ResNet models.
- These skipped connections are called 'identity shortcut connections" and uses what is called residual blocks:



Residual learning: A building block

- Earliest variant is ResNet34 (ResNet50 also follows a similar technique with just more layers)
- ResNet50 is a convolutional neural network which has a depth of 50 layers
- Build and trained by **Microsoft in 2015**
- Ultra-deep learning model (50 to 152 layers)
- ResNet50 was improved several times and newer versions such as ResNet101, ResNet152, ResNet50V2, ResNet101V2, ResNet152V2

Incredible performance (3.6% top five error)

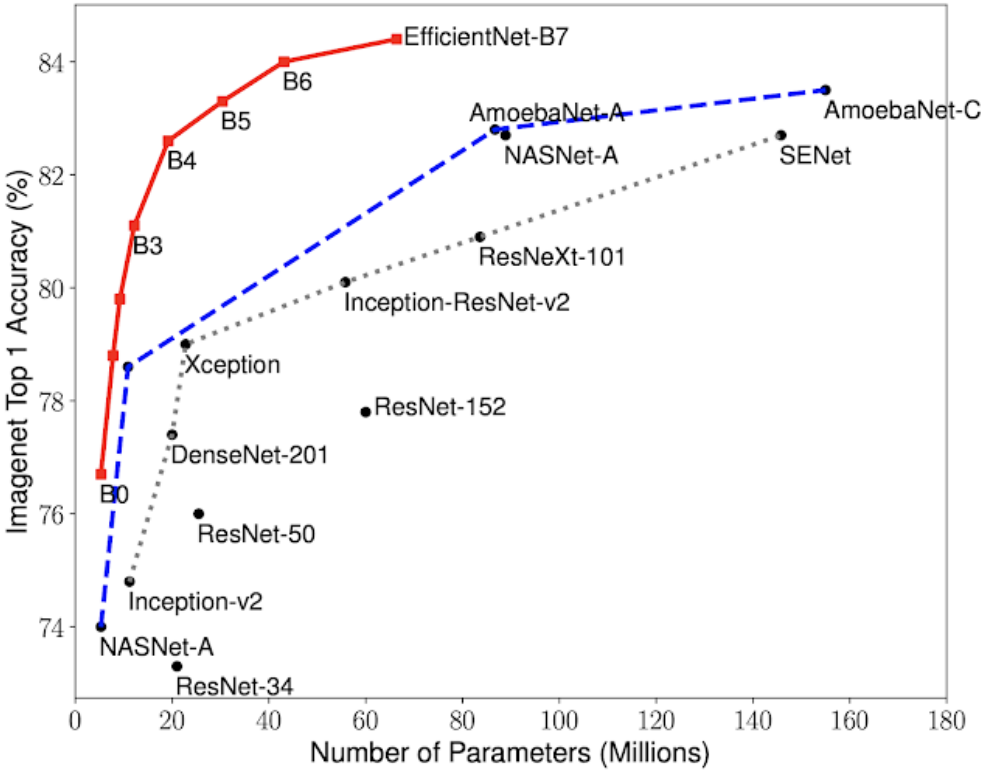
For the first time, a computer model could beat human brains (with 5% to 10% error) in image classification.

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).

EfficientNet

- Propose a new Scaling method called Compound Scaling
- Models like ResNet follow scaling the dimensions arbitrarily and by adding up more and more layers
- Scale the dimensions by a fixed amount at the same time & achieve much better performance
- 8 alternative implementations of EfficientNet (B0 to B7)

MBConv stands for mobile inverted bottleneck Convolution similar to MobileNetv2



- ❑ EfficientNets significantly out perform other ConvNets.
- ❑ EfficientNet-B7 achieves new state-of-the-art
- ❑ EfficientNet-B1 is 7.6x smaller & 5.7x faster than ResNet-152.

Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In International Conference on Machine Learning (pp. 6105-6114). PMLR.

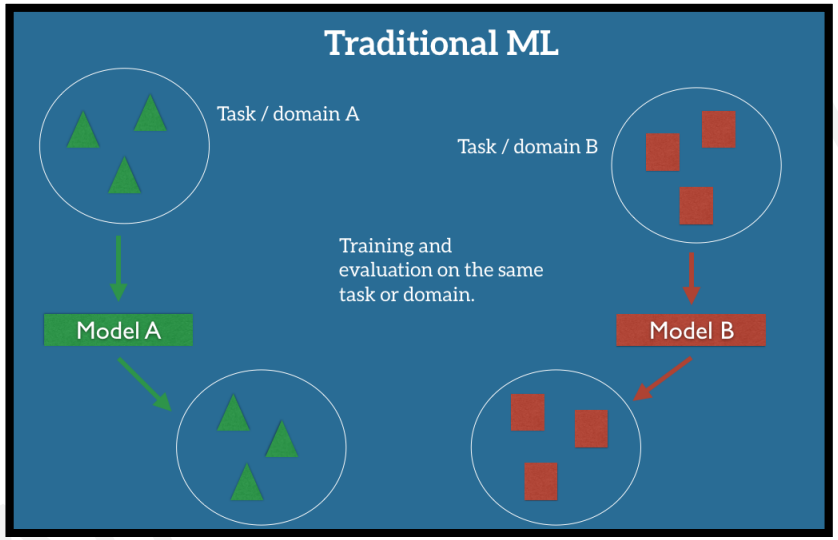
Distinguishing Natural and Computer-Generated Images using Multi-Colorspace fused EfficientNet

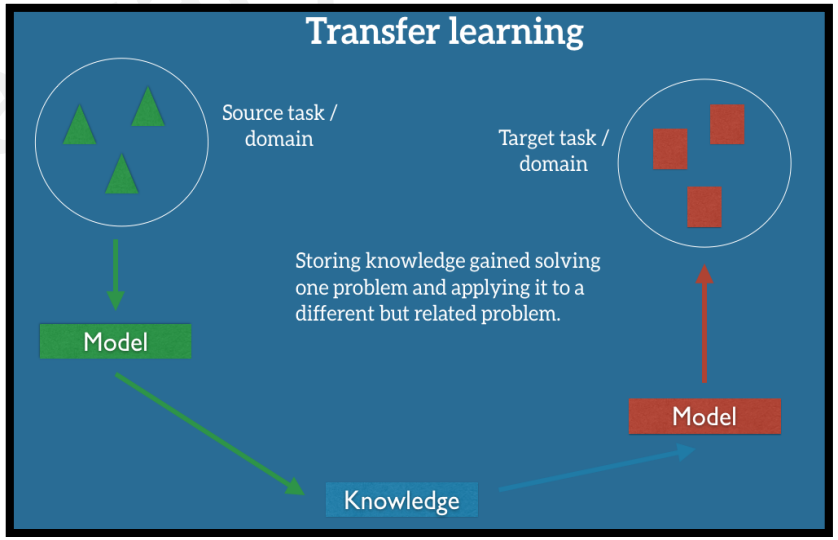
Manjary P Gangan, Anoop K, and Lajish V L
Computational Intelligence and Data Analytics (CIDA Lab)
Department of Computer Science
University of Calicut, India

Paper: https://arxiv.org/abs/2110.09428
Github: https://github.com/manjaryp/GANvsGraphicsvsReal
Other link: https://dcs.uoc.ac.in/cida/projects/dif/mceffnet.html

Recent Research Directions

- ❑ Transfer Learning Strategies
- ❑ Attention & Transformers
- ❑ Visual Biases



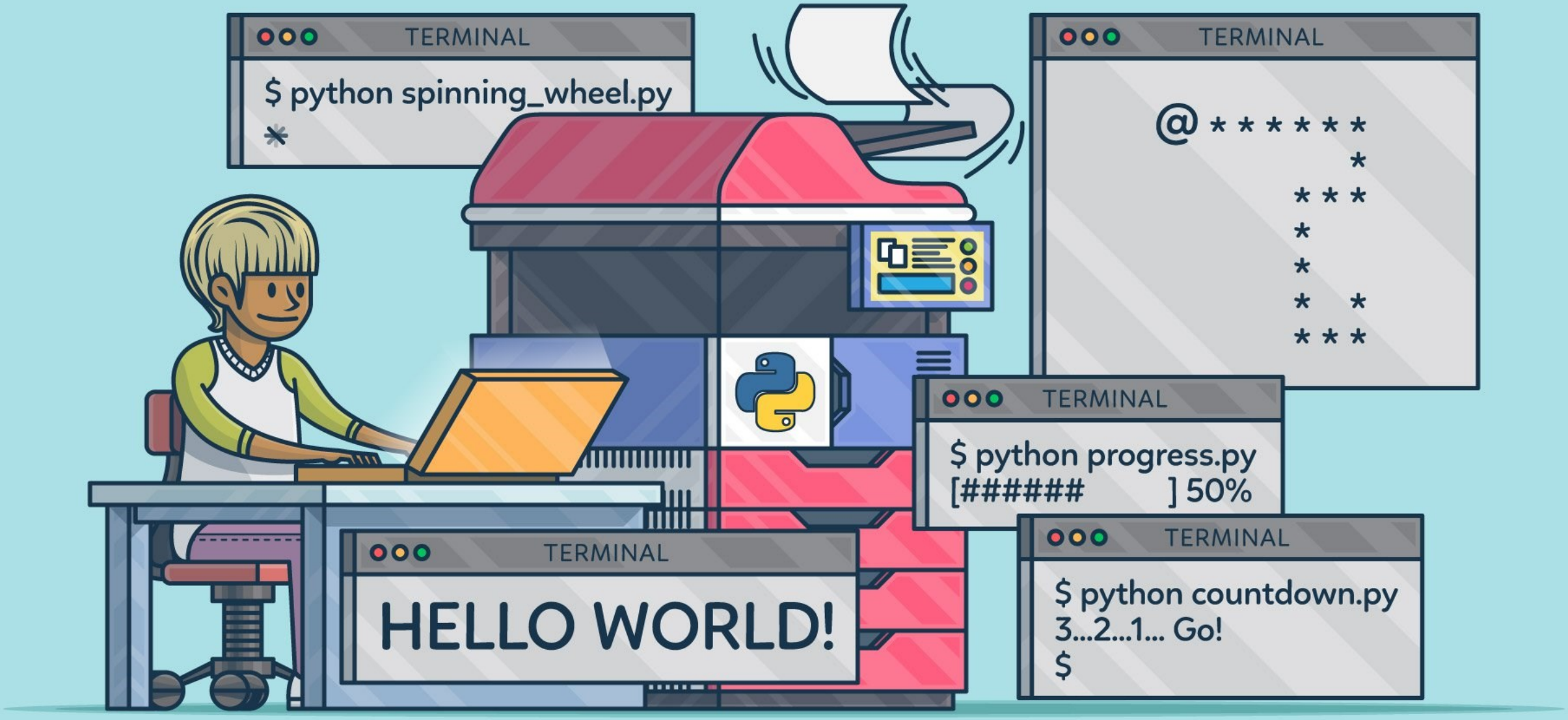


Bridging
Theory & Practice

https://github.com/anoopkdcs/DLwithCNN

https://colab.research.google.com/

1. ANN.ipynb
2. CNN_CvD.ipynb
3. CNN_textClassifier.ipynb
4. KimsModel.ipynb



Come Let's Practice!

thank you!

Have a Nice Day

Anoop K
anoopk_dcs@uoc.ac.in
dcs.uoc.ac.in/~anoop
YouTube: Notes2Learn