CHULA ENGINEERING
Foundation toward Innovation
COMPUTER

Data Mining Group
Machine Intelligence and Knowledge Discovery Lab
Chulalongkorn University

Introduction to Deep Learning

Assoc. Prof. Peerapon Vateekul, Ph.D.
Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University
Peerapon.v@chula.ac.th
www.cp.eng.chula.ac.th/~peerapon/

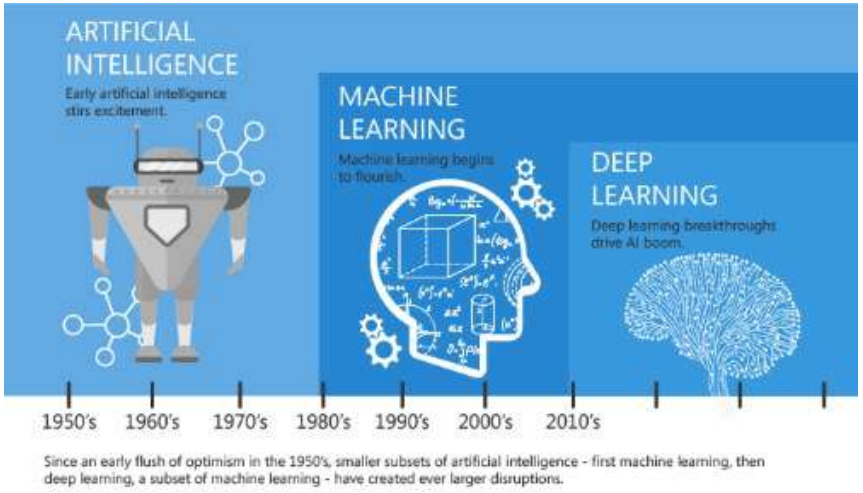
Outline

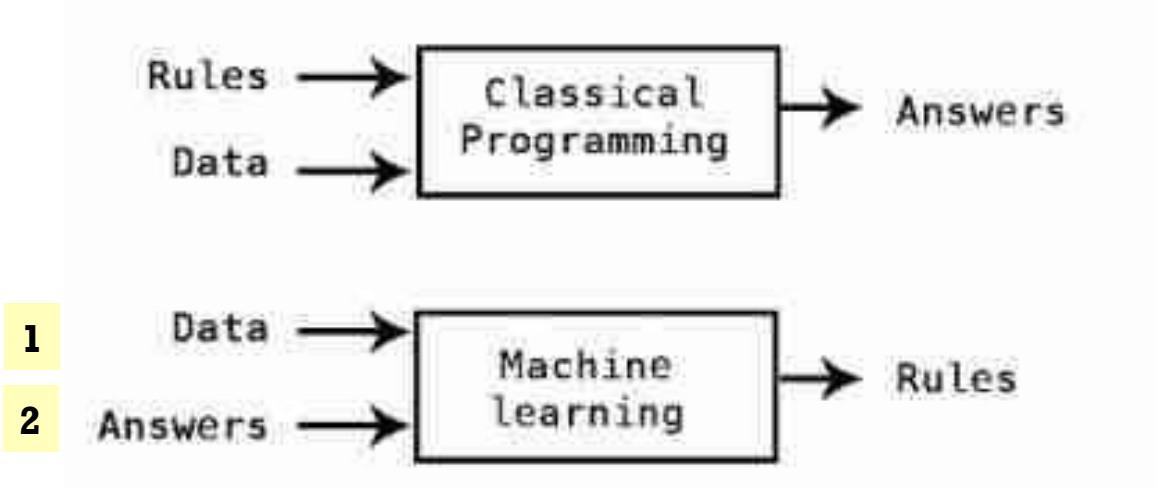
- Introduction to Deep Learning
- Convolutional Neural Networks (CNN) [Imaging Task]
- Recurrent Neural Networks (RNN) [Time Series Forecasting]
- Language Technologies

Introduction to Deep Learning

AI = Automation

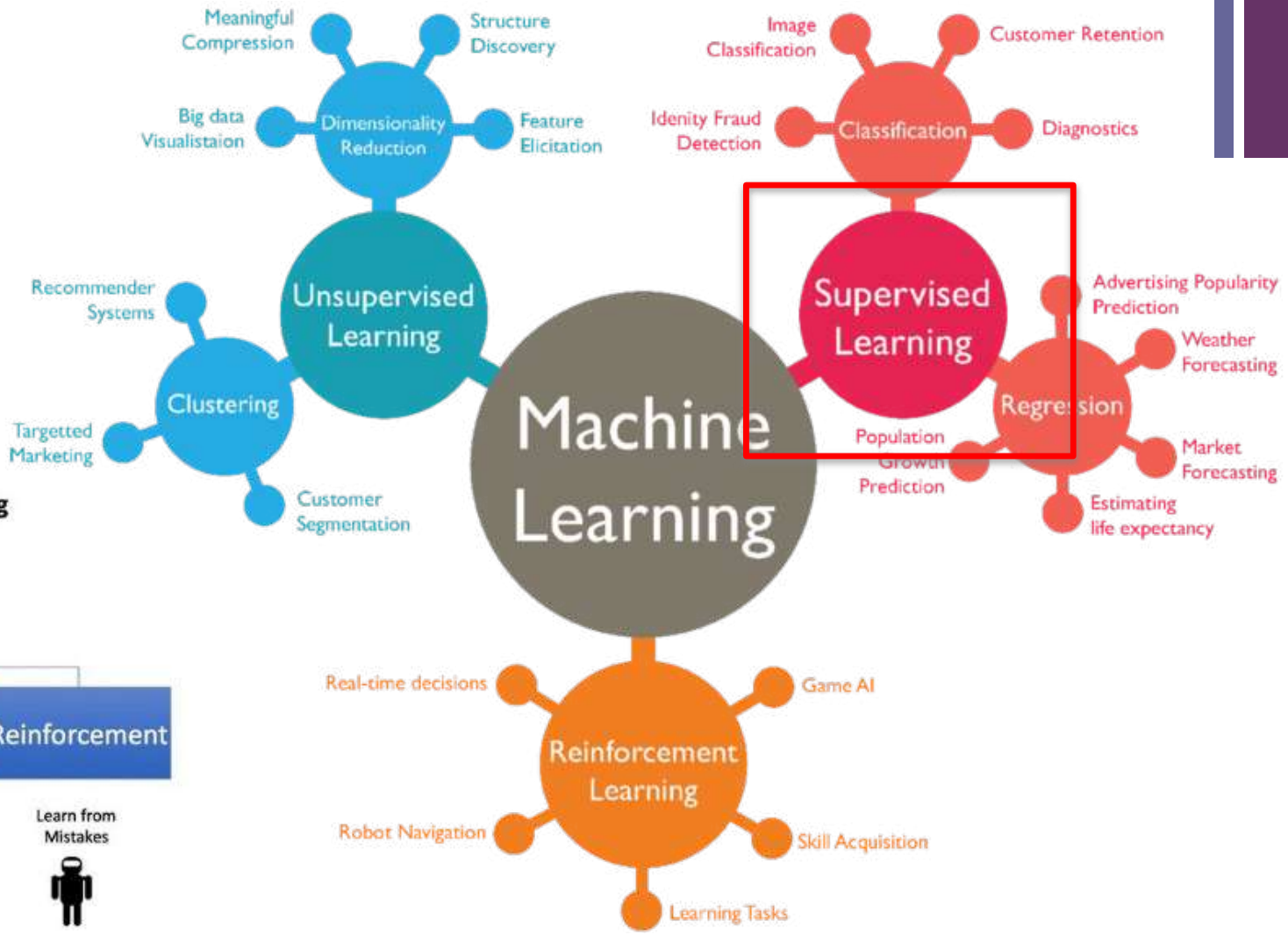
- 1) Rule-based AI (Symbolic AI)
- 2) Machine Learning



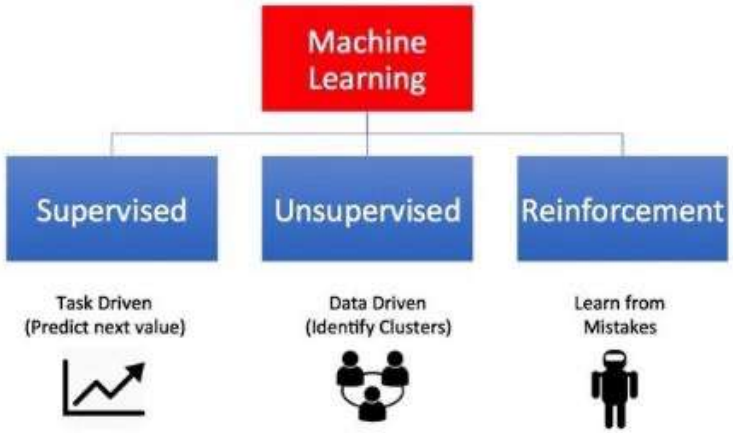


https://mc.ai/machine-learning-basics-artificial-intelligence-machine-learning-and-deep-learning/

Machine Learning



Types of Machine Learning



Task1: Supervised learning

Handcrafted features

Training Data

	inputs			target
Age	**Gender**	**BodyTemp**	**Cough**	**Corona**
12	Female	37	Yes	Yes
35	Female	39	No	Yes
32	Male	38	Yes	No

Testing Data

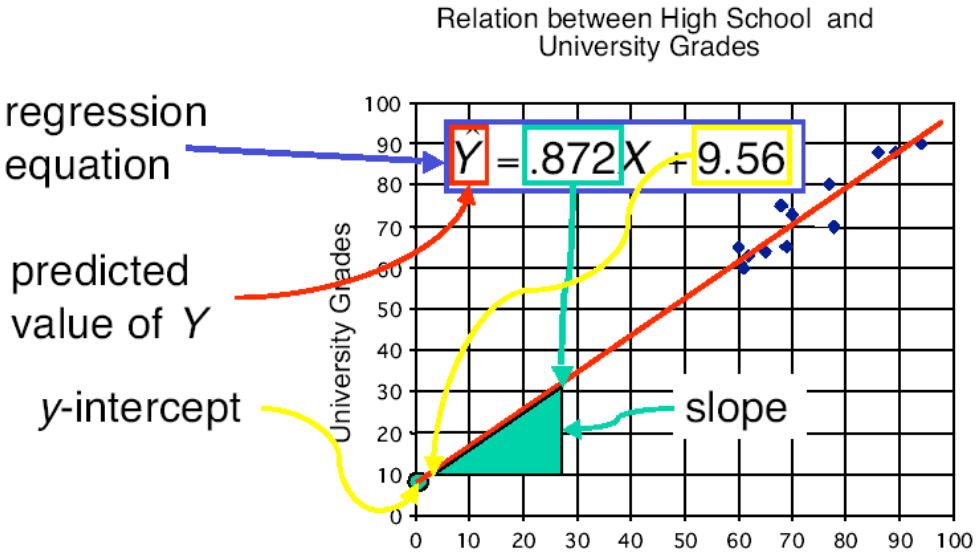
Age	**Gender**	**BodyTemp**	**Cough**	**Corona**
25	Male	40	No	?

Application: Corona Prediction

Prediction algorithms

- Decision Tree
- (Logistic) Regression
- kNN
- Support Vector Machine
- Neural Networks (NN)
- Deep Learning

Regression – Linear Relationship



weight, coefficient

$$\underset{\text{target}}{\hat{y}} = \underset{\text{intercept}}{\hat{w}_0} + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

input

- The least square method aims to minimize the following term

$$\sum_{\substack{training \\ data}} (y_i - \hat{y}_i)^2$$

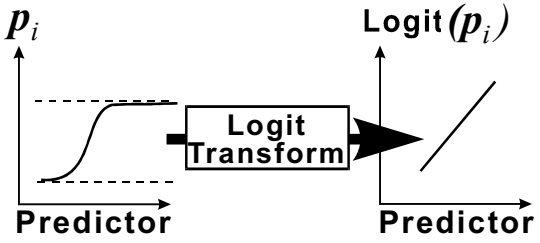
Logistic Regression (cont.)
Linear Relationship

Training Data

	inputs			target
Age	**Gender**	**BodyTemp**	**Cough**	**Corona**
12	Female (0)	37	Yes (1)	Yes
35	Female (0)	39	No (0)	Yes
32	Male (1)	38	Yes (1)	No

Logit_score = w0 + w1*Age + w2*Gender + w3*Temp + w4*Cough
Logit_score = 0.01 – 0.3*Age +0.2*Gender + 0.2*Temp + 0.9*Cough

Example
Logit_score = 0.01 – 0.3*12 +0.2*0 + 0.2*37 + 0.9*1 = 4.71
prob = 0.9911 → "Yes"

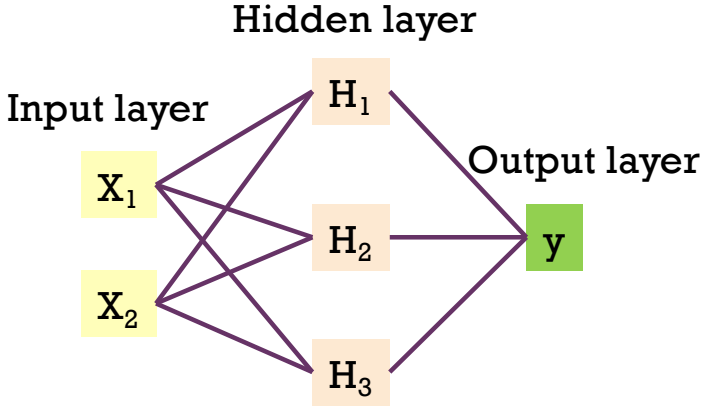


$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$

Application: Corona Prediction

Neural Networks (universal approximator)

Non-linear relationship

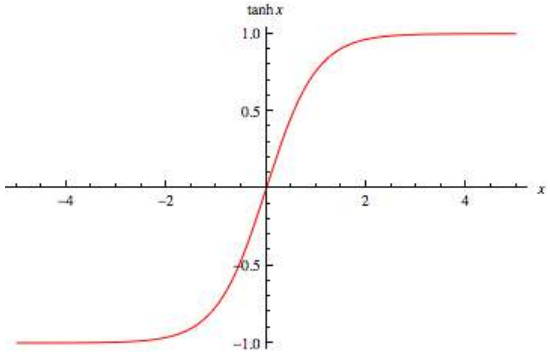


$$log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{w}_0 + \hat{w}_1 H_1 + \hat{w}_2 H_2 + \hat{w}_3 H_3$$

$$H_1 = tanh(\hat{w}_{10} + \hat{w}_{11} x_1 + \hat{w}_{12} x_2)$$

$$H_2 = tanh(\hat{w}_{20} + \hat{w}_{21} x_1 + \hat{w}_{22} x_2)$$

$$H_3 = tanh(\hat{w}_{30} + \hat{w}_{31} x_1 + \hat{w}_{32} x_2)$$

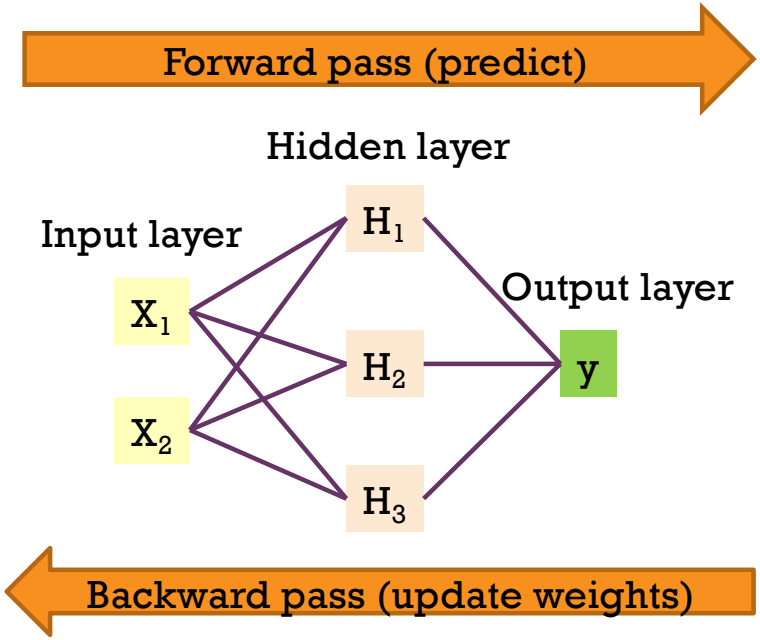


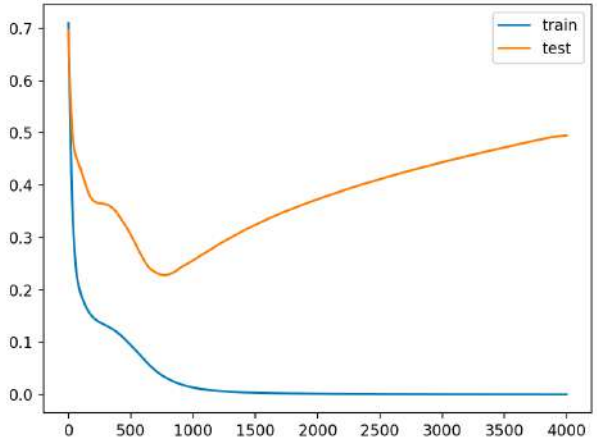
Stop when?
- Converge (no change in loss)
- Max epochs

Important Params:
- #hidden units, #hidden layers
- Learning rate, Momentum, decay
- Seed number, etc.

Neural Networks (cont.): Training

Non-linear relationship





Age	Income	Gender	Province	Corona
25	25,000	Female	Bangkok	Yes
35	50,000	Female	Nontaburi	Yes
32	35,000	Male	Bangkok	No

$$log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{w}_0 + \hat{w}_1 H_1 + \hat{w}_2 H_2 + \hat{w}_3 H_3$$

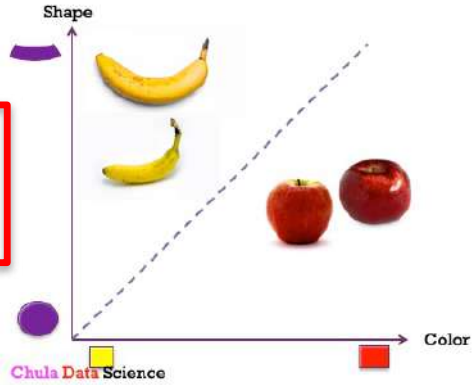
$$H_1 = tanh(\hat{w}_{10} + \hat{w}_{11} x_1 + \hat{w}_{12} x_2)$$

$$H_2 = tanh(\hat{w}_{20} + \hat{w}_{21} x_1 + \hat{w}_{22} x_2)$$

$$H_3 = tanh(\hat{w}_{30} + \hat{w}_{31} x_1 + \hat{w}_{32} x_2)$$

Handcrafted features

Age	Income	Gender	Province	Corona
25	25,000	Female	Bangkok	Yes
35	50,000	Female	Nontaburi	Yes
32	35,000	Male	Bangkok	No

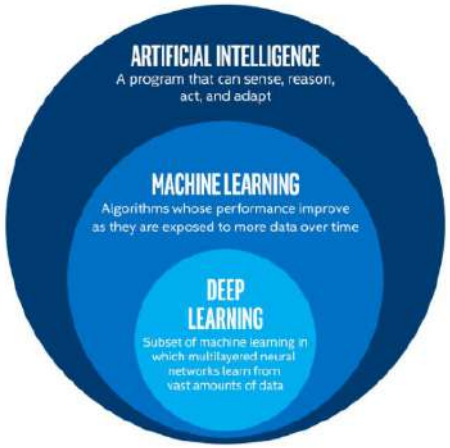


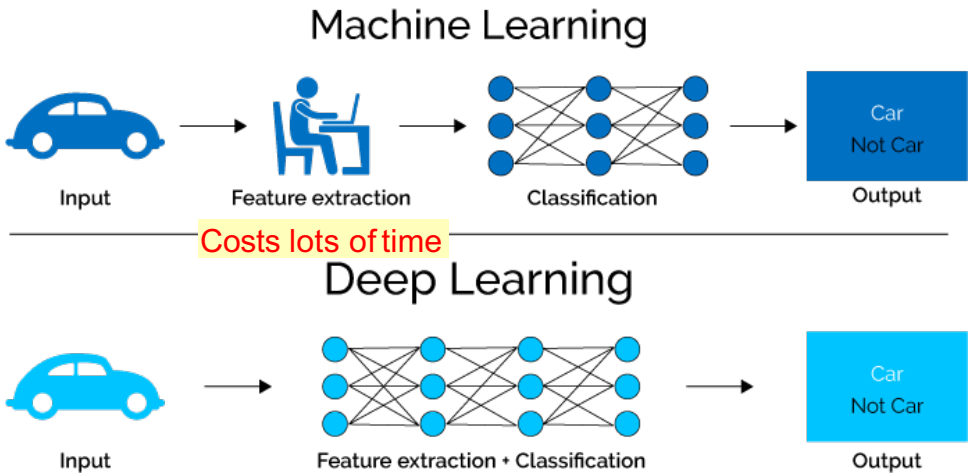
Can we still tell the features (columns)?

What is Deep Learning (DL)?

Part of the machine learning field of learning representations of data. Exceptional effective at learning patterns.

Utilizes learning algorithms that derive meaning out of data by using a hierarchy of multiple layers that mimic the neural networks of our brain.



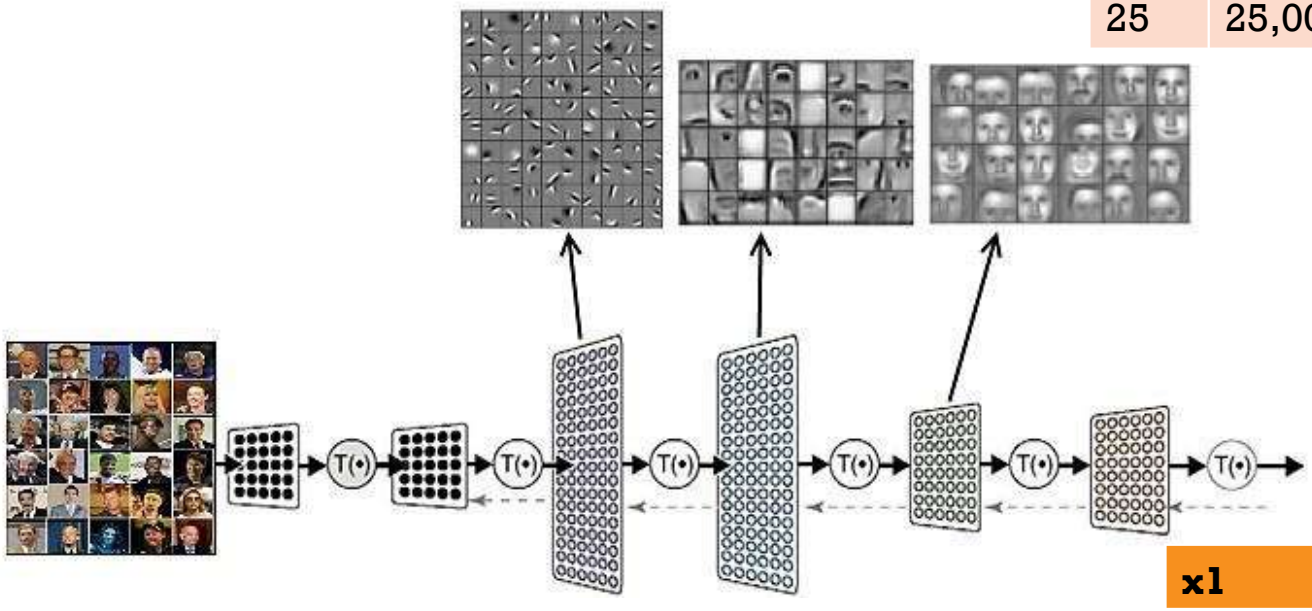


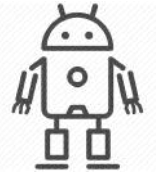
Deep Learning – Basics (cont.)

What did it learn?

A deep neural network consists of a hierarchy of layers, whereby each layer transforms the input data into more abstract representations (e.g., edge -> nose -> face). The output layer combines those features to make predictions.

Age	Income	Gender	Province	Corona
25	25,000	Female	Bangkok	Yes





x1	x2	x3	x4	Corona
0.7	0.2	-0.5	-0.1	Yes

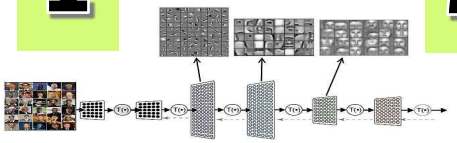
Deep Learning Application

Speech Recognition

Computer Vision

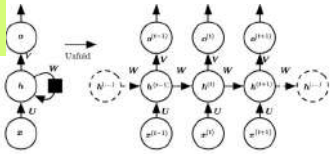
Natural Language Processing

1



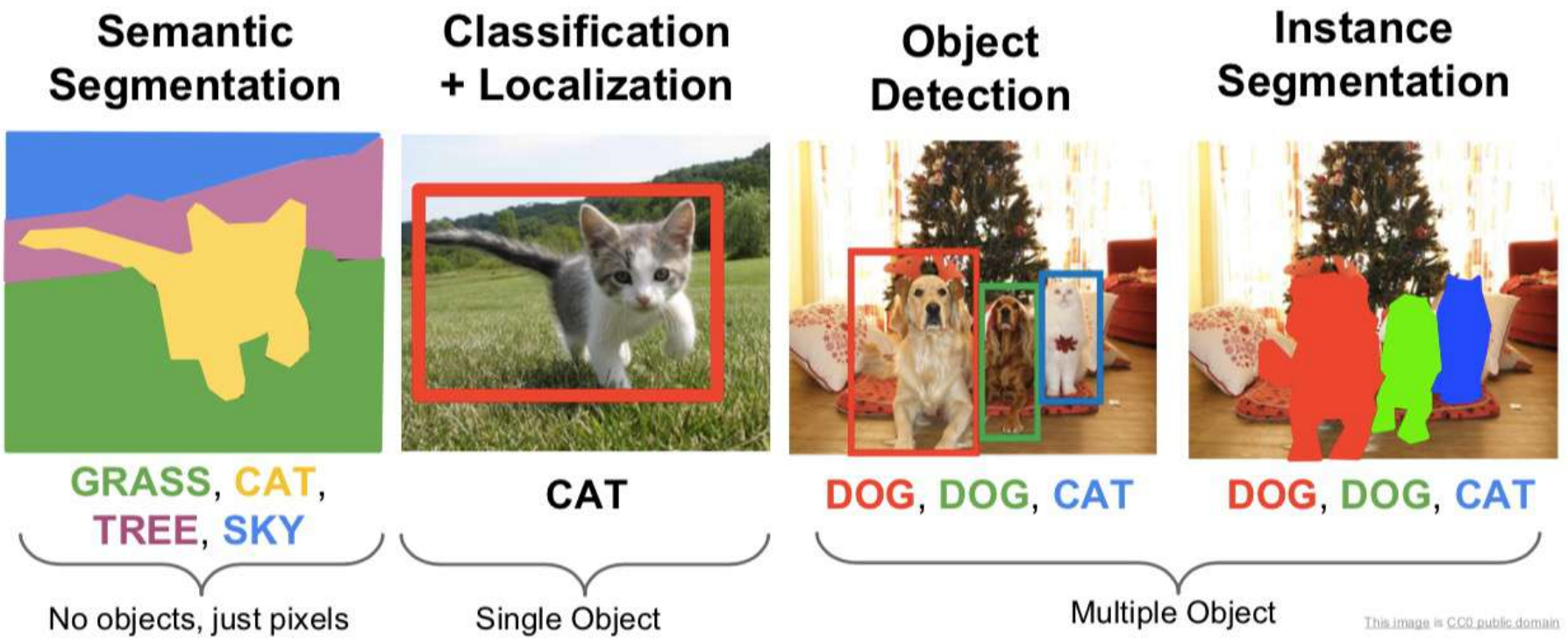
CNN

2



RNN

Type of image tasks

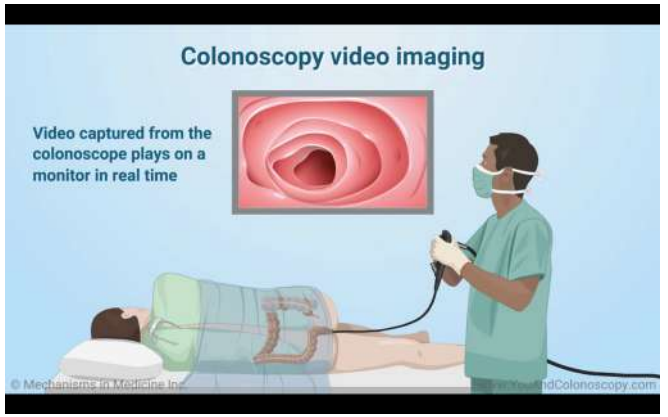


Face Recognition

Smart Medical Diagnosis

CHULA ENGINEERING
Foundation toward Innovation

King Chulalongkorn Memorial Hospital
The Thai Red Cross Society



+

CNN

Outline

- Introduction
- Basic Image Processing
- Overview of CNN
- Image Processing Tasks with SOTA
- Case Study in Polyp Detection

IMAGENET www.image-net.org

The **ImageNet** project is a large visual database designed for use in visual object recognition software research. Over **14M** URLs of images have been hand-annotated by ImageNet to indicate what objects are pictured on **22K** categories.

IMAGENET Large Scale Visual Recognition Challenge

ILSVRC

The Image Classification Challenge:

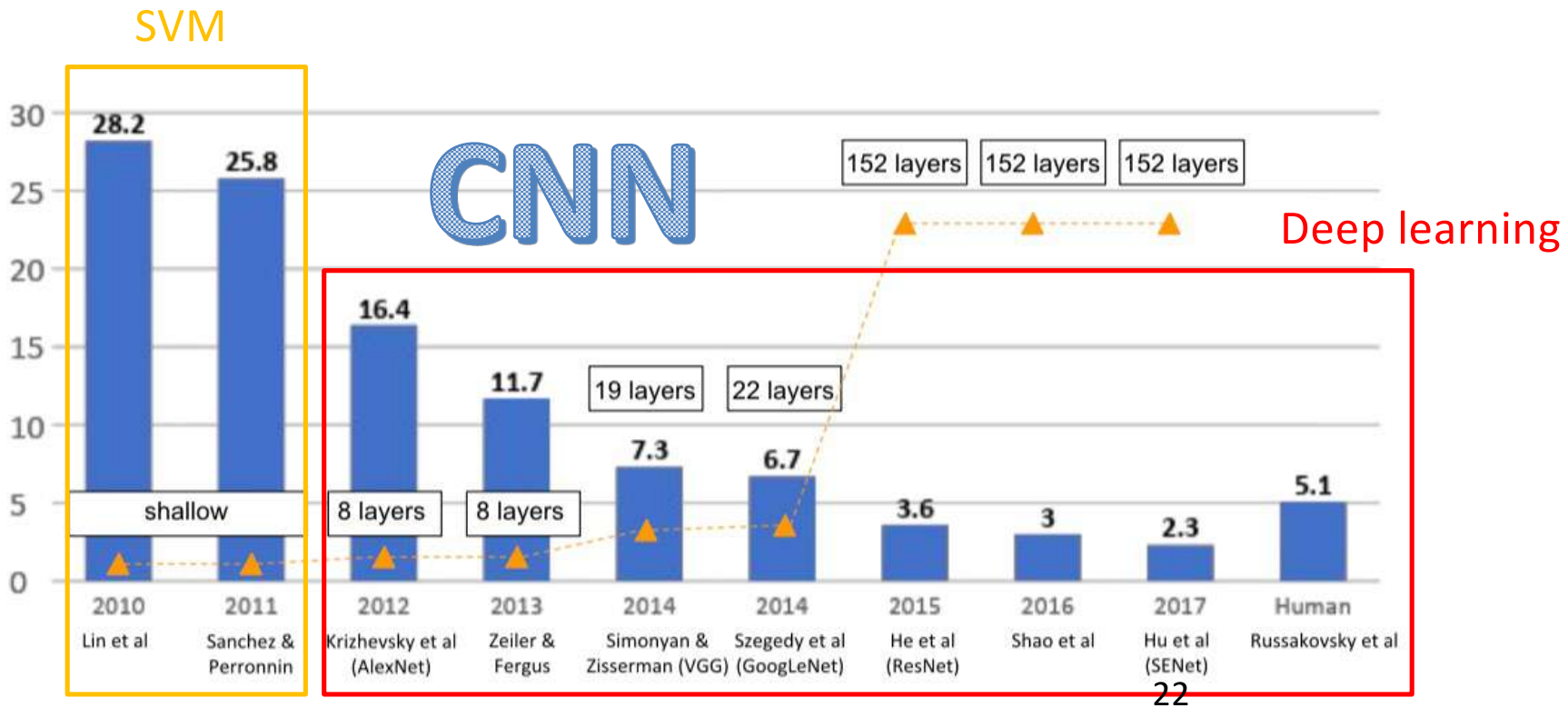
1,000 object classes

1,431,167 images

ImageNet 2017 is the last challenge.

21

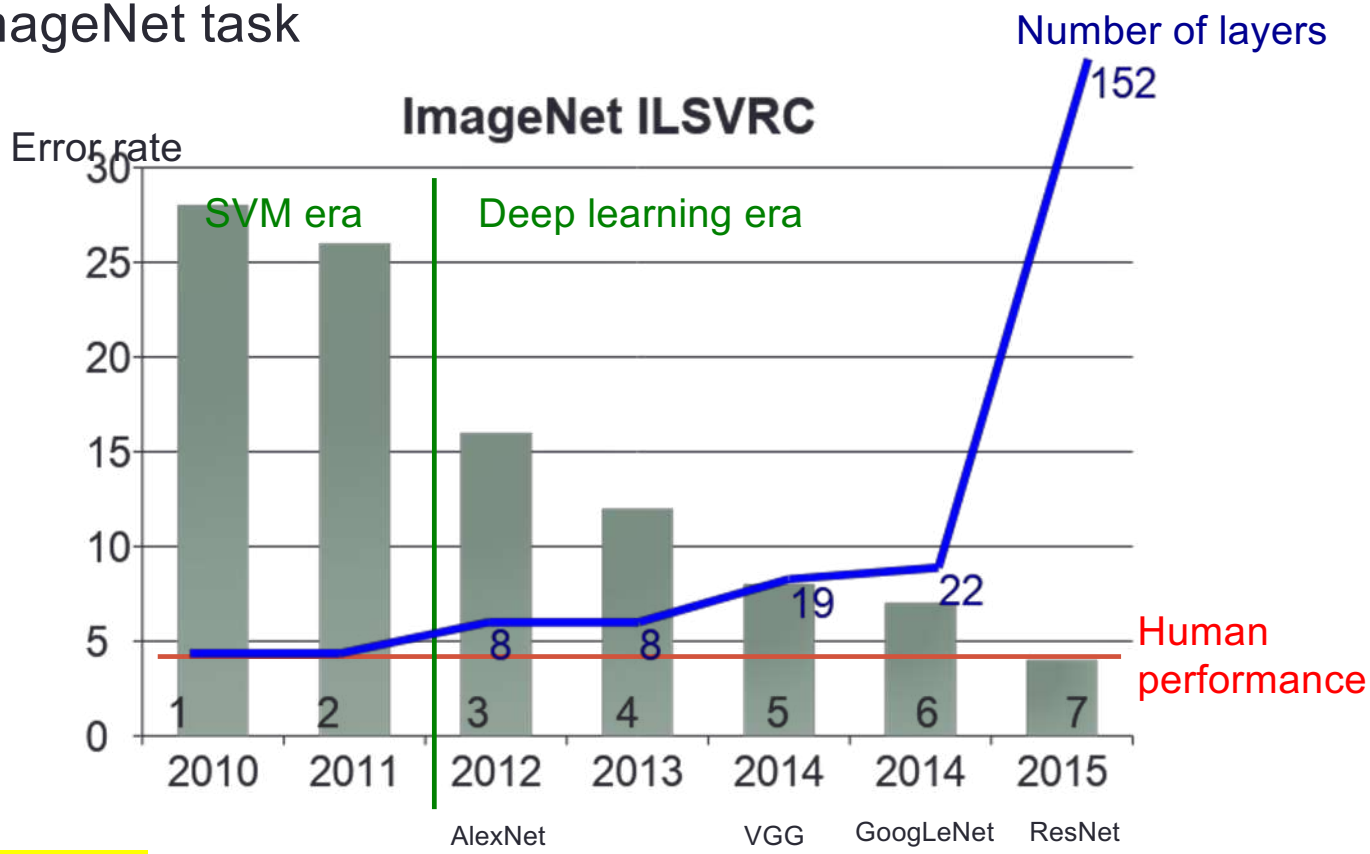
ImageNet Large Scale Visual Recognition Challenge (ILSVRC winners): From SVM to Deep Learning



22

Wider and deeper networks (Beyond Human)

- ImageNet task

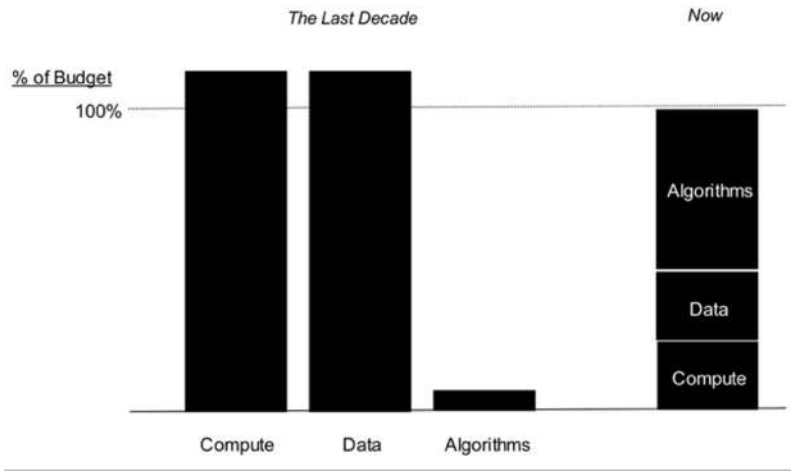


Based on the slide of Aj.Ekapol

Net Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Why now

- Neural Networks has been around since 1990s
- Big data – DNN can take advantage of large amounts of data better than other models
- GPU – Enable training bigger models possible
- Deep – Easier to avoid bad local minima when the model is large



Based on the slide of Aj.Ekapol /www.kdnuggets.com/2017/06/practical-guide-machine-learning-understand-differentiate-apply.html

Application 1: Basic Image Processing

https://softwaredevelopmentperestroika.wordpress.com/2014/02/11/image-processing-with-python-numpy-scipy-image-convolution/

ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย

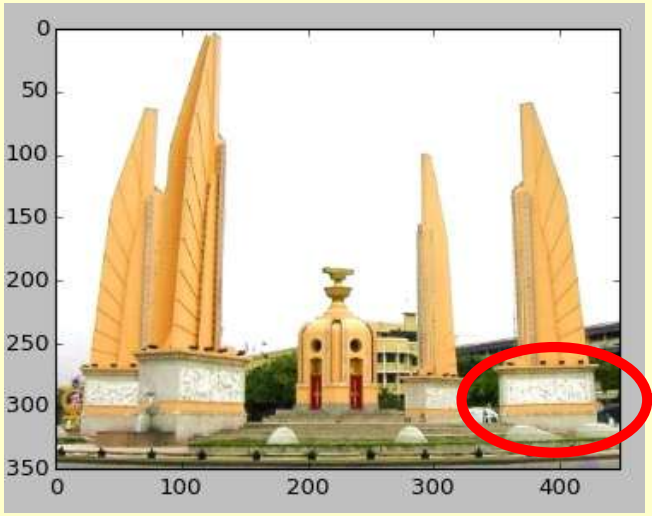
การแสดงรูปภาพใน python ด้วย matplotlib

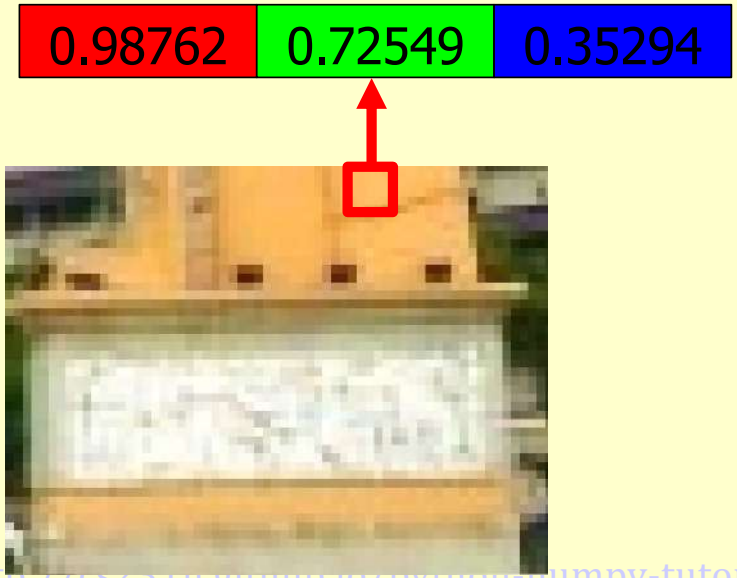
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

image = mpimg.imread('monument.png')
plt.imshow(image)

plt.show()
```

เพื่อความง่ายโปรแกรมที่จะเขียนจากนี้ไปใช้กับแฟ้ม png เท่านั้น

image เป็นอาเรย์ที่แต่ละช่อง
มีค่า 0 ถึง 1 แทนความเข้มของสี

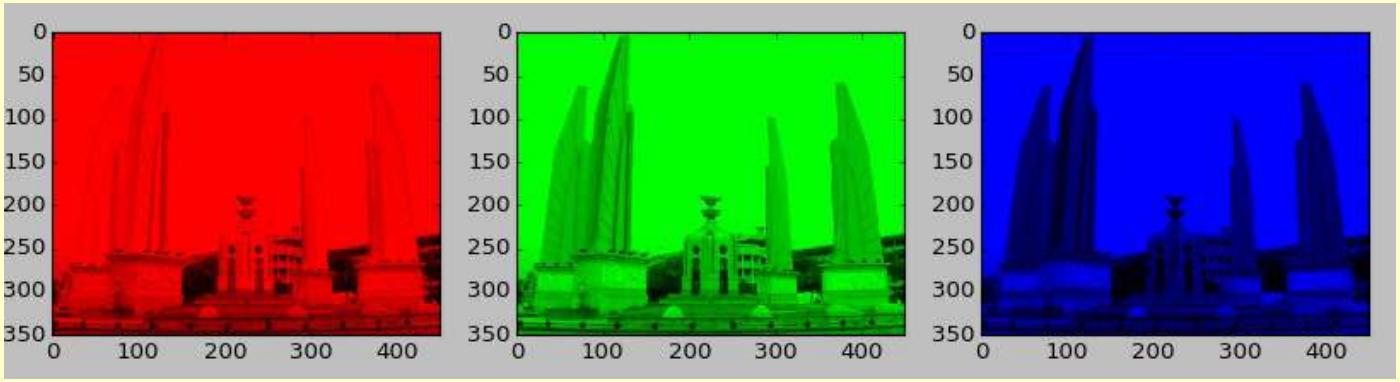
1 pixel = 1 จุดสี มี 3 สีย่อย





1 ภาพ เป็นอาเรย์ 3 สามมิติเก็บค่าความเข้มของ R G B

```
>>> image.shape
(350, 449, 3)
```



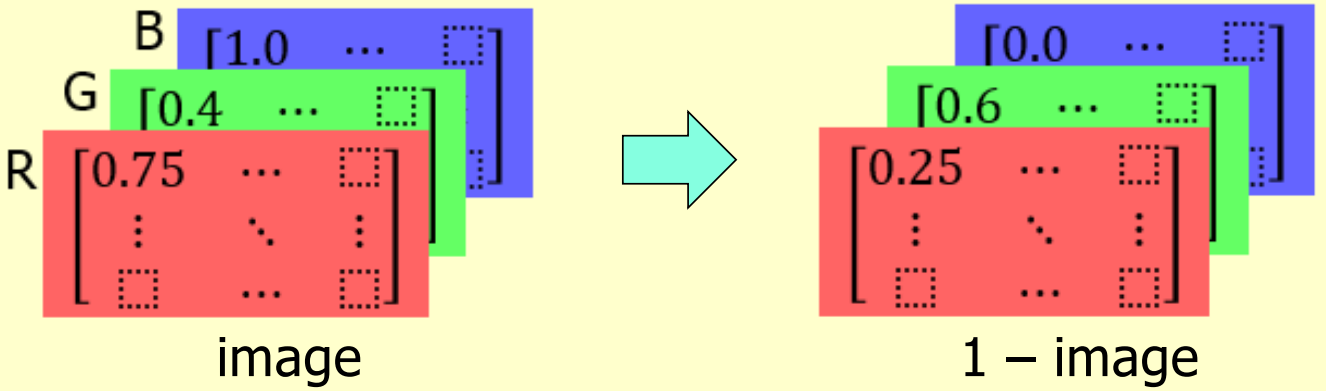
อาร์เรย์ 2 มิติสีแดง
image[:,:,0]

อาร์เรย์ 2 มิติสีเขียว
image[:,:,1]

อาร์เรย์ 2 มิติสีน้ำเงิน
image[:,:,2]

การประมวลผลภาพเบื้องต้น

- image = mpimg.imread('monument.png')
- image.shape → (350, 449, 3)
- image = 1 – image (broadcast & element-wise op.)

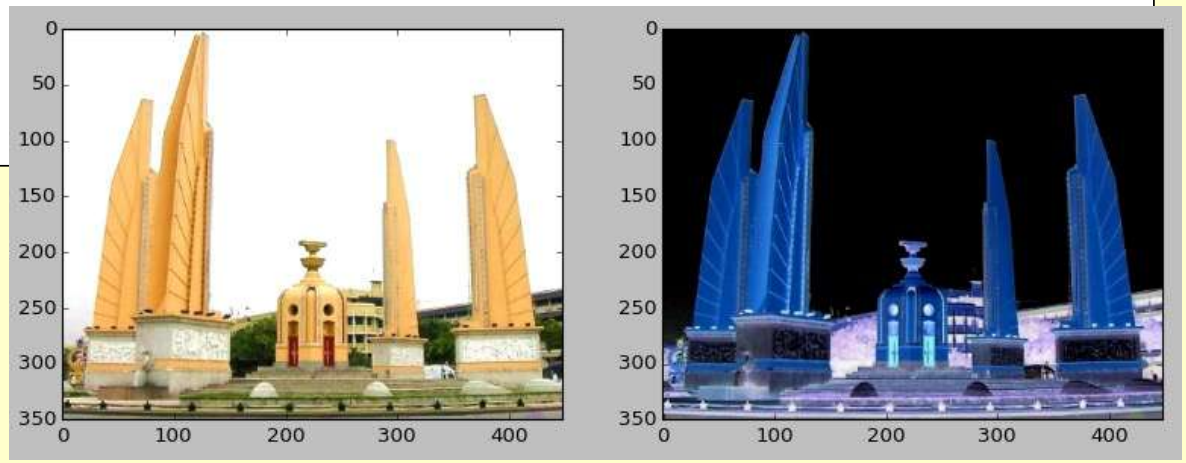


ทำแบบนี้แล้ว ภาพเปลี่ยนแปลงไปอย่างไร ?

การประมวลผลภาพเบื้องต้น : ภาพ Negative

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

image = mpimg.imread('monument.png')
plt.subplot(1, 2, 1)
plt.imshow(image)

negative = 1 - image
plt.subplot(1, 2, 2)
plt.imshow(negative)

plt.show()
```

1 – image คือนำค่าทุกช่องไปลบออกจาก 1
ความเข้มสีเปลี่ยนเป็นตรงข้าม
(ขาว → ดำ, เหลือง → น้ำเงิน, ...)
broadcast & element-wise subtract

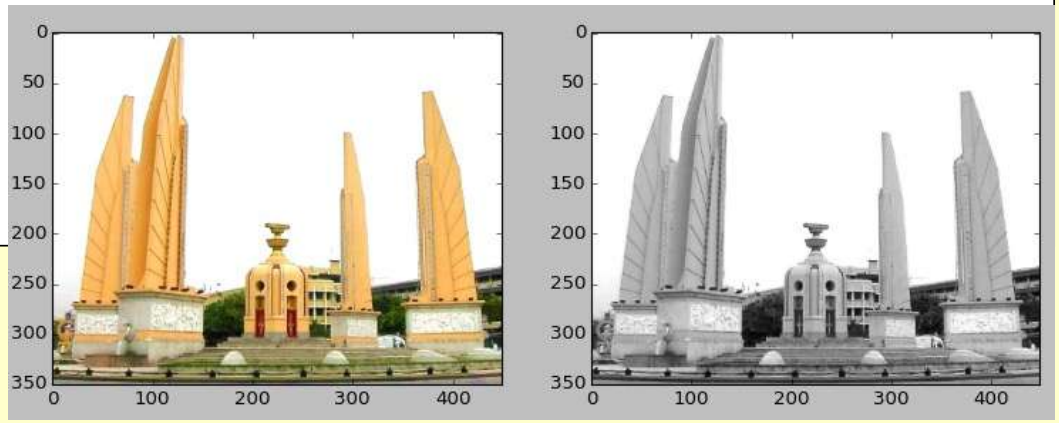


การประมวลผลภาพเบื้องต้น : ภาพสีเทา

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

image = mpimg.imread('monument.png')
plt.subplot(1, 2, 1)
plt.imshow(image)
gray = np.ndarray(image.shape)
gray[:,:,0] = \
gray[:,:,1] = \
gray[:,:,2] = (image[:,:,0]+image[:,:,1]+image[:,:,2]) / 3
plt.subplot(1, 2, 2)
plt.imshow(gray)

plt.show()
```

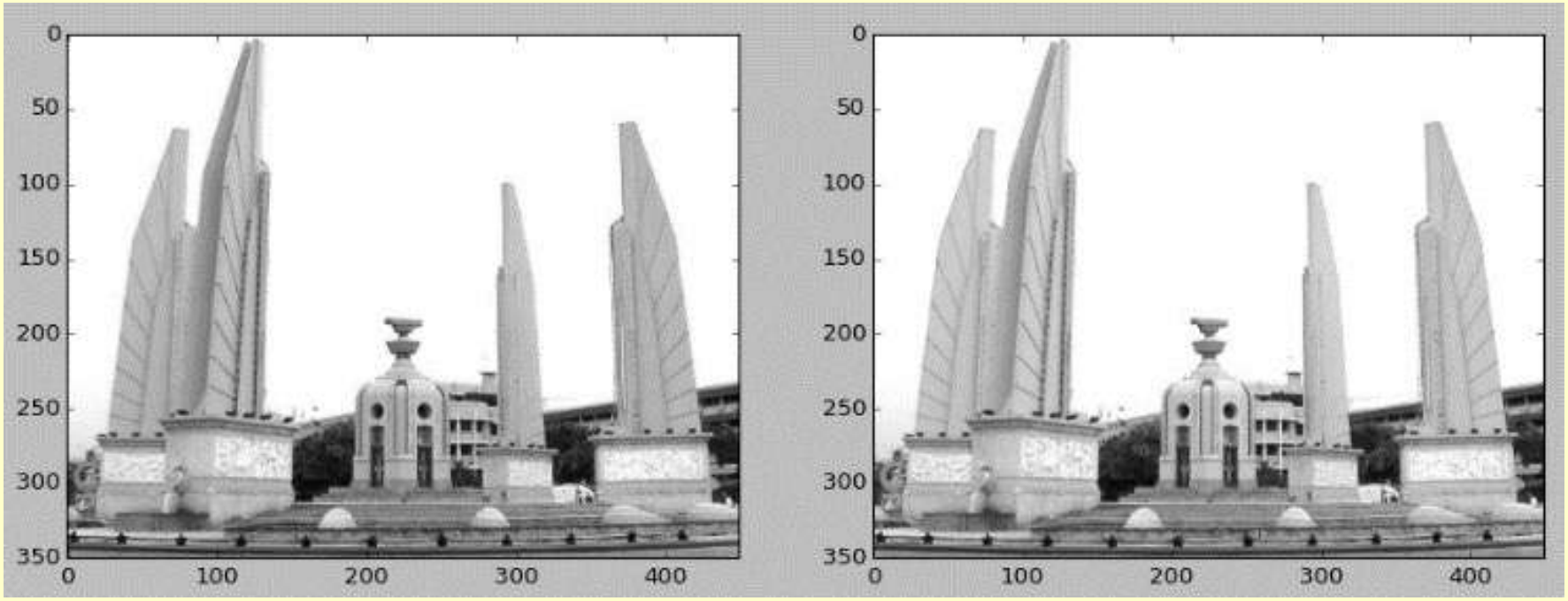
จุดสีที่ R G B มีค่าความเข้มเท่ากัน ได้จุดสีเทา

นำค่าความเข้มของ R G B มาหาค่าเฉลี่ย แล้วเปลี่ยน R G B ให้เป็นความเข้มระดับเดียวกัน

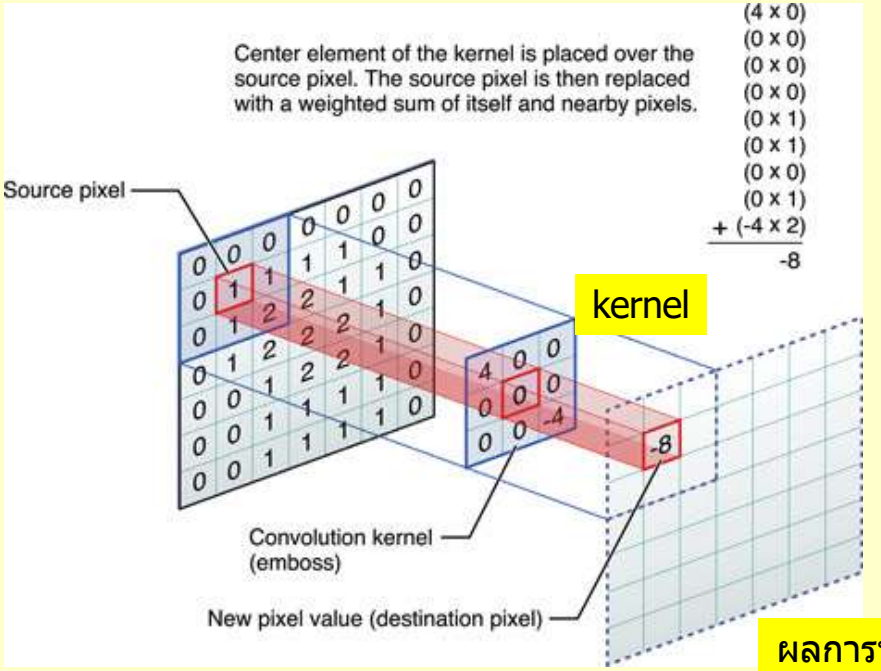


สีเทา เขาว่าสูตรนี้ดีกว่า 0.299*R + 0.587*G + 0.114*B

- CH08_6 จงเขียนโปรแกรมเพื่อแปลงรูปภาพจากรูปสีให้เป็นรูปสีเทา (gray scale) ด้วยสูตรข้างบนนี้ เทียบกับสูตร (R + G + B)/3



Convolution

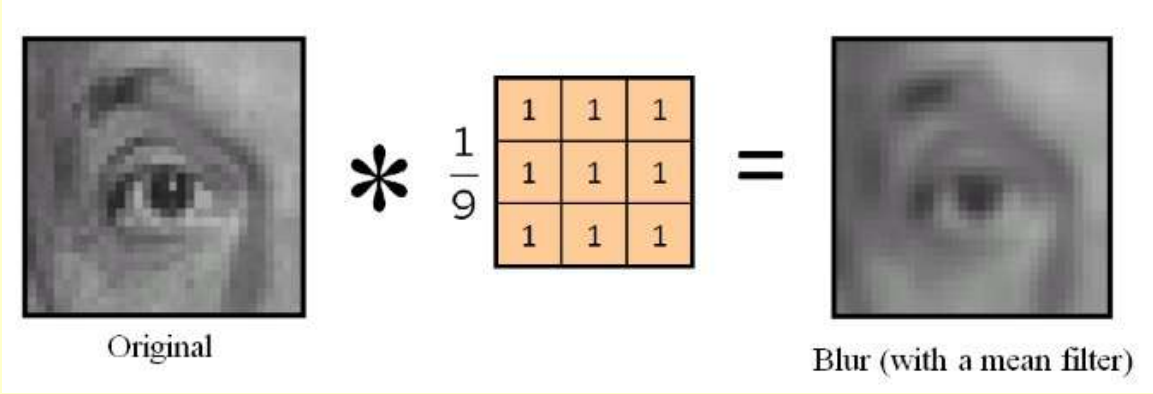


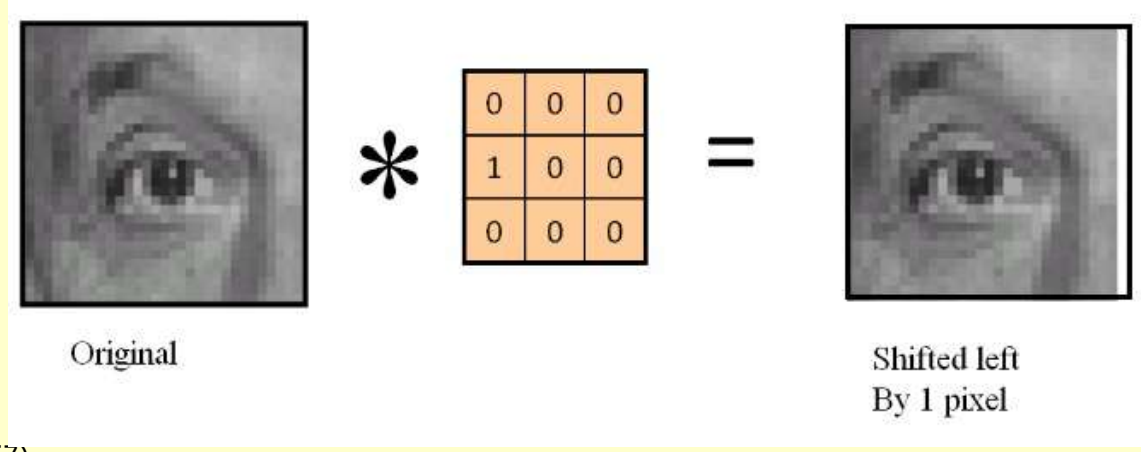
ผลการทำ convolute

เปลี่ยนค่าของเมทริกซ์ kernel
จะได้การประมวลผลภาพแบบอื่น ๆ

Image Convolution

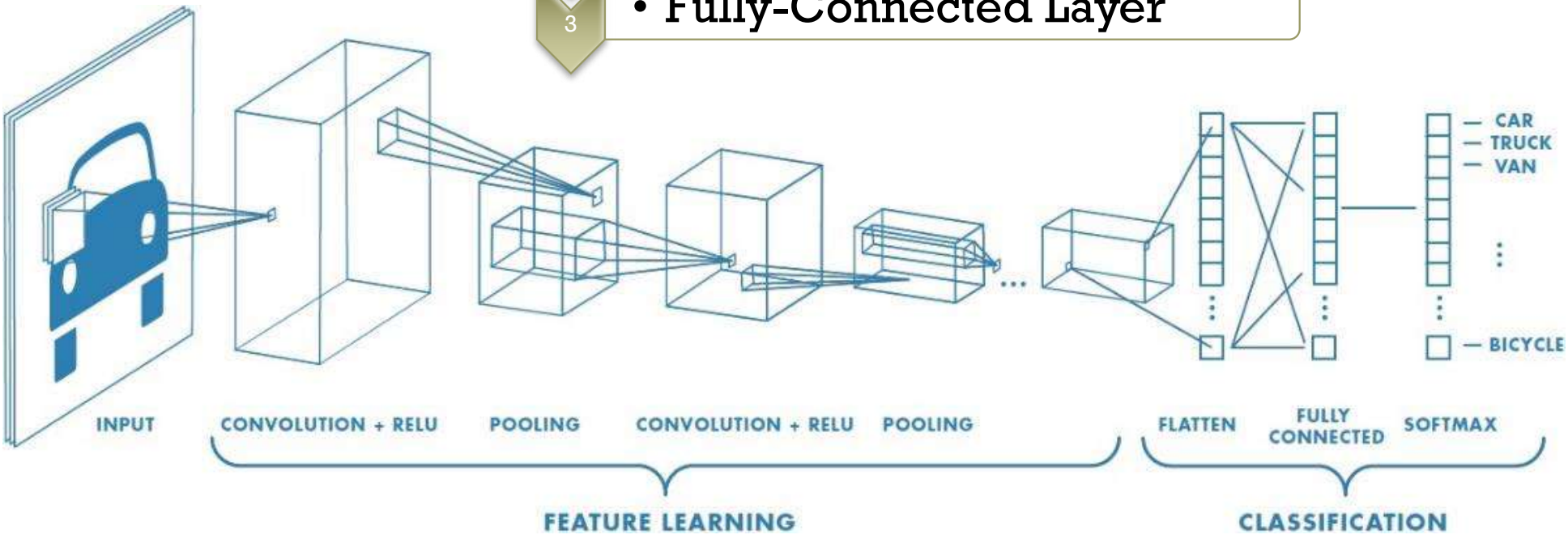
- Image Convolution คือการนำรูปภาพมาผ่านตัวกรอง (kernel) เพื่อให้ได้ผลลัพธ์ตามที่ต้องการ เช่น blur, ขยับรูป, จับขอบรูป เป็นต้น



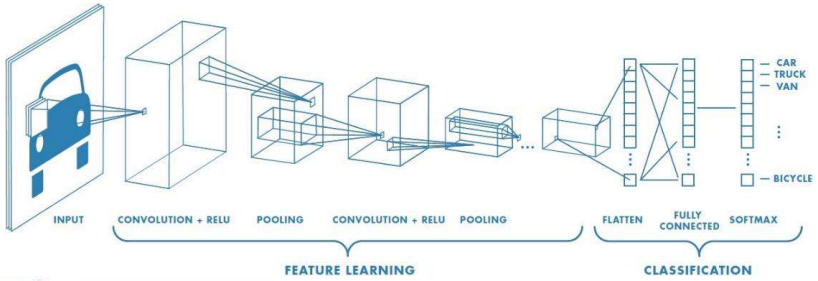


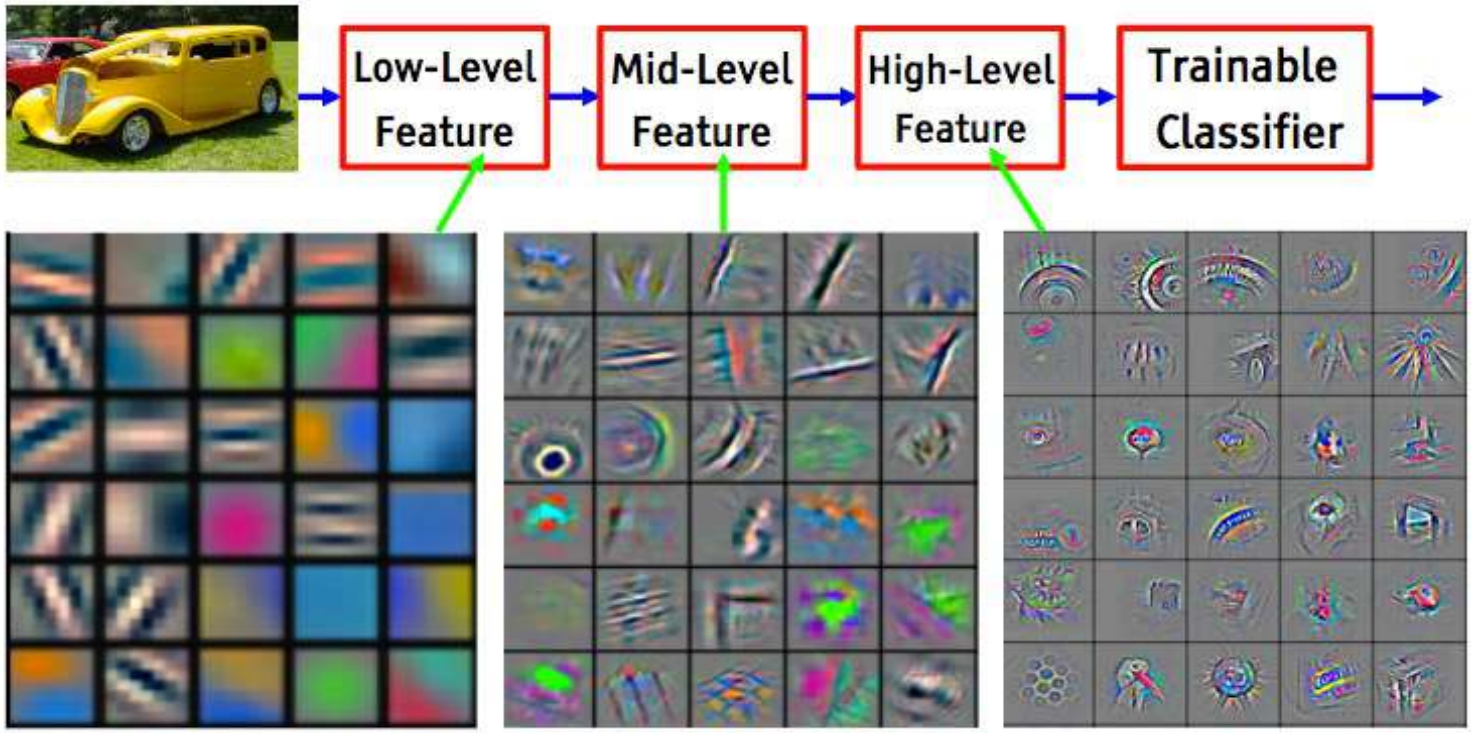
Convolutional Neural Networks (CNN)

1. • Convolutional Layer
2. • Pooling Layer
3. • Fully-Connected Layer



CNN (cont.)

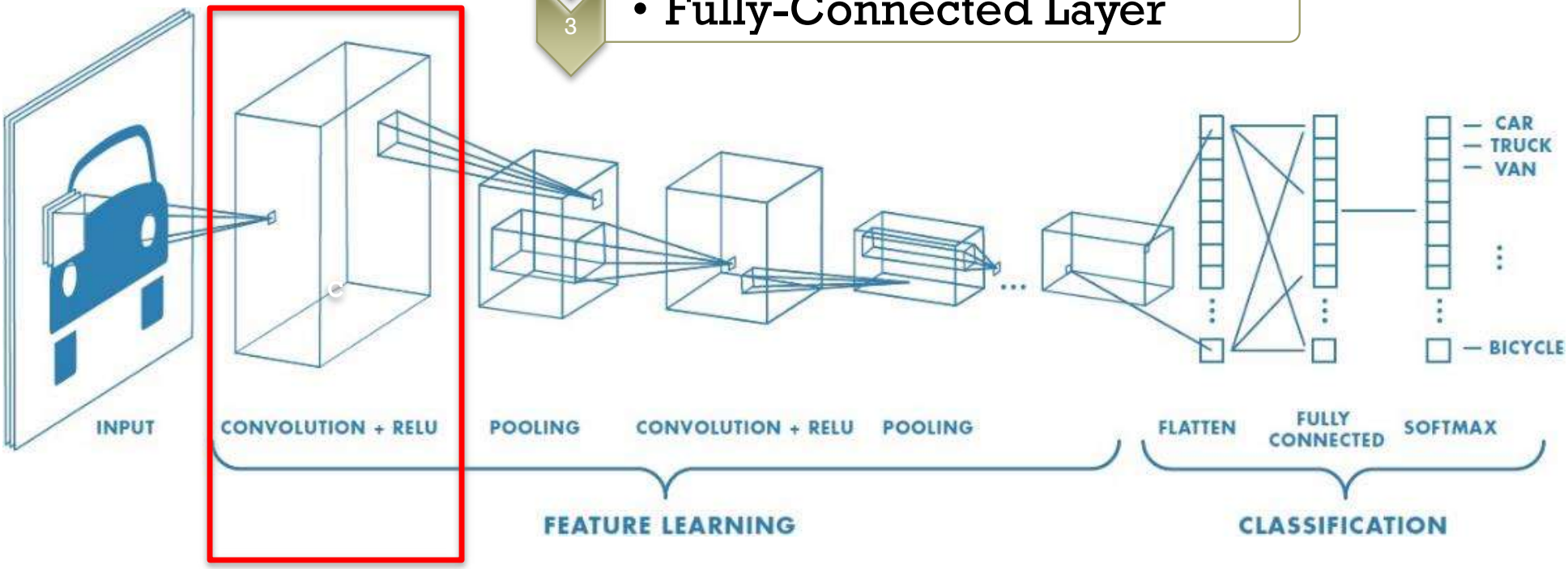

CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION


Low-Level Feature
Mid-Level Feature
High-Level Feature
Trainable Classifier

Convolutional Neural Networks (CNN)

1) Convolutional Layer

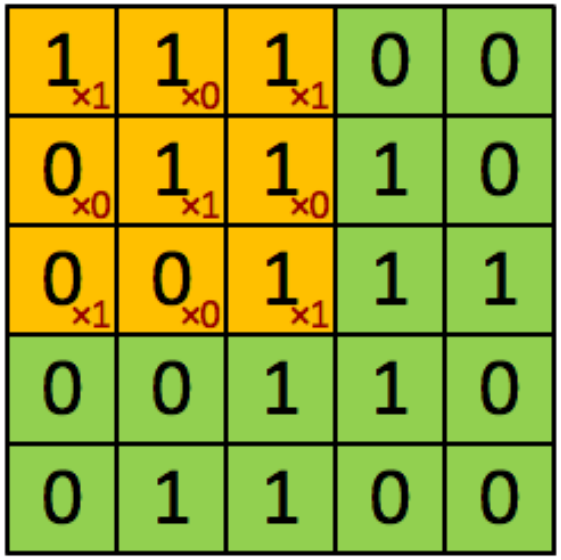
1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer



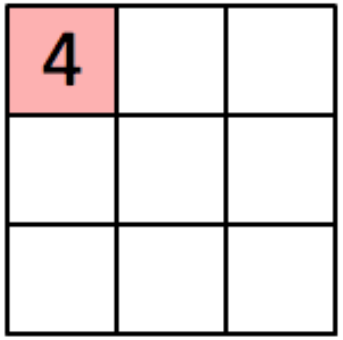
Layer 1: Convolutional Layer

(3*3 filter)

1	0	1
0	1	0
1	0	1

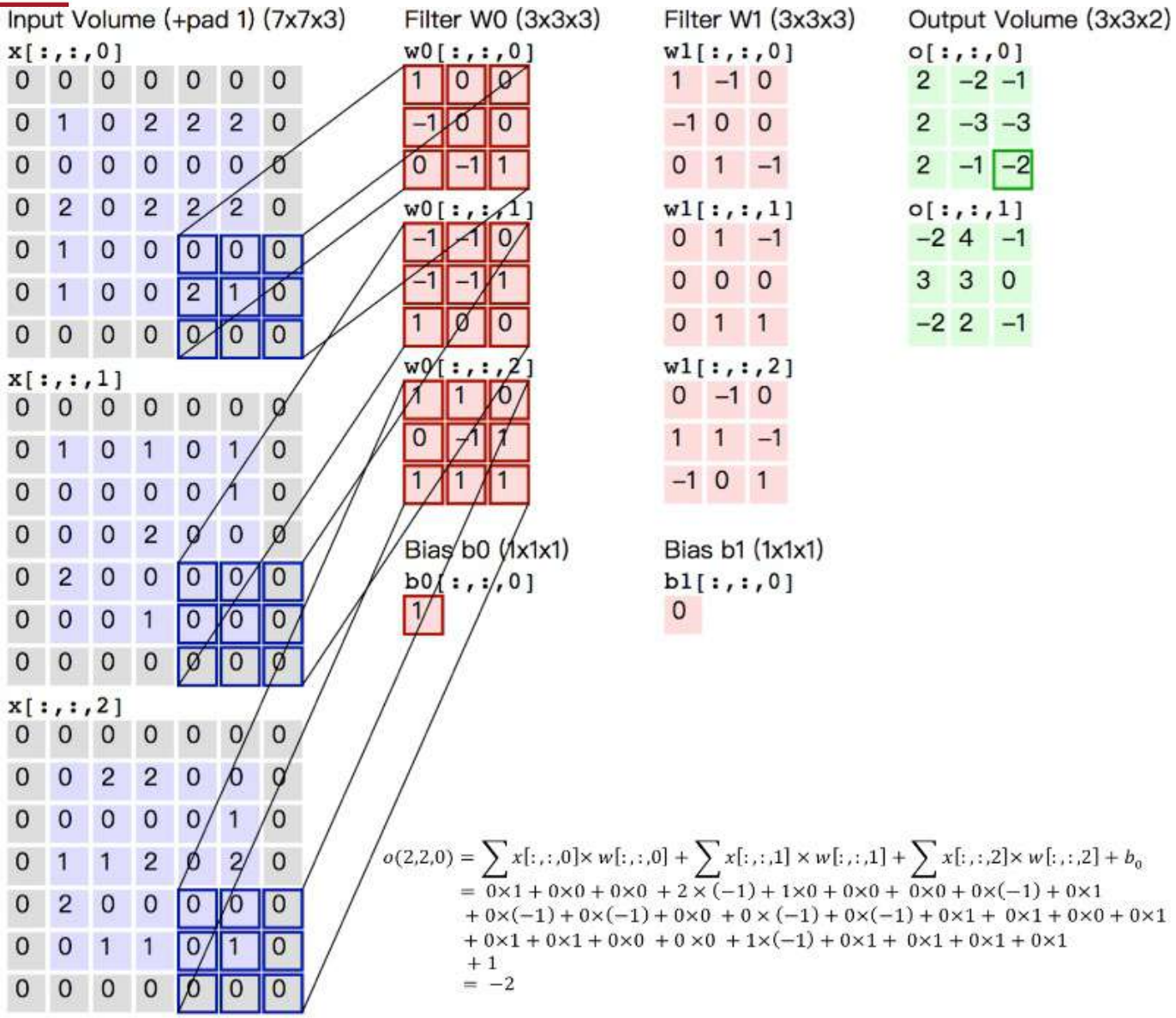


Image



Convolved Feature

1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer



$$o(2,2,0) = \sum x[:,:,0] \times w[:,:,0] + \sum x[:,:,1] \times w[:,:,1] + \sum x[:,:,2] \times w[:,:,2] + b_0$$
$$= 0\times1 + 0\times0 + 0\times0 + 2\times(-1) + 1\times0 + 0\times0 + 0\times0 + 0\times(-1) + 0\times1$$
$$+ 0\times(-1) + 0\times(-1) + 0\times0 + 0\times(-1) + 0\times(-1) + 0\times1 + 0\times1 + 0\times0 + 0\times1$$
$$+ 0\times1 + 0\times1 + 0\times0 + 0\times0 + 1\times(-1) + 0\times1 + 0\times1 + 0\times1 + 0\times1$$
$$+ 1$$
$$= -2$$

CNNs: 1) Convolutional Layer

Feature Extraction

http://cs231n.github.io/convolutional-networks/

$$w^T x + b$$

(3*3 filter)

1	0	1
0	1	0
1	0	1

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

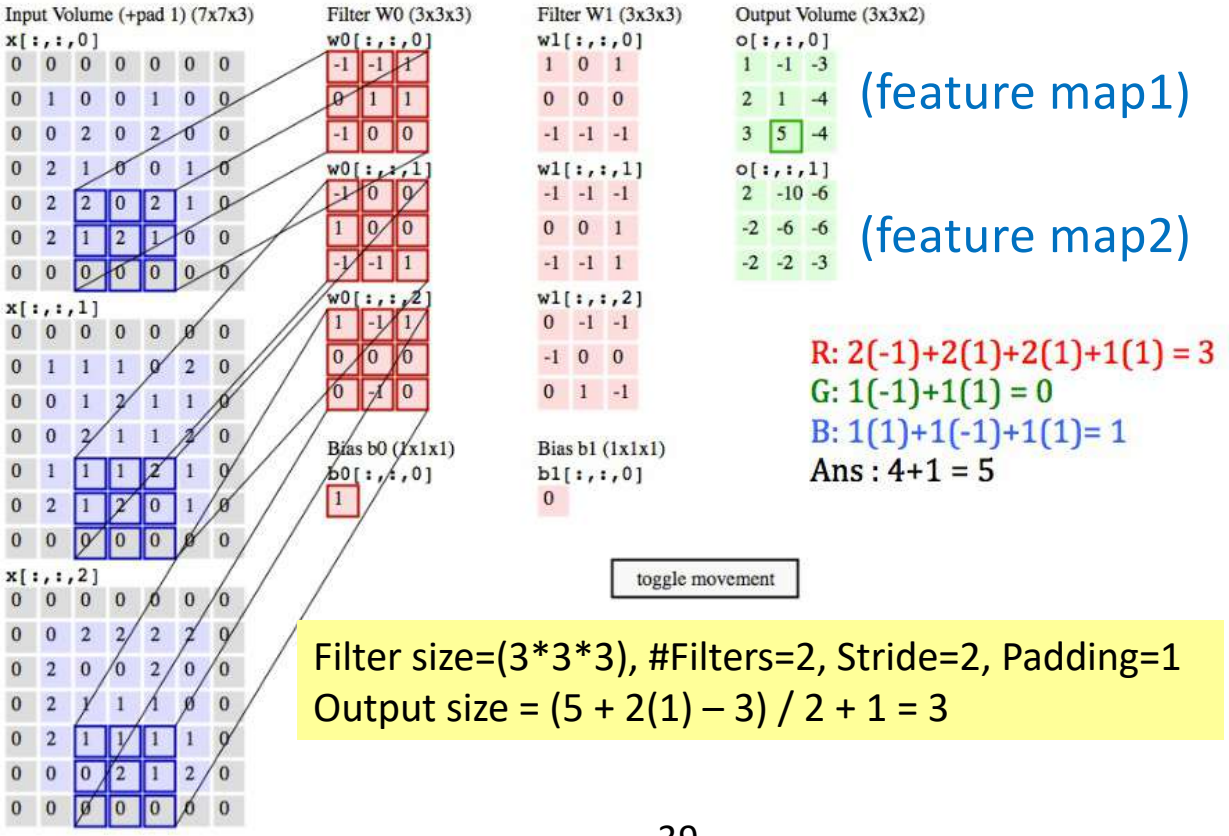
Convolved Feature (feature map)

Filter size=(3*3), #Filters=1, Stride=1, Padding=0
Output size = (5 +2(0) – 3) / 1 + 1 = 3

Summary: **4 parameters** for convolutional layer

- (1) Filter size, (2) #Filters, (3) Stride, (4) Padding

Output size = (N + 2*P – F) / S + 1



(feature map1)

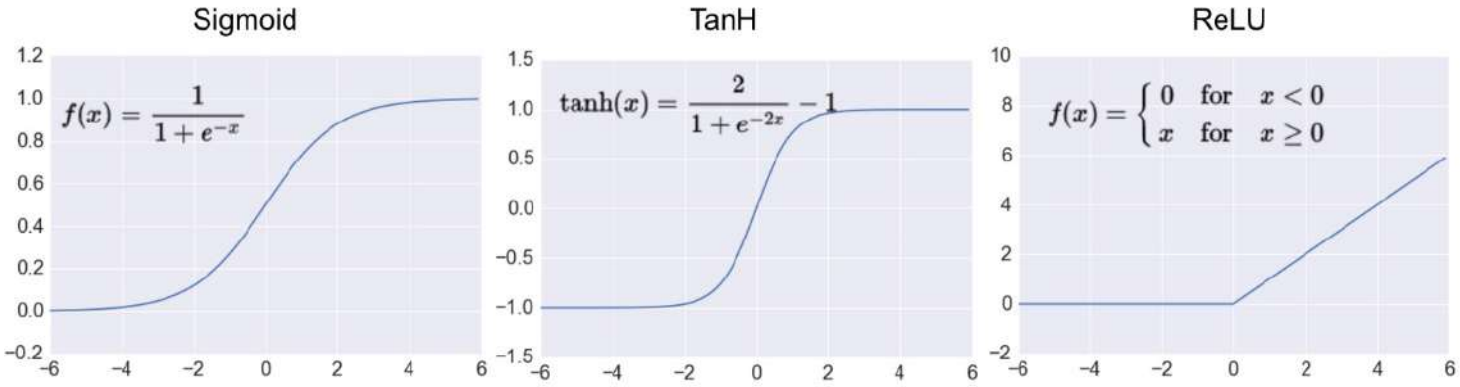
(feature map2)

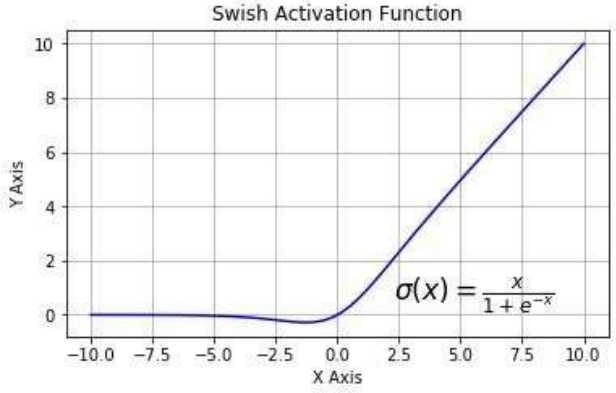
R: 2(-1)+2(1)+2(1)+1(1) = 3
G: 1(-1)+1(1) = 0
B: 1(1)+1(-1)+1(1)= 1
Ans : 4+1 = 5

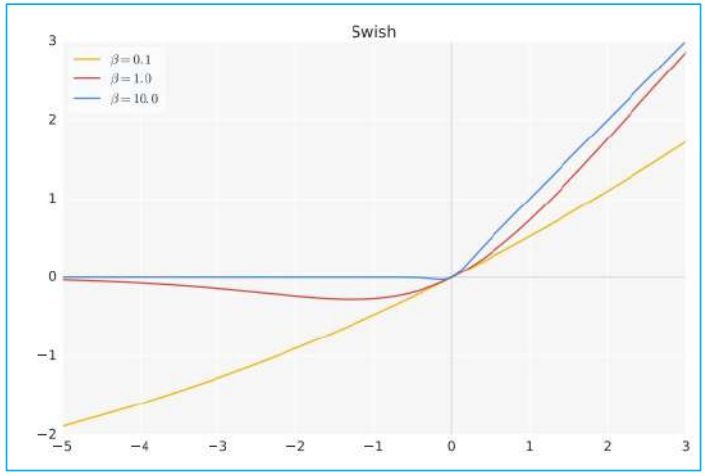
Filter size=(3*3*3), #Filters=2, Stride=2, Padding=1
Output size = (5 + 2(1) – 3) / 2 + 1 = 3

CNNs: 1) Convolutional Layer (cont.)

Non-Linear Activation Function



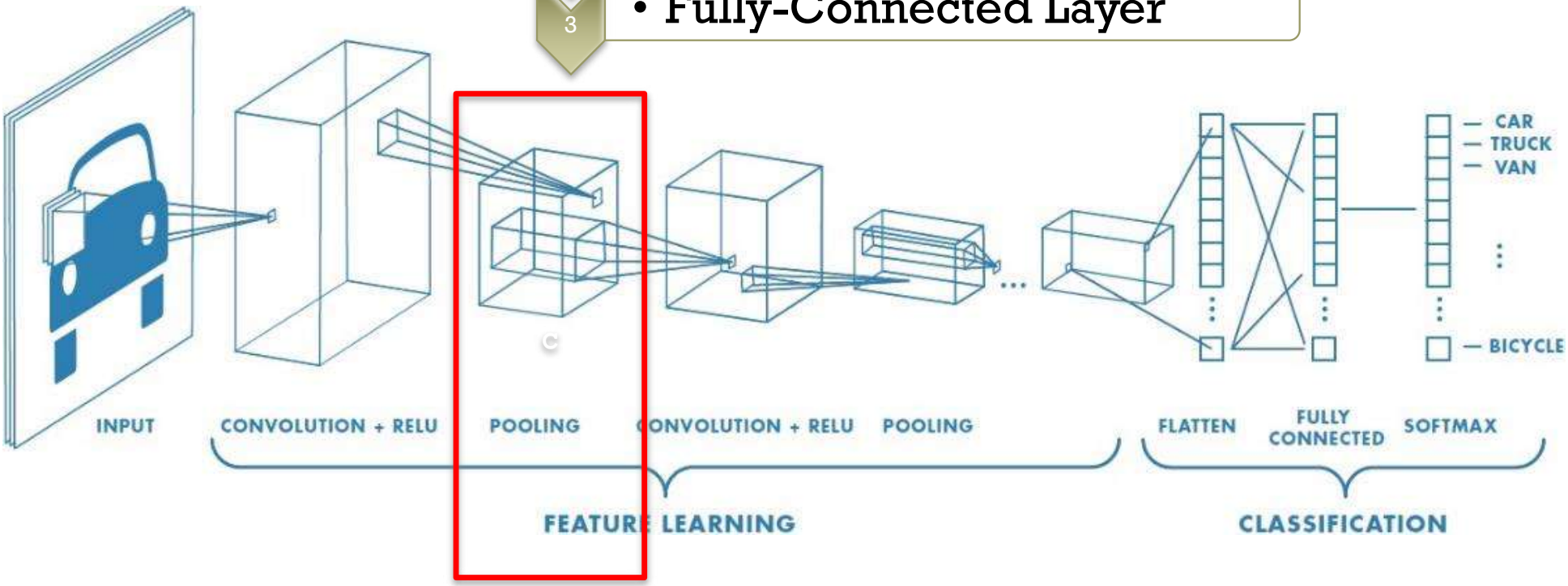




Convolutional Neural Networks (CNN)

2) Pooling Layer

1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer



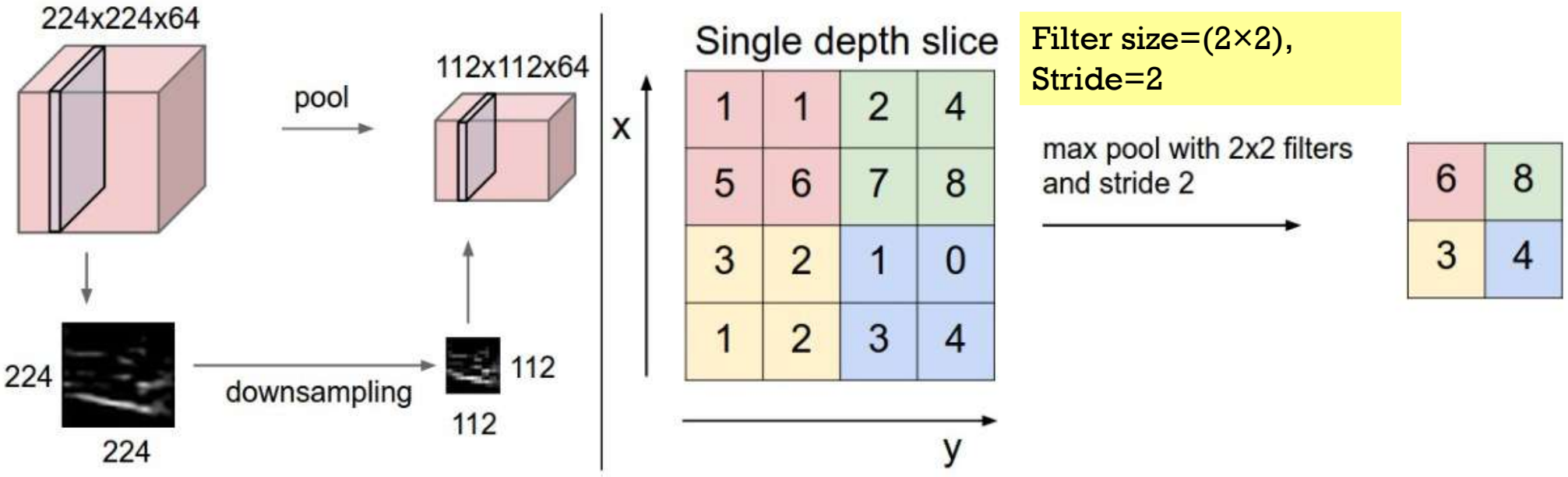
CNNs: 2) Pooling Layer

http://cs231n.github.io/convolutional-networks/

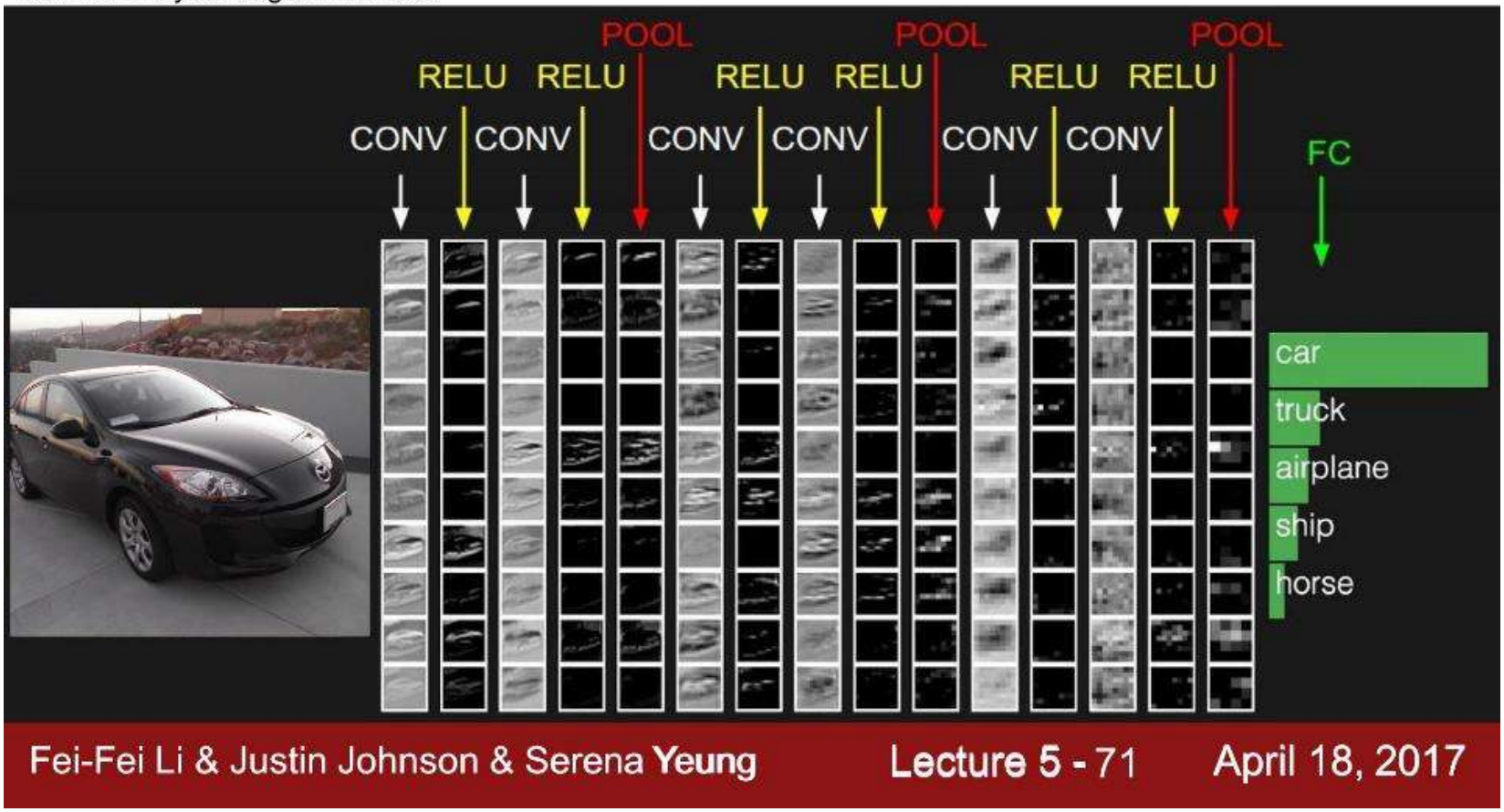
Summary: **2 parameters** for pooling layer

- Filter size, Stride

- Feature selection (reduction); downsampling



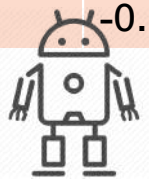
two more layers to go: POOL/FC



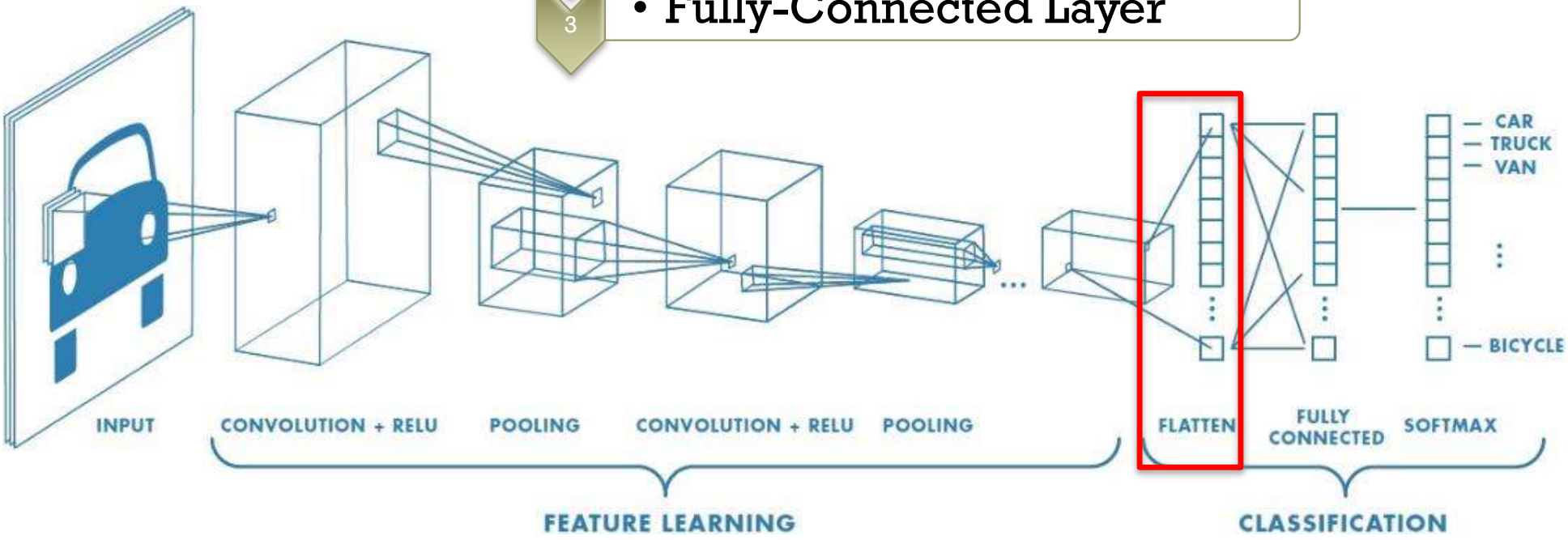
Convolutional Neural Networks

Embedding Vector

x1	x2	x3	x4	Corona
0.7	0.2	-0.5	-0.1	Yes



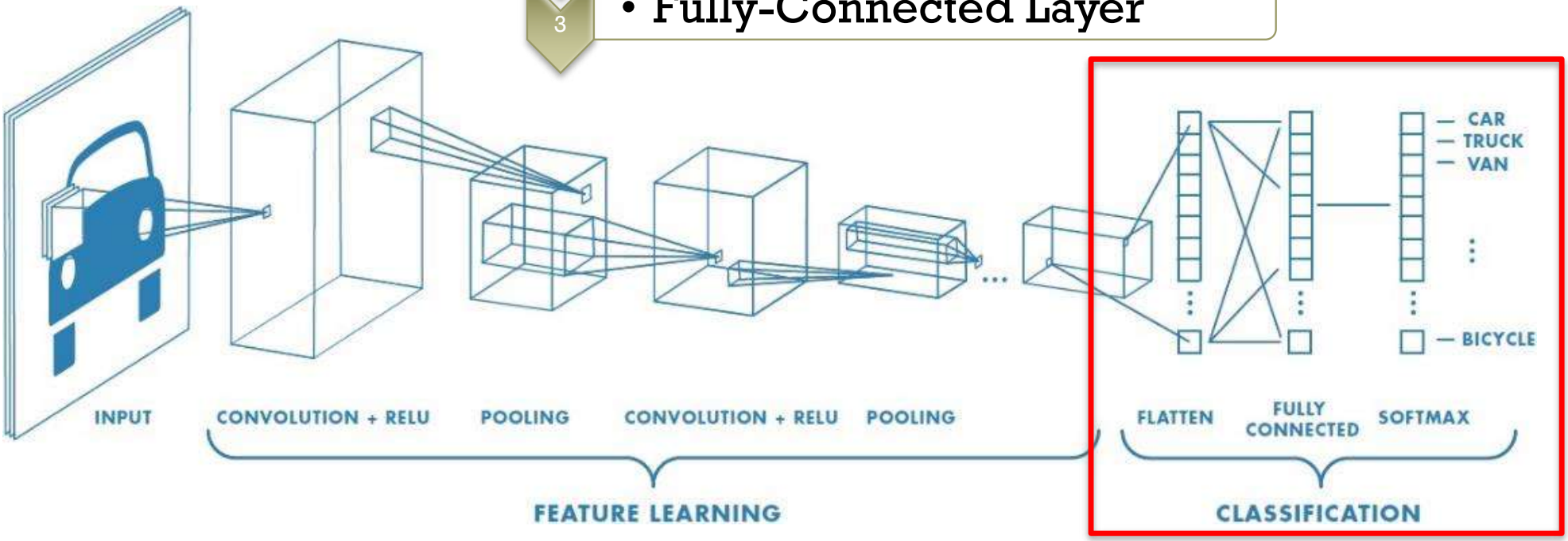
1. • Convolutional Layer
2. • Pooling Layer
3. • Fully-Connected Layer



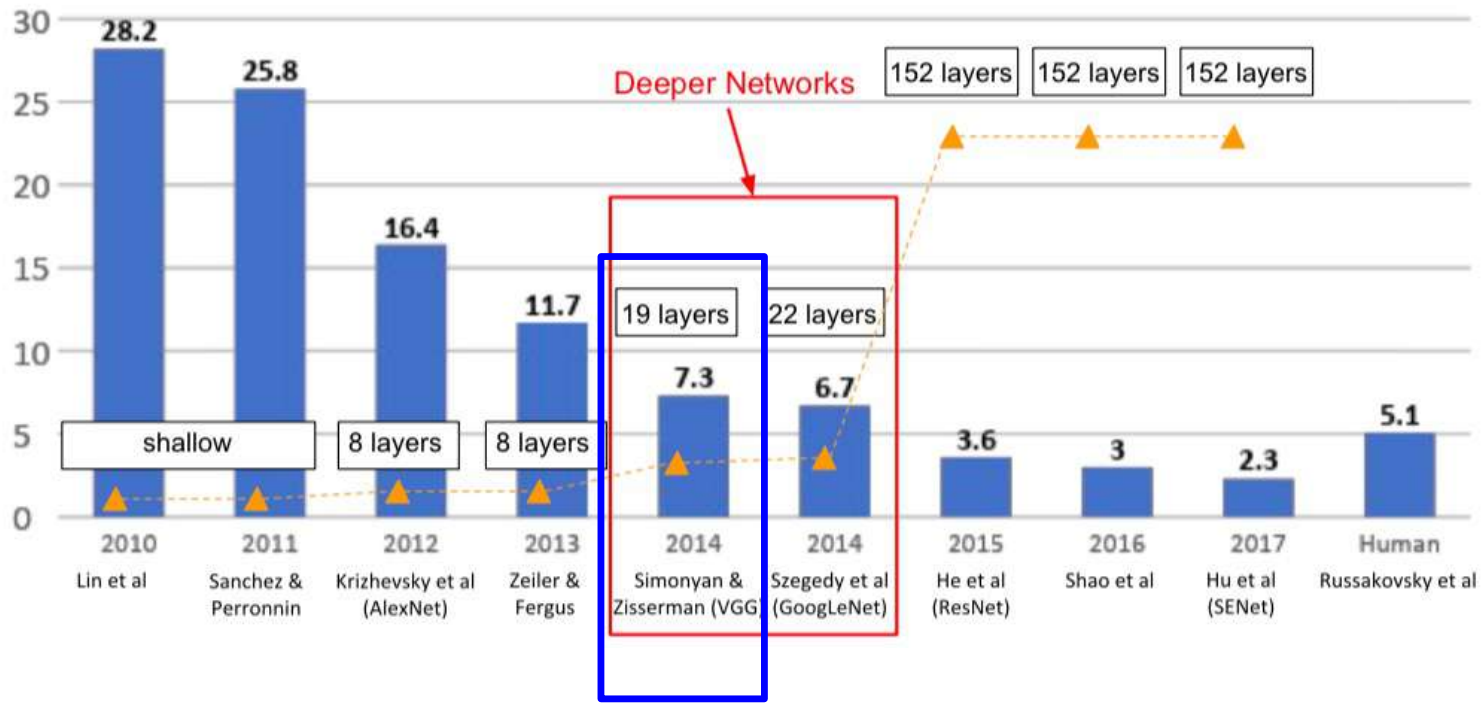
Convolutional Neural Networks (CNN)

3) Fully-Connected Layer (FC) = Neural Networks

1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer



ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2nd runner-up in 2014): VGG19



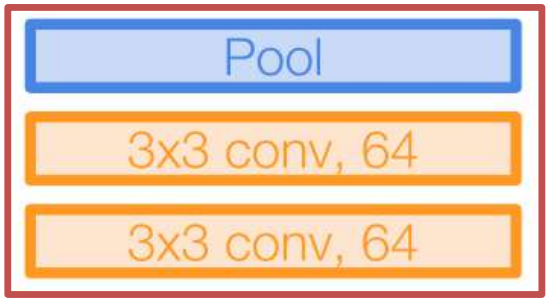
Case Study: VGGNet (2014)

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

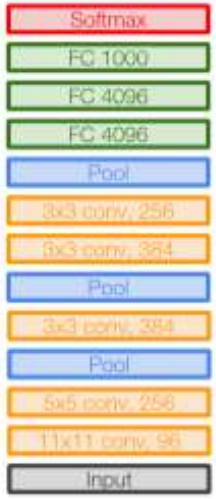
8 layers (AlexNet) → 16-19 layers (VGG16Net)

Only 3x3 CONV Stride 1, pad 1
And 2x2 MAX POOL stride 2

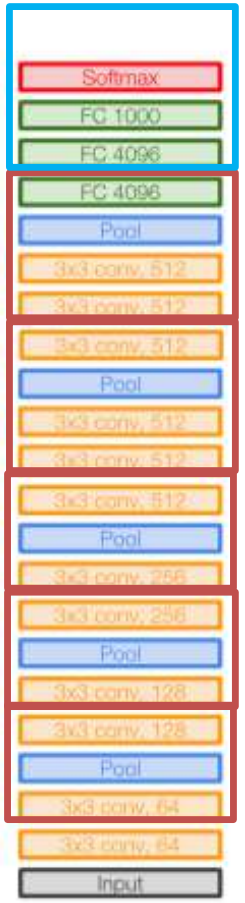


1) Feature extraction block

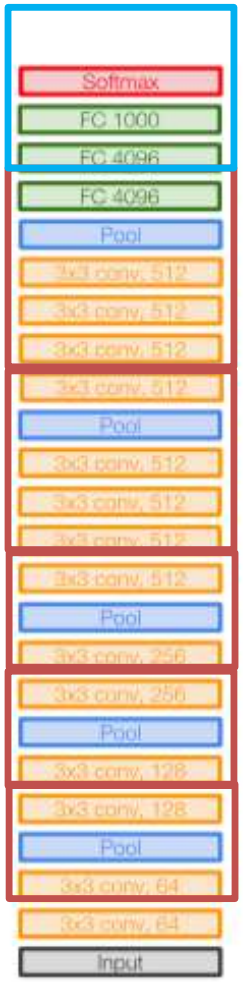
2) Classification block



AlexNet



VGG16



VGG19

Case Study: VGGNet (2014)

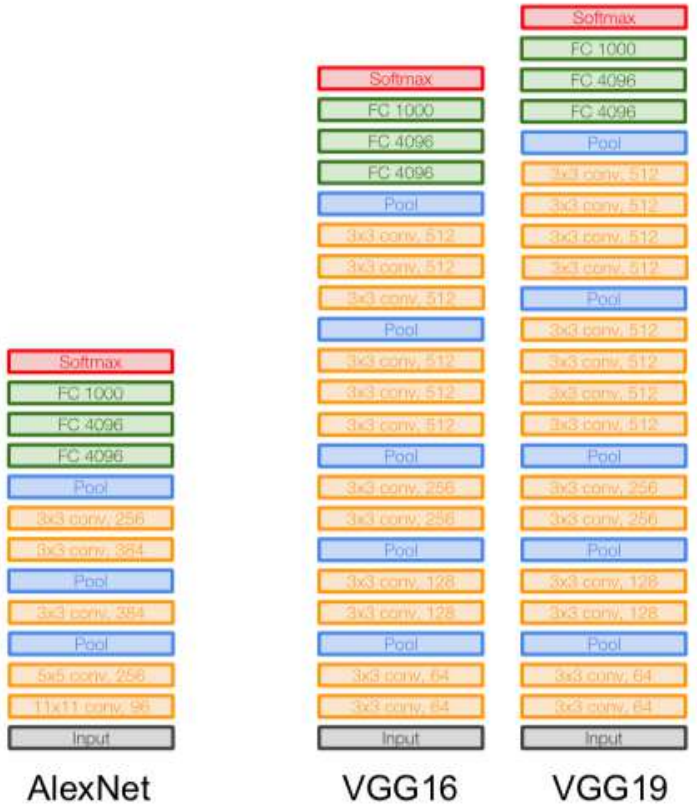
[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3×3 conv (stride 1) layers has same **effective receptive field** as one 7×7 conv layer in ZFNet

But deeper, more non-linearities

And fewer parameters: $3 \times (3^2C^2)$ vs. 7^2C^2 for C channels per layer



Keras

```
#VGG Block
conv1 = Conv2D(64, (3,3), strides=(1,1), padding='same', activation='ReLU')(x)
conv2 = Conv2D(64, (3,3), strides=(1,1), padding='same', activation='ReLU')(conv1)
maxpool1 = MaxPooling2D((2,2))(conv2)
```

Padding='valid'
No padding

48

Variants of VGG

- **ConvNet configurations**
  - The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold)
- The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩"

- ILSVRC'14 2nd in classification, 1st in localization
- Use VGG16 or VGG 19 (VGG19 only slightly better, more memory)

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					

VGG16 VGG19

49

11/04/2018 | DEEP LEARNING, IMAGE PROCESSING, MATHEMATICS, POST, TECHNOLOGY

Calculate the number of parameters on VGG Net

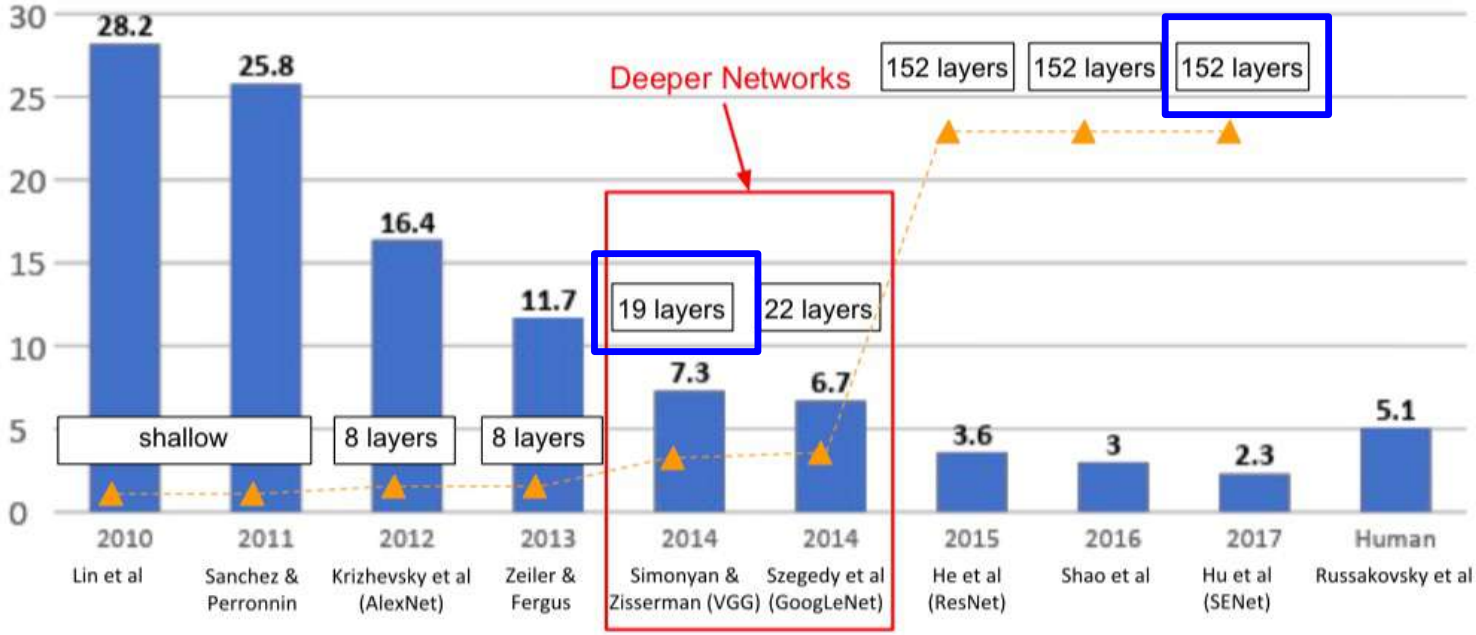
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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

Network	A	B	C	D	E
Number of parameters (Millions)	133	133	134	138	144

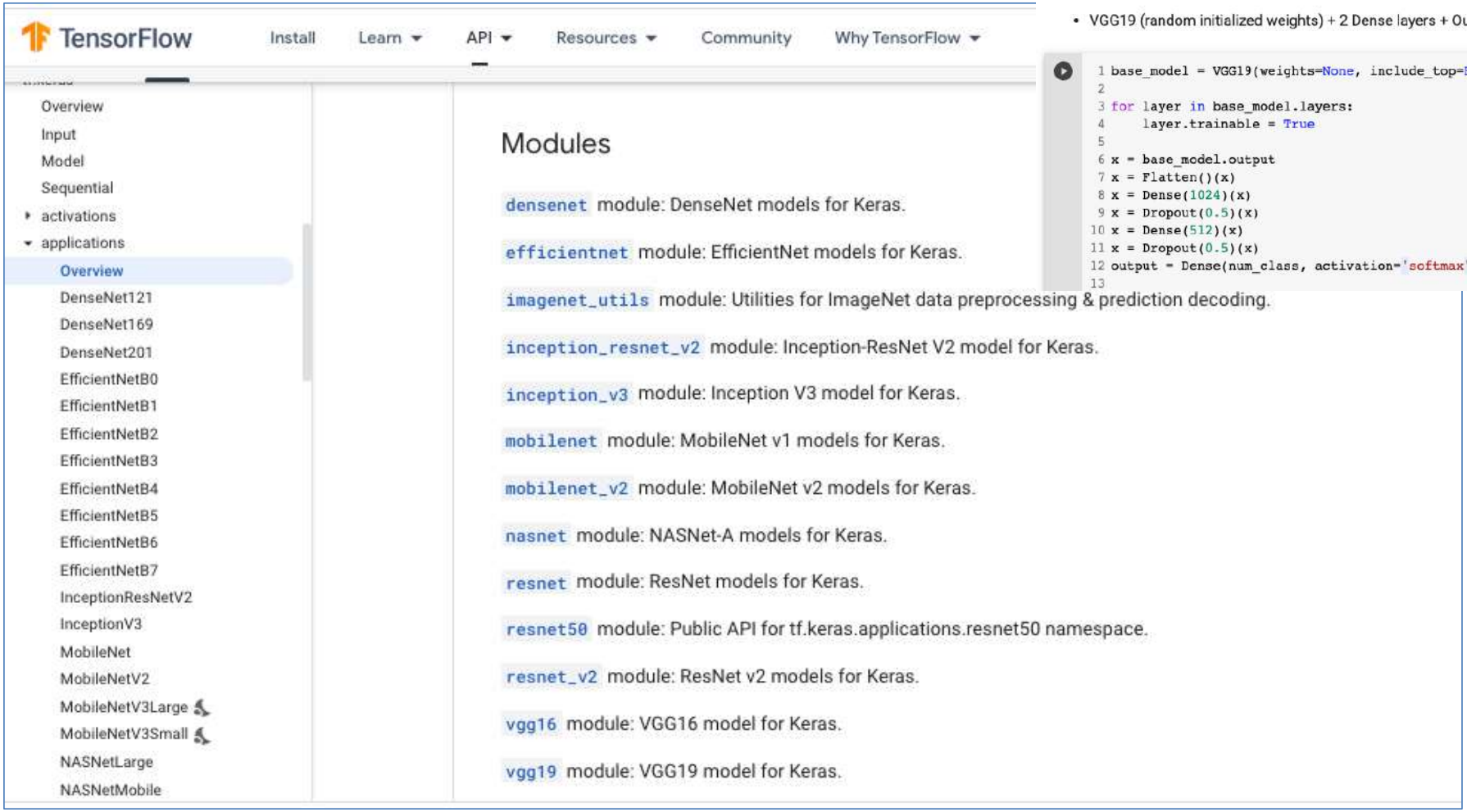
https://hoanglehaithanh.com/calculate-the-number-of-parameters-on-vgg-net/

50

ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2nd runner-up in 2014): Deeper than VGG19



51



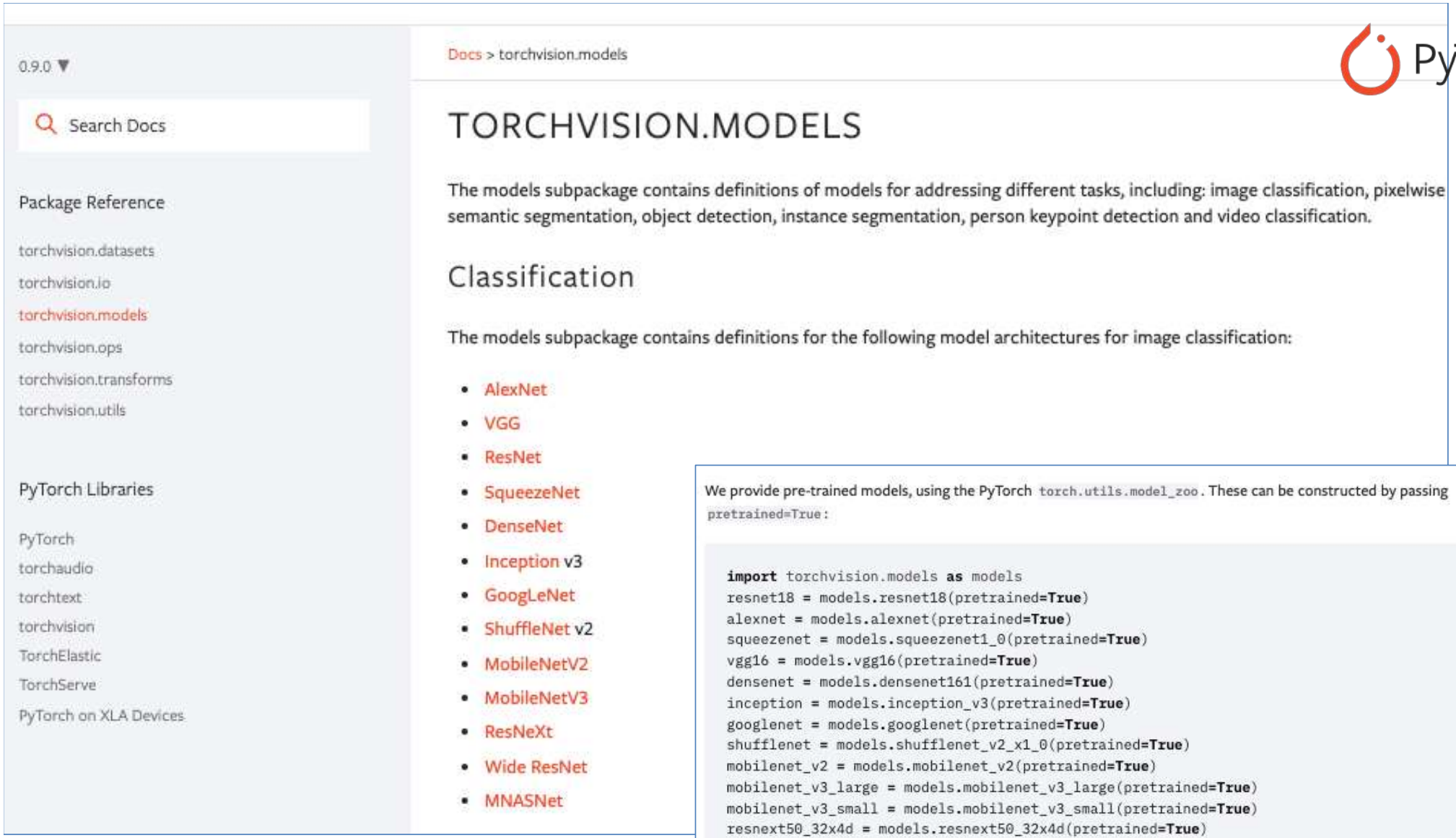
Model 1

- VGG19 (random initialized weights) + 2 Dense layers + Output layer

```
base_model = VGG19(weights=None, include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = True

x = base_model.output
x = Flatten()(x)
x = Dense(1024)(x)
x = Dropout(0.5)(x)
x = Dense(512)(x)
x = Dropout(0.5)(x)
output = Dense(num_class, activation='softmax')(x)

```

https://www.tensorflow.org/api_docs/python/tf/keras/applications

52



PyTorch

https://pytorch.org/vision/0.9/models.html

v1.1.0 ▼

pytorch-transformers

Star 131,473

Search docs

Pytorch-Transformers

PyTorch-Transformers is a library of state-of-the-art pre-trained models for Natural Language Processing (NLP).

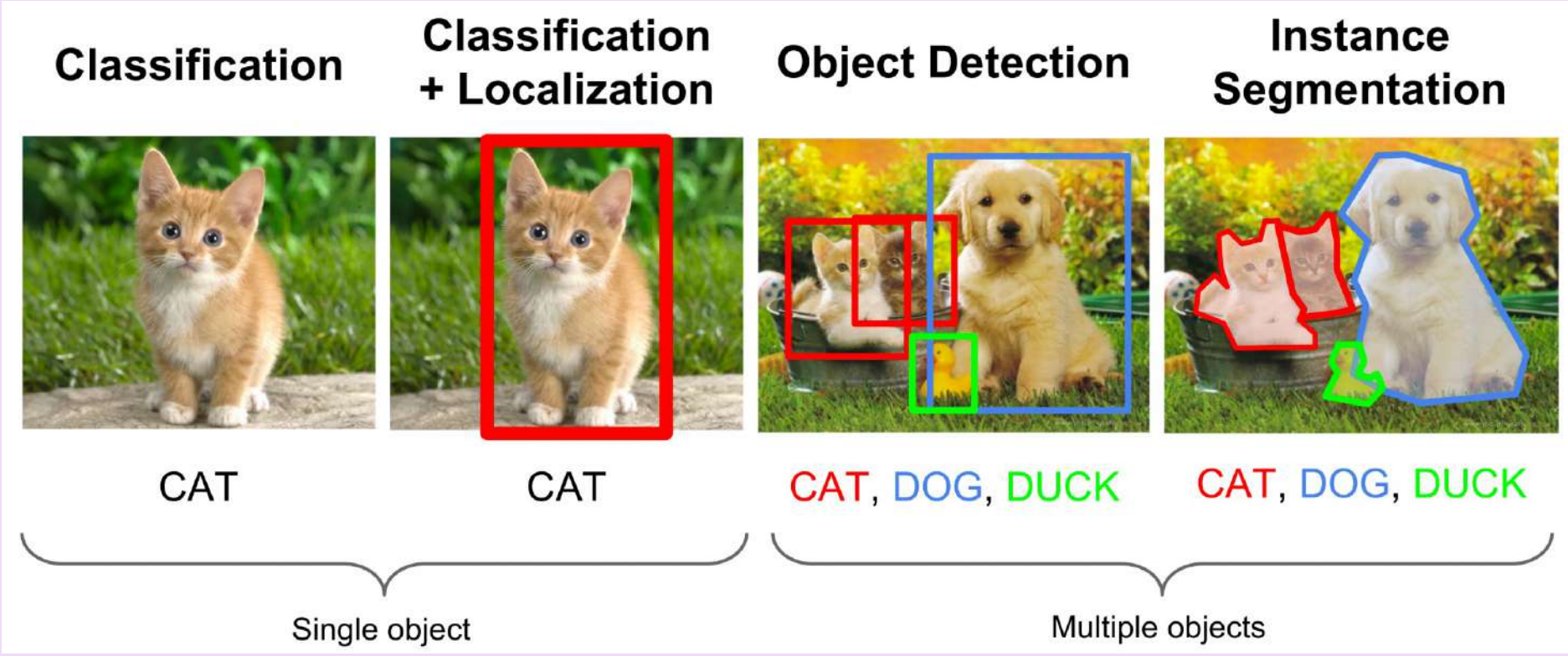
The library currently contains PyTorch implementations, pre-trained model weights, usage scripts and conversion utilities for the following models:

1. BERT (from Google) released with the paper BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin, Ming-Wei Lee and Kristina Toutanova.
2. GPT (from OpenAI) released with the paper Improving Language Understanding by Generative Pre-Training by Alec Radford, Karthik Narasimhan, Tim Salim Sutskever.
3. GPT-2 (from OpenAI) released with the paper Language Models are Unsupervised Multitask Learners by Alec Radford*, Jeffrey Wu*, Rewon Child, David Lua and Ilya Sutskever**.
4. Transformer-XL (from Google/CMU) released with the paper Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context by Zihang Dai*, Z Yang, Jaime Carbonell, Quoc V. Le, Ruslan Salakhutdinov.
5. XLNet (from Google/CMU) released with the paper XLNet: Generalized Autoregressive Pretraining for Language Understanding by Zhilin Yang*, Zihang Dai Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le.
6. XLM (from Facebook) released together with the paper Cross-lingual Language Model Pretraining by Guillaume Lample and Alexis Conneau.

Notes

- Installation

Image Classification Tasks

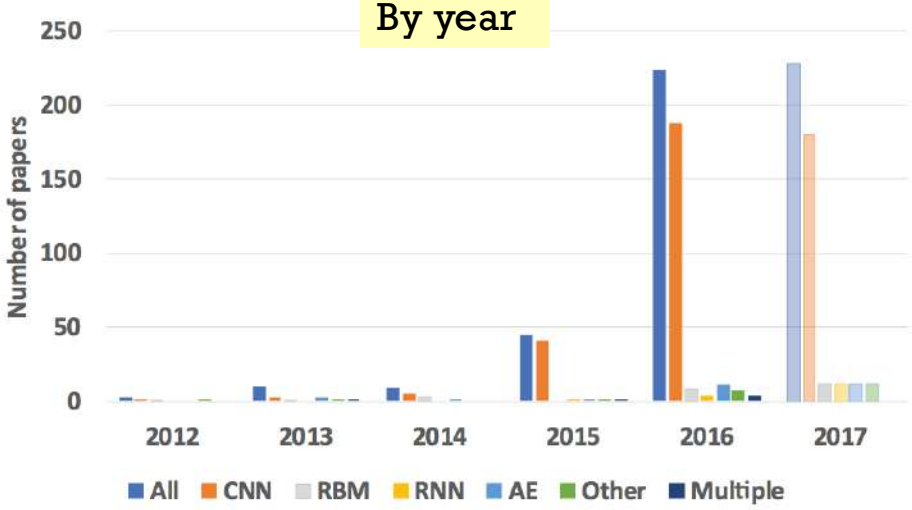


https://analyticsindiamag.com/top-5-image-classification-research-papers-every-data-scientist-should-know/

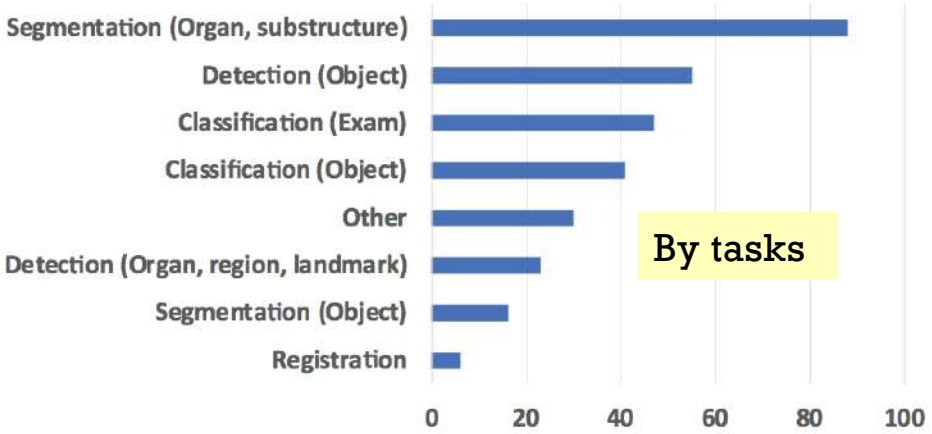
https://arxiv.org/pdf/1702.05747.pdf

Deep Learning in Medical Image Analysis [arXiv 2017]

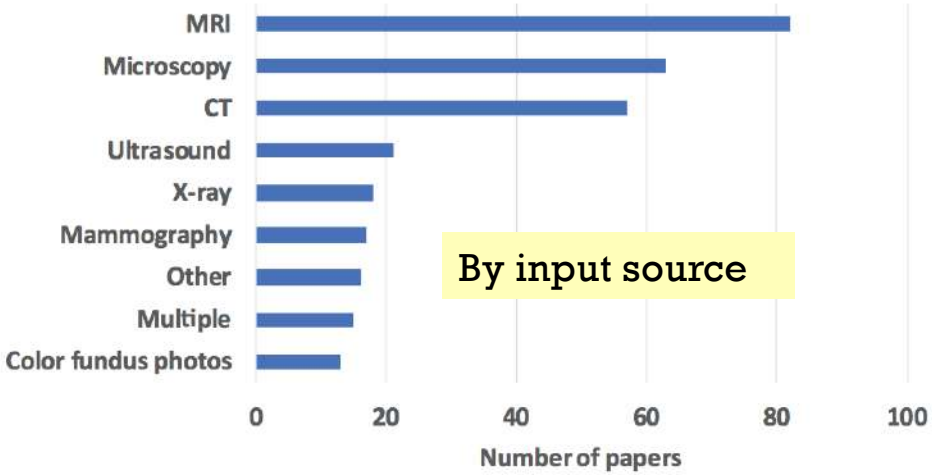
By year



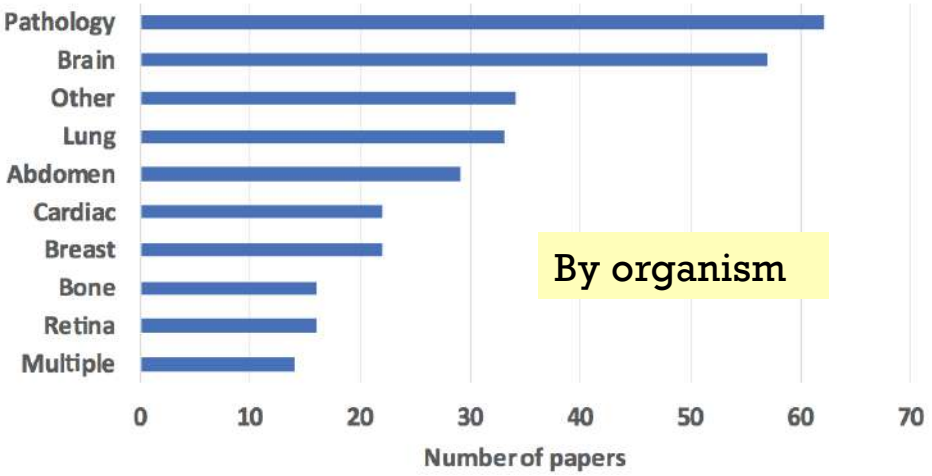
By tasks



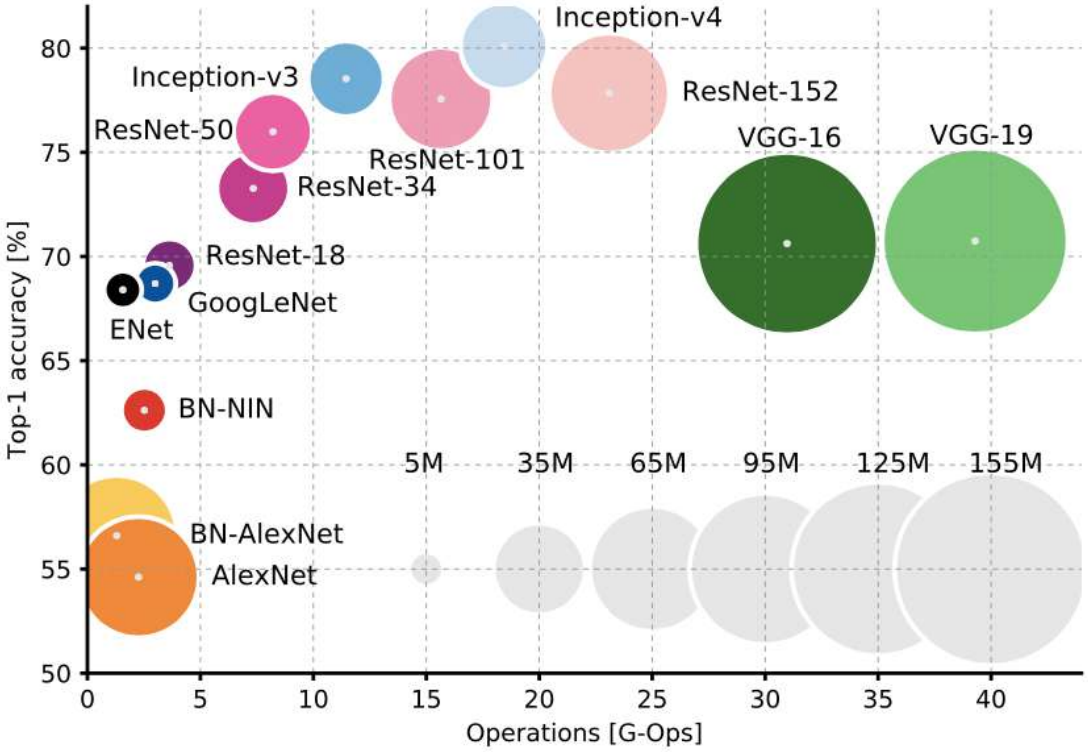
By input source



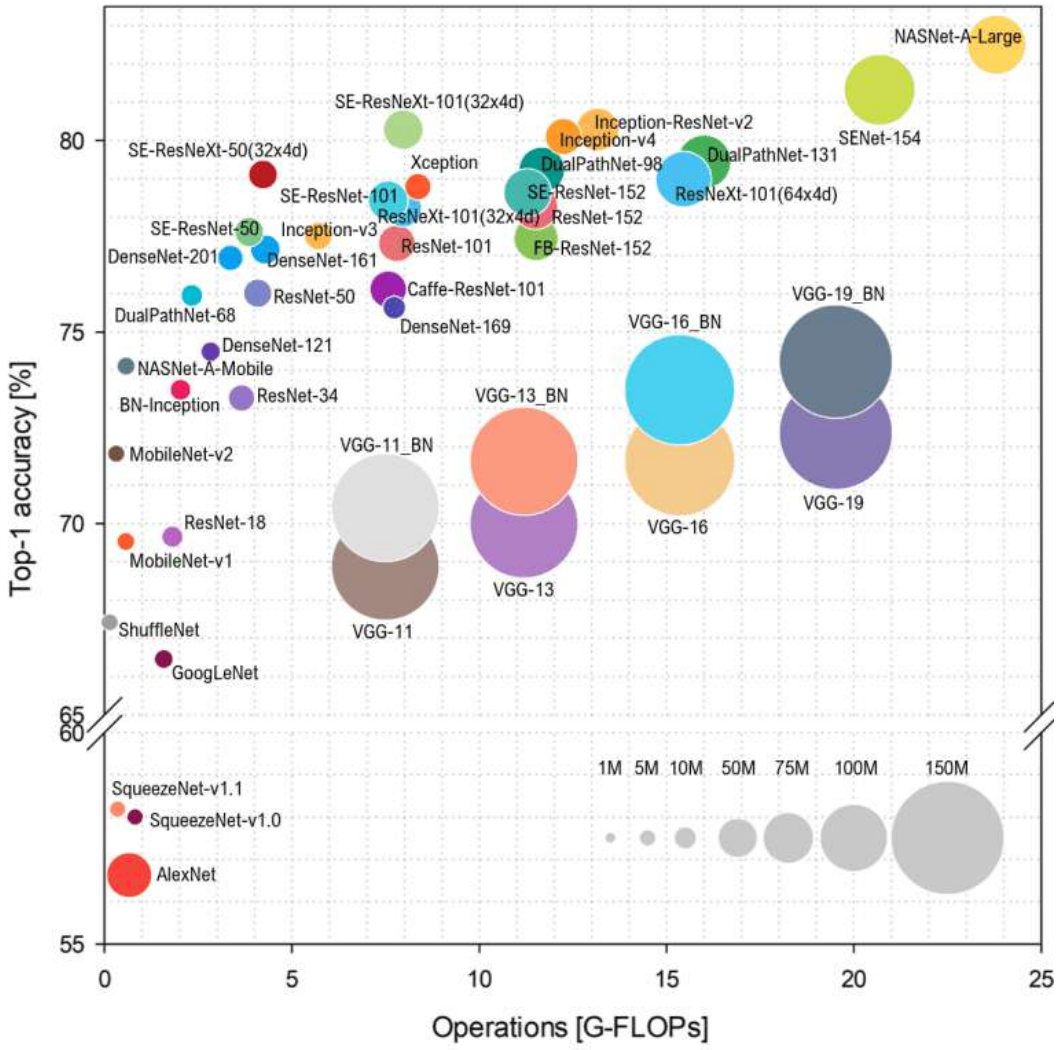
By organism



SOTA of Image Classification

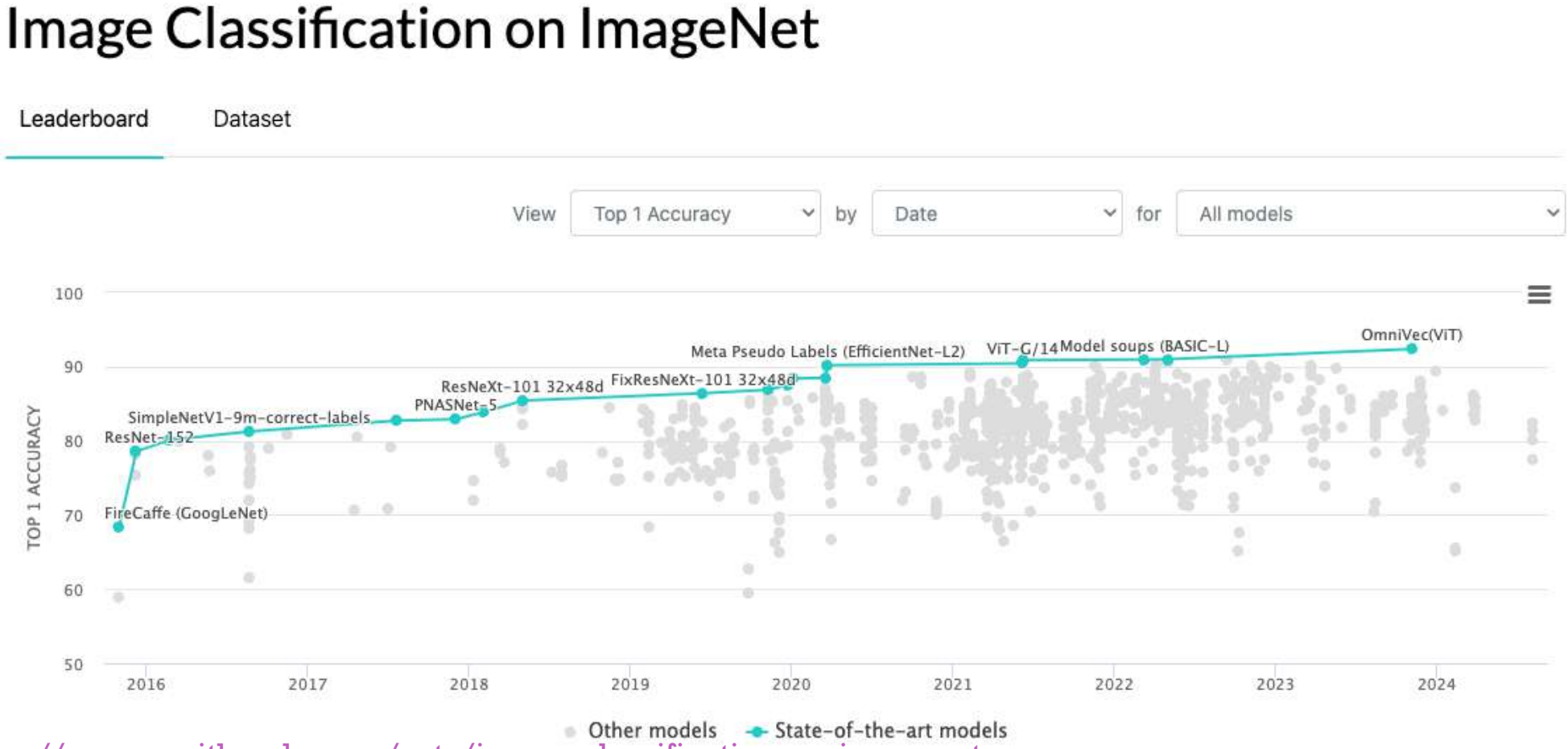


https://blog.csdn.net/qq_34216467/article/details/83061692



https://theaisummer.com/cnn-architectures/

Image classfication dashboard (2024)



https://paperswithcode.com/sota/image-classification-on-imagenet

EfficientNet (May, 2019) → EffNetV2 (2021)

Transformers: ViT (2020), ConvNeXt v2 (2023), OmiVec (2023)

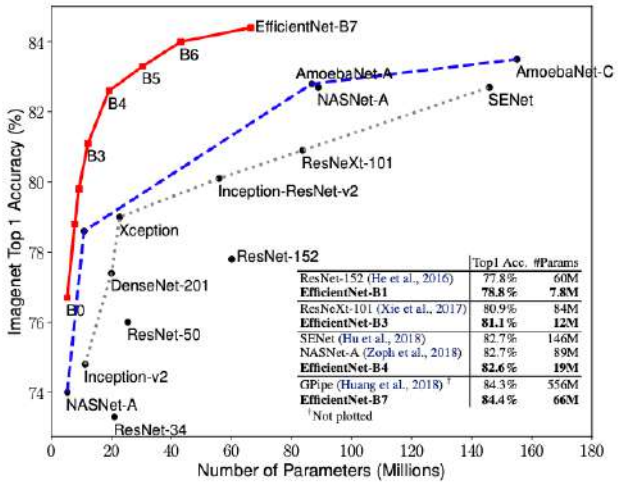
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan [1] Quoc V. Le [1]

Abstract

Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more resources are available. In this paper, we systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, we propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective *compound coefficient*. We demonstrate the effectiveness of this method on scaling up MobileNets and ResNet.

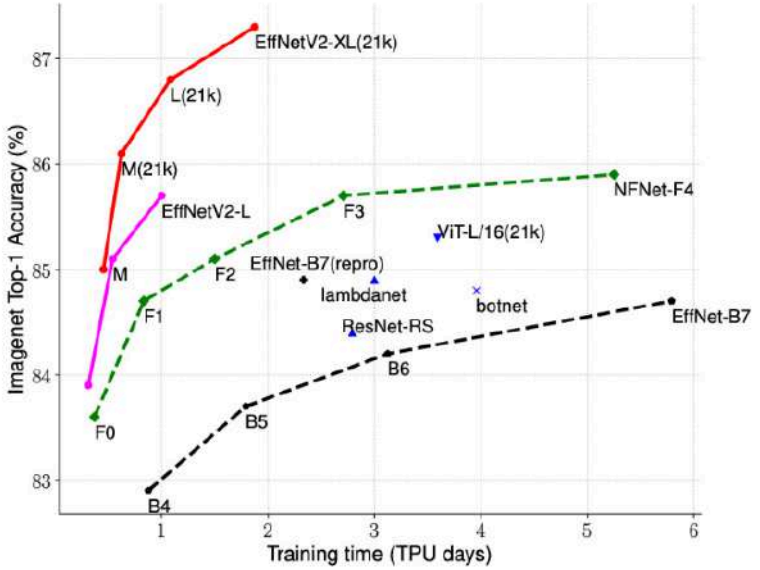
To go even further, we use neural architecture



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Figure 1. Model Size vs. ImageNet Accuracy. All numbers are



(a) Training efficiency.

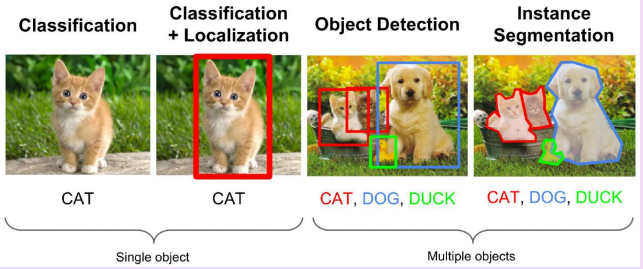
	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

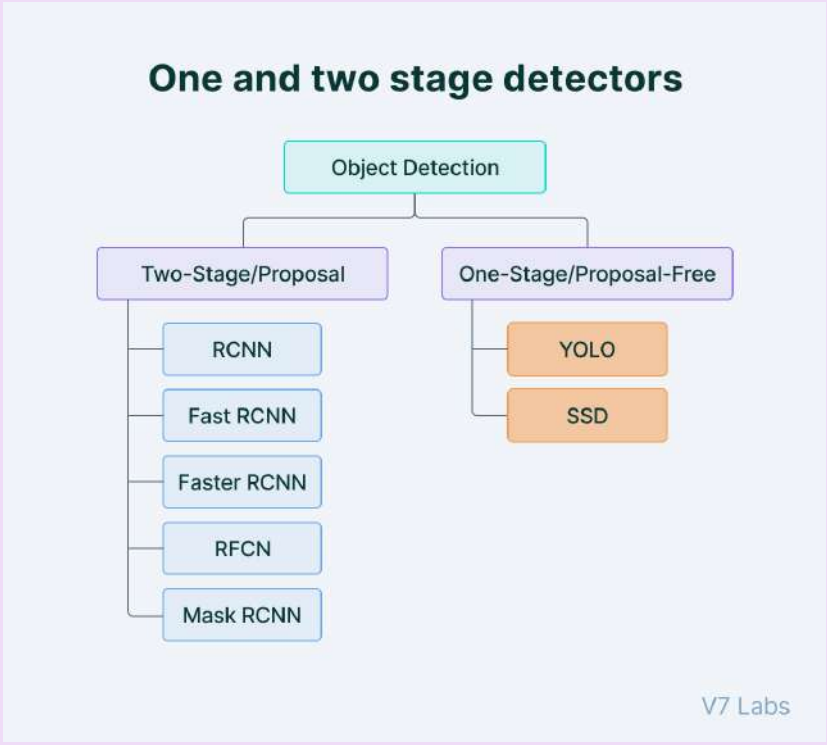
(b) Parameter efficiency.

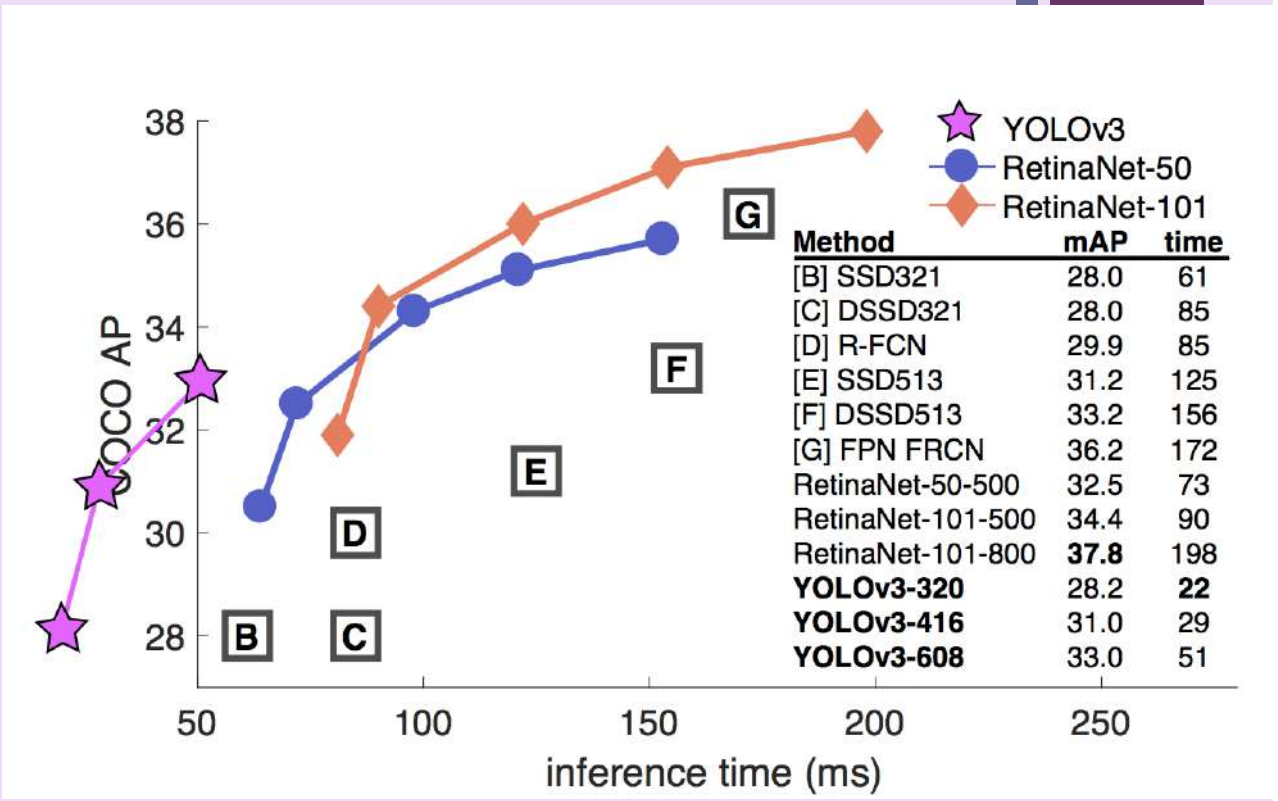
http://proceedings.mlr.press/v97/tan19a/tan19a.pdf

https://twitter.com/TensorFlow/status/1423359562724175873/photo/1

SOTA of Object Detection



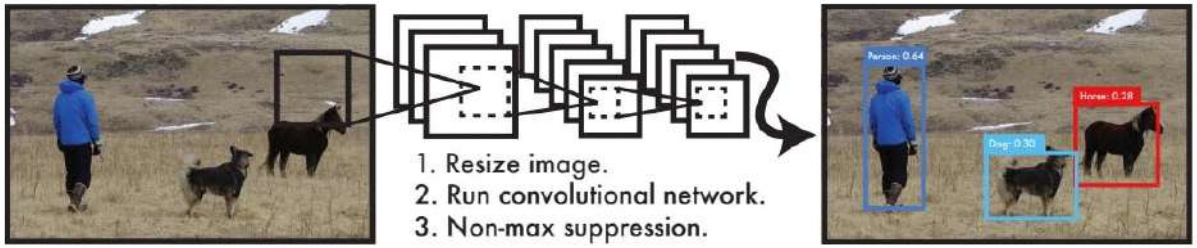




https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

YOLO Open Images in New York

YOLO's output



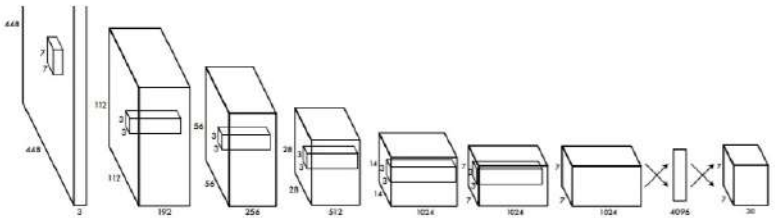
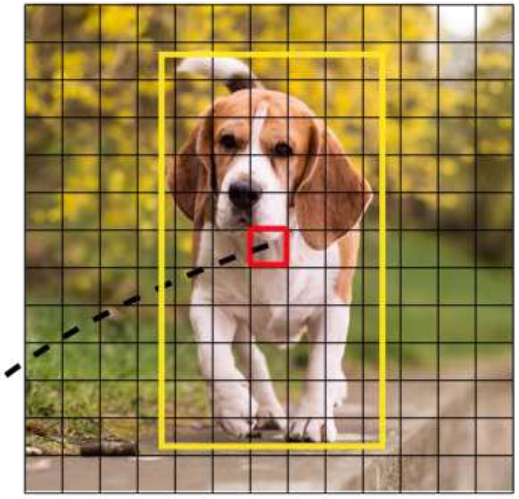
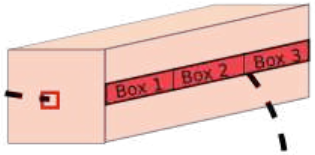


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

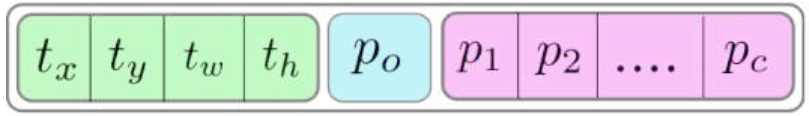
Image Grid. The Red Grid is responsible for detecting the dog



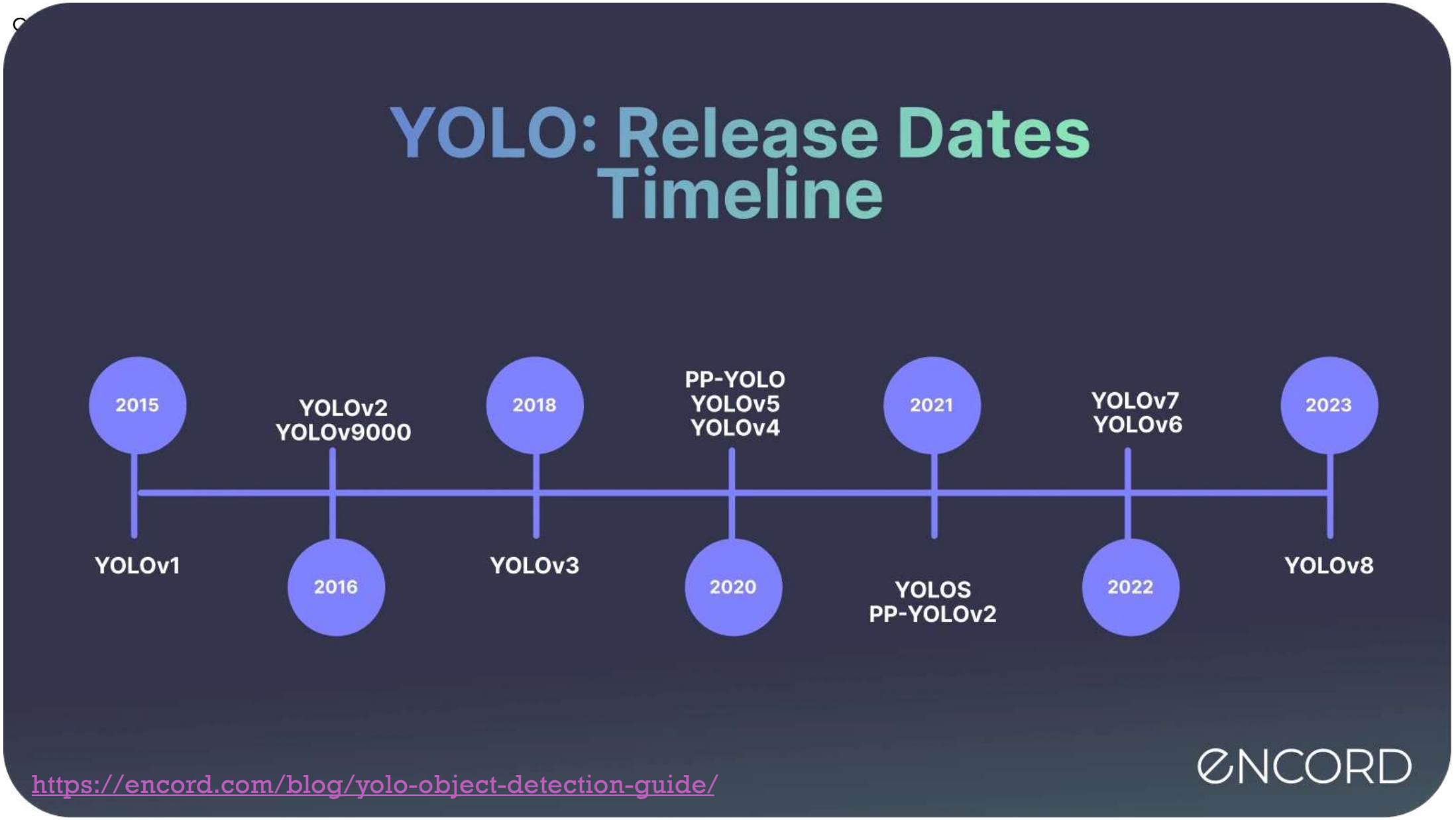
Prediction Feature Map



Attributes of a bounding box



https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/


YOLO: Release Dates Timeline
2015
YOLOv1
YOLOv2
YOLOv9000
2016
2018
YOLOv3
PP-YOLO
YOLOv5
YOLOv4
2020
2021
YOLOS
PP-YOLOv2
YOLOv7
YOLOv6
2022
2023
YOLOv8
https://encord.com/blog/yolo-object-detection-guide/
ENCORD

YoloV5 (2020) → YoloV8 (2023) → YoloV10 (2024)

English | 简体中文

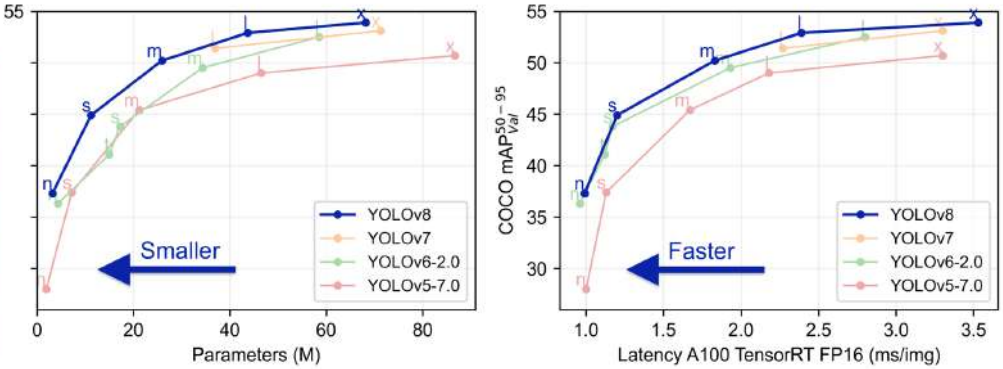
Ultralytics CI passing | codecov 87% | DOI 10.5281/zenodo.7347926 | docker pulls 22k

Run on Gradient | Open in Colab | Open in Kaggle

Ultralytics YOLOv8 is a cutting-edge, state-of-the-art (SOTA) model that builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility. YOLOv8 is designed to be fast, accurate, and easy to use, making it an excellent choice for a wide range of object detection and tracking, instance segmentation, image classification and pose estimation tasks.

We hope that the resources here will help you get the most out of YOLOv8. Please browse the YOLOv8 Docs for details, raise an issue on GitHub for support, and join our Discord community for questions and discussions!

To request an Enterprise License please complete the form at Ultralytics Licensing.

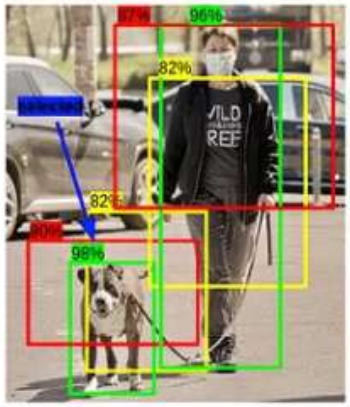


Models

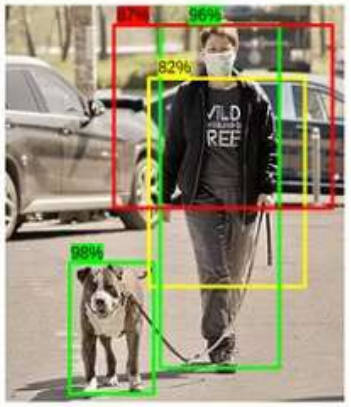
YOLOv8 Detect, Segment and Pose models pretrained on the COCO dataset are available here, as well as YOLOv8 Classify models pretrained on the ImageNet dataset. Track mode is available for all Detect, Segment and Pose models.

All Models download automatically from the latest Ultralytics release on first use.

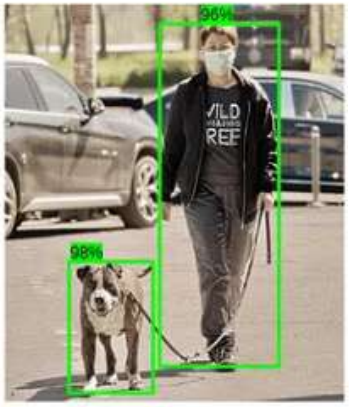
YOLOv10: Real-Time End-to-End Object Detection (2024); Remove Non-Maximum Suppression (NMS)



Step 1: Selecting Bounding box with highest score

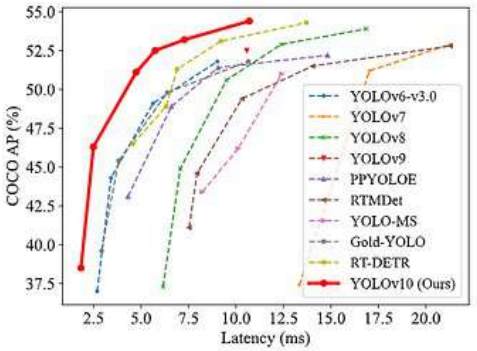


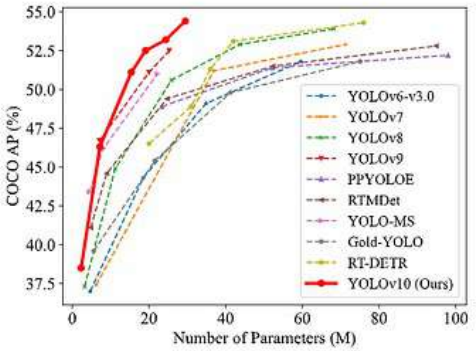
Step 3: Delete Bounding box with high overlap



Step 5: Final Output

Overlapping Bounding Boxes



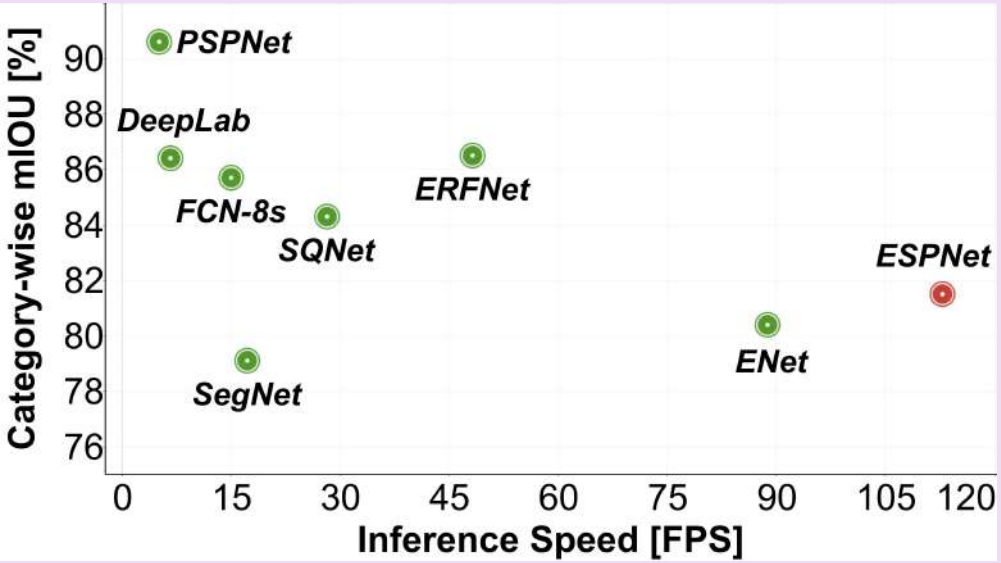


Consider the above image which has two objects: a person and a dog. YOLO might initially detect multiple bounding boxes for these objects, in above context — resulting in five bounding boxes where there should ideally be just two (one for each object).

```
IoU = Intersect Area / Union Area
```

- However, YOLO v10 eliminates the need for NMS by using consistent dual assignments during training. This innovation reduces computational overhead and latency while maintaining high accuracy, making YOLO v10 faster and more efficient for real-time applications
- 6 sizes: N, S, M, B, L, X

SOTA of Semantic Segmentation

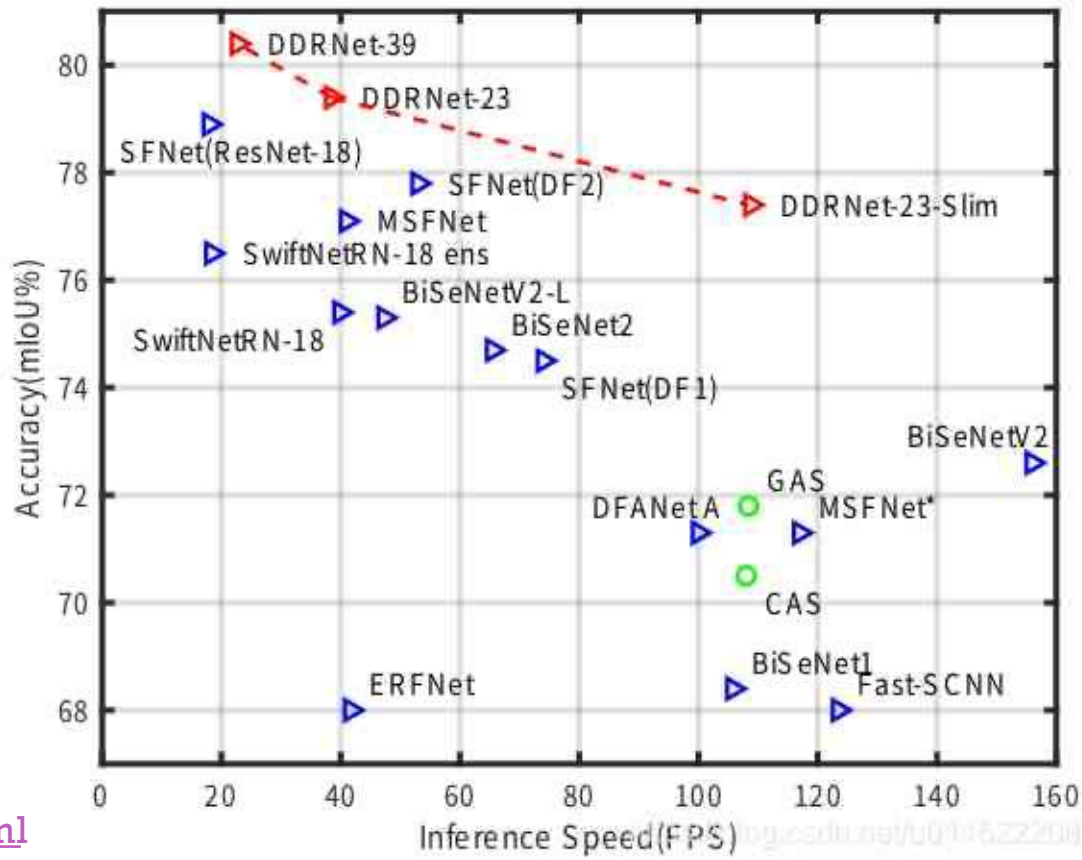


- **Real-time semantic segmentation**
- BiSeNet V1,V2
- DDRNet

https://chowdera.com/2021/11/20211109042359120o.html

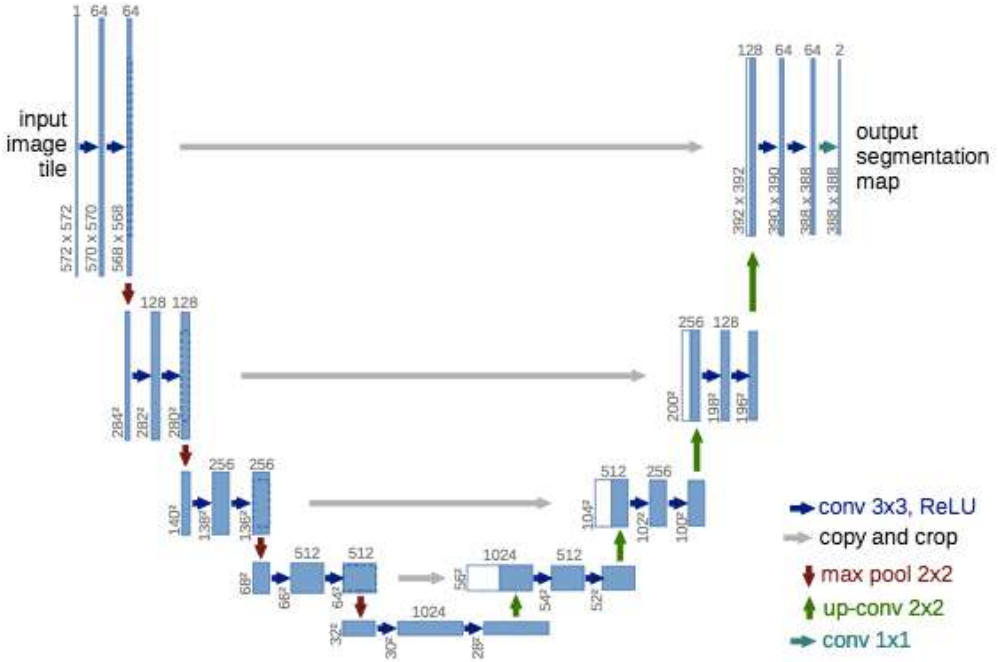
[Personal open source] - -
Real - time Semantic Segmentation ddrnet

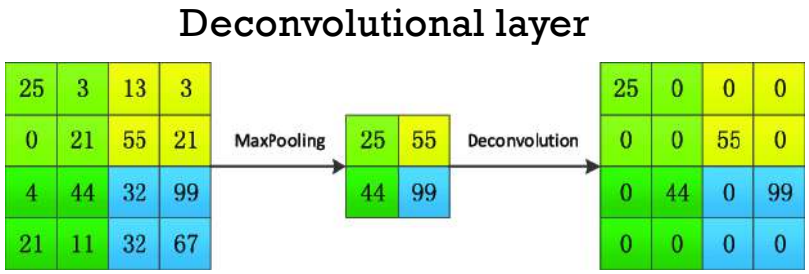
2021-11-09 04:24:01 【Nongfu Mountain Spring 2】



UNet: Encoder-Decoder Network

- **U-Net**





Architecture of the U-net for a given input image. The blue boxes correspond to feature maps blocks with their denoted shapes. The white boxes correspond to the copied and cropped feature maps.
Source: O. Ronneberger et al. (2015)

Semantic Segmentation

Easy to use and customizable SOTA Semantic Segmentation models with abundant datasets in PyTorch

Open in Colab

Supported Backbones:

- ResNet (CVPR 2016)
- ResNetD (ArXiv 2018)
- MobileNetV2 (CVPR 2018)
- MobileNetV3 (ICCV 2019)
- MiT (NeurIPS 2021)
- ResT (NeurIPS 2021)
- MicroNet (ICCV 2021)
- ResNet+ (ArXiv 2021)
- PVTv2 (CVMJ 2022)
- PoolFormer (CVPR 2022)
- ConvNeXt (CVPR 2022)
- UniFormer (ArXiv 2022)
- VAN (ArXiv 2022)
- DaViT (ArXiv 2022)

Supported Heads/Methods:

- FCN (CVPR 2015)
- UPerNet (ECCV 2018)
- BiSeNetv1 (ECCV 2018)
- FPN (CVPR 2019)
- SFNet (ECCV 2020)
- SegFormer (NeurIPS 2021)
- FaPN (ICCV 2021)
- CondNet (IEEE SPL 2021)
- Light-Ham (ICLR 2021)
- Lawin (ArXiv 2022)
- TopFormer (CVPR 2022)

Supported Standalone Models:

- BiSeNetv2 (IJCV 2021)
- DDRNet (ArXiv 2021)

Supported Modules:

- PPM (CVPR 2017)
- PSA (ArXiv 2021)

Refer to MODELS for benchmarks and available pre-trained models.

https://github.com/sithu31296/semantic-segmentation

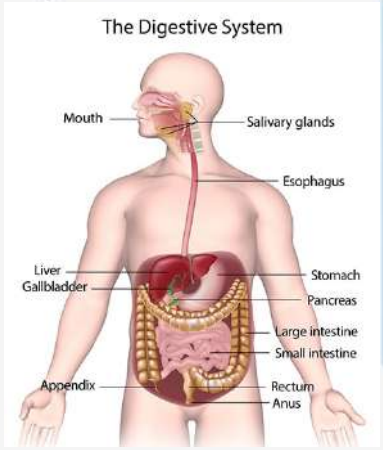
Introducing SAM 2: The next generation of Meta Segment Anything Model for videos and images

July 29, 2024 · 15 minute read

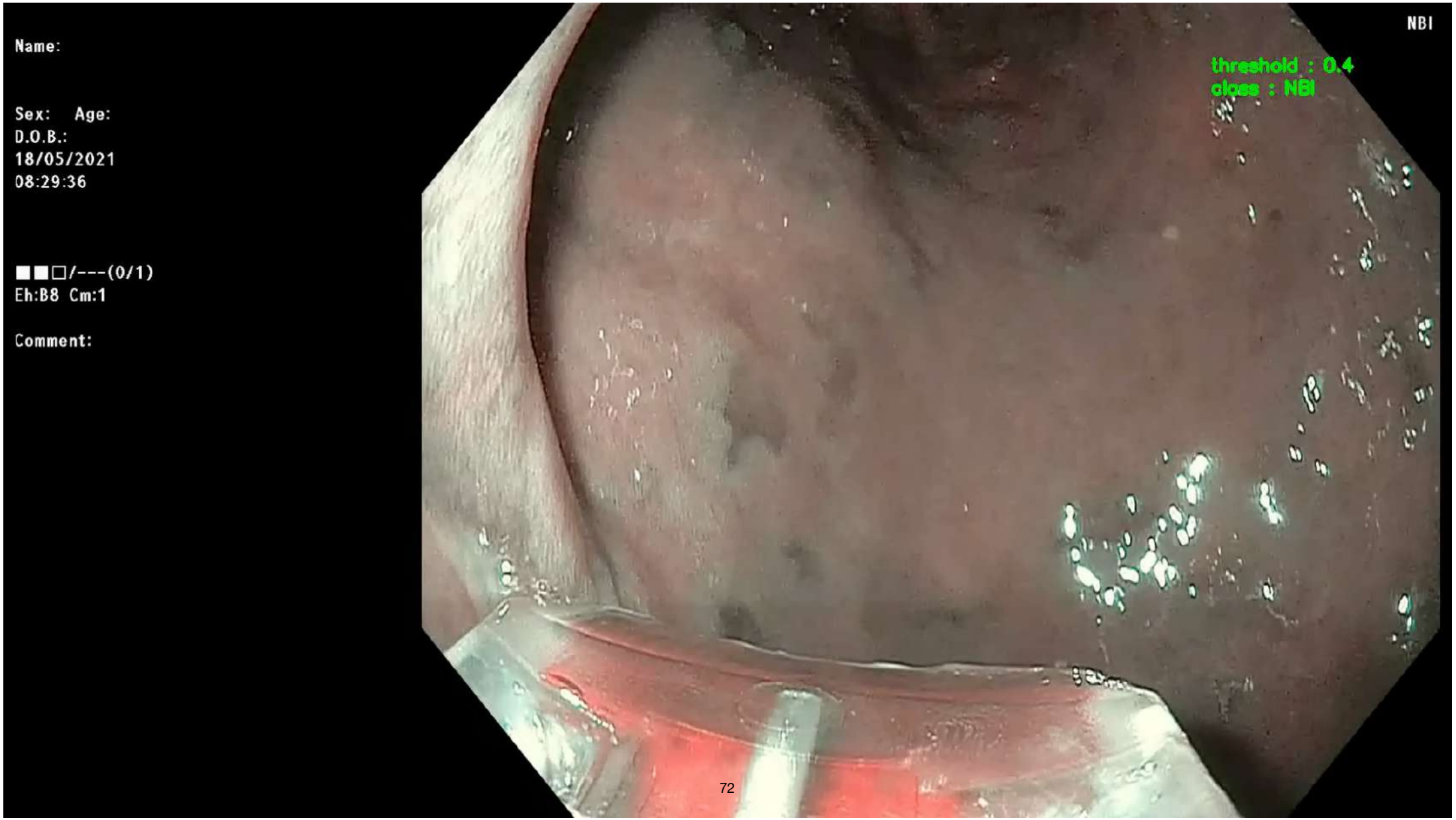
https://ai.meta.com/blog/segment-anything-2/

Real-Time Colonic Polyp Detection

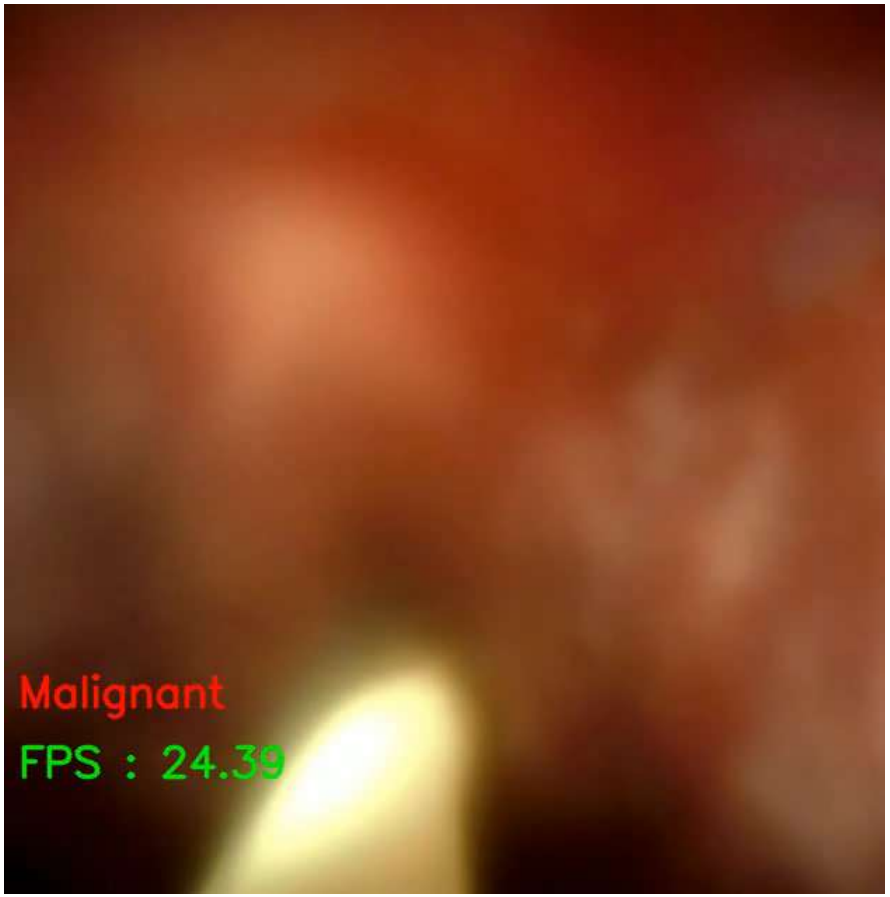
An AI-assisted solution that aims to improve colonic polyp detection in real-time. It is compatible with all standard colonoscope systems.

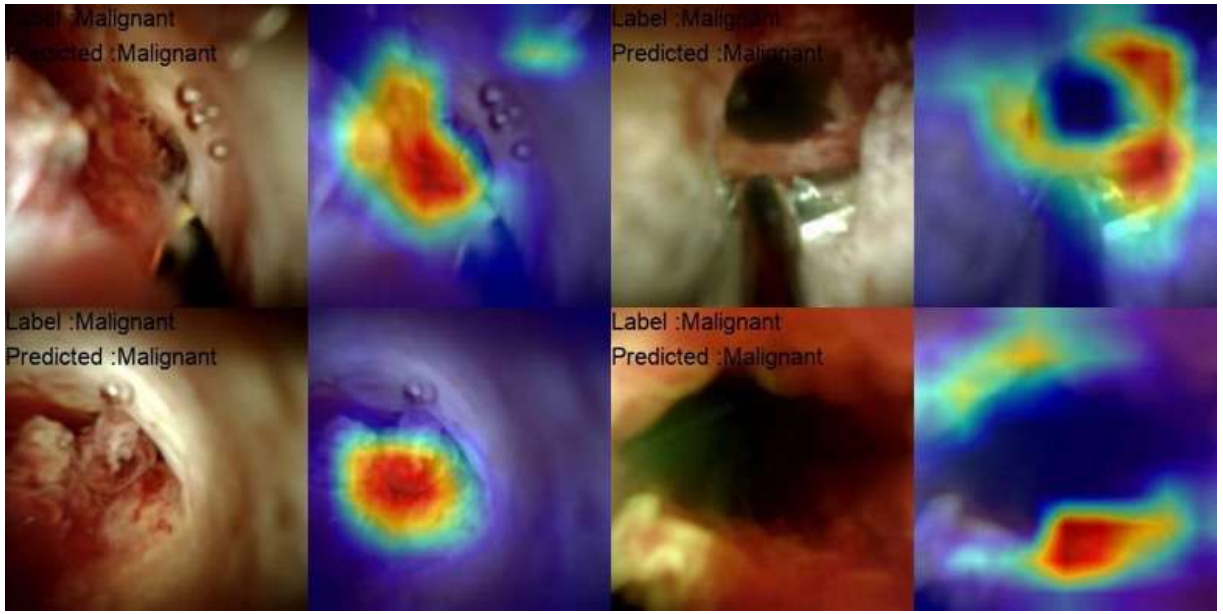



09/11/2021
13:53:24
COLON
*1/200
Lv+2 AUTO
HT NR
SE
3.8
12.0
12.0
S1: F+T
S2: LM
S3: CAD
S4:
EC-760R-V/L
3C729K035
BL-7000
CHULALONGKORN HOS


Name:
Sex: Age:
D.O.B.:
18/05/2021
08:29:36
■■□/---(0/1)
Eh:B8 Cm:1
Comment:
NBI
threshold : 0.4
class : NBI

Cholangioscopy





+

RNN

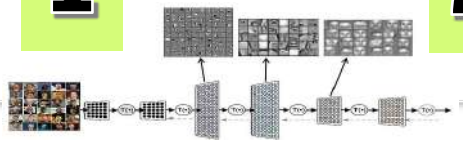
Deep Learning Application

Speech Recognition

Computer Vision

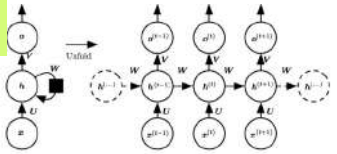
Natural Language Processing

1



CNN

2



RNN (LSTM)

Top 10 Strategic Technology Trends for 2020

People-centric

- Hyperautomation
- Multiexperience
- Democratization
- Human Augmentation
- Transparency and Traceability

Smart spaces

- Empowered Edge
- Distributed Cloud
- Autonomous Things
- Practical Blockchain
- AI Security

gartner.com/SmarterWithGartner

Gartner

Gartner Top Strategic Technology Trends for 2021

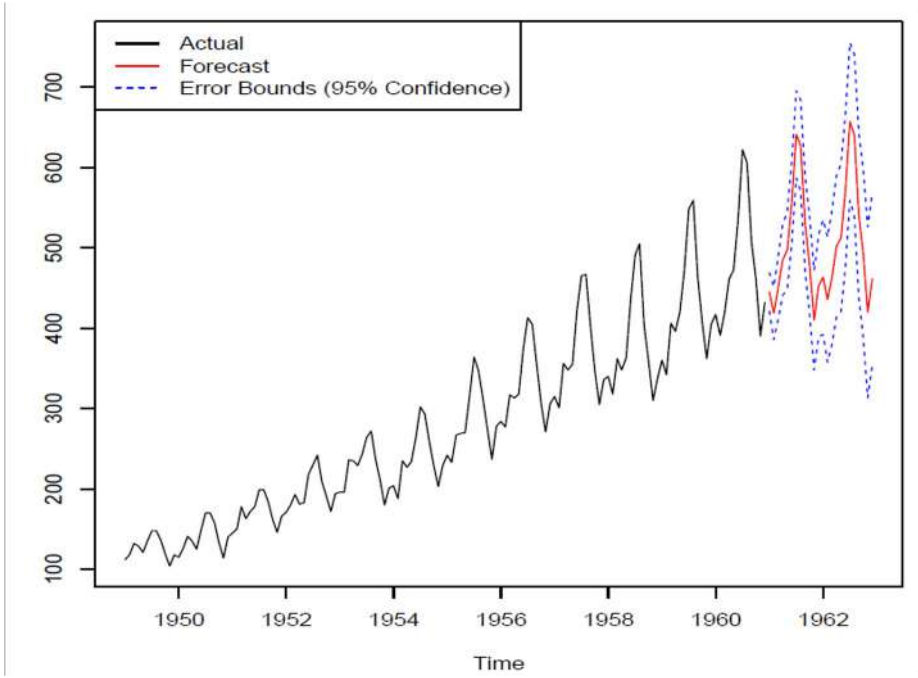
People centricity	Location independence	Resilient delivery
Internet of Behaviors	Distributed cloud	Intelligent composable business
Total experience strategy	Anywhere operations	AI engineering
Privacy-enhancing computing	Cybersecurity mesh	Hyperautomation

Combinatorial innovation

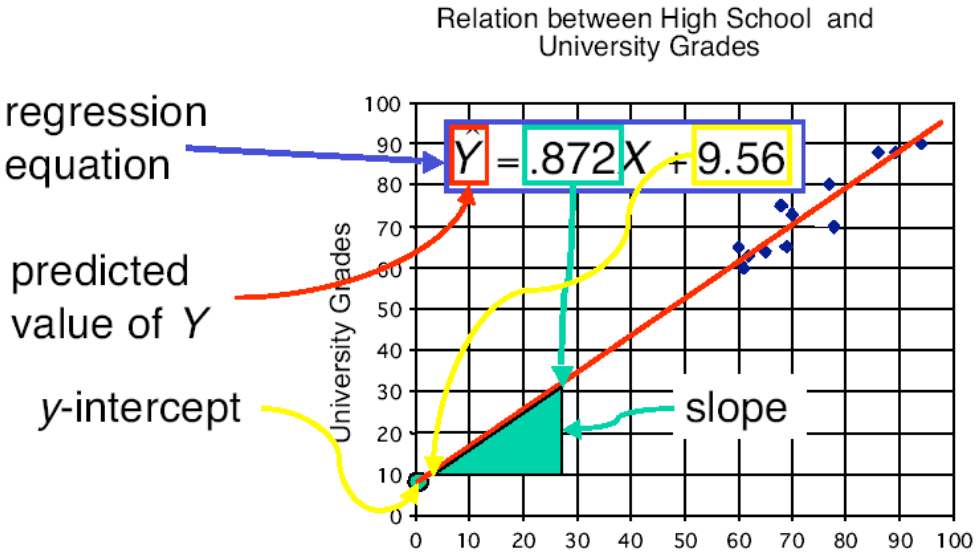
gartner.com/SmarterWithGartner

Source: Gartner

Gartner



Regression – Linear Relationship



weight, coefficient

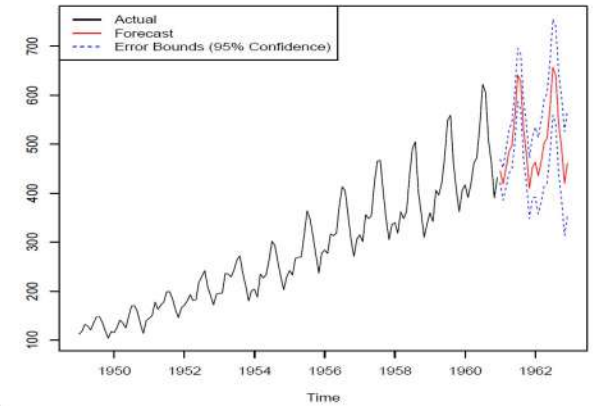
$$\underset{\text{target}}{\hat{y}} = \underset{\text{intercept}}{\hat{w}_0} + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

input

- The least square method aims to minimize the following term

$$\sum_{\substack{training \\ data}} (y_i - \hat{y}_i)^2$$

Autoregressive Model – Linear Relationship



- Based on linear regression, but using previous timestep data

$$X_t = c + \Sigma_{i=1}^{p} \varphi_i X_{t-i}$$

- Where X_t is data at timestep t, c is constant, and $\varphi_1, \varphi_2, \varphi_3, \cdots, \varphi_p$ are parameters for each timestep data

Example: When p = 2 (looking back two steps), the equation will be

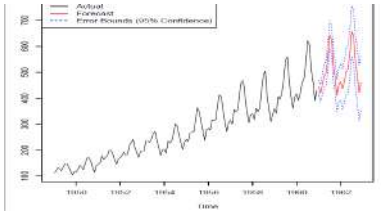
$$X_t = c + \varphi_1 X_{t-1} + \varphi_2 X_{t-2}$$

- Parameters are able to calculate using various method, such as ordinary least square procedure or Yule–Walker equations

$$x(t+1) = \hat{w}_0 + \hat{w}_1 x(t) + \hat{w}_2 x(t-1)$$

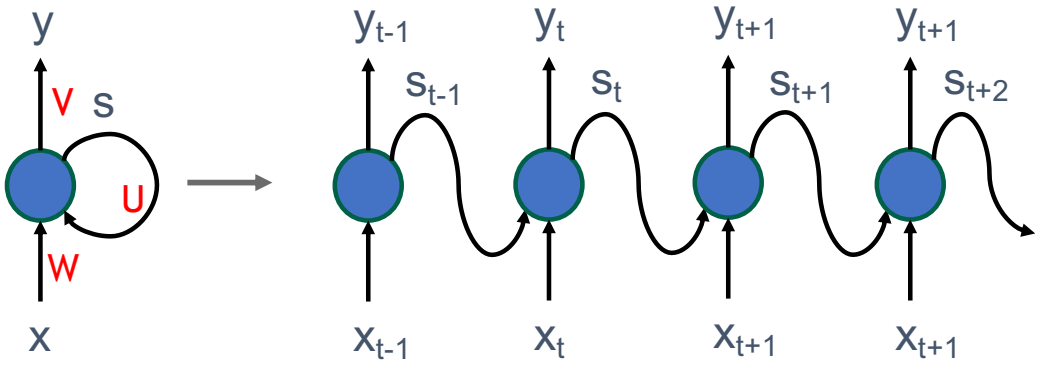
RECURRENT NEURONS (RNN)

Non-Linear Relationship



Forward Pass

x(t) x(t+1) x(t+2) x(t+3)



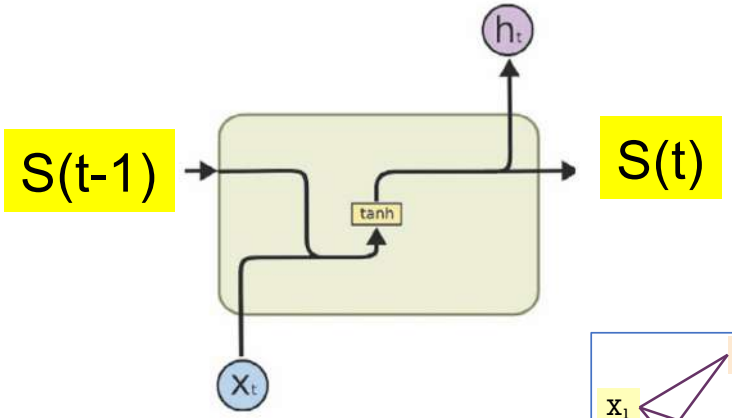
Unrolling in Time

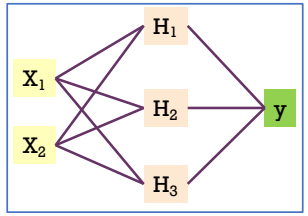
$$s_t = \tanh(Ux_t + Ws_{t-1})$$

x(t+1) $$\hat{y}_t = softmax(Vs_t)$$

x(t+1)

y(t)



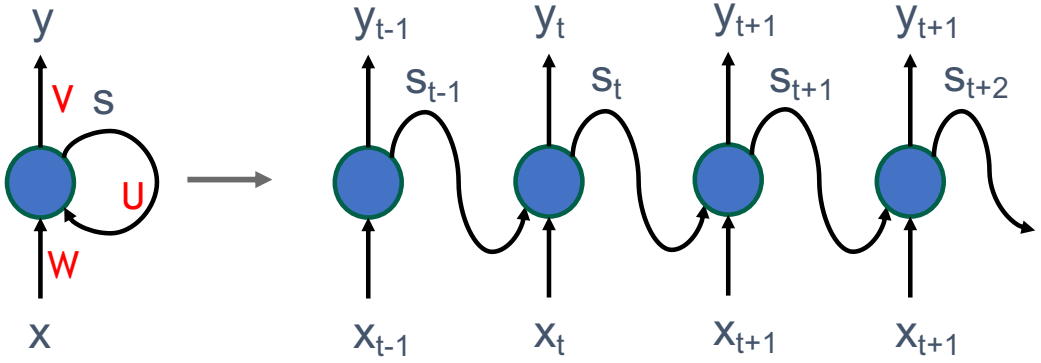


Reference: https://www.cse.iitk.ac.in/users/sigml/lec/Slides/LSTM.pdf

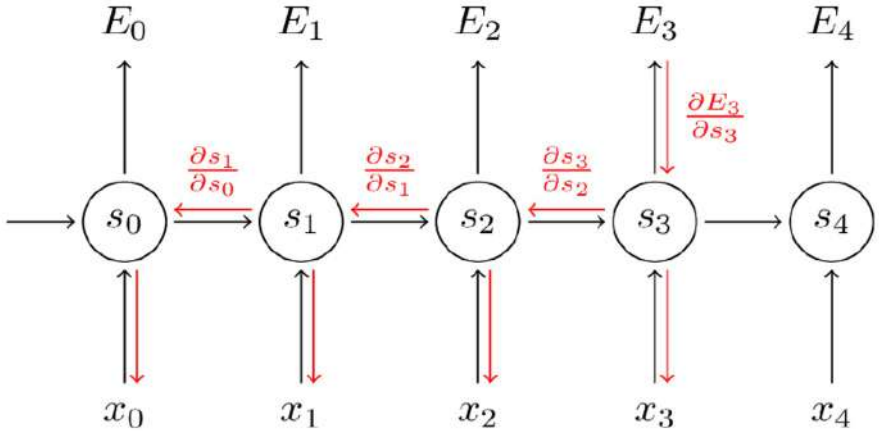
RECURRENT NEURONS (RNN)

Ggradient vanishing
= multiplication

Forward Pass



Backward Pass

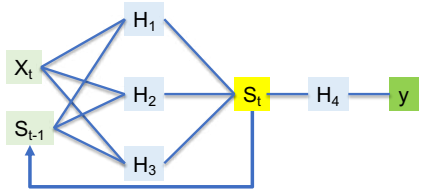


Unrolling in Time

Backpropagation-Through-Time (BPTT)

$$s_t = \tanh\left(Ux_t + Ws_{t-1}\right)$$

$$\hat{y}_t = softmax\left(Vs_t\right)$$

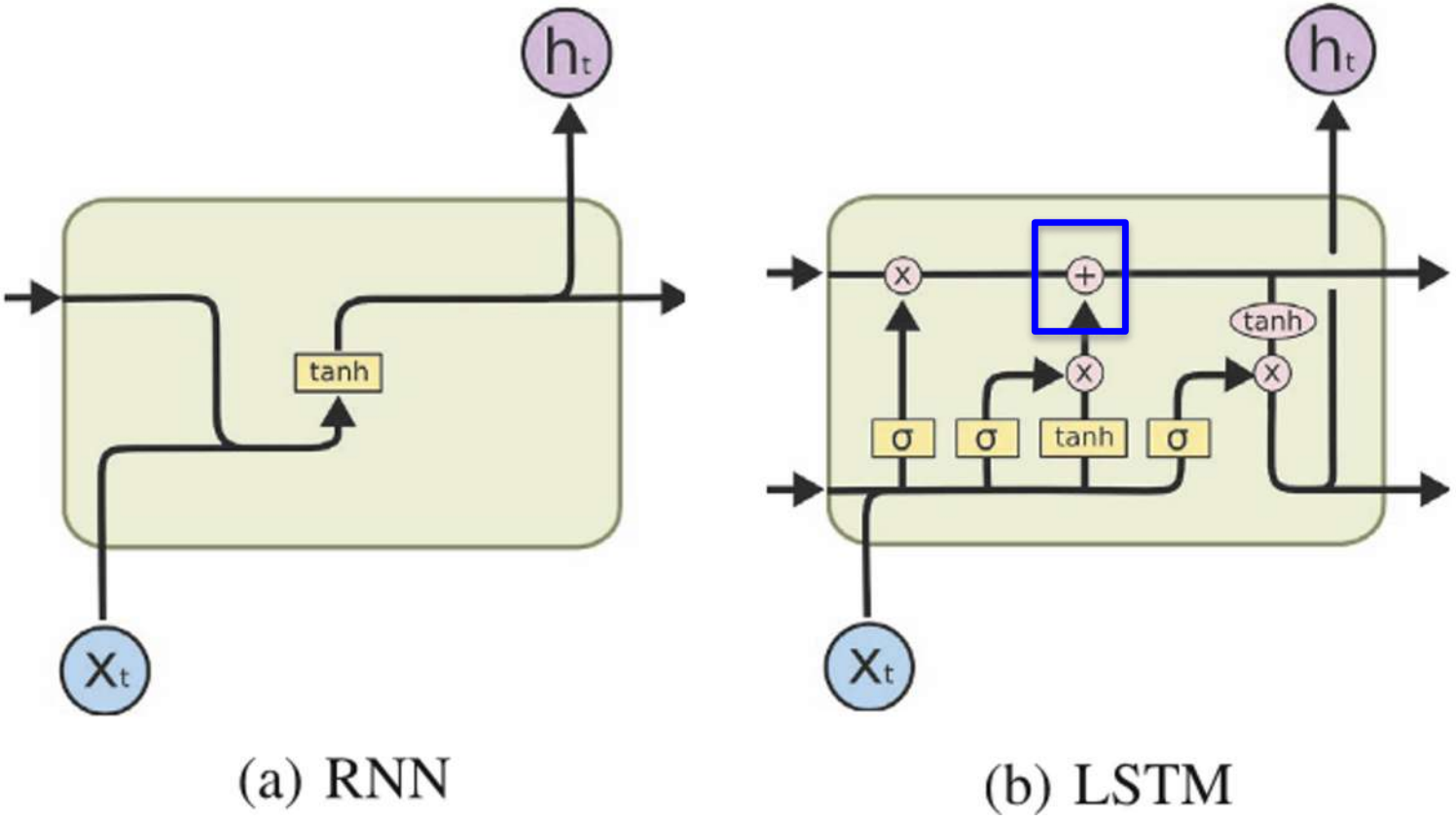


$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

Reference: https://www.cse.iitk.ac.in/users/sigml/lec/Slides/LSTM.pdf

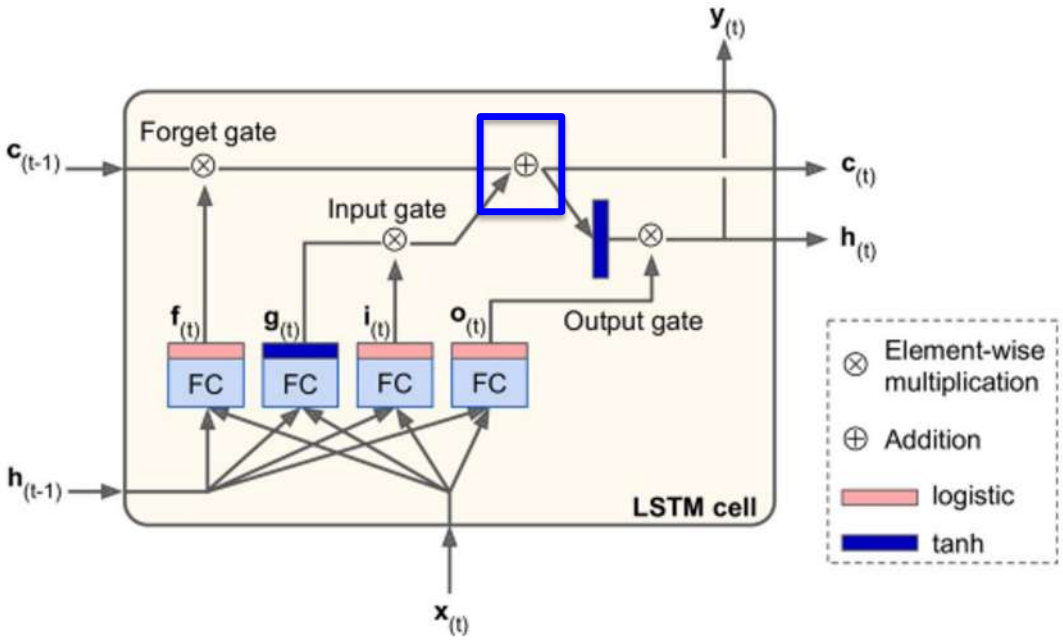
RNN VS LSTM



(a) RNN (b) LSTM

Reference: https://cs.uwaterloo.ca/~mli/Deep-Learning-2017-Lecture6RNN.ppt

LONG SHORT TERM (LSTM) CELL



There are 3 gates with 2 outputs.

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$

$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

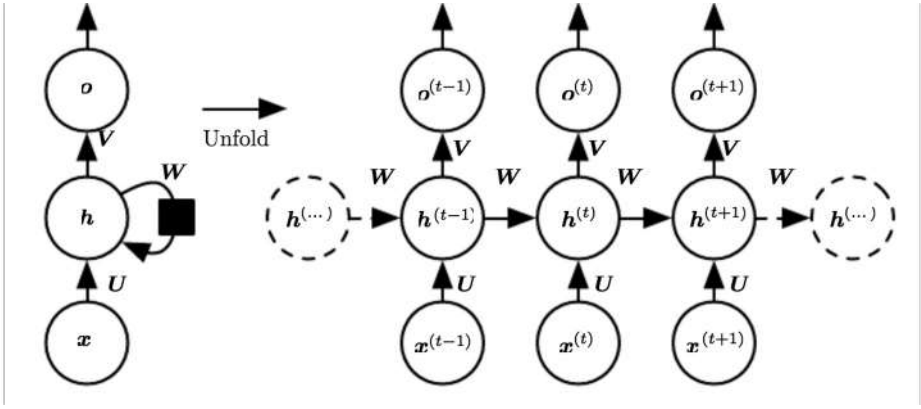
$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

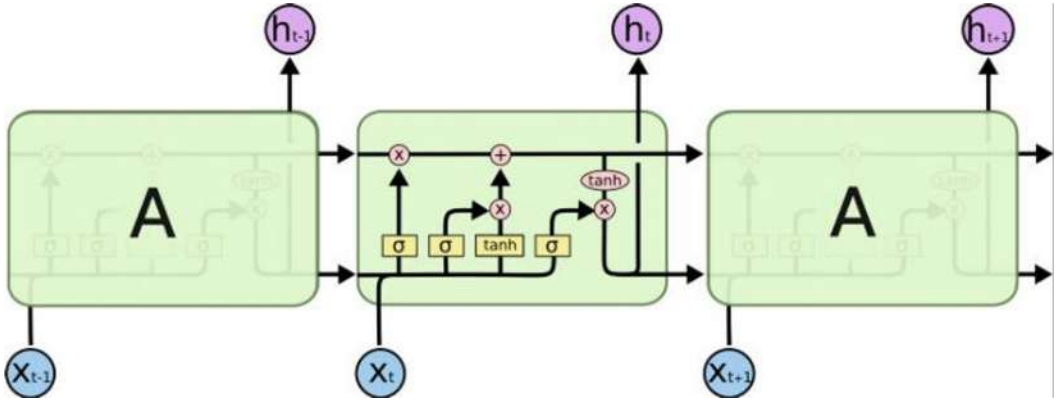
$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$

$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

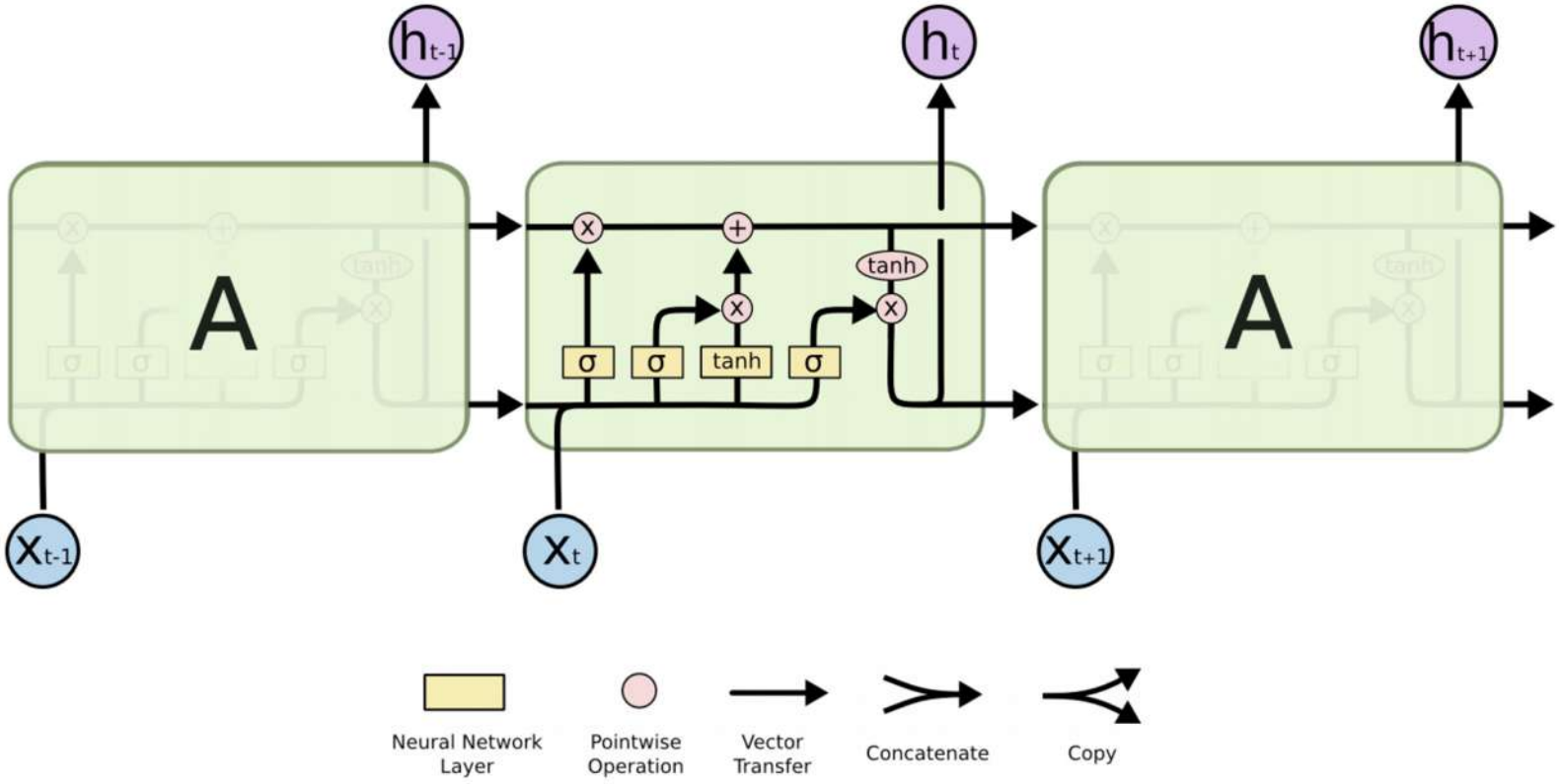
Deep Learning Approach: LSTM



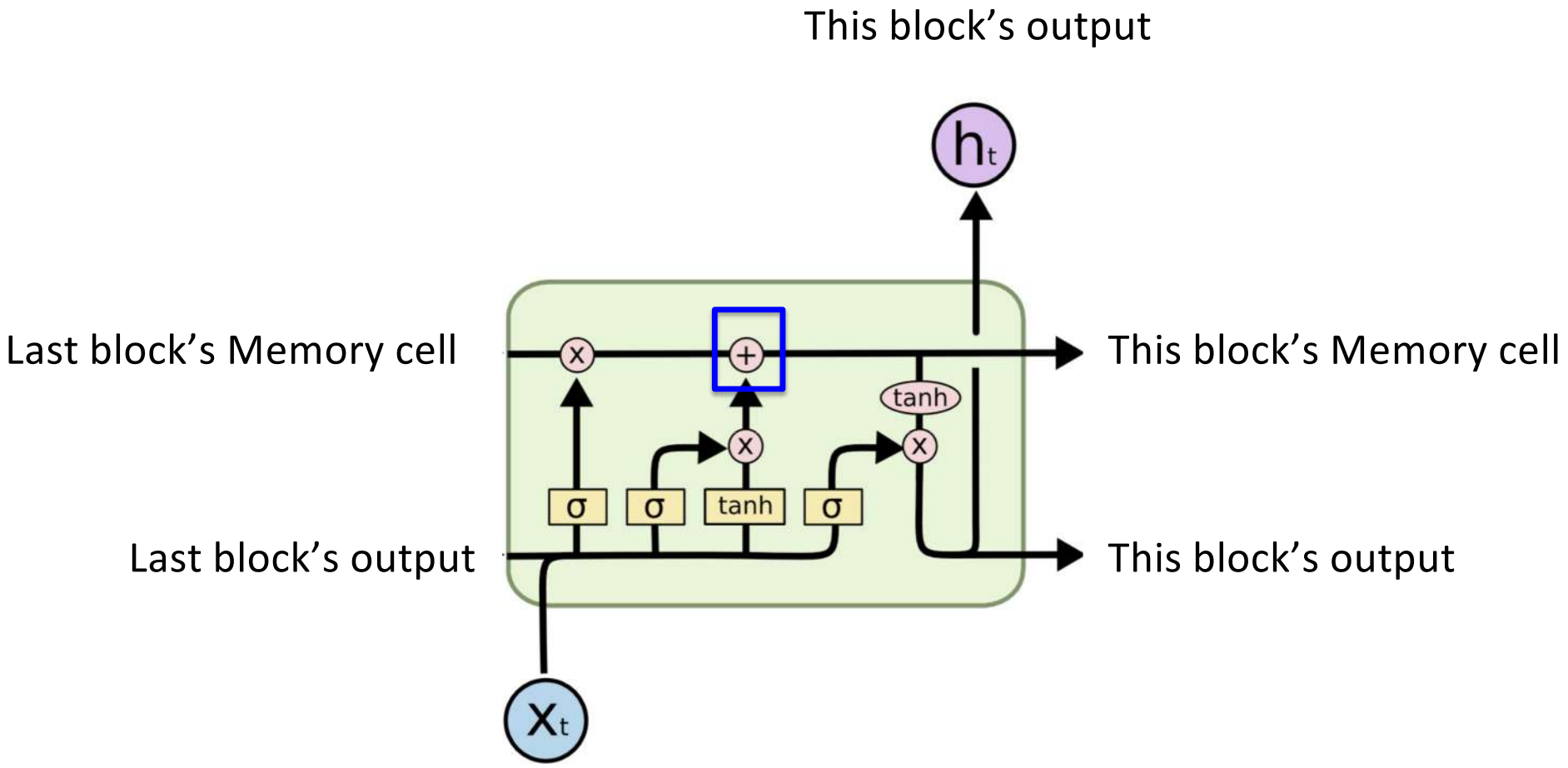


Reference: Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. MIT Press.
http://www.deeplearningbook.org

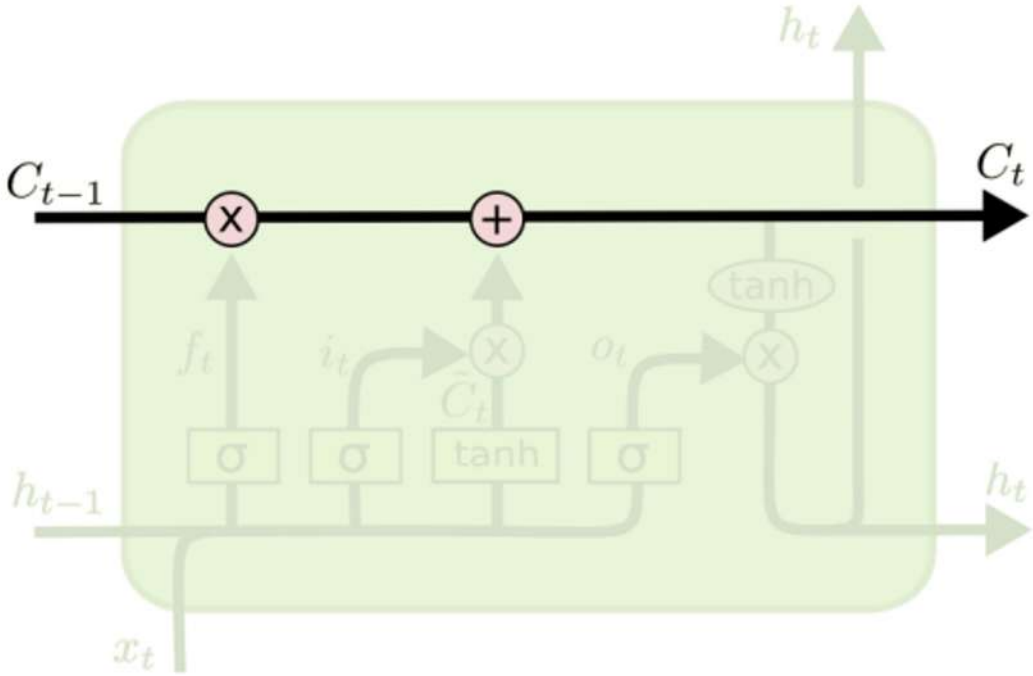
Inside LSTM

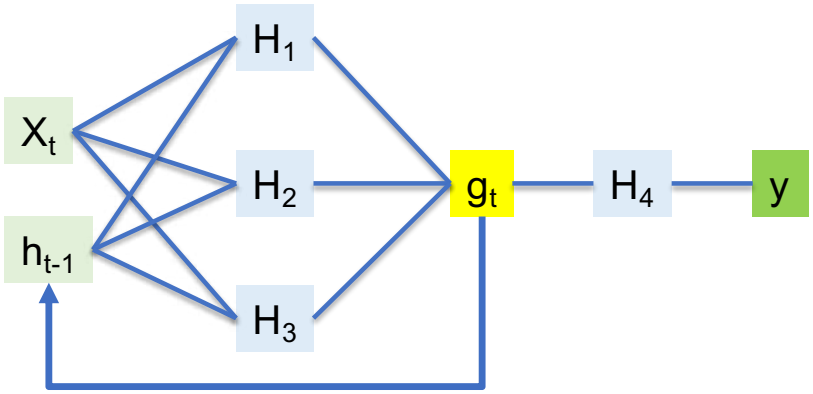

h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

Inside LSTM


This block's output
Last block's Memory cell
This block's Memory cell
Last block's output
This block's output
tanh
σ
σ
tanh
σ

Inside LSTM: (1) Cell Memory





There are 3 gates with 2 outputs.

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$

$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

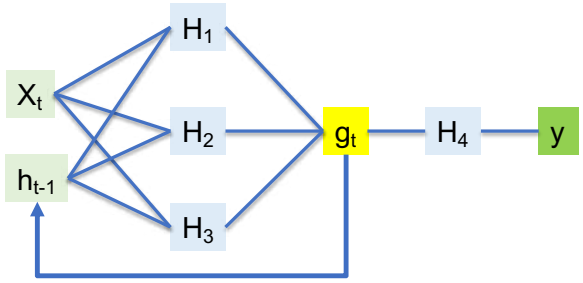
$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$

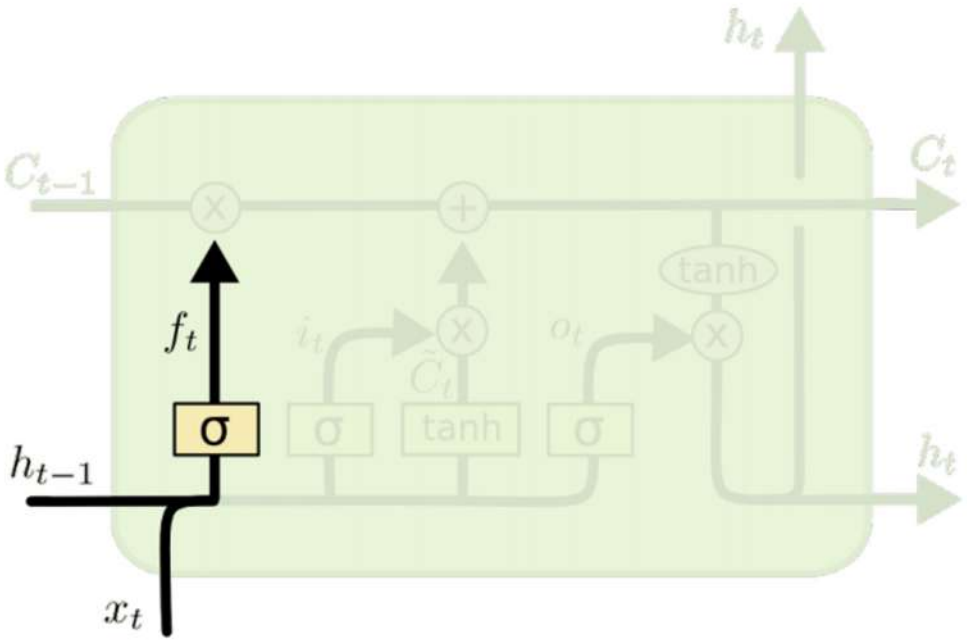
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

Inside LSTM: (2) Forget Gate



Should we continue to remember this “bit” of information or not?



There are 3 gates with 2 outputs.

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$

$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

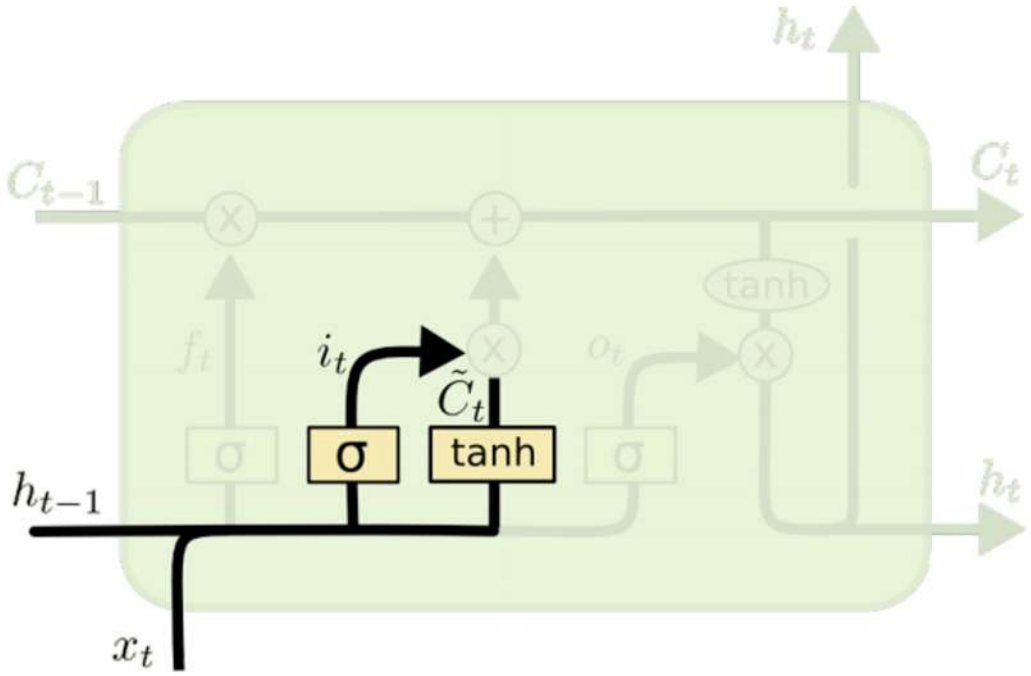
$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$

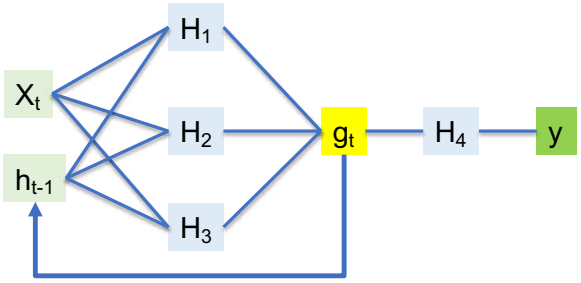
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

Inside LSTM: (3) Input Gate

Should we update this “bit” of information or not?
If yes, then what should we remember?





There are 3 gates with 2 outputs.

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$

$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

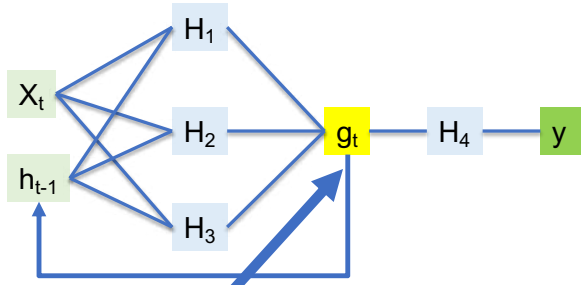
$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$

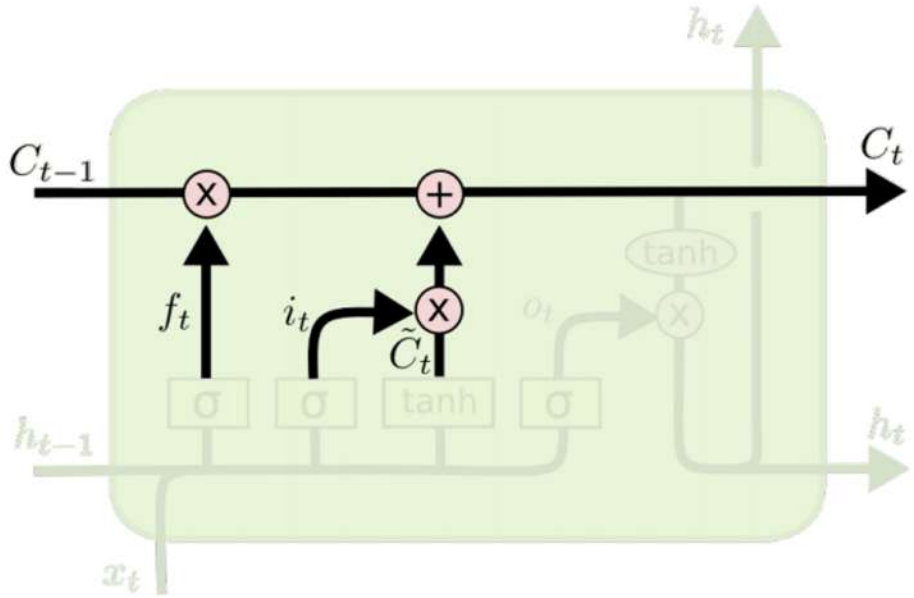
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

Inside LSTM: (4) Memory Update



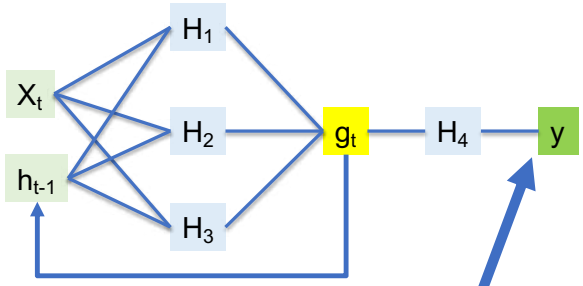
Forget what needs to be forgotten + memorize what needs to be remembered



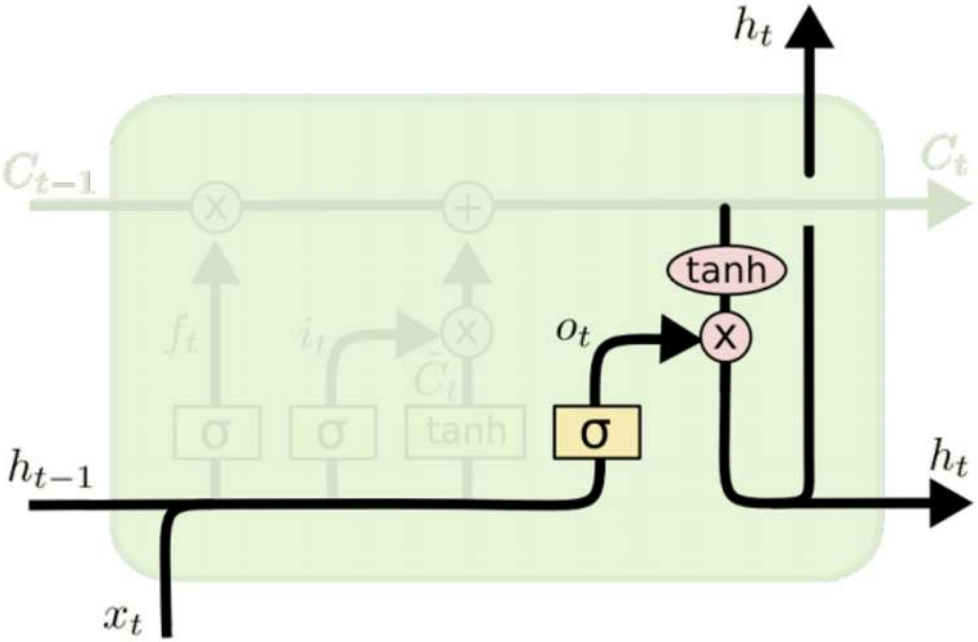
There are 3 gates with 2 outputs.

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$
$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$
$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$
$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$
$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

Inside LSTM: Output Gate



Should we output this bit of information (e.g., to “deeper” LSTM layers)?



There are 3 gates with 2 outputs.

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$

$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

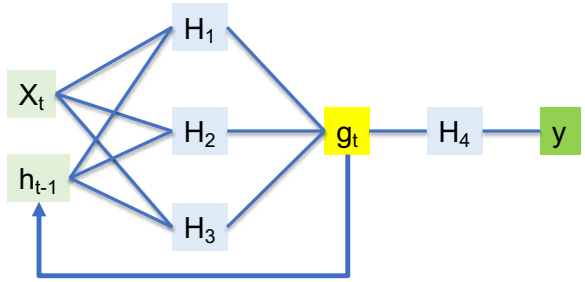
$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$

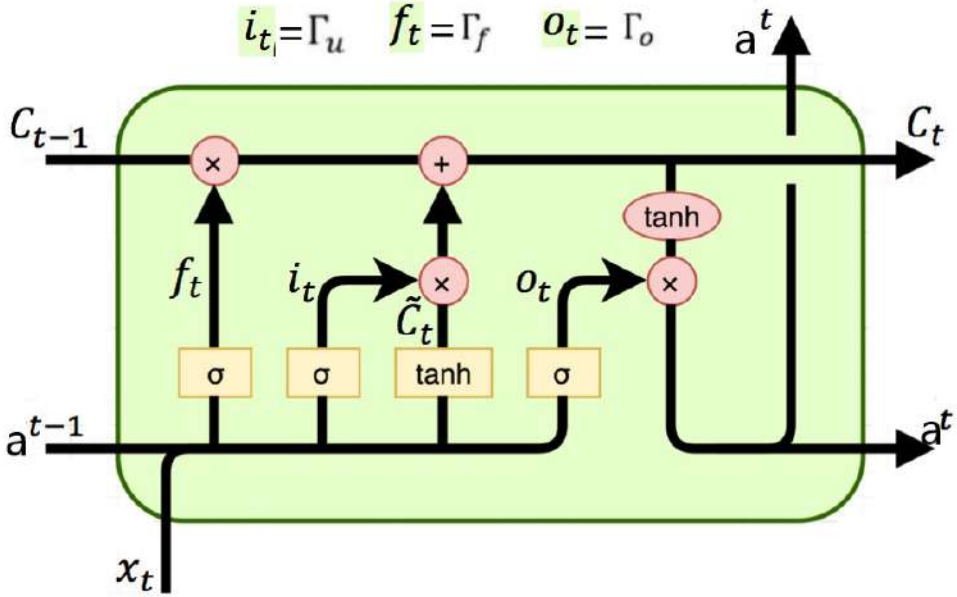
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

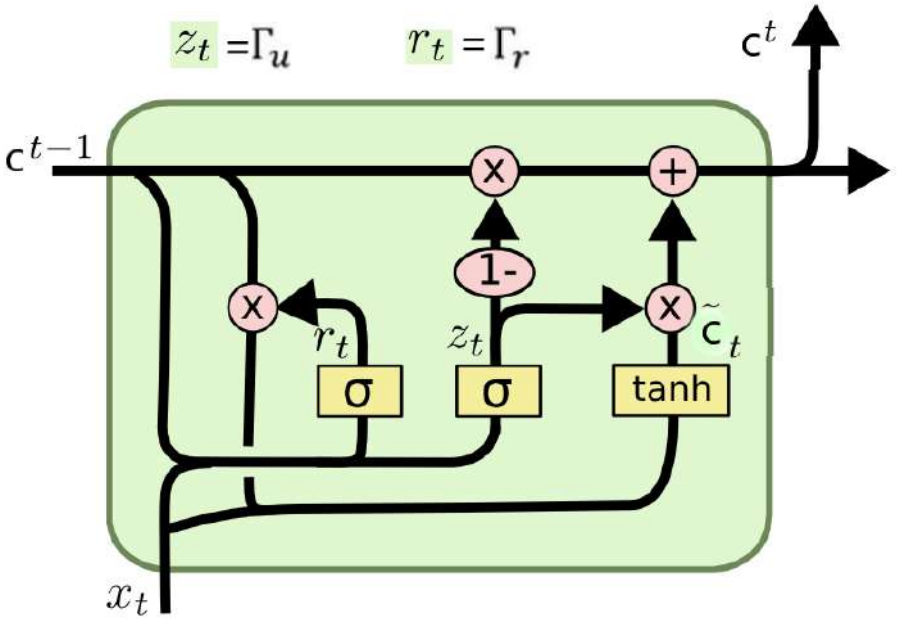
LSTM vs. Gated Recurrent Units (GRU)



There are 3 gates (i_t, f_t, o_t)
2 outputs (C(t), a(t)),
3 inputs (x(t), C(t-1), a(t-1))

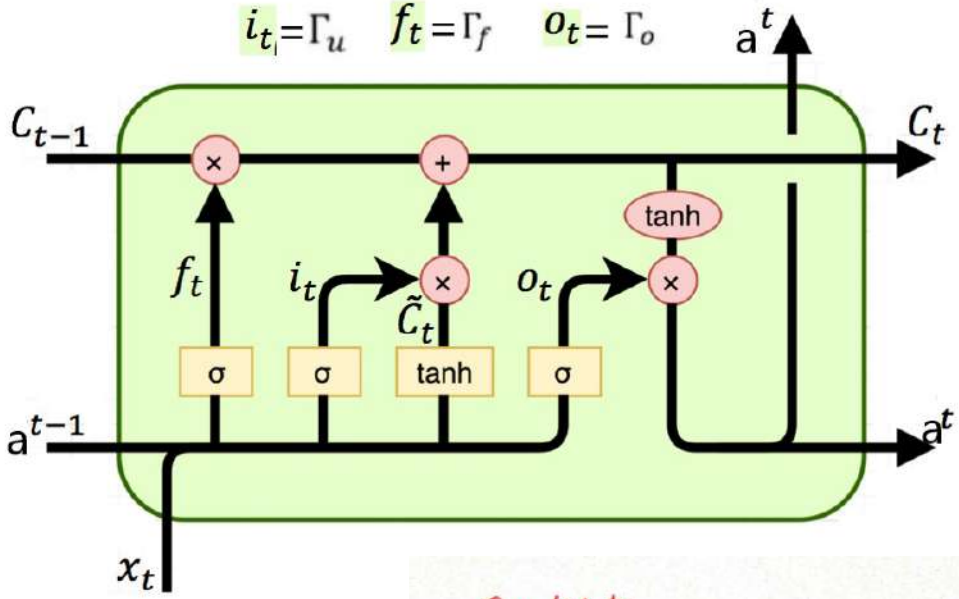


- There are 2 gates (update & reset gates)
- 1 output
- 2 inputs (x(t), C(t-1), ~~a(t-1)~~)



https://medium.com/analytics-vidhya/rnn-vs-gru-vs-lstm-863b0b7b1573

There are 3 gates with 2 outputs.



$$\mathbf{i}_{\langle t \rangle} = \sigma(\mathbf{W}_{xi}{}^T \cdot \mathbf{x}_{\langle t \rangle} + \mathbf{W}_{hi}{}^T \cdot \mathbf{h}_{\langle t-1 \rangle} + \mathbf{b}_i)$$
$$\mathbf{f}_{\langle t \rangle} = \sigma(\mathbf{W}_{xf}{}^T \cdot \mathbf{x}_{\langle t \rangle} + \mathbf{W}_{hf}{}^T \cdot \mathbf{h}_{\langle t-1 \rangle} + \mathbf{b}_f)$$
$$\mathbf{o}_{\langle t \rangle} = \sigma(\mathbf{W}_{xo}{}^T \cdot \mathbf{x}_{\langle t \rangle} + \mathbf{W}_{ho}{}^T \cdot \mathbf{h}_{\langle t-1 \rangle} + \mathbf{b}_o)$$
$$\mathbf{g}_{\langle t \rangle} = \tanh(\mathbf{W}_{xg}{}^T \cdot \mathbf{x}_{\langle t \rangle} + \mathbf{W}_{hg}{}^T \cdot \mathbf{h}_{\langle t-1 \rangle} + \mathbf{b}_g)$$
$$\mathbf{c}_{\langle t \rangle} = \mathbf{f}_{\langle t \rangle} \otimes \mathbf{c}_{\langle t-1 \rangle} + \mathbf{i}_{\langle t \rangle} \otimes \mathbf{g}_{\langle t \rangle}$$
$$\mathbf{y}_{\langle t \rangle} = \mathbf{h}_{\langle t \rangle} = \mathbf{o}_{\langle t \rangle} \otimes \tanh(\mathbf{c}_{\langle t \rangle})$$

g(t) Candidate cell → $\tilde{c}^{\langle t \rangle} = \tanh(W_c[a^{\langle t-1 \rangle} + x^{\langle t \rangle}] + b_c)$

i(t) Update / Input gate → $\Gamma_u = \sigma(W_u[a^{\langle t-1 \rangle} + x^{\langle t \rangle}] + b_u)$

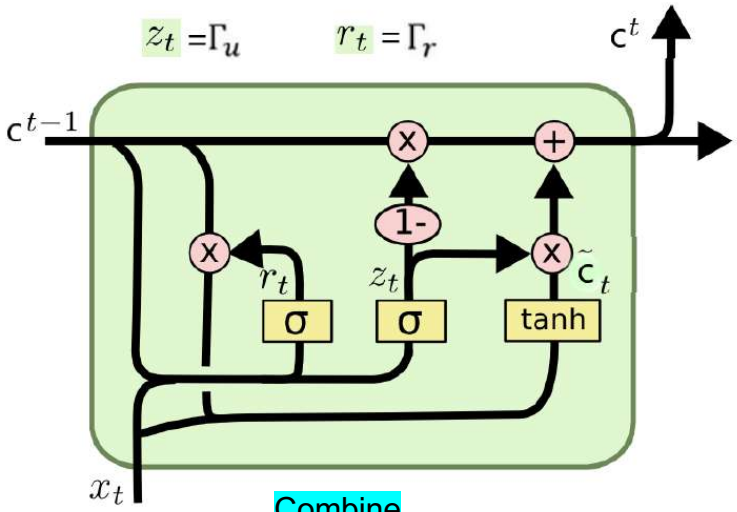
f(t) Forget gate → $\Gamma_f = \sigma(W_f[a^{\langle t-1 \rangle} + x^{\langle t \rangle}] + b_f)$

o(t) Output gate → $\Gamma_o = \sigma(W_o[a^{\langle t-1 \rangle} + x^{\langle t \rangle}] + b_o)$

C(t) Cell state → $c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + \Gamma_f * c^{\langle t-1 \rangle}$

y(t) Activation → $a^{\langle t \rangle} = \Gamma_o * \tanh c^{\langle t \rangle}$

- There are 2 gates (update & reset gates)
- 1 output
- 2 inputs (x(t), C(t-1), ~~a(t-1)~~)



Combine
- Previous → how much to reset C(t-1)
- Current [x(t)]

reset previous memory

g(t) Candidate cell → $\tilde{c}^{\langle t \rangle} = \tanh(W_c[\Gamma_r * c^{\langle t-1 \rangle} + x^{\langle t \rangle}] + b_c)$

i(t) Update gate → $\Gamma_u = \sigma(W_u[c^{\langle t-1 \rangle} + x^{\langle t \rangle}] + b_u)$

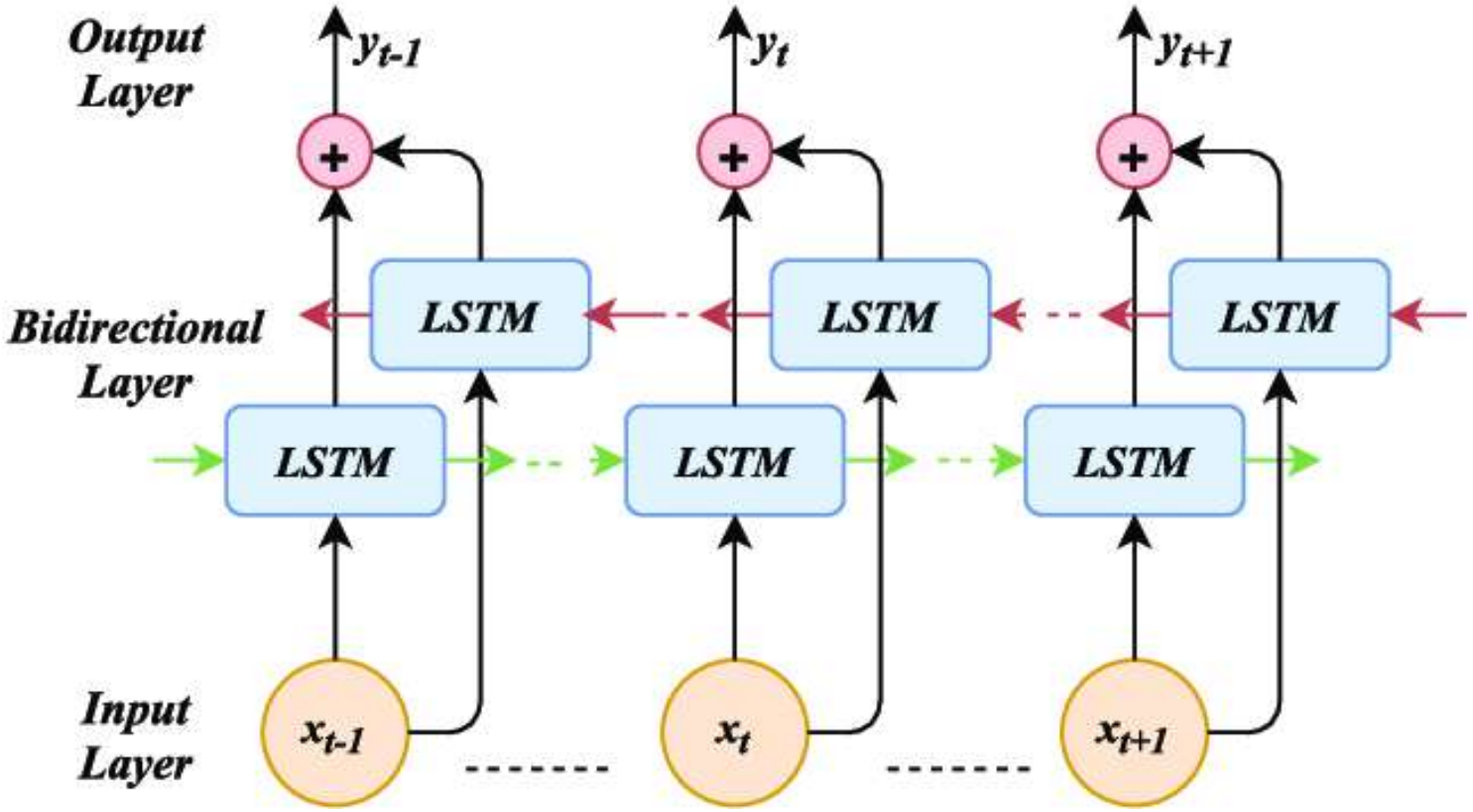
f(t) Reset gate → $\Gamma_r = \sigma(W_r[c^{\langle t-1 \rangle} + x^{\langle t \rangle}] + b_r)$

C(t) Cell state → $c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$

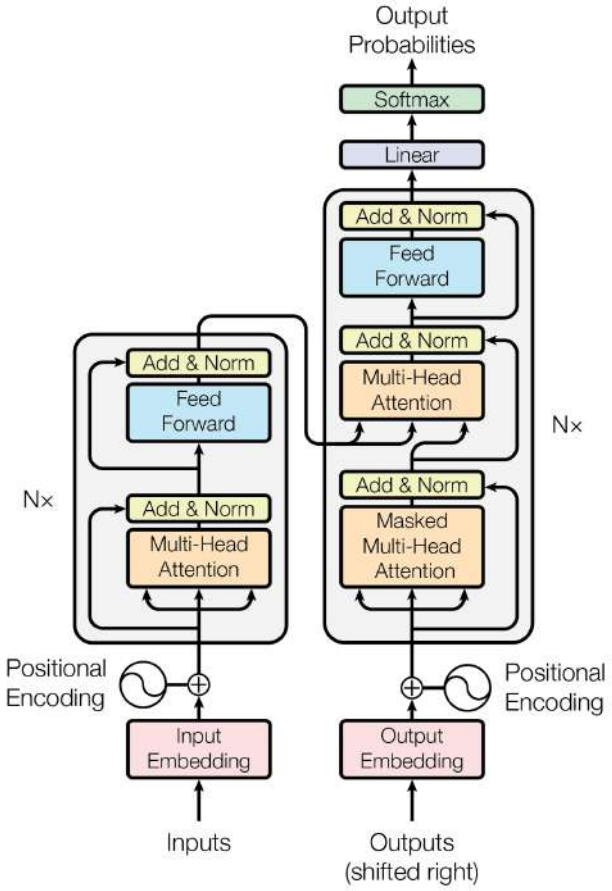
Update current memory

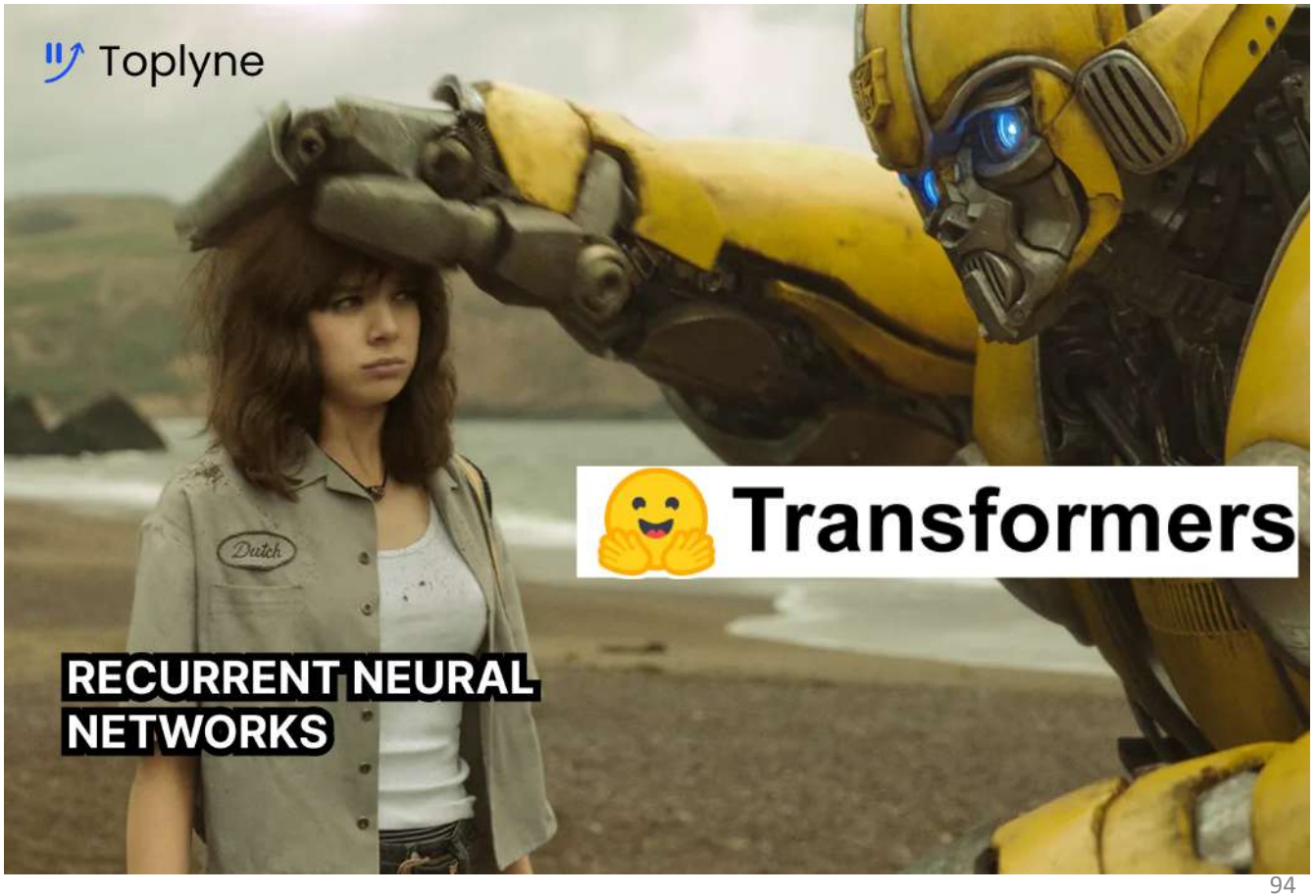
y(t) Activation / Output → $a^{\langle t \rangle} = c^{\langle t \rangle}$

Bidirectional LSTM



Rise of Transformer (2017)



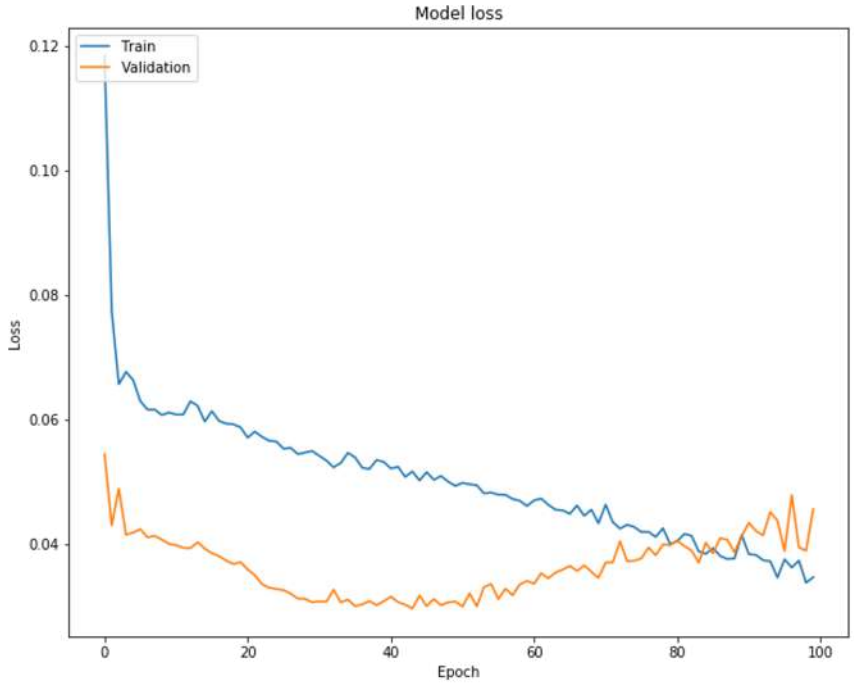


LSTM Parameters Tuning

- Epochs
- Learning Rate
- LSTM Parameters
  - Number of hidden Unit
  - Number of layers
- Loss function

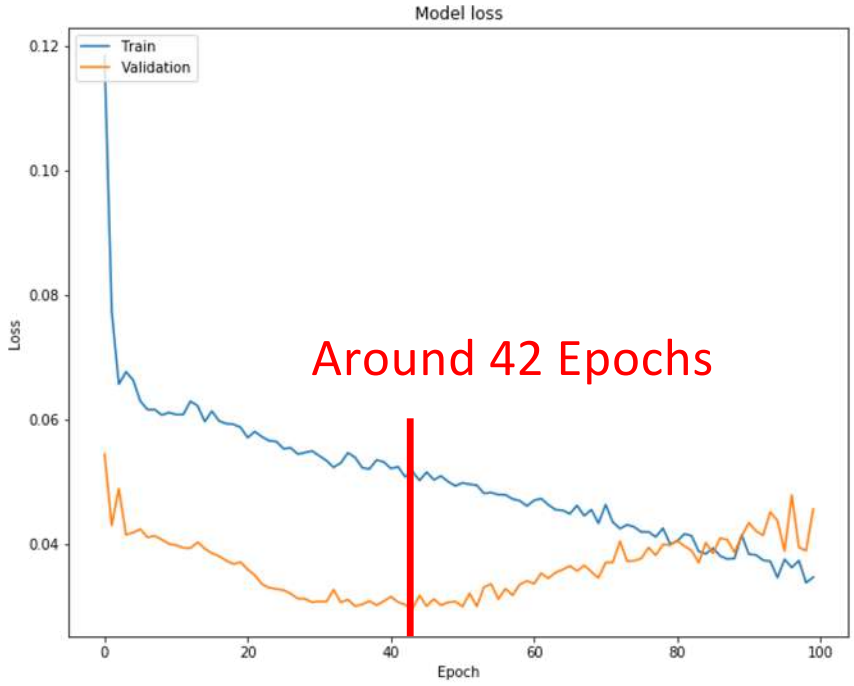
LSTM Parameters Tuning: Epoch

- The single time you see all examples in the dataset.
- Choosing by Plotting Train, Validation loss



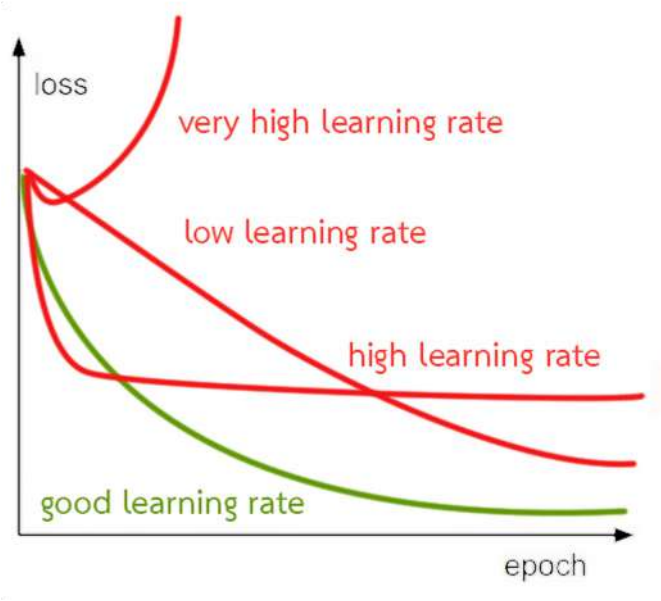
LSTM Parameters Tuning: Epoch (cont.)

- The single time you see all examples in the dataset.
- Choosing by Plotting Train, Validation loss



LSTM Parameters Tuning: Learning Rate

- The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated.

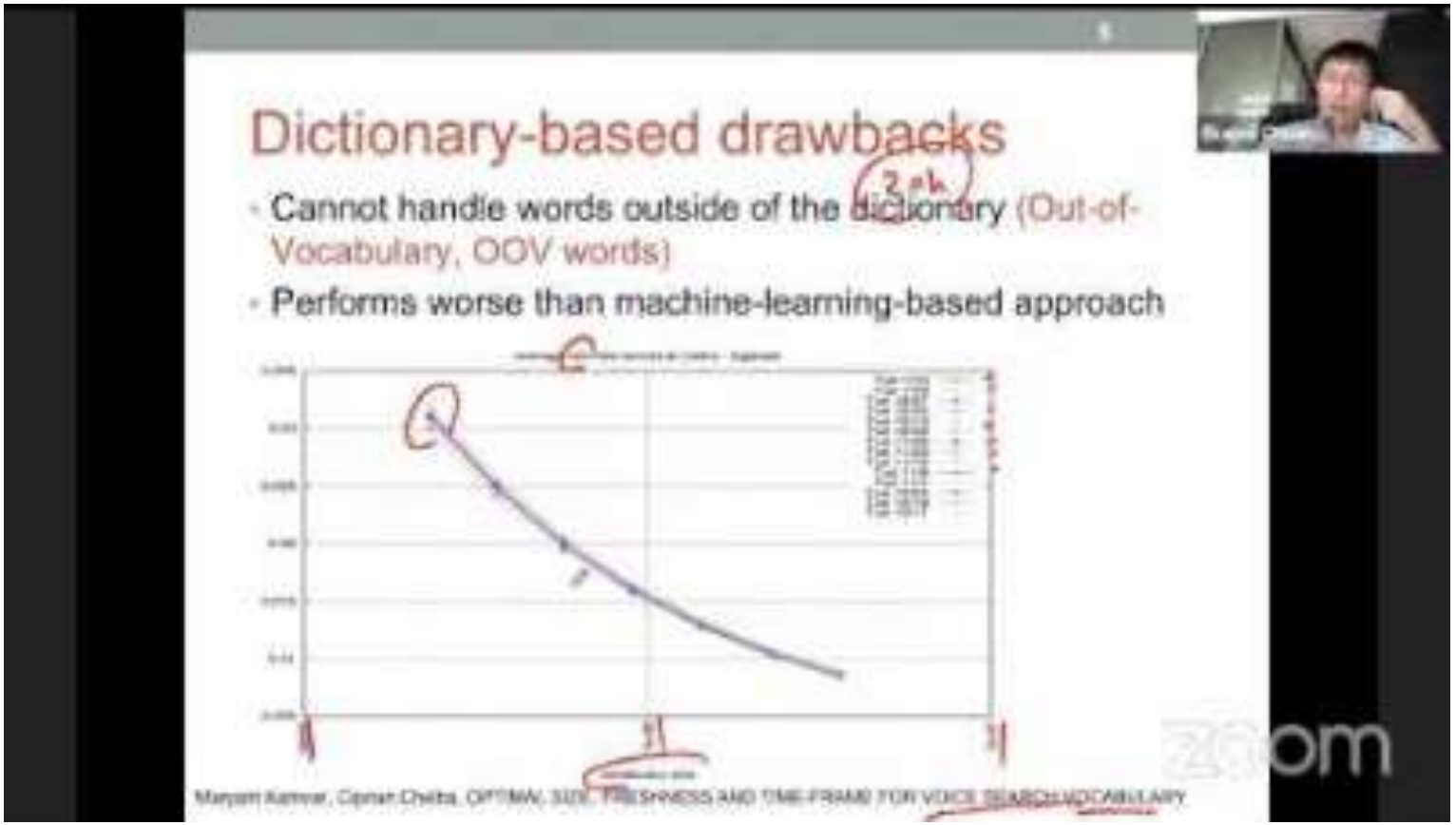


LSTM Parameters Tuning: Loss Function

- We are currently use Mean Square Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

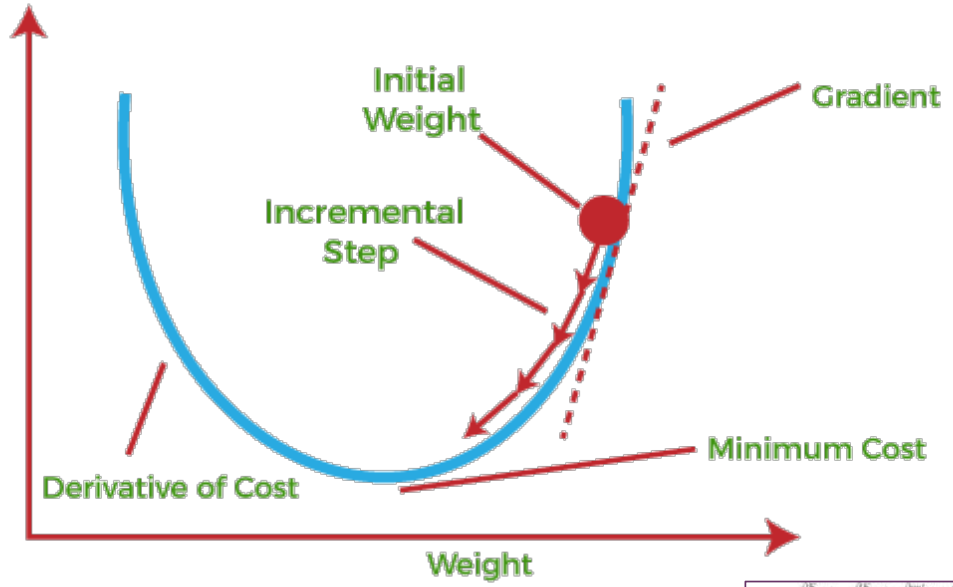
NLP Lecture2 (2020s2) by Aj.Ekapol
https://youtu.be/WxiO3wvKhCE



Appendix

Differences Between Epoch, Batch, and Mini-batch

- The gradient descent algorithm works in two steps that are performed continuously for a specific number of iterations:
  - First, we compute the gradient, which is the first-order derivative of the objective function with respect to the variables.
  - Then, we update the variables in the opposite direction of the gradient
- Terms:
  - Epoch = use all training examples
  - Iteration = each round of updating weight

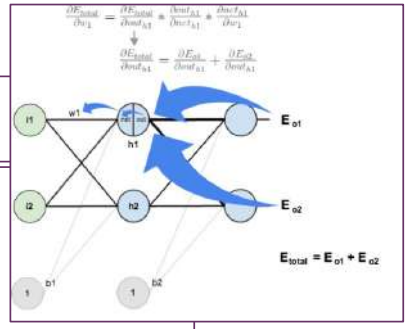


$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

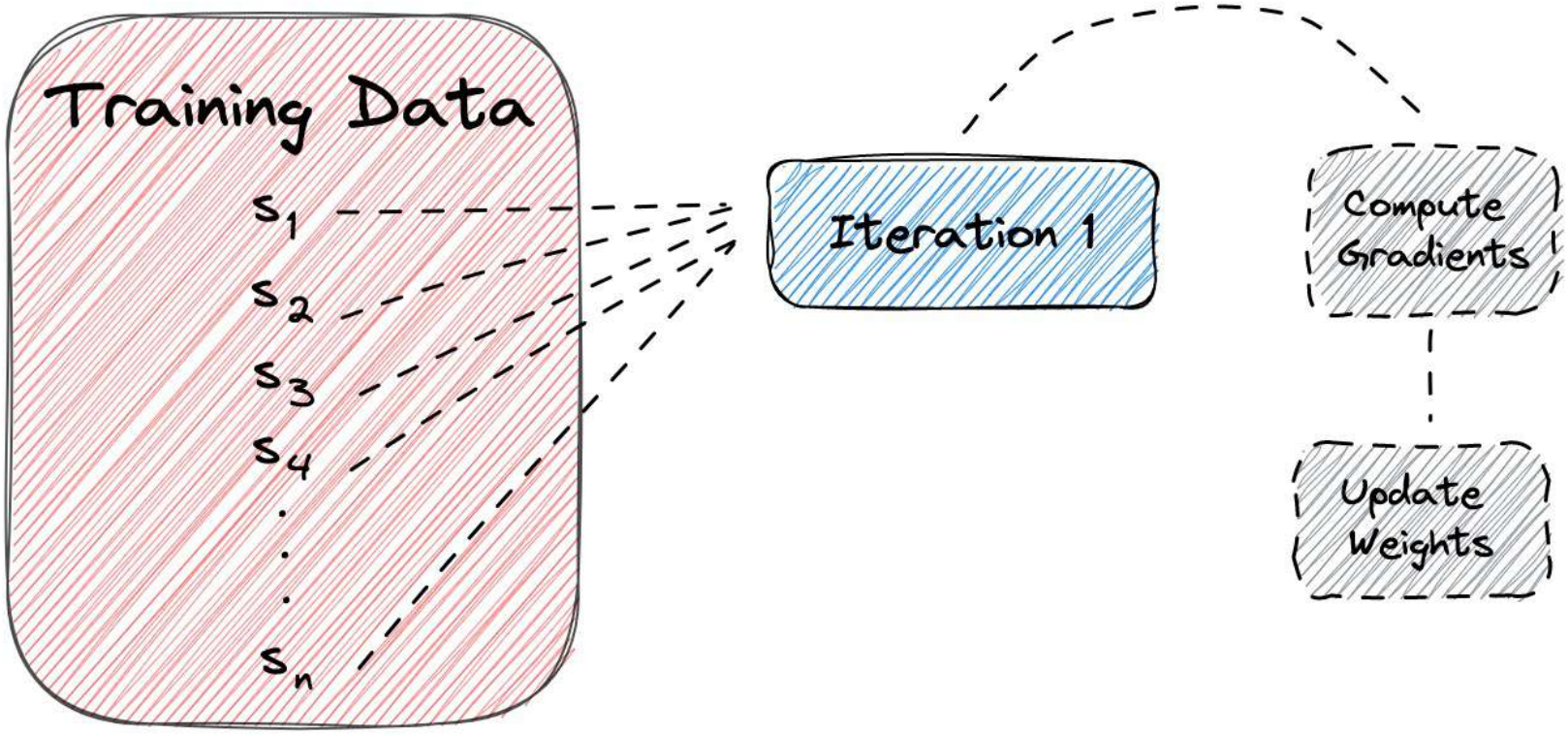
$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$



https://www.baeldung.com/cs/epoch-vs-batch-vs-batch#:~:text=So%2C%20a%20batch%20is%20equal,in%20mini%2Dbatch%20gradient%20descent

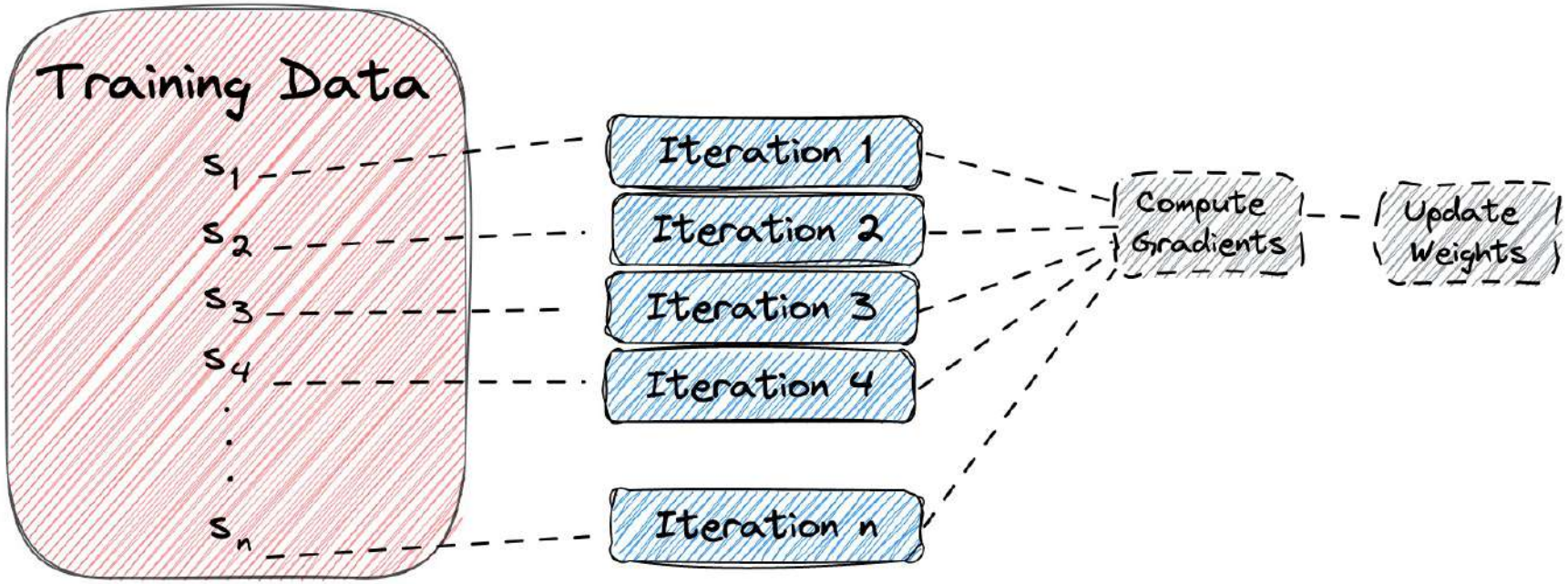
1) Batch Gradient Descent

- In batch gradient descent, we use all our training data in a single iteration of the algorithm.



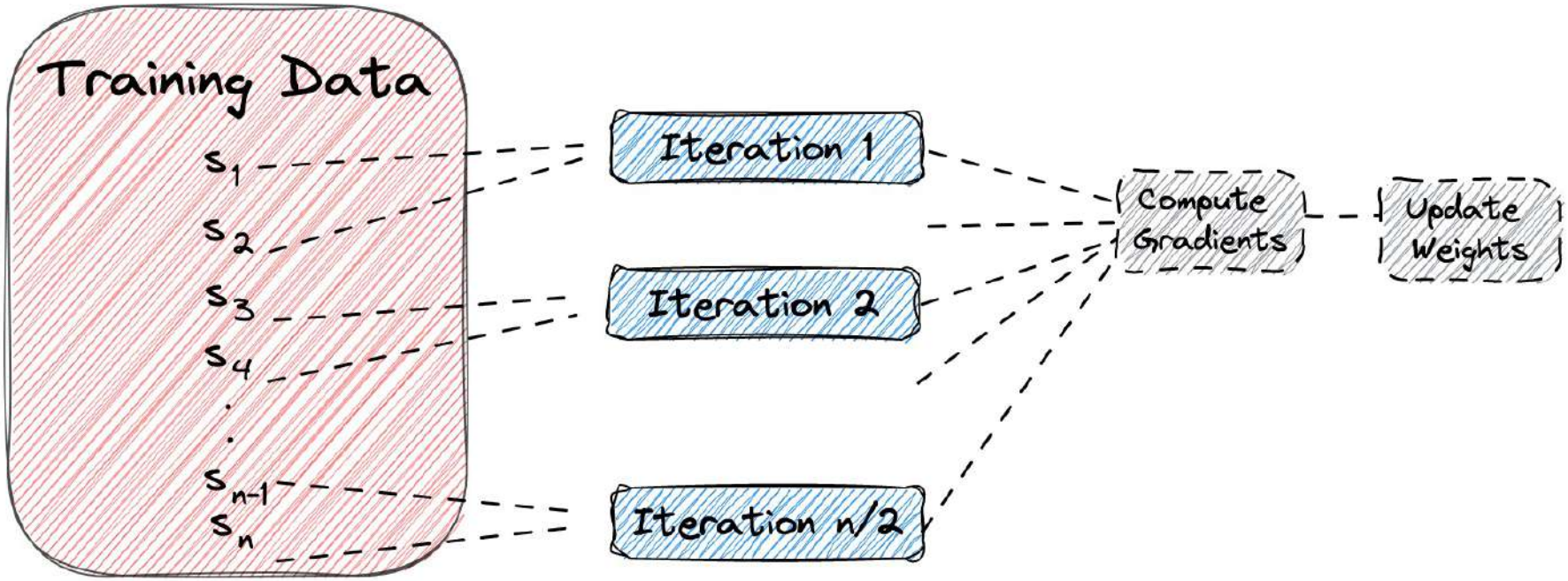
2) Stochastic Gradient Descent

- The previous method can be very time-consuming and inefficient in case the size of the training dataset is large. To deal with this, we use stochastic gradient descent, where we use just one sample in a single iteration of the algorithm.



3) Mini-Batch Gradient Descent

- Mini-batch gradient descent is a combination of the previous methods where we use a group of samples called mini-batch in a single iteration of the training algorithm.
- The mini-batch is a fixed number of training examples that is less than the actual dataset. So, in each iteration, we train the network on a different group of samples until all samples of the dataset are used.



Conclusion about batch

An epoch means that we have passed each sample of the training set one time through the network to update the parameters. Generally, the number of epochs is a hyperparameter that defines the number of times that gradient descent will pass the entire dataset.

If we look at the previous methods, we can see that:

- In batch gradient descent, one epoch corresponds to a single iteration.
- In stochastic gradient descent, one epoch corresponds to n iterations where n is the number of training samples.
- In mini-batch gradient descent, one epoch corresponds to $\frac{n}{b}$ iterations where b is the size of the mini-batch.

Finally, let's present a simple example to better understand the three terms.

Let's assume that we have a dataset with $\mathbf{n = 2000}$ samples, and we want to train a deep learning model using gradient descent for $\mathbf{10}$ epochs and mini-batch size $\mathbf{b = 4}$:

- In batch gradient descent, we'll update the network's parameters (using all the data) 10 times which corresponds to 1 time for each epoch.
- In stochastic gradient descent, we'll update the network's parameters (using one sample each time) $2000 * 10 = 20000$ times which corresponds to 2000 times for each epoch.
- In mini-batch gradient descent, we'll update the network's parameters (using $b = 4$ samples each time) $\frac{2000}{4} * 10 = 5000$ times that corresponds to $\frac{2000}{4} = 500$ times for each epoch.