



CHULA ENGINEERING
Foundation toward Innovation

COMPUTER



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]



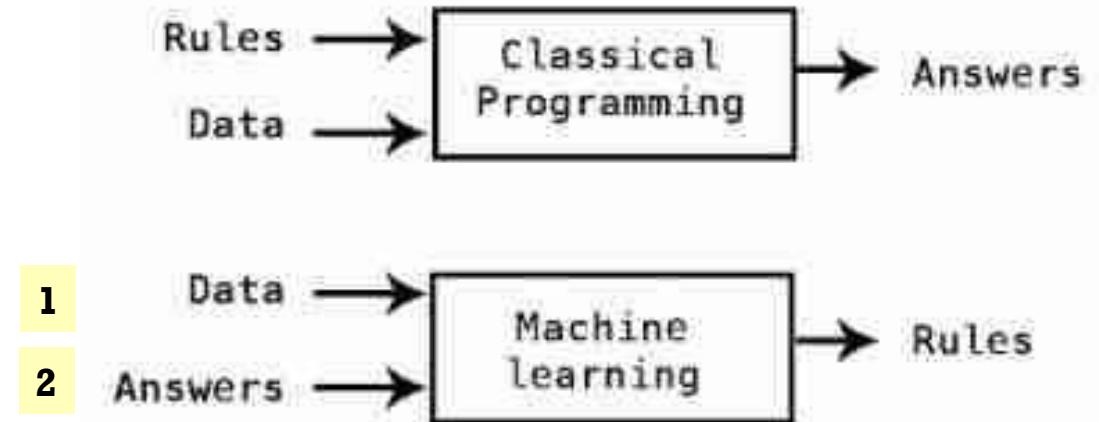
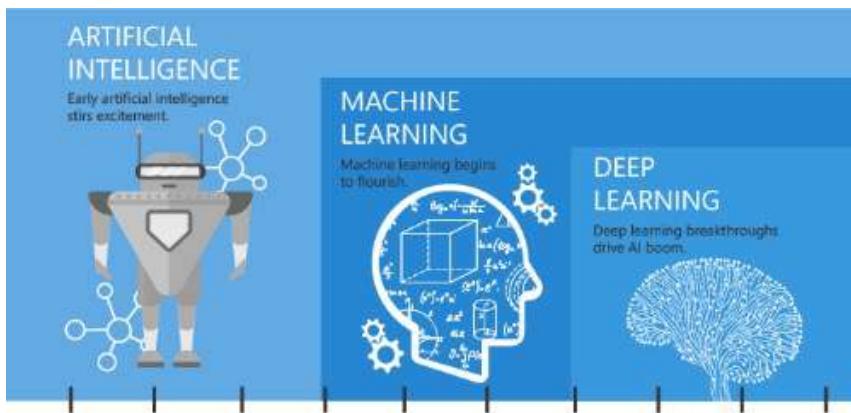


Introduction to Deep Learning



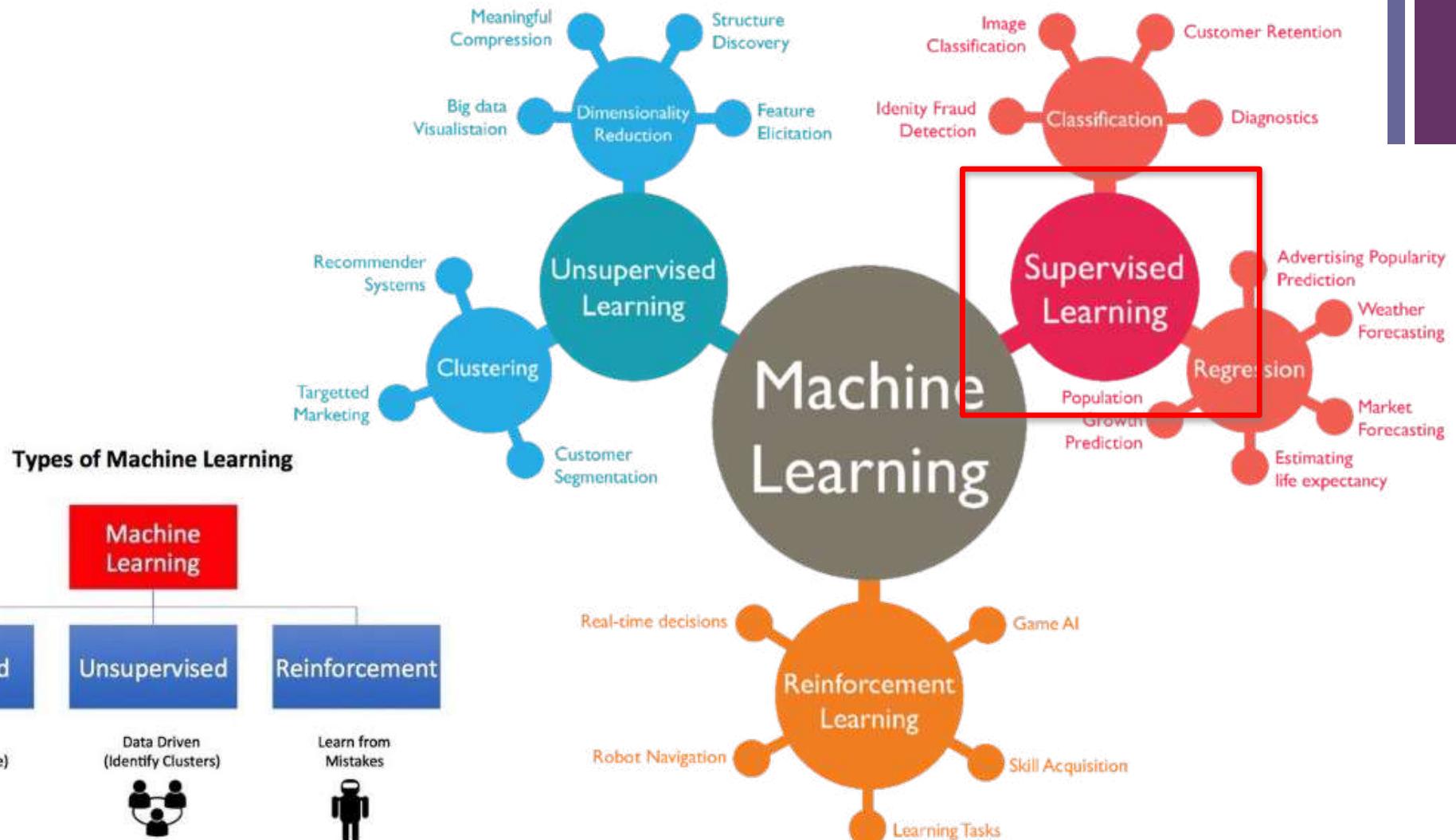
AI = Automation

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



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

+ Machine Learning





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



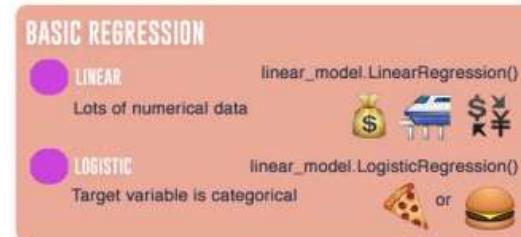
Age	Gender	BodyTemp	Cough	Corona
25	Male	40	No	?

Application: Corona Prediction



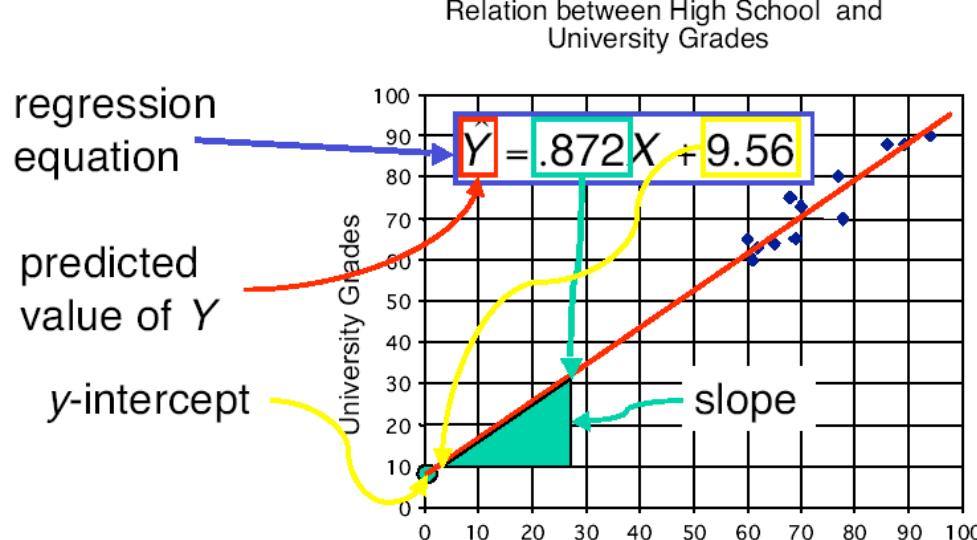
Prediction algorithms

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





Regression – Linear Relationship



$$\hat{y} = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

target intercept input

weight, coefficient

- The least square method aims to minimize the following term

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



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	

$$\text{Logit_score} = w_0 + w_1 * \text{Age} + w_2 * \text{Gender} + w_3 * \text{Temp} + w_4 * \text{Cough}$$

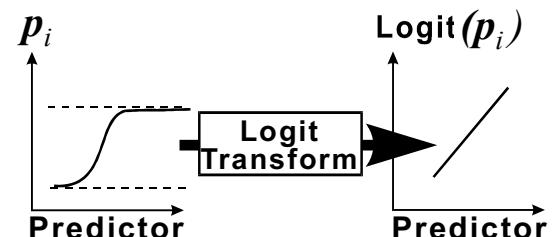
$$\text{Logit_score} = 0.01 - 0.3 * \text{Age} + 0.2 * \text{Gender} + 0.2 * \text{Temp} + 0.9 * \text{Cough}$$

Example

$$\text{Logit_score} = 0.01 - 0.3 * 12 + 0.2 * 0 + 0.2 * 37 + 0.9 * 1 = 4.71$$

prob = 0.9911 → "Yes"

Application: Corona Prediction

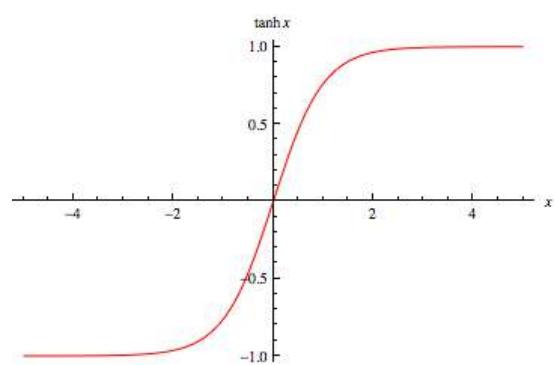
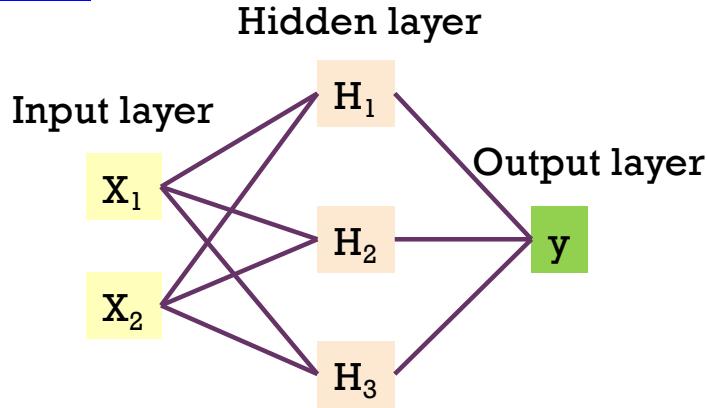


$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$



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?

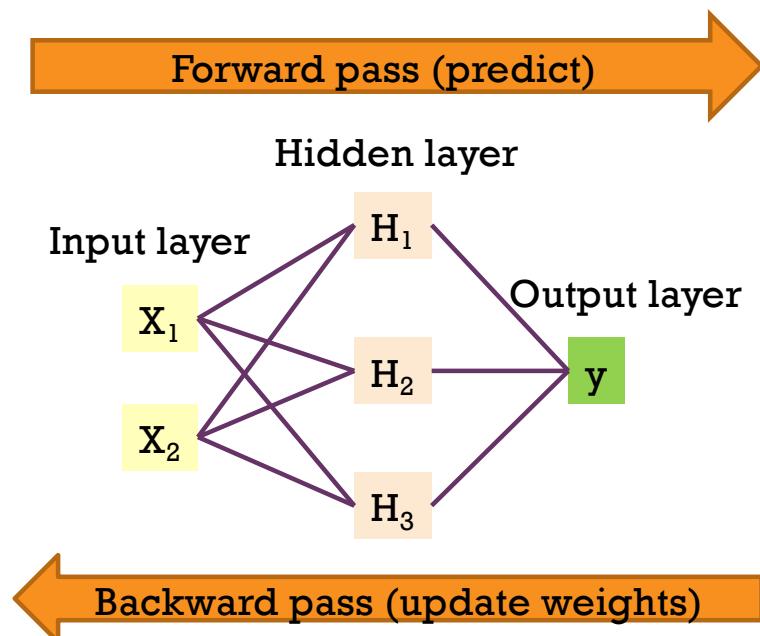
- Converge (no change in loss)
- Max epochs

Important Params:

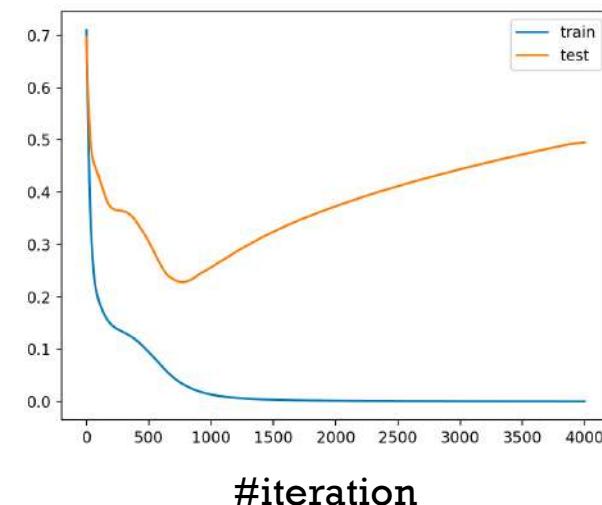
- #hidden units, #hidden layers
- Learning rate, Momentum, decay
- Seed number, etc.



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



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

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

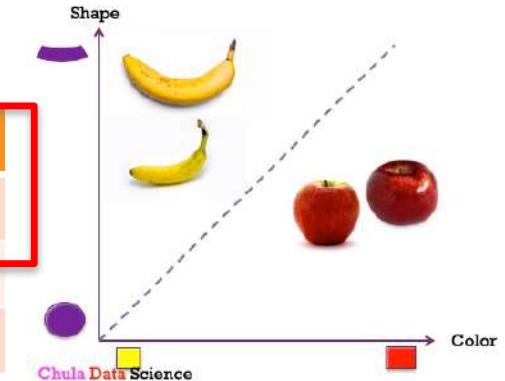
$$H_2 = \tanh(\hat{w}_{20} + \hat{w}_{21} x_1 + \hat{w}_{22} x_2)$$

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



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



12

Can we still tell the features (columns)?



shutterstock.com - 451802557



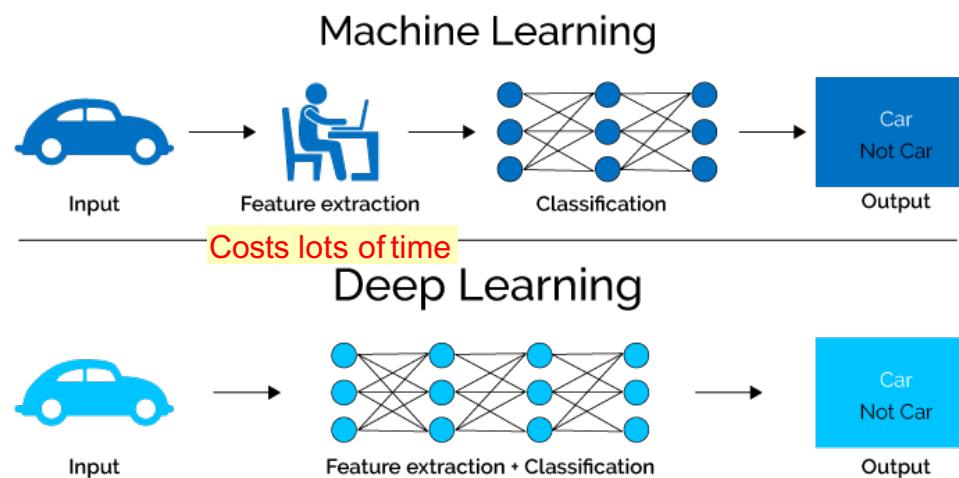
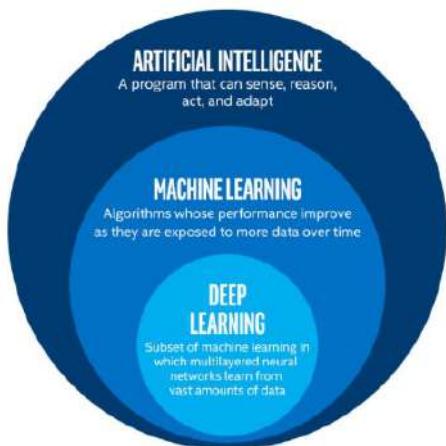
What is Deep Learning (DL)?



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.

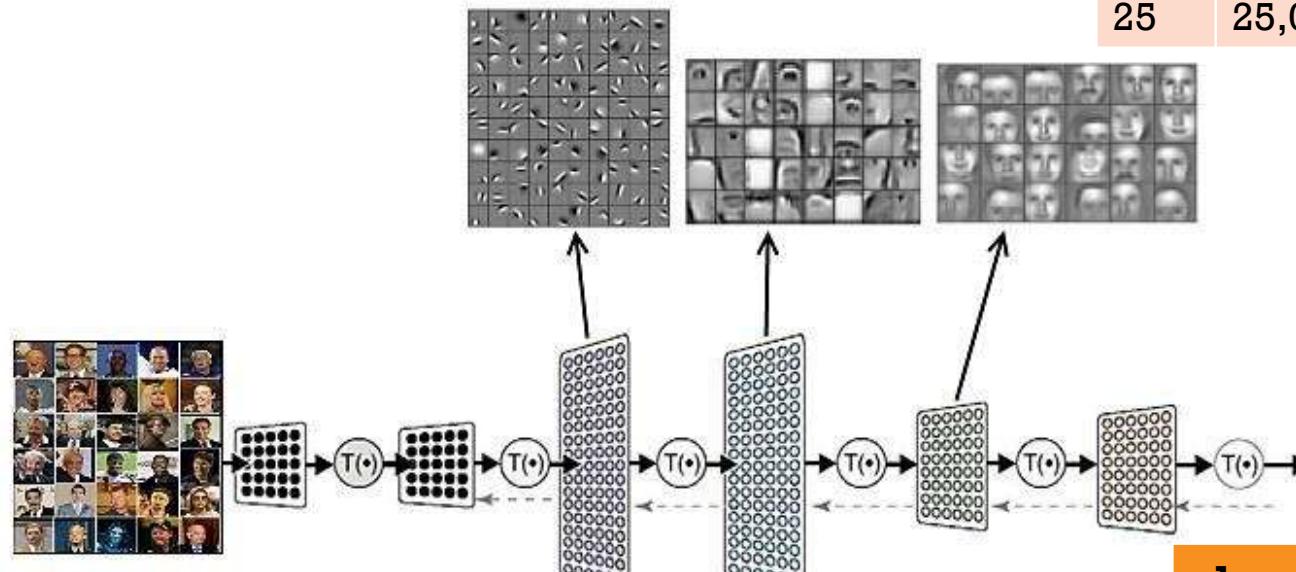




Deep Learning – Basics (cont.)

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

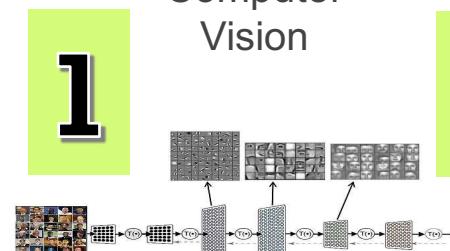
Deep Learning Application



Speech
Recognition



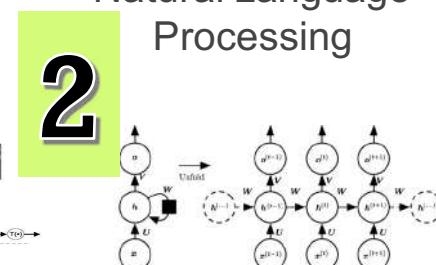
Computer
Vision



CNN

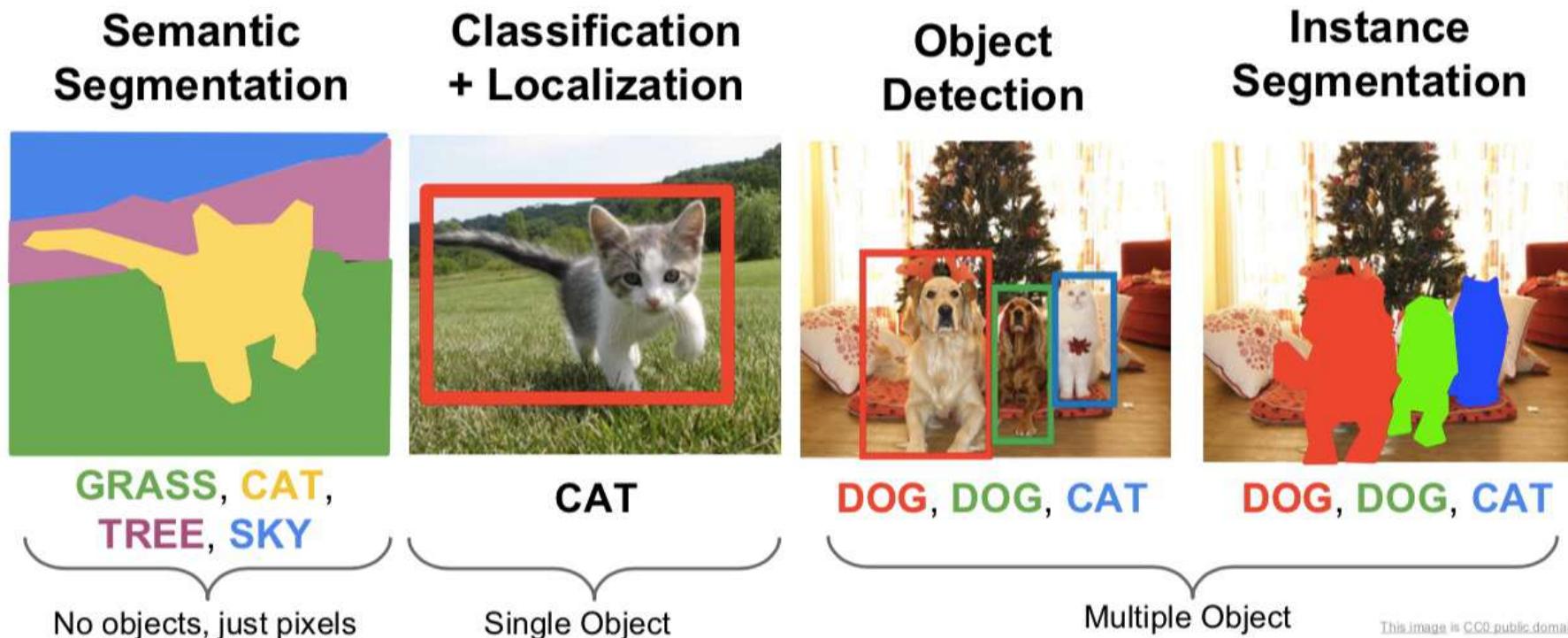


Natural Language
Processing



RNN

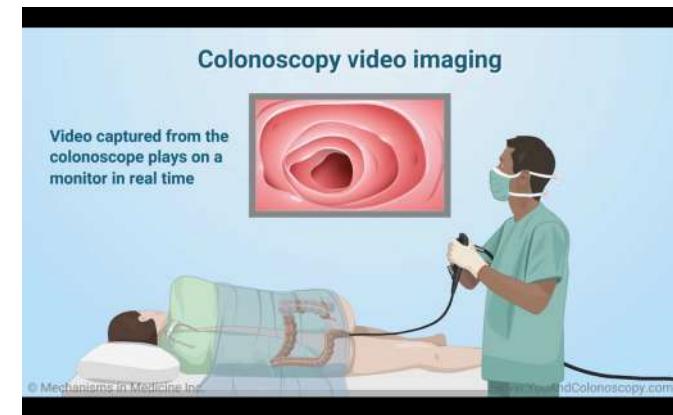
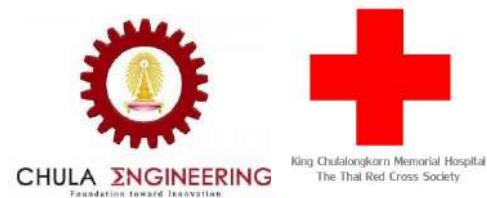
Type of image tasks



Face Recognition



Smart Medical Diagnosis



+

CNN



Outline

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

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.

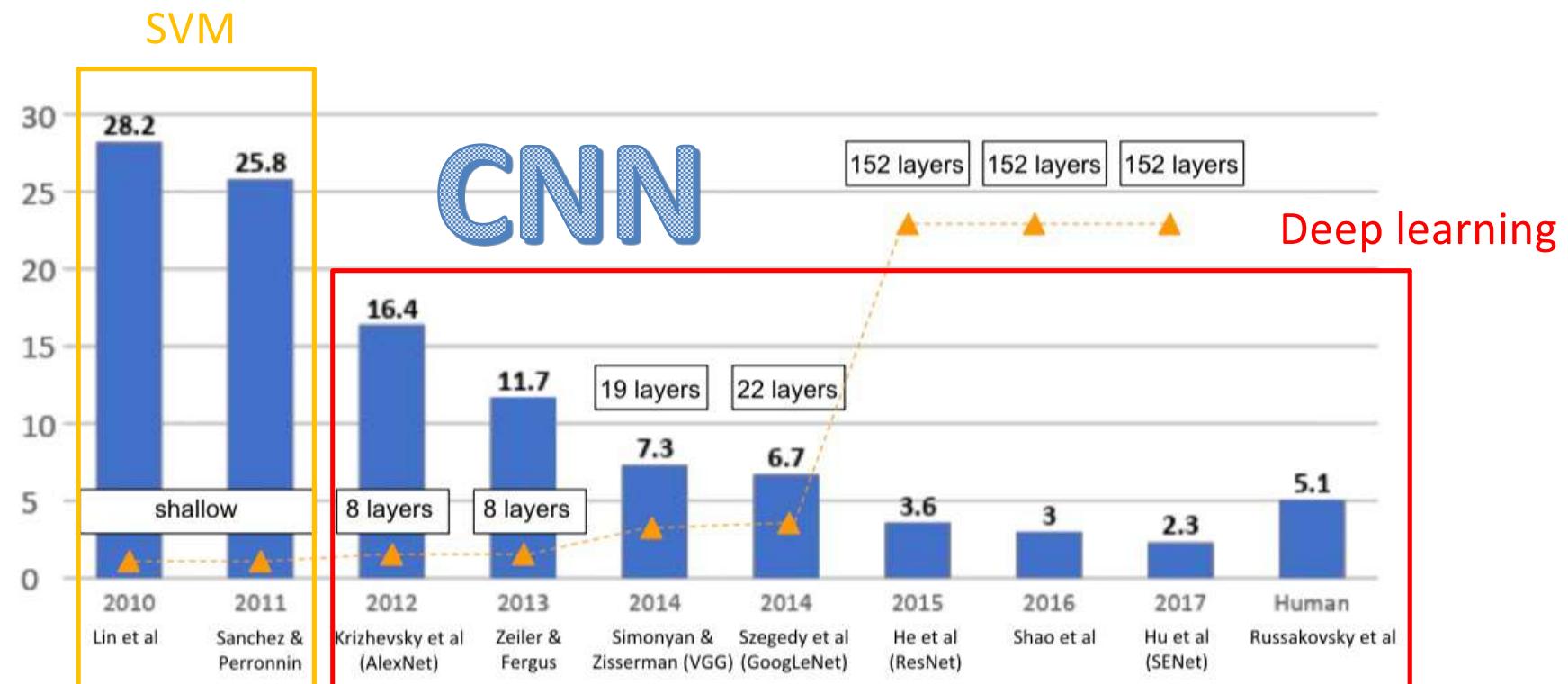


The Image Classification Challenge:
1,000 object classes
1,431,167 images
ImageNet 2017 is the last challenge.

ILSVRC

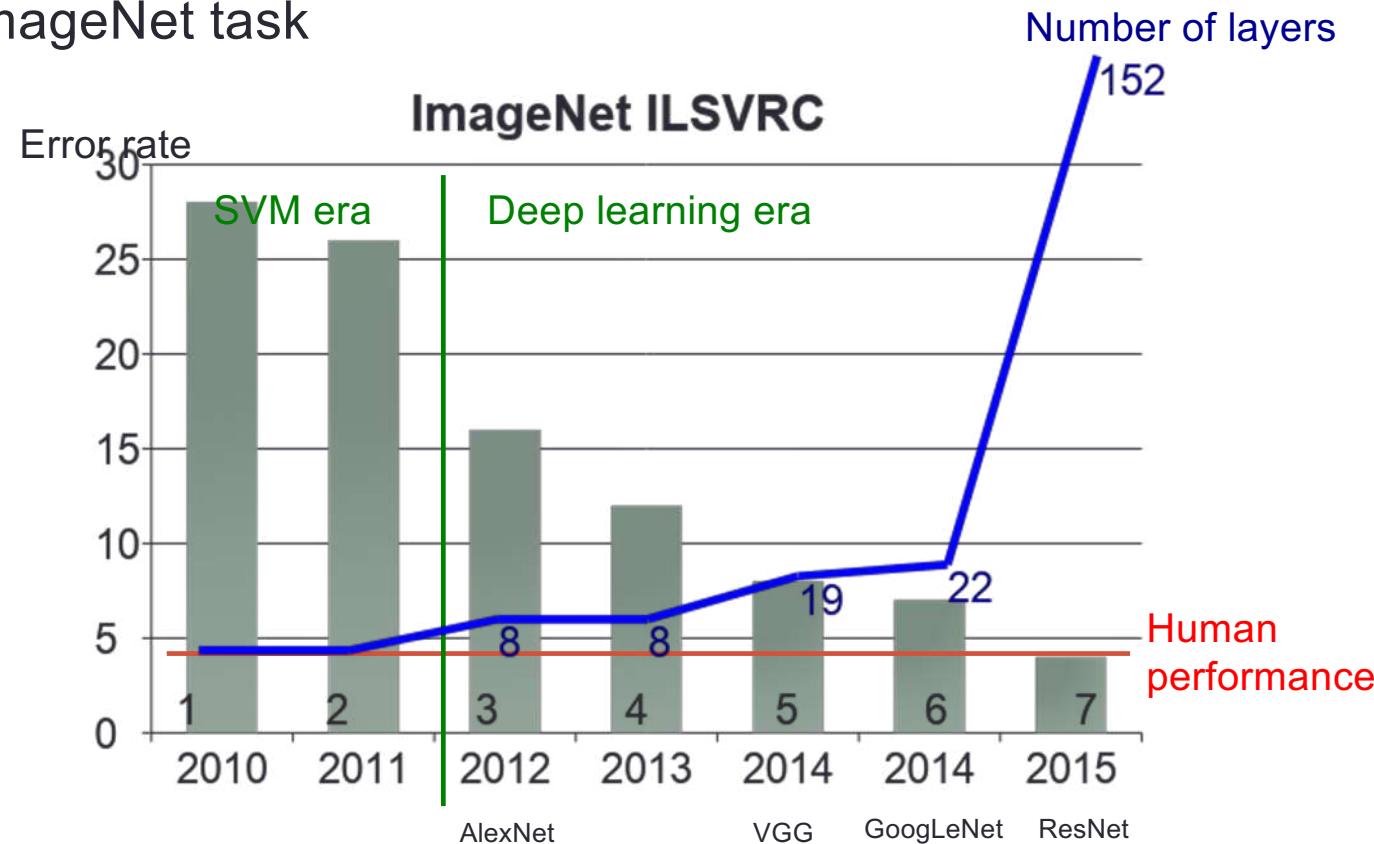


ImageNet Large Scale Visual Recognition Challenge (ILSVRC winners): From SVM to Deep Learning



Wider and deeper networks (Beyond Human)

- ImageNet task

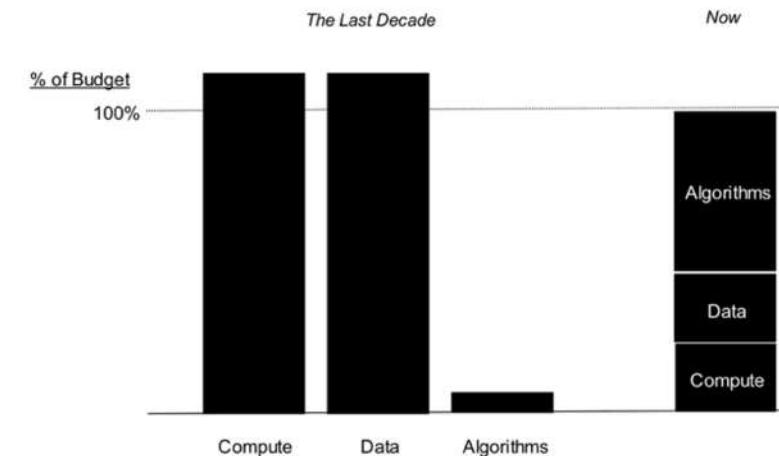


Based on the slide of Aj.Ekapol

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](http://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

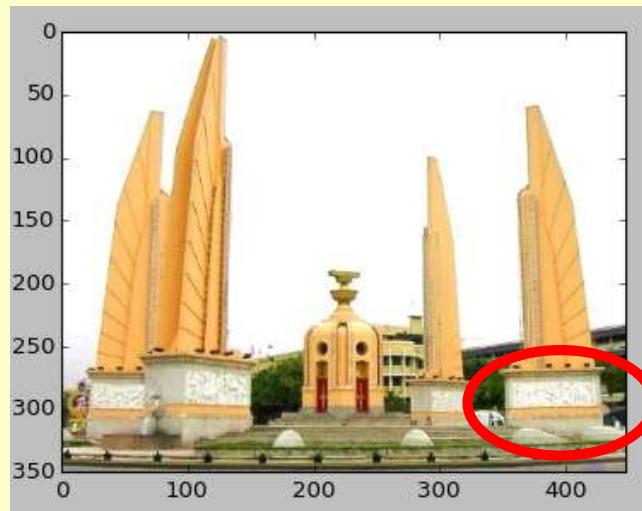
การแสดงรูปภาพใน python ด้วย matplotlib

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

เพื่อความง่ายโปรแกรมที่จะเขียน
จากนี้ไปใช้กับไฟล์ png เท่านั้น

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

image เป็นอาร์เรย์ที่แต่ละช่อง
มีค่า 0 ถึง 1 แทนความเข้มของสี



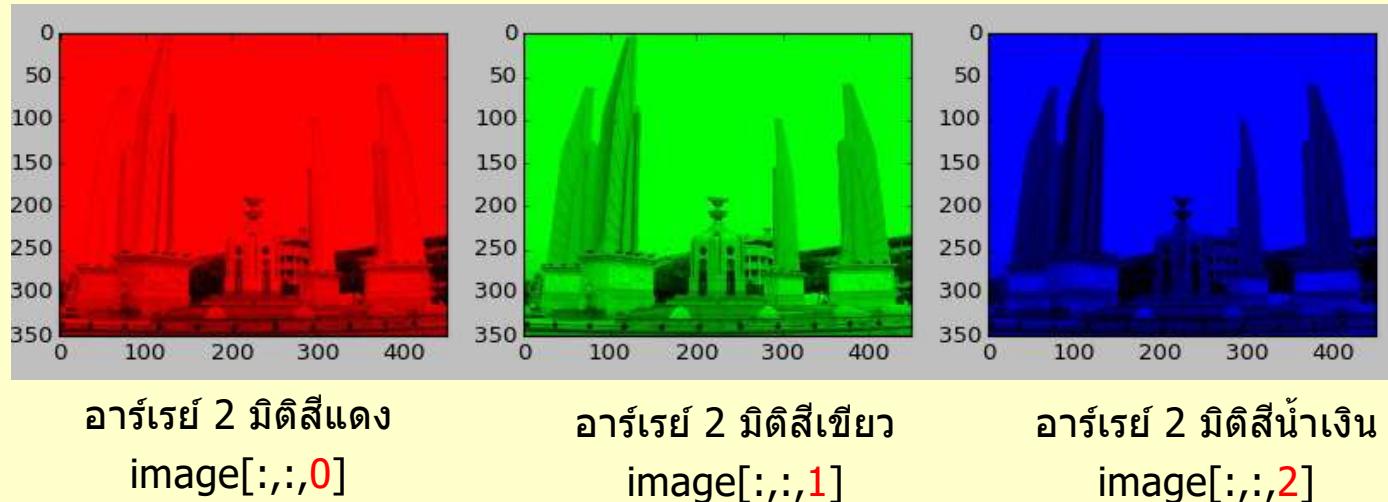
1 pixel = 1 จุดสี มี 3 สีอยู่

0.98762 | 0.72549 | 0.35294



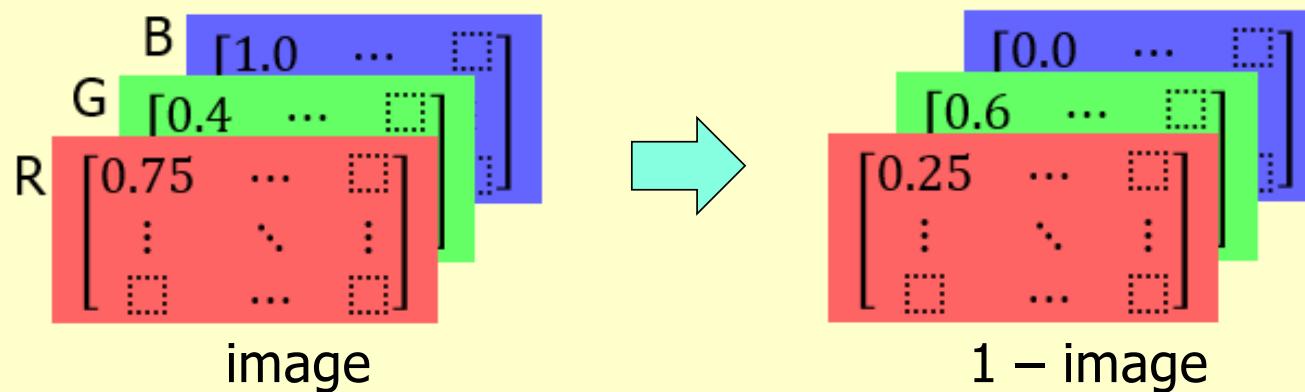
1 ภาพ เป็นอารเรย์ 3 สามมิติเก็บค่าความเข้มของ R G B

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



การประมวลผลภาพเบื้องต้น

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



ทำแบบนี้แล้ว ภาพเปลี่ยนแปลงไปอย่างไร ?

การประมวลผลภาพเบื้องต้น : ภาพ Negative

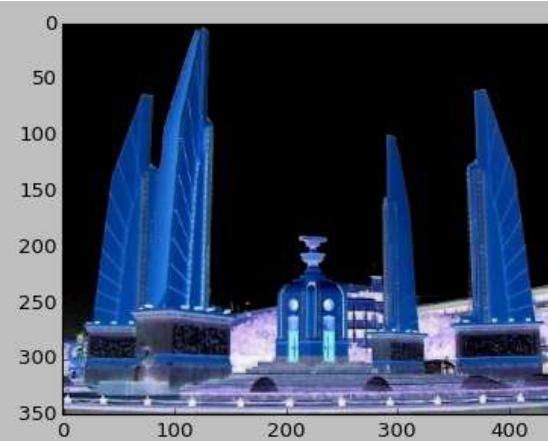
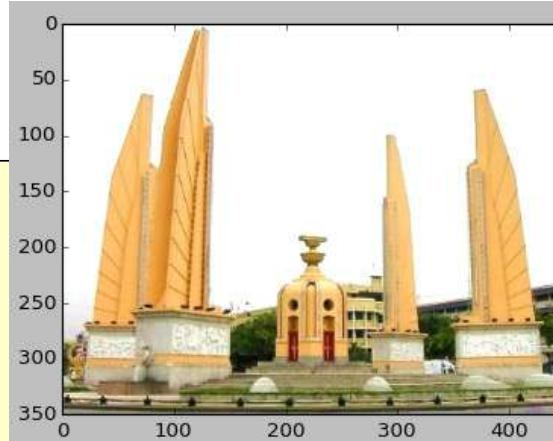
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

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 → 黑, เหลือง → น้ำเงิน, ...)
broadcast & element-wise subtract



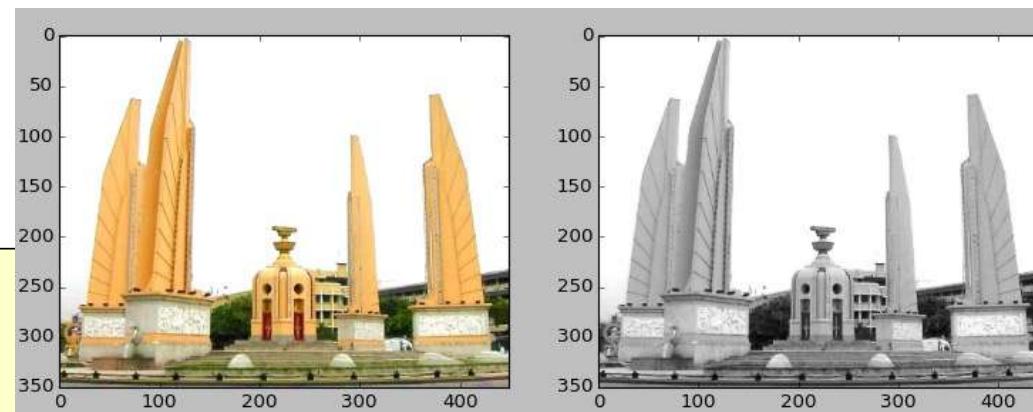
การประมวลผลภาพเบื้องต้น : ภาพสีเทา

```
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()
```

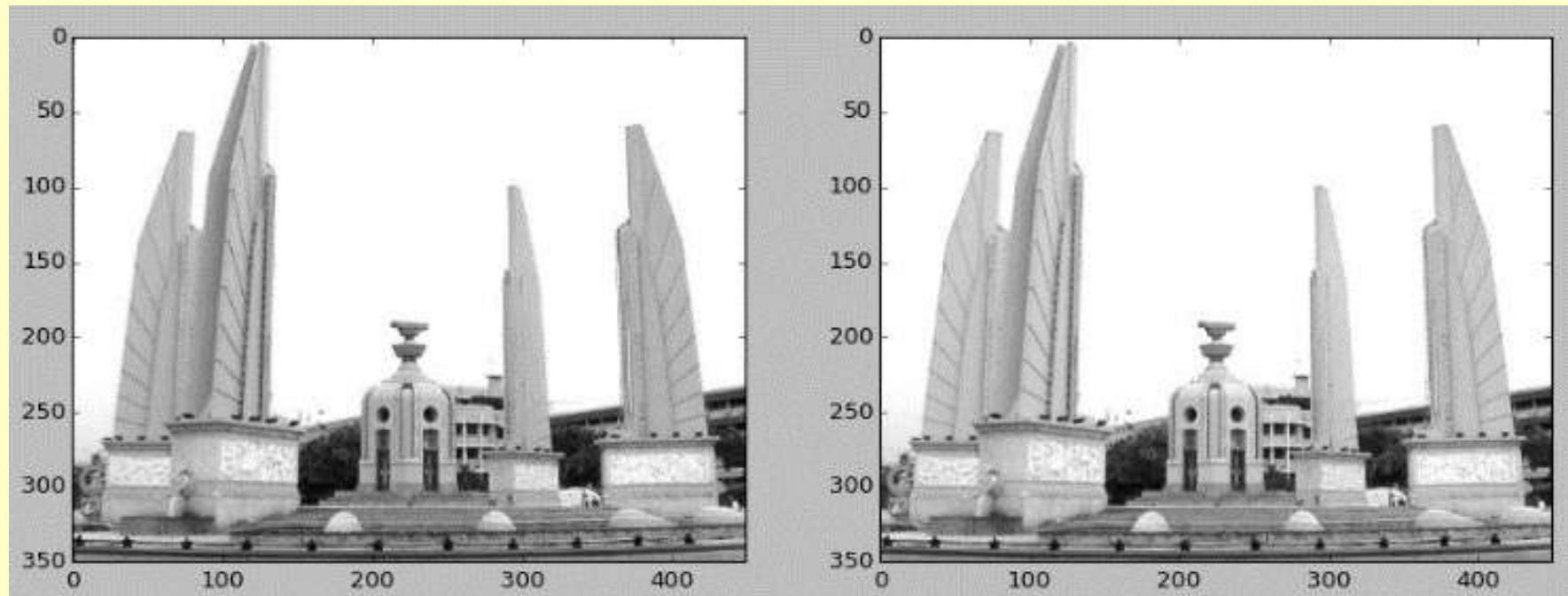
จุดสีที่ R G B มีค่าความเข้มเท่ากัน
ได้จุดสีเทา

นำค่าความเข้มของ R G B มาหารค่าเฉลี่ย^{แล้วเปลี่ยน R G B ให้เป็นความเข้มระดับเดียวกัน}

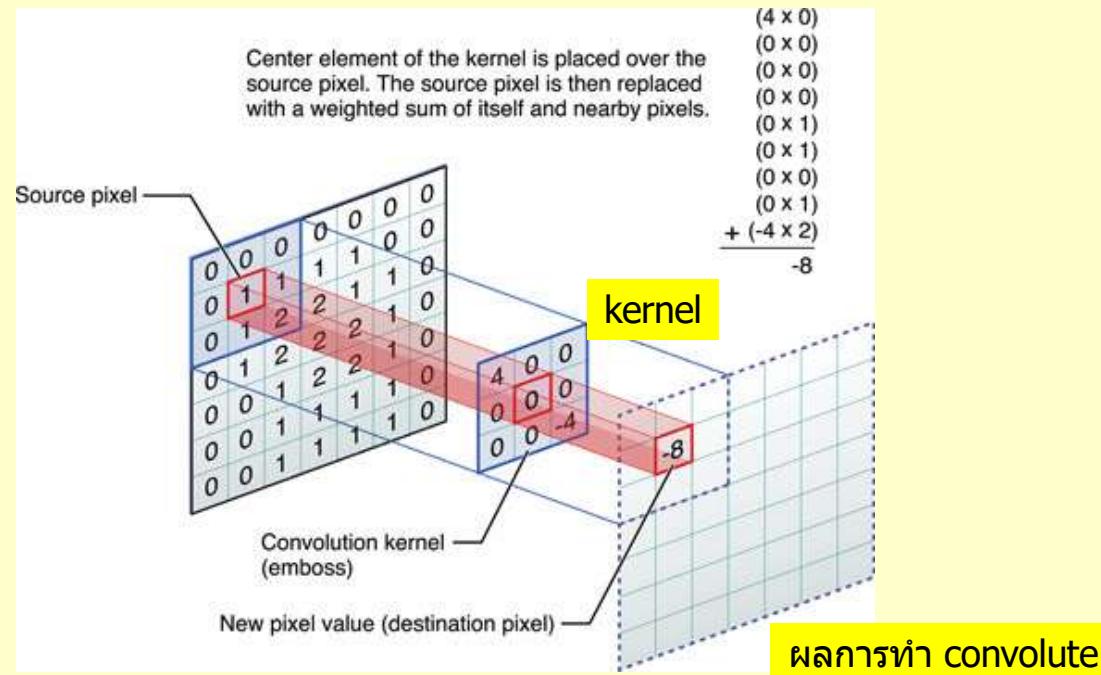


สีเทา เข้าว่าสูตรนี้ดีกว่า $0.299*R + 0.587*G + 0.114*B$

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



Convolution



เปลี่ยนค่าของเมทริกซ์ kernel
จะได้การประมวลผลภาพแบบอื่น ๆ

Image Convolution

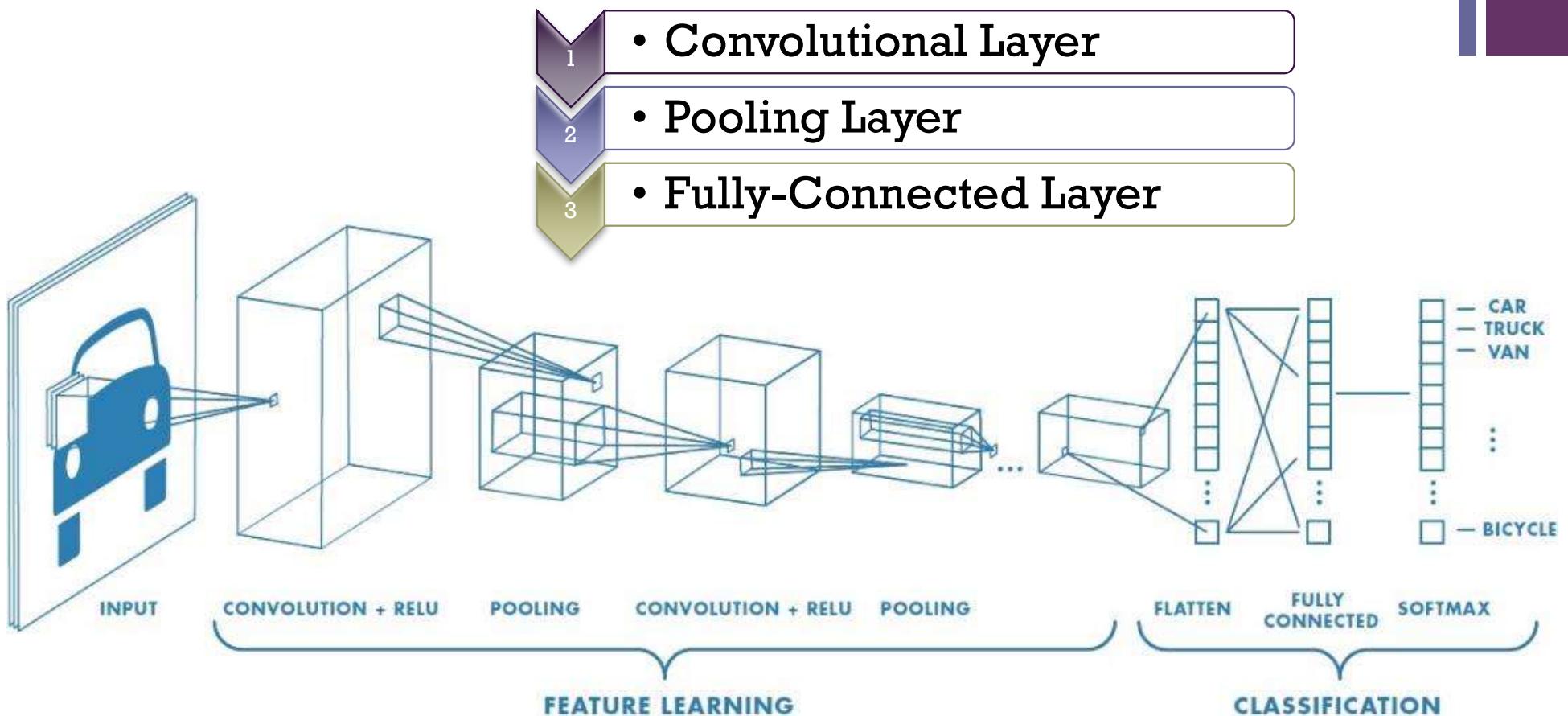
- Image Convolution คือการนำรูปภาพมาผ่านตัวกรอง (kernel) เพื่อให้ได้ผลลัพธ์ตามที่ต้องการ เช่น blur, ขยายรูป, จับขอบรูป เป็นต้น

$$\text{Original} \quad * \quad \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \text{Blur (with a mean filter)}$$

$$\text{Original} \quad * \quad \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 1 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} = \text{Shifted left
By 1 pixel}$$

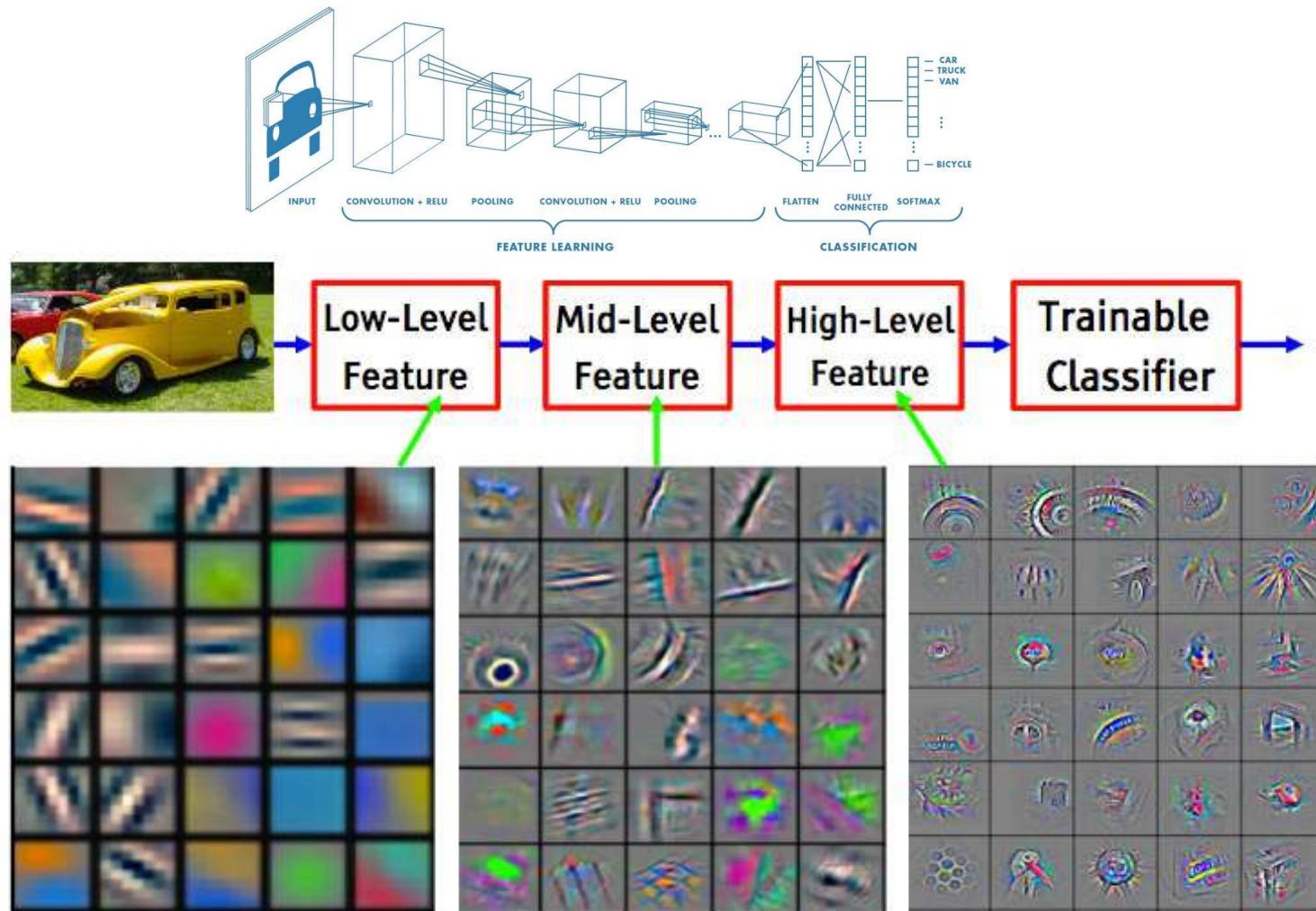


Convolutional Neural Networks (CNN)





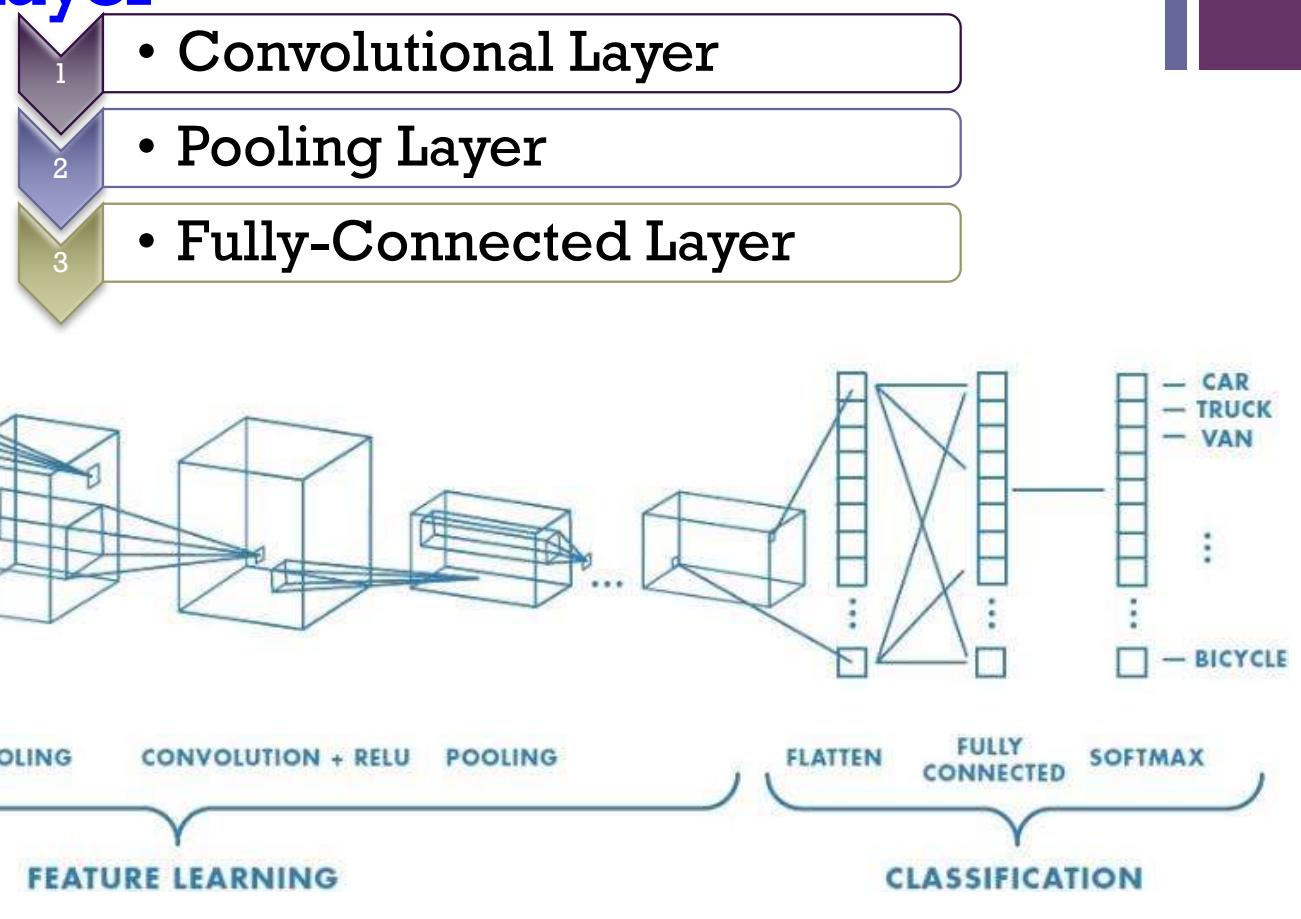
CNN (cont.)





Convolutional Neural Networks (CNN)

1) Convolutional Layer





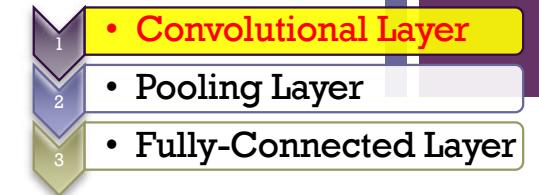
Layer1: Convolutional Layer

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

(3*3 filter)

1	0	1
0	1	0
1	0	1



4		

Convolved
Feature

x[::,::,0]	0 0 0 0 0 0 0	0 1 0 2 2 2 0	0 0 0 0 0 0 0	0 2 0 2 2 2 0	0 1 0 0 0 0 0	0 1 0 0 2 1 0	0 0 0 0 0 0 0
x[::,::,1]	0 0 0 0 0 0 0	0 1 0 1 0 1 0	0 0 0 0 0 1 0	0 0 0 2 0 0 0	0 2 0 0 0 0 0	0 0 0 1 0 0 0	0 0 0 0 0 0 0
x[::,::,2]	0 0 0 0 0 0 0	0 0 2 2 0 0 0	0 0 0 0 0 1 0	0 1 1 2 0 2 0	0 2 0 0 0 0 0	0 0 1 1 0 1 0	0 0 0 0 0 0 0

Filter W0 (3x3x3)

w0[::,:,0]	1 0 0	-1 0 0	0 -1 1	-1 -1 0	0 -1 1	1 0 0	
w0[::,:,1]							
w0[::,:,2]							
Bias b0 (1x1x1)	1						

Filter W1 (3x3x3)

w1[::,:,0]	1 -1 0	-1 0 0	0 1 -1	0 1 -1	0 0 0	0 1 1	0 -1 0
w1[::,:,1]							
w1[::,:,2]							

Output Volume (3x3x2)

o[::,:,0]	2 -2 -1	2 -3 -3	2 -1 -2	0 1 -1	-2 4 -1	3 3 0	-2 2 -1
o[::,:,1]							

$$\begin{aligned}
 o(2,2,0) &= \sum x[::,0] \times w[::,0] + \sum x[::,1] \times w[::,1] + \sum x[::,2] \times w[::,2] + b_0 \\
 &= 0 \times 1 + 0 \times 0 + 0 \times 0 + 2 \times (-1) + 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times (-1) + 0 \times 1 \\
 &\quad + 0 \times (-1) + 0 \times (-1) + 0 \times 0 + 0 \times (-1) + 0 \times (-1) + 0 \times 1 + 0 \times 1 + 0 \times 0 + 0 \times 1 \\
 &\quad + 0 \times 1 + 0 \times 1 + 0 \times 0 + 0 \times 0 + 1 \times (-1) + 0 \times 1 + 0 \times 1 + 0 \times 1 + 0 \times 1 \\
 &\quad + 1 \\
 &= -2
 \end{aligned}$$

CNNs: 1) Convolutional Layer

Feature Extraction

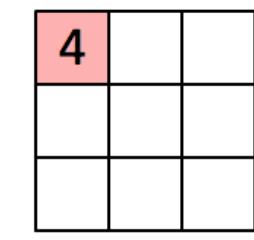
<http://cs231n.github.io/convolutional-networks/>

(3*3 filter)

$$w^T x + b$$

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

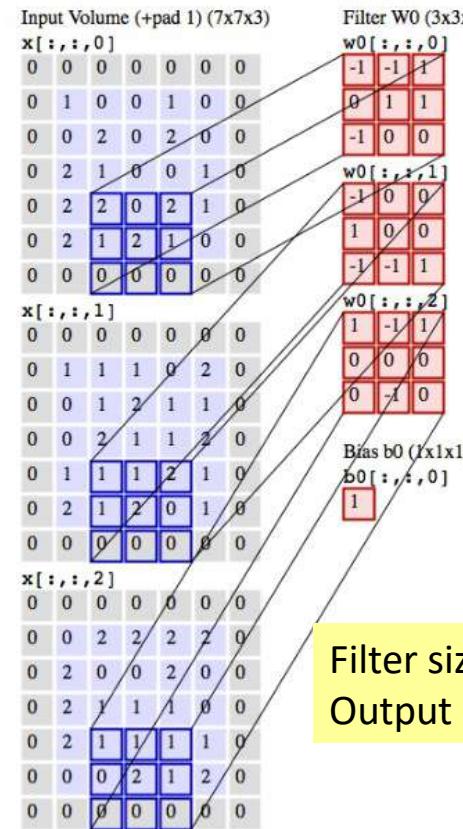


Convolved Feature
(feature map)

Filter size=(3*3), #Filters=1, Stride=1, Padding=0

Output size = $(5 + 2(0) - 3) / 1 + 1 = 3$

1	0	1
0	1	0
1	0	1



$$\text{Output size} = (N + 2*P - F) / S + 1$$

(feature map1)

(feature map2)

$$\begin{aligned}
 R: & 2(-1)+2(1)+2(1)+1(1) = 3 \\
 G: & 1(-1)+1(1) = 0 \\
 B: & 1(1)+1(-1)+1(1) = 1 \\
 \text{Ans:} & 4+1 = 5
 \end{aligned}$$

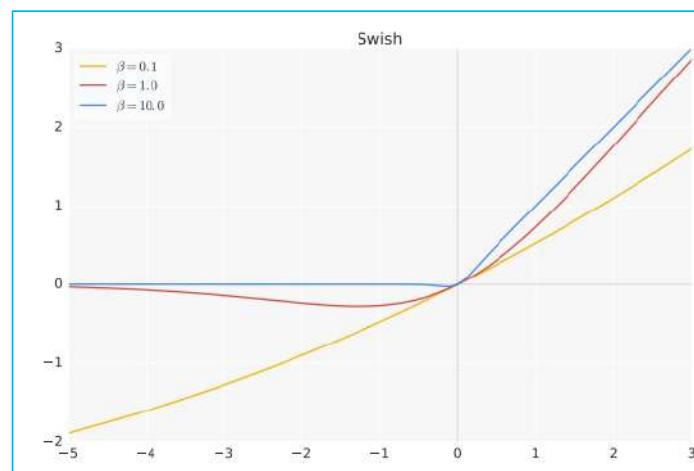
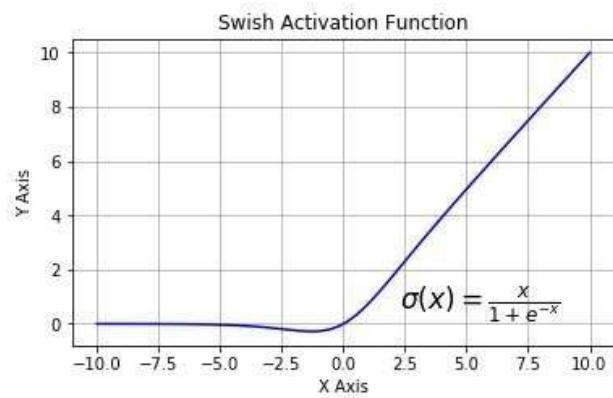
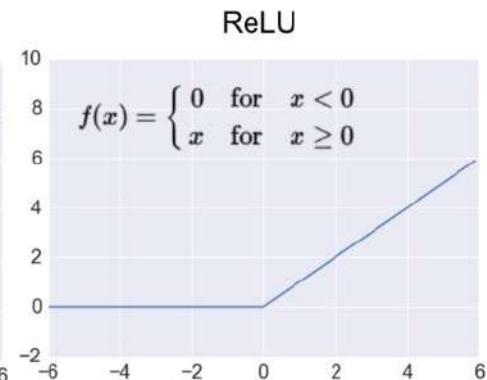
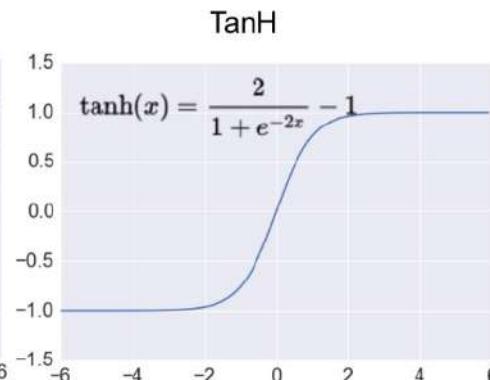
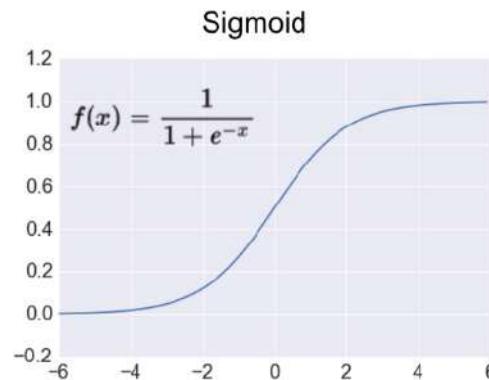
toggle movement

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

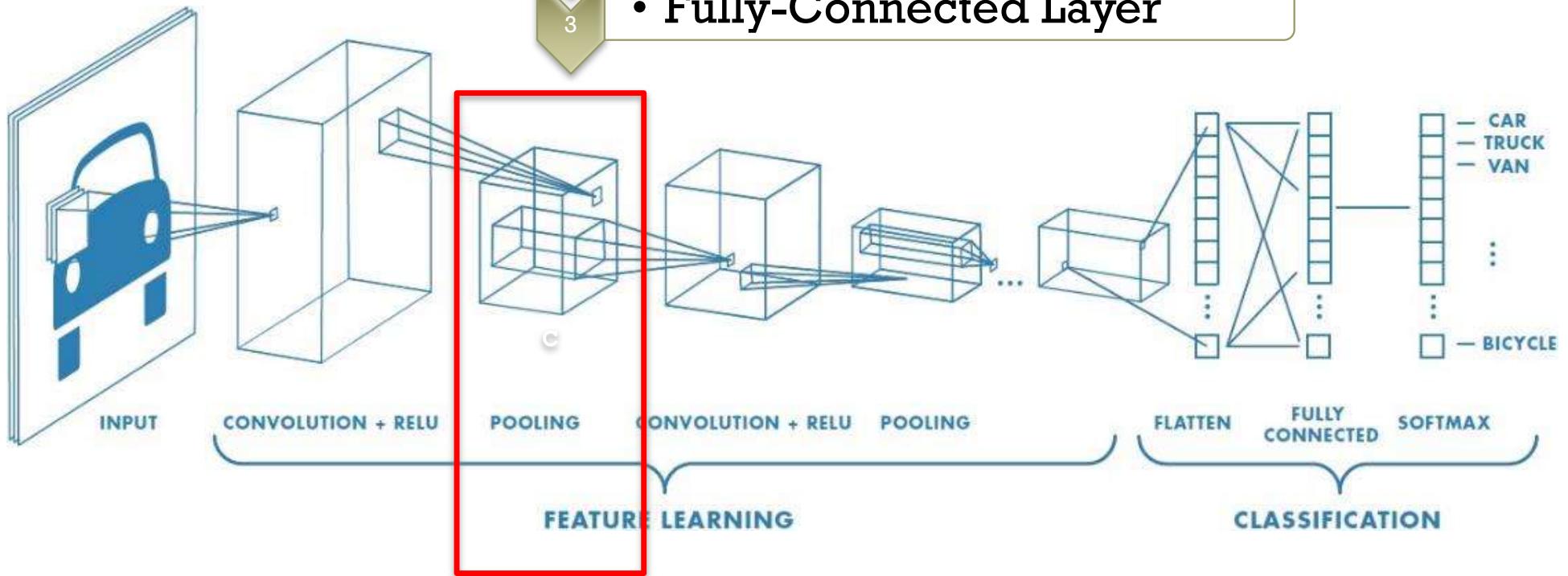




Convolutional Neural Networks (CNN)

2) Pooling Layer

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer

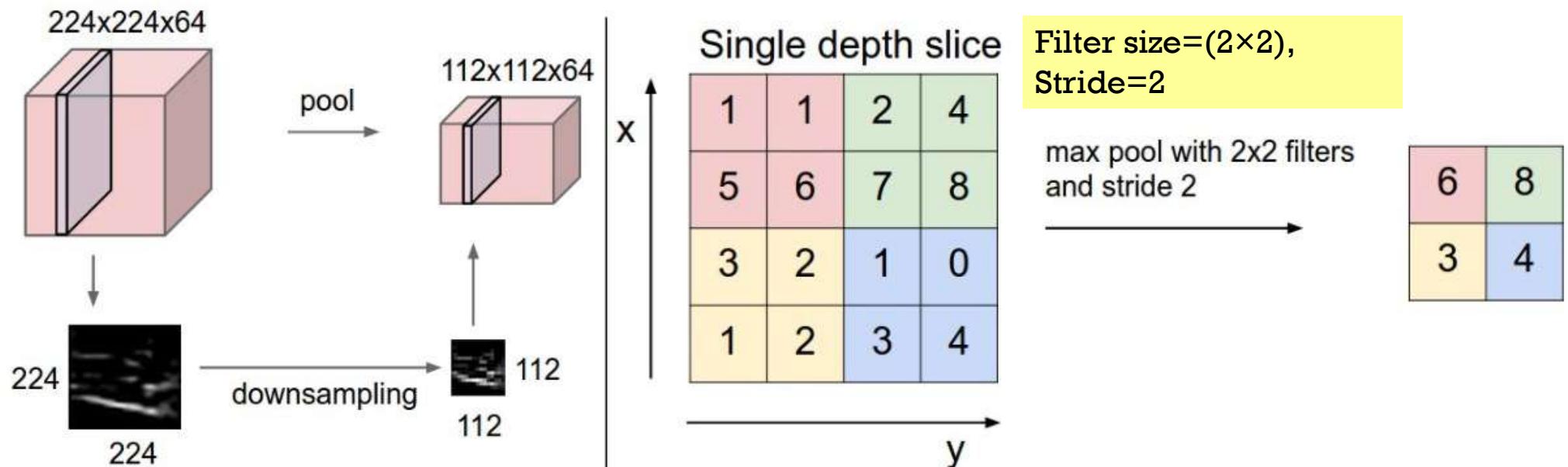


CNNs: 2) Pooling Layer

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

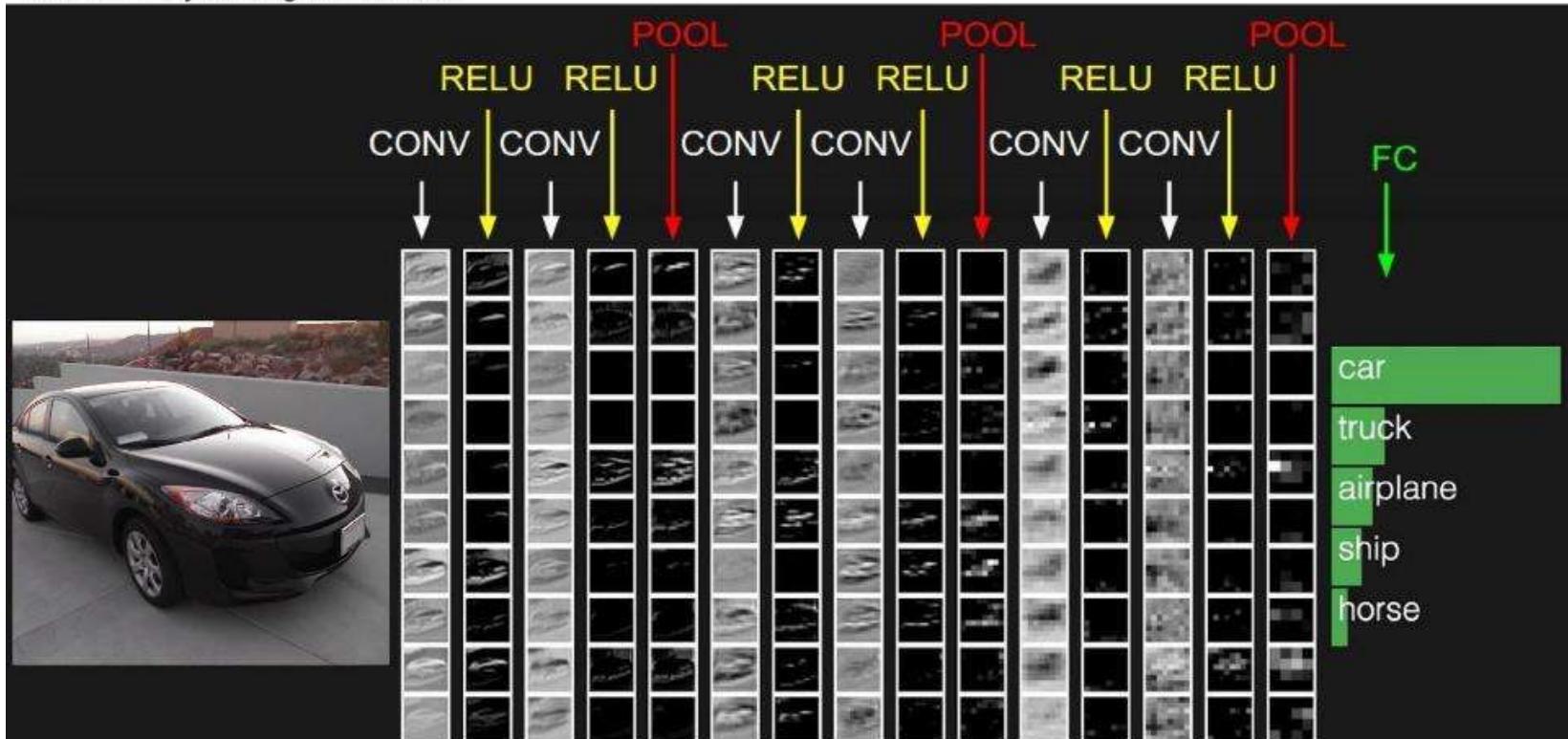
Summary: **2 parameters** for pooling layer
• Filter size, Stride

- Feature selection (reduction); downsampling





two more layers to go: POOL/FC

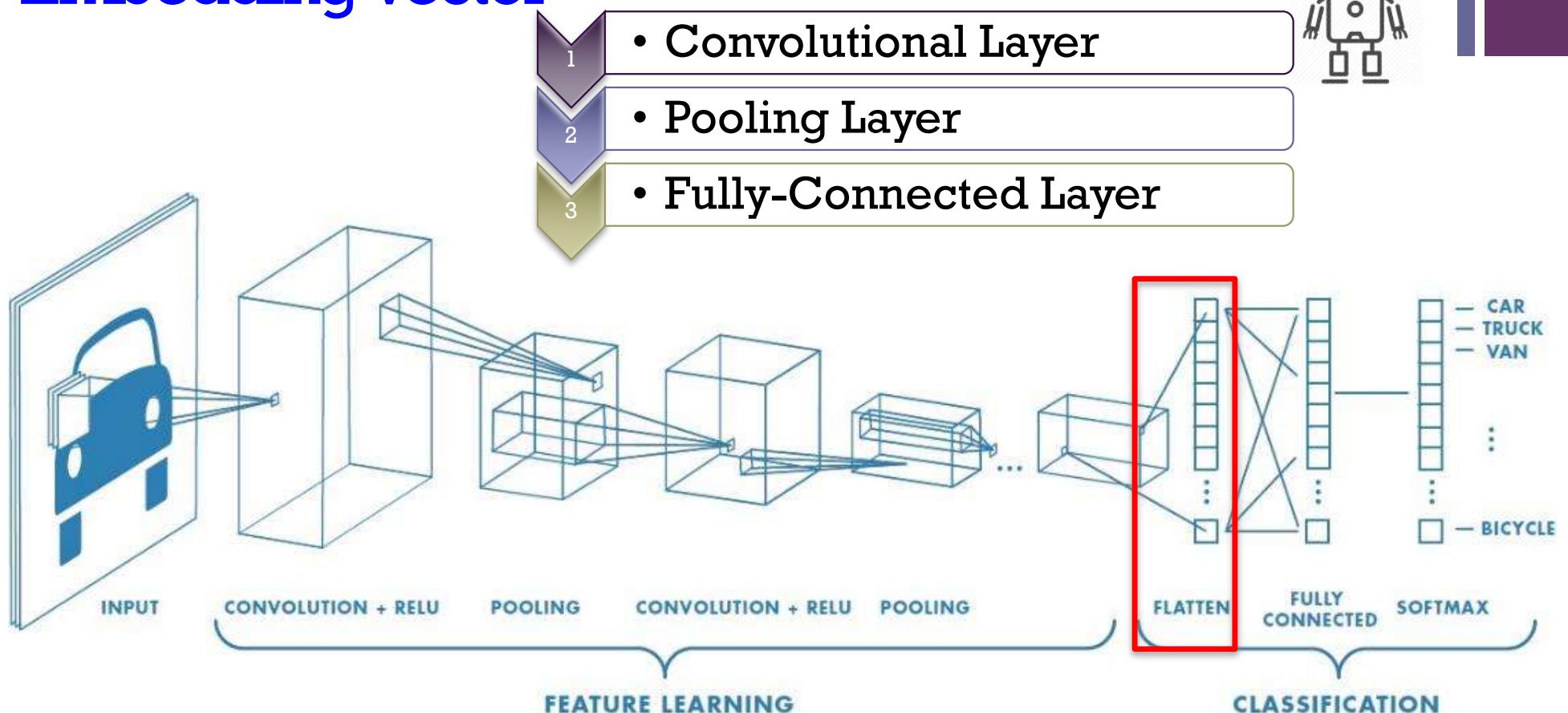




Convolutional Neural Networks

Embedding Vector

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

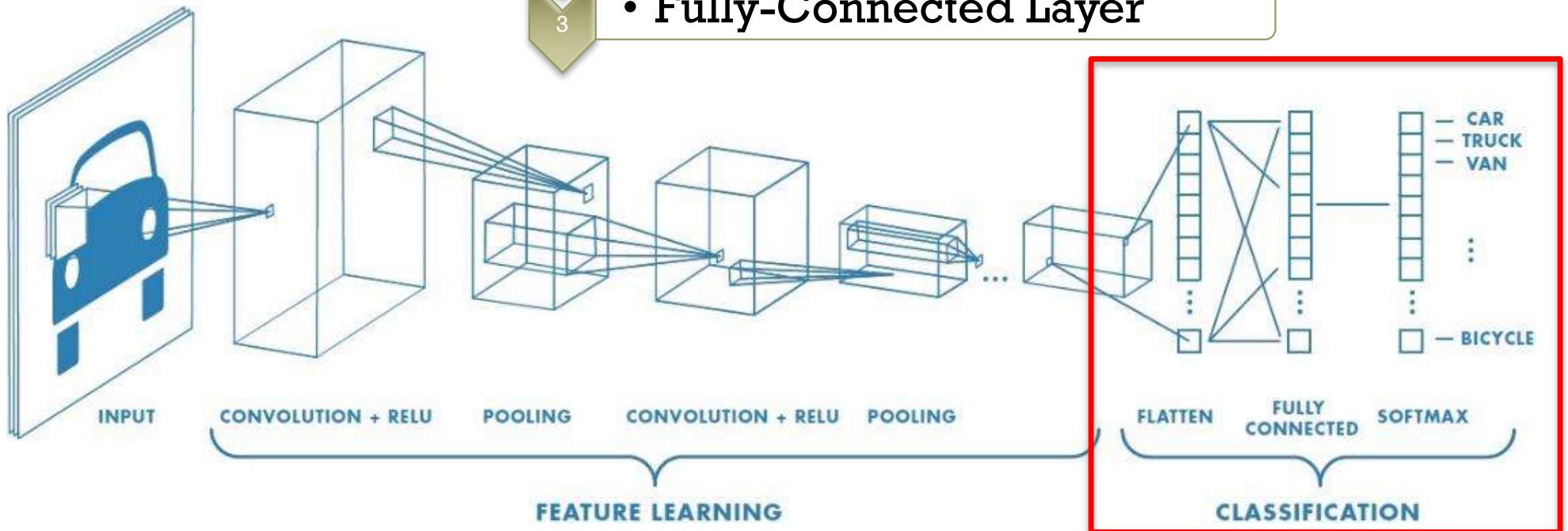




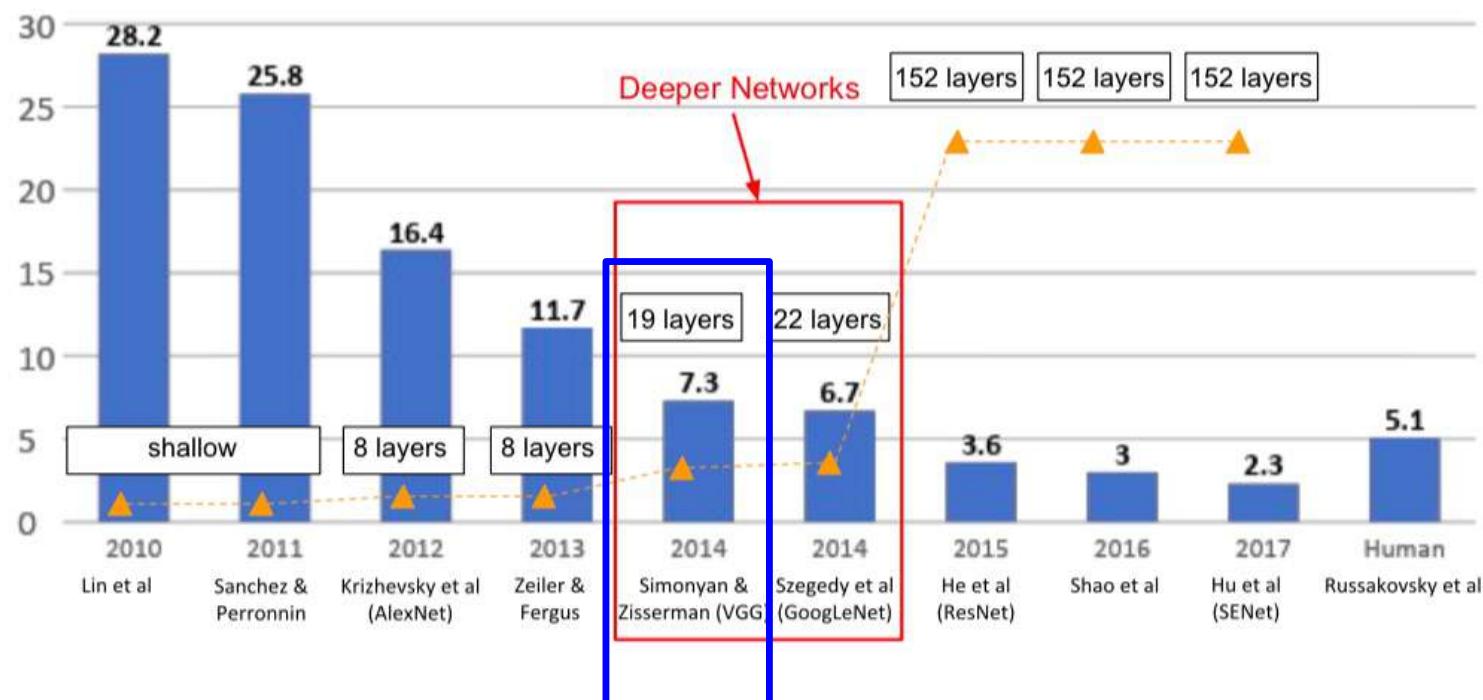
Convolutional Neural Networks (CNN)

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

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer



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



Case Study: VGGNet (2014)

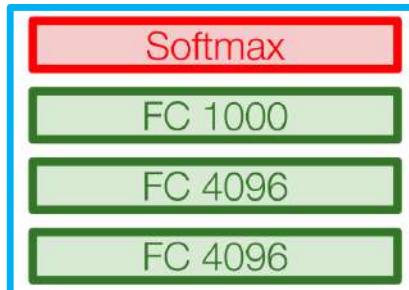
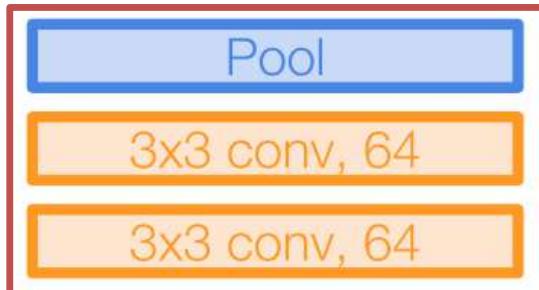
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

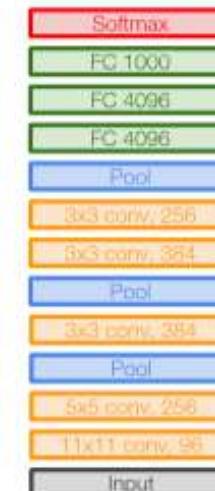
8 layers (AlexNet) → 16-19 layers (VGG16Net)

Only 3x3 CONV Stride 1, pad 1

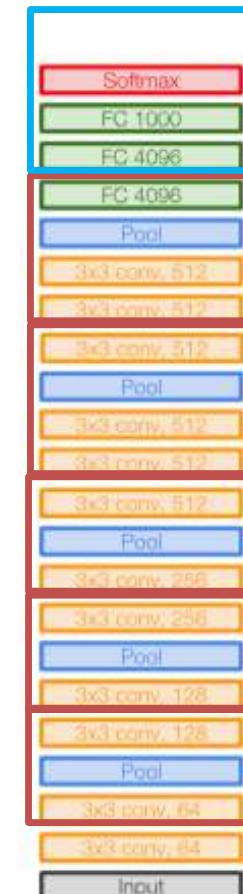
And 2x2 MAX POOL stride 2



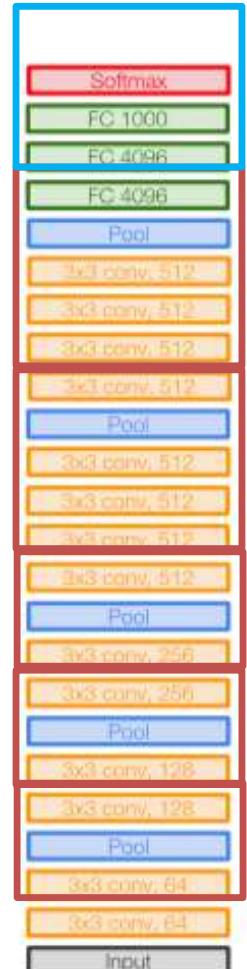
1) Feature extraction block 2) Classification block



AlexNet



VGG16



VGG19

Case Study: VGGNet (2014)

[Simonyan and Zisserman, 2014]

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

Stack of three 3x3 conv (stride 1)
layers has same **effective receptive
field** as one 7x7 conv layer in ZFNet

But deeper, more non-linearities

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



K 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-4096	FC-1000		
soft-max					

VGG16 VGG19

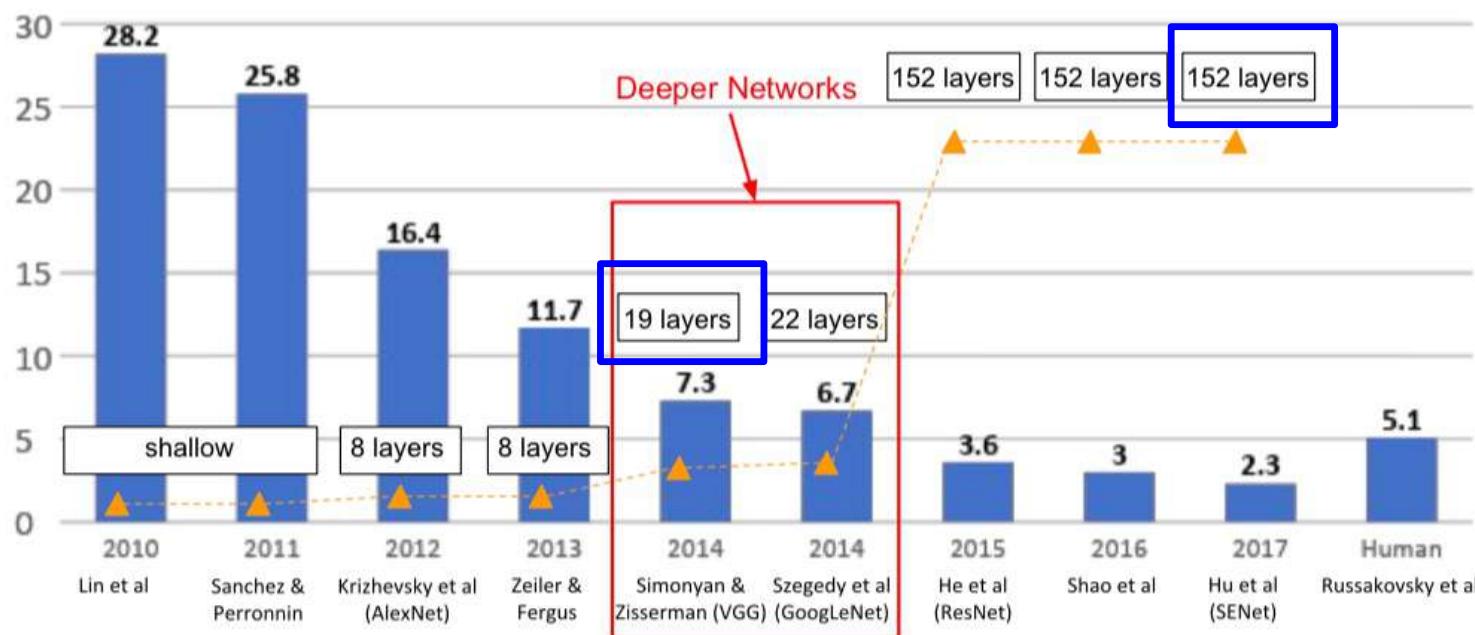
Calculate the number of parameters on VGG Net

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 conv1-256	conv3-256 conv3-256 conv3-256
maxpool					

Network	A	B	C	D	E
Number of parameters (Millions)	133	133	134	138	144
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

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

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



- Overview
- Input
- Model
- Sequential
- activations
- applications
 - Overview
 - DenseNet121
 - DenseNet169
 - DenseNet201
 - EfficientNetB0
 - EfficientNetB1
 - EfficientNetB2
 - EfficientNetB3
 - EfficientNetB4
 - EfficientNetB5
 - EfficientNetB6
 - EfficientNetB7
 - InceptionResNetV2
 - InceptionV3
 - MobileNet
 - MobileNetV2
 - MobileNetV3Large
 - MobileNetV3Small
 - NASNetLarge
 - NASNetMobile

Modules

[densenet](#) module: DenseNet models for Keras.

[efficientnet](#) module: EfficientNet models for Keras.

[imagenet_utils](#) module: Utilities for ImageNet data preprocessing & prediction decoding.

[inception_resnet_v2](#) module: Inception-ResNet V2 model for Keras.

[inception_v3](#) module: Inception V3 model for Keras.

[mobilenet](#) module: MobileNet v1 models for Keras.

[mobilenet_v2](#) module: MobileNet v2 models for Keras.

[nasnet](#) module: NASNet-A models for Keras.

[resnet](#) module: ResNet models for Keras.

[resnet50](#) module: Public API for tf.keras.applications.resnet50 namespace.

[resnet_v2](#) module: ResNet v2 models for Keras.

[vgg16](#) module: VGG16 model for Keras.

[vgg19](#) module: VGG19 model for Keras.

Model 1

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

```
1 base_model = VGG19(weights=None, include_top=False, input_shape=(224, 224, 3))
2
3 for layer in base_model.layers:
4     layer.trainable = True
5
6 x = base_model.output
7 x = Flatten()(x)
8 x = Dense(1024)(x)
9 x = Dropout(0.5)(x)
10 x = Dense(512)(x)
11 x = Dropout(0.5)(x)
12 output = Dense(num_class, activation='softmax')(x)
13
```

TORCHVISION.MODELS

The models subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.

Classification

The models subpackage contains definitions for the following model architectures for image classification:

- [AlexNet](#)
- [VGG](#)
- [ResNet](#)
- [SqueezeNet](#)
- [DenseNet](#)
- [Inception v3](#)
- [GoogLeNet](#)
- [ShuffleNet v2](#)
- [MobileNetV2](#)
- [MobileNetV3](#)
- [ResNeXt](#)
- [Wide ResNet](#)
- [MNASNet](#)

We provide pre-trained models, using the PyTorch `torch.utils.model_zoo`. These can be constructed by passing `pretrained=True`:

```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeeze1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)
mobilenet_v2 = models.mobilenet_v2(pretrained=True)
mobilenet_v3_large = models.mobilenet_v3_large(pretrained=True)
mobilenet_v3_small = models.mobilenet_v3_small(pretrained=True)
resnext50_32x4d = models.resnext50_32x4d(pretrained=True)
wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
mnasnet = models.mnasnet1_0(pretrained=True)
```



v1.1.0 ▾

[pytorch-transformers](#)[Star 131,473](#)[Search docs](#)

NOTES

[Installation](#)[Quickstart](#)[Pretrained models](#)[Examples](#)[Notebooks](#)[Loading Google AI or OpenAI pre-trained weights or PyTorch dump](#)[Serialization best-practices](#)[Converting Tensorflow Checkpoints](#)[Migrating from pytorch-pretrained-bert](#)[BERTology](#)

ⓘ You are viewing legacy docs. Go to [latest documentation](#) instead.

[Docs » Pytorch-Transformers](#)

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 Luan 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*, Zhen 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

Classification



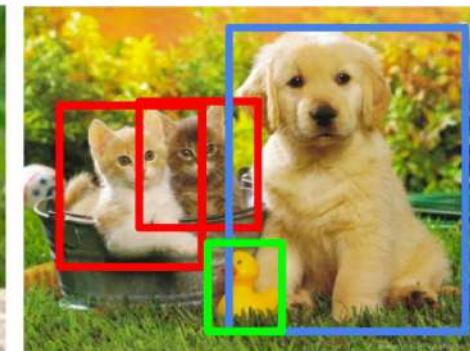
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation



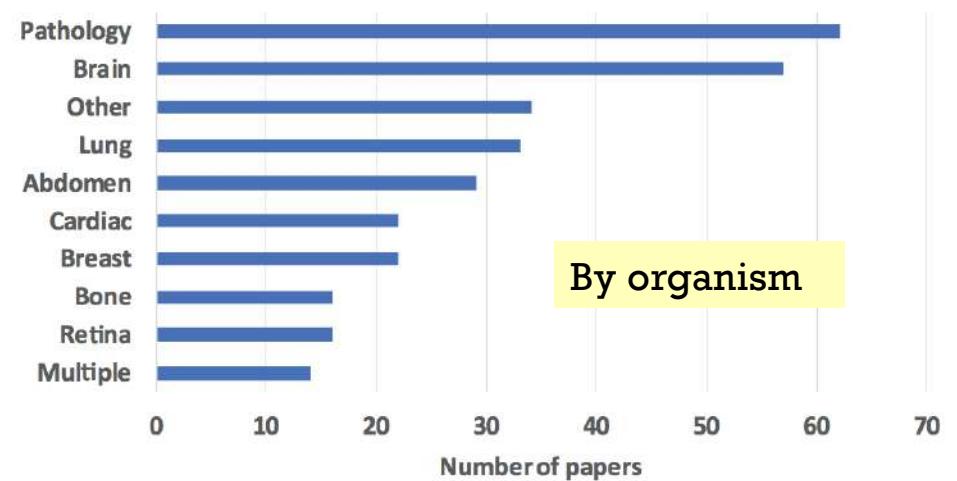
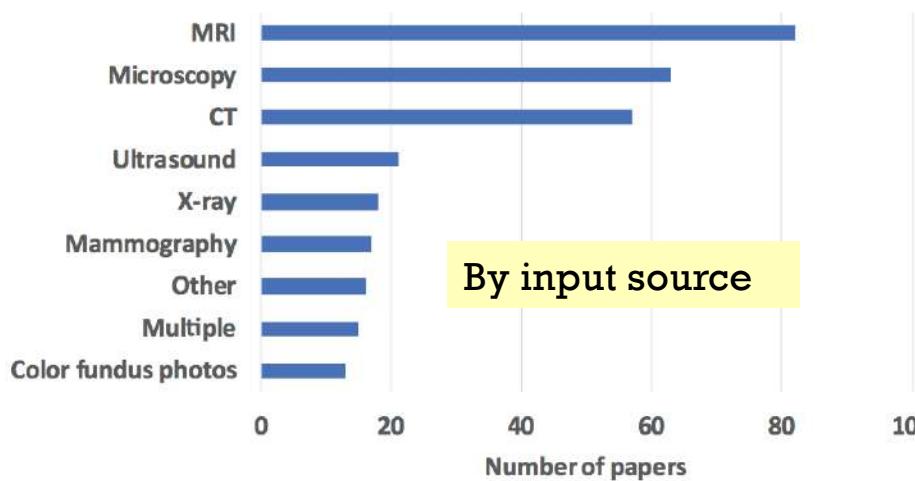
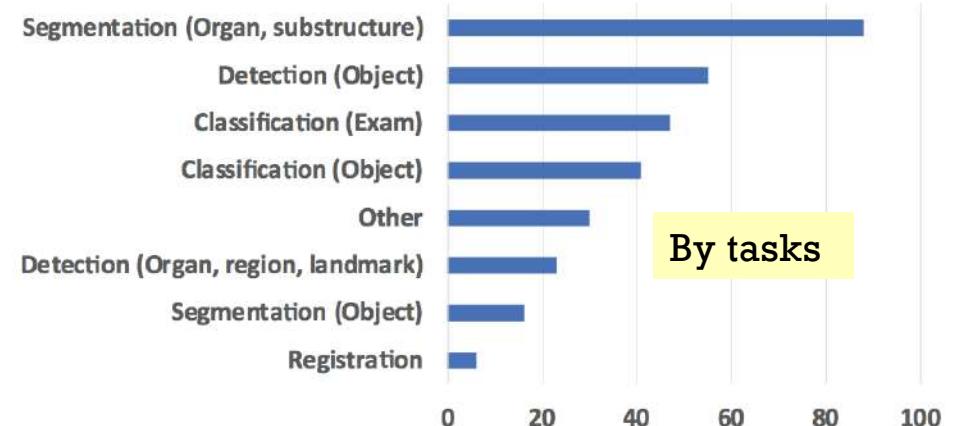
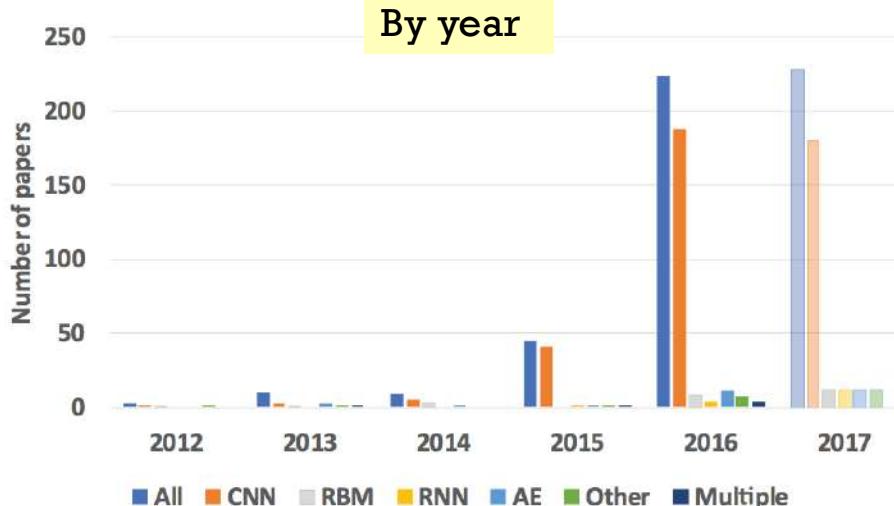
CAT, DOG, DUCK

Single object

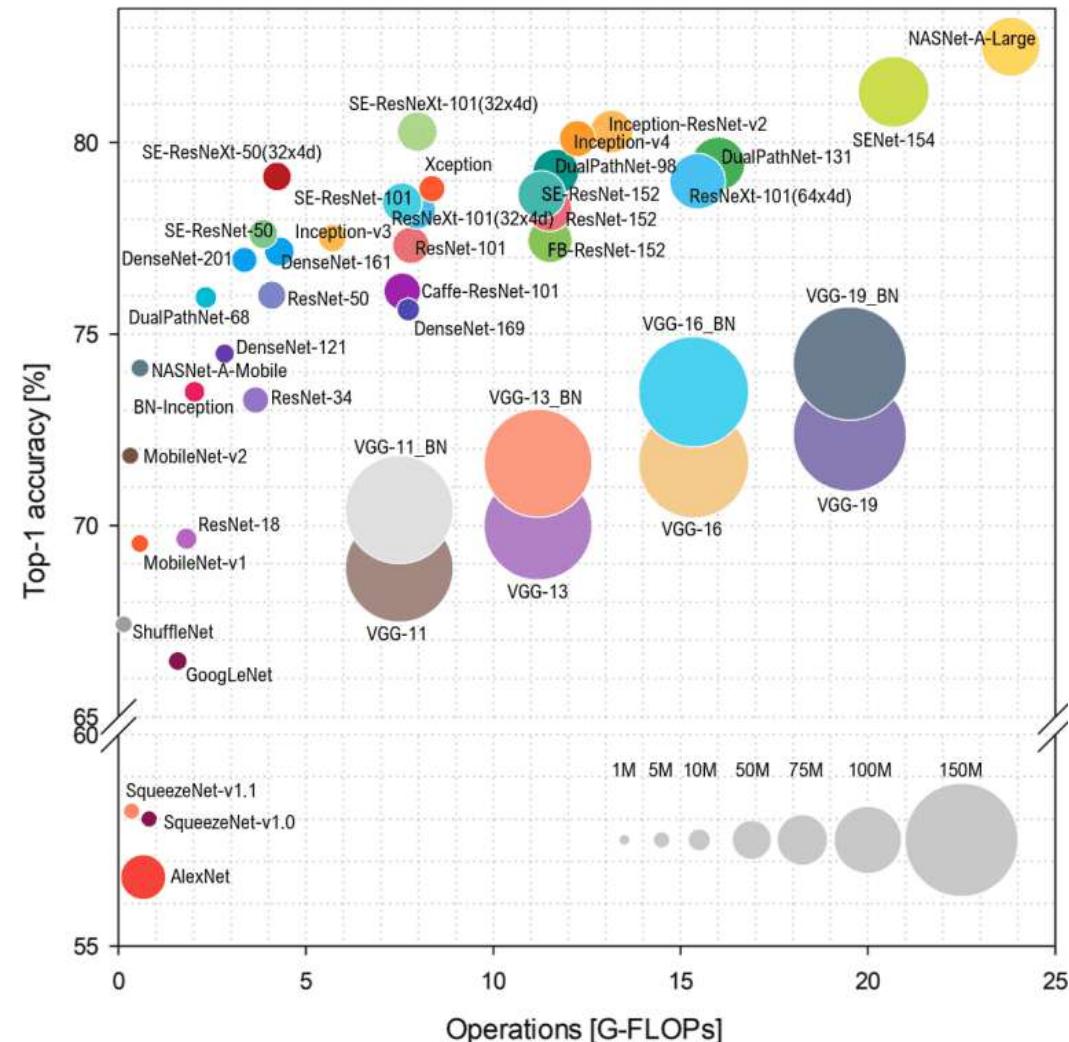
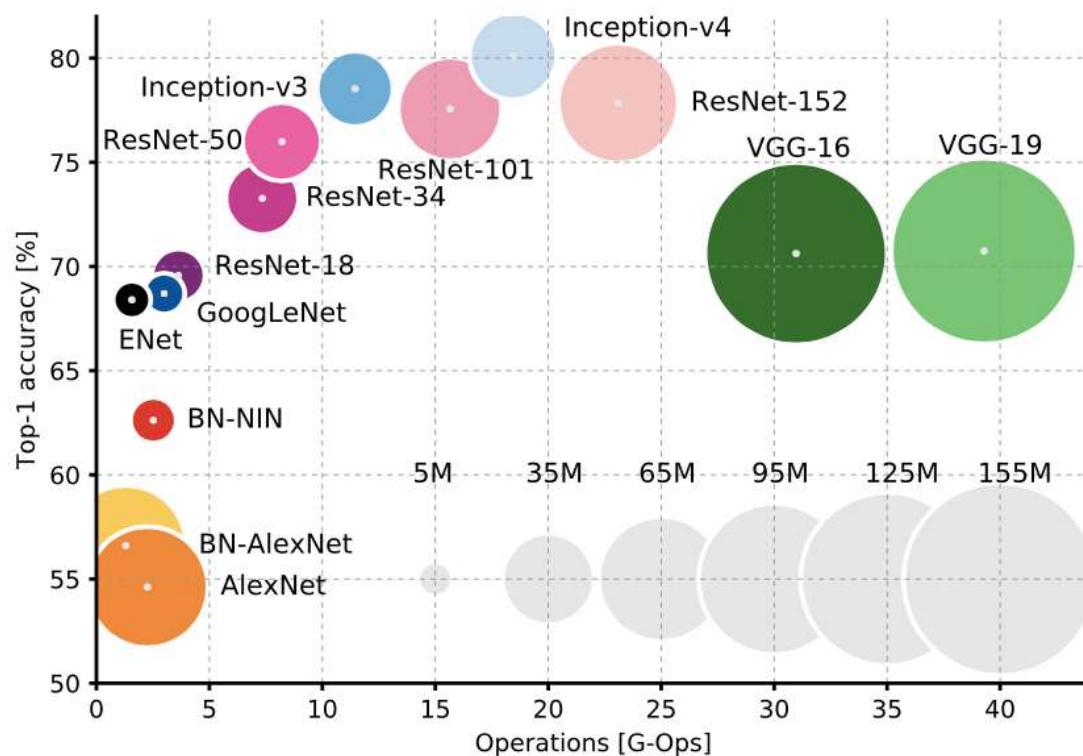
Multiple objects



Deep Learning in Medical Image Analysis [arXiv 2017]



SOTA of Image Classification



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

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

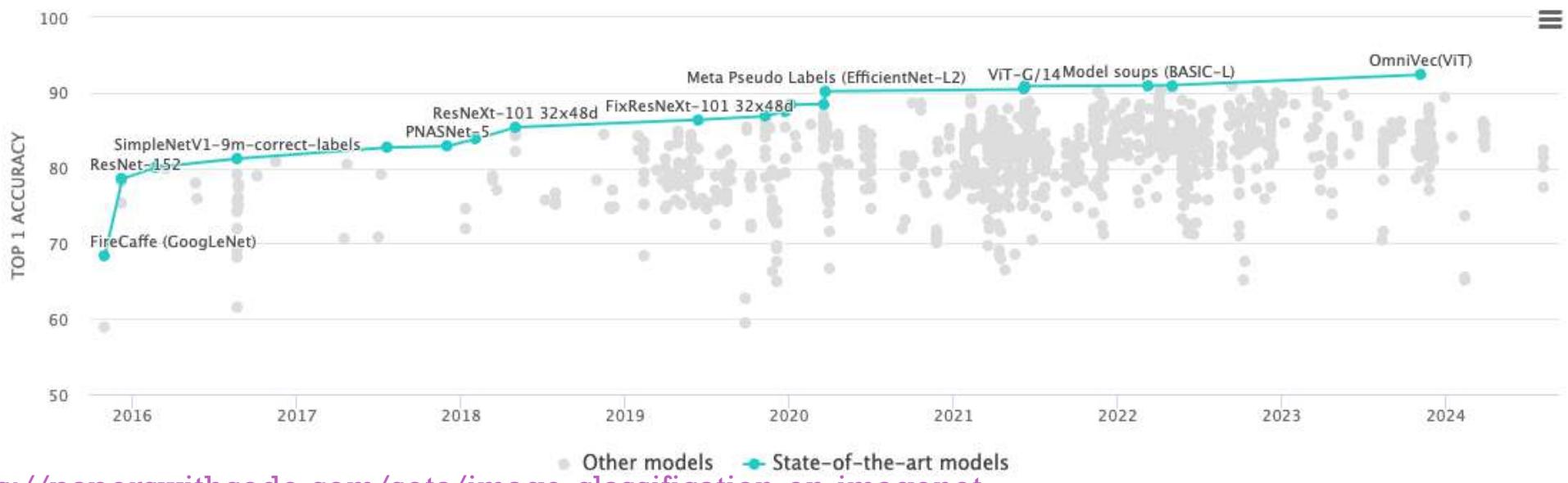


Image classification dashboard (2024)

Image Classification on ImageNet

[Leaderboard](#)[Dataset](#)

View Top 1 Accuracy by Date for All models



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



Image classification dashboard (2025)

■ CNN, Transformer (ViT, VLM)

- 2021–2022 → CNNs (EfficientNetV2) were strong, but Transformers started dominating.
- 2022–2023 → Vision–Language models (VLM) like CoCa + giant ViTs (EVA, Model Soups) pushed past 90%.
- 2024 → Specialized ViT variants (DaViT, Florence-CoSwin) consolidated.
- 2025 → Large-scale foundation models (DINOv3) + fine-tuned CoCa remain at the frontier.

■ Note:

- CoCa / CLIP → **Vision–Language**, trained with image–text pairs, suitable for cross-modal tasks such as captioning and retrieval
- DINOv3 → **Vision-only**, self-supervised ViT, backbone for computer vision

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



EfficientNet (May, 2019) → EffNetV2 (2021)

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan¹ Quoc V. Le¹

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

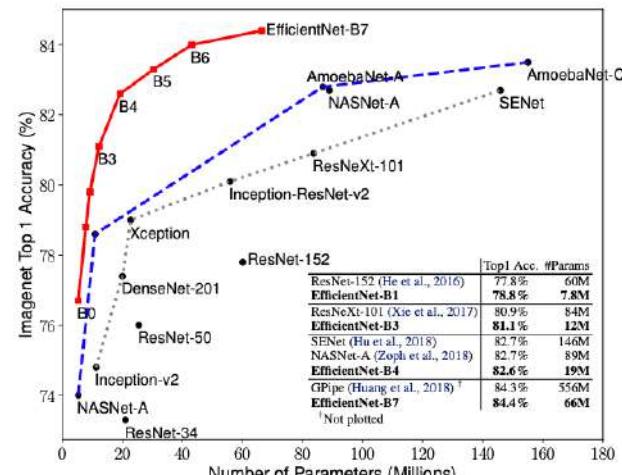
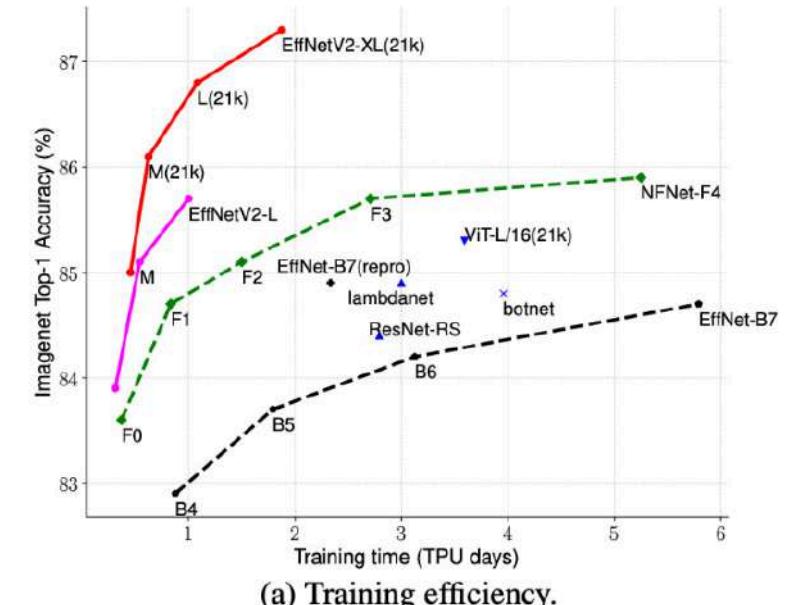


Figure 1: Model Size vs. ImageNet Accuracy. All numbers are



	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>

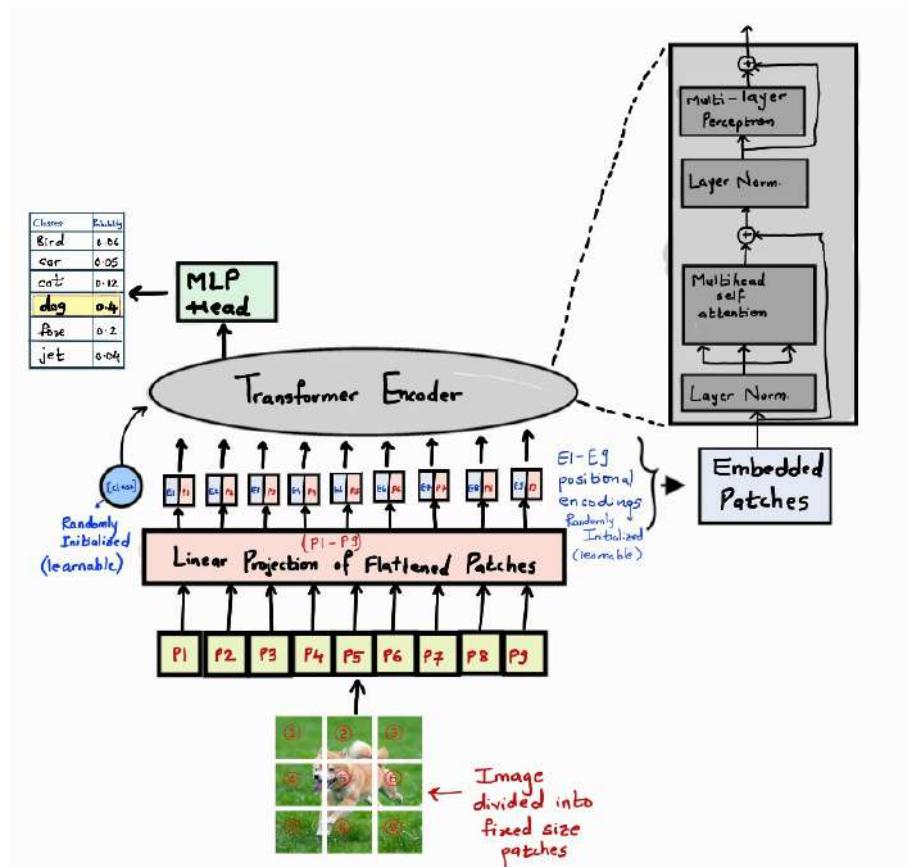
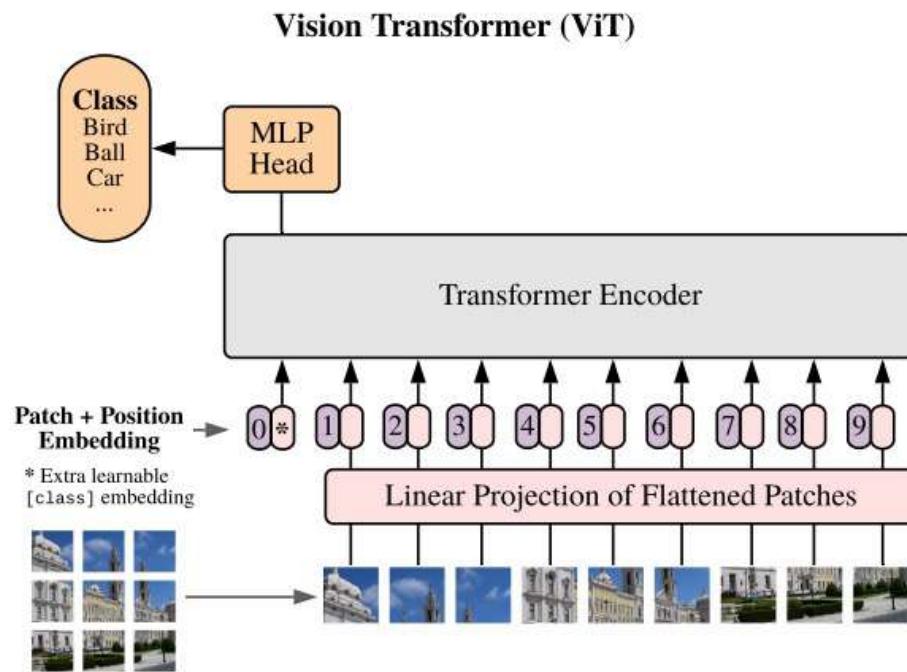
<https://twitter.com/TensorFlow/status/1423359562724175873/photo/1>



Vision Transformer (ViT) (ICLR2021)

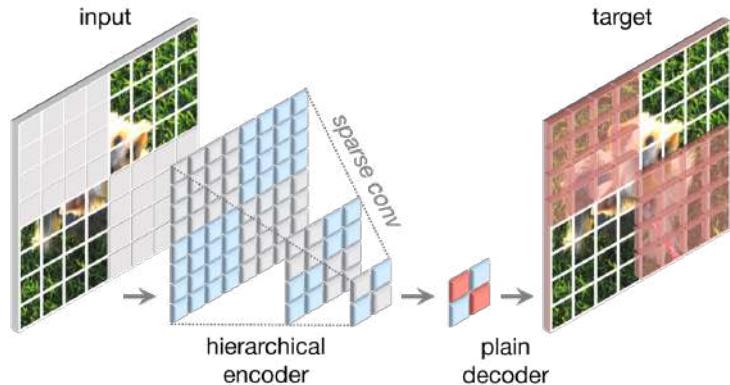
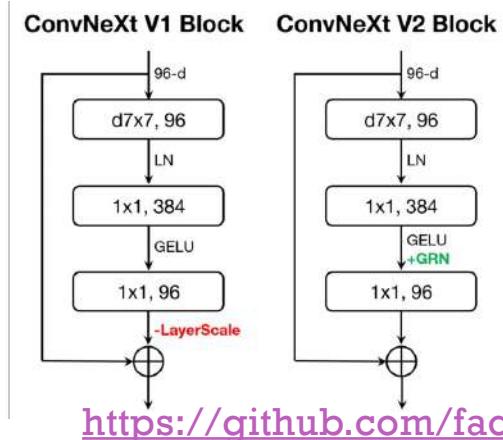
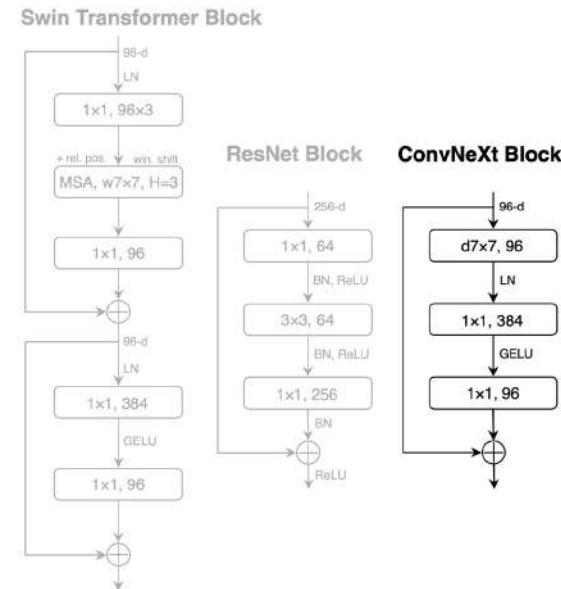
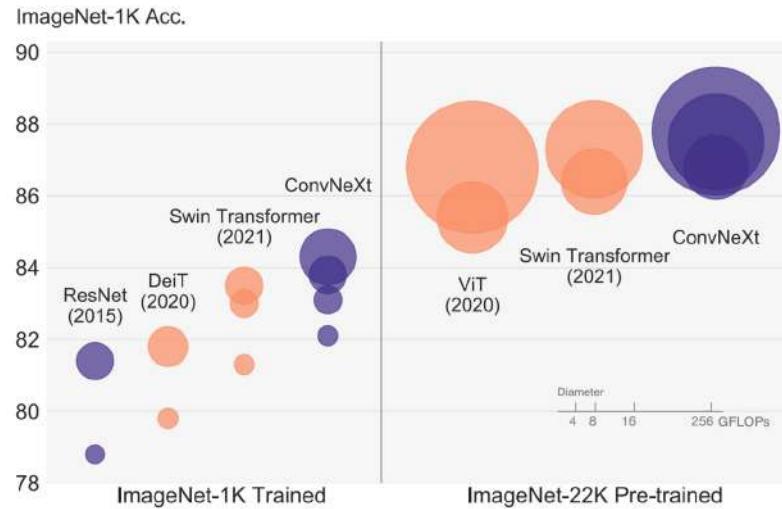


Hugging Face



<https://medium.com/machine-intelligence-and-deep-learning-lab/vit-vision-transformer-cc56c8071a20>

+ ConvNeXt (CVPR2022) → ConvNeXt V2 (2023)



+

DINOv3: Self-supervised learning for vision at unprecedented scale

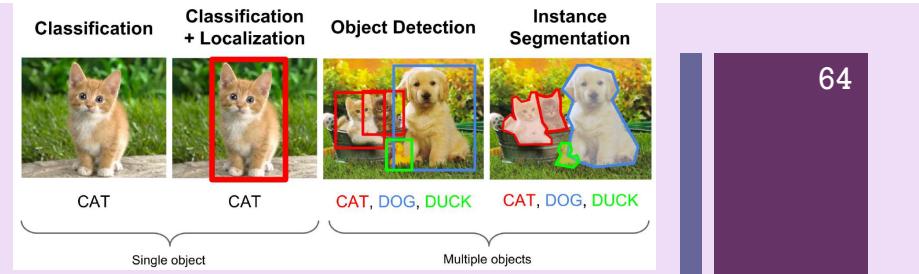
August 14, 2025 • 11 minute read

- DINOv3: Self-Supervised Vision Transformer (Meta AI, 2025)
- Vision-only (**encoder-only**) backbone (not multimodal, unlike CLIP/CoCa).
- Self-supervised learning → no human labels needed.
- **Massive scale: up to 7B parameters, trained on 1.7B images.**
- Key innovations:
 - Gram Anchoring → stabilizes feature learning.
 - Axial RoPE → improved positional encoding.
 - Resolution robustness → better handling of varied image sizes.
- Strengths:
 - Provides general-purpose visual representations.
 - **Excels in downstream tasks (classification, detection, segmentation).**
 - Foundation model role for vision, similar to LLMs in NLP.

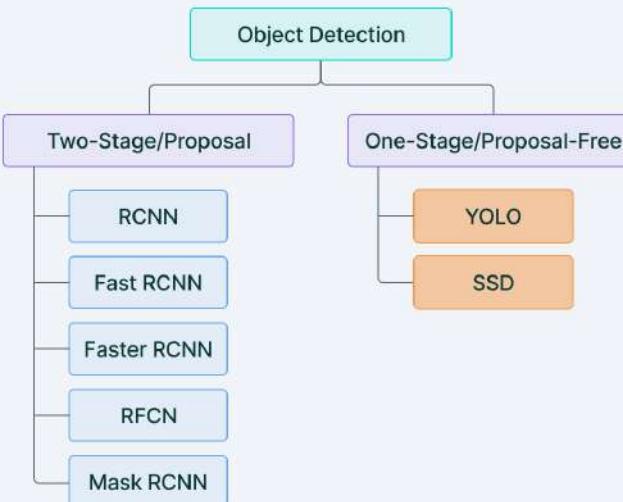
[Link](#)



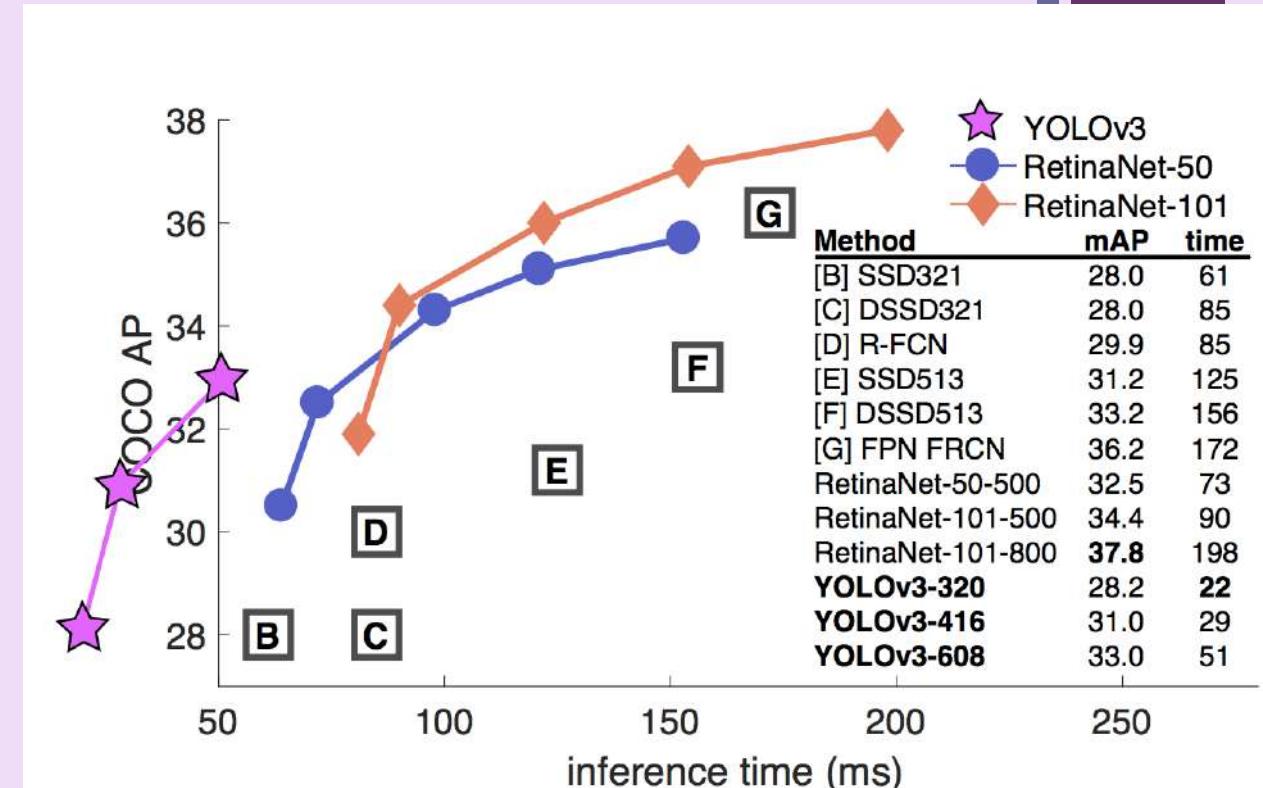
SOTA of Object Detection



One and two stage detectors



V7 Labs



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



YOLO Open Images in New York





YOLO's output



1. Resize image.
2. Run convolutional network.
3. Non-max suppression.

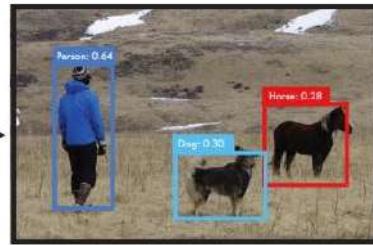
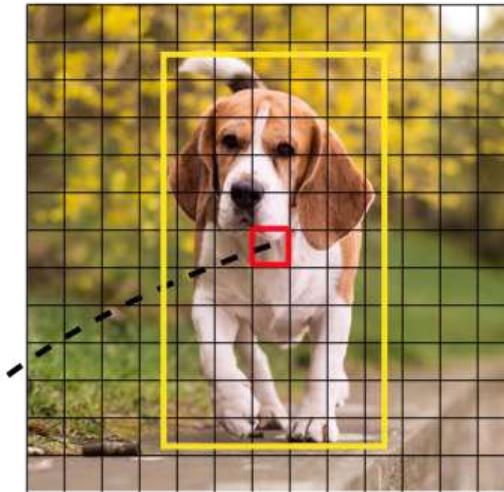


Image Grid. The Red Grid is responsible for detecting the dog



66

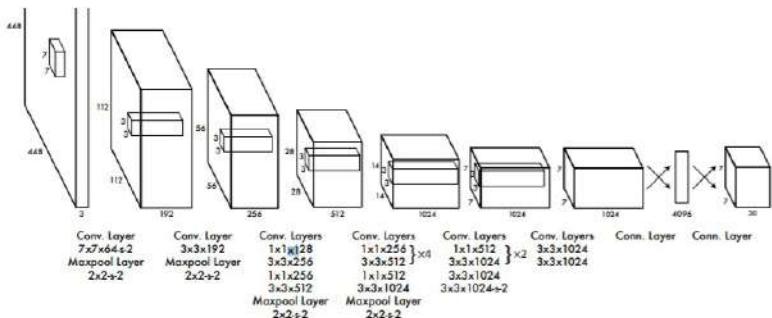
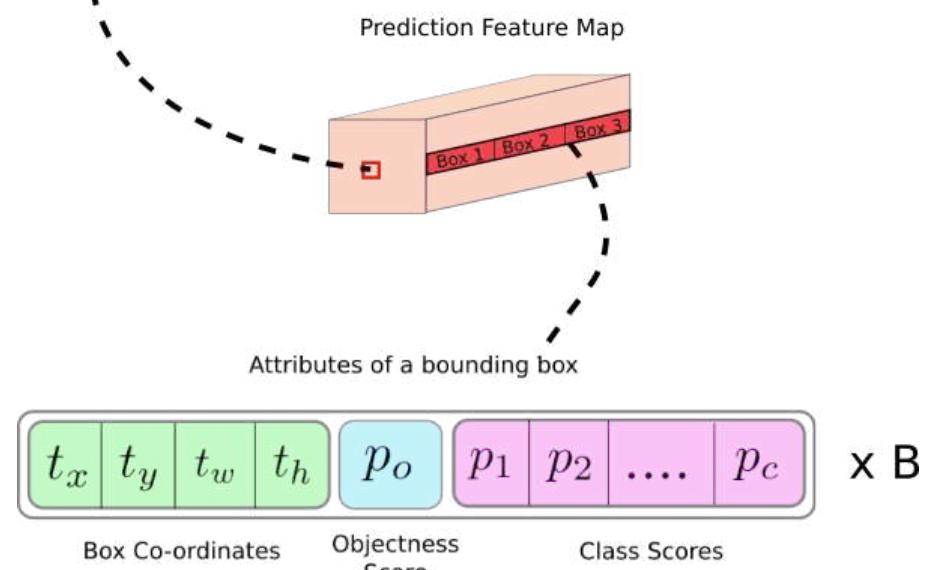


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224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

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



Ultralytics YOLO → production-ready, practical framework (YOLOv5 → YOLOv8 → YOLO11).
 Academic YOLO → new research architectures (YOLOv9, v10, v11, v12).



<https://blog.roboflow.com/guide-to-yolo-models/>





Ultralytic's YOLO

YOLOv5 (2020) → YOLOv8 (2023) → Ultralytics YOLO 11 (2024)

68



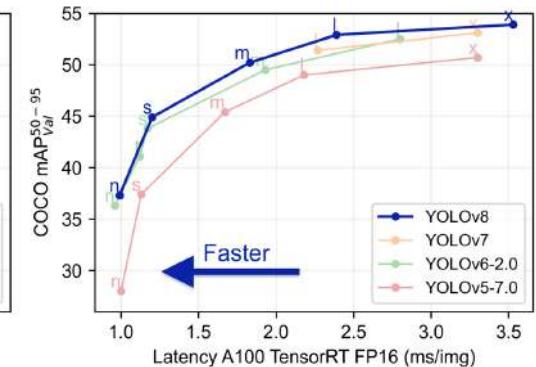
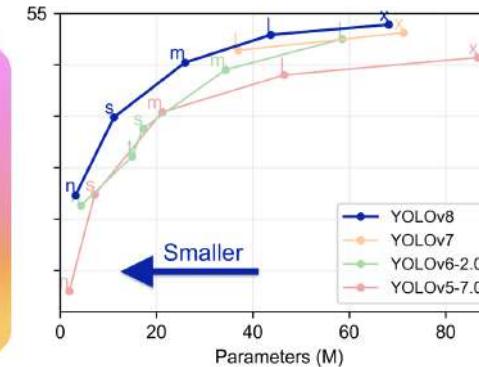
English | 简体中文

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.



YOLO Models (v9 – v12) Timeline

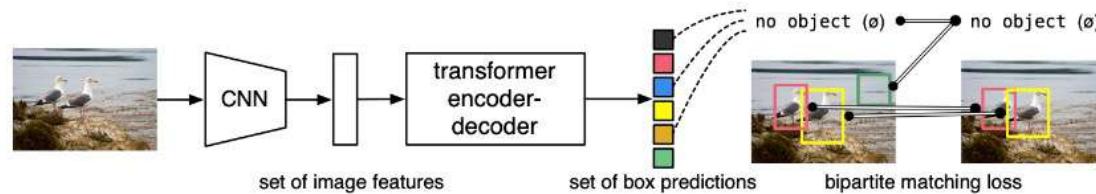
Model	Release Timeframe	Team / Organization	Key Highlight
YOLOv12	Feb 2025	Google Research + Tsinghua University (collaboration)	Attention-centric architecture with high accuracy and real-time speed
YOLOv11	Late 2024	Tencent Youtu Lab	Enhanced architecture (C3k2, SPPF, C2PSA), versatile across detection, segmentation, pose
YOLOv10	Mid 2024	Tsinghua University + Huawei Noah's Ark Lab	End-to-end design without NMS, major speed and efficiency gains
YOLOv9	Early 2024	Wong Kin Yiu (original YOLO contributor) + Community researchers	PGI and GELAN for better gradients and efficiency
Ultralytics YOLO11	2024–2025	Ultralytics	Practical SOTA release, supports detection, segmentation, pose, classification, track

+ DETR (Detection Transformer) [Meta, 2020]

DETR: End-to-End Object Detection with Transformers

Support Ukraine

PyTorch training code and pretrained models for DETR (DEtection TTransformer). We replace the full complex hand-crafted object detection pipeline with a Transformer, and match Faster R-CNN with a ResNet-50, obtaining **42 AP** on COCO using half the computation power (FLOPs) and the same number of parameters. Inference in 50 lines of PyTorch.



- 1. Deformable DETR (2020)
■ Team: Microsoft Research Asia
- 2. Conditional DETR (2021)
■ Team: Chinese Academy of Sciences
- 3. Anchor DETR (2021)
■ Team: SenseTime and Shanghai Jiao Tong University
- 4. Efficient DETR (2021)
■ Team: UC Berkeley and Google Research
- 5. DINO (DETR with Improved Non-Autoregressive Decoding) (2022)
■ Team: Meta AI (formerly Facebook AI Research)
- 6. RT-DETR (Real-Time DETR) (2024)
■ Team: Alibaba DAMO Academy

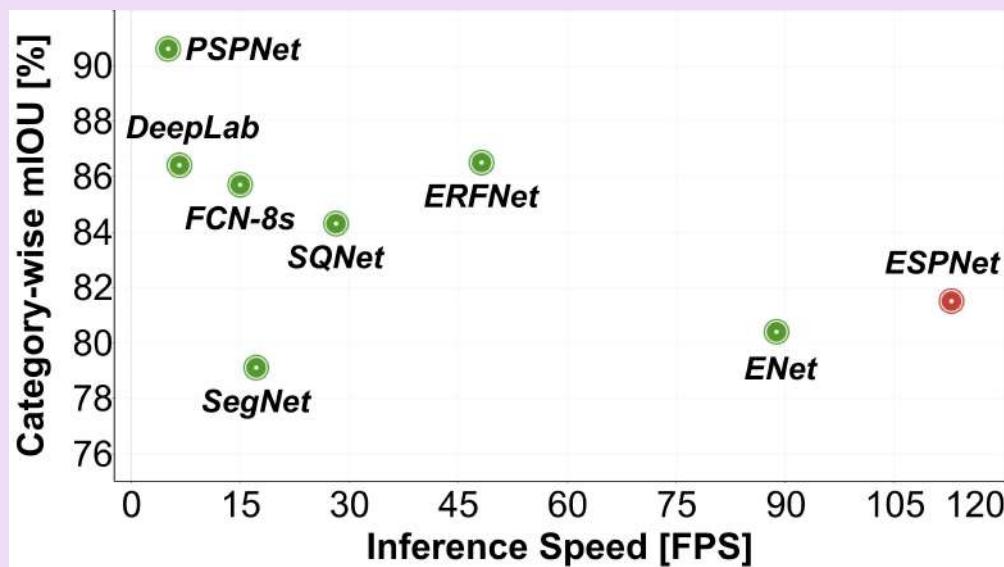


Top Object Detection Models (2025)

Model	Team / Source	Notable Metric (COCO)	Strengths
RF-DETR	Roboflow	73.6 AP ₅₀ / 54.7 AP_{50:95} (Medium)	Real-time DETR; best mAP at low latency
YOLOv12	YOLO authors (YOLO Series)	—	Latest YOLO, attention-enhanced
YOLO11 (Ultralytics)	Ultralytics	Up to ~50 AP ₅₀ / ~48.6 AP_{50:95}	Multi-task, production-ready

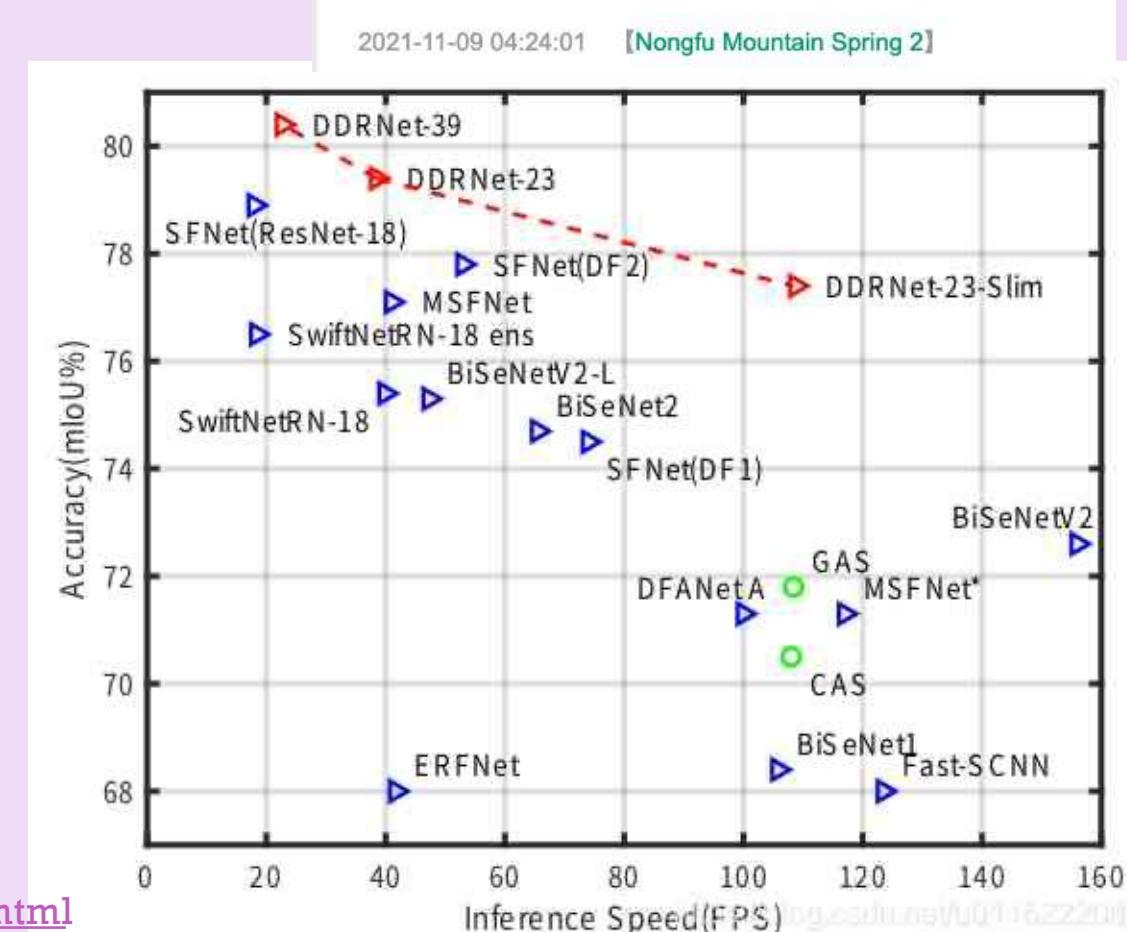


SOTA of Semantic Segmentation



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

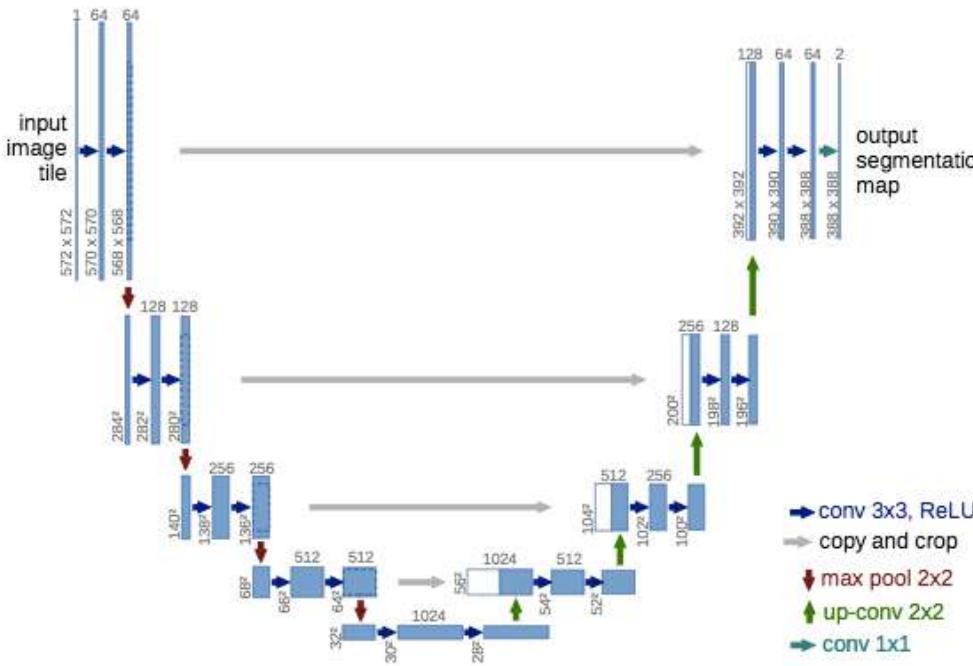
<https://chowdera.com/2021/11/20211109042359120o.html>



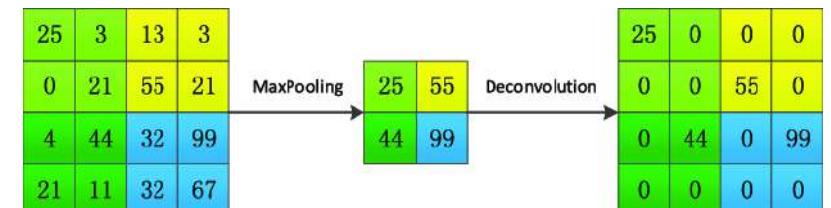


UNet: Encoder-Decoder Network

■ U-Net



Deconvolutional layer



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. \(2015\)](#)

+ Recent interesting segmentation networks

- **1. ContextFormer (2025)**
- Team: Abid, Mehta, Wu, Dharejo, Timofte
- Architecture: Hybrid CNN + Vision Transformer
- Strengths: Combines high accuracy with real-time efficiency
- Applications: Urban scene understanding, robotics, autonomous driving

- **2. Segment Anything Model (SAM) [Meta, 2023] → SAM V2 [Meta, 2024]**
- Explanation: SAM is a highly versatile model designed to segment any object within an image with minimal user input. It uses a prompt-based approach, where the user provides a prompt (e.g., a point or bounding box), and the model generates a mask for the object. SAM is particularly noted for its generalizability across different segmentation tasks.

- **3. Mask2Former [Meta, 2022]**
- Explanation: Mask2Former is a unified model for instance, semantic, and panoptic segmentation tasks. It improves upon the original MaskFormer by incorporating transformers into the architecture, enabling it to handle different segmentation tasks within a single framework efficiently.

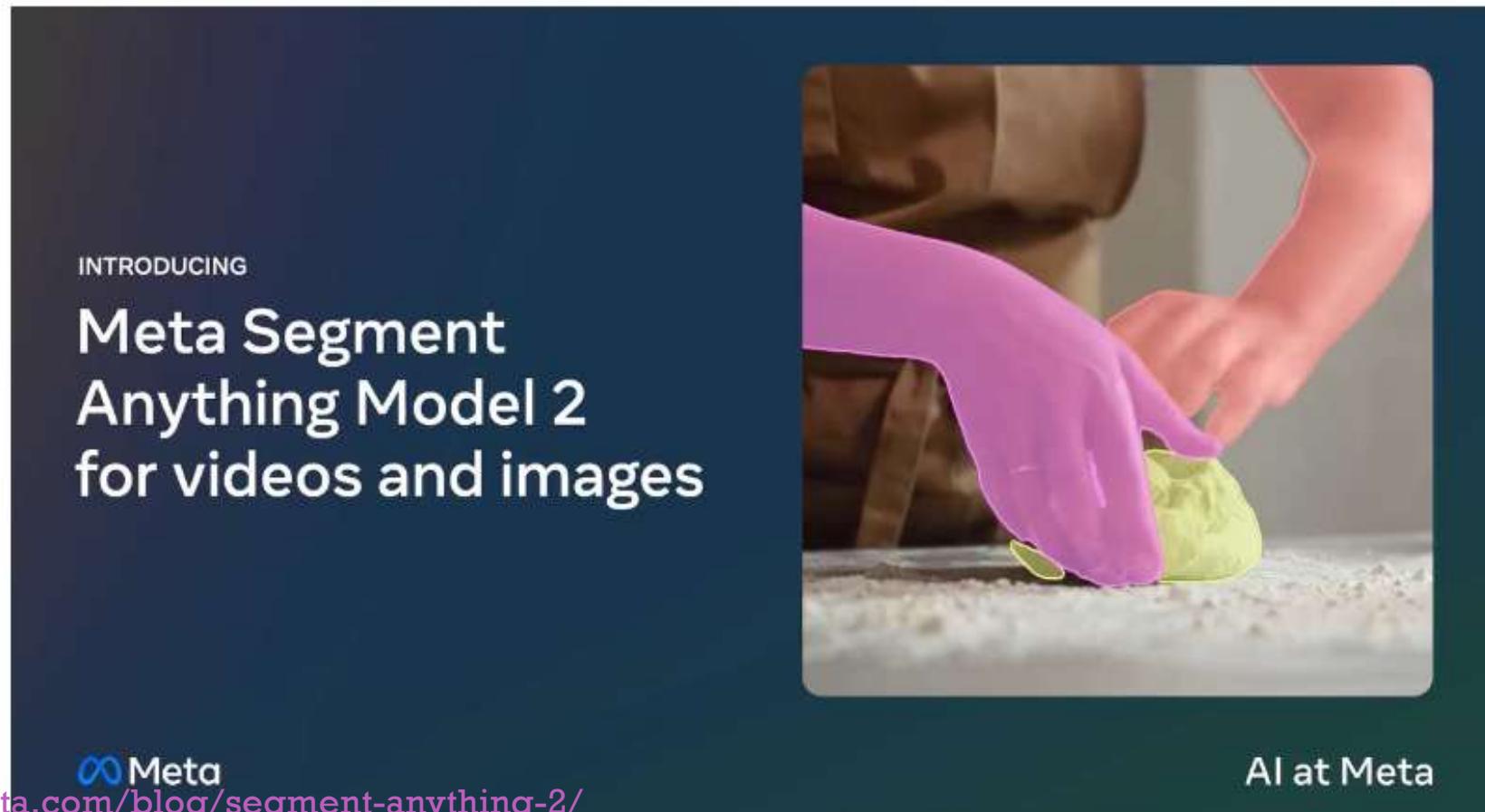
- **4. Swin Transformer V2 [Microsoft, 2022]**
- Explanation: Swin Transformer V2 builds on the hierarchical vision transformer design introduced in the original Swin Transformer. It enhances performance in both segmentation and classification tasks and has achieved SOTA results on benchmarks such as COCO and ADE20K.

- **5. SegFormer [NVIDIA, 2021]**
- Explanation: SegFormer is a transformer-based model that is designed to be both efficient and accurate for semantic segmentation tasks. It integrates transformers with lightweight MLP decoders, enabling high performance while maintaining computational efficiency.

- **6. DeepLabV3+ [Google, 2018]**
- Explanation: DeepLabV3+ is an extension of the DeepLab series, which combines spatial pyramid pooling and atrous convolutions. The "+" version introduces a decoder module to improve object boundary delineation, making it a strong performer in semantic segmentation.

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

July 29, 2024 • ① 15 minute read



INTRODUCING

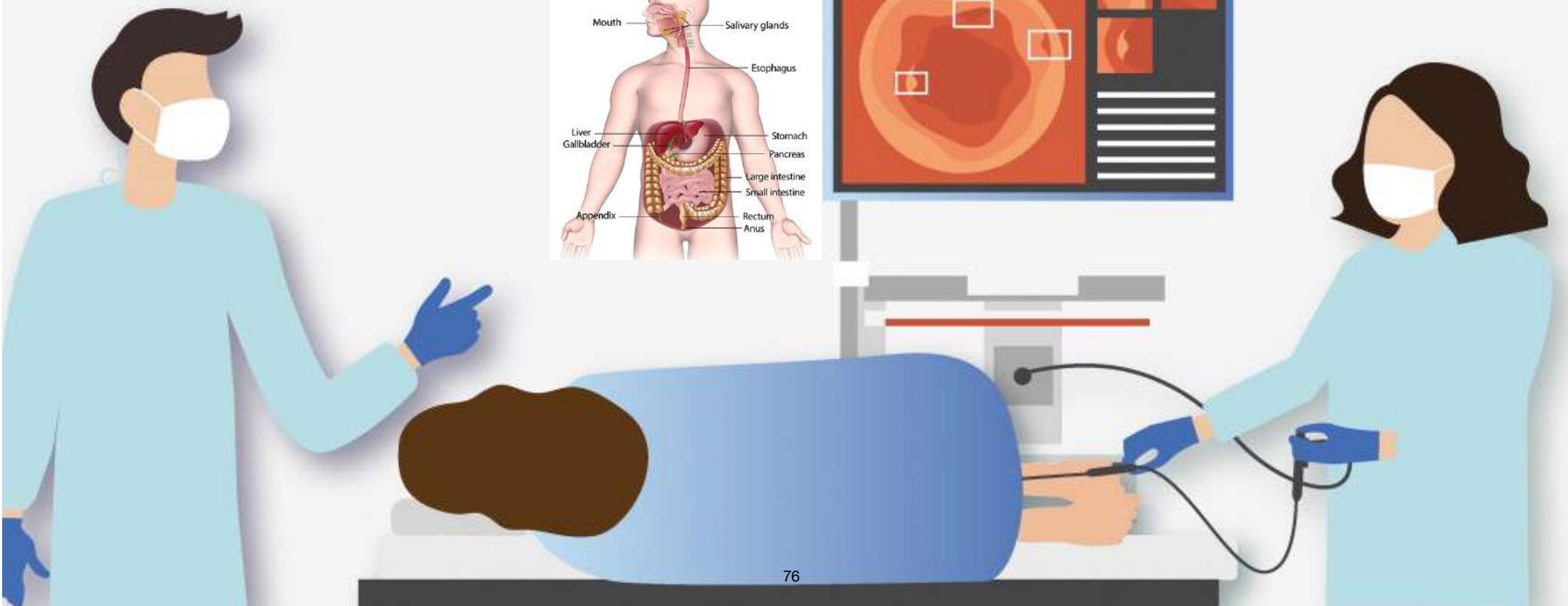
Meta Segment Anything Model 2 for videos and images

AI at Meta

<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.

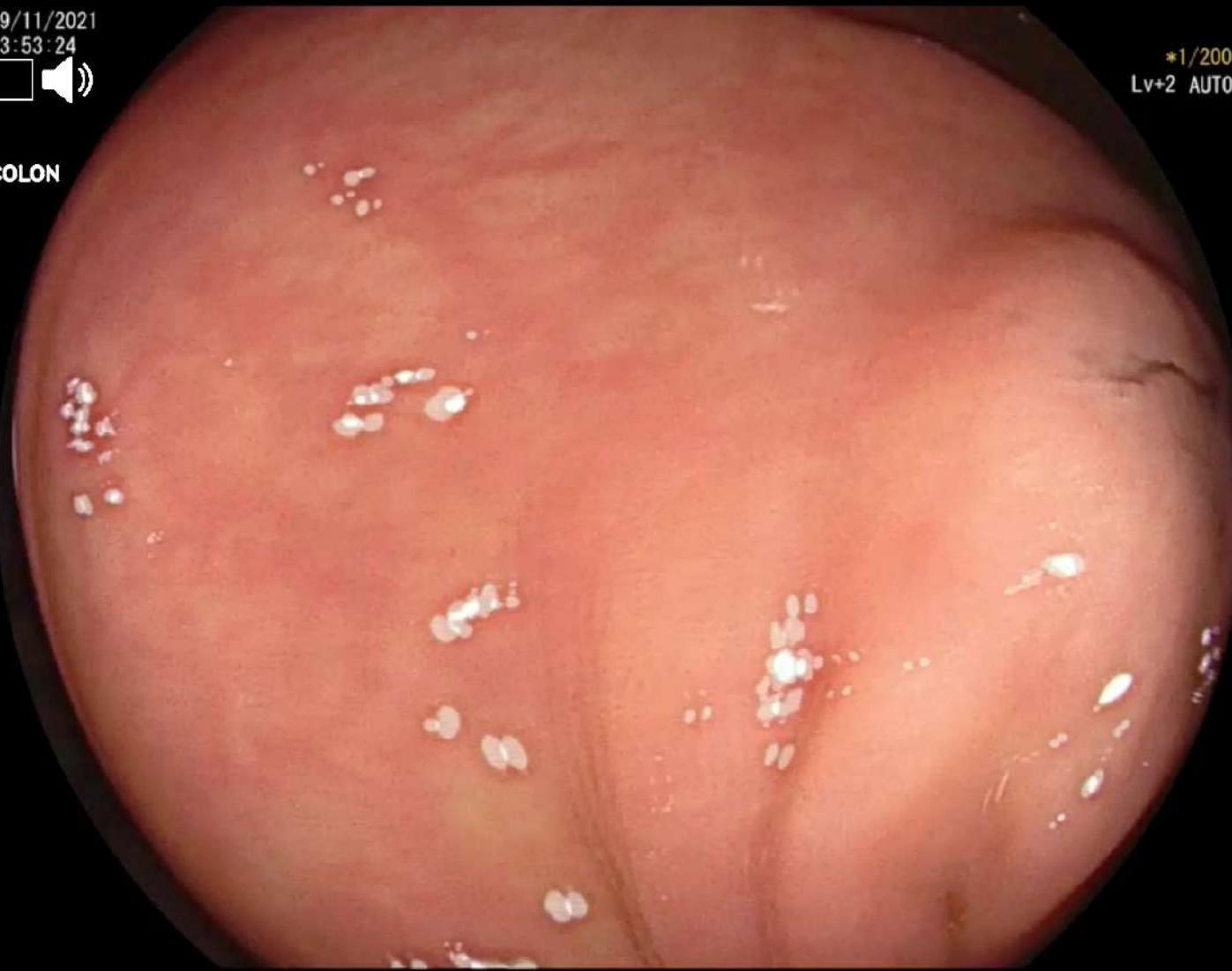


09/11/2021

13:53:24



COLON



*1/200
Lv+2 AUTO

HT NR
SE
3.8 12.0 s1: F+T
12.0 s2: LM
s3: CAD
s4:
EC-760R-V/L
3C729K035
BL-7000
CHULALONGKORN HOS



NBI

Name:

Sex: Age:

D.O.B.:

18/05/2021

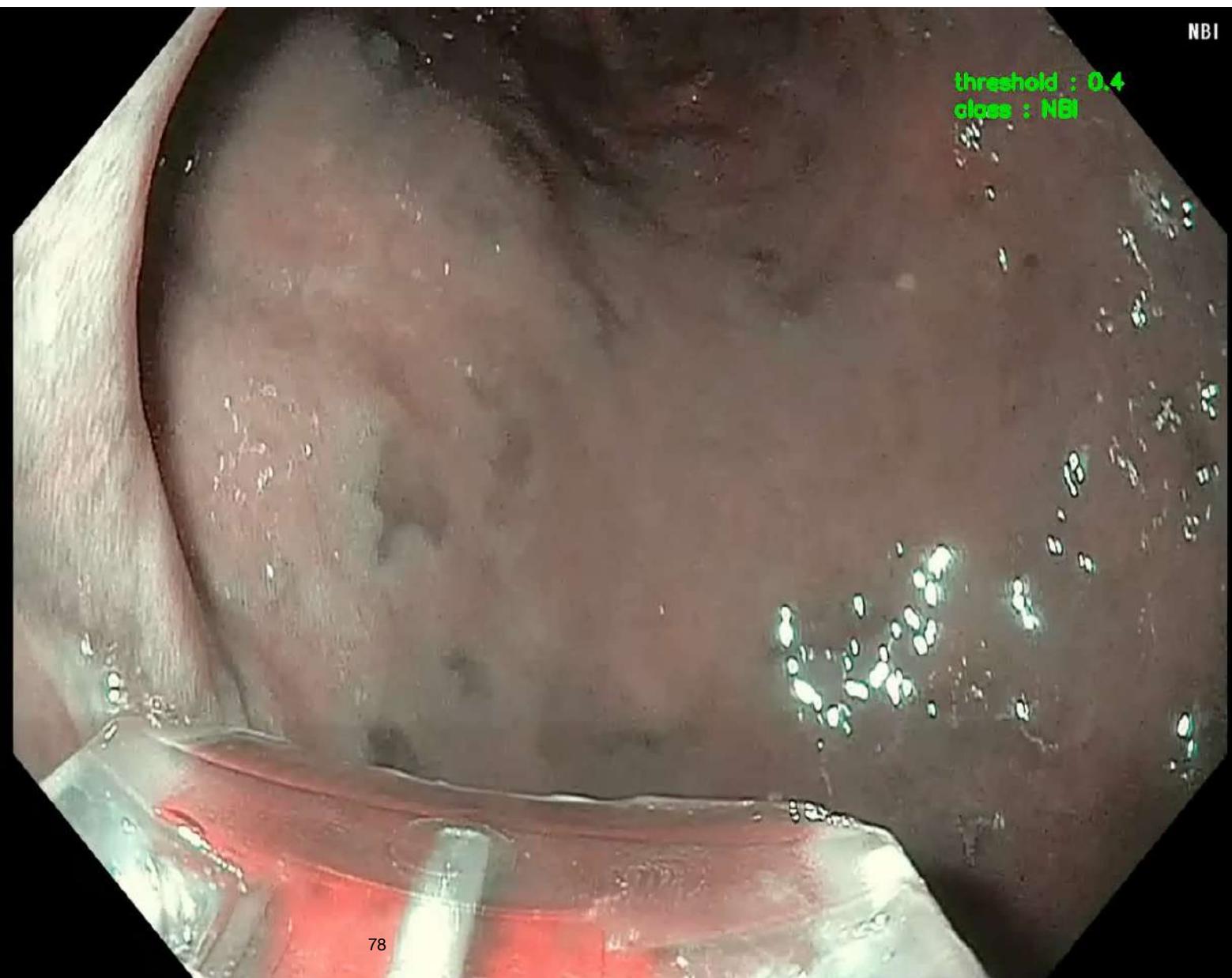
08:29:36

threshold : 0.4
close : NBI

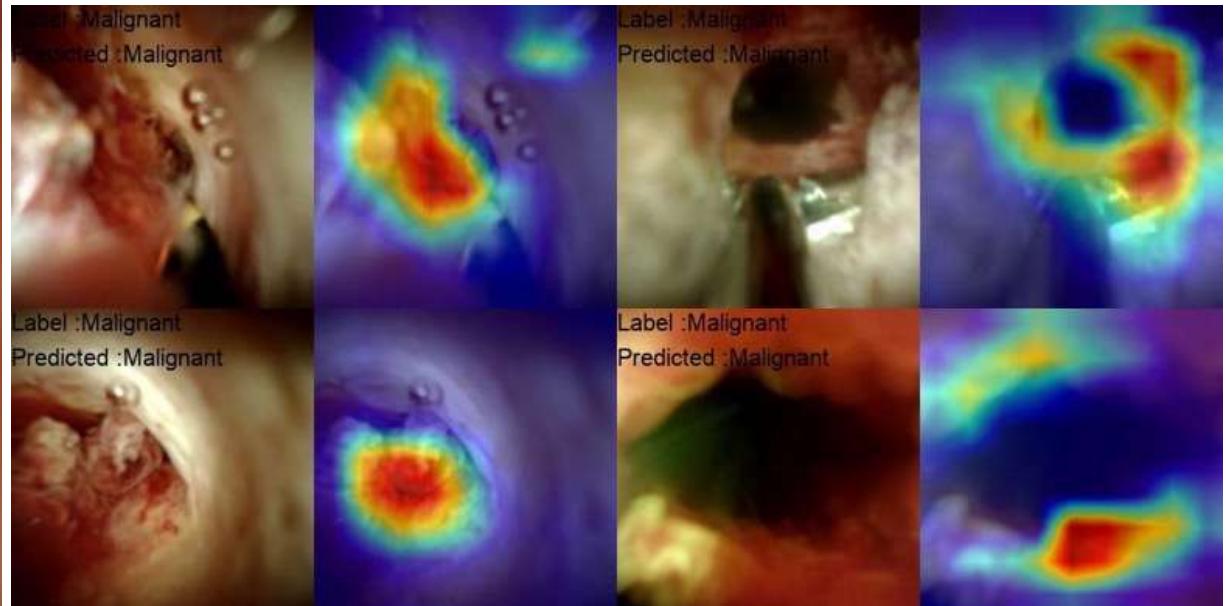
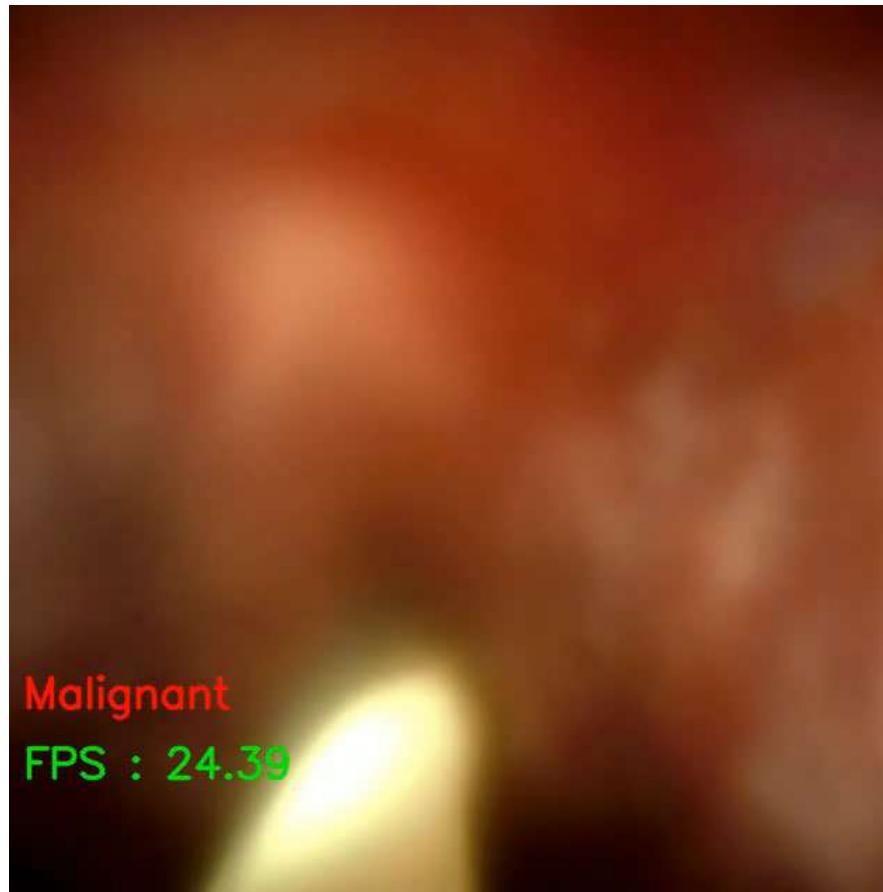
■■□/---(0/1)

Eh:B8 Cm:1

Comment:



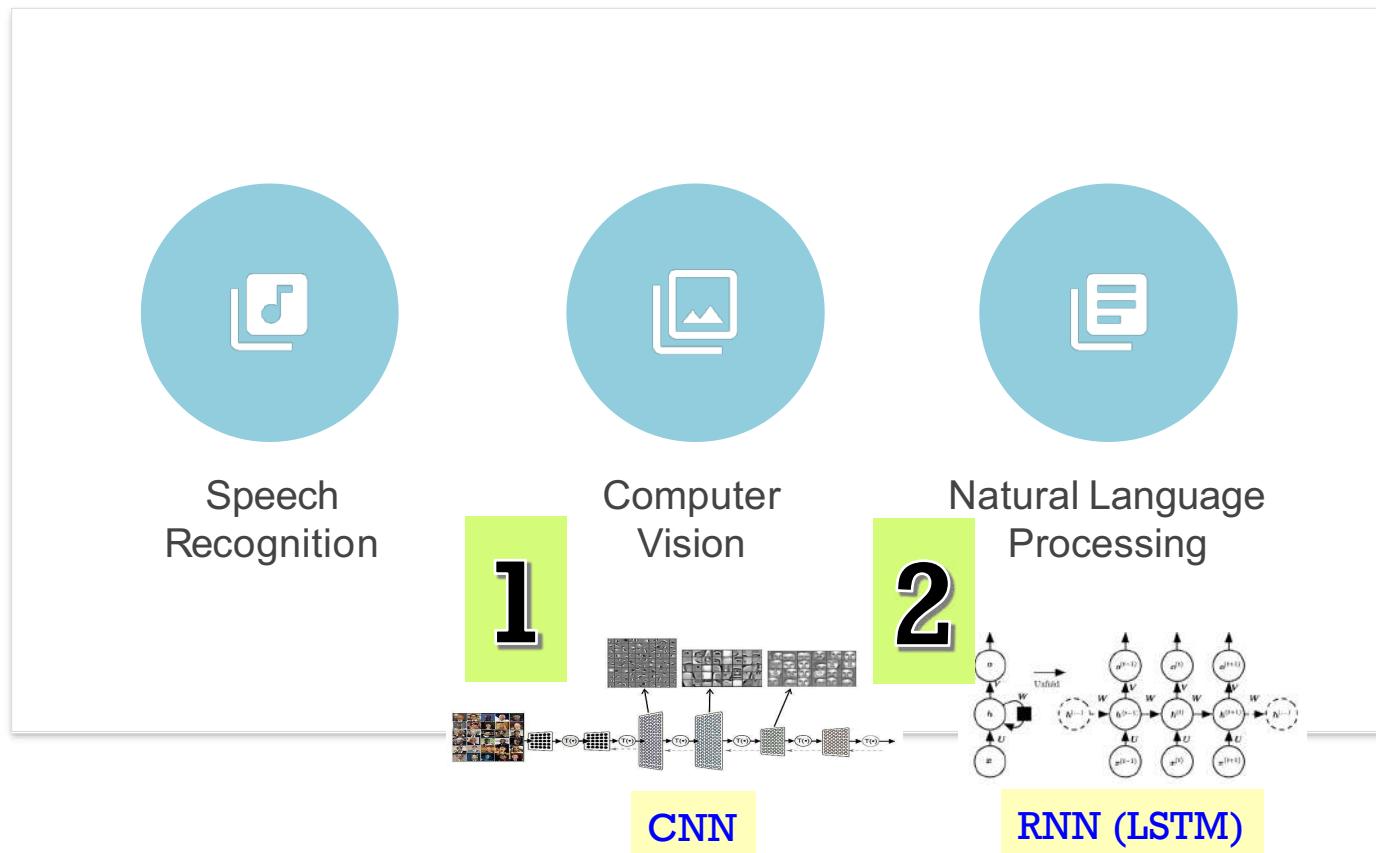
Cholangioscopy



+

RNN

Deep Learning Application



Top 10 Strategic Technology Trends for 2020

People-centric Smart spaces



Hyperautomation



Empowered Edge



Multiexperience



Distributed Cloud



Democratization



Autonomous Things



Human Augmentation



Practical Blockchain



Transparency and Traceability



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

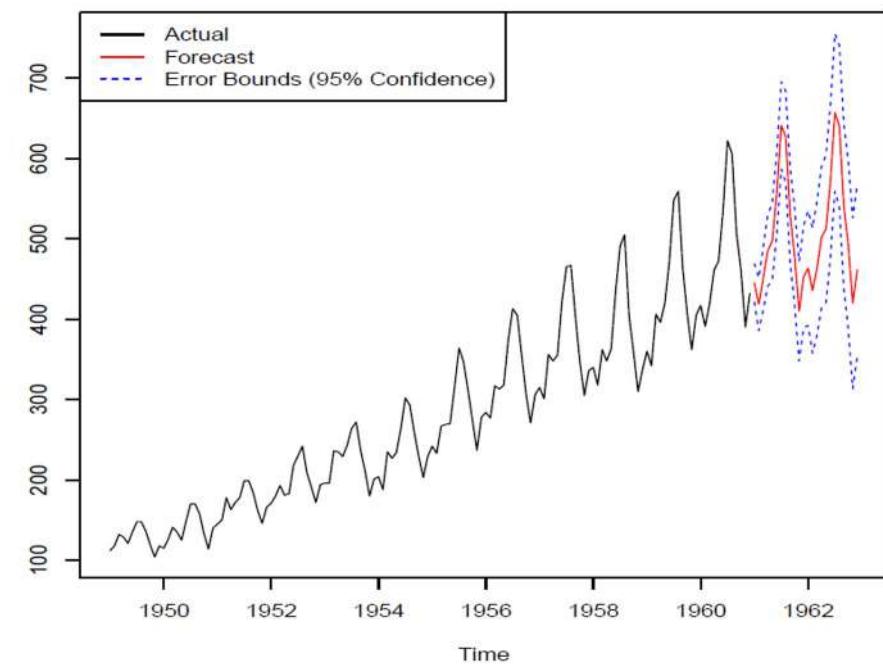


Hyperautomation

Combinatorial innovation

gartner.com/SmarterWithGartner

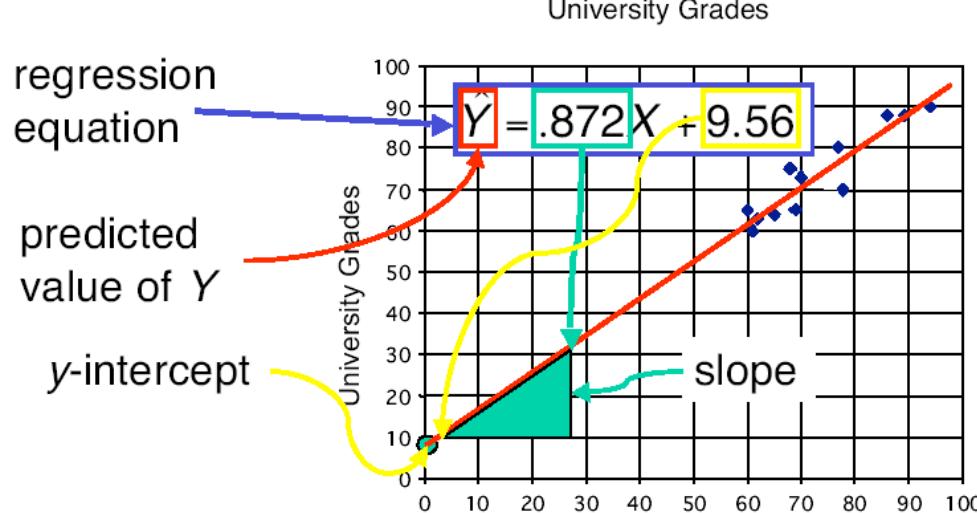
Gartner



Source: Gartner
© 2020 Gartner, Inc. and/or its affiliates. All rights reserved. GARTNER is a registered trademark and service mark of Gartner, Inc. and/or its affiliates in the U.S. and international markets.



Regression – Linear Relationship



$$\hat{y} = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

target intercept input

weight, coefficient

- The least square method aims to minimize the following term

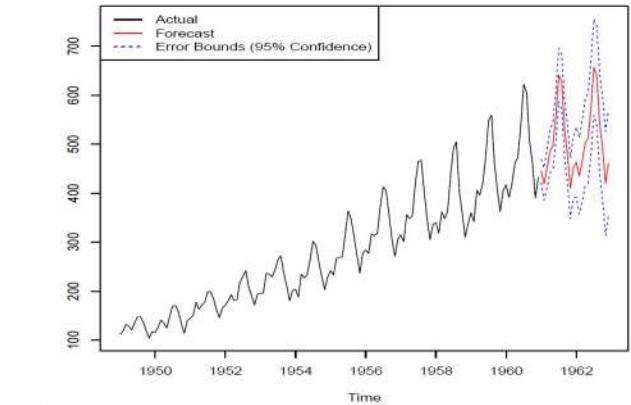
$$\sum_{\text{training data}} (y_i - \hat{y}_i)^2$$

Autoregressive Model – Linear Relationship

- Based on linear regression, but using previous timestep data

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

- Where X_t is data at timestep t , c is constant, and each timestep data



are parameters for
 $\varphi_1, \varphi_2, \varphi_3, \dots, \varphi_p$

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

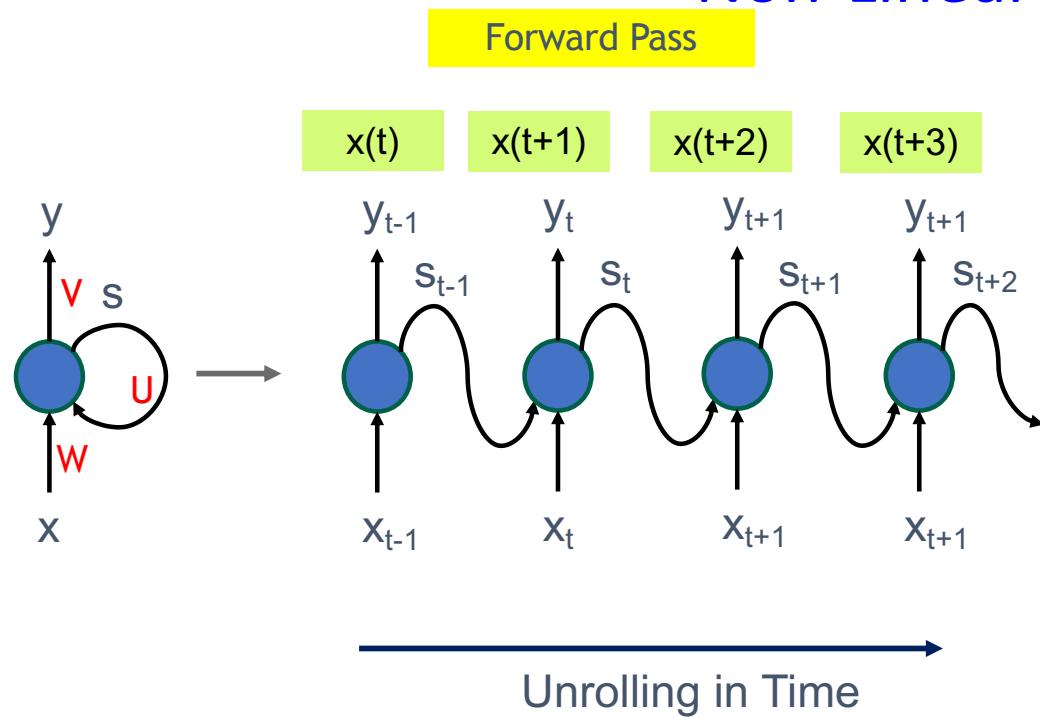
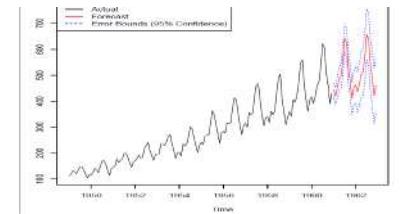
$$X_t = c + \varphi_1 X_{t-1} + \varphi_2 X_{t-2}$$

- 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)

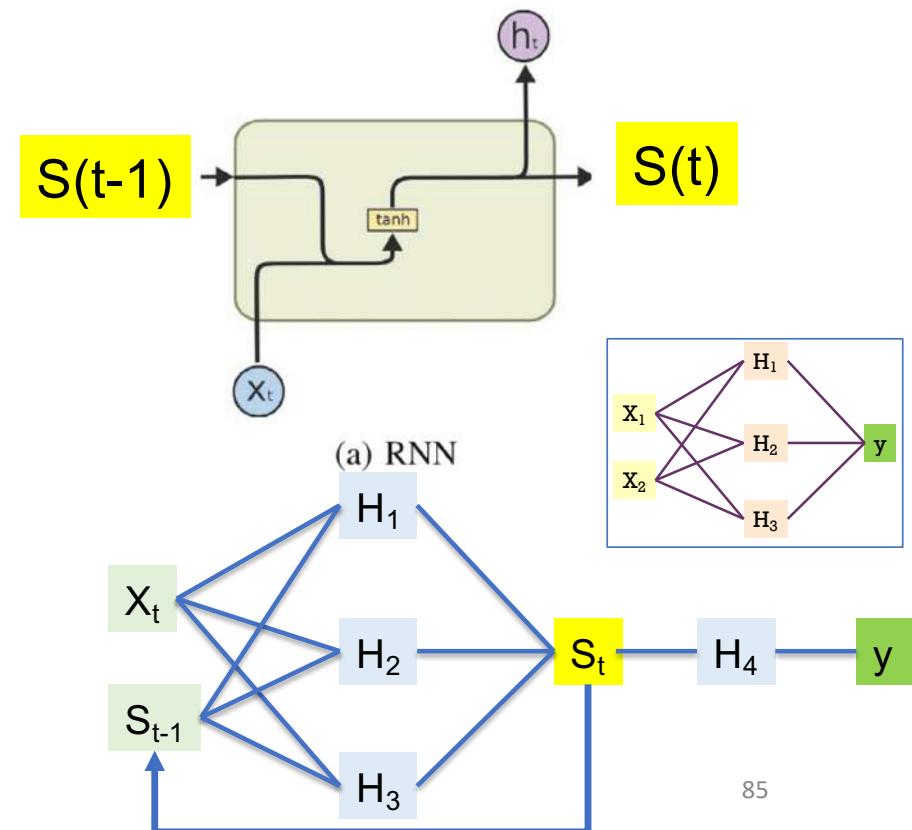
Non-Linear Relationship



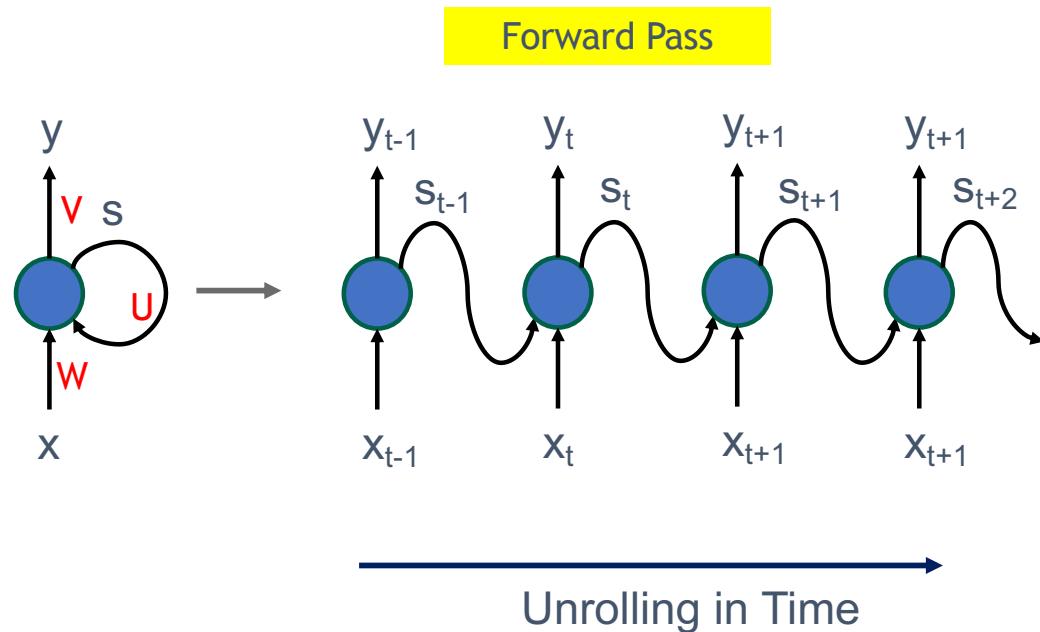
$$s_t = \tanh(Ux_t + Vs_{t-1})$$

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

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

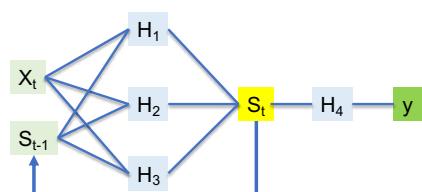


RECURRENT NEURONS (RNN)

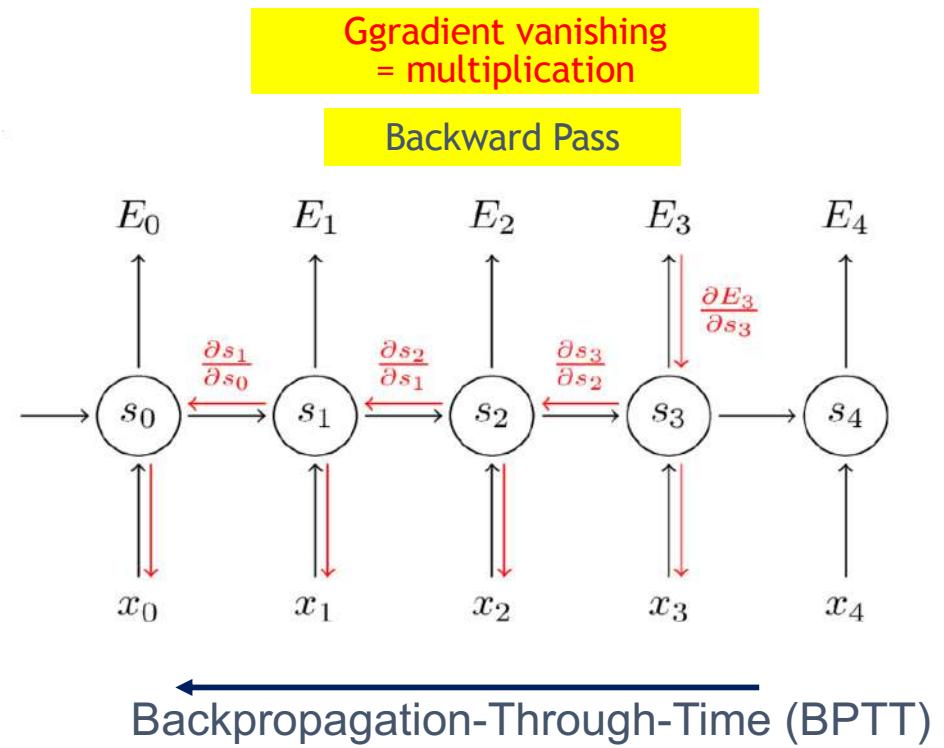


$$s_t = \tanh(Ux_t + Vs_{t-1})$$

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



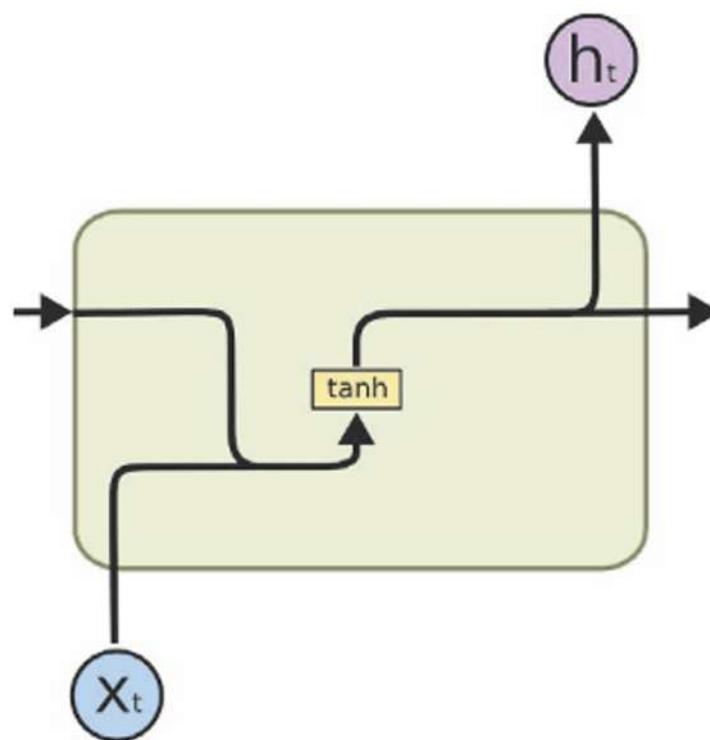
Reference: <https://www.cse.iitk.ac.in/users/sigml/lec/Slides/LSTM.pdf>



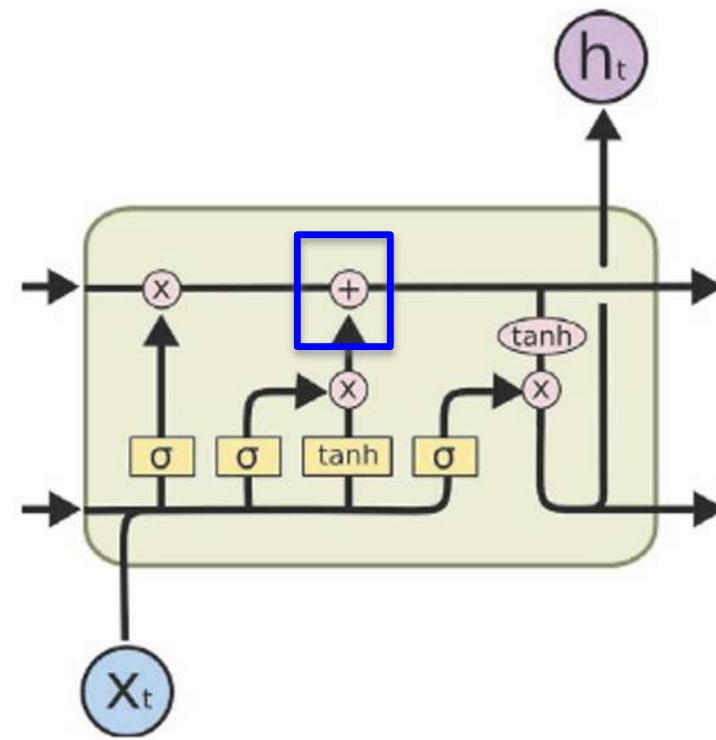
$$\frac{\partial E}{\partial \mathbf{W}} = \sum_t \frac{\partial E_t}{\partial \mathbf{W}}$$

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

RNN VS LSTM



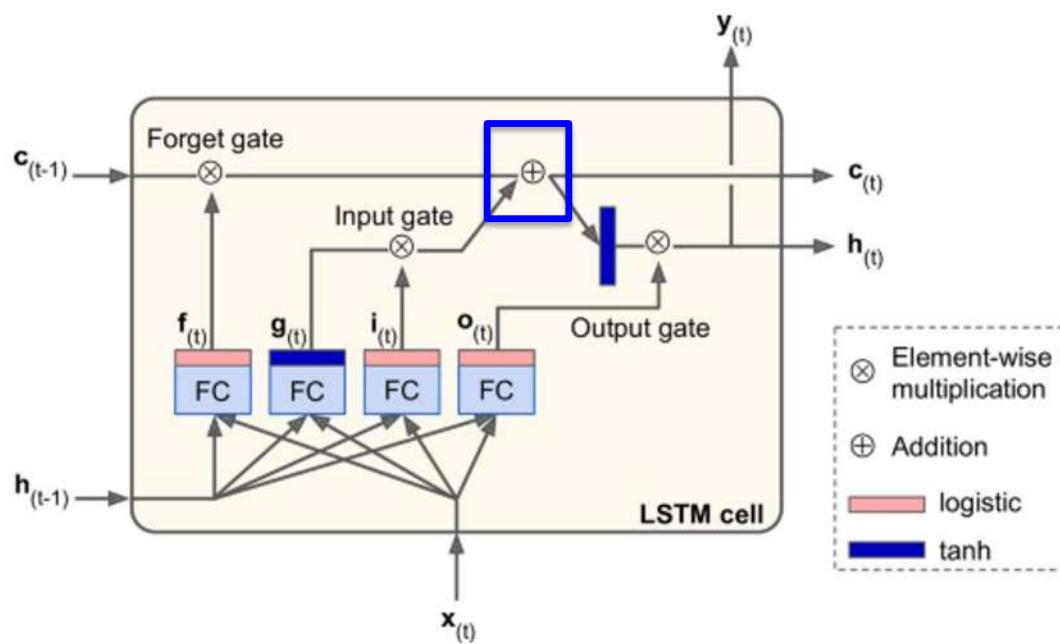
(a) RNN



(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.

$$i_{(t)} = \sigma(W_{xi}^T \cdot x_{(t)} + W_{hi}^T \cdot h_{(t-1)} + b_i)$$

$$f_{(t)} = \sigma(W_{xf}^T \cdot x_{(t)} + W_{hf}^T \cdot h_{(t-1)} + b_f)$$

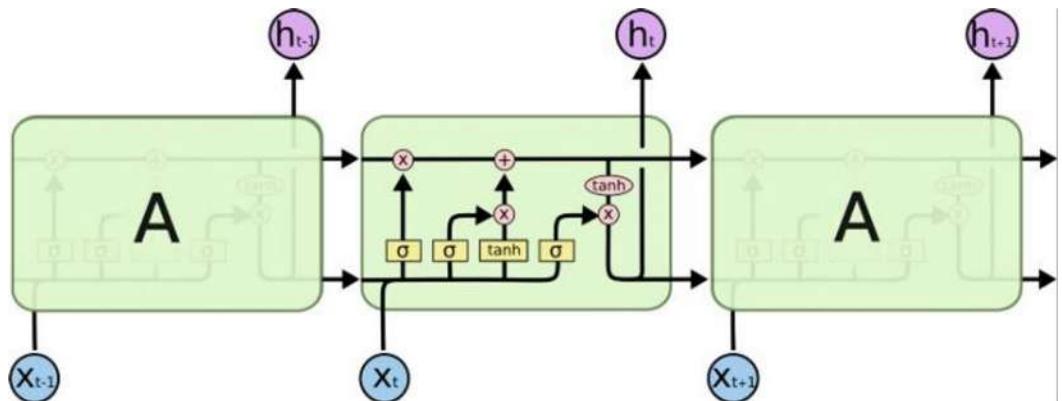
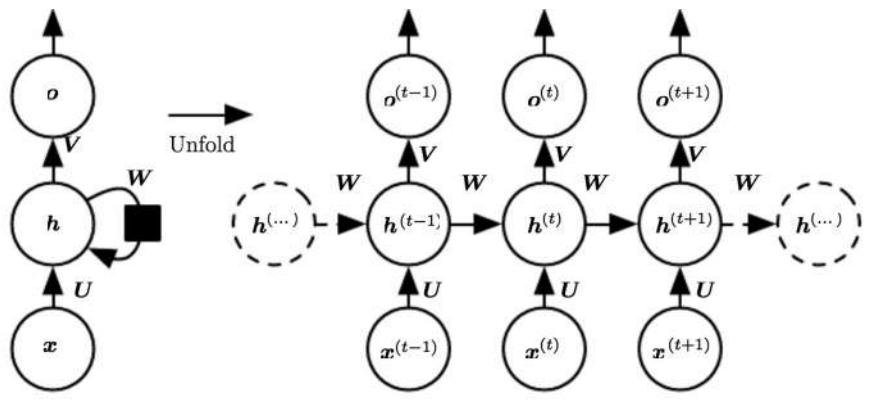
$$o_{(t)} = \sigma(W_{xo}^T \cdot x_{(t)} + W_{ho}^T \cdot h_{(t-1)} + b_o)$$

$$g_{(t)} = \tanh(W_{xg}^T \cdot x_{(t)} + W_{hg}^T \cdot h_{(t-1)} + b_g)$$

$$c_{(t)} = f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)}$$

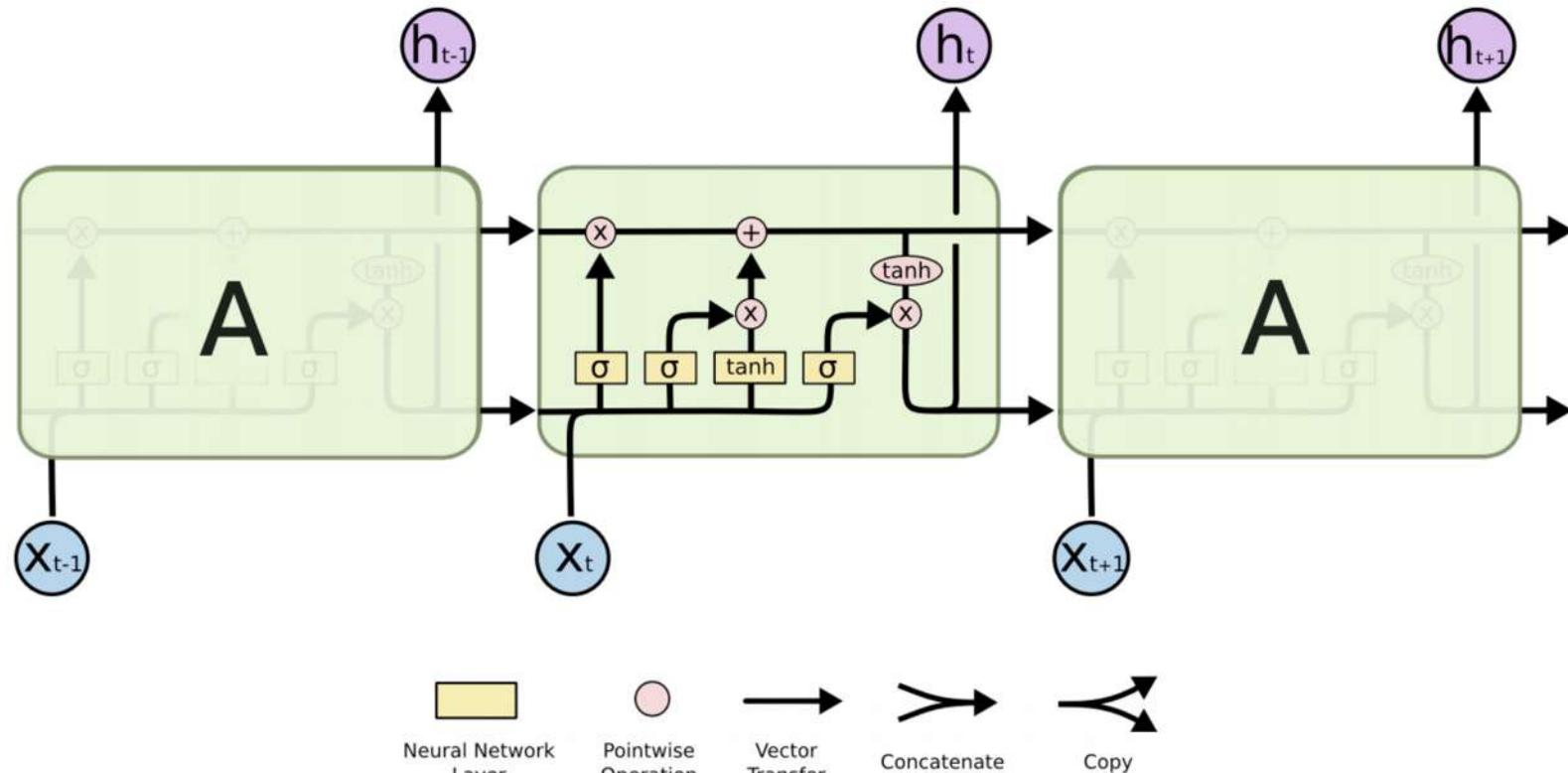
$$y_{(t)} = h_{(t)} = o_{(t)} \otimes \tanh(c_{(t)})$$

Deep Learning Approach: LSTM

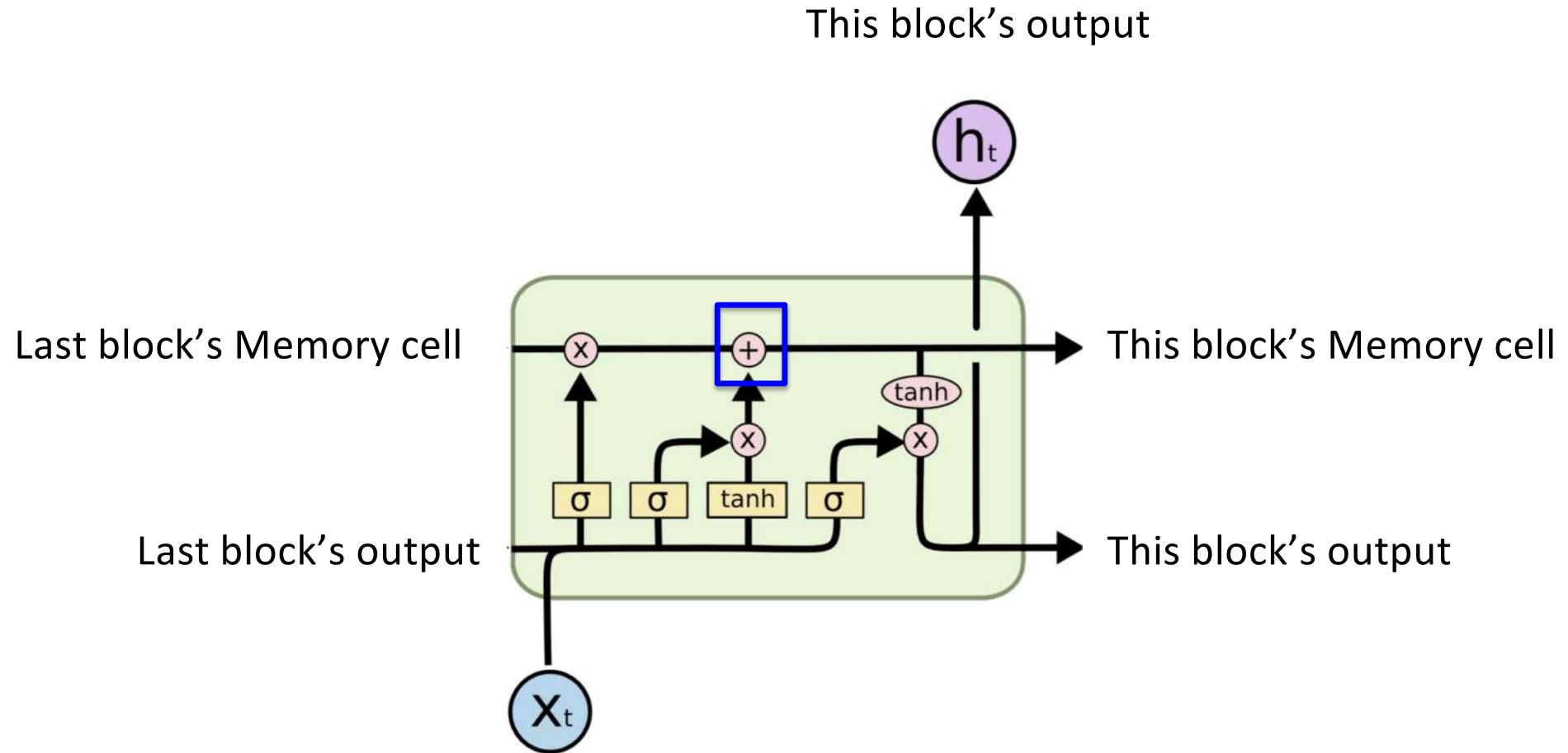


Reference: Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. MIT Press.
<http://www.deeplearningbook.org>

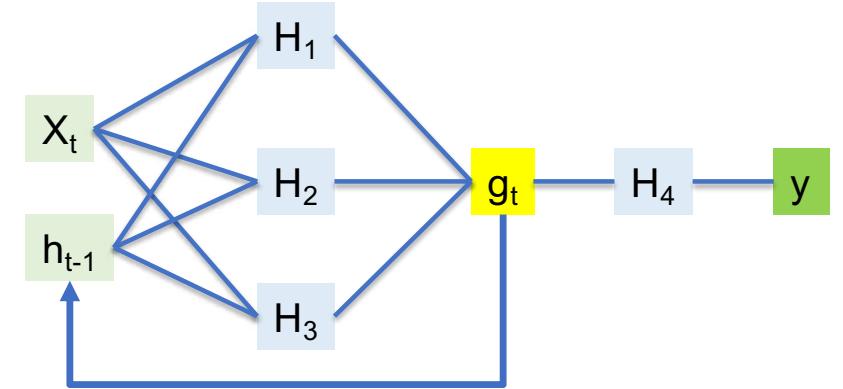
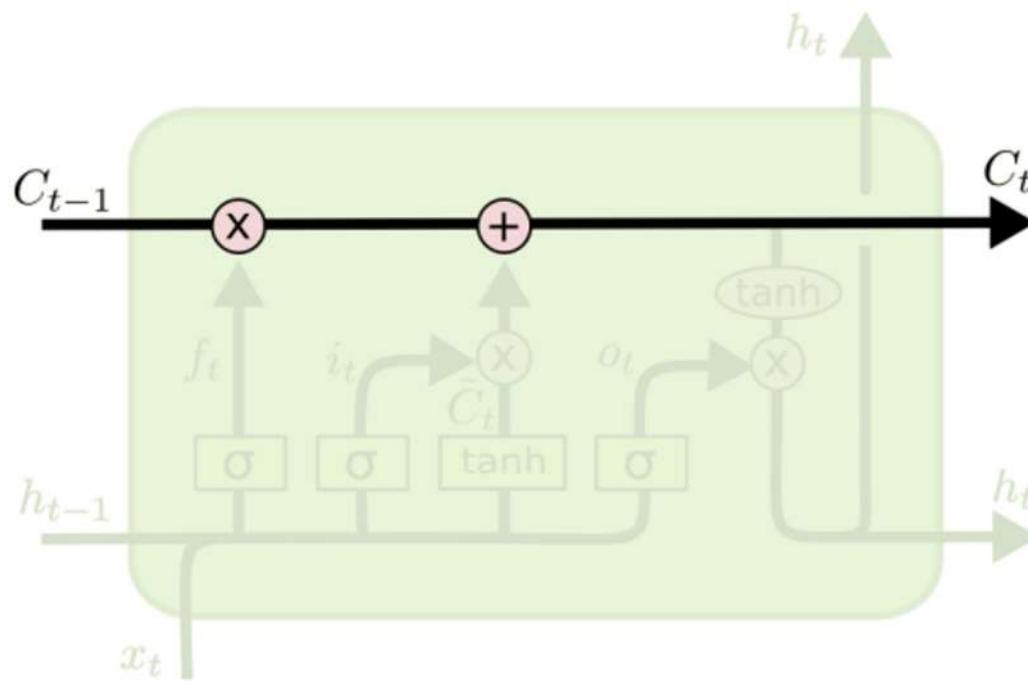
Inside LSTM



Inside LSTM



Inside LSTM: (1) Cell Memory



There are 3 gates with 2 outputs.

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

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

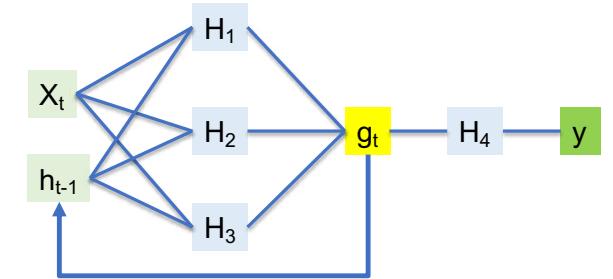
$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

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

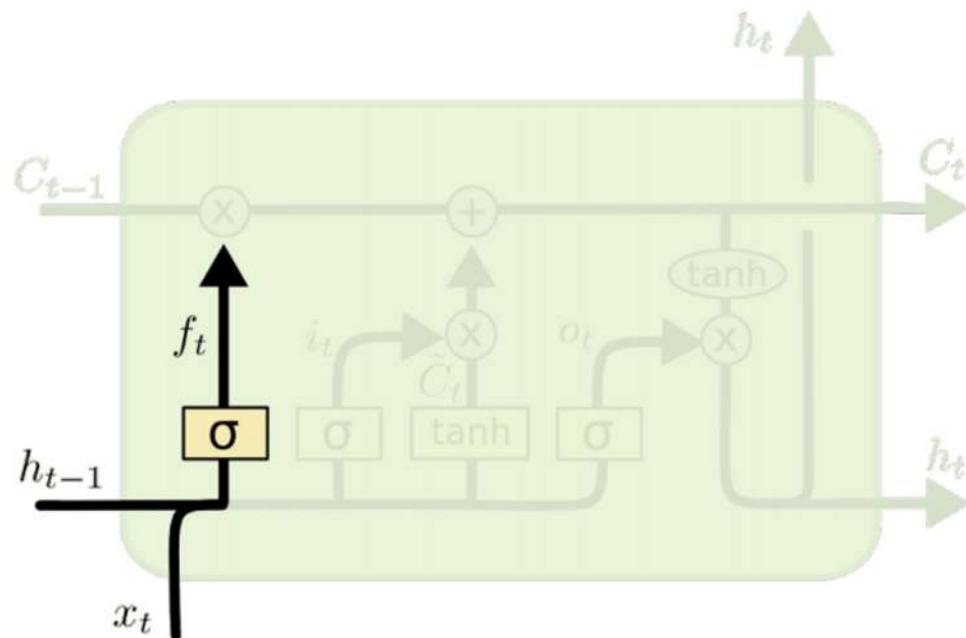
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

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

Inside LSTM: (2) Forget Gate



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

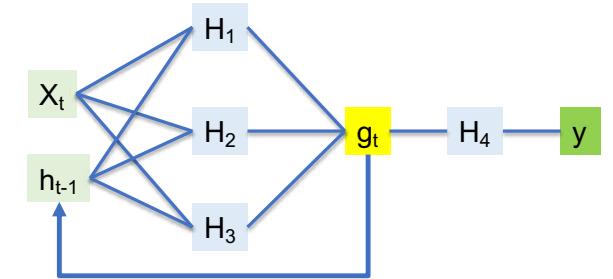
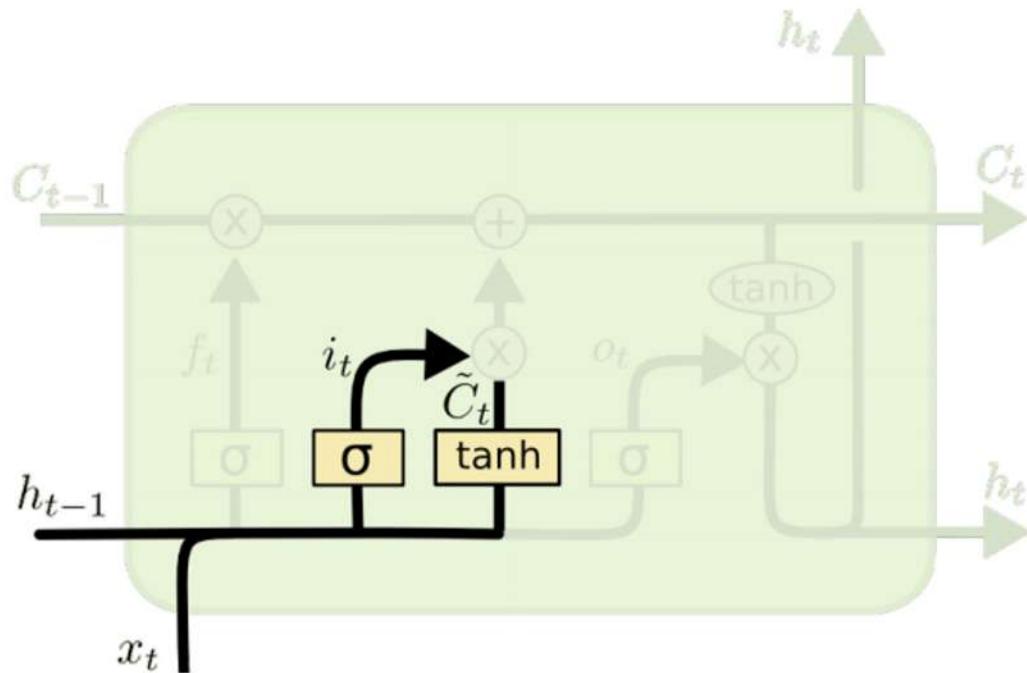


There are 3 gates with 2 outputs.

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

Inside LSTM: (3) Input Gate

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

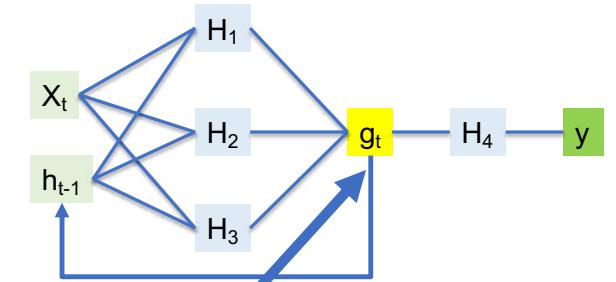
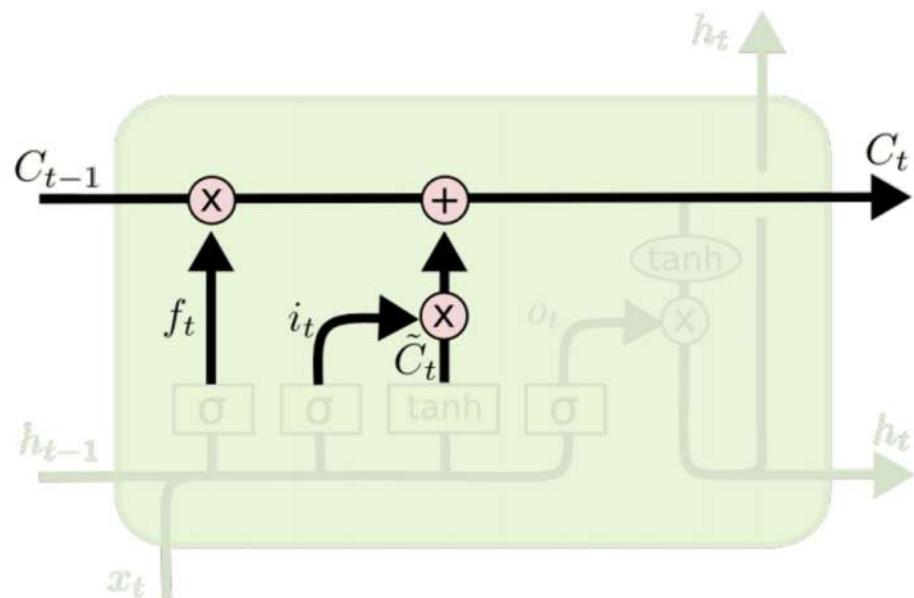


There are 3 gates with 2 outputs.

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

Inside LSTM: (4) Memory Update

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

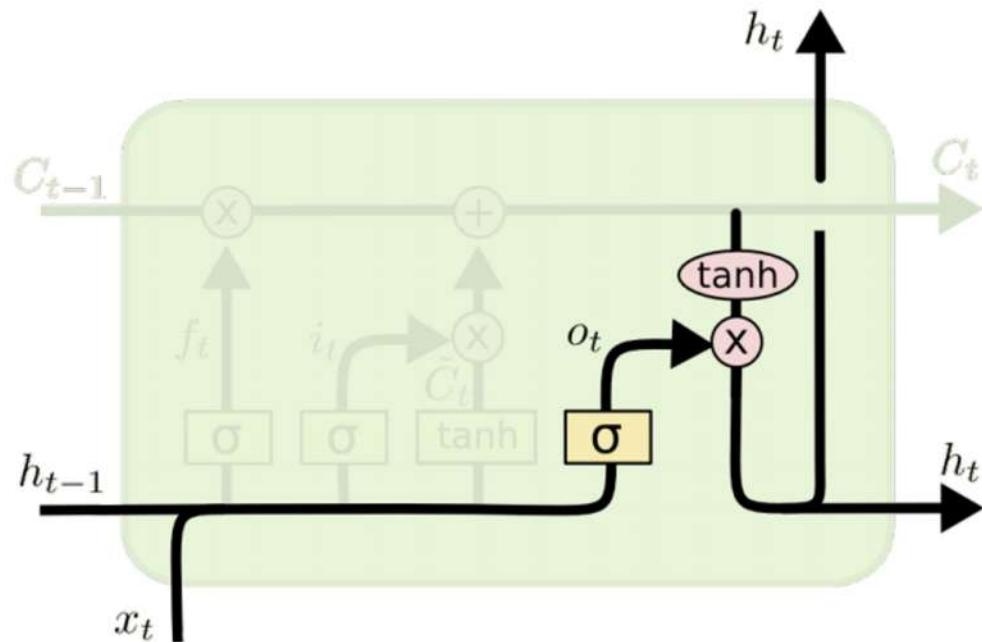


There are 3 gates with 2 outputs.

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

Inside LSTM: Output Gate

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

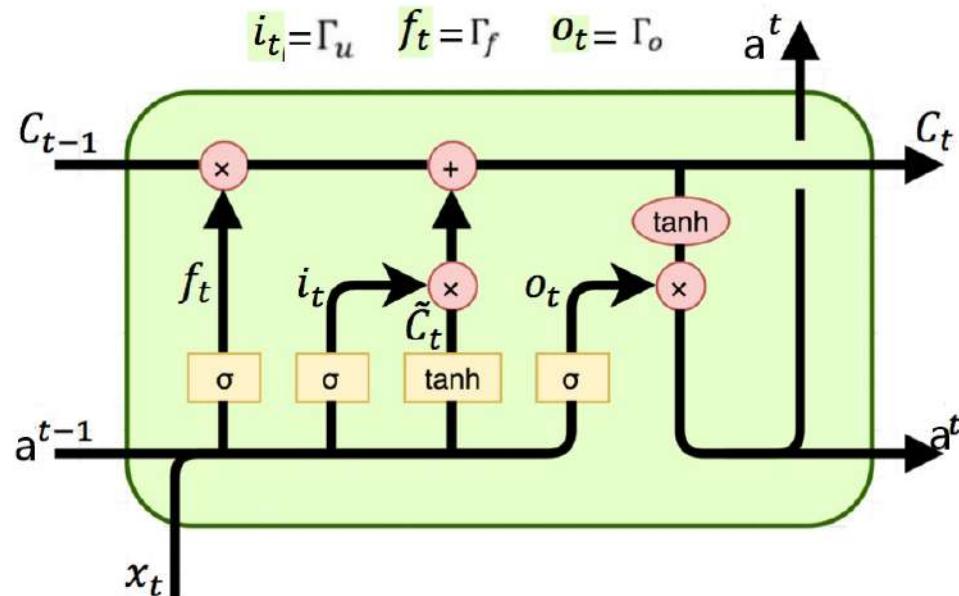


There are 3 gates with 2 outputs.

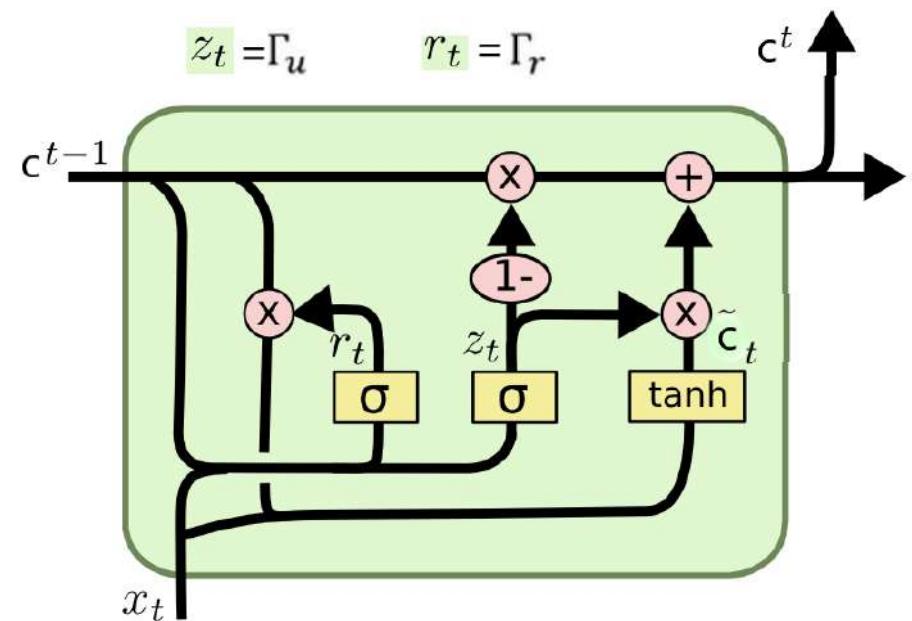
$$\begin{aligned}\mathbf{i}_{(t)} &= \sigma(\mathbf{W}_{xi}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i) \\ \mathbf{f}_{(t)} &= \sigma(\mathbf{W}_{xf}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f) \\ \mathbf{o}_{(t)} &=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

LSTM vs. Gated Recurrent Units (GRU)

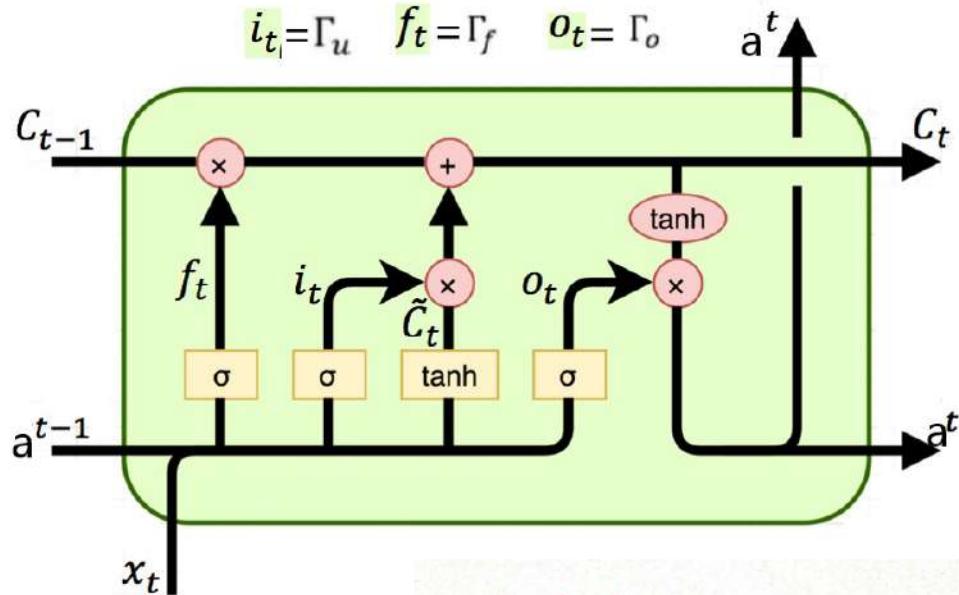
There are 3 gates (i_t , f_t , o_t)
 2 outputs ($C(t)$, $a(t)$),
 3 inputs ($x(t)$, $C(t-1)$, $a(t-1)$)



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



There are 3 gates with 2 outputs.

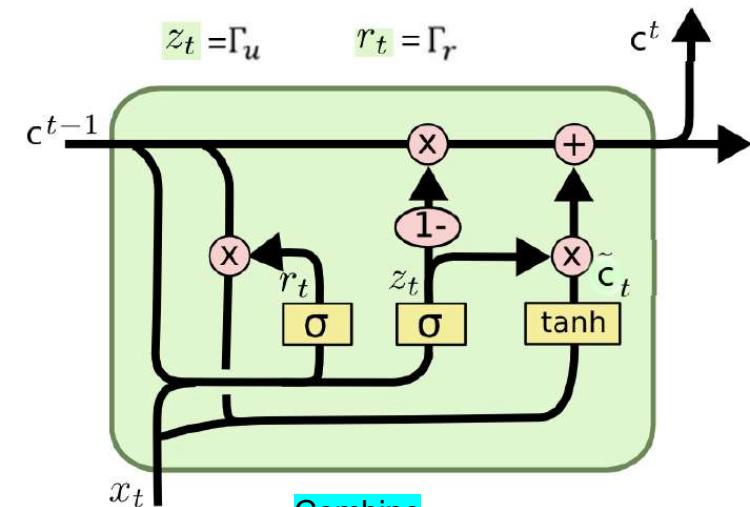


$$\begin{aligned}
 i_{(t)} &= \sigma(W_{xi}^T \cdot x_{(t)} + W_{hi}^T \cdot h_{(t-1)} + b_i) \\
 f_{(t)} &= \sigma(W_{xf}^T \cdot x_{(t)} + W_{hf}^T \cdot h_{(t-1)} + b_f) \\
 o_{(t)} &= \sigma(W_{xo}^T \cdot x_{(t)} + W_{ho}^T \cdot h_{(t-1)} + b_o) \\
 g_{(t)} &= \tanh(W_{xg}^T \cdot x_{(t)} + W_{hg}^T \cdot h_{(t-1)} + b_g) \\
 c_{(t)} &= f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)} \\
 y_{(t)} &= h_{(t)} \otimes \tanh(c_{(t)})
 \end{aligned}$$

Handwritten Equations:

- Candidate cell** $\rightarrow \tilde{c}^{(t)} = \tanh(W_c[a^{(t-1)} + x^{(t)}] + b_c)$
- Update / Input gate** $\rightarrow T_u = \sigma(W_u[a^{(t-1)} + x^{(t)}] + b_u)$
- Forget gate** $\rightarrow T_f = \sigma(W_f[a^{(t-1)} + x^{(t)}] + b_f)$
- Output gate** $\rightarrow T_o = \sigma(W_o[a^{(t-1)} + x^{(t)}] + b_o)$
- Cell state** $\rightarrow c^{(t)} = T_u * \tilde{c}^{(t)} + T_f * c^{(t-1)}$
- Activation / Output** $\rightarrow a^{(t)} = T_o * \tanh(c^{(t)})$

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

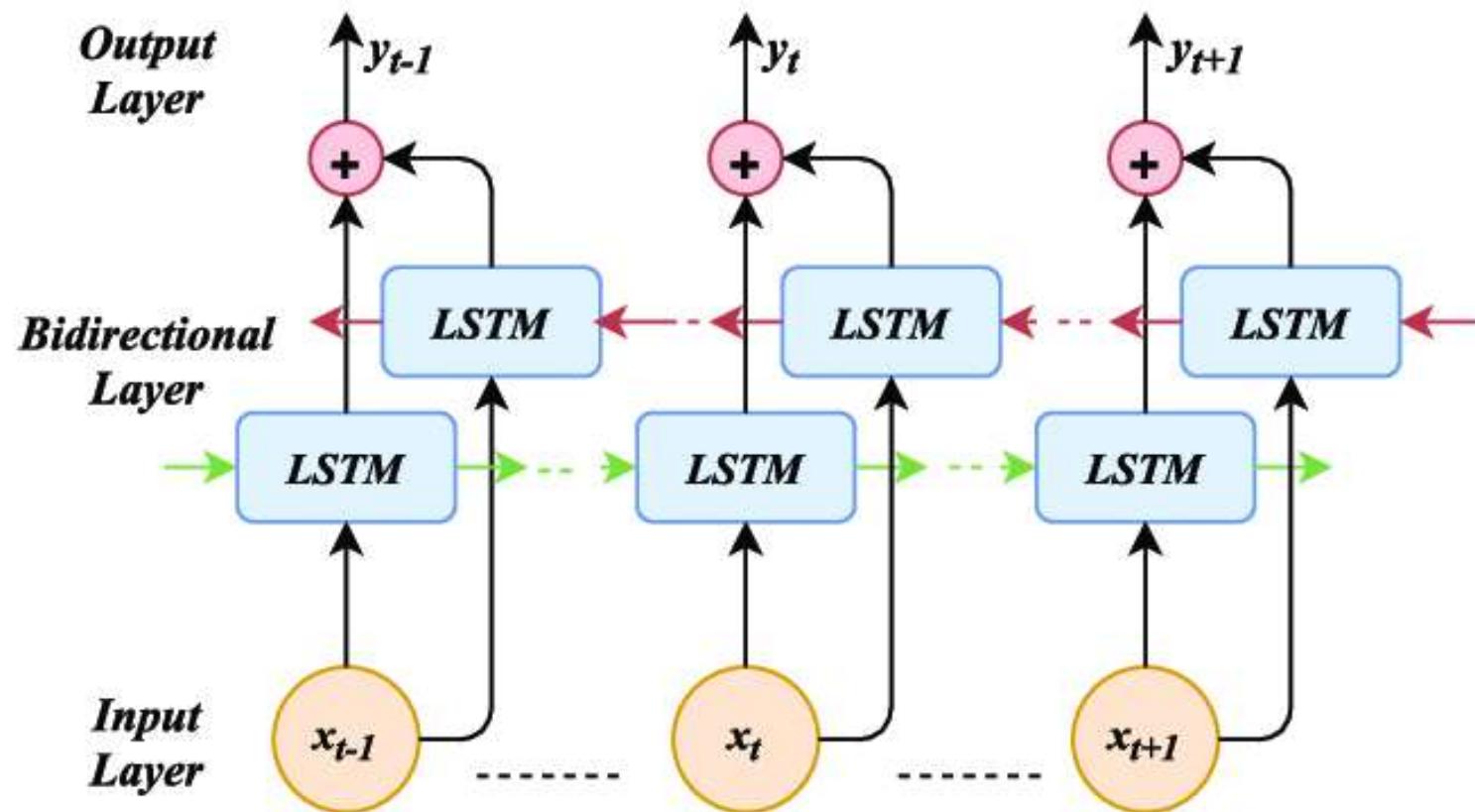


Combine

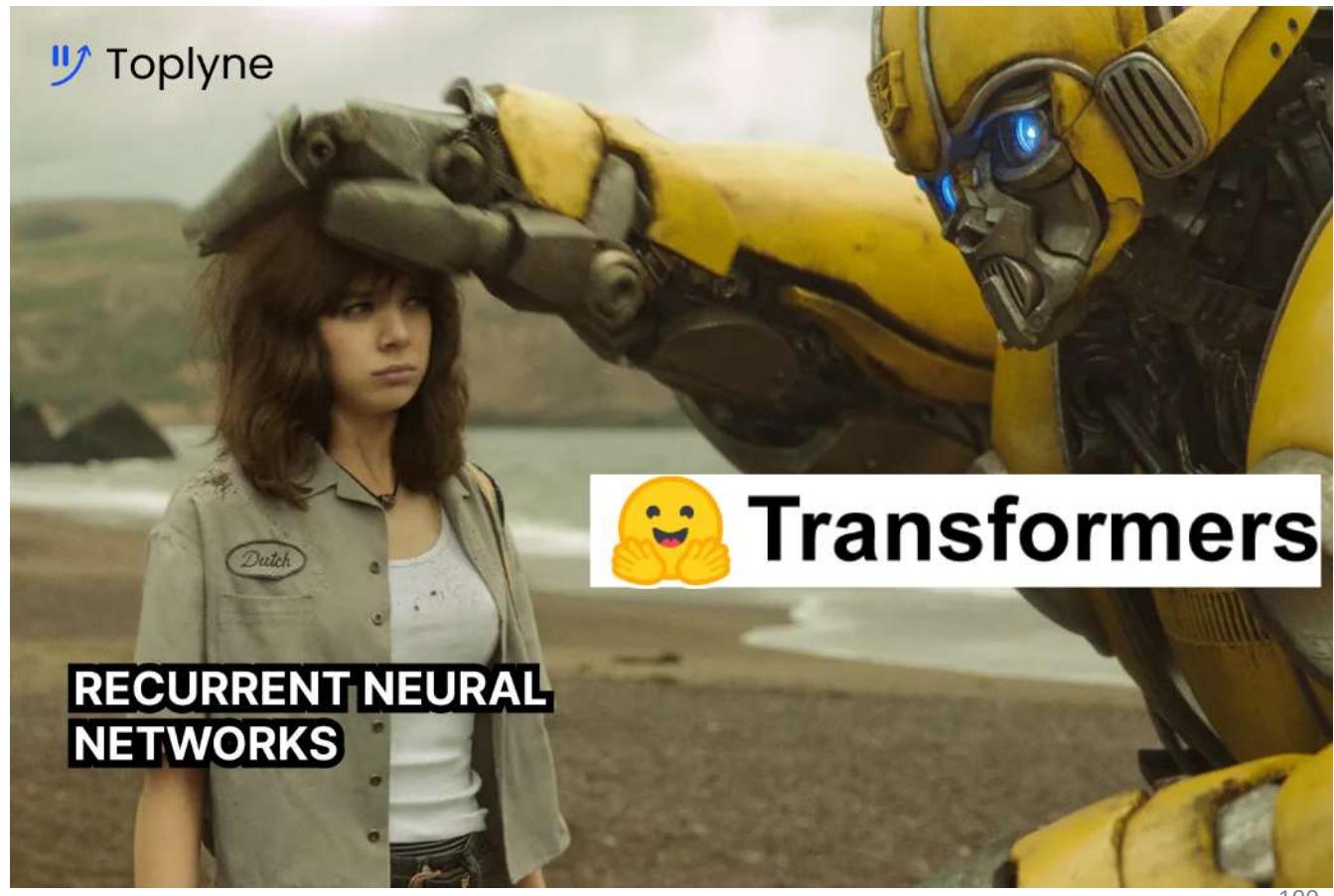
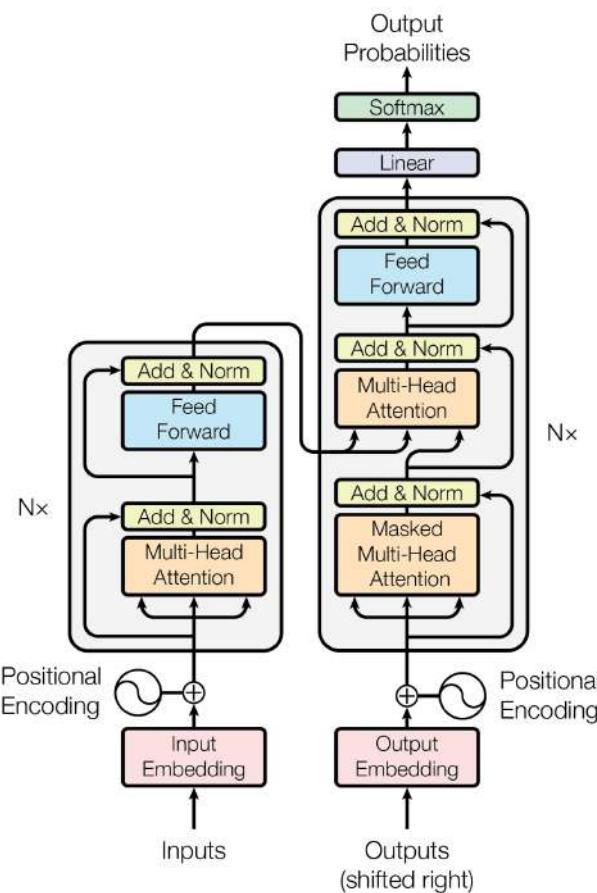
- Previous \rightarrow how much to reset $C(t-1)$
- Current [$x(t)$] reset previous memory

$$\begin{aligned}
 \text{g}(t) \quad \text{Candidate cell} &\rightarrow \tilde{c}^{(t)} = \tanh(W_c[T_r * c^{(t-1)} + x^{(t)}] + b_c) \\
 \text{i}(t) \quad \text{Update gate} &\rightarrow T_u = \sigma(W_u[c^{(t-1)} + x^{(t)}] + b_u) \\
 \text{f}(t) \quad \text{Reset gate} &\rightarrow T_r = \sigma(W_r[c^{(t-1)} + x^{(t)}] + b_r) \\
 \text{C}(t) \quad \text{Cell state} &\rightarrow c^{(t)} = T_u * \tilde{c}^{(t)} + (1 - T_u) * c^{(t-1)} \\
 \text{y}(t) \quad \text{Activation / Output} &\rightarrow a^{(t)} = \tilde{c}^{(t)} \quad \text{Update current memory}
 \end{aligned}$$

Bidirectional LSTM



Rise of Transformer (2017)



+

Appendix