



CHULA ENGINEERING COMPUTER
Foundation toward Innovation



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



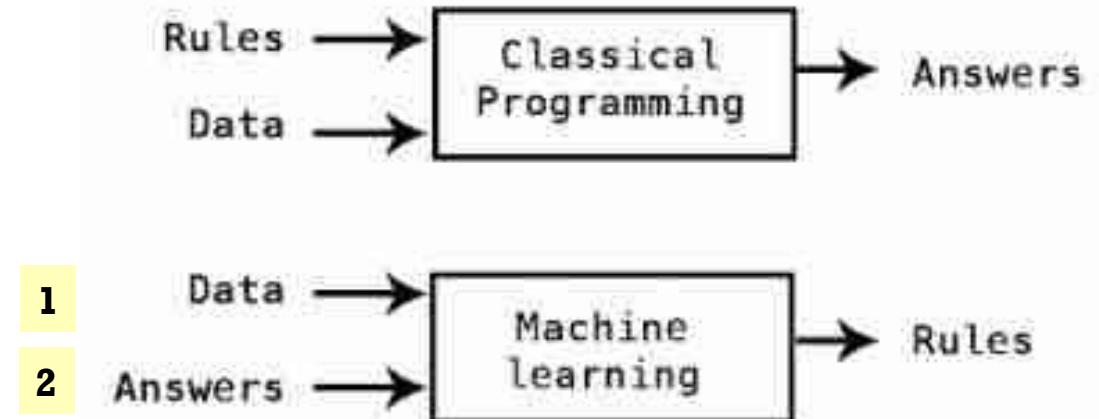
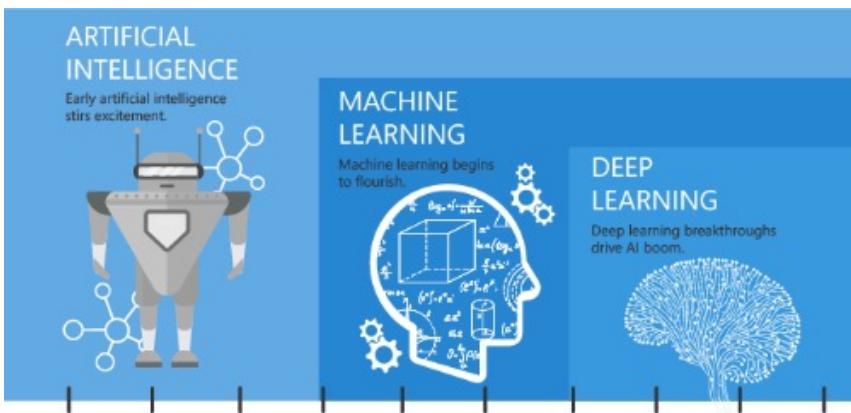


Introduction to Deep Learning



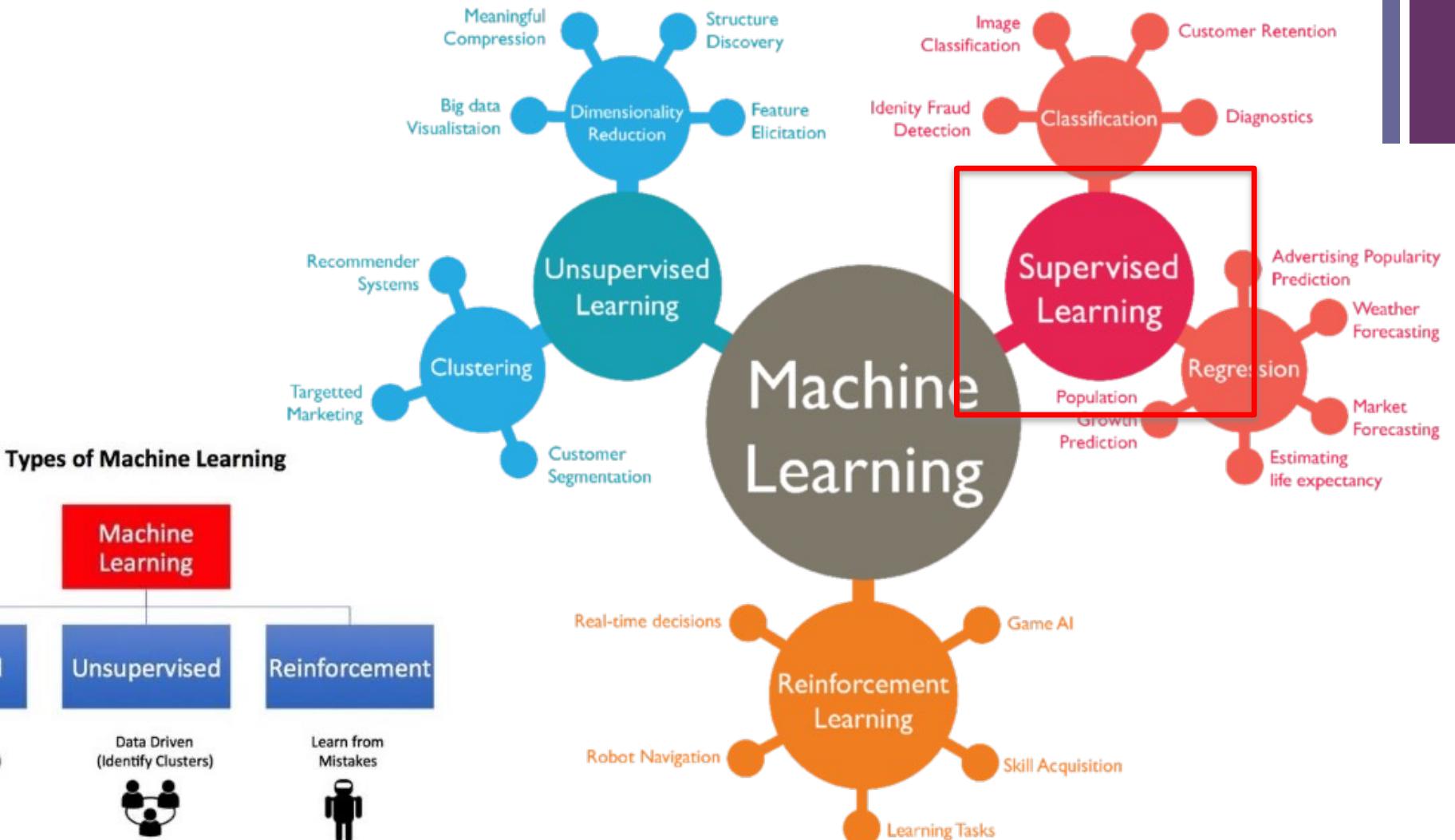
AI = Automation

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



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

+ Machine Learning





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



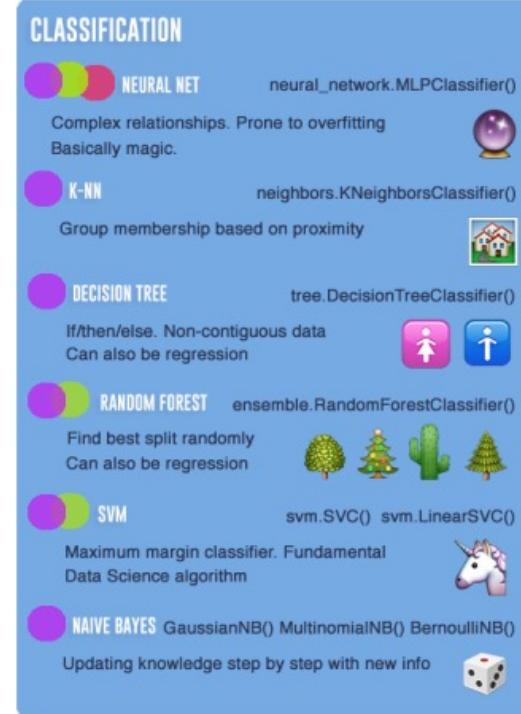
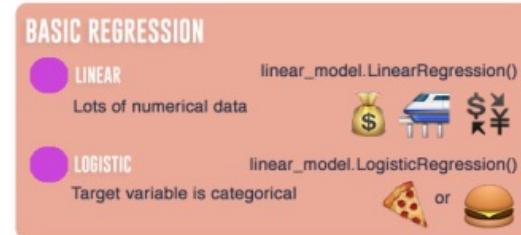
Age	Gender	BodyTemp	Cough	Corona
25	Male	40	No	?

Application: Corona Prediction



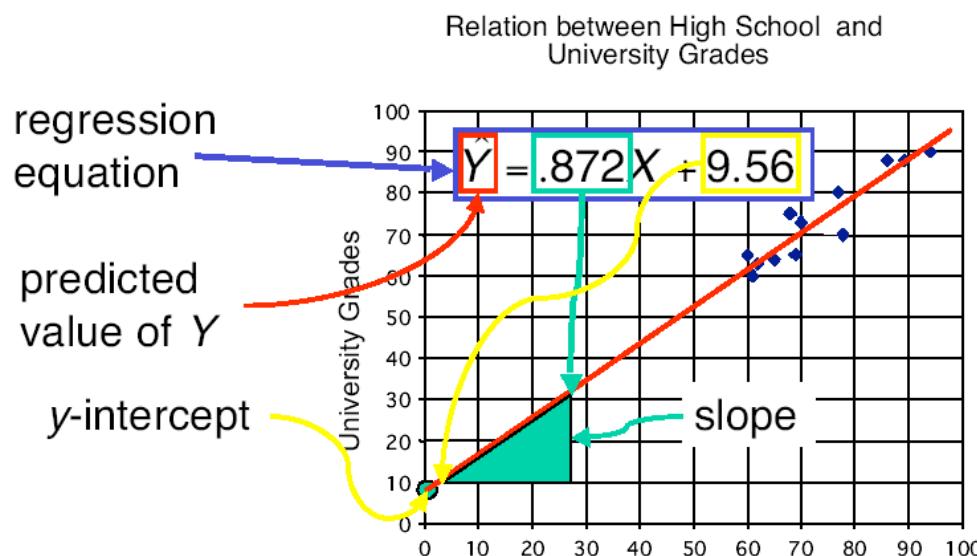
Prediction algorithms

- 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	

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

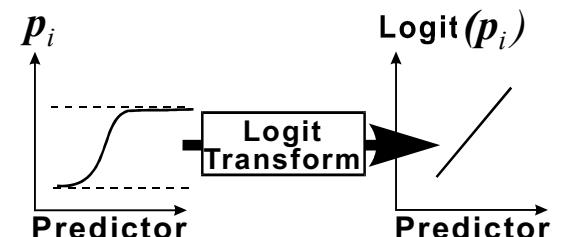
$$\text{Logit_score} = 0.01 - 0.3 * \text{Age} + 0.2 * \text{Gender} + 0.2 * \text{Temp} + 0.9 * \text{Cough}$$

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

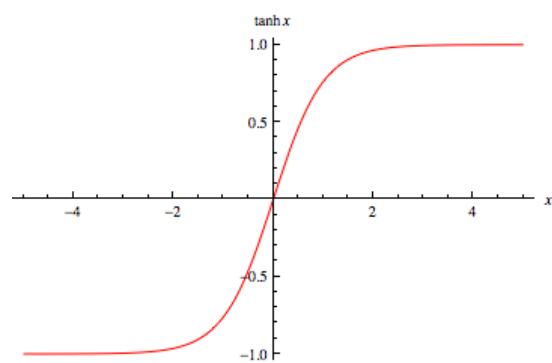
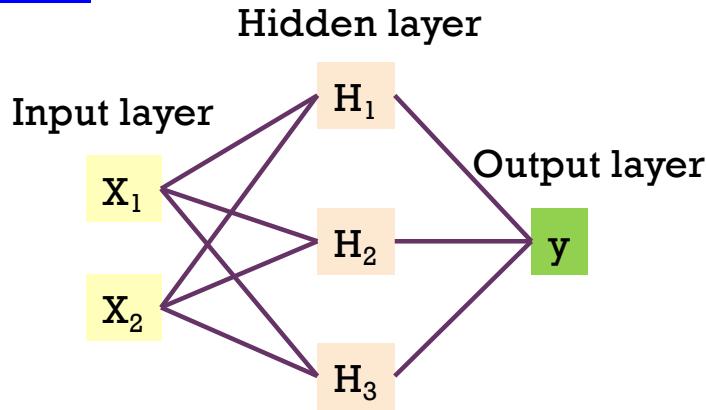


$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$



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?

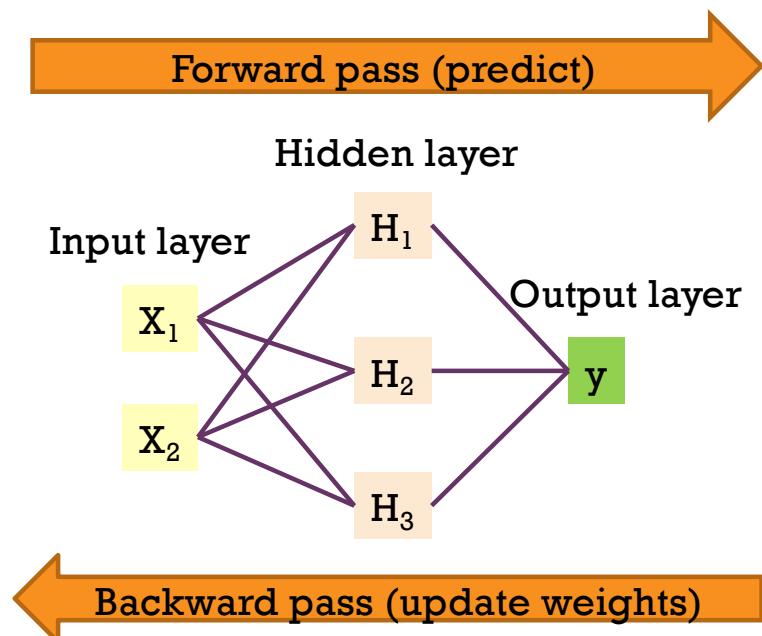
- Converge (no change in loss)
- Max epochs

Important Params:

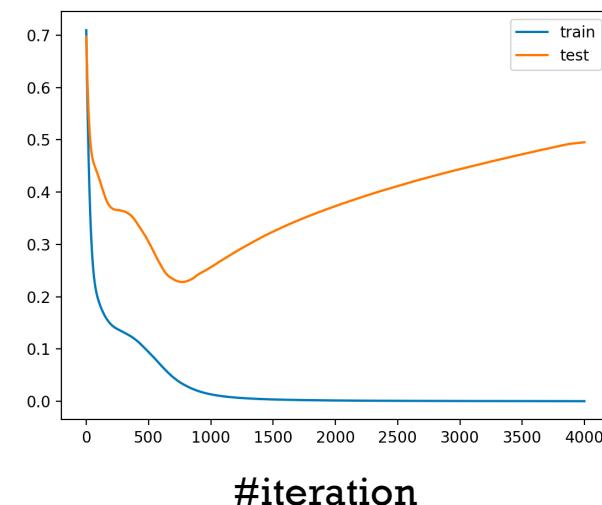
- #hidden units, #hidden layers
- Learning rate, Momentum, decay
- Seed number, etc.



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



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

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

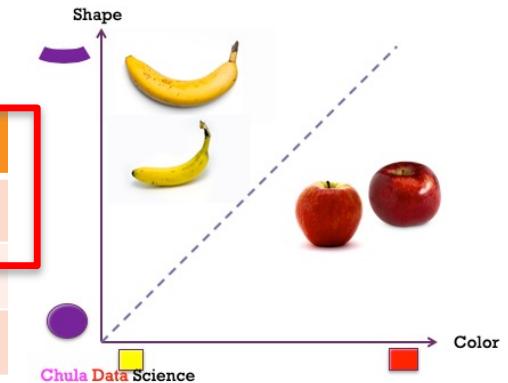
$$H_2 = \tanh(\hat{w}_{20} + \hat{w}_{21}x_1 + \hat{w}_{22}x_2)$$

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



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



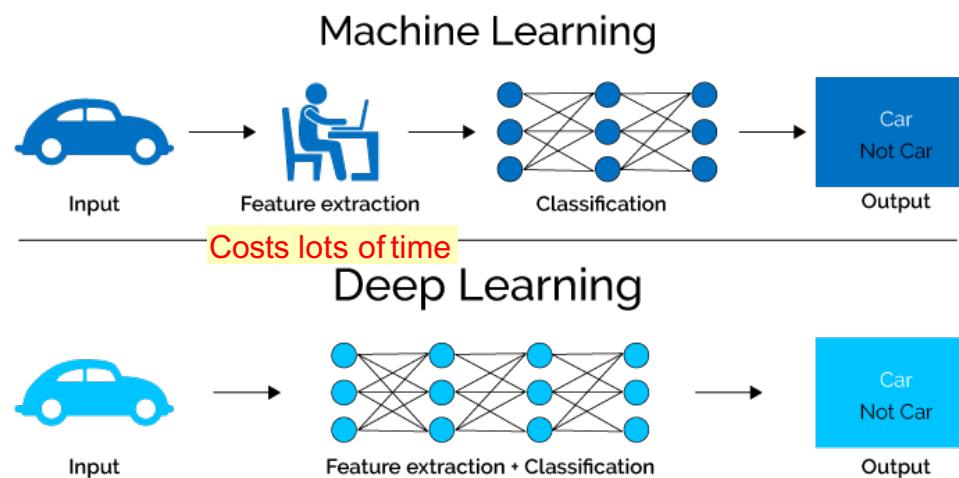
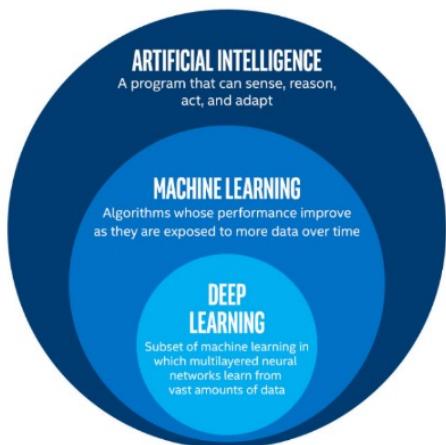
What is Deep Learning (DL)?



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.





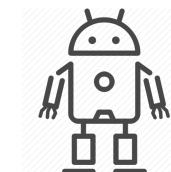
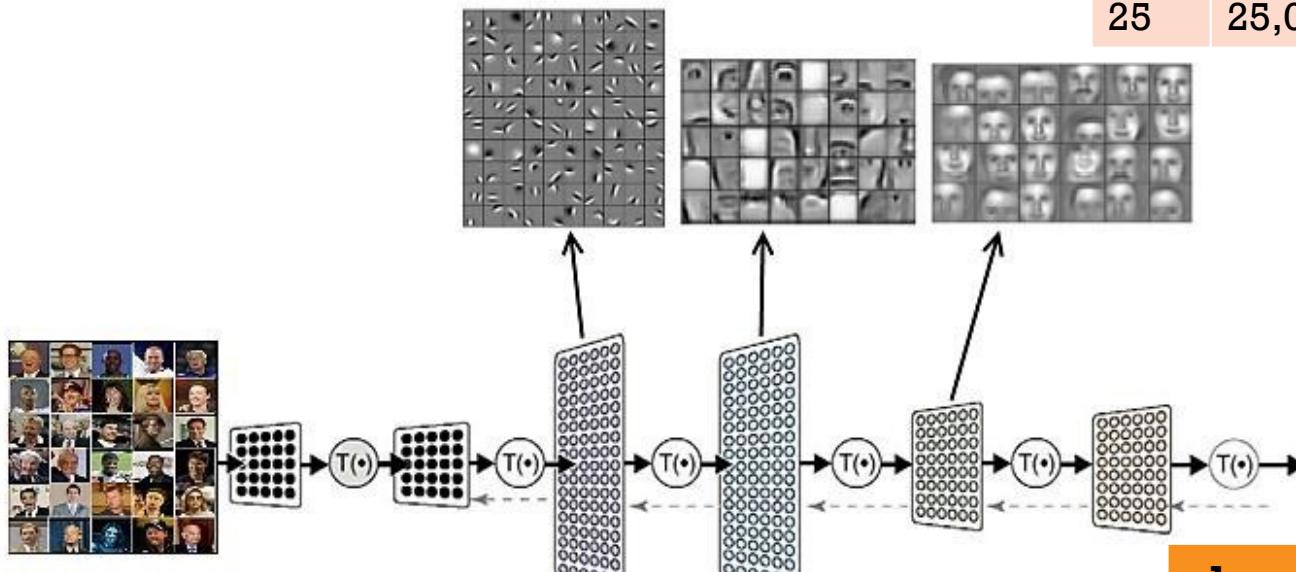
Deep Learning – Basics (cont.)

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.

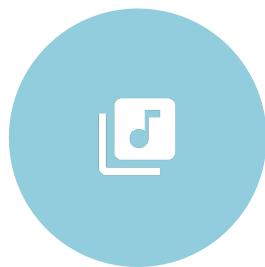


Age	Income	Gender	Province	Corona
25	25,000	Female	Bangkok	Yes



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

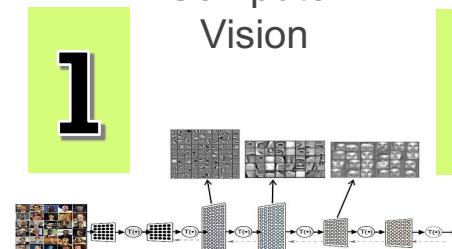
Deep Learning Application



Speech
Recognition



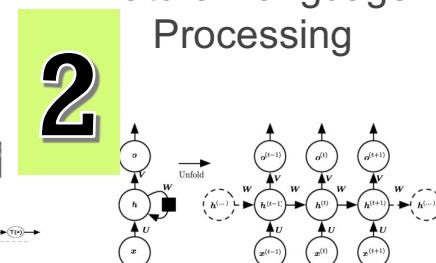
Computer
Vision



CNN

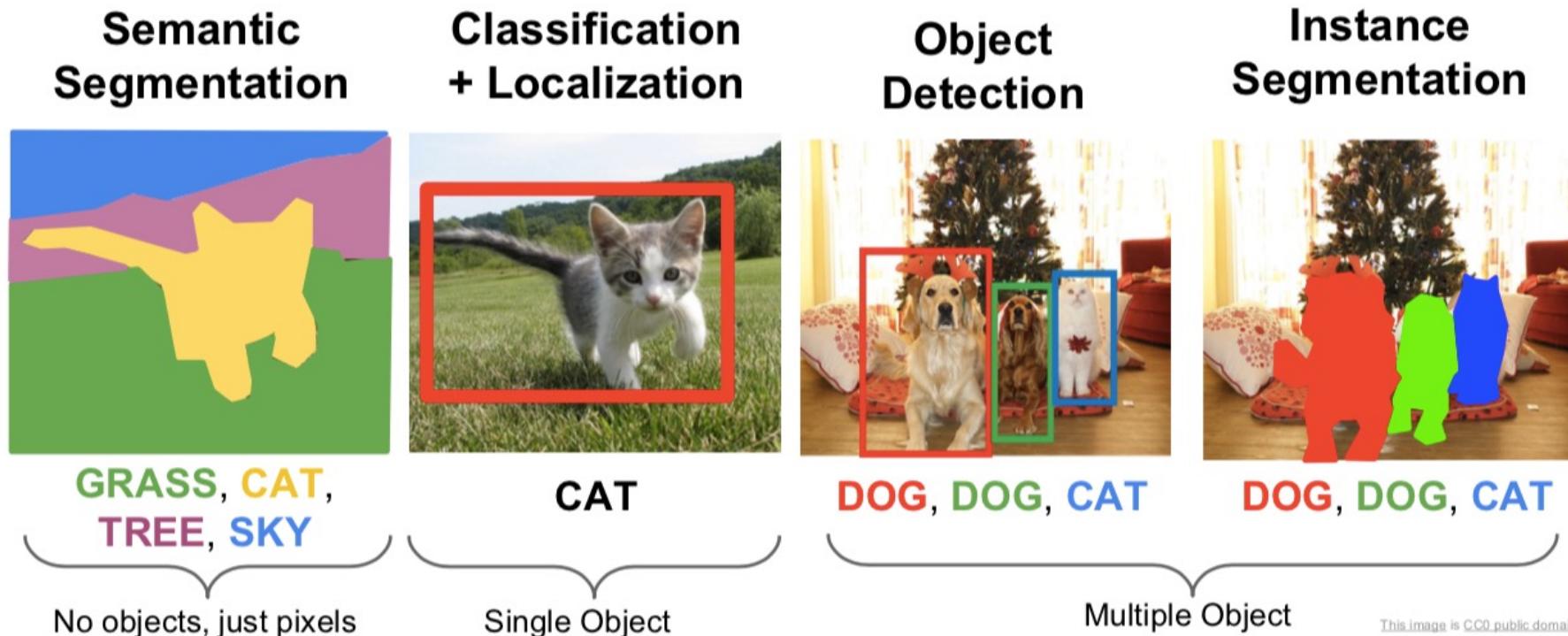


Natural Language
Processing



RNN

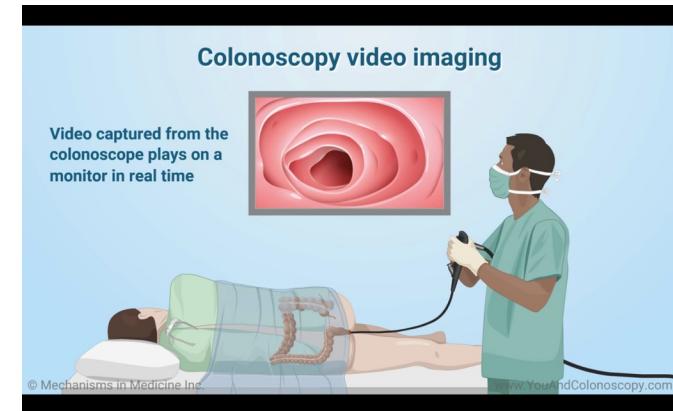
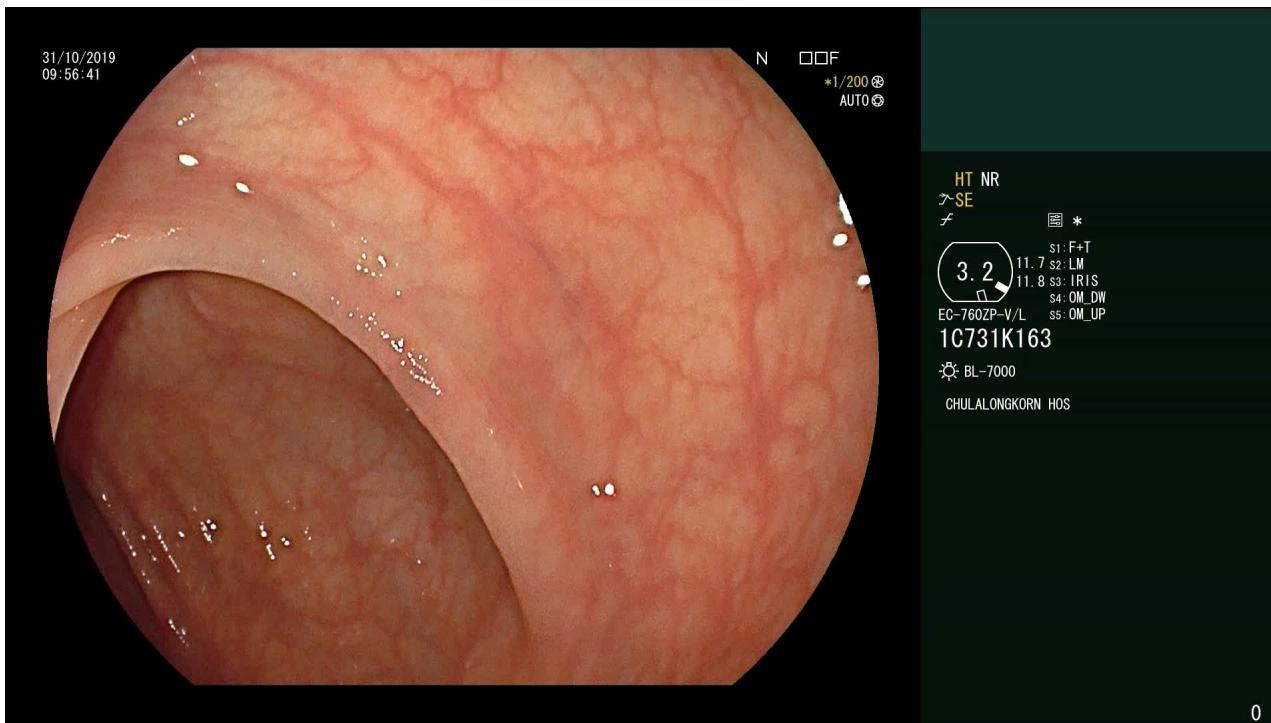
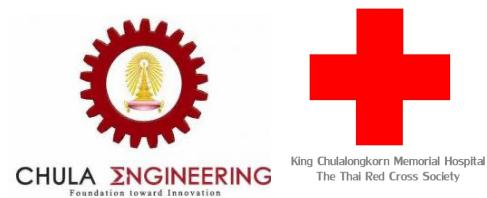
Type of image tasks



Face Recognition



Smart Medical Diagnosis



+

CNN



Outline

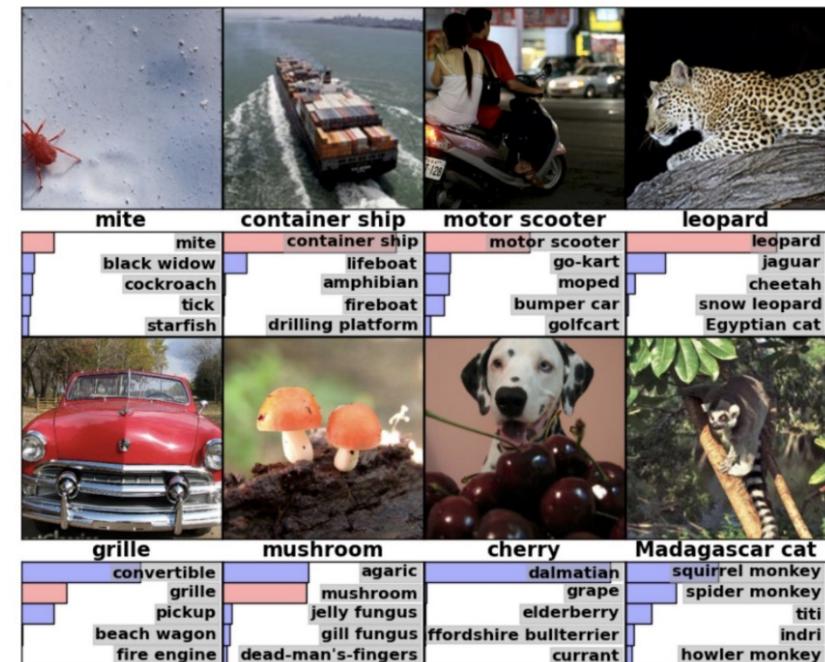
- Introduction
- Basic Image Processing
- Overview of CNN
- Image Processing Tasks with SOTA
- Case Study in Polyp Detection

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.

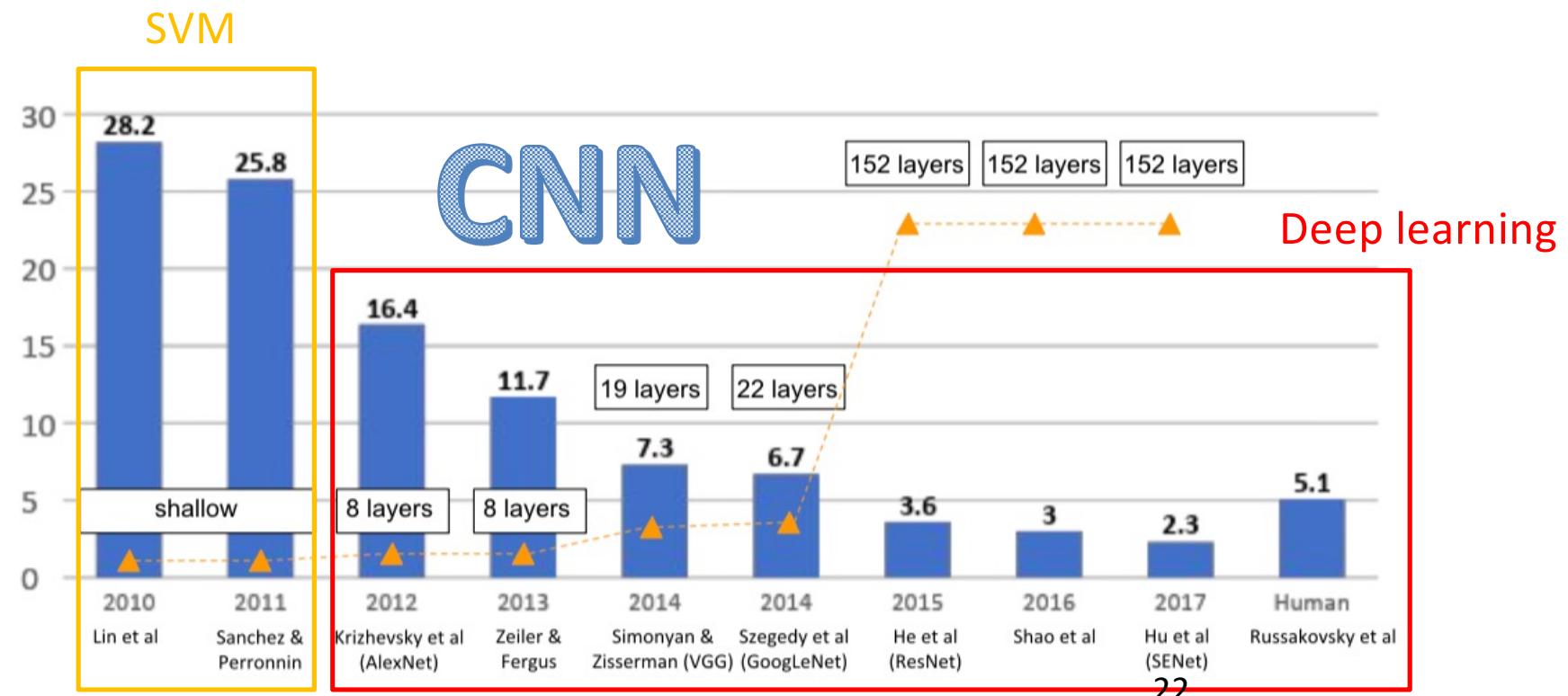


The Image Classification Challenge:
1,000 object classes
1,431,167 images
ImageNet 2017 is the last challenge.

ILSVRC

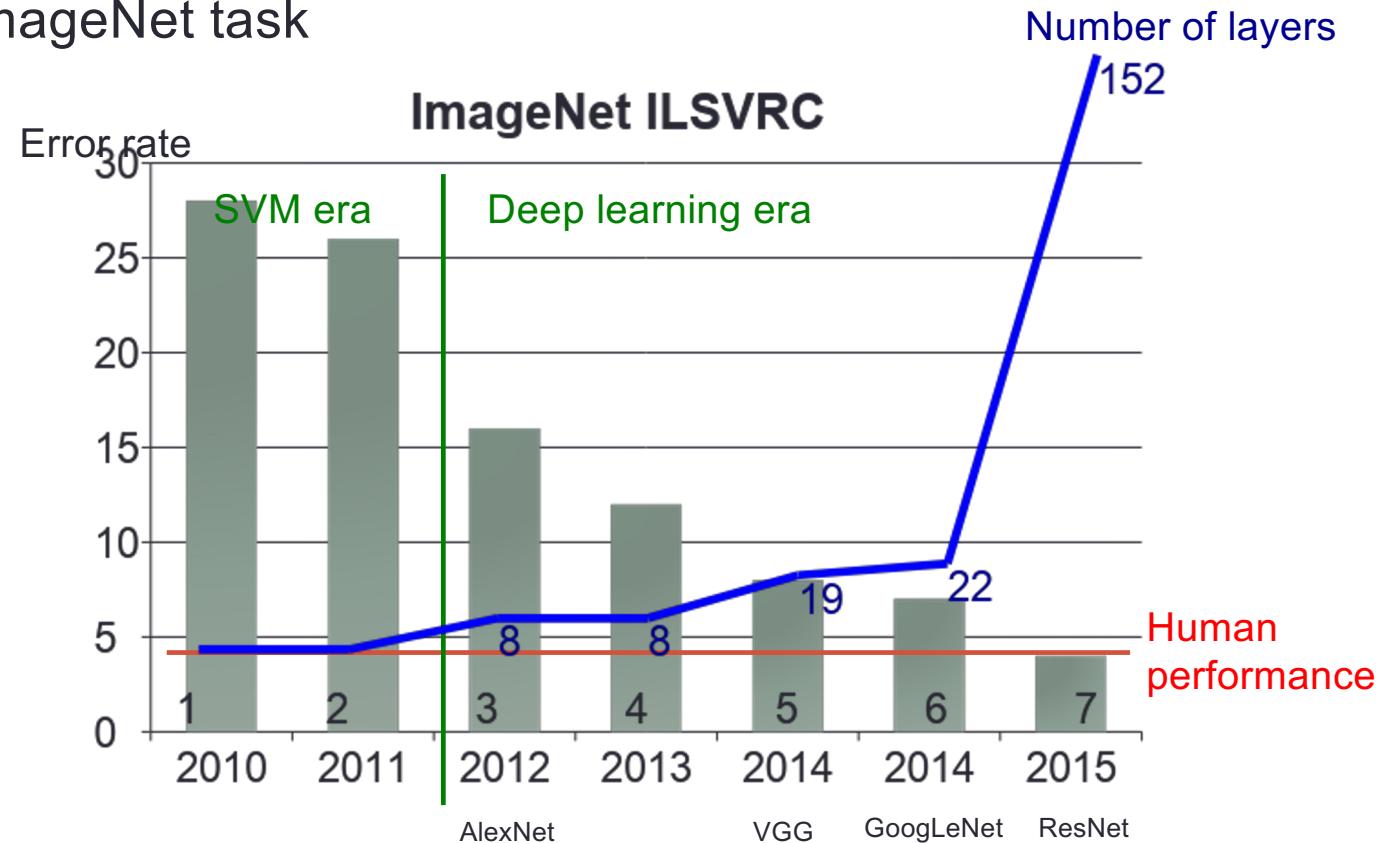


ImageNet Large Scale Visual Recognition Challenge (ILSVRC winners): From SVM to Deep Learning



Wider and deeper networks (Beyond Human)

- ImageNet task

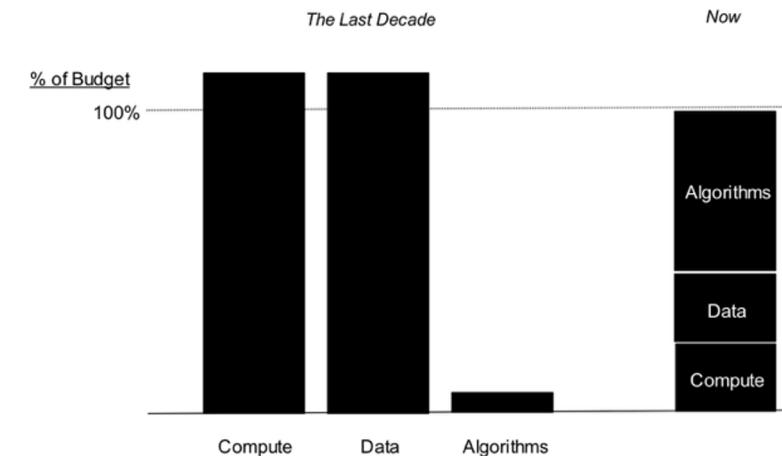


Based on the slide of Aj.Ekapol

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

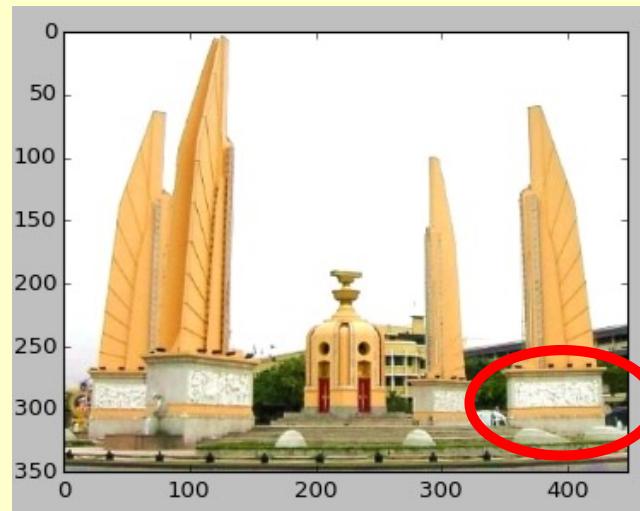
การแสดงรูปภาพใน python ด้วย matplotlib

```
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



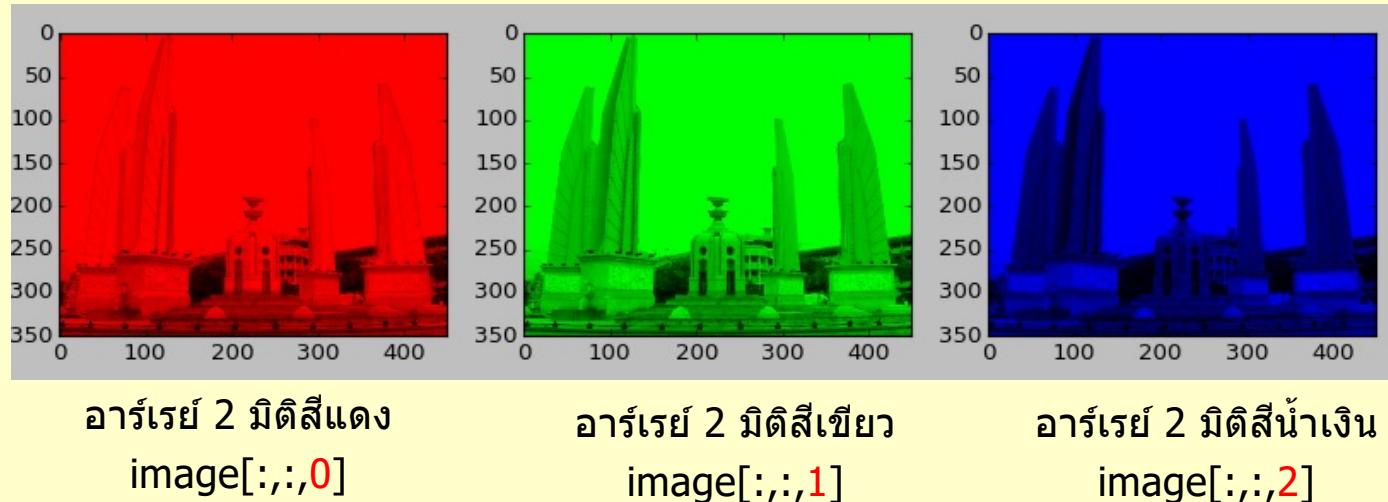
1 pixel = 1 จุดสี มี 3 สีอยู่

0.98762 | 0.72549 | 0.35294



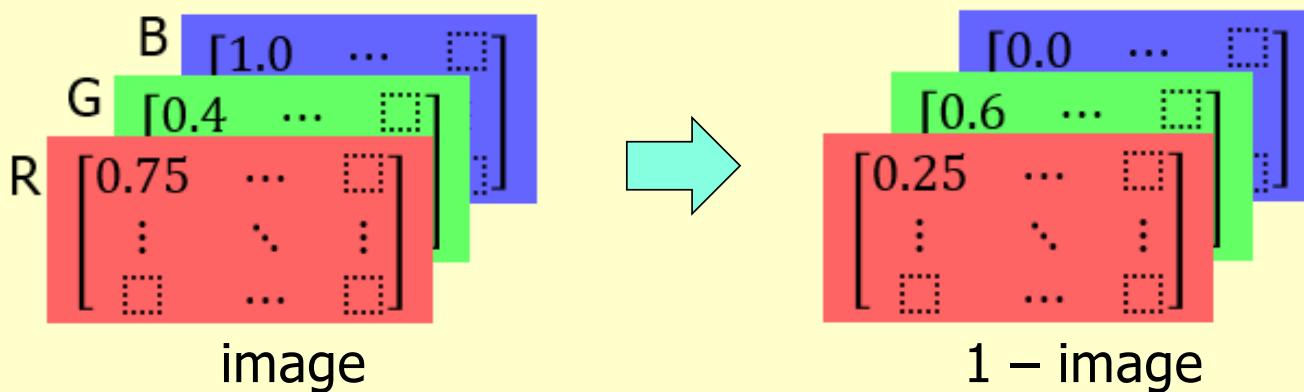
1 ภาพ เป็นอาร์เรย์ 3 มิติเก็บค่าความเข้มของ R G B

```
>>> 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

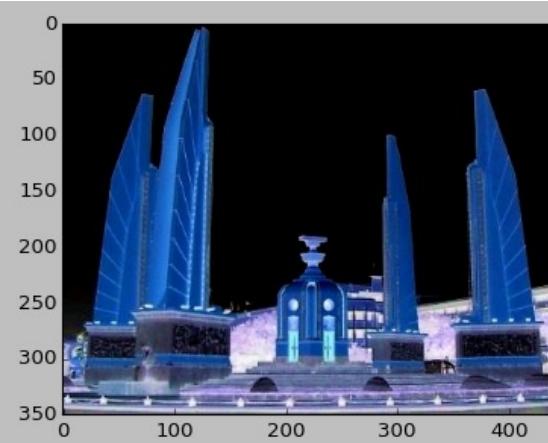
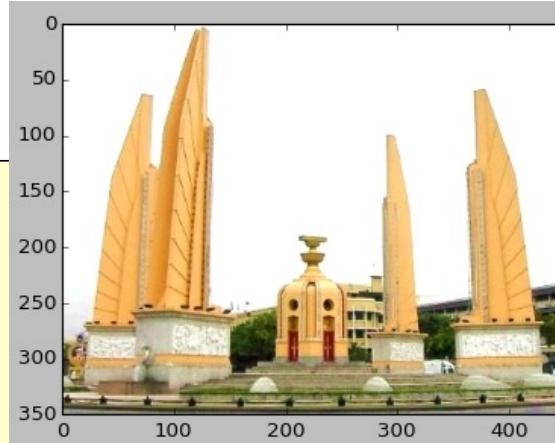
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

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



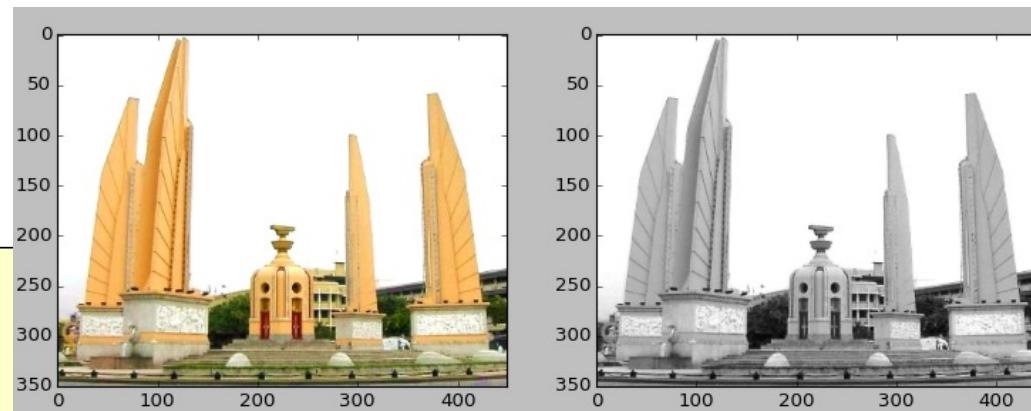
การประมวลผลภาพเบื้องต้น : ภาพสีเทา

```
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()
```

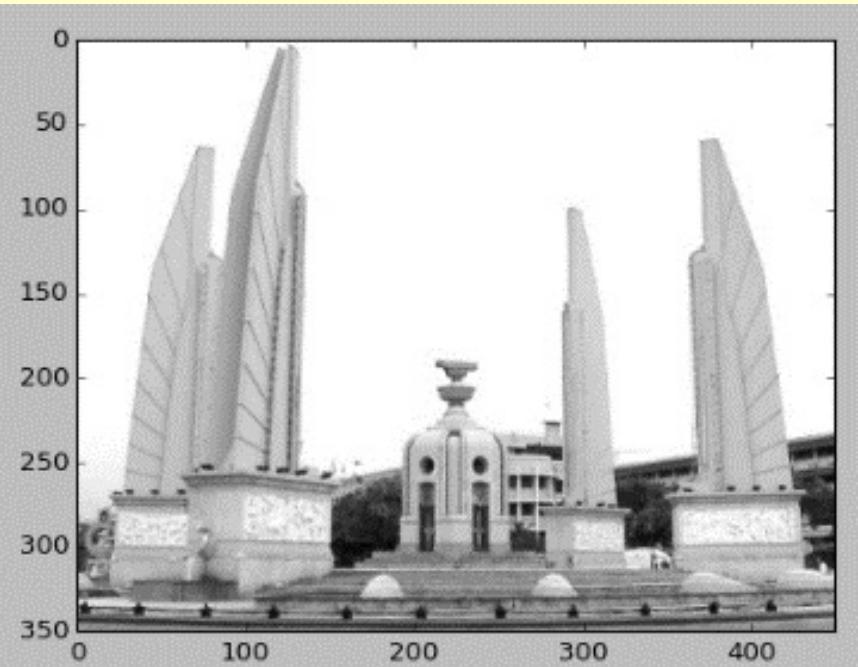
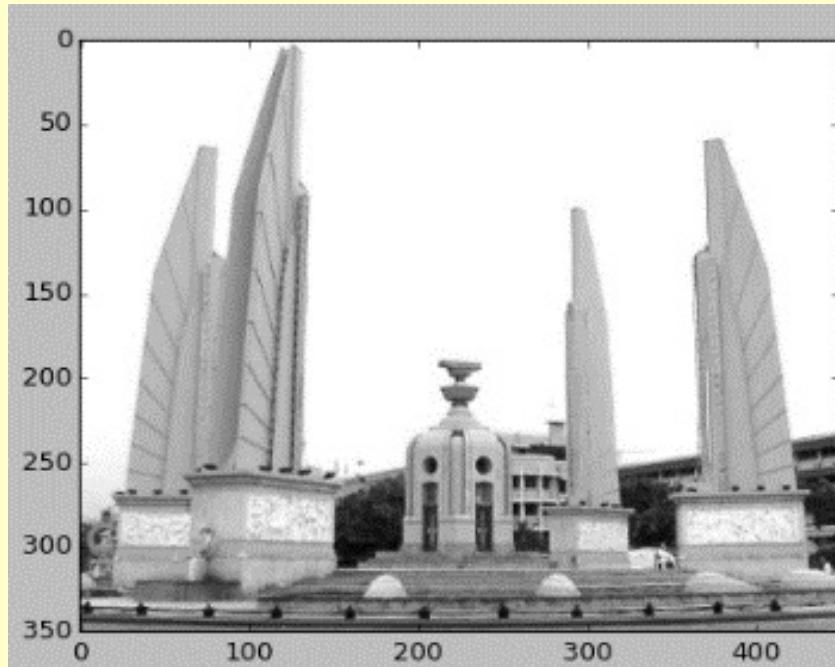
จุดสีที่ R G B มีค่าความเข้มเท่ากัน
ได้จุดสีเทา

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

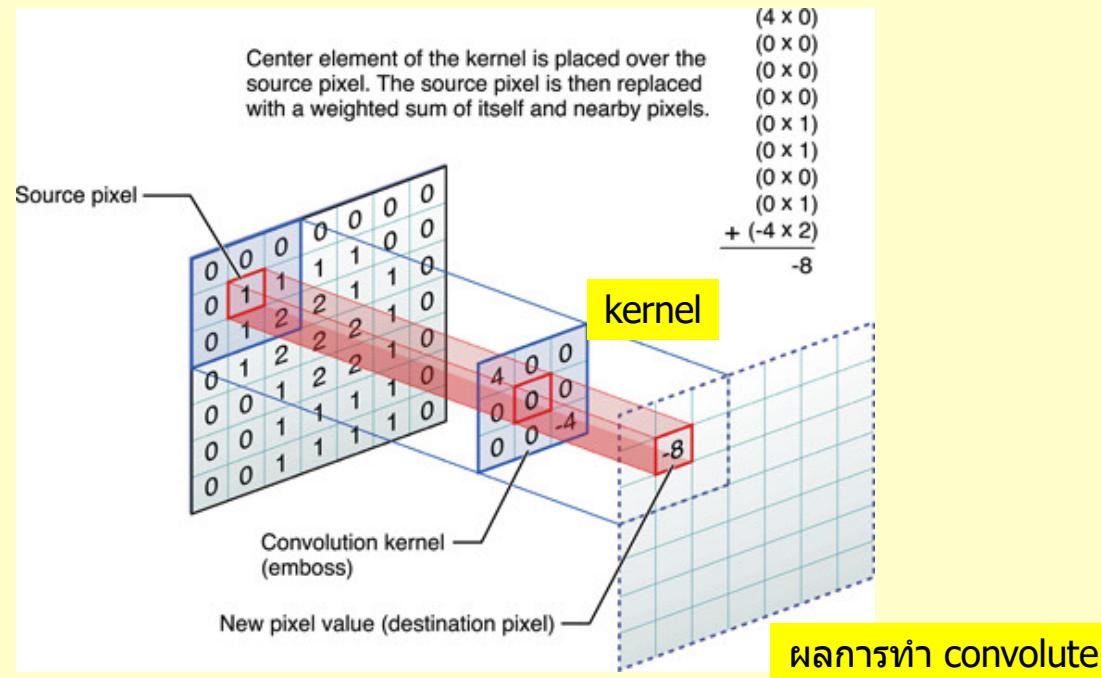


สีเทา เข้าว่าสูตรนี้ดีกว่า $0.299*R + 0.587*G + 0.114*B$

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



Convolution



เปลี่ยนค่าของเมทริกซ์ kernel
จะได้การประมวลผลภาพแบบอื่น ๆ

Image Convolution

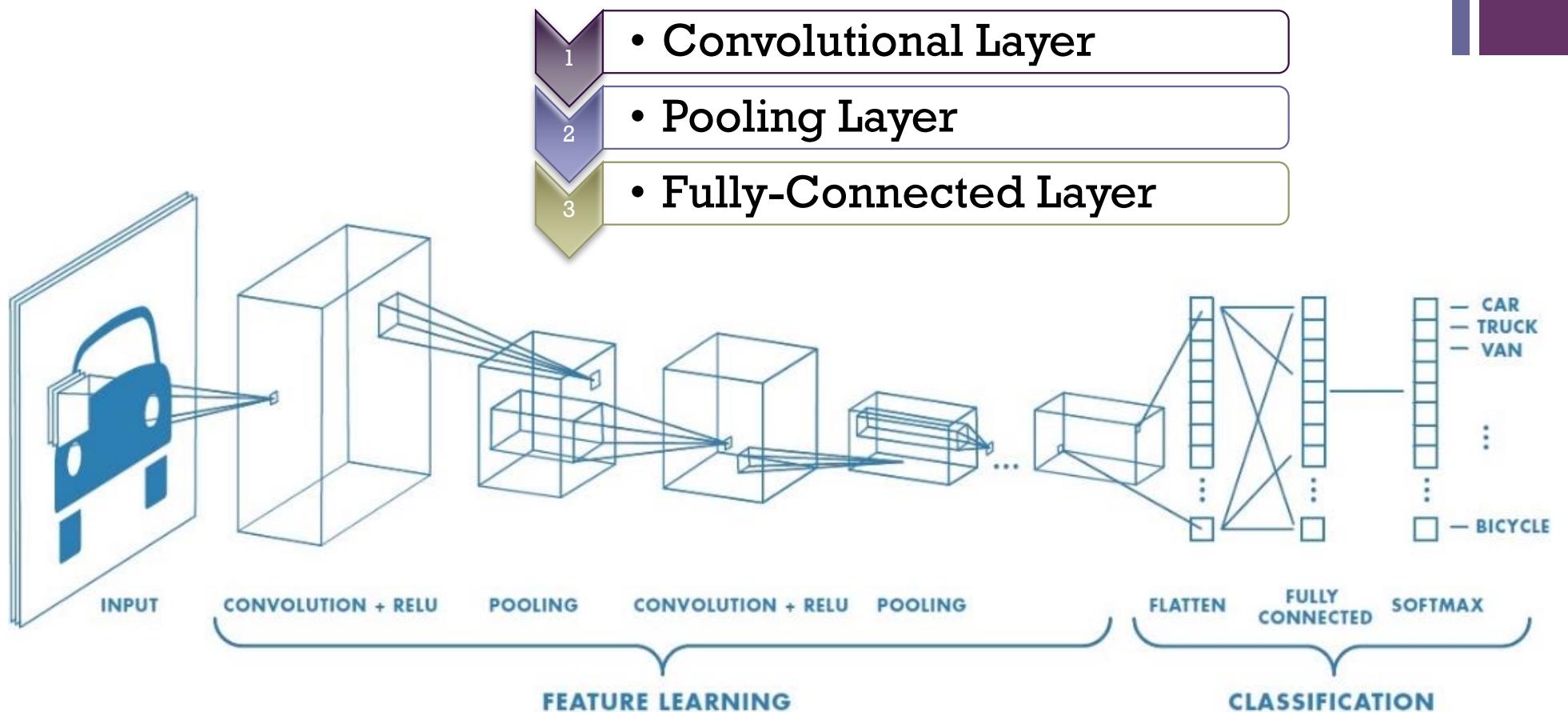
- Image Convolution คือการนำรูปภาพมาผ่านตัวกรอง (kernel) เพื่อให้ได้ผลลัพธ์ตามที่ต้องการ เช่น blur, ขยายรูป, จับขอบรูป เป็นต้น

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

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

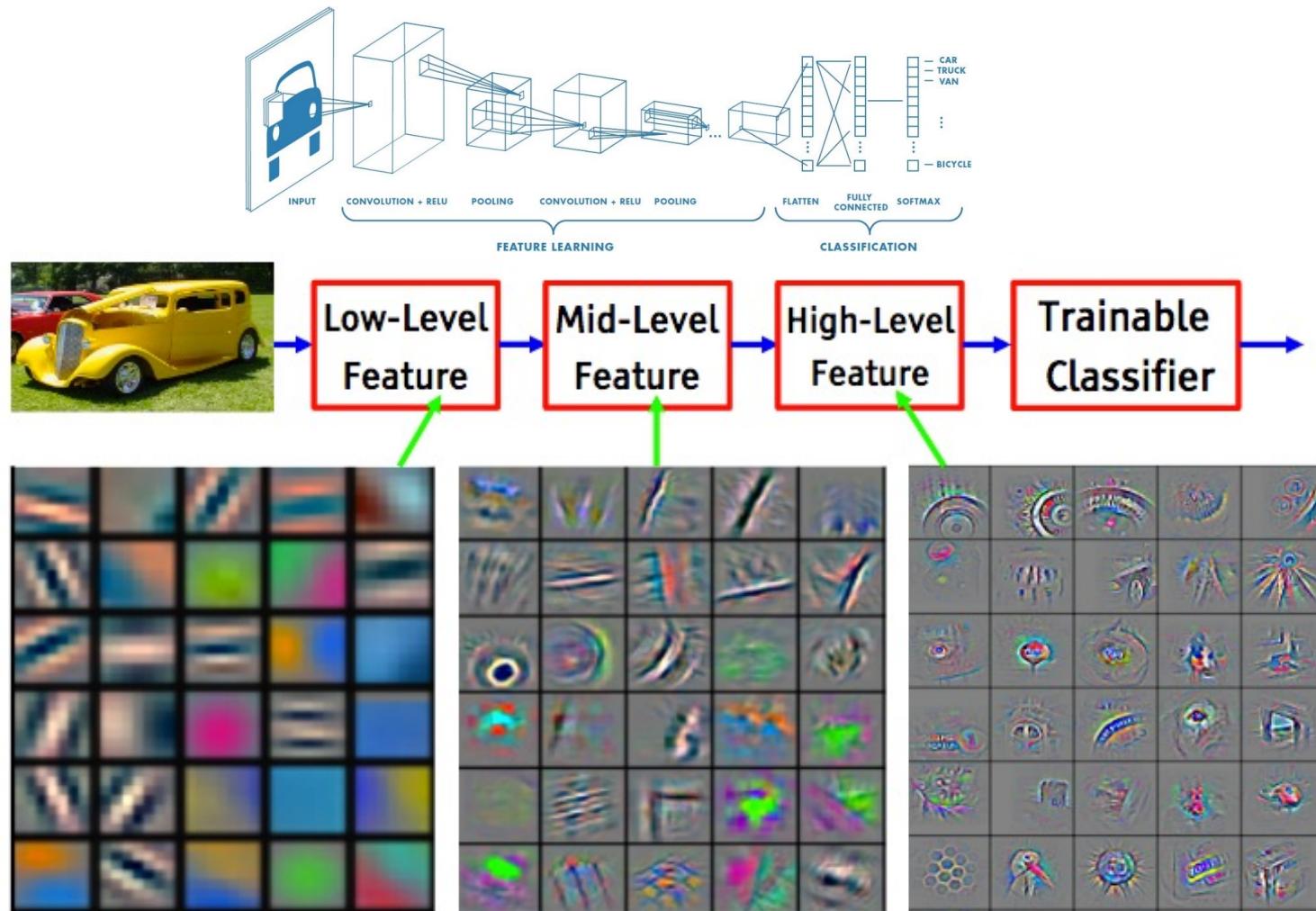


Convolutional Neural Networks (CNN)





CNN (cont.)

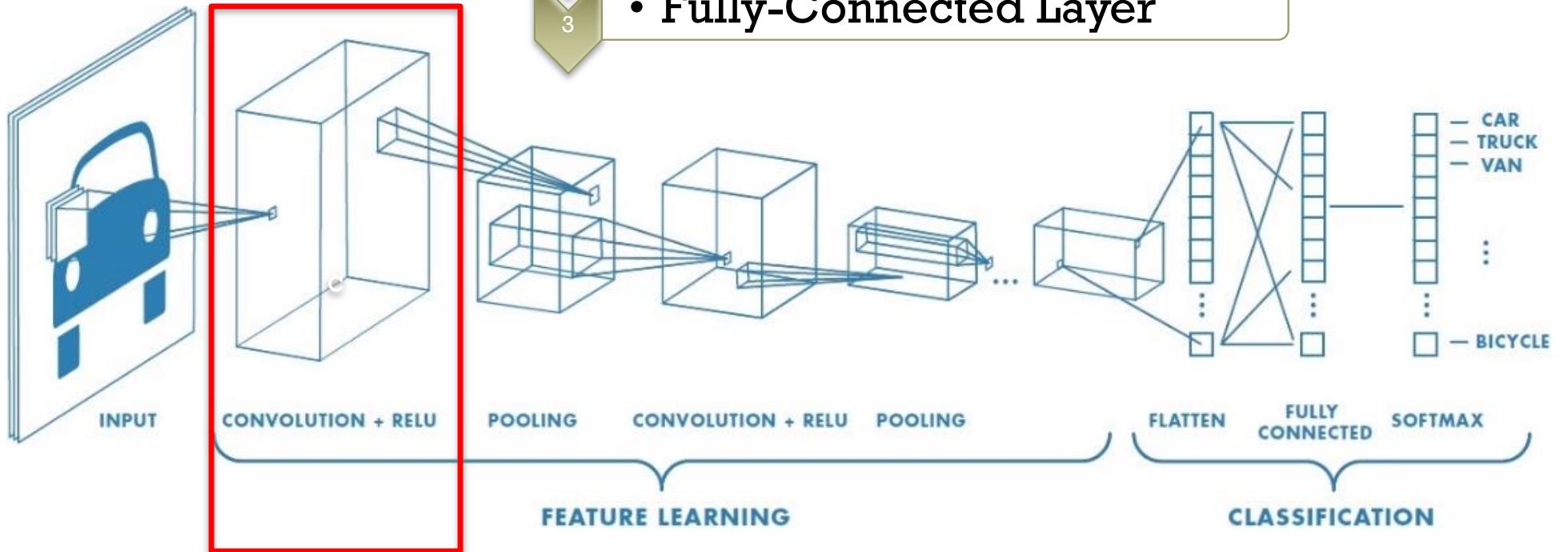




Convolutional Neural Networks (CNN)

1) Convolutional Layer

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer





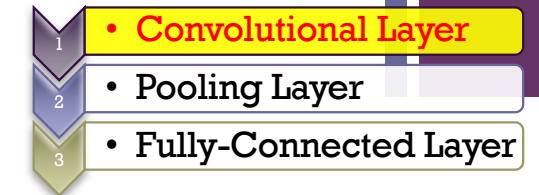
Layer1: Convolutional Layer

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

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$	$w_0[:, :, 0]$
0 0 0 0 0 0 0	1 0 0
0 1 0 2 2 2 0	-1 0 0
0 0 0 0 0 0 0	0 -1 1
0 2 0 2 2 2 0	-1 -1 0
0 1 0 0 0 0 0	-1 -1 1
0 1 0 0 2 1 0	1 0 0
0 0 0 0 0 0 0	-1 0 0

$x[:, :, 1]$	$w_0[:, :, 1]$
0 0 0 0 0 0 0	1 1 0
0 1 0 1 0 1 0	0 -1 1
0 0 0 0 0 1 0	1 1 1
0 0 0 2 0 0 0	1 1 0
0 2 0 0 0 0 0	0 0 0
0 0 0 1 0 0 0	0 0 0
0 0 0 0 0 0 0	0 0 0

$x[:, :, 2]$	$w_0[:, :, 2]$
0 0 0 0 0 0 0	1 1 0
0 0 2 2 0 0 0	0 -1 1
0 0 0 0 0 1 0	1 1 1
0 1 1 2 0 2 0	0 0 0
0 2 0 0 0 0 0	0 0 0
0 0 1 1 0 1 0	0 0 0
0 0 0 0 0 0 0	0 0 0

Filter W_0 (3x3x3)

$w_0[:, :, 0]$
1 0 0
-1 0 0
0 -1 1
-1 -1 0
-1 -1 1
1 0 0
-1 0 0

$w_0[:, :, 1]$
1 1 0
0 -1 1
1 1 1
1 1 0
0 0 0
0 0 0
0 0 0

$w_0[:, :, 2]$
1 1 0
0 -1 1
1 1 1
1 1 0
0 0 0
0 0 0
0 0 0

Filter W_1 (3x3x3)

$w_1[:, :, 0]$
1 -1 0
-1 0 0
0 1 -1
0 1 -1
0 1 -1
0 1 -1
0 1 -1

$w_1[:, :, 1]$
0 1 -1
-2 4 -1
0 0 0
3 3 0
0 1 1
-2 2 -1

$w_1[:, :, 2]$
0 -1 0
1 1 -1
-1 0 1
0 0 0
0 0 0
0 0 0

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

$b_1[:, :, 0]$
0

Bias b_0 (1x1x1)

$b_0[:, :, 0]$
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

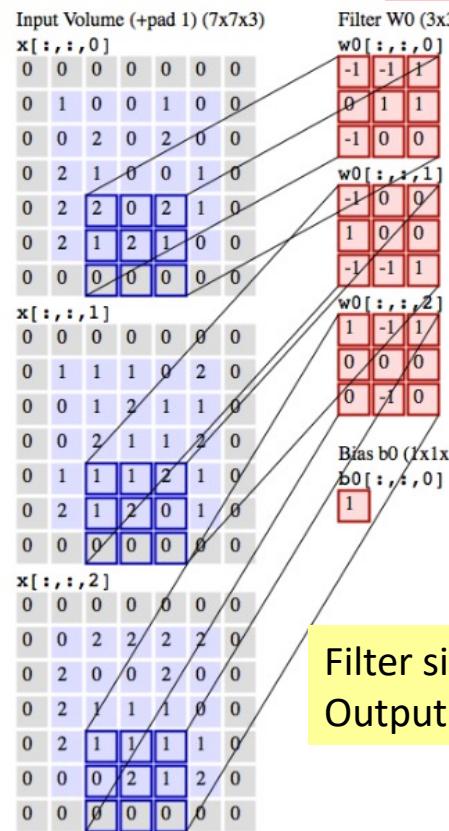
Image

1	0	1
0	1	0
1	0	1

Convolved
Feature
(feature map)

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

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



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

Summary: **4 parameters** for convolutional layer

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

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

(feature map1)

(feature map2)

R: $2(-1) + 2(1) + 2(1) + 1(1) = 3$

G: $1(-1) + 1(1) = 0$

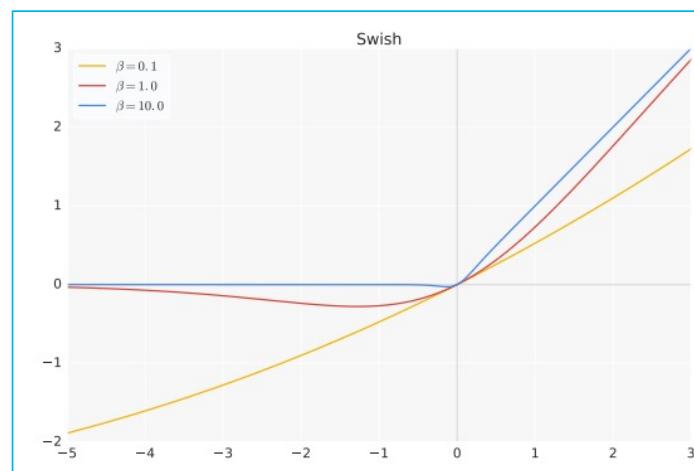
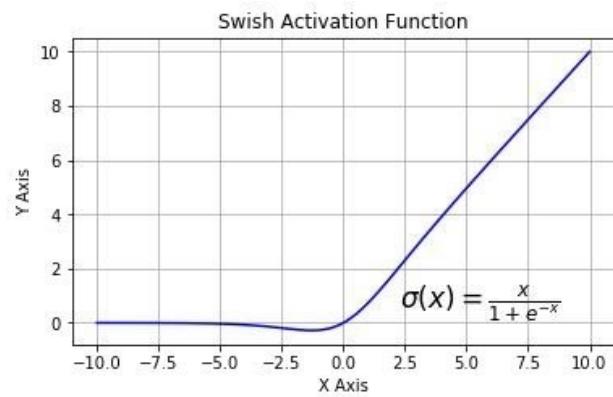
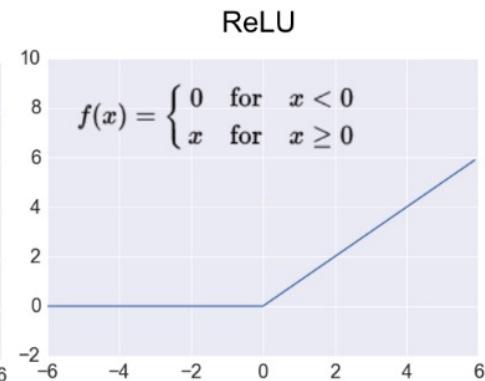
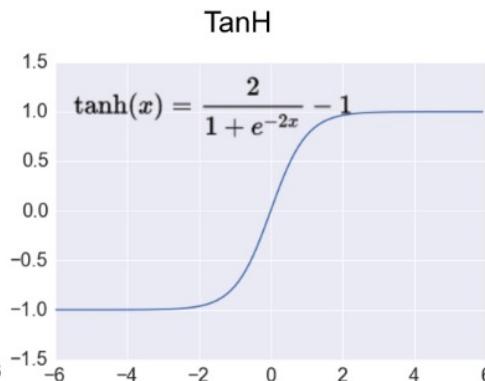
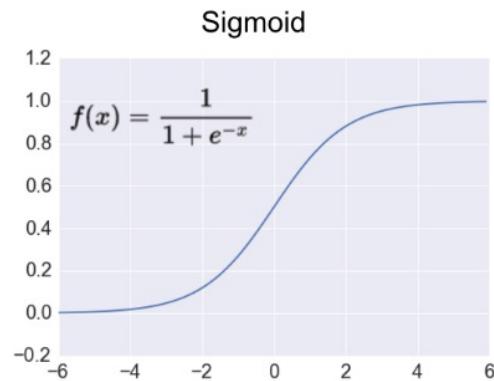
B: $1(1) + 1(-1) + 1(1) = 1$

Ans : $4 + 1 = 5$



CNNs: 1) Convolutional Layer (cont.)

Non-Linear Activation Function

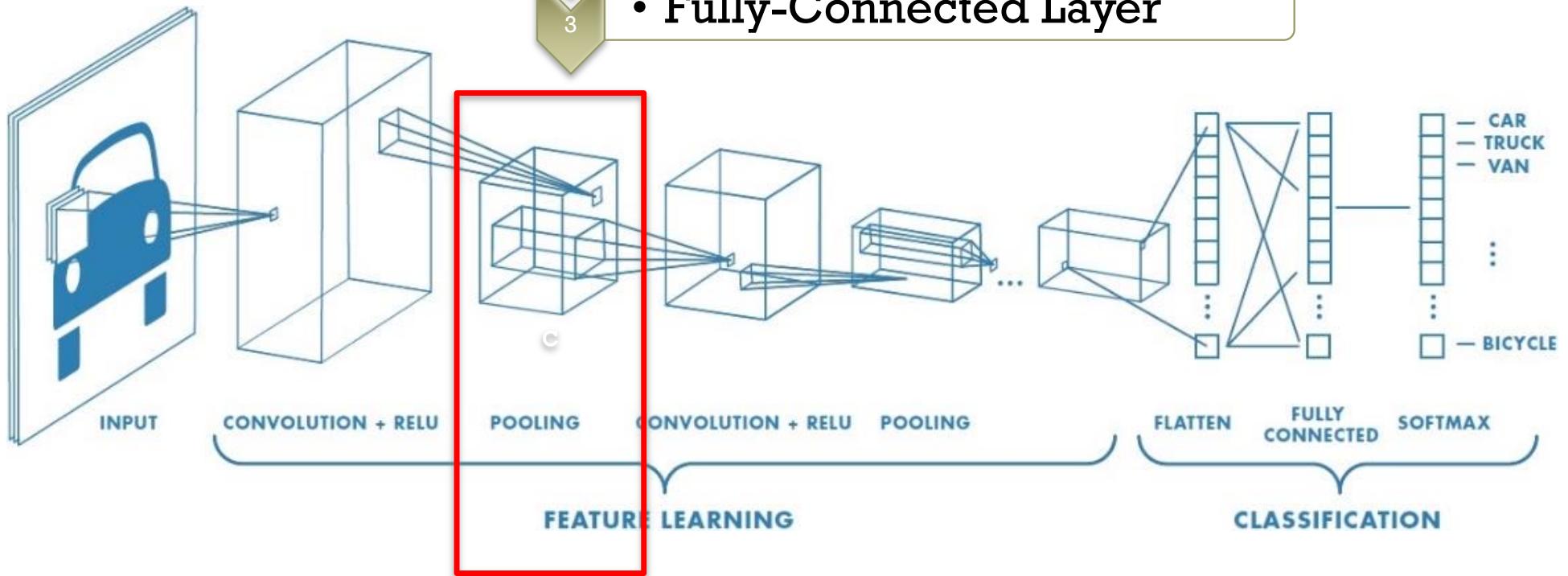




Convolutional Neural Networks (CNN)

2) Pooling Layer

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer

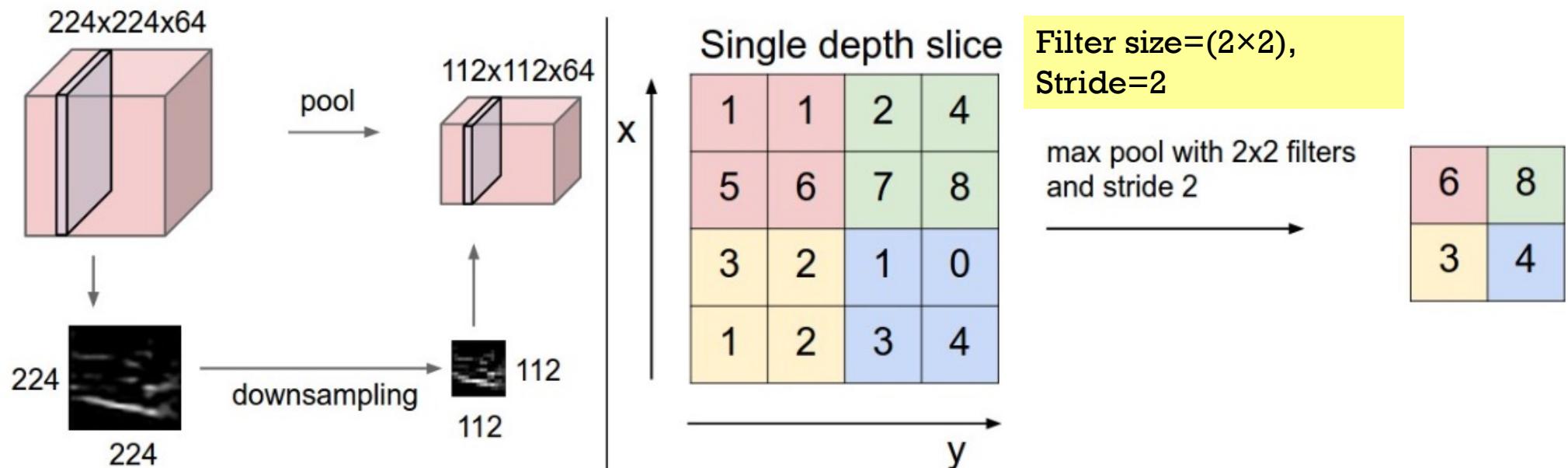


CNNs: 2) Pooling Layer

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

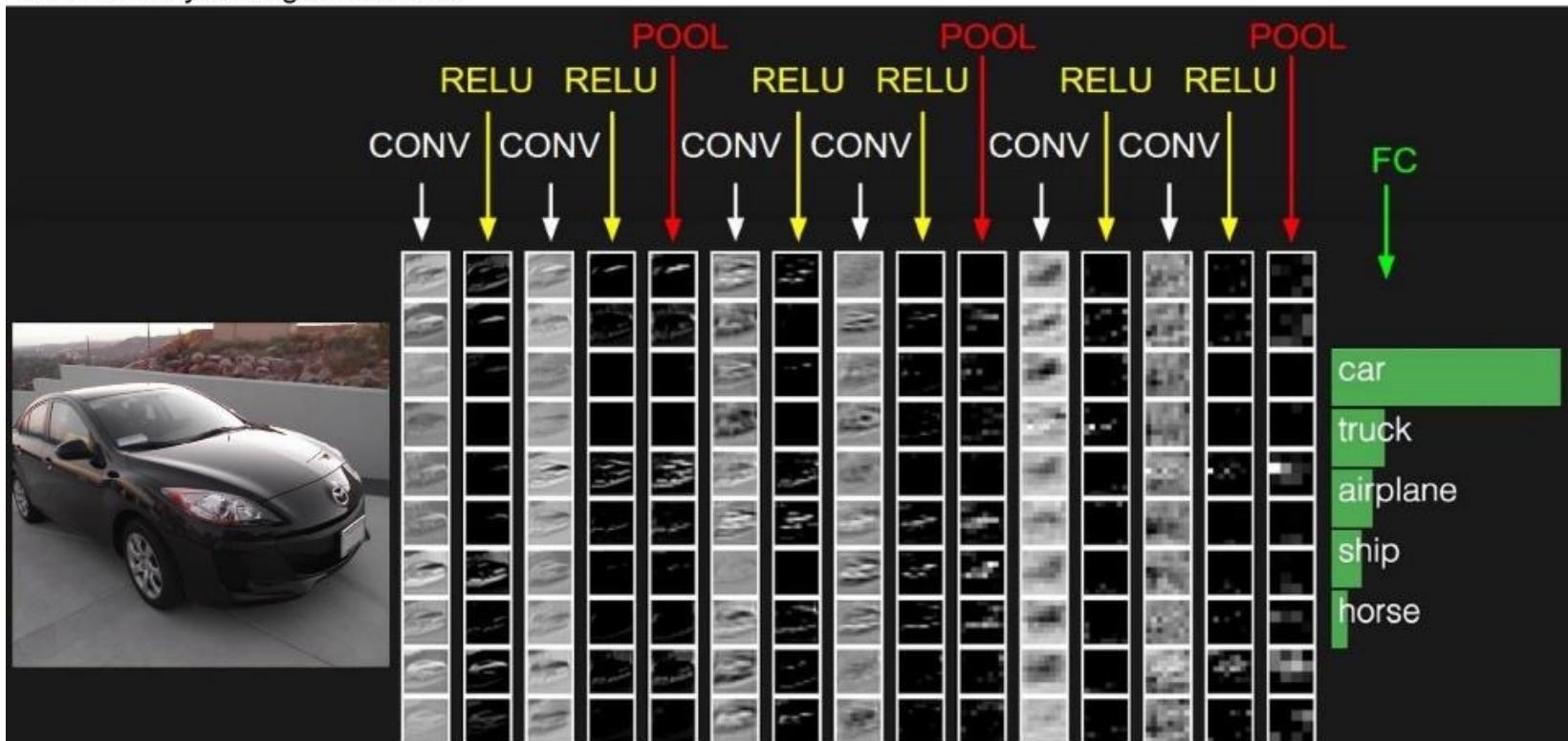
Summary: **2 parameters** for pooling layer
• Filter size, Stride

- Feature selection (reduction); downsampling





two more layers to go: POOL/FC

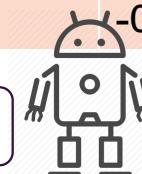




Convolutional Neural Networks

Embedding Vector

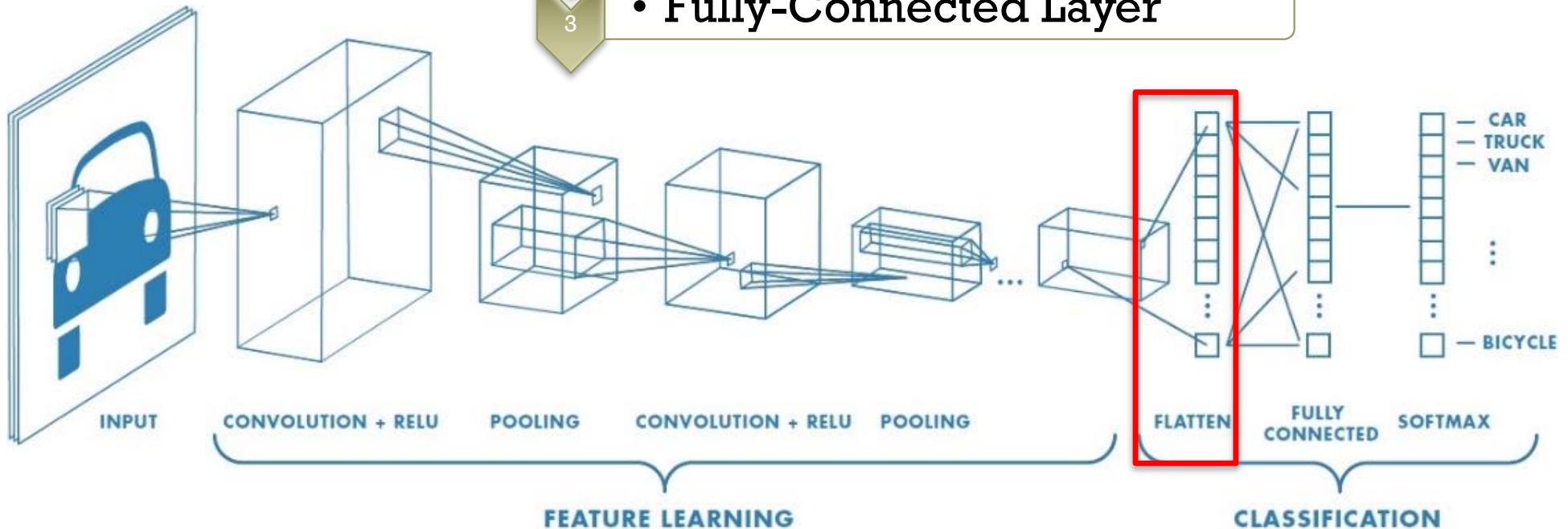
x1	x2	x3	x4	Corona
0.7	0.2	-0.5	-0.1	Yes



- Convolutional Layer

- Pooling Layer

- Fully-Connected Layer

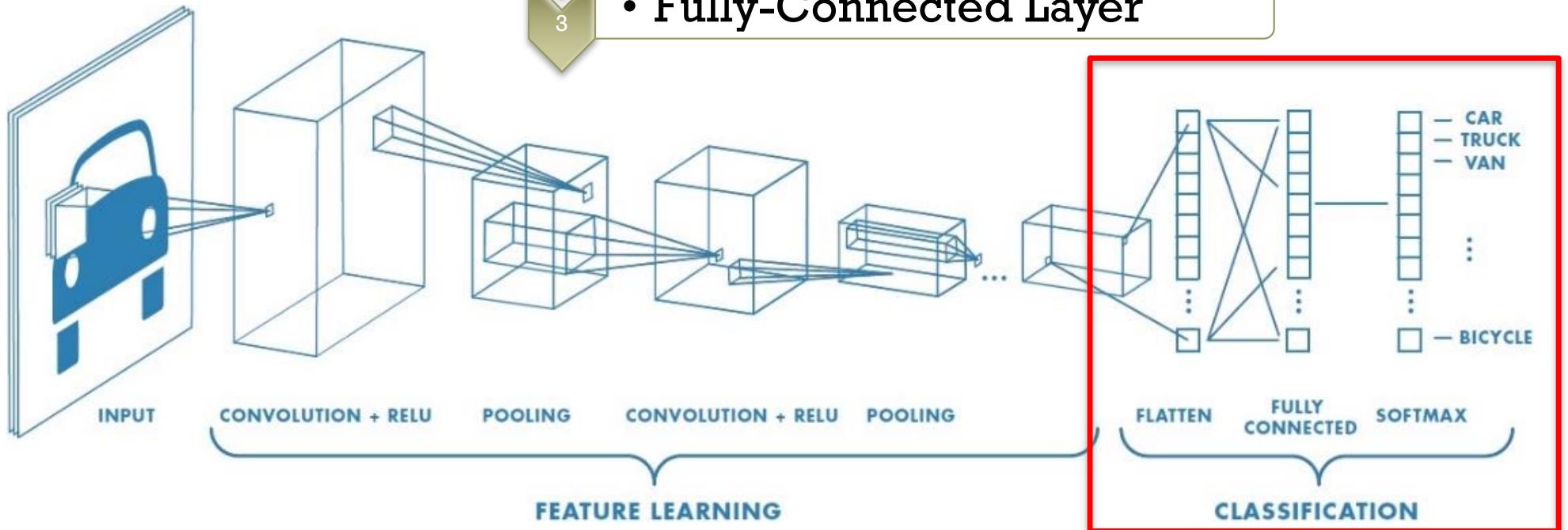




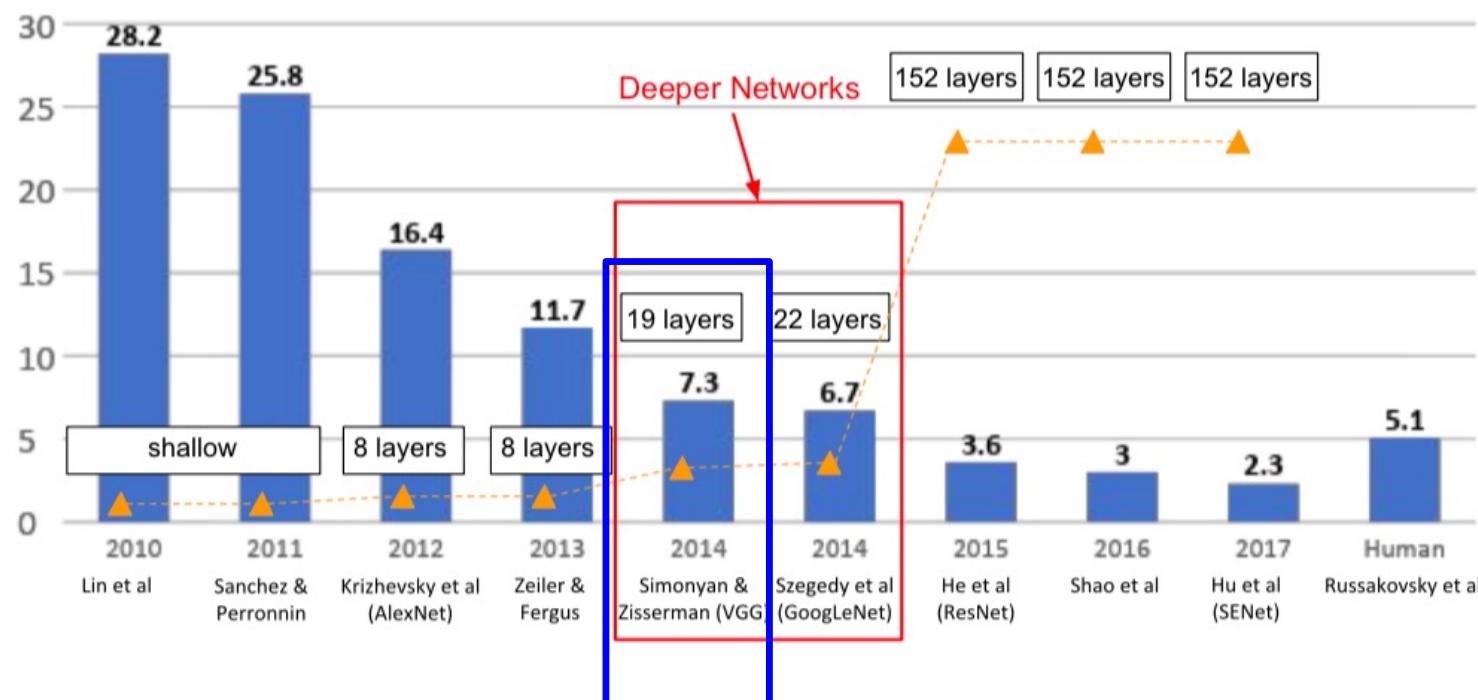
Convolutional Neural Networks (CNN)

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)

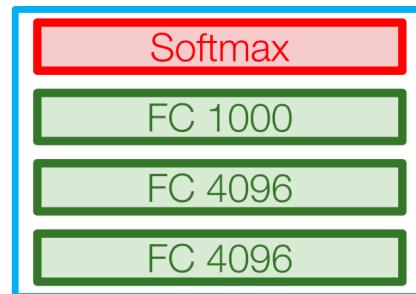
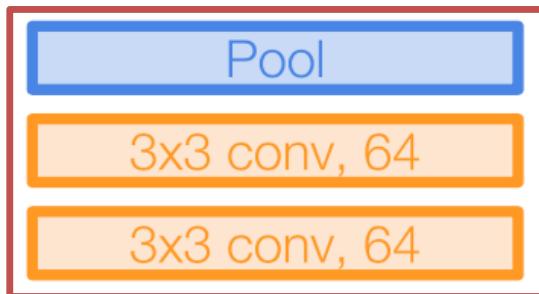
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

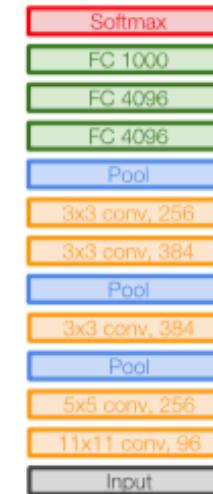
8 layers (AlexNet) → 16-19 layers (VGG16Net)

Only 3x3 CONV Stride 1, pad 1

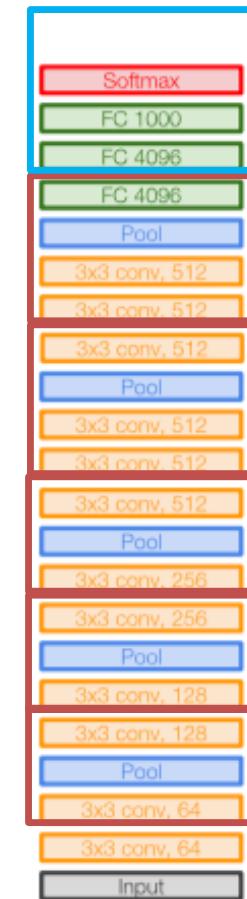
And 2x2 MAX POOL stride 2



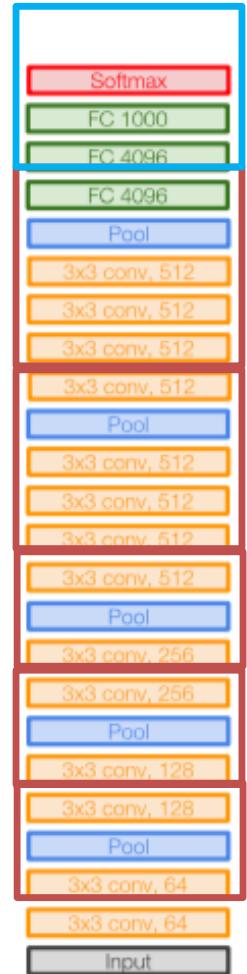
1) Feature extraction block 2) Classification block



AlexNet



VGG16



VGG19

Case Study: VGGNet (2014)

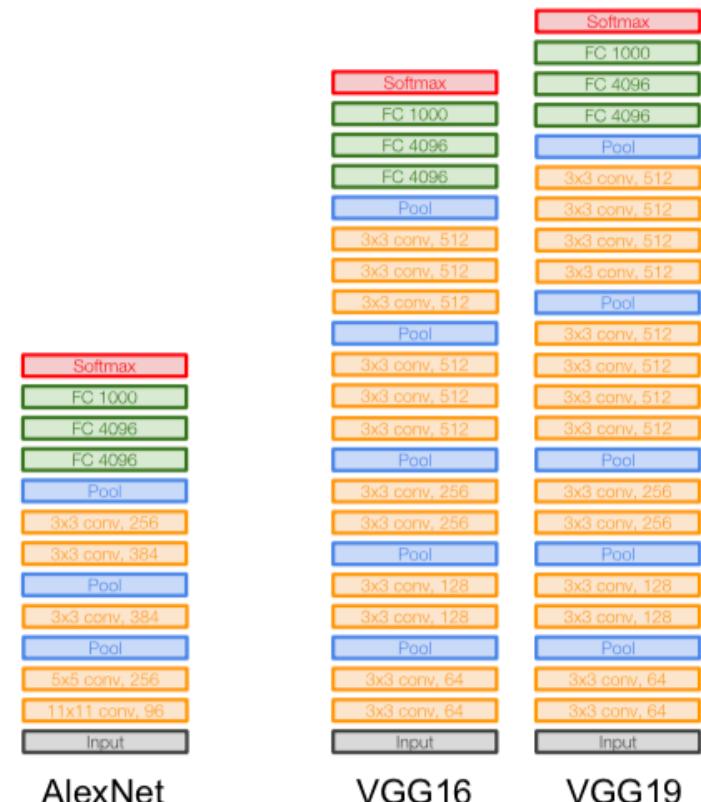
[Simonyan and Zisserman, 2014]

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					

Calculate the number of parameters on VGG Net

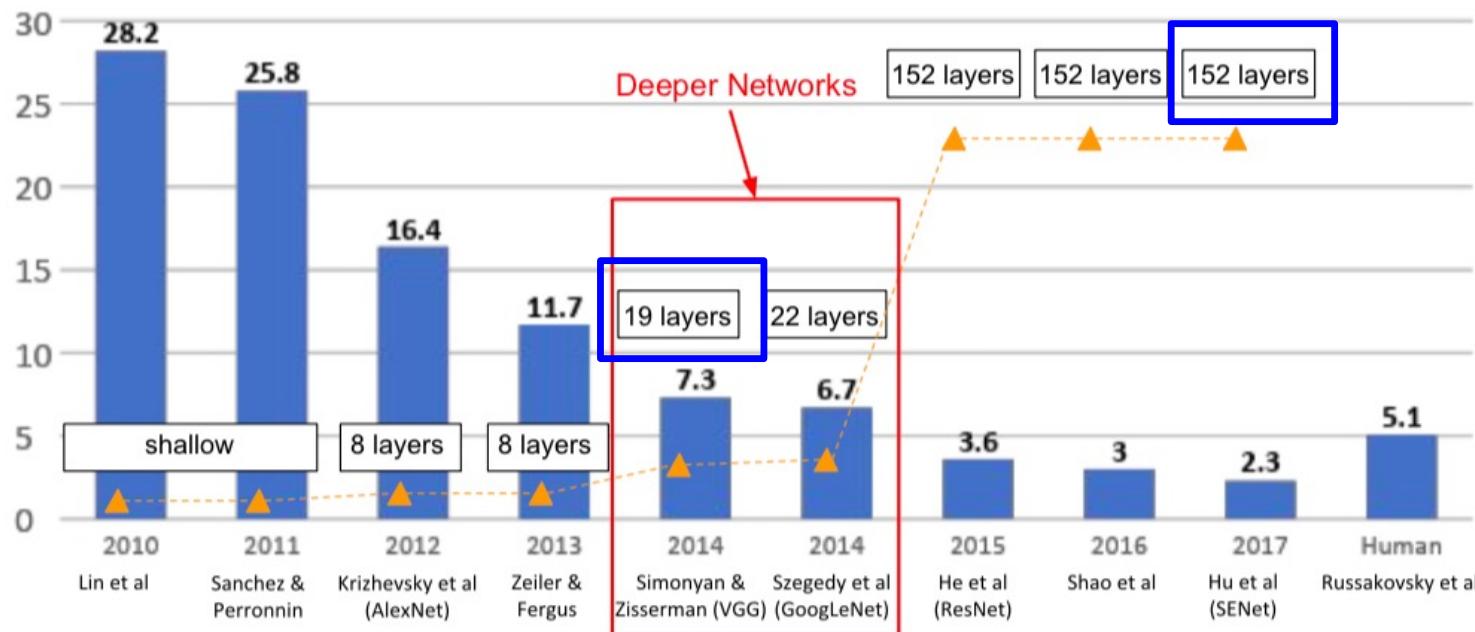
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128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

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:

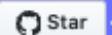
- [AlexNet](#)
- [VGG](#)
- [ResNet](#)
- [SqueezeNet](#)
- [DenseNet](#)
- [Inception v3](#)
- [GoogLeNet](#)
- [ShuffleNet v2](#)
- [MobileNetV2](#)
- [MobileNetV3](#)
- [ResNeXt](#)
- [Wide ResNet](#)
- [MNASNet](#)

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](#)

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*, Zihang 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



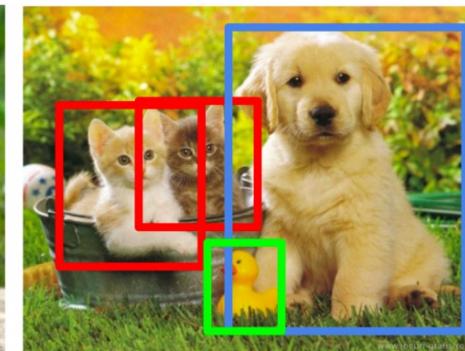
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation



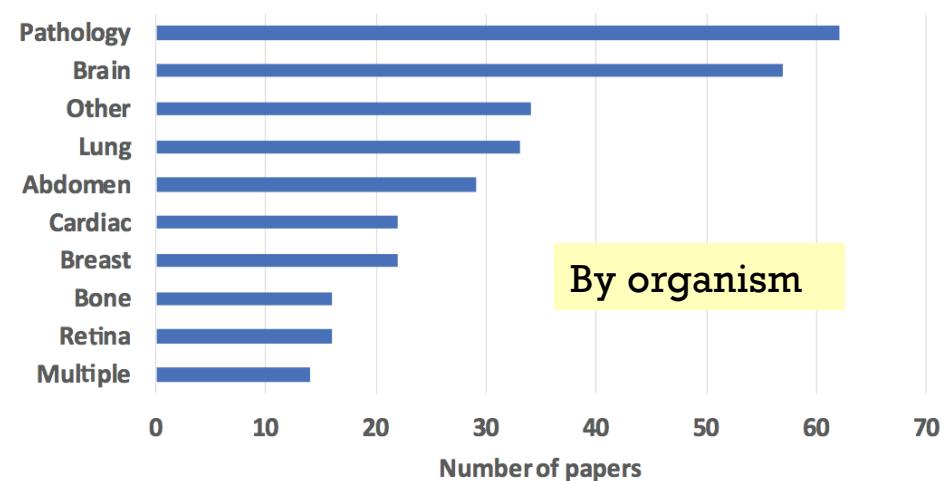
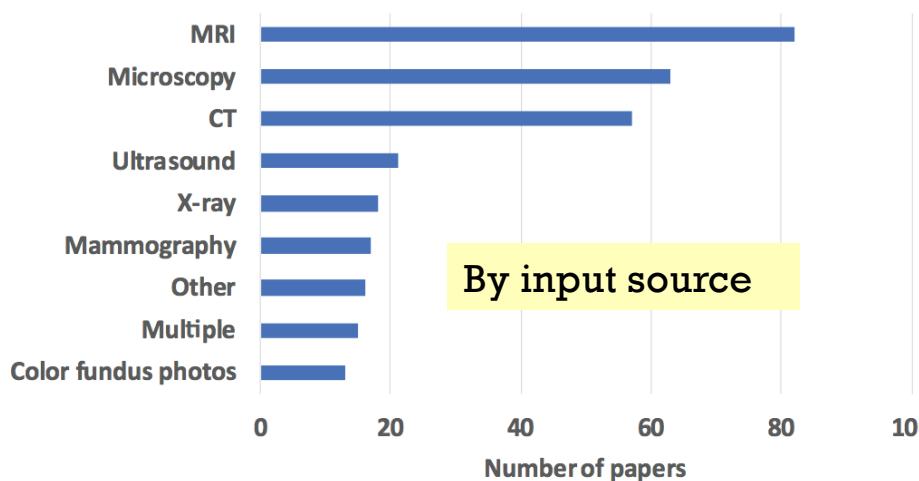
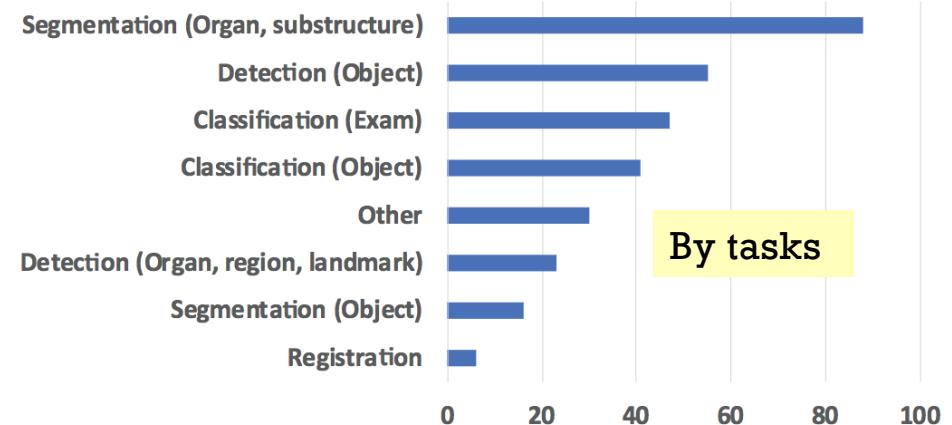
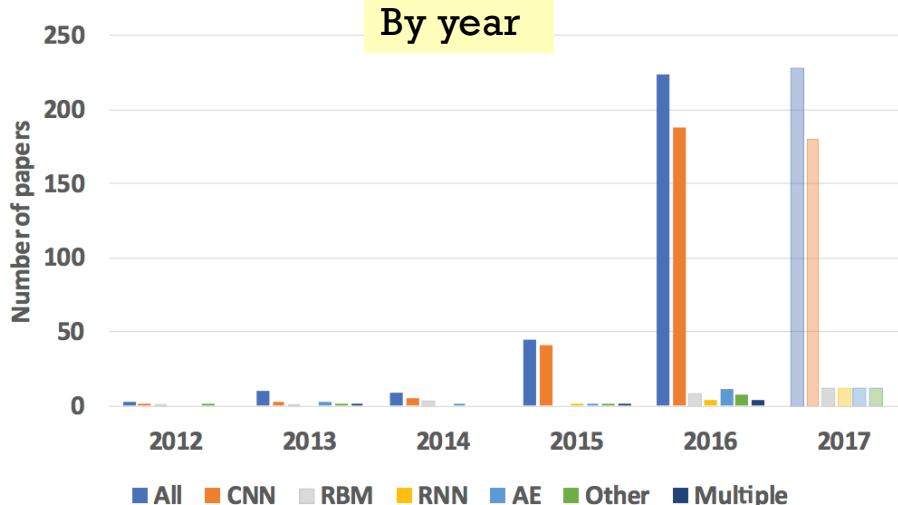
CAT, DOG, DUCK

Single object

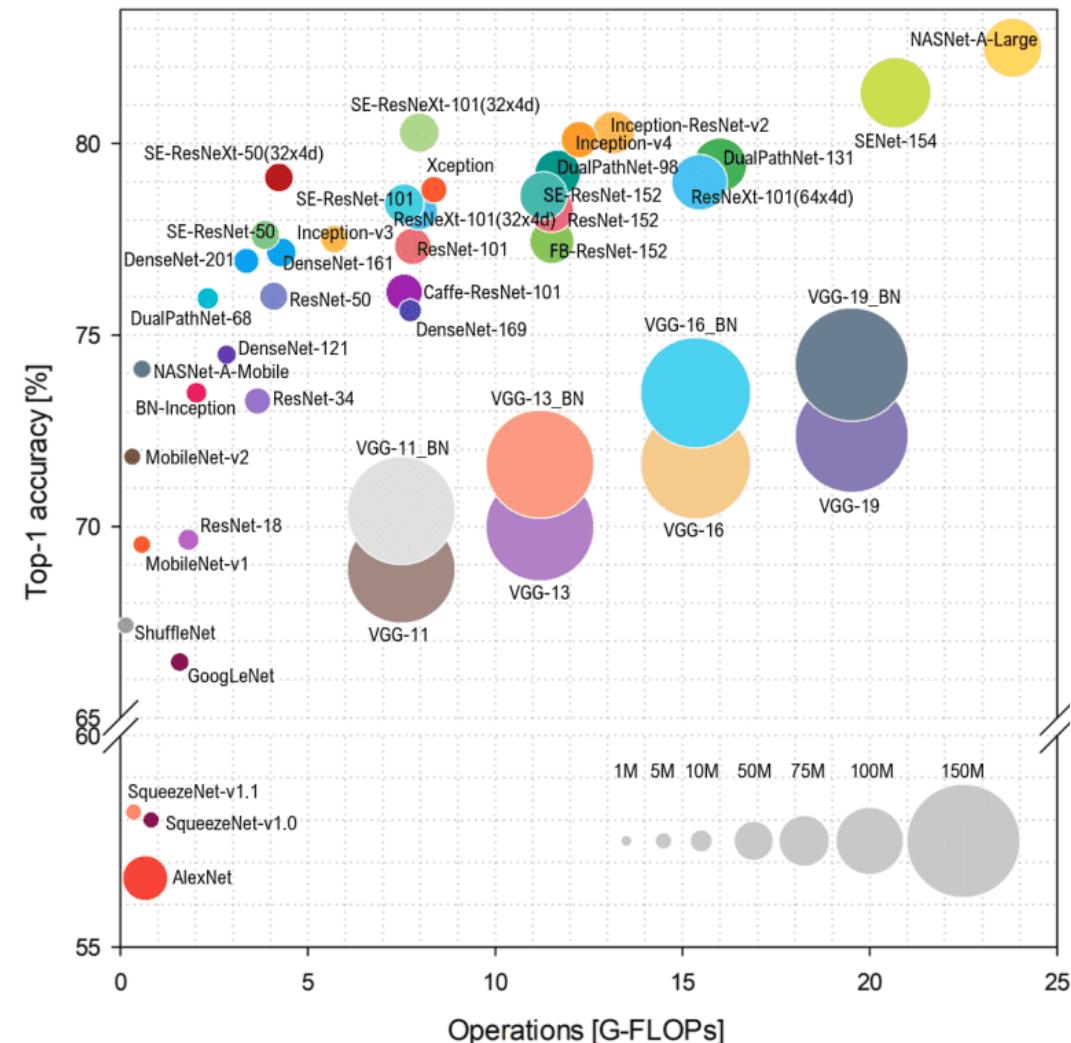
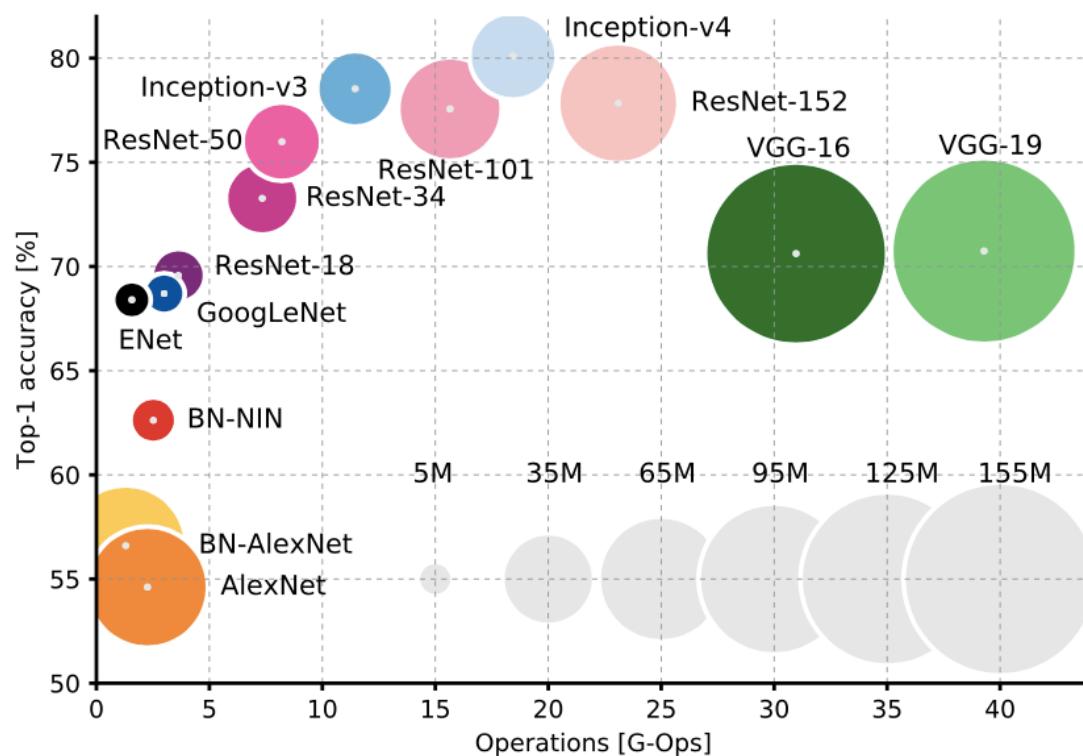
Multiple objects



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/>

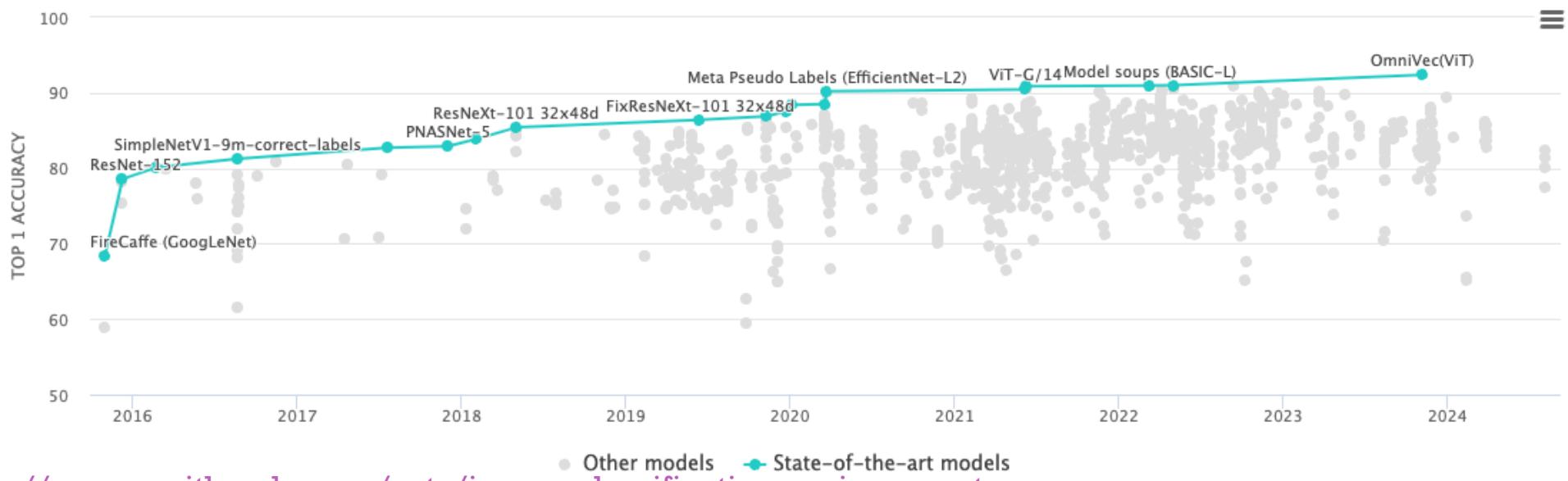


Image classification dashboard (2024)

Image Classification on ImageNet

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

View Top 1 Accuracy by Date for All models



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



EfficientNet (May, 2019) → EffNetV2 (2021)

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

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

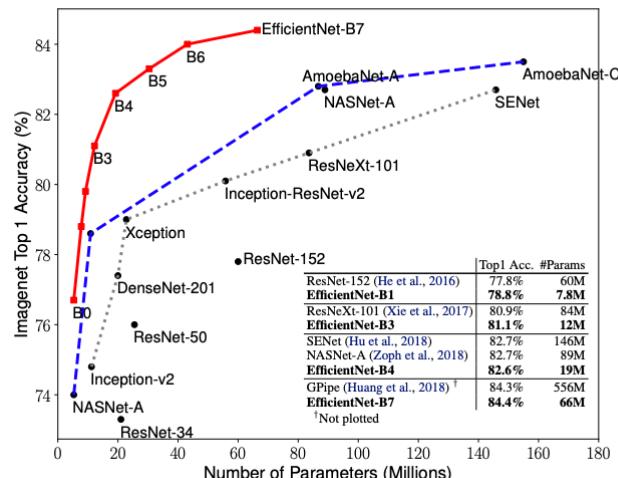
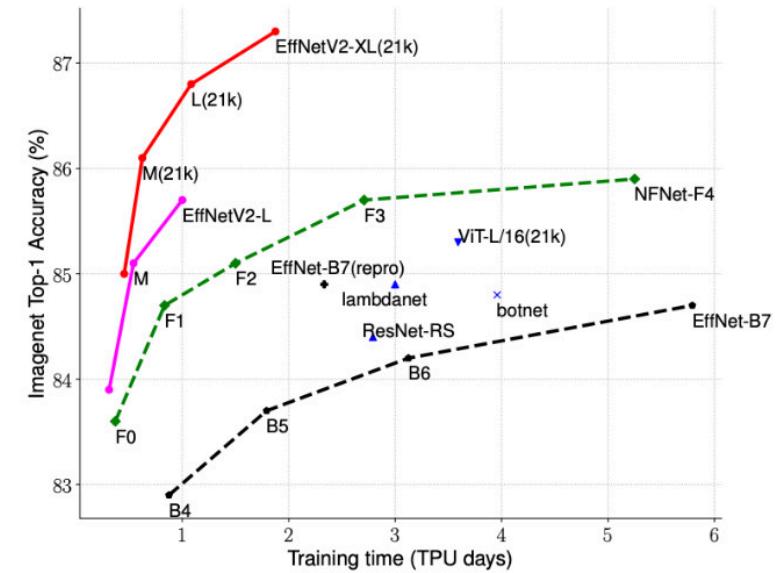


Figure 1 Model Size vs. ImageNet Accuracy. All numbers are



	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>

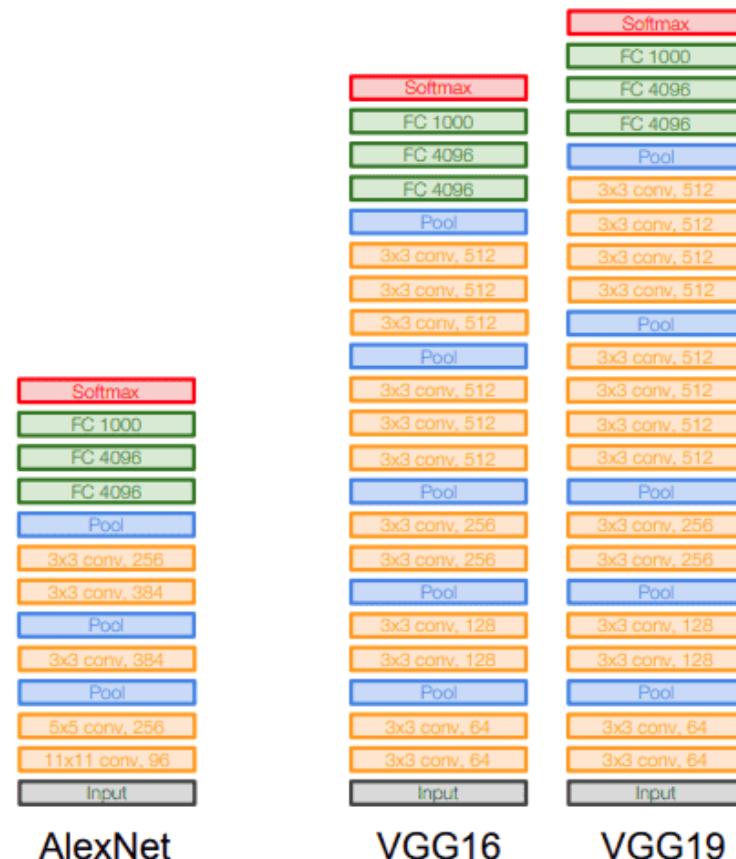
<https://twitter.com/TensorFlow/status/1423359562724175873/photo/1>



Network series

1. VGG (2014)
2. Inception/GoogleNet (2014)
3. Inception V2, V3
4. ResNet
5. DenseNet
6. Inception-V4 & Inception-ResNet
7. EfficientNet V1, V2
8. Transformer-based model (TBC)

1) VGG (2014)

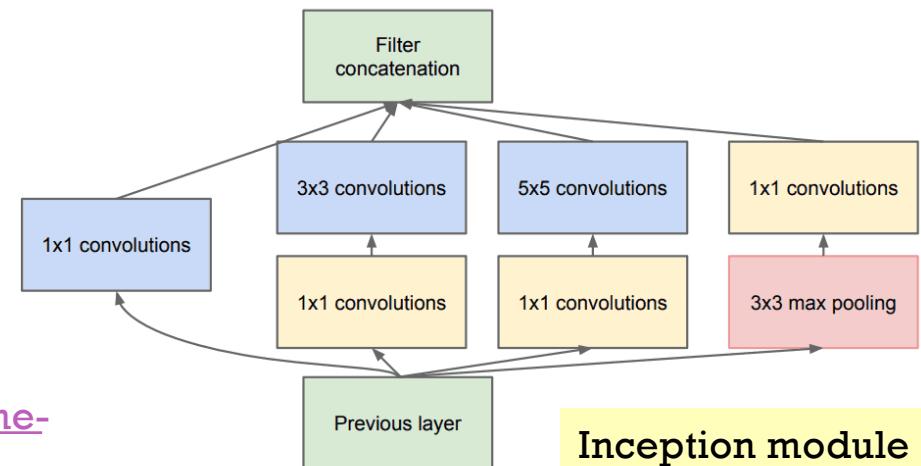


+ 2) Inception V1/GoogLeNet (2014)

- After VGG, the paper “Going Deeper with Convolutions” [3] by Christian Szegedy et al. was a huge breakthrough. What about increasing both the depth (more layers) and width (more feature maps) of the network while keeping computations to a constant level?
- The answer is with **1×1 convolutions!** The main purpose is dimension reduction, by reducing the output channels of each convolution block.
 - 1×1 convolutions are used to compute reductions before the computationally expensive convolutions (3×3 and 5×5).
- Moreover, it uses convolutions of different kernel sizes (5×5 , 3×3 , 1×1) to capture details at multiple scales.
- The InceptionNet/GoogLeNet architecture consists of **9 inception modules stacked** together, with **max-pooling layers between** (to halve the spatial dimensions). It consists of 22 layers (**27 with the pooling layers**). It uses global average pooling after the last inception module.



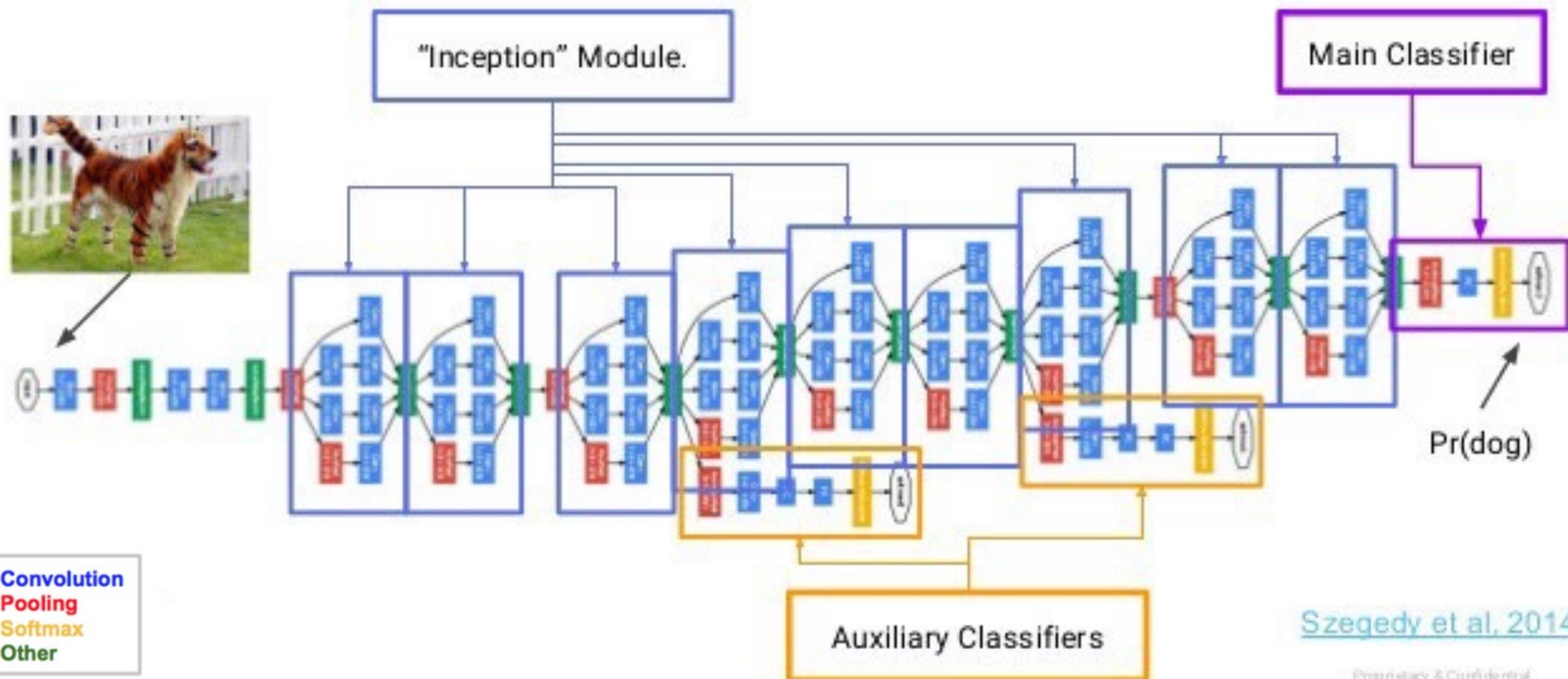
<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet (aka “Inception”) Architecture



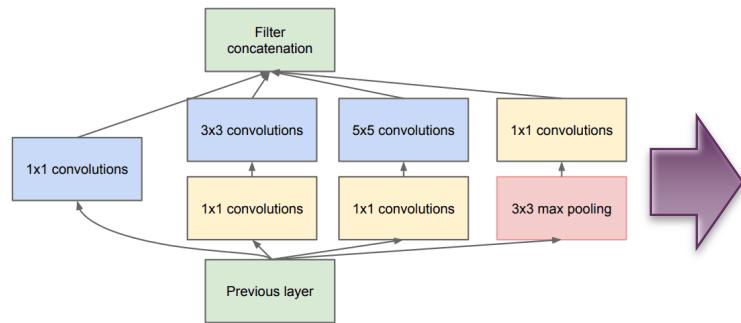
Inception is a deep network, to prevent the **middle part** of the network from “**dying out**”(vanishing gradient problem), the authors introduced **two auxiliary classifiers**. Softmax is applied in each of them, and then **Auxiliary loss** is calculated on the same labels of the output classifier.



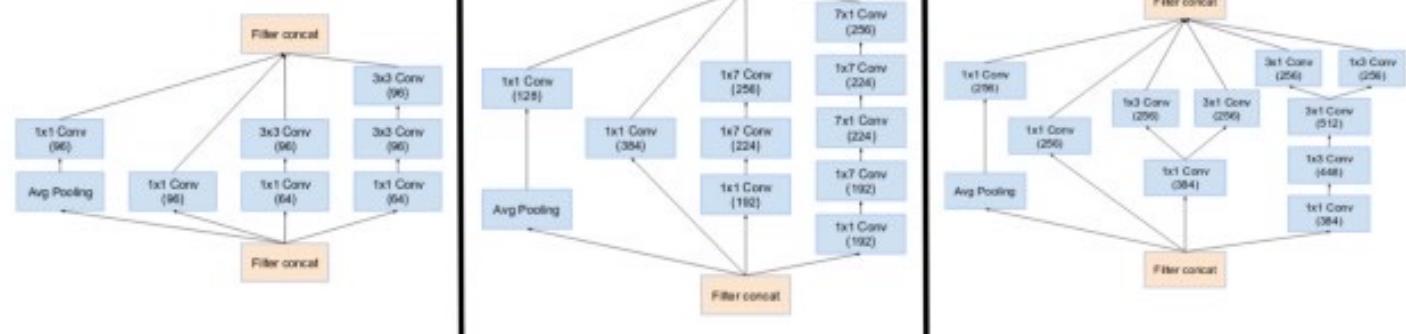


3) Inception V2, V3 (2015) – improve speed

- Factorize 5x5 and 7x7 (in InceptionV3) convolutions to two and three 3x3 sequential convolutions respectively. This improves computational speed. This is the same principle as VGG.
(analogy: $5 \times 5 = 25$ parameters vs. $2 \times (3 \times 3) = 18$ parameters)
- Simply, a 3x3 kernel is decomposed into two smaller ones: a 1x3 and a 3x1 kernel, which are applied sequentially. (analogy: $3 \times 3 = 9$ parameters vs. $(1 \times 3) + (3 \times 1) = 6$ parameters)
- The inception modules became wider (more feature maps).
- They added batch normalization.



Inception-V1 module



Inception Module (A, B, C)
in Inception V2, V3



Batch normalization

- Batch Normalization focuses on standardizing the inputs to **any particular layer** (i.e., activations from previous layers).

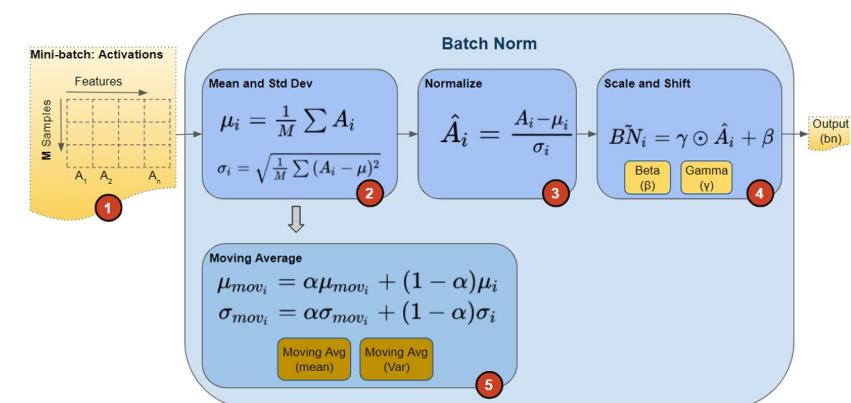
Advantages of Batch Normalization Layer

- Batch normalization improves the training time and accuracy of the neural network.
- It decreases the effect of weight initialization.
- It also adds a regularization effect on the network.
- It works better with the fully Connected Neural Network (FCN) and Convolutional Neural Network.**

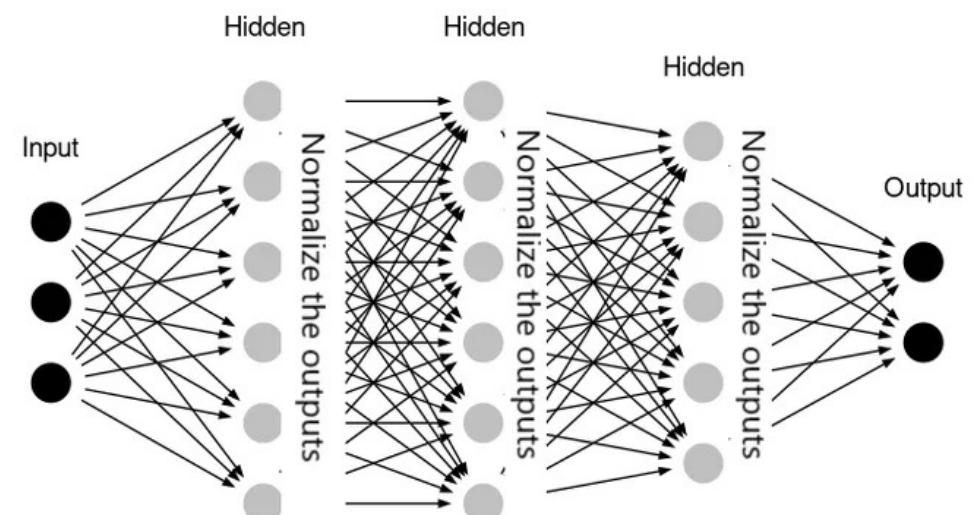
Disadvantages of Batch Normalization Layer

- Batch normalization is **dependent on mini-batch size** which means if the mini-batch size is small, it will have little to no effect
- Batch normalization does not work well with Recurrent Neural Networks (RNN)**

$$\hat{x} = \frac{x - \text{mean}(\bar{x})}{\text{std}(\bar{x})}$$



<https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739>



<https://machinelearningknowledge.ai/keras-normalization-layers-explained-for-beginners-batch-normalization-vs-layer-normalization/#Layer Normalization Layer>

Layer normalization

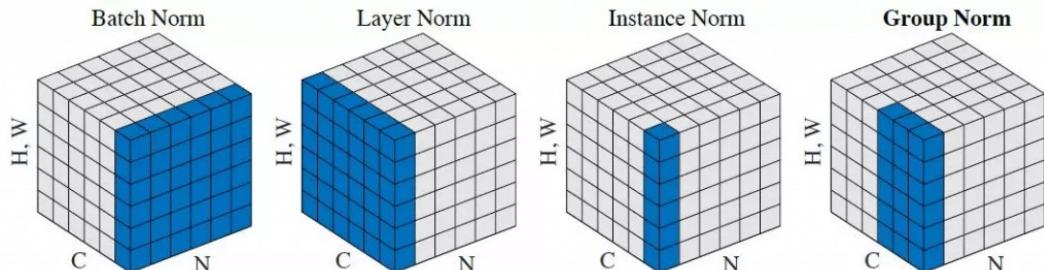
- Layer Normalization which addresses the drawbacks of batch normalization. This technique is not dependent on batches and the normalization is applied on the neuron for a single instance across all features.

■ Advantages of Layer Normalization

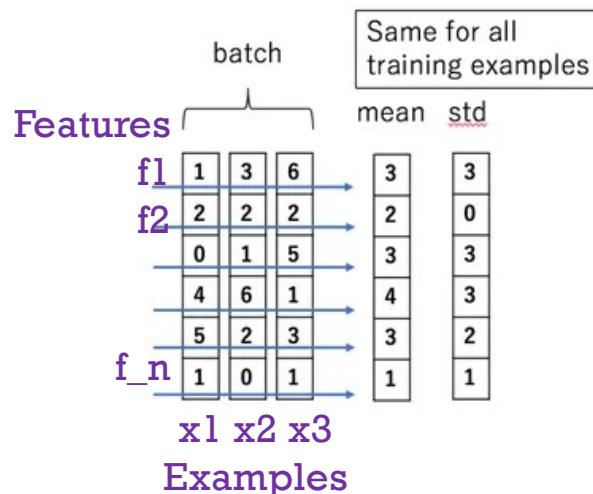
- It is not dependent on any batch sizes during training.
- It works better with Recurrent Neural Network.

■ Disadvantages of Layer Normalization

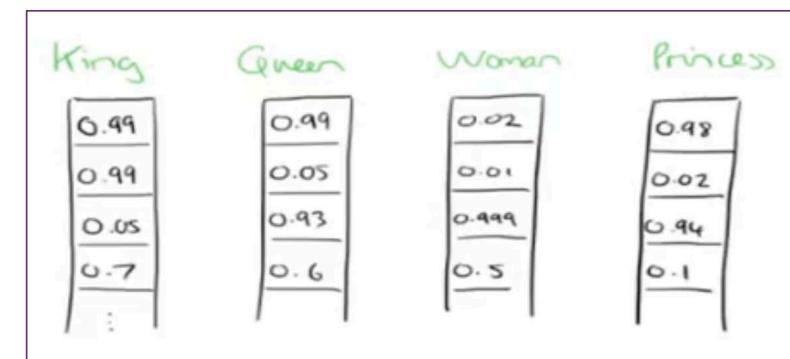
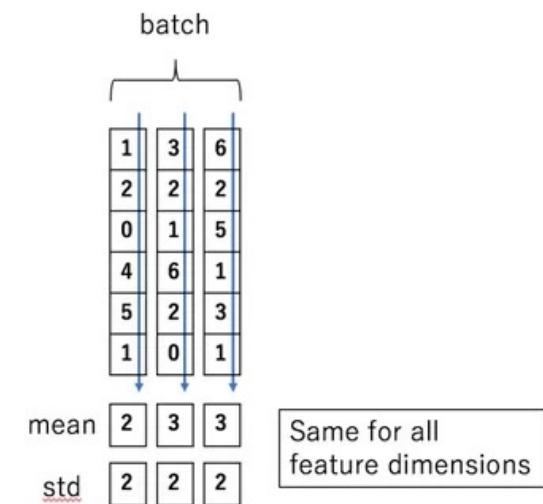
- It may not produce good results with Convolutional Neural Networks (CNN)



Batch Normalization



Layer Normalization

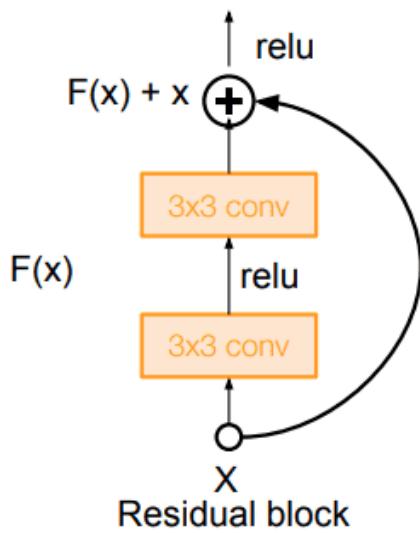


C = features (channels)

N = each mini-batch

+ 4) ResNet: Deep Residual Learning for Image Recognition (2015)

- All the predescribed issues such as vanishing gradients were addressed with two tricks:
 - **batch normalization** and
 - **short skip connections** - Instead of $H(x) = F(x)$, we ask them model to learn the difference (**residual**) $H'(x) = F(x) + x$, which means $H(x) - x = F(x)$ will be the residual part [4].
- Designed deeper architectures ranging from 18 (Resnet-18) to 150 (Resnet-150) layers.
- For the deepest models they adopted **1x1 convs**, as illustrated on the right:



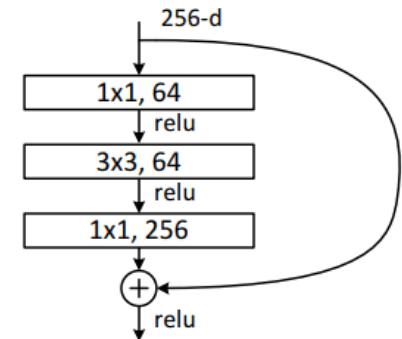
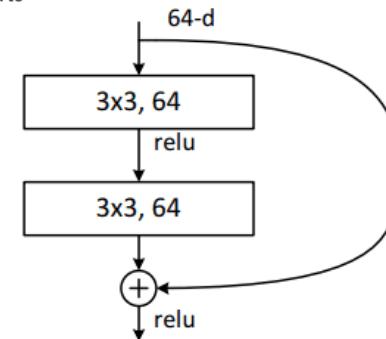
the layer followed by adding a bias term.

Then comes the activation function, $f()$ and we get the output as $H(x)$.

$H(x)=f(wx+b)$ or $H(x)=f(x)$

Now with the introduction of a new skip connection technique, the output is $H(x)$ is changed to

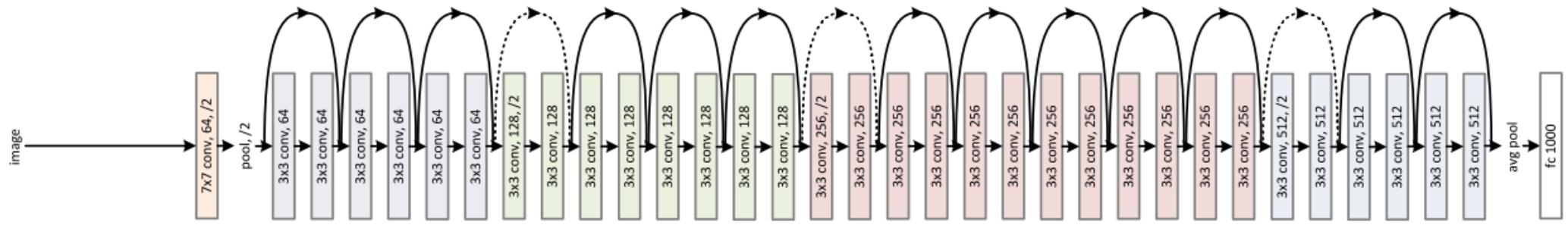
$H(x)=f(x)+x$



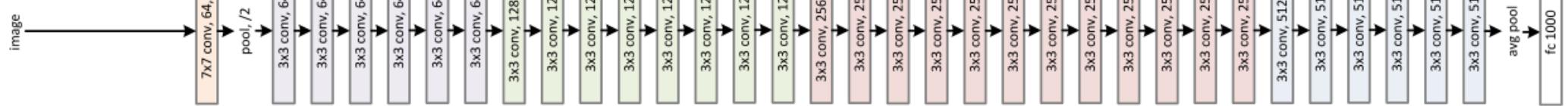


ResNet-34 vs. Plain Network (no skip connection)

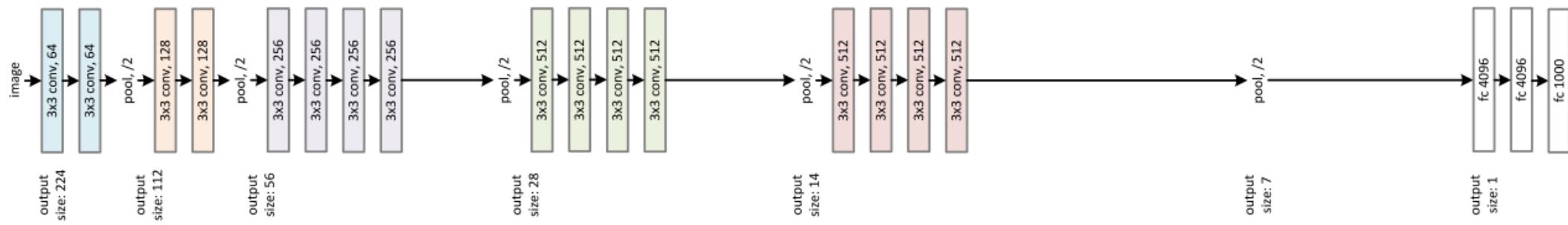
34-layer residual



34-layer plain



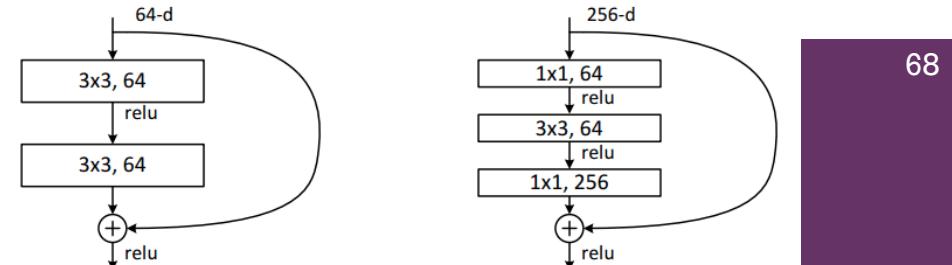
VGG-19





ResNet (cont.)

ResNet-18, 34, 50, 101, 152



68

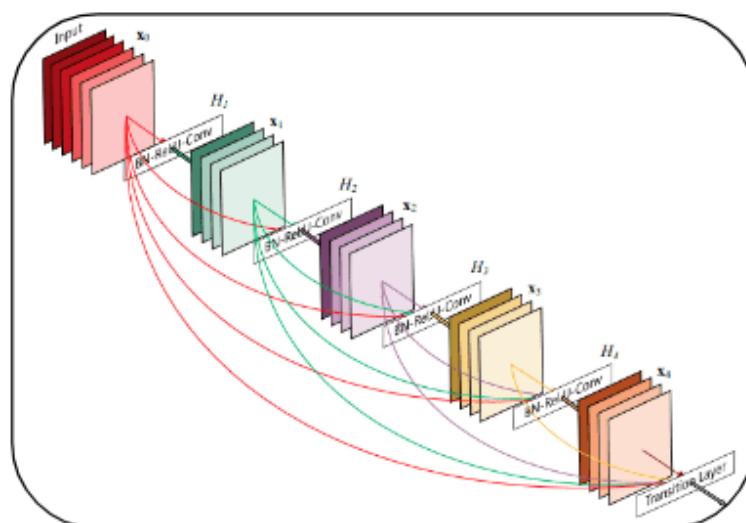
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



5) DenseNet: Densely Connected Convolutional Networks (2017)

- Skip connections are a pretty cool idea. Why don't we just skip-connect everything?

Dense Block 1

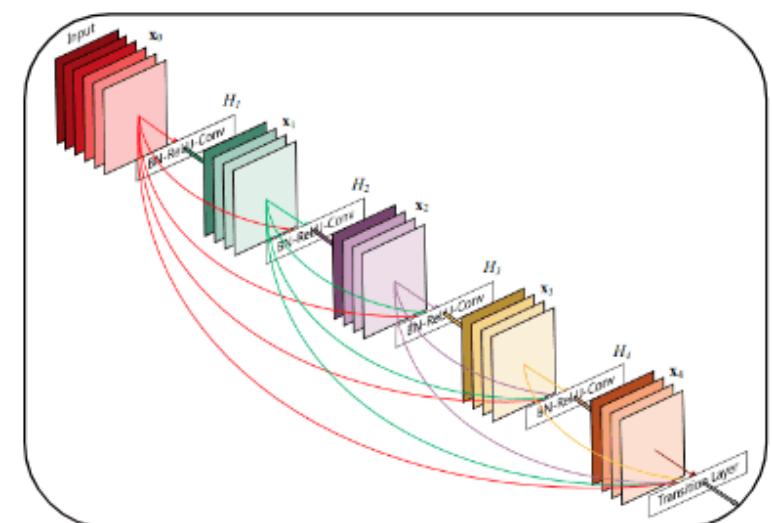


Spatial dims: 256x256
512 feature maps

Transition Layer

$1 \times 1 \times K$ convs
+
2x2 average pooling

Dense Block 2

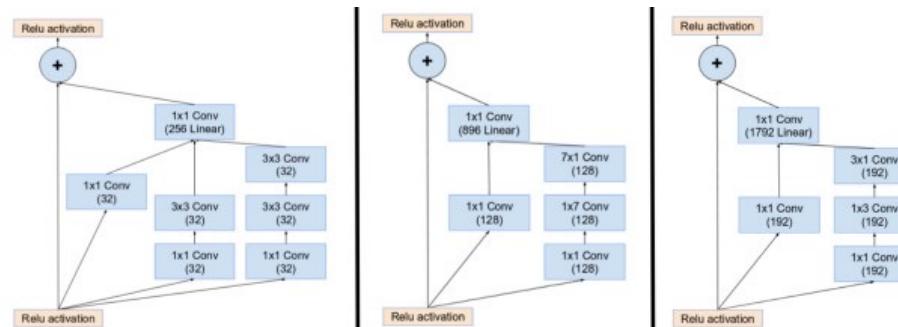


Spatial dims: 128x128
K feature maps < 512



6) Inception-V4 & Inception-ResNet

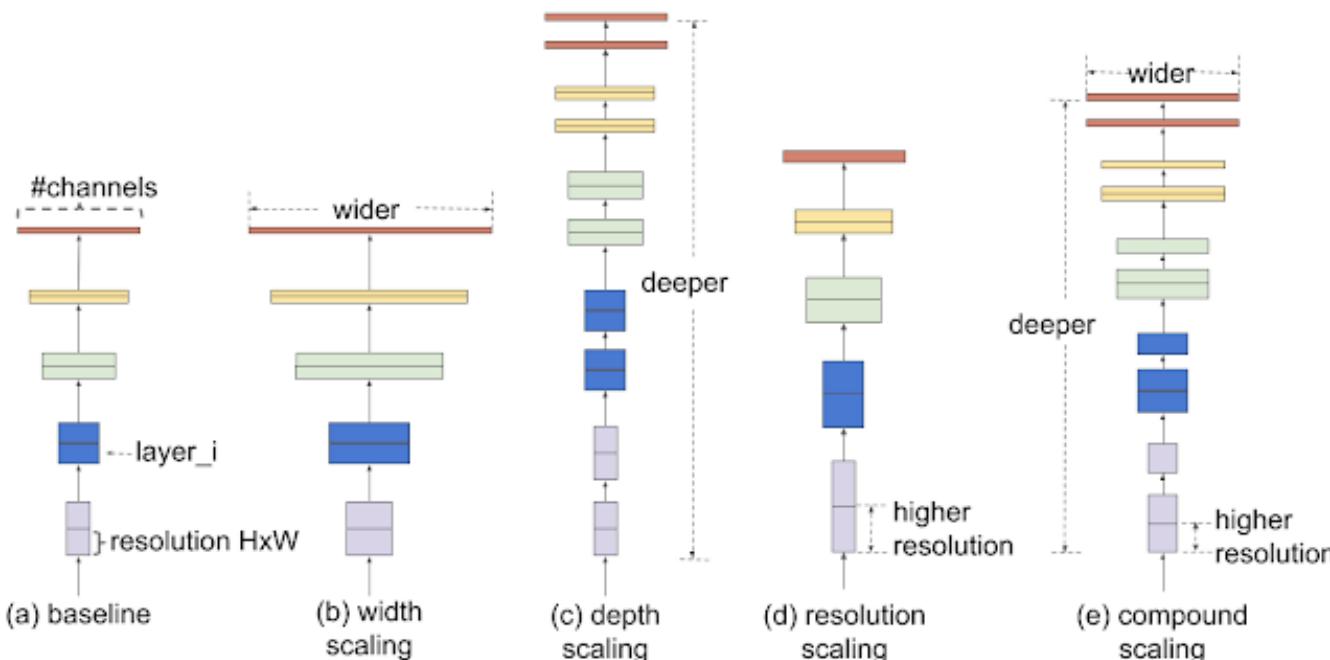
- Inception v4 and Inception-ResNet were introduced in the same paper.
- Inception-V4 (simplified previous version)
 - Make the modules more uniform. The authors also noticed that some of the modules were more complicated than necessary. **Thus; the “stem” of Inception v4 was modified (simplified).**
 - Inception v4 introduced specialized **“Reduction Blocks” (simplified)** which are used to change the width and height of the grid.
- Inception-ResNet V1, V2
 - For **residual addition** to work, the input and output after convolution must have the same dimensions. Hence, we use 1x1 convolutions after the original convolutions, to match the depth sizes (Depth is increased after convolution).
 - Inception-ResNet v1 has a computational cost that is similar to that of Inception v3.
 - Inception-ResNet v2 has a computational cost that is similar to that of Inception v4.





7) EfficientNet (May, 2019)

- So, let's instead scale up network depth (more layers), width (more feature maps), resolution (various sizes of filters) simultaneously. This is known as compound scaling.



For B0 to B7 base models, the input shapes are different. Here is a list of input shape expected for each model:

Base model	resolution
EfficientNetB0	224
EfficientNetB1	240
EfficientNetB2	260
EfficientNetB3	300
EfficientNetB4	380
EfficientNetB5	456
EfficientNetB6	528
EfficientNetB7	600

<https://blog.roboflow.com/how-to-train-efficientnet/>



7) EfficientNetV2 (2021)

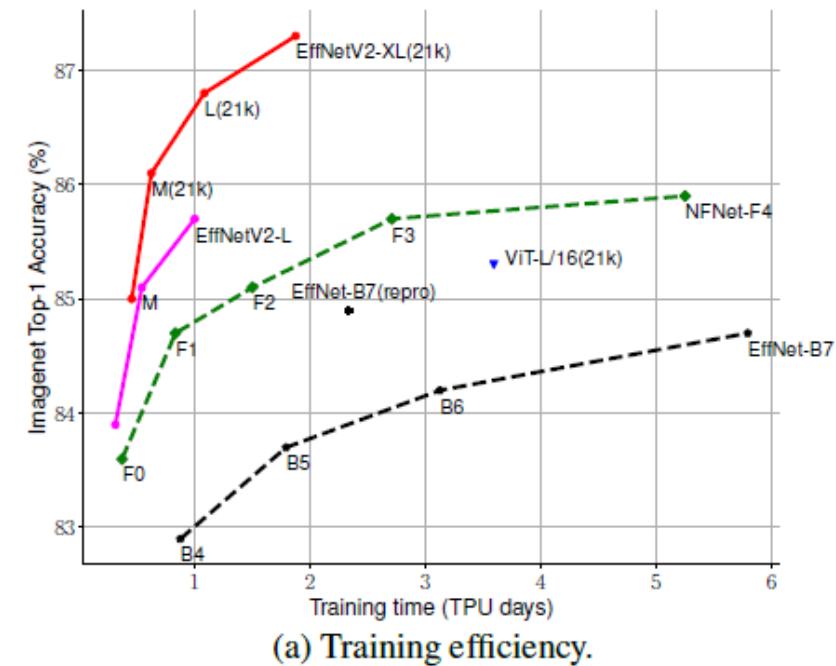
Smaller models & faster training

- Our experiments show that EfficientNetV2 models train much faster than state-of-the-art models while being up to **6.8x smaller**.
- There are many versions: small, medium, large, and extra-large (XL).

Some changes proposed in the latest architecture and methodology:

- Combination of Fused-MBConv and MB Conv instead of only MbConv
- Training-aware NAS - jointly optimizes for accuracy, parameter efficiency and training speed this time.
- Progressive Learning - Low regularization + small images initially during training, followed by high regularization
- large images later
- **Small architecture changes.**

<https://clearsignal.xyz/efficientnet-v1-v2-a-smart-heuristic/>



	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

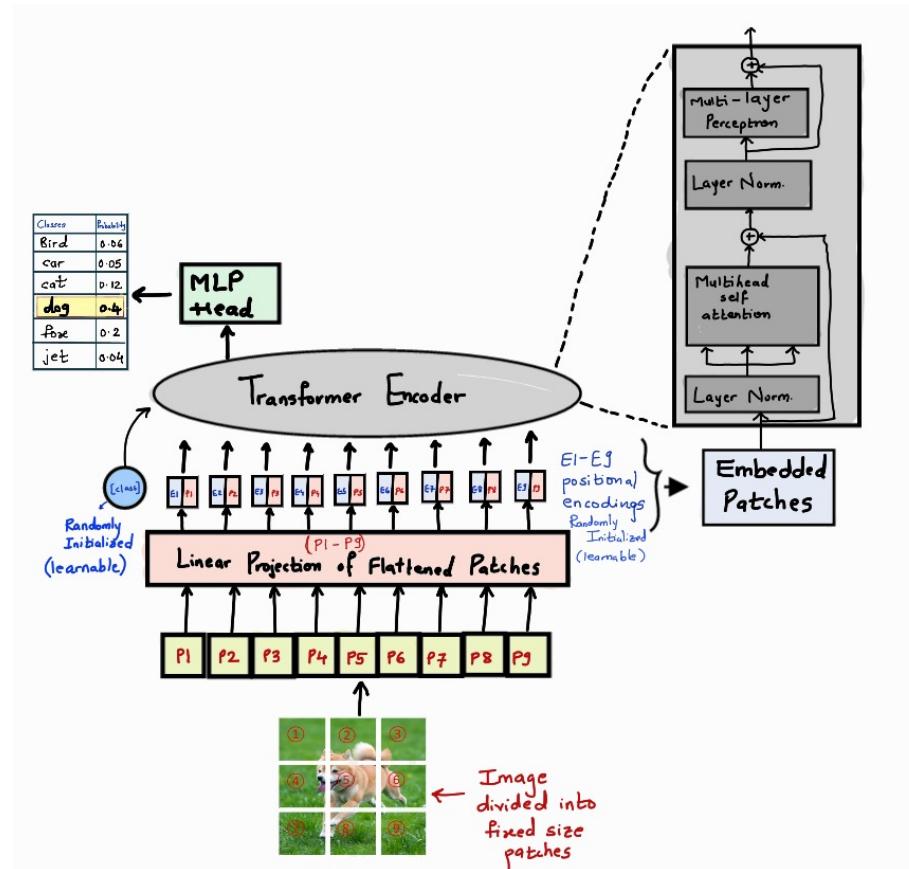
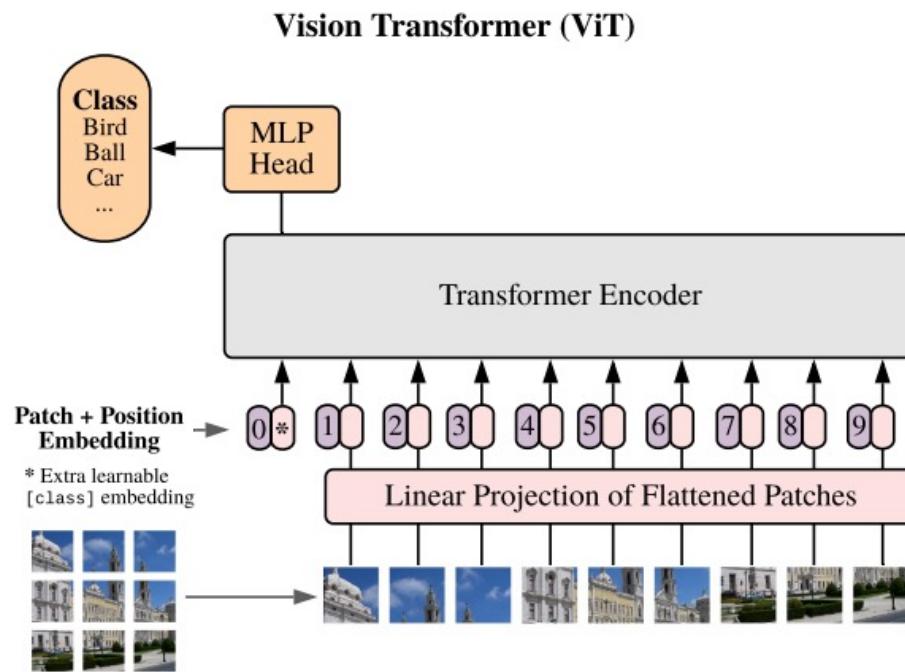
(b) Parameter efficiency.



8) Vision Transformer (ViT) (ICLR2021)

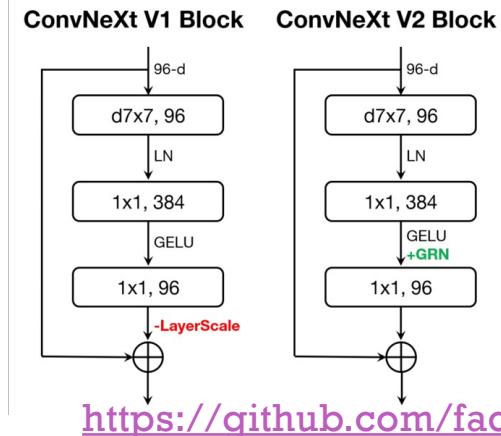
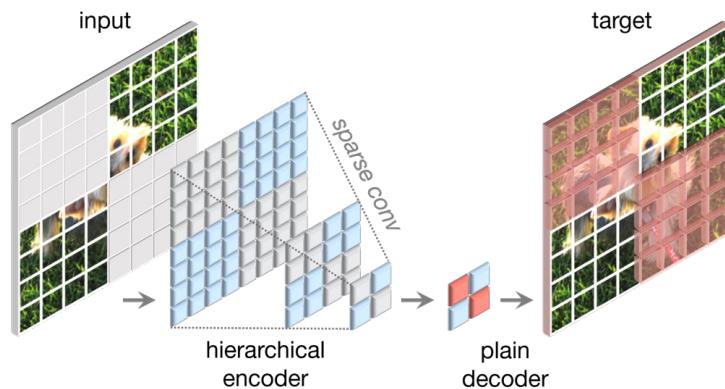
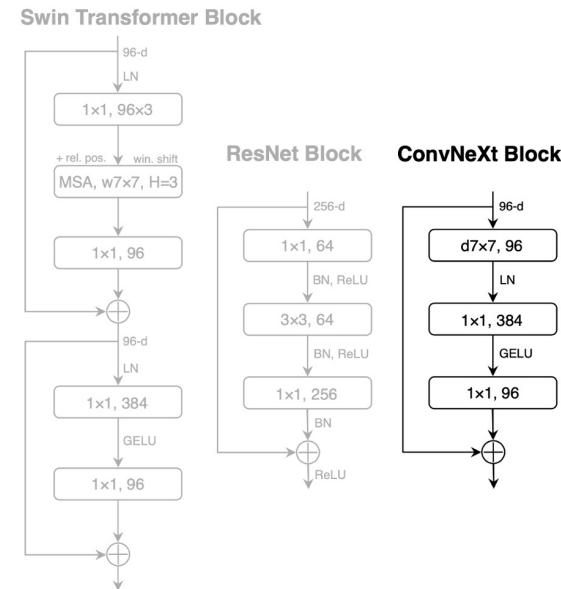
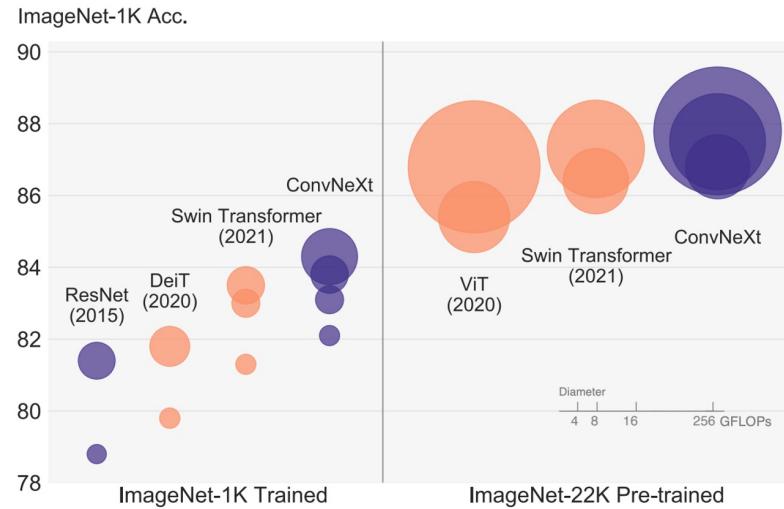


Hugging Face



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

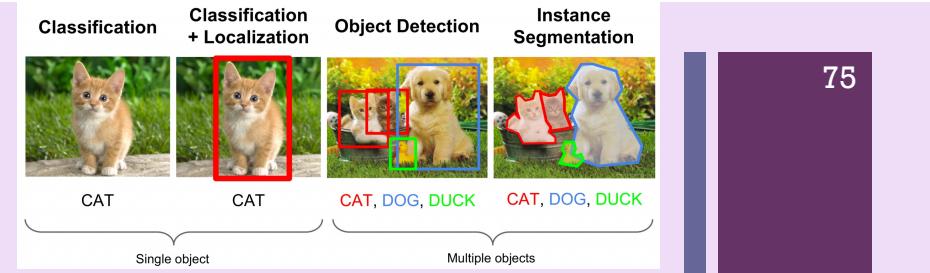
+ 9) ConvNeXt (CVPR2022) → ConvNeXt V2 (2023)



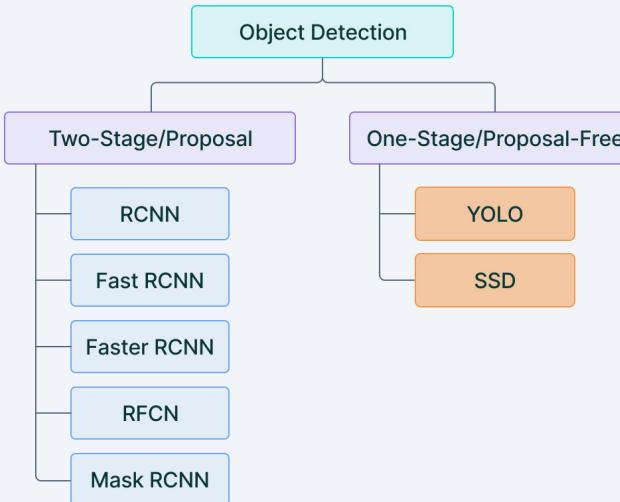
<https://github.com/facebookresearch/ConvNeXt-V2>

<https://github.com/facebookresearch/ConvNeXt>

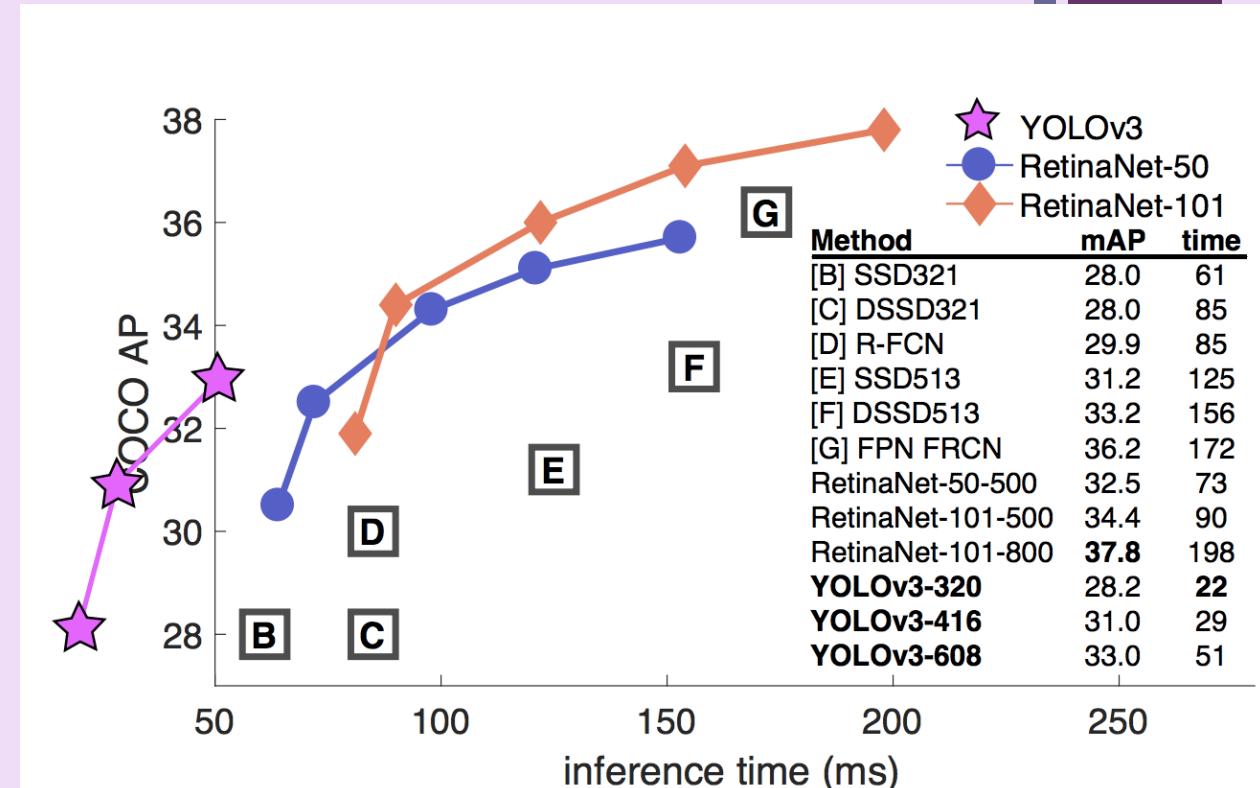
SOTA of Object Detection



One and two stage detectors



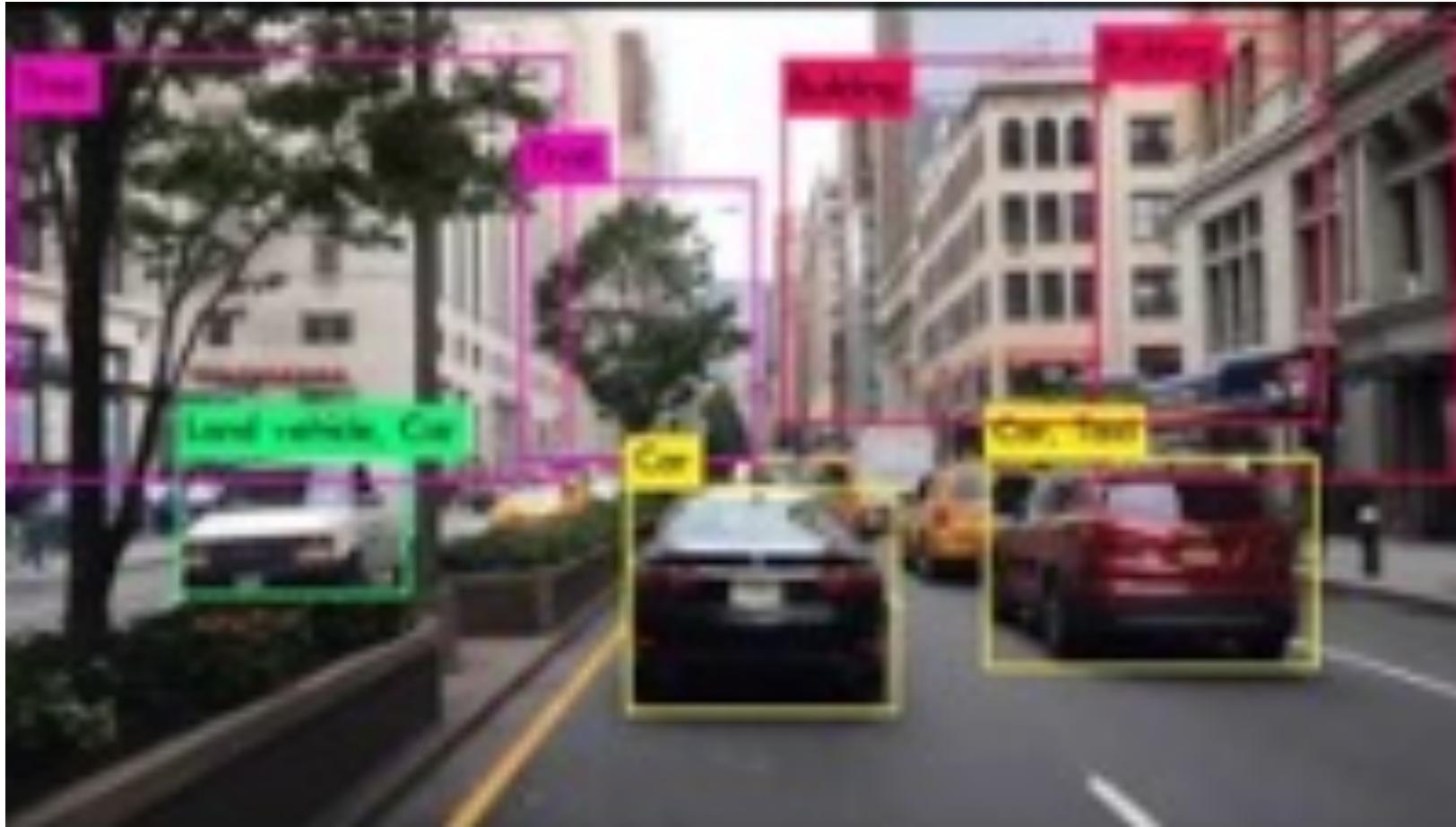
V7 Labs



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



YOLO Open Images in New York





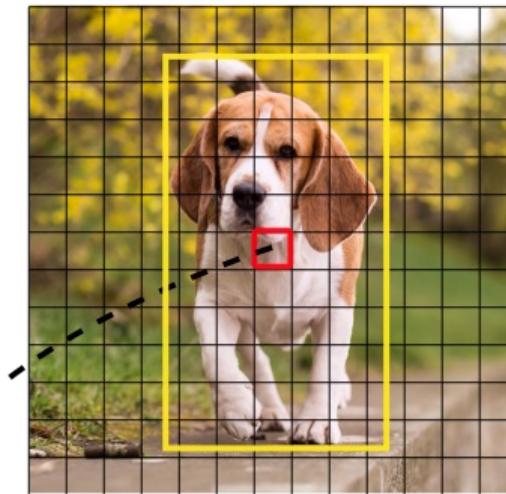
YOLO's output



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



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



77

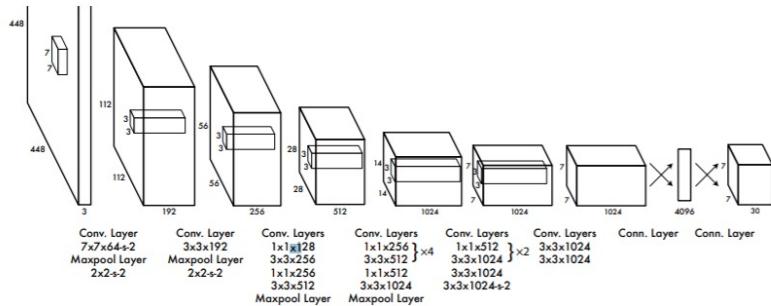
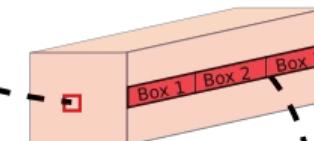


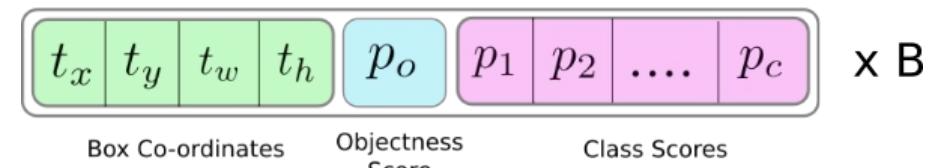
Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224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

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

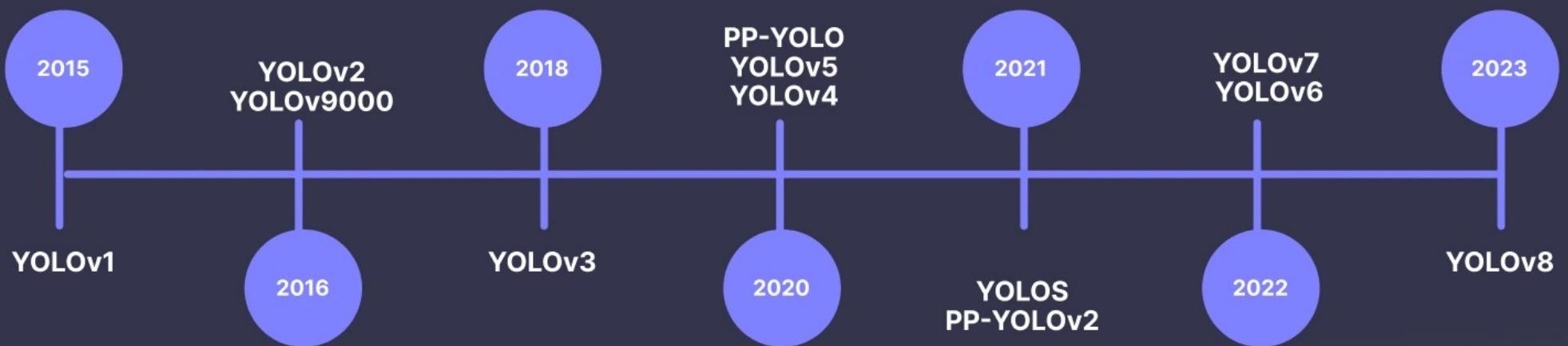
Prediction Feature Map



Attributes of a bounding box



YOLO: Release Dates Timeline



<https://encord.com/blog/yolo-object-detection-guide/>

ENCORD



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

ultralytics
YOLOv8

DOWNLOAD THE APP

GET ON Google Play | DOWNLOAD on the App Store

English | 简体中文

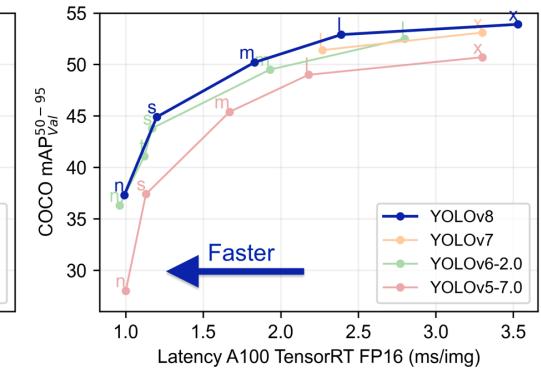
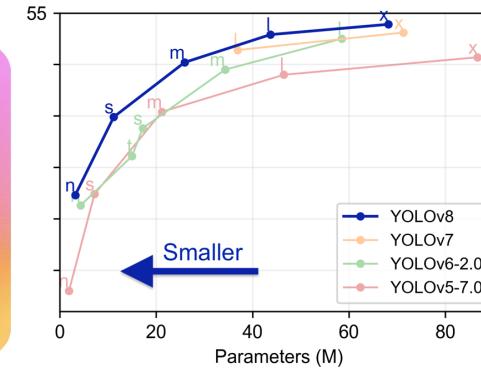
Ultralytics CI passing | codecov 87% | DOI 10.5281/zenodo.7347926 | docker pulls 22k

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.

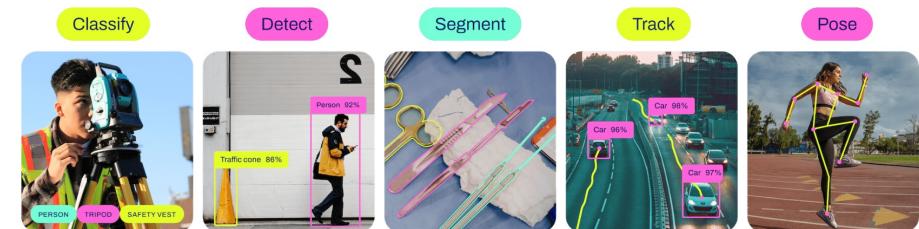
We hope that the resources here will help you get the most out of YOLOv8. Please browse the YOLOv8 [Docs](#) for details, raise an issue on [GitHub](#) for support, and join our [Discord](#) community for questions and discussions!

To request an Enterprise License please complete the form at [Ultralytics Licensing](#).



Models

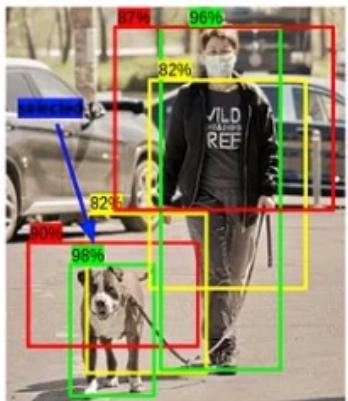
YOLOv8 Detect, Segment and Pose models pretrained on the COCO dataset are available here, as well as YOLOv8 Classify models pretrained on the ImageNet dataset. Track mode is available for all Detect, Segment and Pose models.



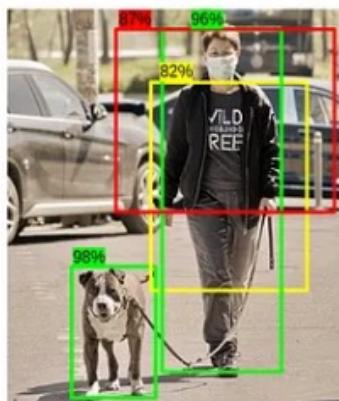
All [Models](#) download automatically from the latest Ultralytics [release](#) on first use.



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

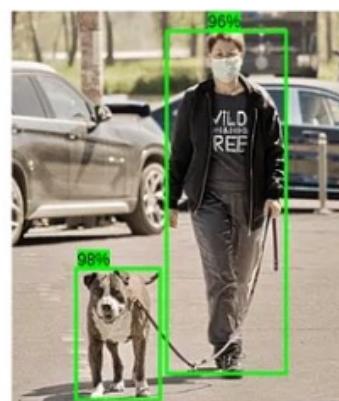


Step 1: Selecting Bounding box with highest score



Step 3: Delete Bounding box with high overlap

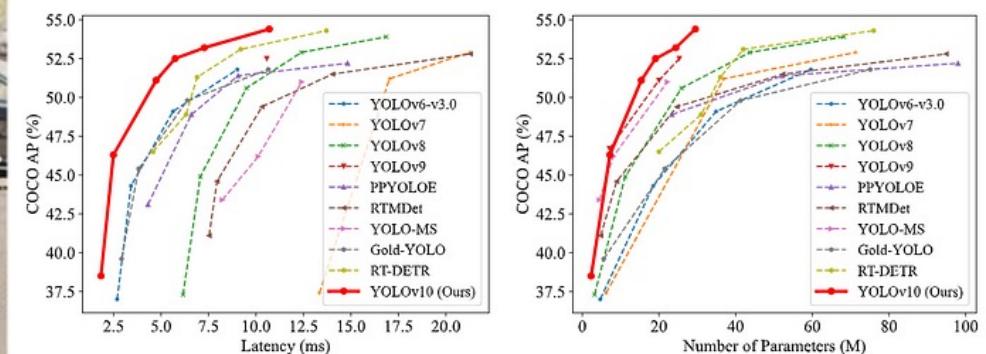
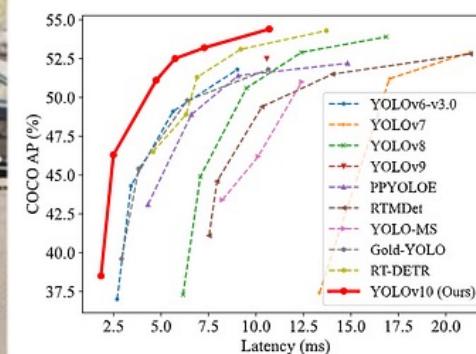
Overlapping Bounding Boxes



Step 5: Final Output

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).

$$\text{IoU} = \text{Intersect Area} / \text{Union Area}$$



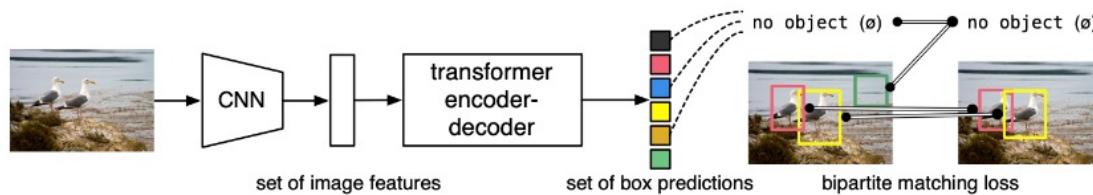
- However, YOLO v10 eliminates the need for NMS by using consistent dual assignments during training. This innovation reduces computational overhead and latency while maintaining high accuracy, making YOLO v10 faster and more efficient for real-time applications
- 6 sizes: N, S, M, B, L, X

+ DETR (Detection Transformer) [Meta, 2020]

DETR: End-to-End Object Detection with Transformers

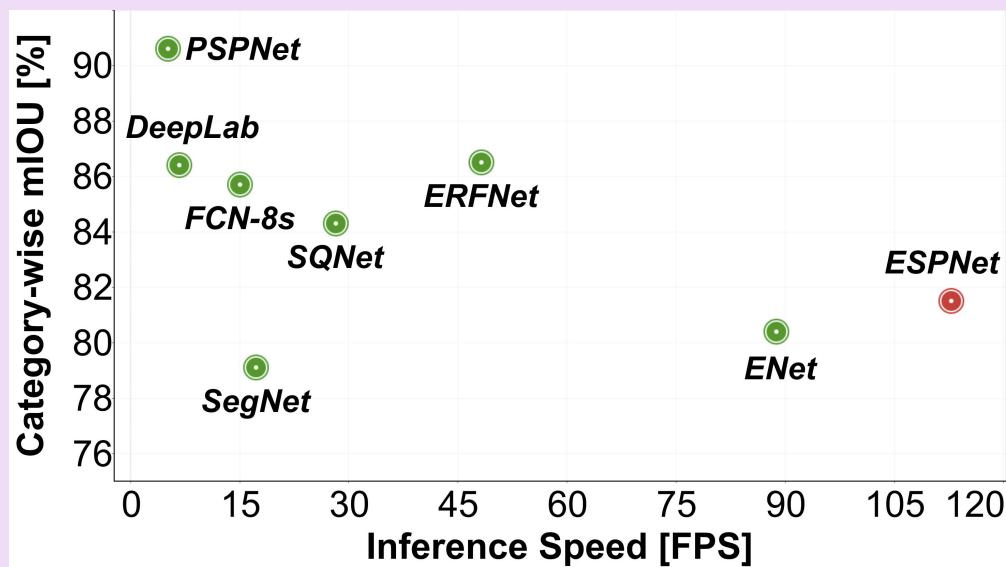
Support Ukraine

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.



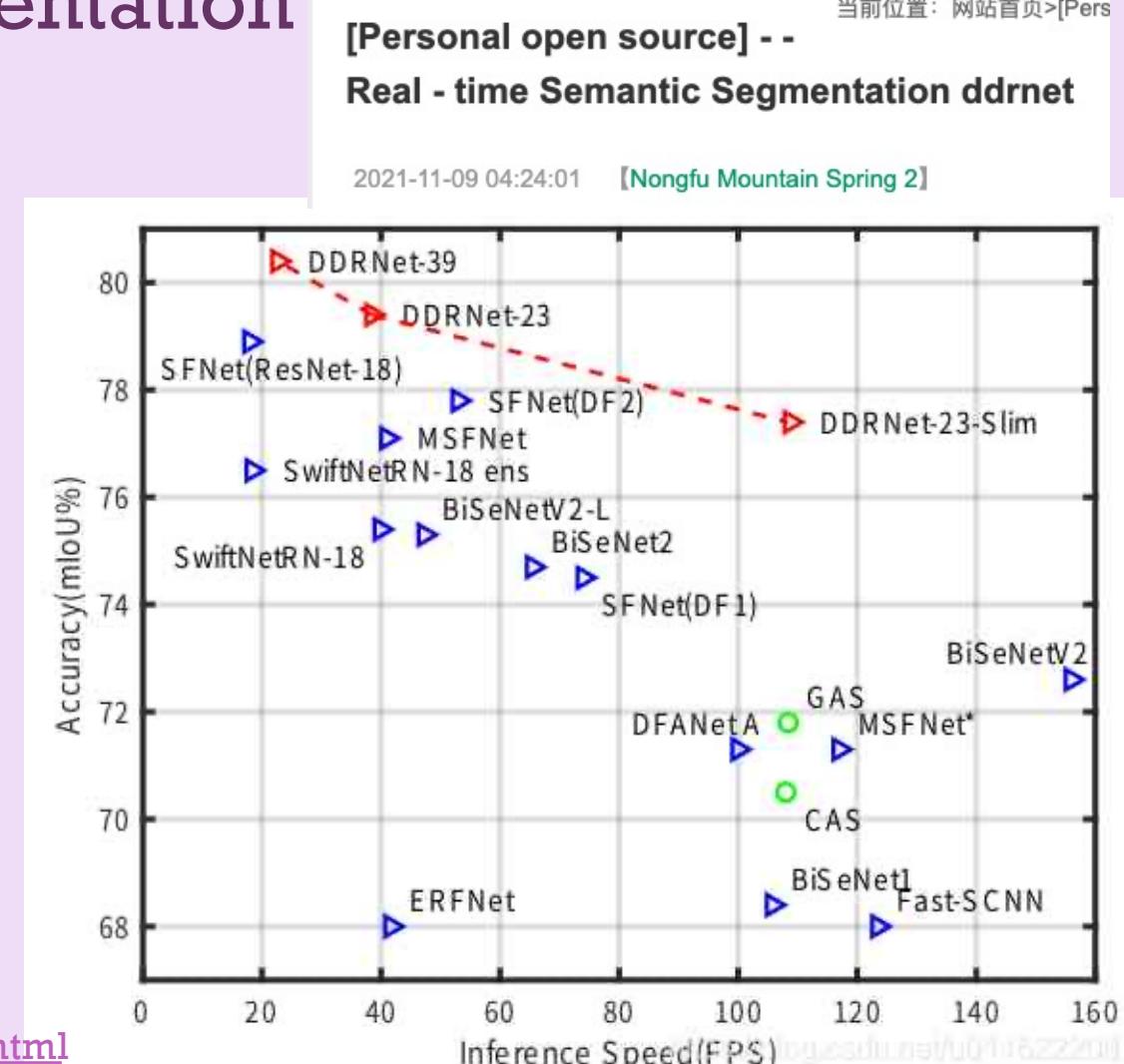
- 1. Deformable DETR (2020)
■ Team: Microsoft Research Asia
- 2. Conditional DETR (2021)
■ Team: Chinese Academy of Sciences
- 3. Anchor DETR (2021)
■ Team: SenseTime and Shanghai Jiao Tong University
- 4. Efficient DETR (2021)
■ Team: UC Berkeley and Google Research
- 5. DINO (DETR with Improved Non-Autoregressive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

SOTA of Semantic Segmentation



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

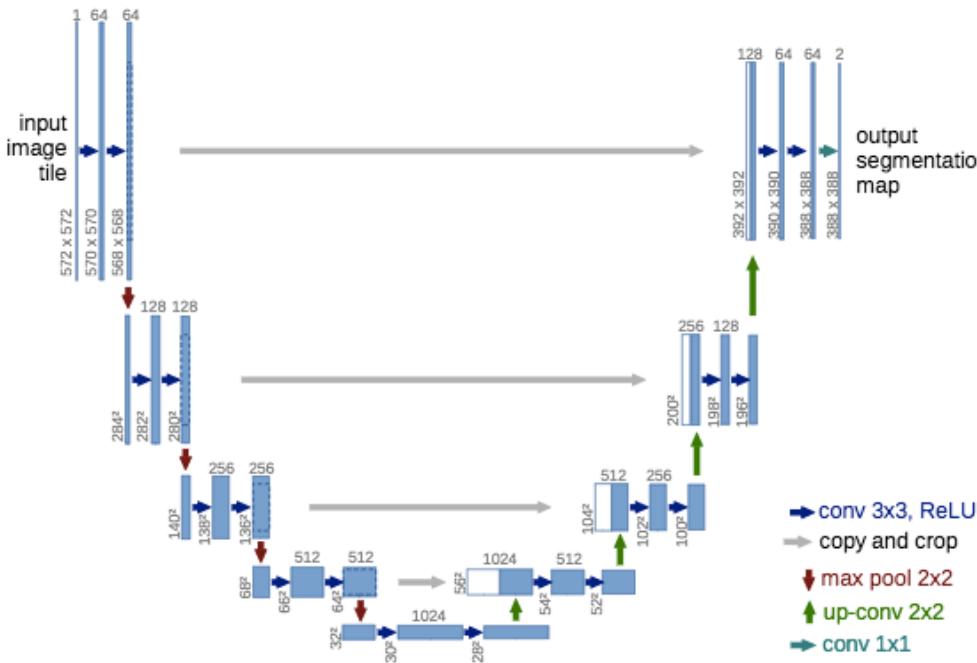
<https://chowdera.com/2021/11/20211109042359120o.html>



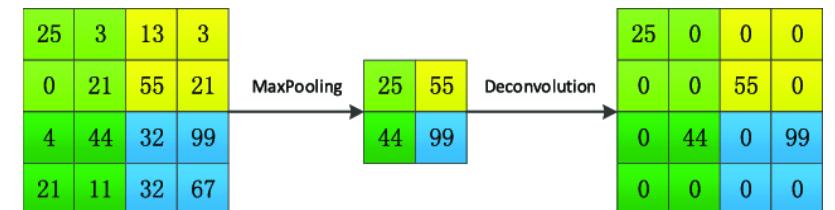


UNet: Encoder-Decoder Network

■ 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. \(2015\)](#)



Semantic Segmentation

84

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>



Recent interesting segmentation networks

- 1. 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.

- 2. 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.

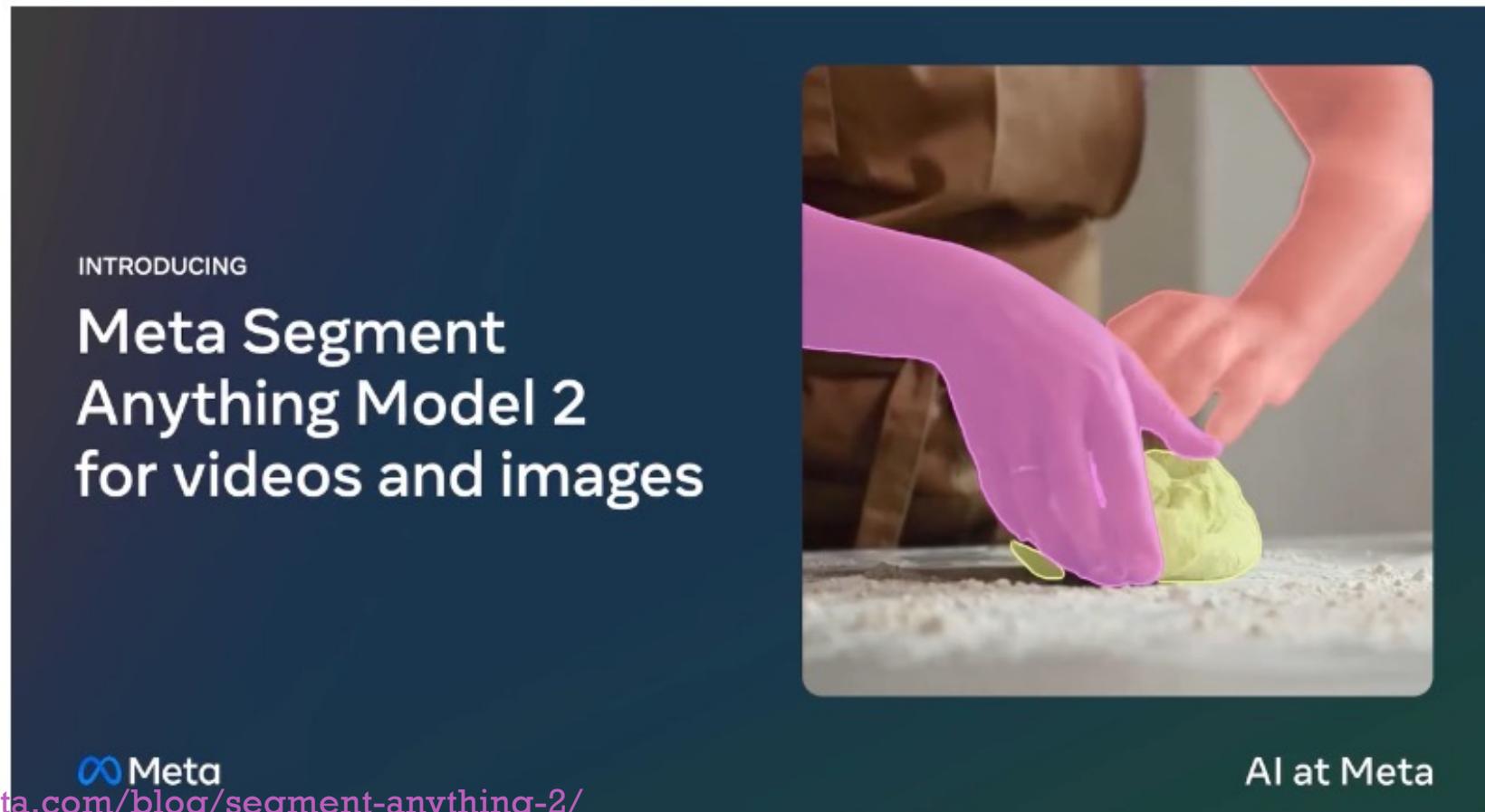
- 3. Swin Transformer V2 [Microsoft, 2022]
- Explanation: Swin Transformer V2 builds on the hierarchical vision transformer design introduced in the original Swin Transformer. It enhances performance in both segmentation and classification tasks and has achieved SOTA results on benchmarks such as COCO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

- 4. 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.

- 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.

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

July 29, 2024 • 15 minute read



INTRODUCING

Meta Segment Anything Model 2 for videos and images

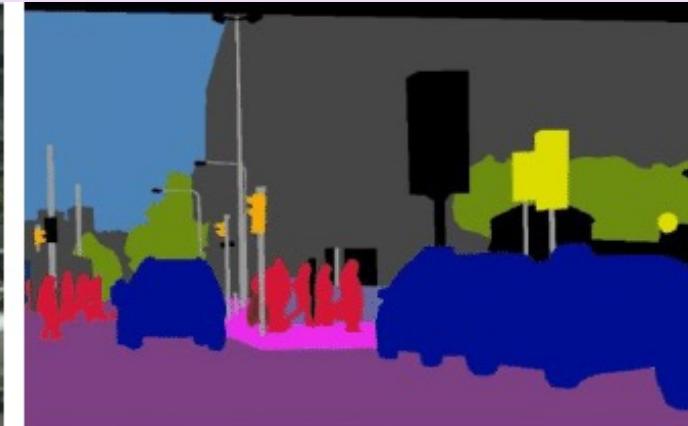
 Meta

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

AI at Meta



Panoptic Segmentation





OneFormer: One Transformer to Rule Universal Image Segmentation (CVPR2023)

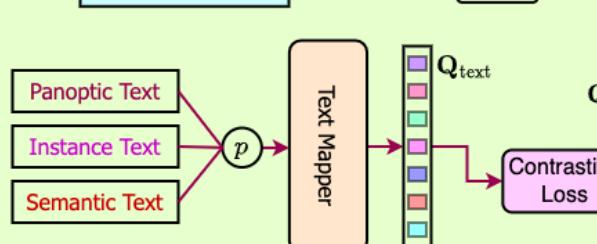
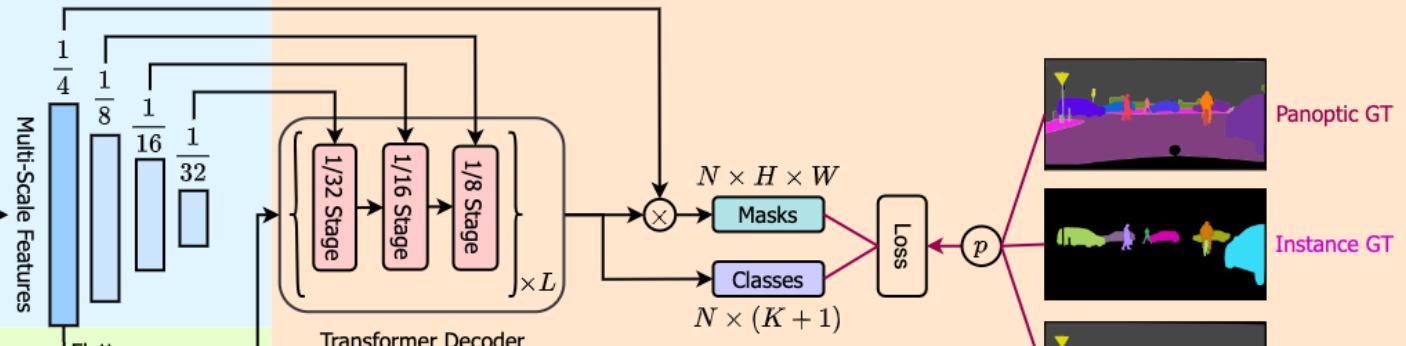
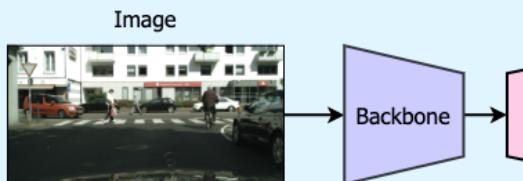
OneFormer: One Transformer to Rule Universal Image Segmentation

Jitesh Jain^{1,2}, Jiachen Li^{1*}, MangTik Chiu^{1*}, Ali Hassani¹, Nikita Orlov³, Humphrey Shi^{1,3}

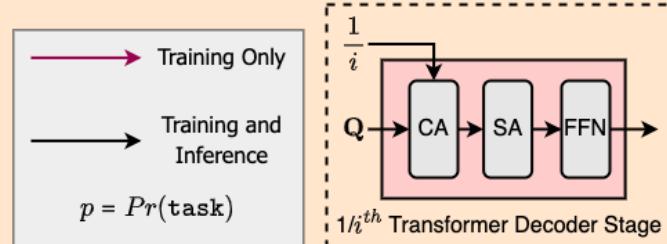
¹SHI Labs @ U of Oregon & UIUC, ²IIT Roorkee, ³Picsart AI Research (PAIR)

<https://github.com/SHI-Labs/OneFormer>

(a) Multi-Scale Feature Modeling



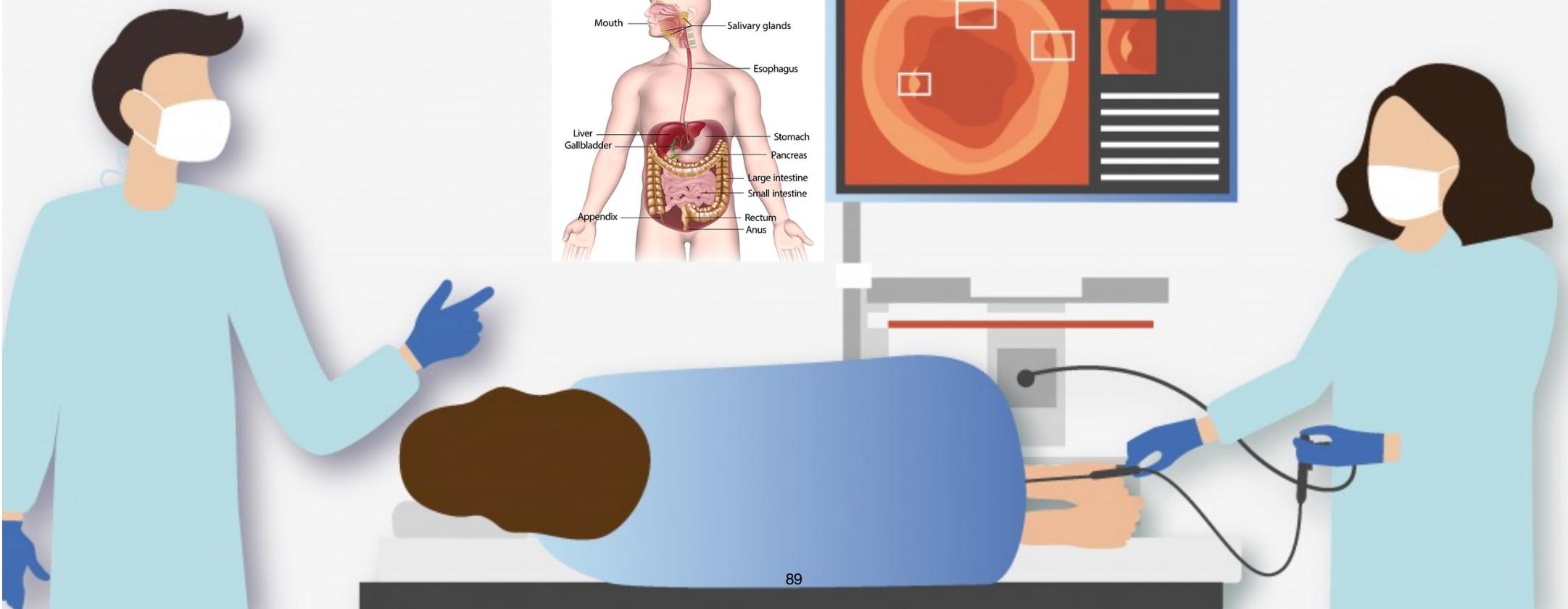
(b) Unified Task-Conditioned Query Formulation



(c) Task-Dynamic Mask and Class Prediction Formation

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.

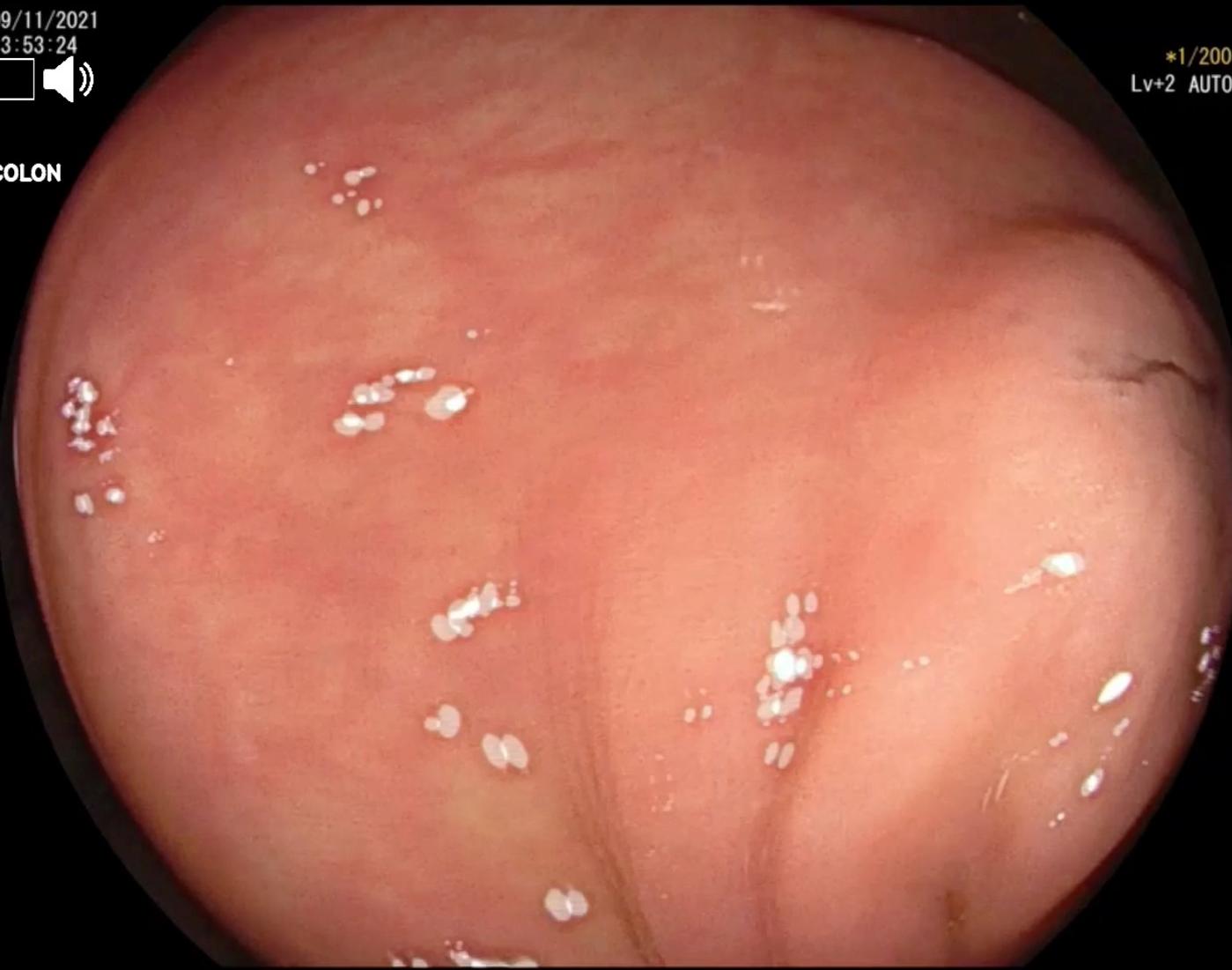


09/11/2021

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

threshold : 0.4
class : NBI

Name:

Sex: Age:

D.O.B.:

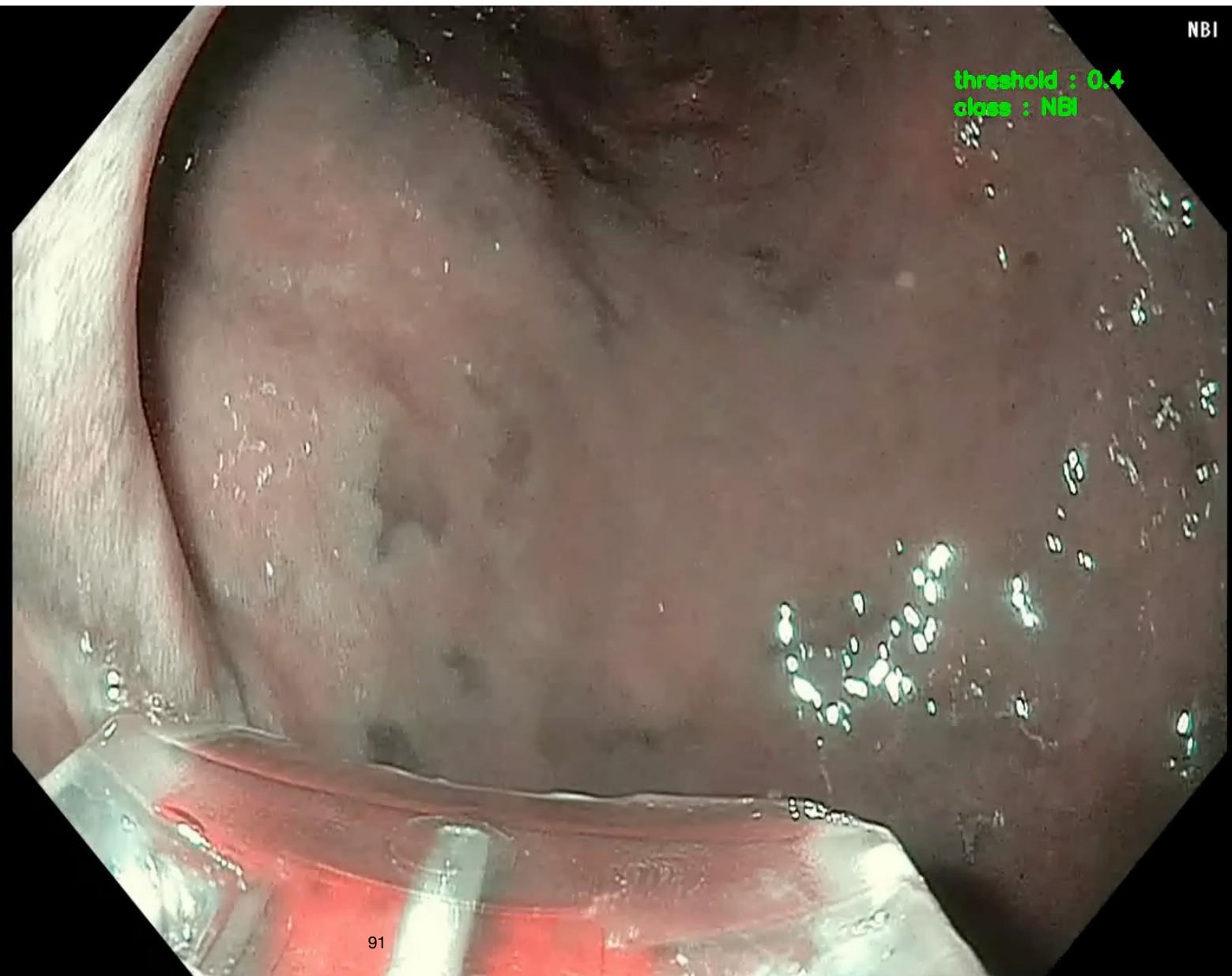
18/05/2021

08:29:36

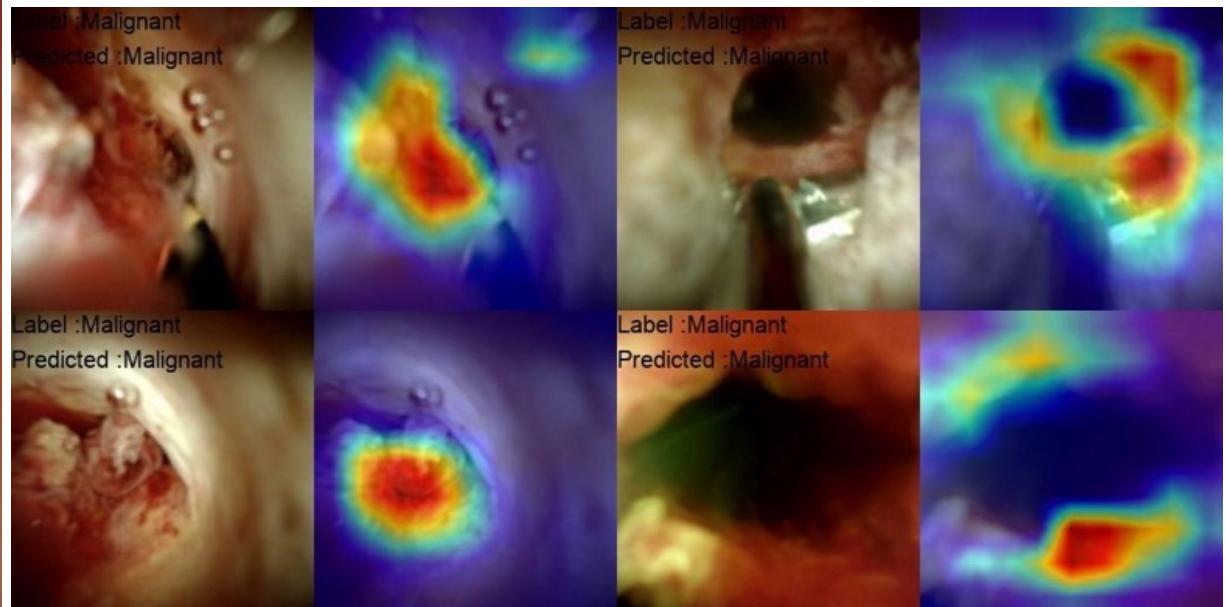
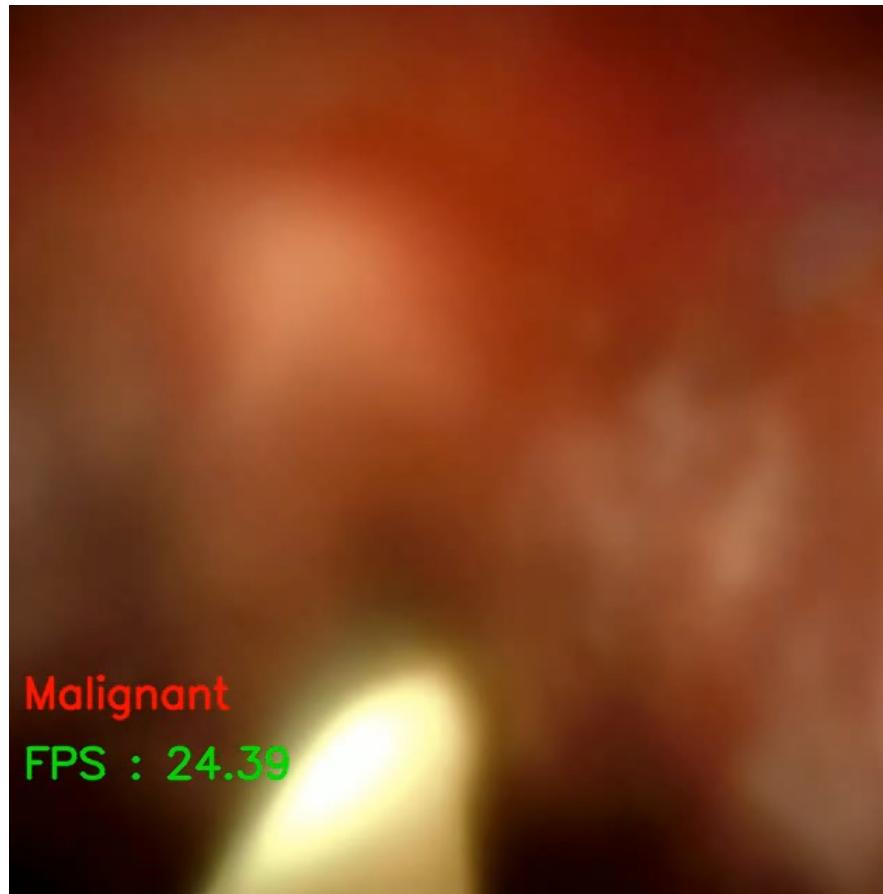
■■□/---(0/1)

Eh:B8 Cm:1

Comment:



Cholangioscopy



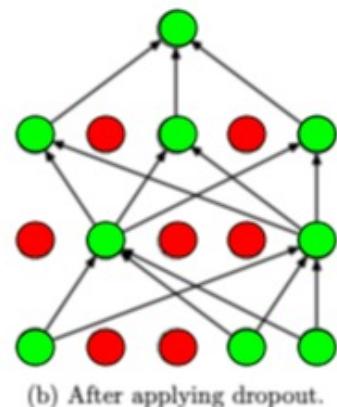
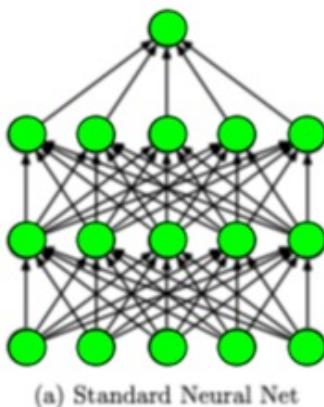


More techniques to prevent overfitting

- Dropout
- Augmentation
- Best validation loss

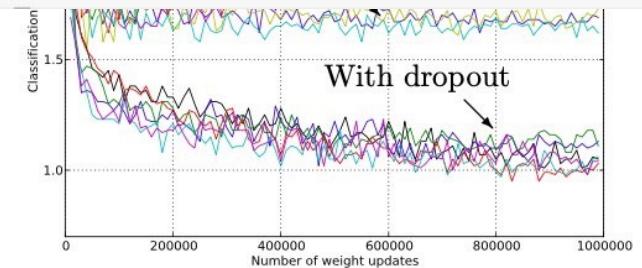
Dropout

- The Dropout layer prevents **overfitting** the model during training. Notably, Dropout **randomly deactivates** some neurons of a layer, thus nullifying their contribution to the output.
- Dropout is only used in training, so we don't want these weights to be fixed at this high a number during testing.



```
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Dropout(.5, noise_shape=None, seed=None)) #dropout layer
model.add(Conv2D(32, kernel_size=3, activation='relu'))

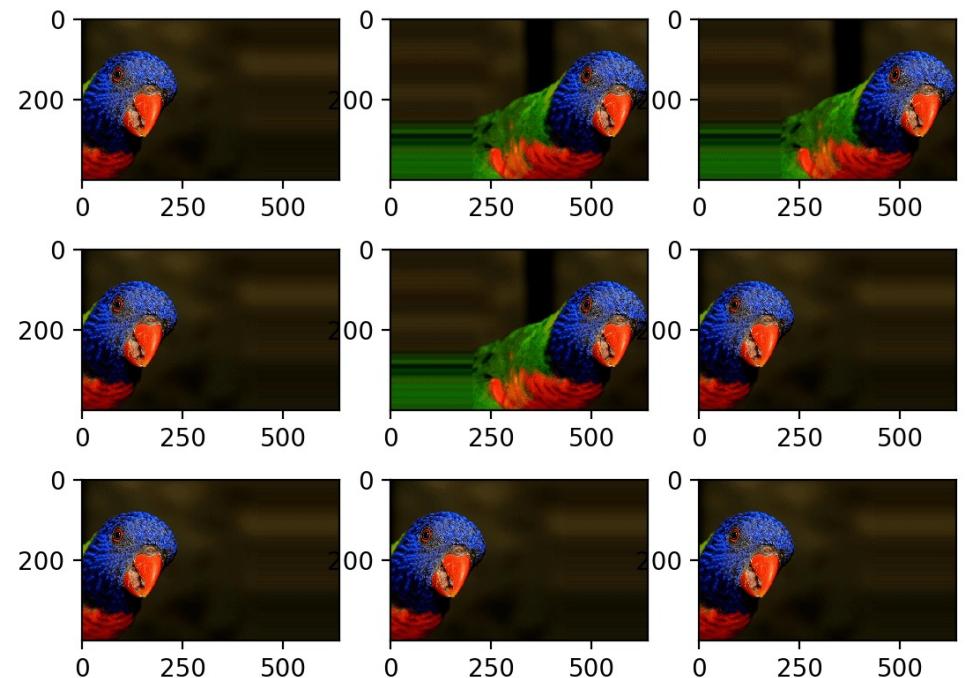
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```



<https://deepakbattini.medium.com/implementing-drop-out-regularization-in-neural-networks-ab2fc8d985e8>



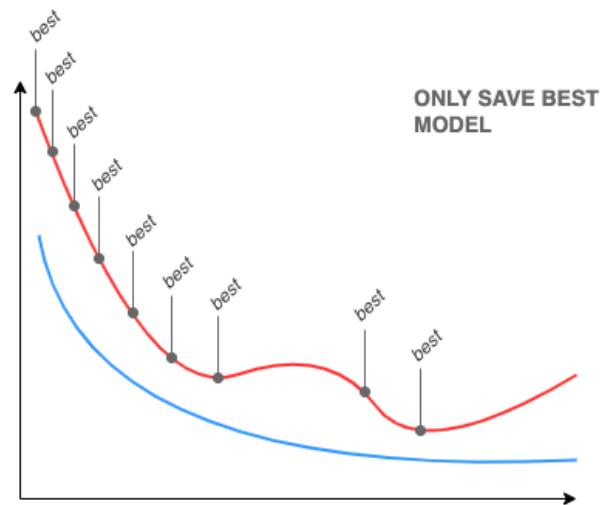
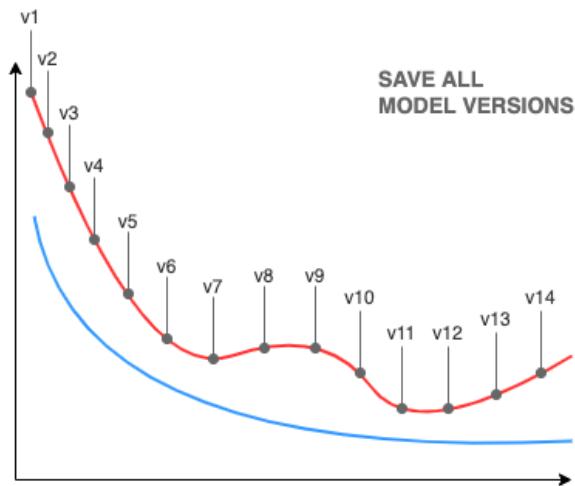
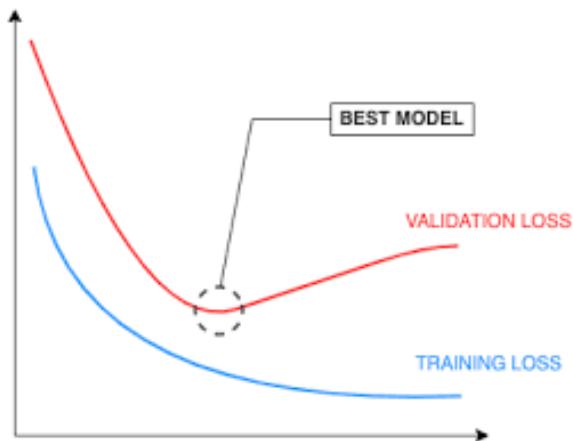
Augmentation



95



Best validation loss

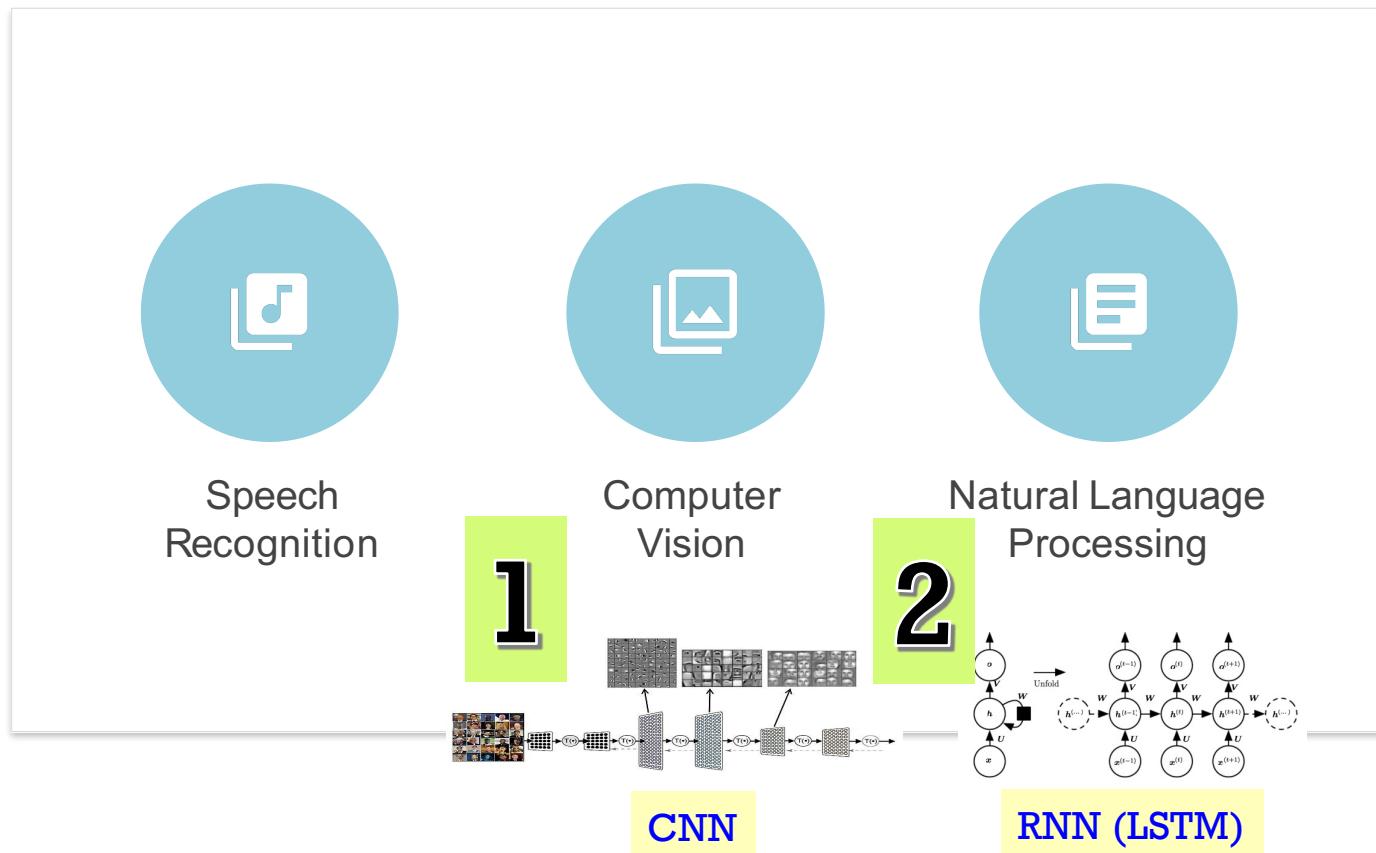


<https://nicjac.dev/posts/identify-best-model/>

+

RNN

Deep Learning Application



Top 10 Strategic Technology Trends for 2020

People-centric



Hyperautomation

Smart spaces



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



Total experience strategy



Privacy-enhancing computing



Anywhere operations



Cybersecurity mesh



AI engineering

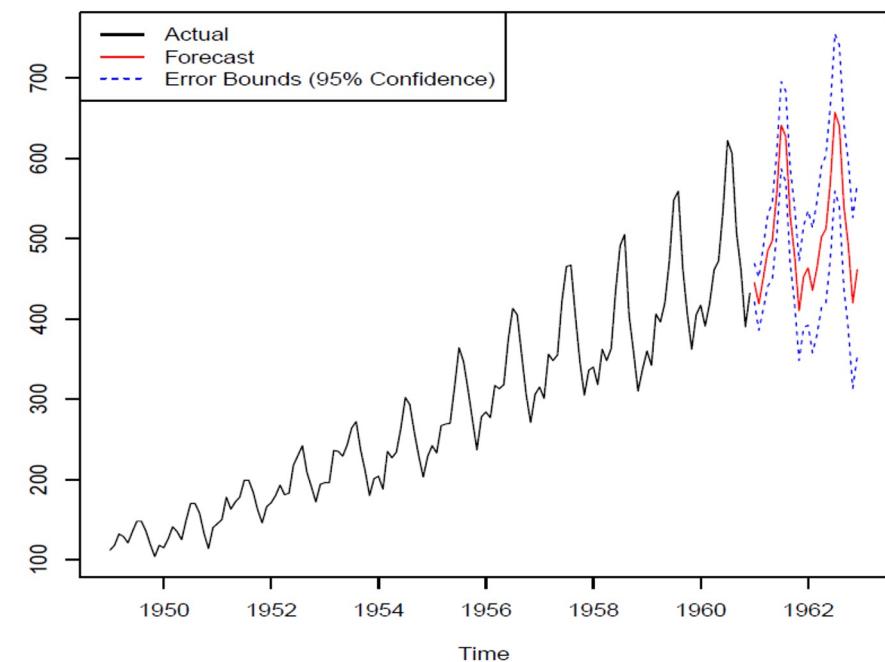


Hyperautomation

Combinatorial innovation

gartner.com/SmarterWithGartner

Gartner



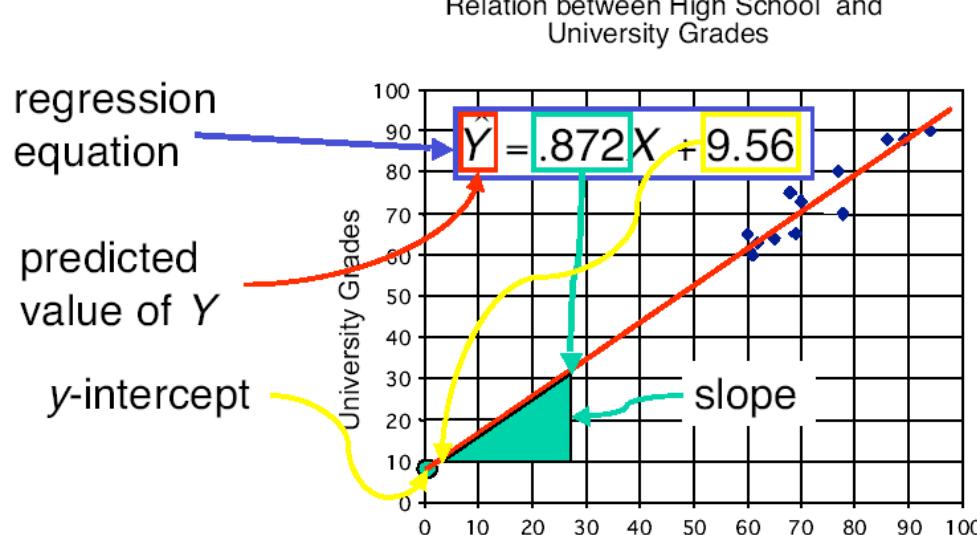
Source: Gartner
© 2019 Gartner, Inc. and/or its affiliates. All rights reserved. CTMKT_34600

Source: Gartner
© 2020 Gartner, Inc. and/or its affiliates. All rights reserved. CTMKT_3026461



Regression – Linear Relationship

100



weight, coefficient

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

target intercept input

- The least square method aims to minimize the following term

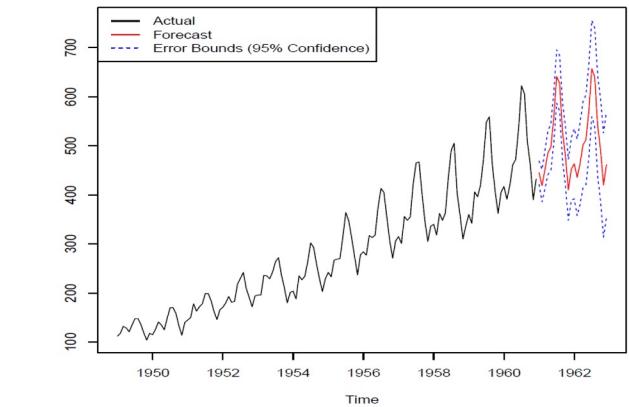
$$\sum_{\text{training data}} (y_i - \hat{y}_i)^2$$

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

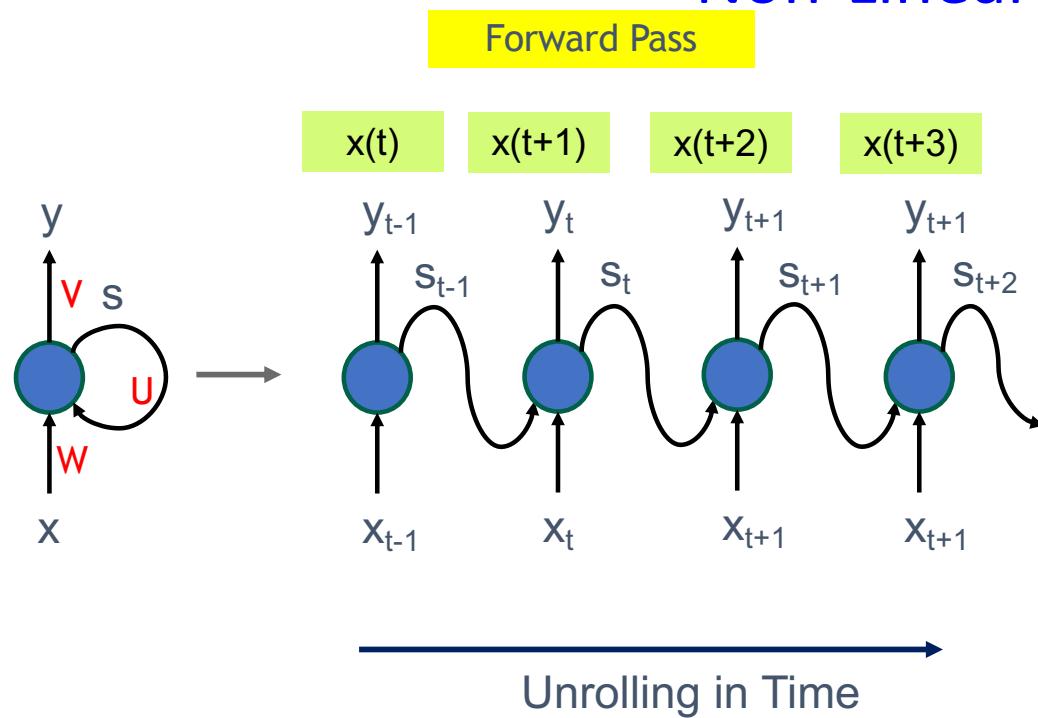
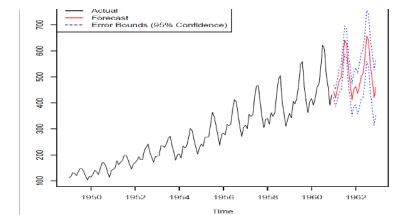
$$X_t = c + \varphi_1 X_{t-1} + \varphi_2 X_{t-2}$$

- 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)

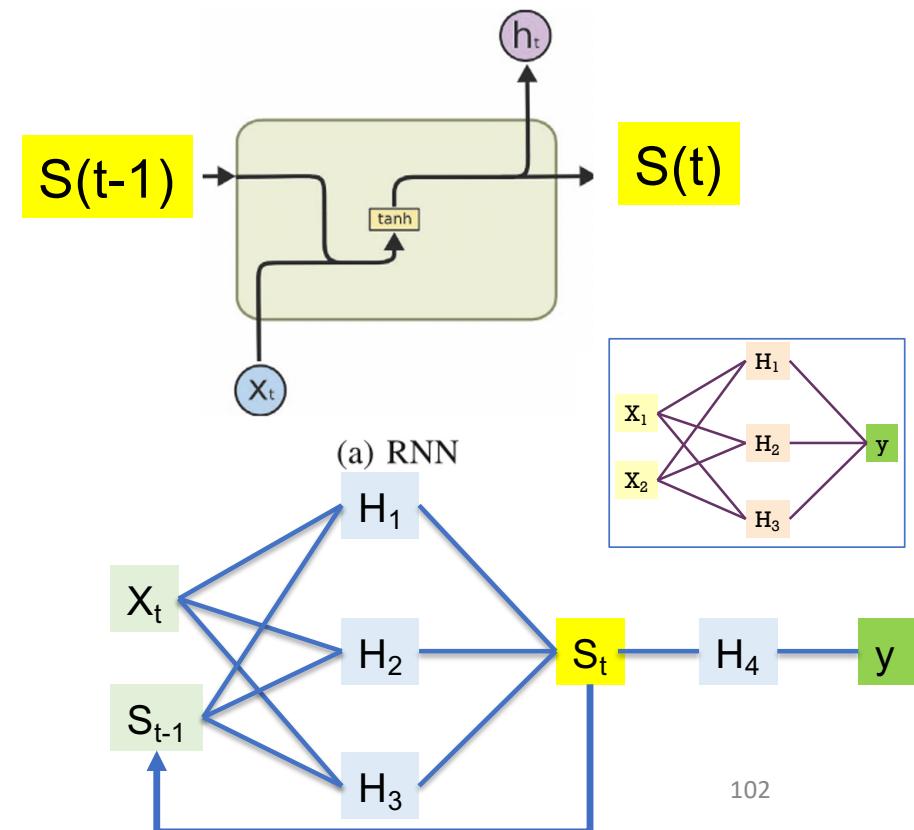
Non-Linear Relationship



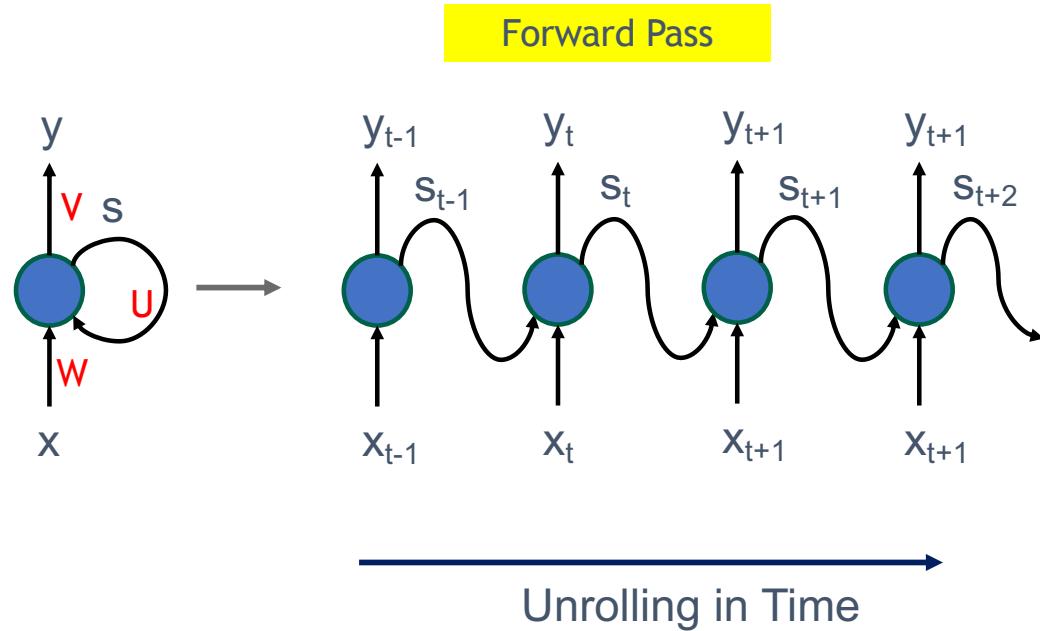
$$s_t = \tanh(Ux_t + Vs_{t-1})$$

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

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

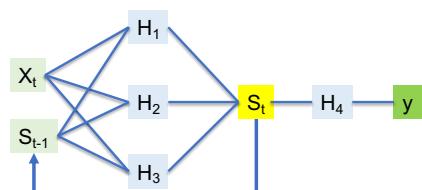


RECURRENT NEURONS (RNN)

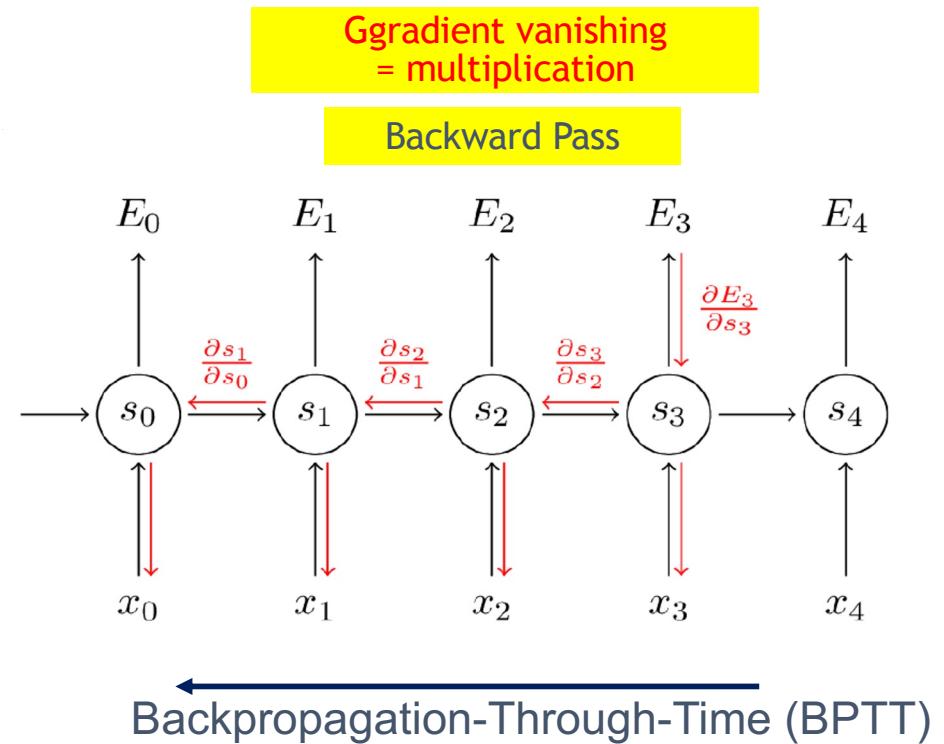


$$s_t = \tanh(Ux_t + Vs_{t-1})$$

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



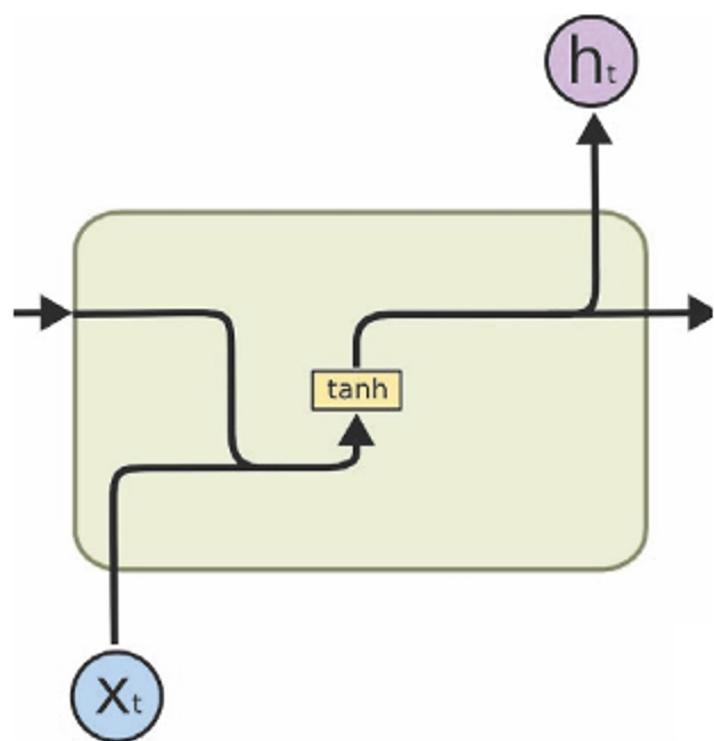
Reference: <https://www.cse.iitk.ac.in/users/sigml/lec/Slides/LSTM.pdf>



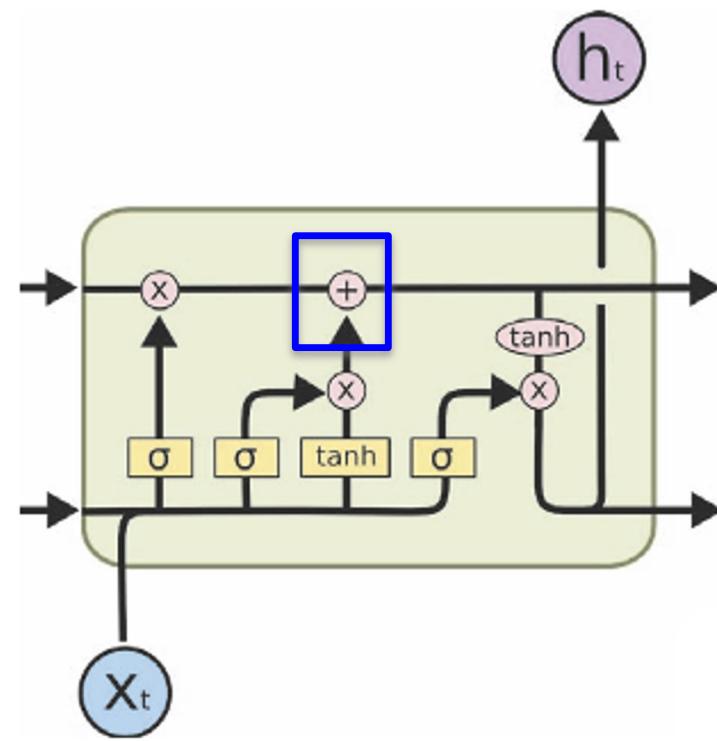
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

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

RNN VS LSTM



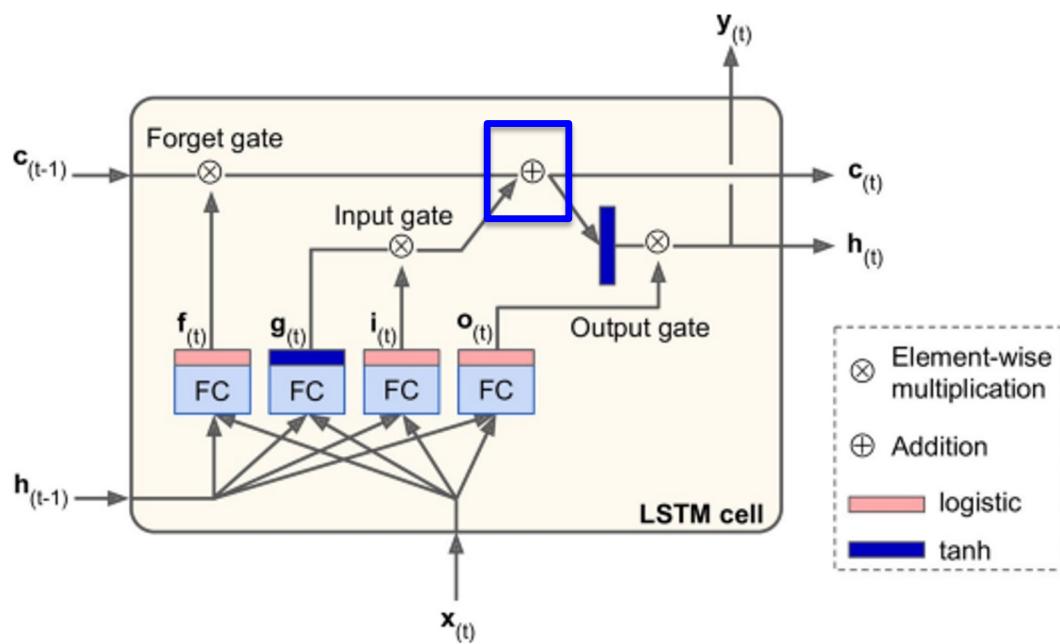
(a) RNN



(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.

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

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

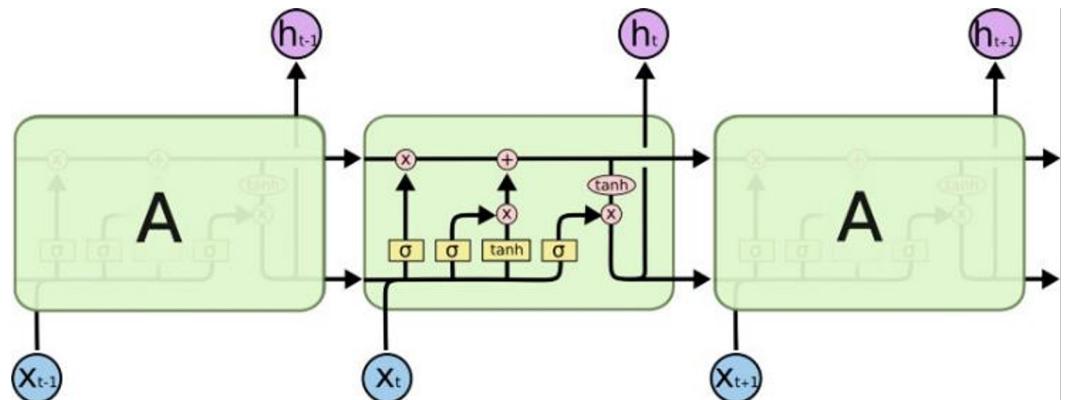
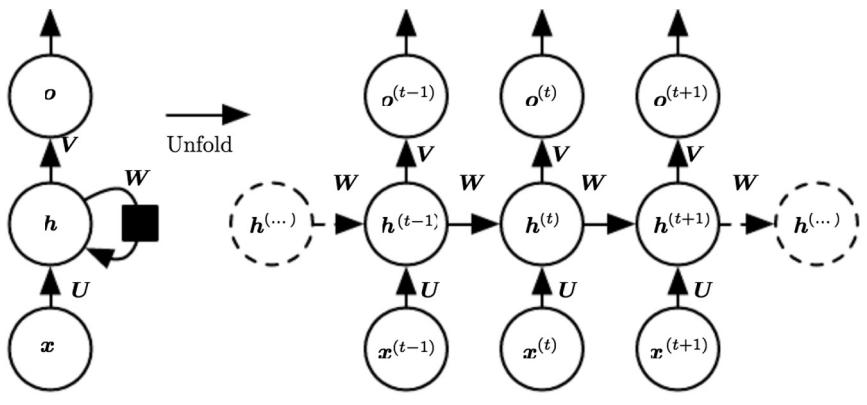
$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

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

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

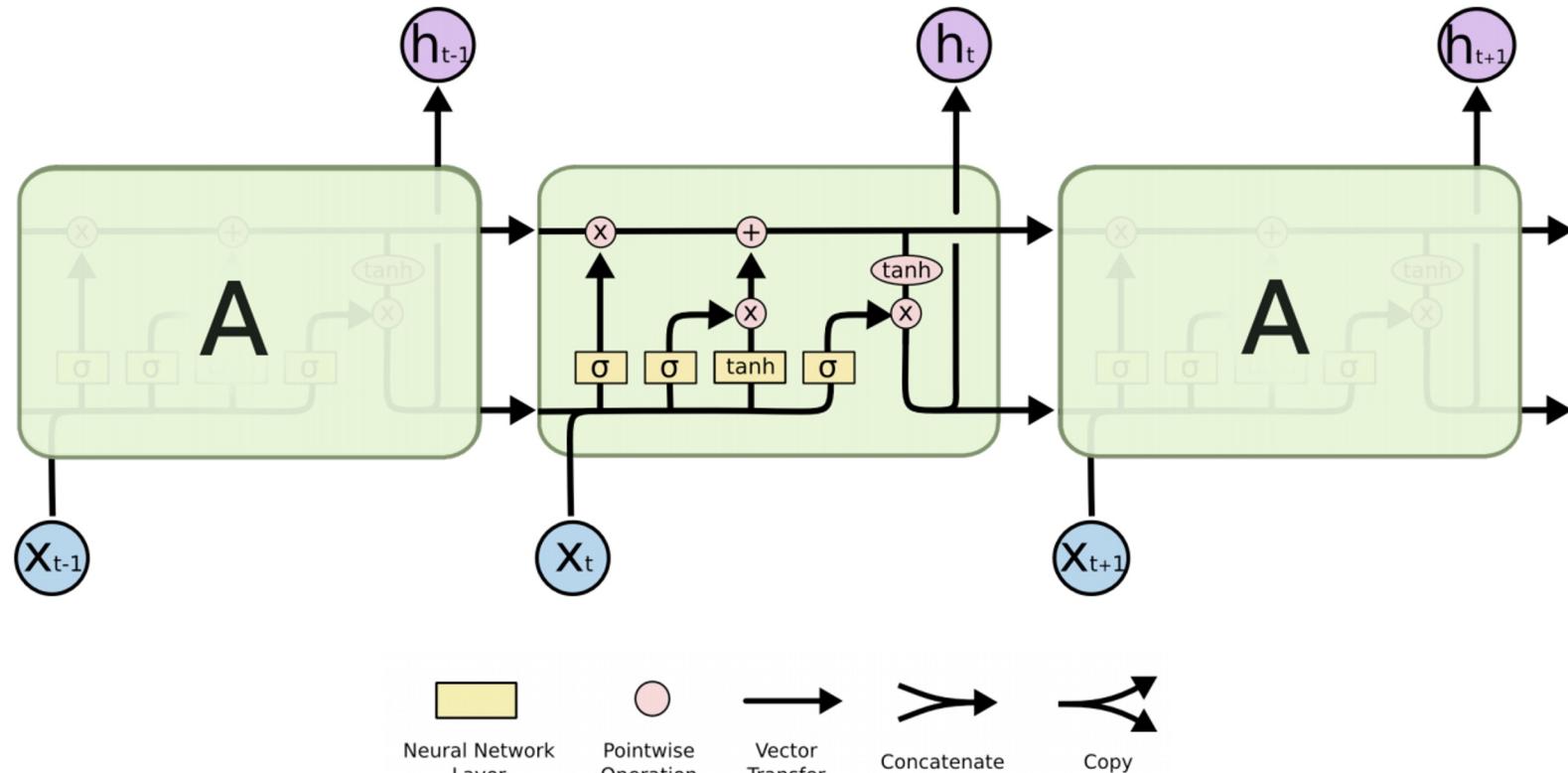
$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

Deep Learning Approach: LSTM

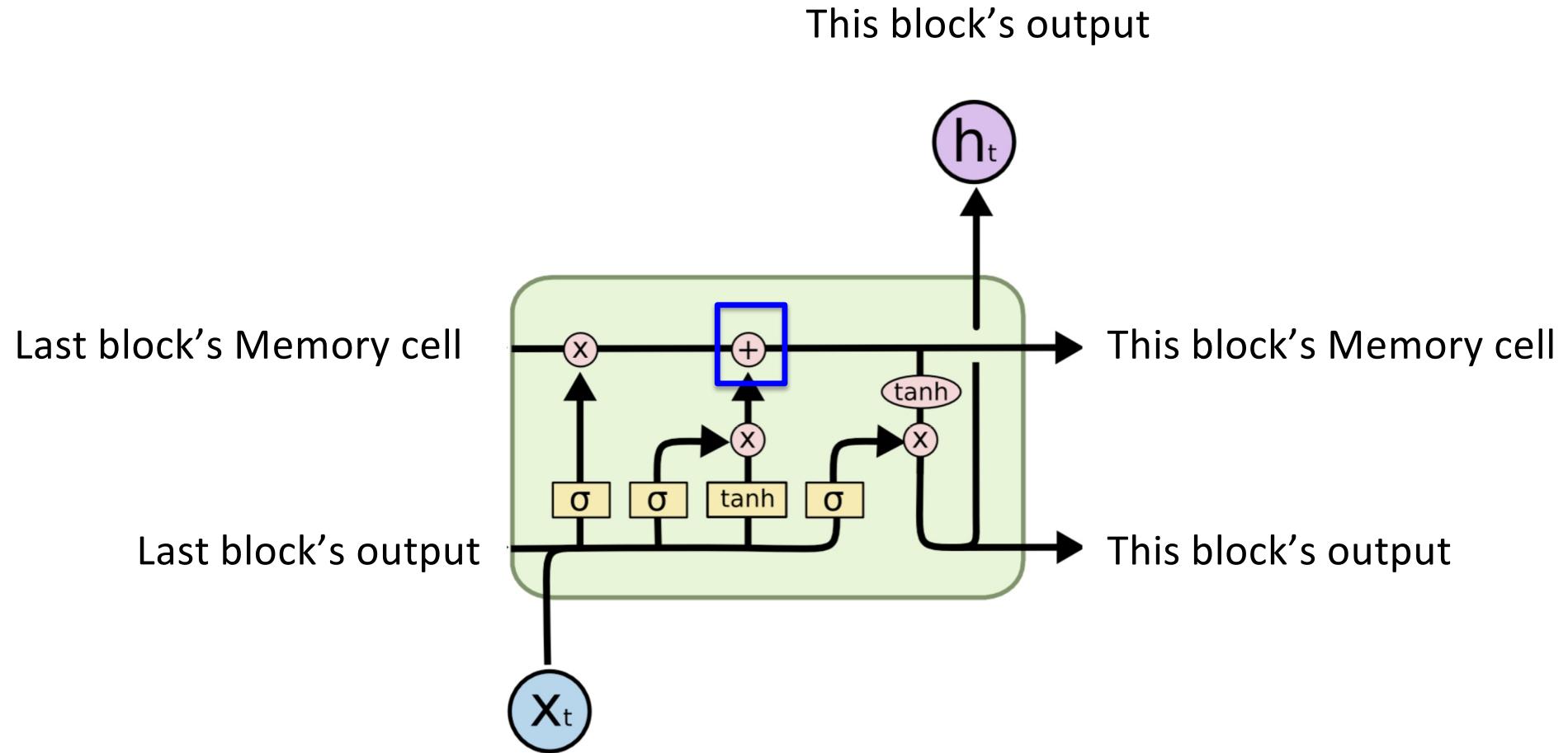


Reference: Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. MIT Press.
<http://www.deeplearningbook.org>

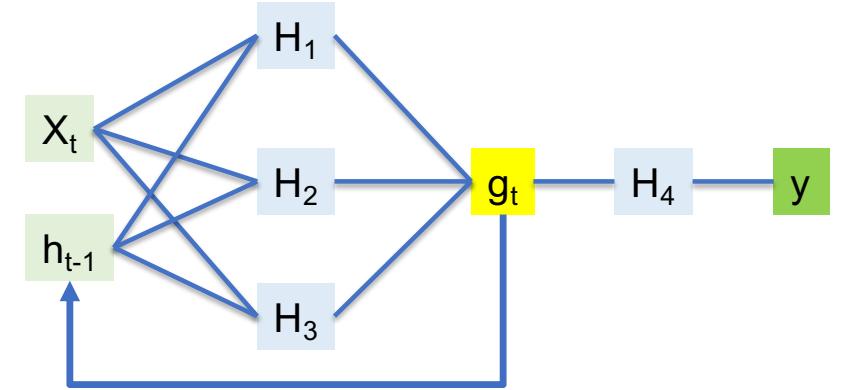
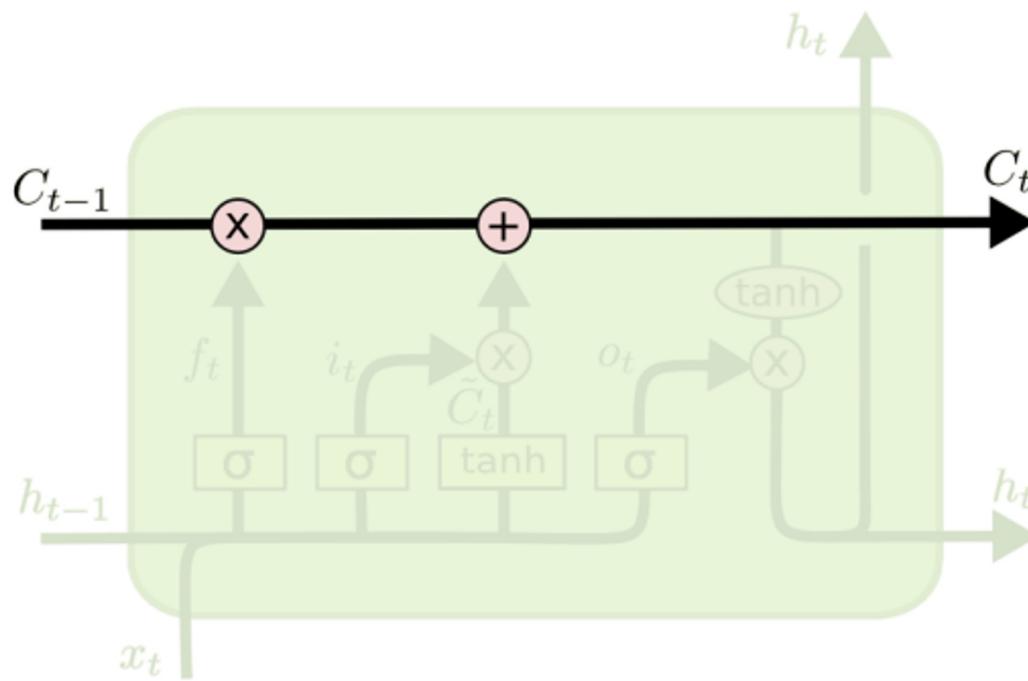
Inside LSTM



Inside LSTM



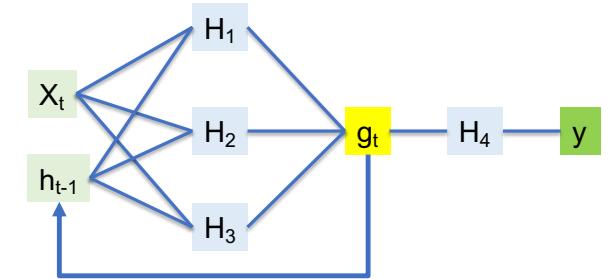
Inside LSTM: (1) Cell Memory



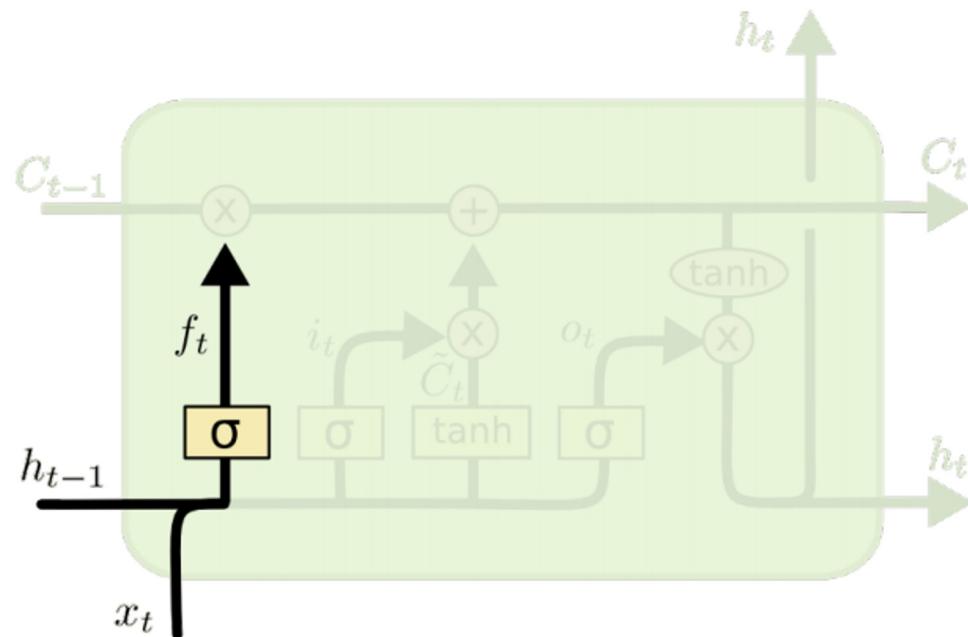
There are 3 gates with 2 outputs.

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

Inside LSTM: (2) Forget Gate



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

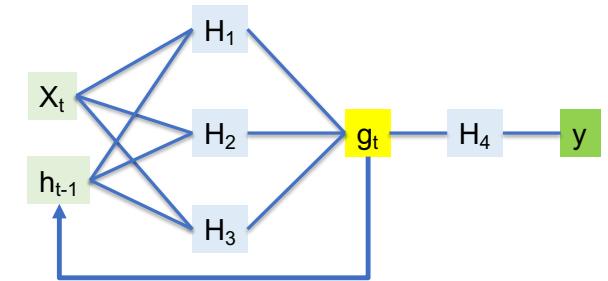
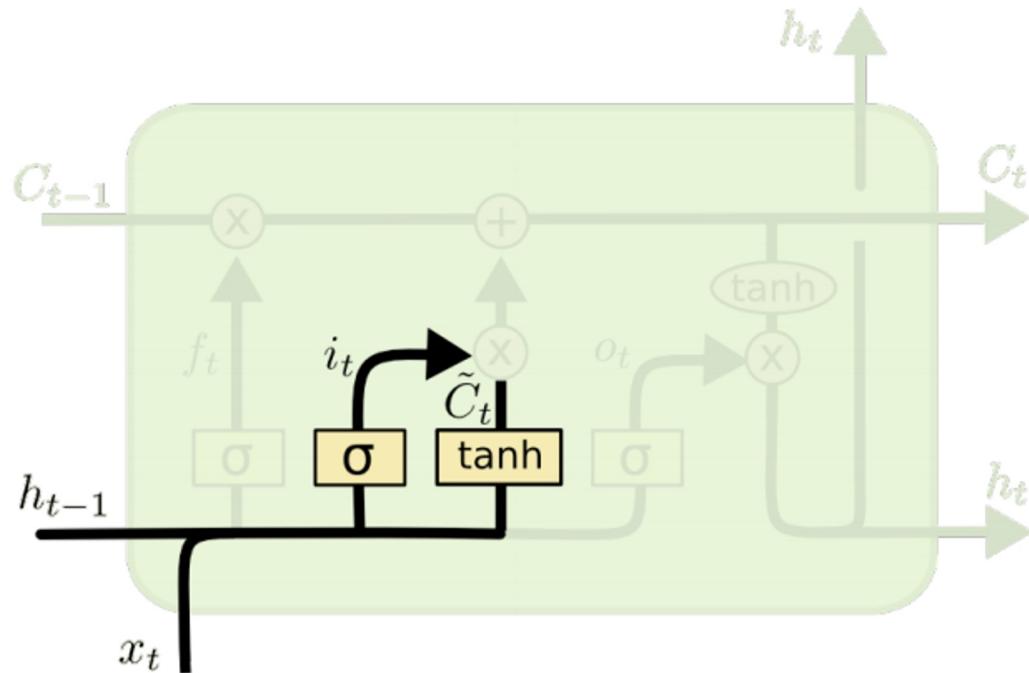


There are 3 gates with 2 outputs.

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

Inside LSTM: (3) Input Gate

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

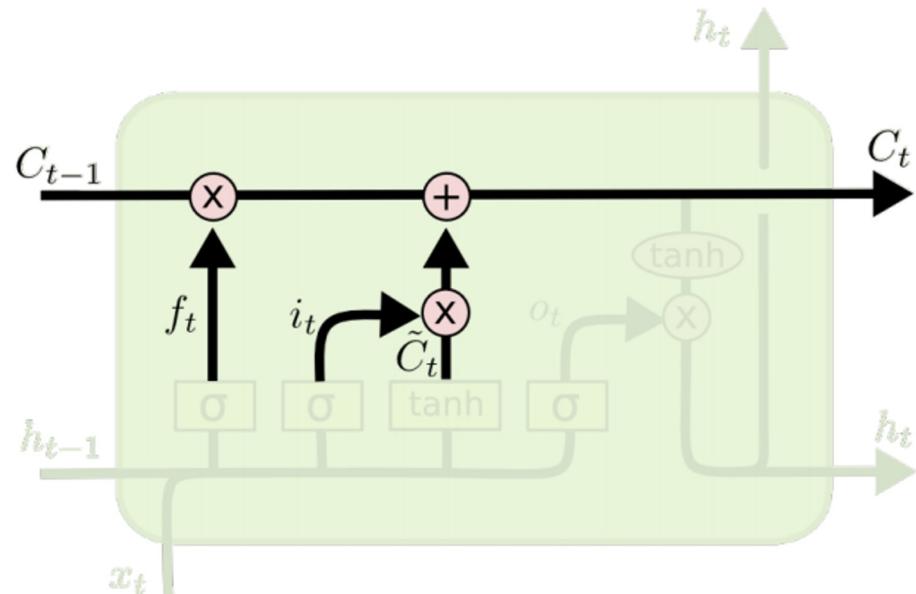


There are 3 gates with 2 outputs.

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

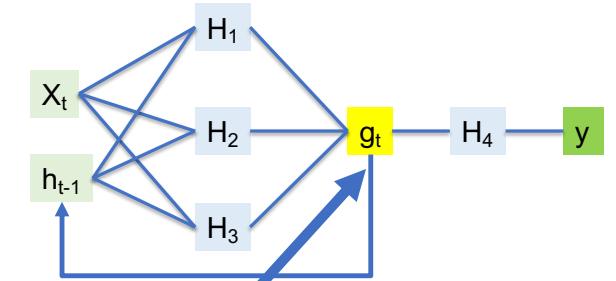
Inside LSTM: (4) Memory Update

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



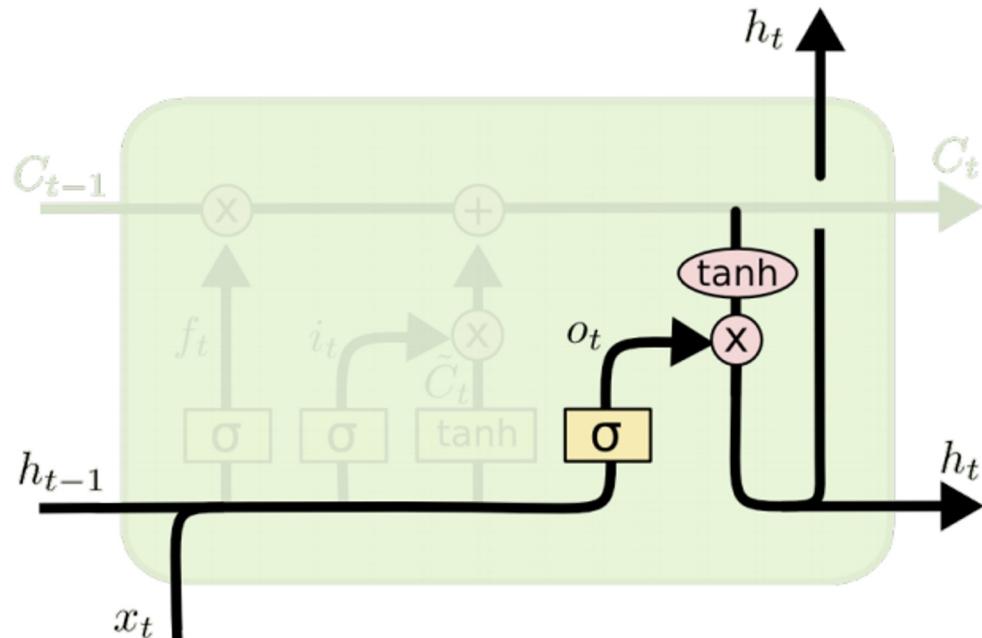
There are 3 gates with 2 outputs.

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



Inside LSTM: Output Gate

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

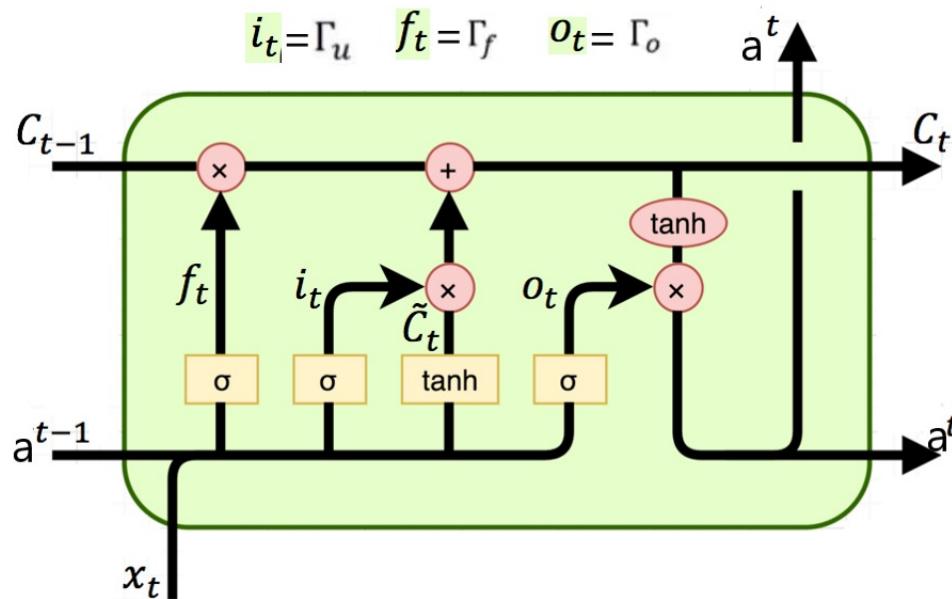


There are 3 gates with 2 outputs.

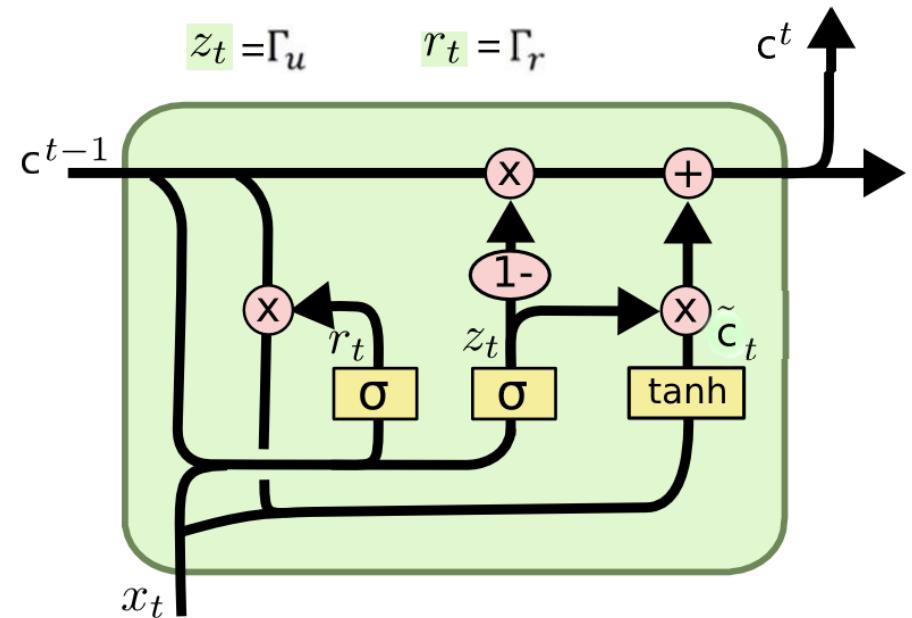
$$\begin{aligned}\mathbf{i}_{(t)} &= \sigma(\mathbf{W}_{xi}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i) \\ \mathbf{f}_{(t)} &= \sigma(\mathbf{W}_{xf}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f) \\ \mathbf{o}_{(t)} &=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

LSTM vs. Gated Recurrent Units (GRU)

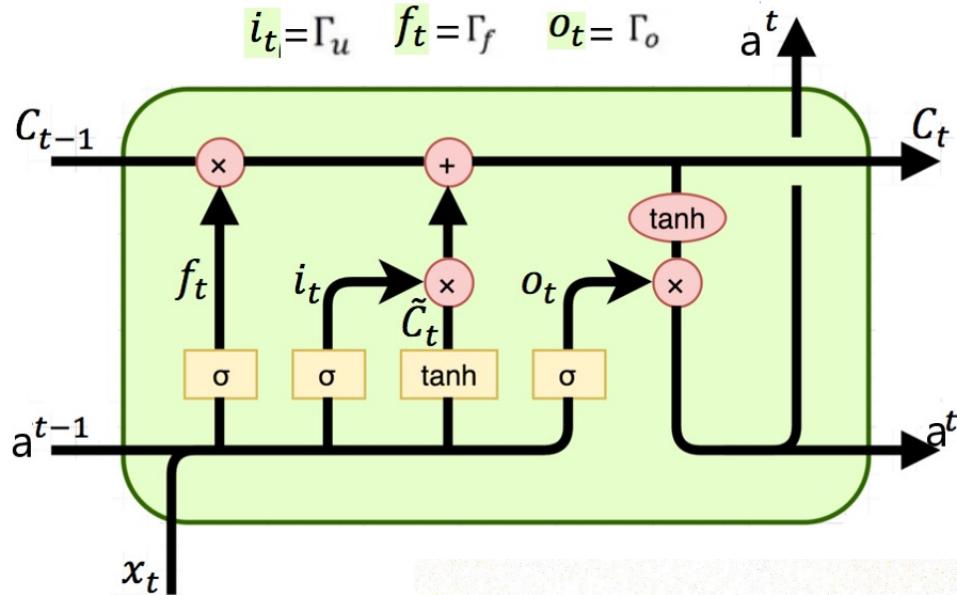
There are 3 gates (i_t , f_t , o_t)
 2 outputs ($C(t)$, $a(t)$),
 3 inputs ($x(t)$, $C(t-1)$, $a(t-1)$)



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



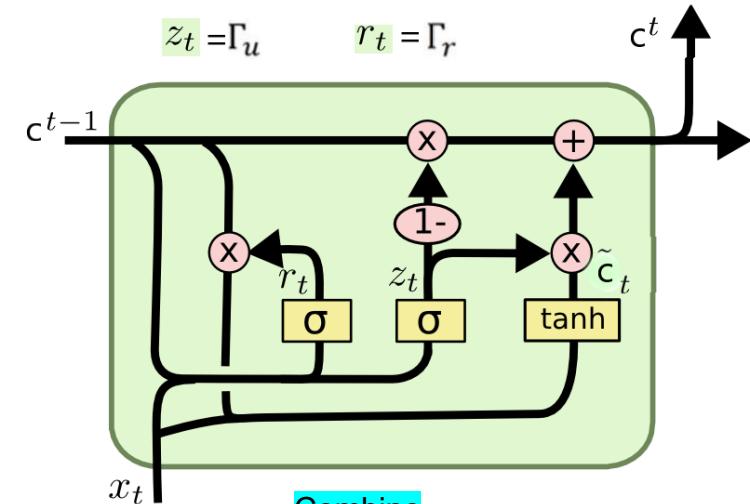
There are 3 gates with 2 outputs.



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

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 $\rightarrow a^{(t)} = T_o * \tanh(c^{(t)})$

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



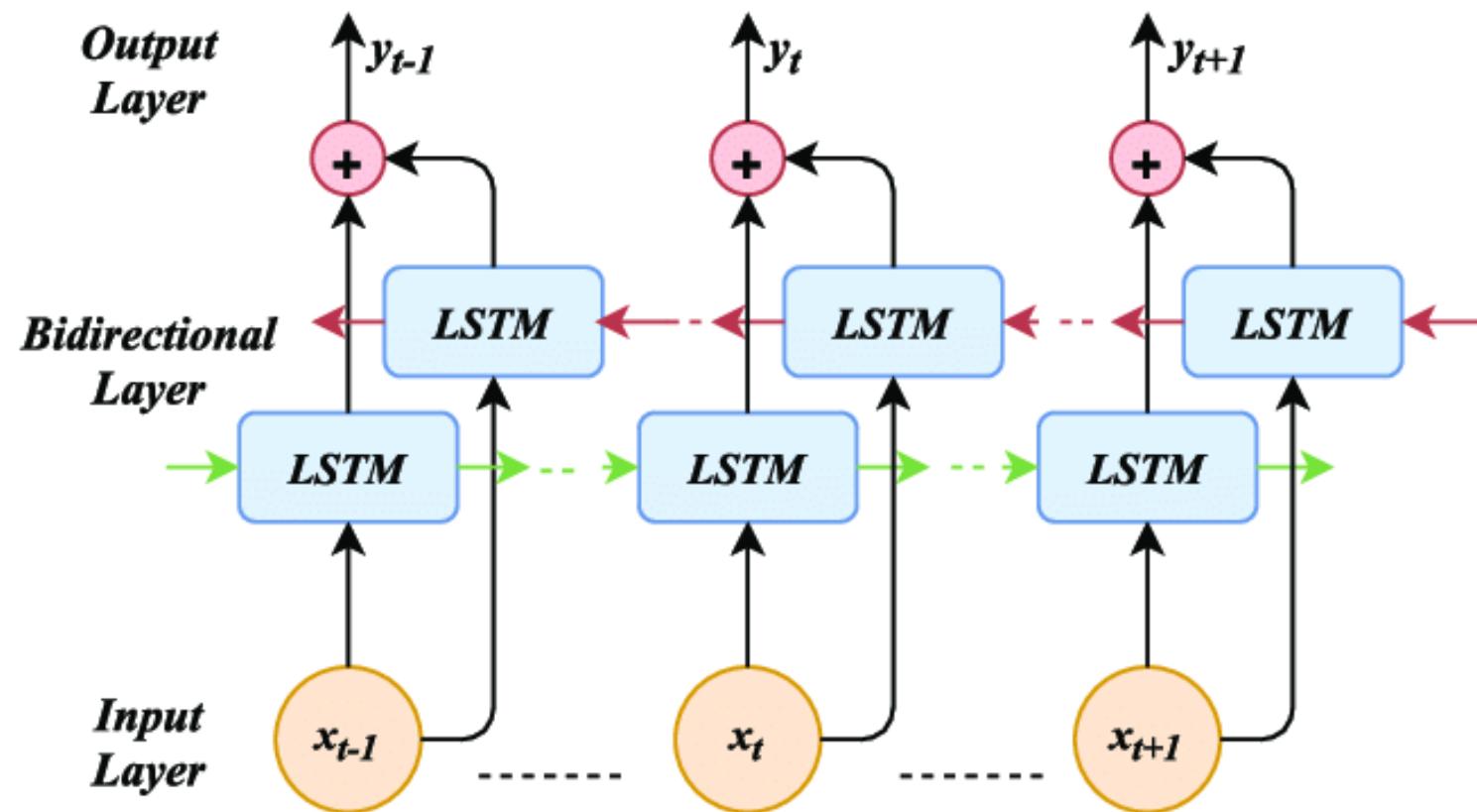
Combine

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

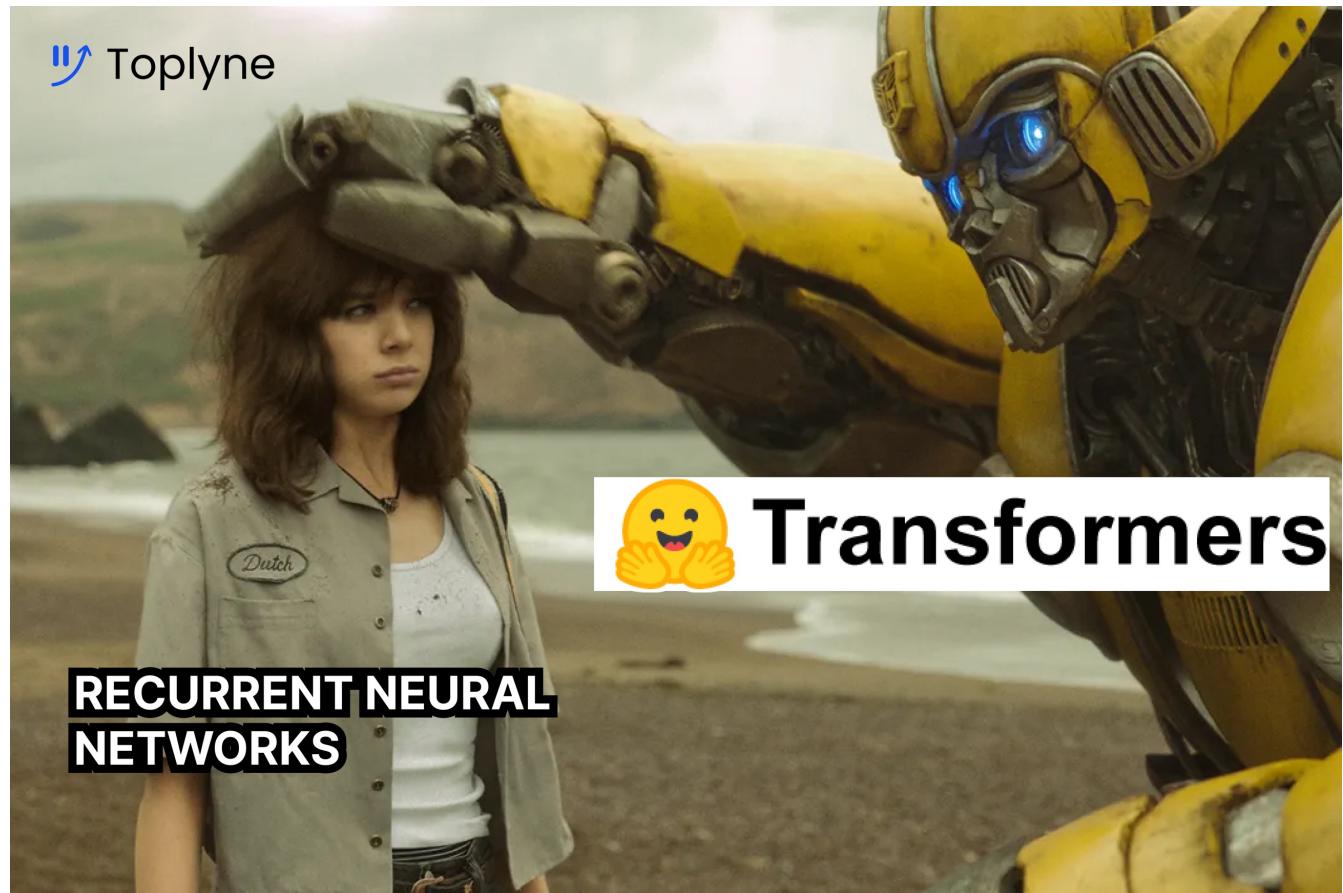
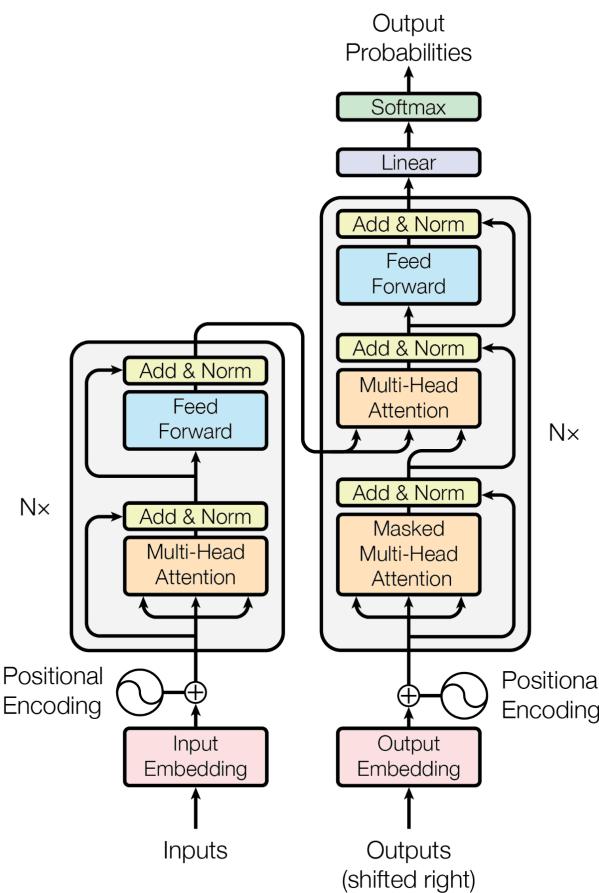
Candidate cell $\rightarrow \tilde{c}^{(t)} = \tanh(W_c[T_r * c^{(t-1)} + x^{(t)}] + b_c)$
Update gate $\rightarrow T_u = \sigma(W_u[c^{(t-1)} + x^{(t)}] + b_u)$
Reset gate $\rightarrow T_r = \sigma(W_r[c^{(t-1)} + x^{(t)}] + b_r)$
Cell state $\rightarrow c^{(t)} = T_u * \tilde{c}^{(t)} + (1 - T_u) * c^{(t-1)}$
Activation $\rightarrow a^{(t)} = \tilde{c}^{(t)}$

Update $g(t)$ & $C(t-1)$

Bidirectional LSTM



Rise of Transformer (2017)

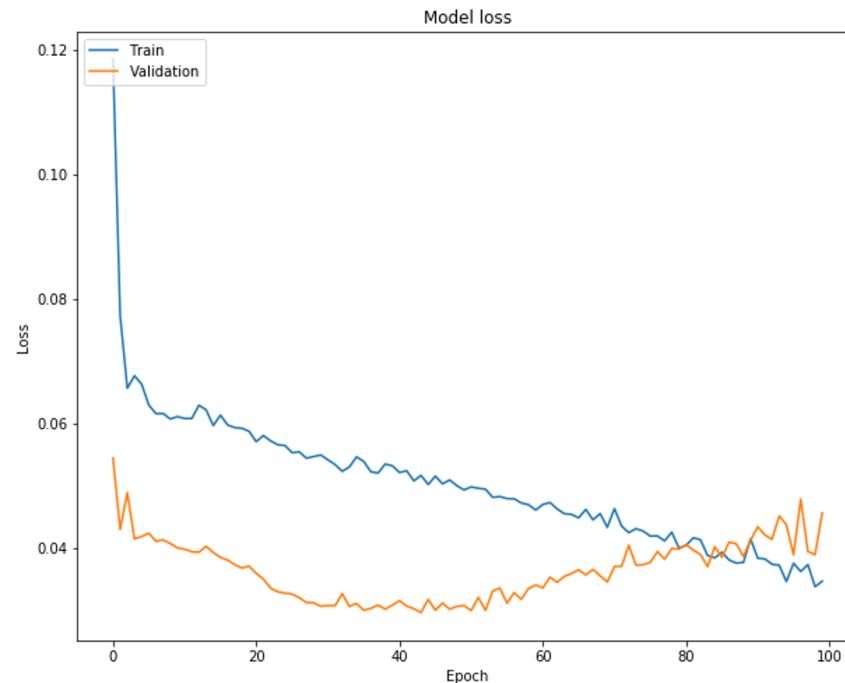


LSTM Parameters Tuning

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

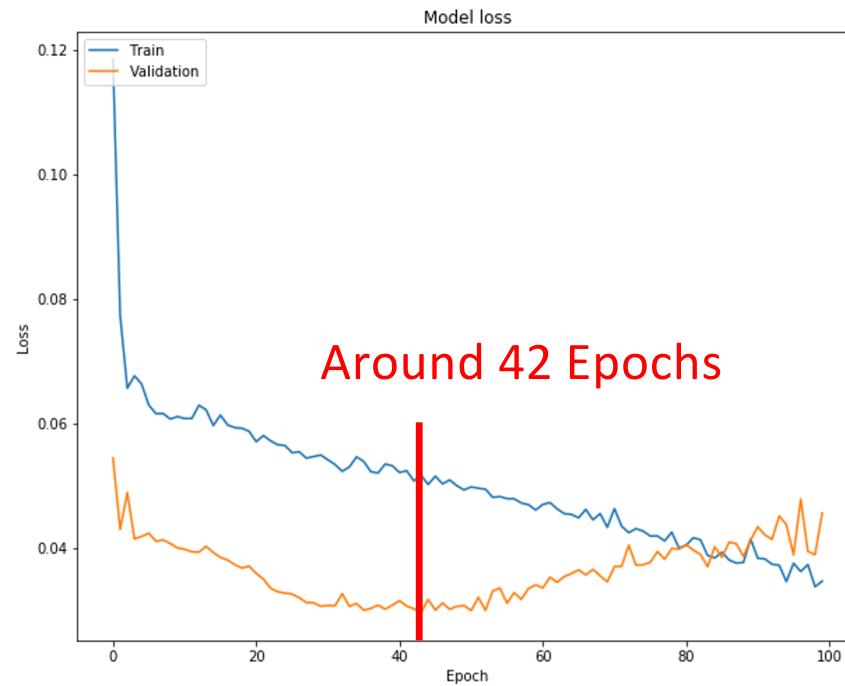
LSTM Parameters Tuning: Epoch

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



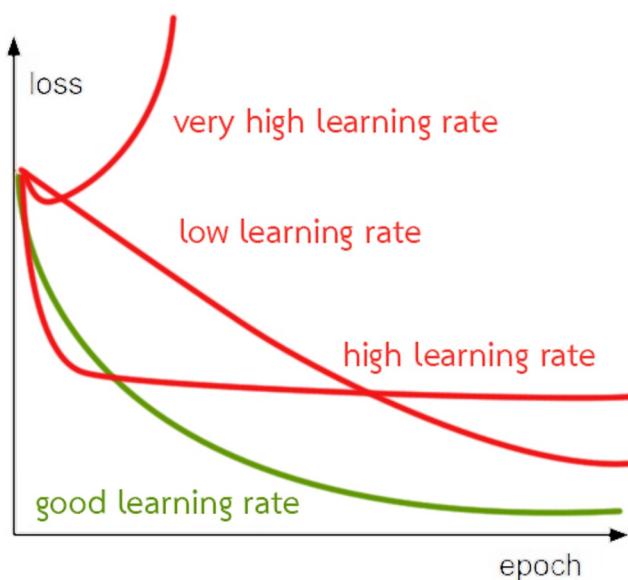
LSTM Parameters Tuning: Epoch (cont.)

- 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

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

NLP Lecture2 (2020s2) by Aj.Ekapol

<https://youtu.be/WxiO3wvKhCE>

The image shows a Zoom video call interface. In the top right corner, there is a small video window of a man speaking. The main area displays a presentation slide with the following content:

Dictionary-based drawbacks

- Cannot handle words outside of the ~~dictionary~~ (Out-of-Vocabulary, OOV words)
- Performs worse than machine-learning-based approach

Below the text is a graph showing a downward-sloping curve, likely representing performance or error rate versus vocabulary size. The x-axis is labeled "Vocabulary size" and the y-axis is labeled "Error rate". A red circle highlights a point on the curve. The graph has a legend at the bottom left: "OPTIMAL SIZE" with a blue arrow pointing to a point on the curve, and "FRESHNESS AND TIME-FRAME FOR VOICE SEARCH VOCABULARY" with a red arrow pointing to the x-axis.

Maryam Kamvar, Ciprian Chiba, OPTIMAL SIZE, FRESHNESS AND TIME-FRAME FOR VOICE SEARCH VOCABULARY

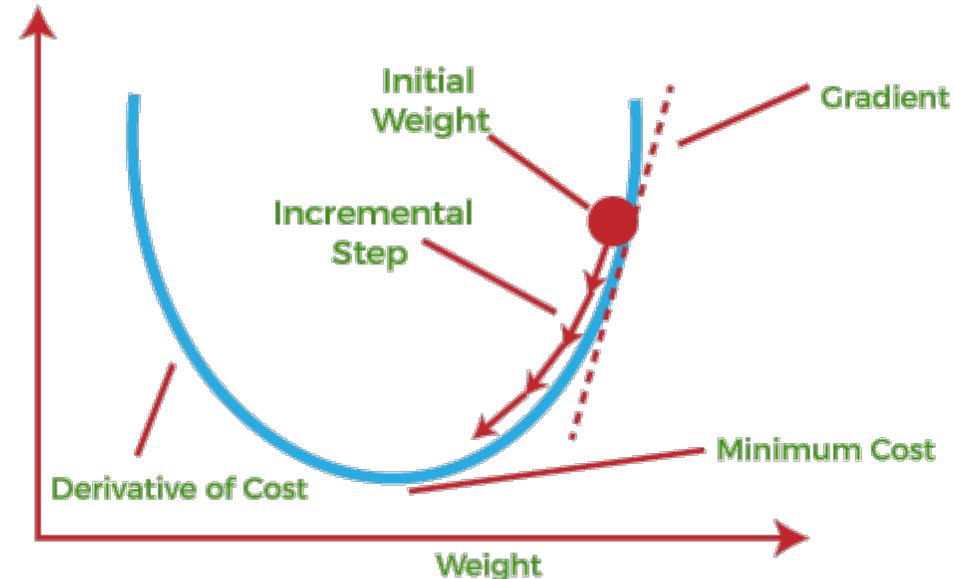
zoom

+

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**

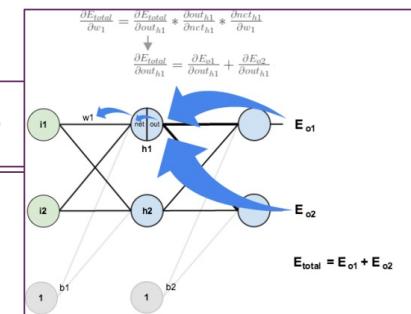


$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Putting it all together:

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

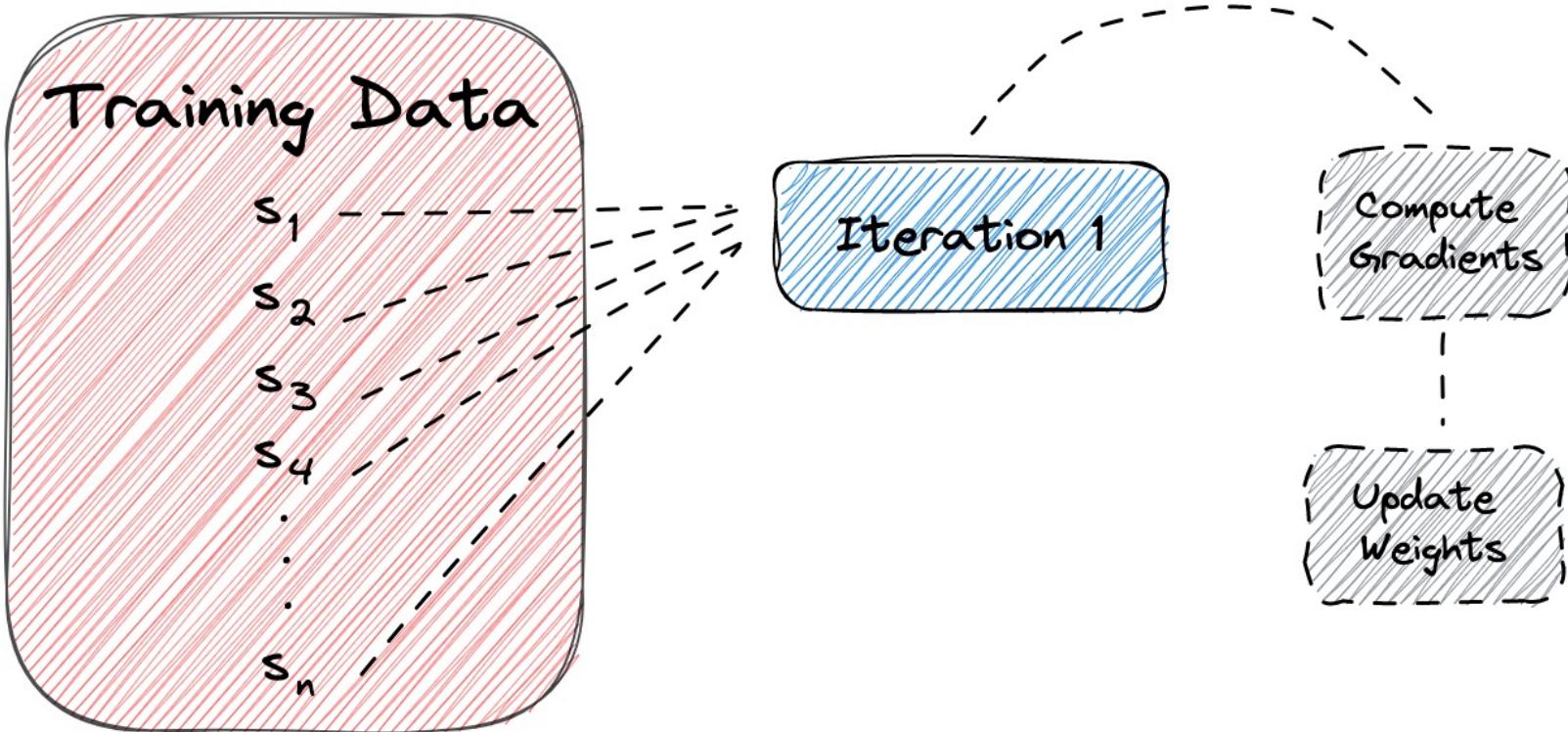
$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$



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

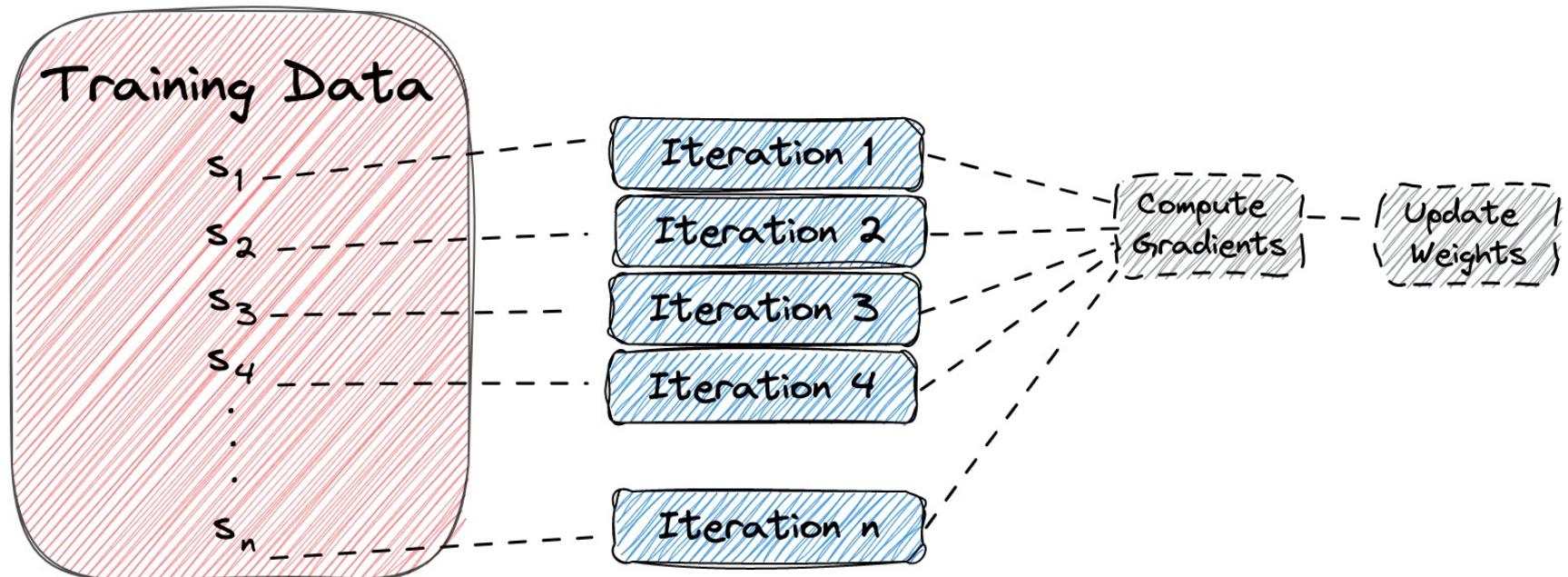
1) Batch Gradient Descent

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



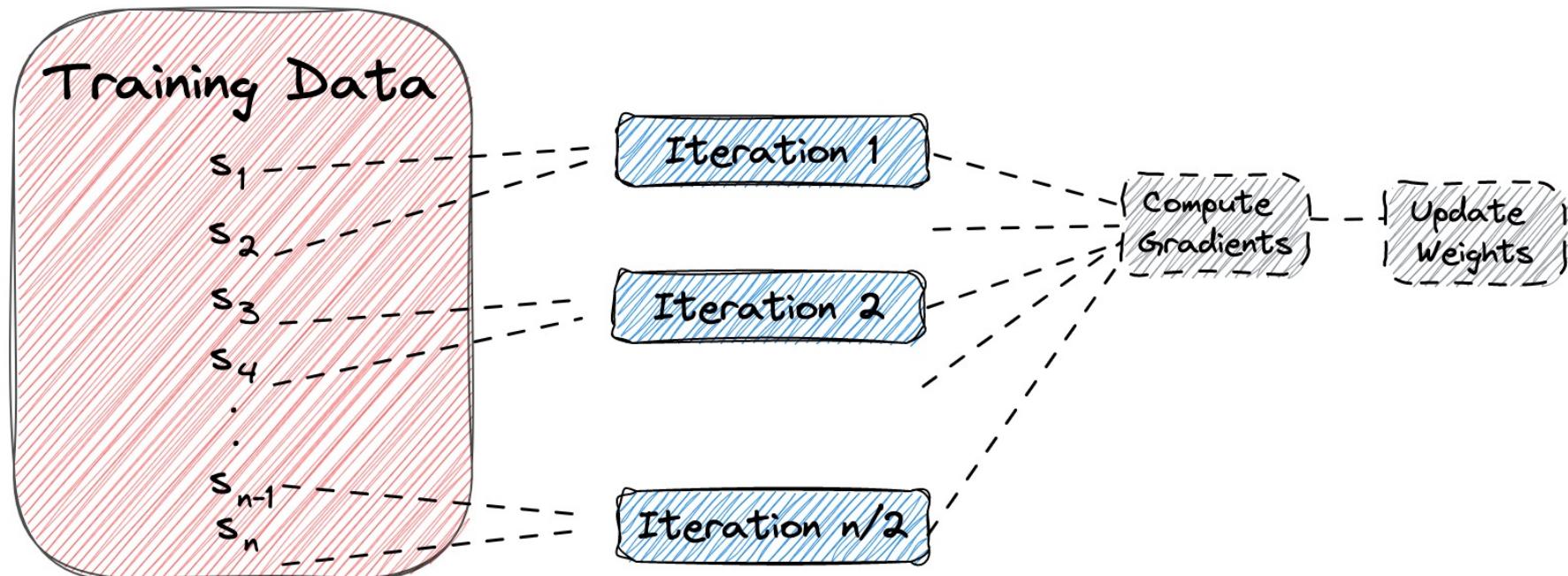
2) Stochastic Gradient Descent

- 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 $n = 2000$ samples, and we want to train a deep learning model using gradient descent for 10 epochs and mini-batch size $b = 4$:

- 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.