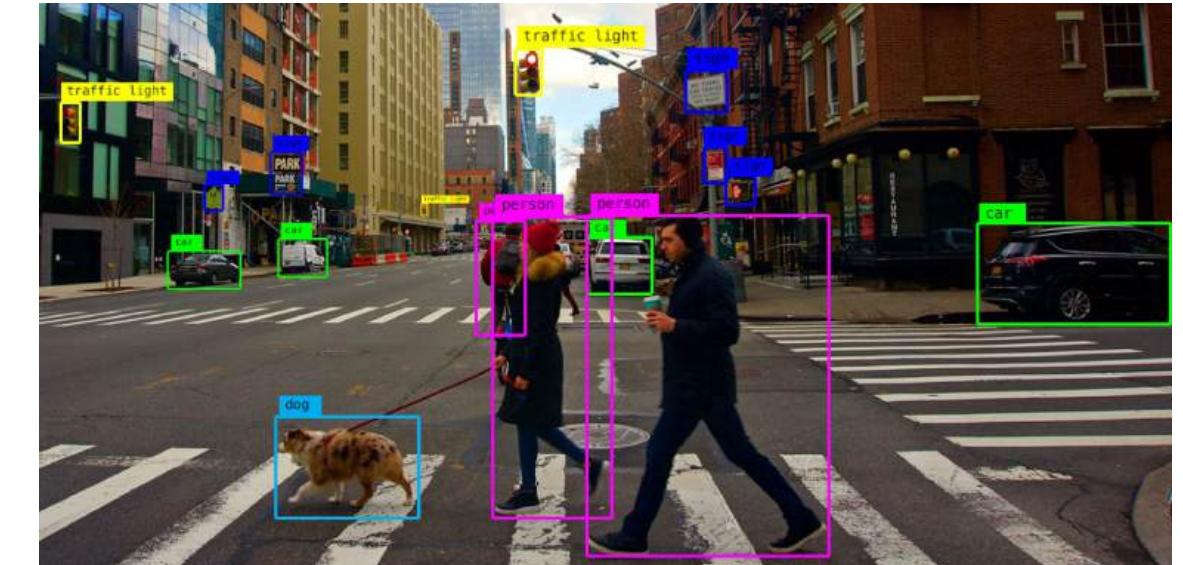


Object Detection

(Part 1: Traditional Algorithms)



Code & Data



Vinh Dinh Nguyen - PhD in Computer Science

Outline



Introduction to Object Detection

Applications of Object Detection

Object Detection Milestones

Traditional Object Detectors

CNNs for Image Classification

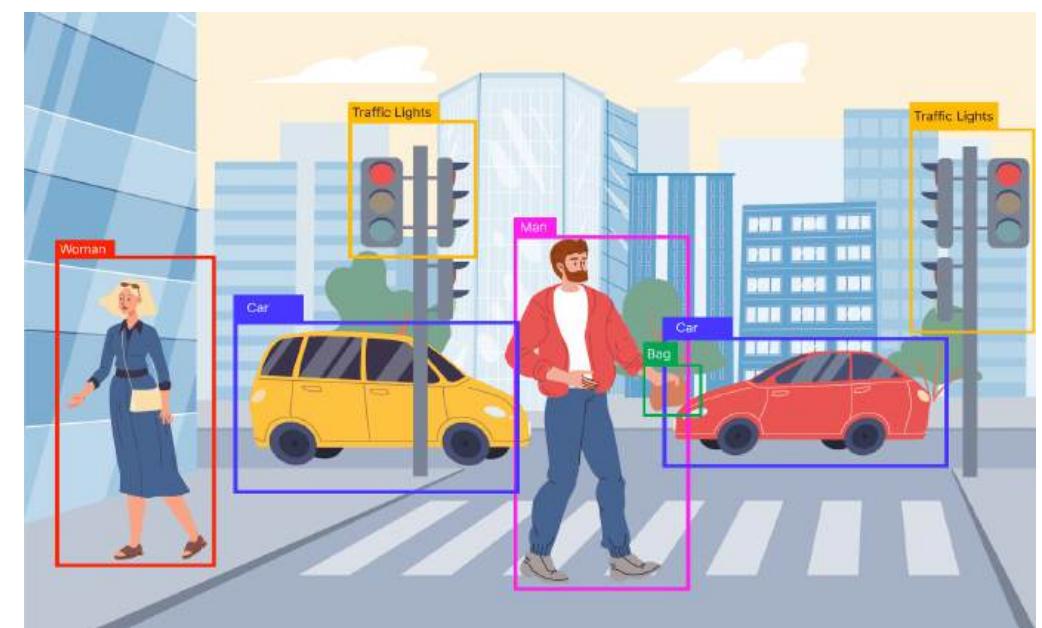
Image Classifier to Object Detectors with CNN

CNN Limitations and Spatial Outputs

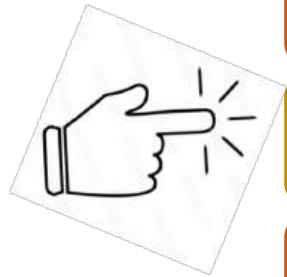
What is an Object Detection?

The objective of object detection is to develop **computational models** and **techniques** that provide one of the most basic pieces of information needed by computer vision applications:

What objects are where?



Outline



Introduction to Object Detection

Applications of Object Detection

Object Detection Milestones

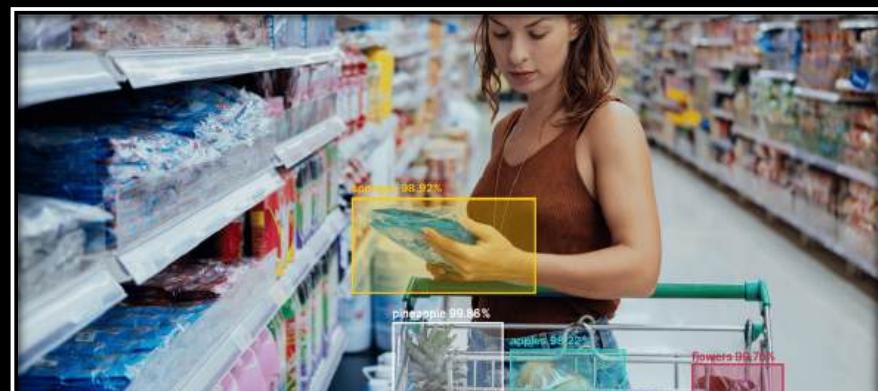
Traditional Object Detectors

CNNs for Image Classification

Image Classifier to Object Detectors with CNN

CNN Limitations and Spatial Outputs

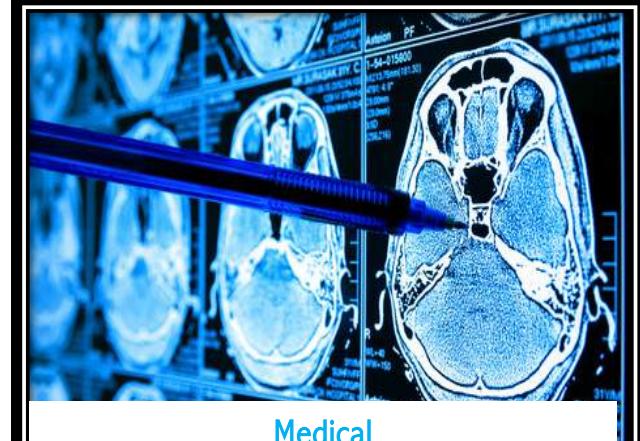
Object Detection for Businesses



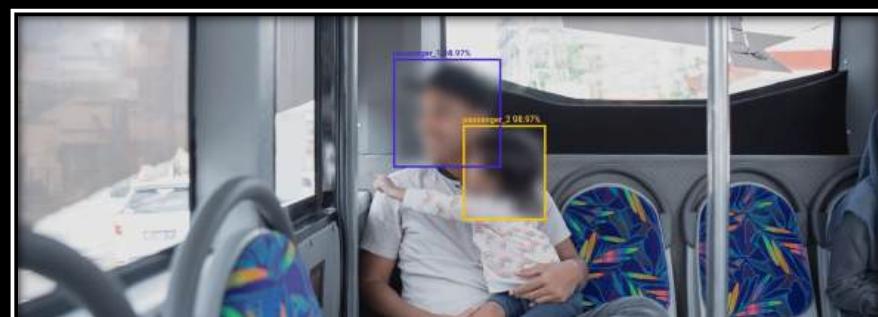
Retail



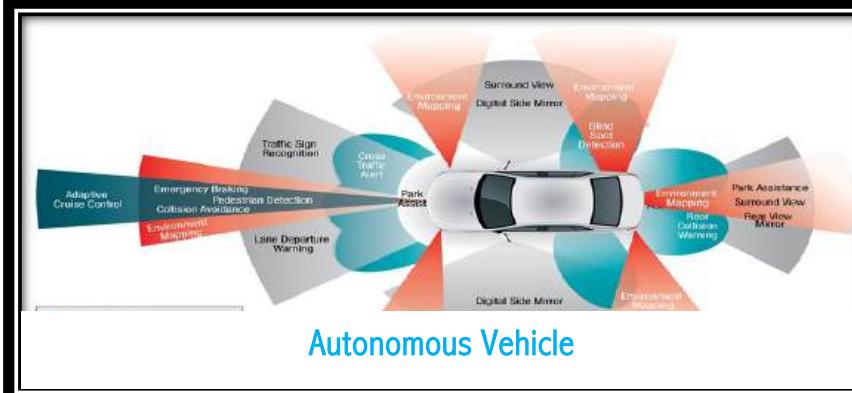
Industrial Use Cases



Medical



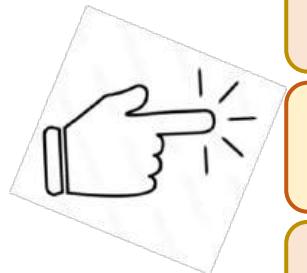
Transportation



Autonomous Vehicle

...

Outline



Introduction to Object Detection

Applications of Object Detection

Object Detection Milestones

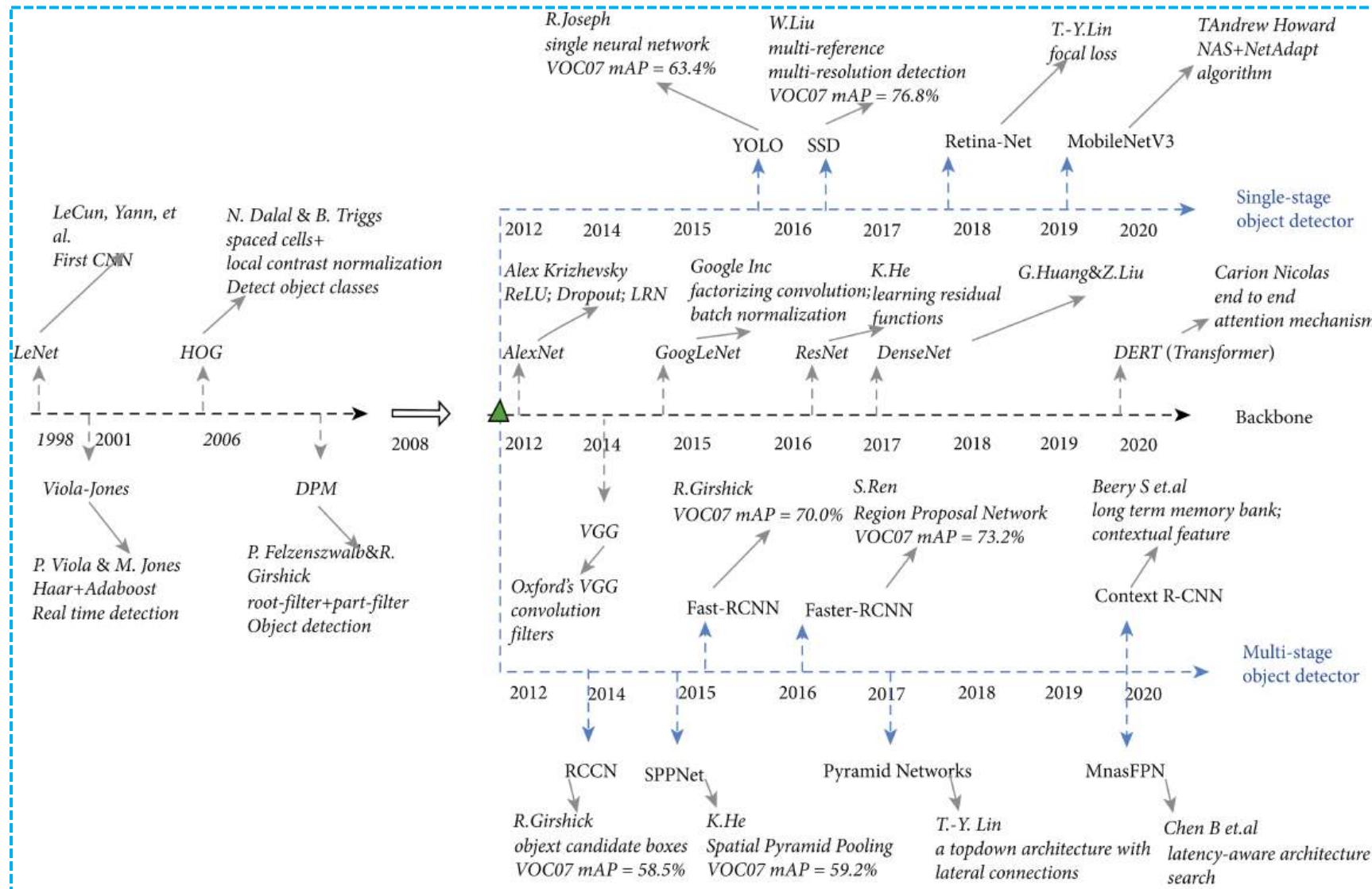
Traditional Object Detectors

CNNs for Image Classification

Image Classifier to Object Detectors with CNN

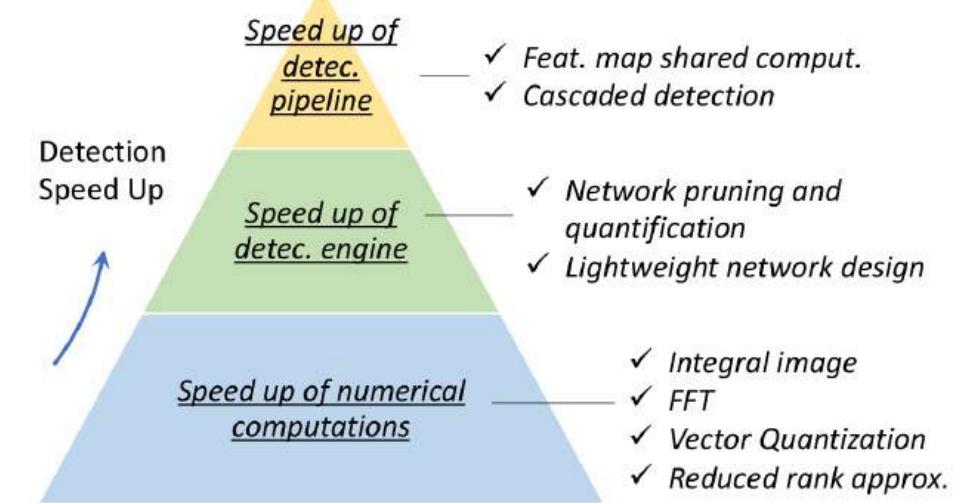
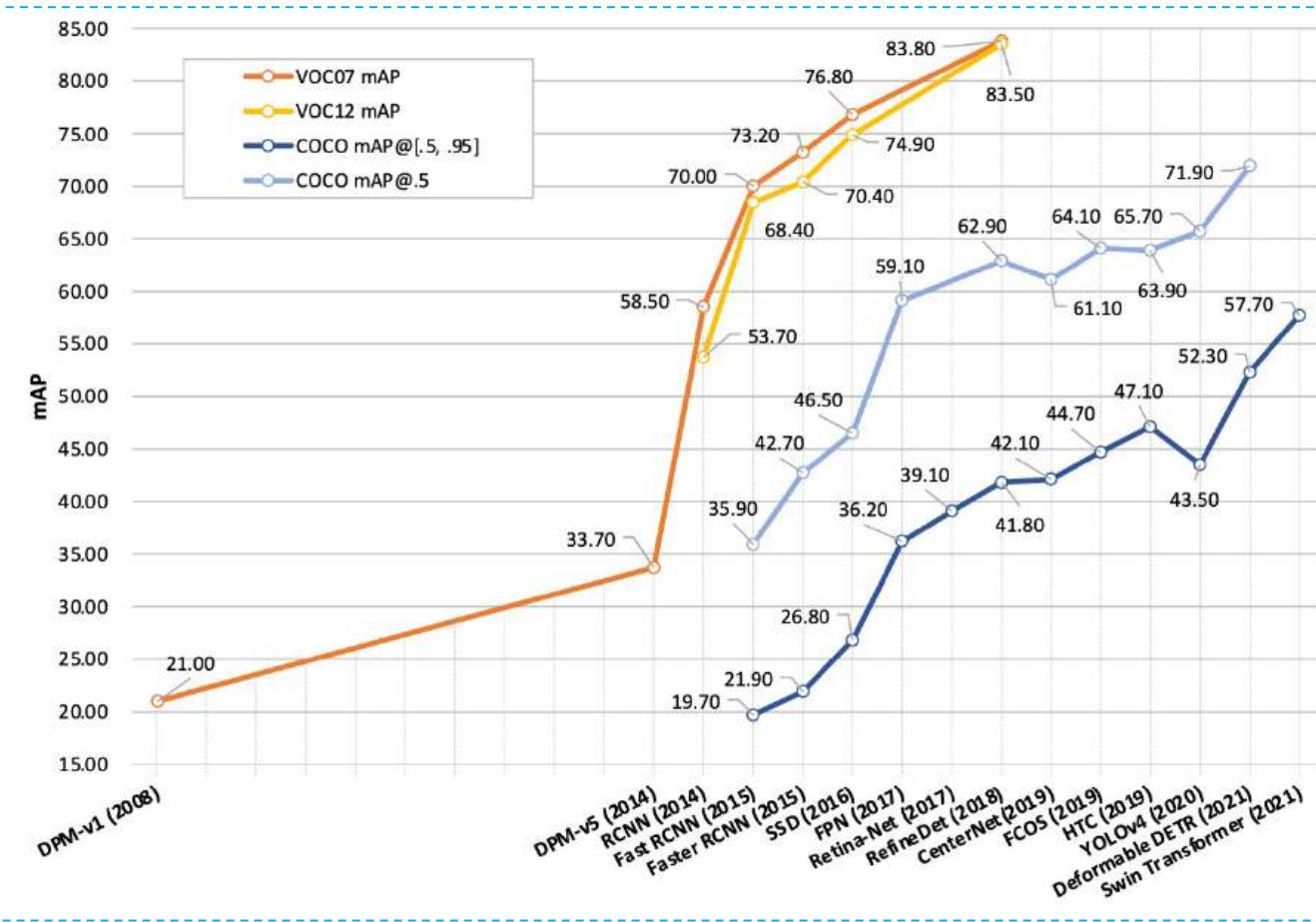
CNN Limitations and Spatial Outputs

Object Detection Milestones

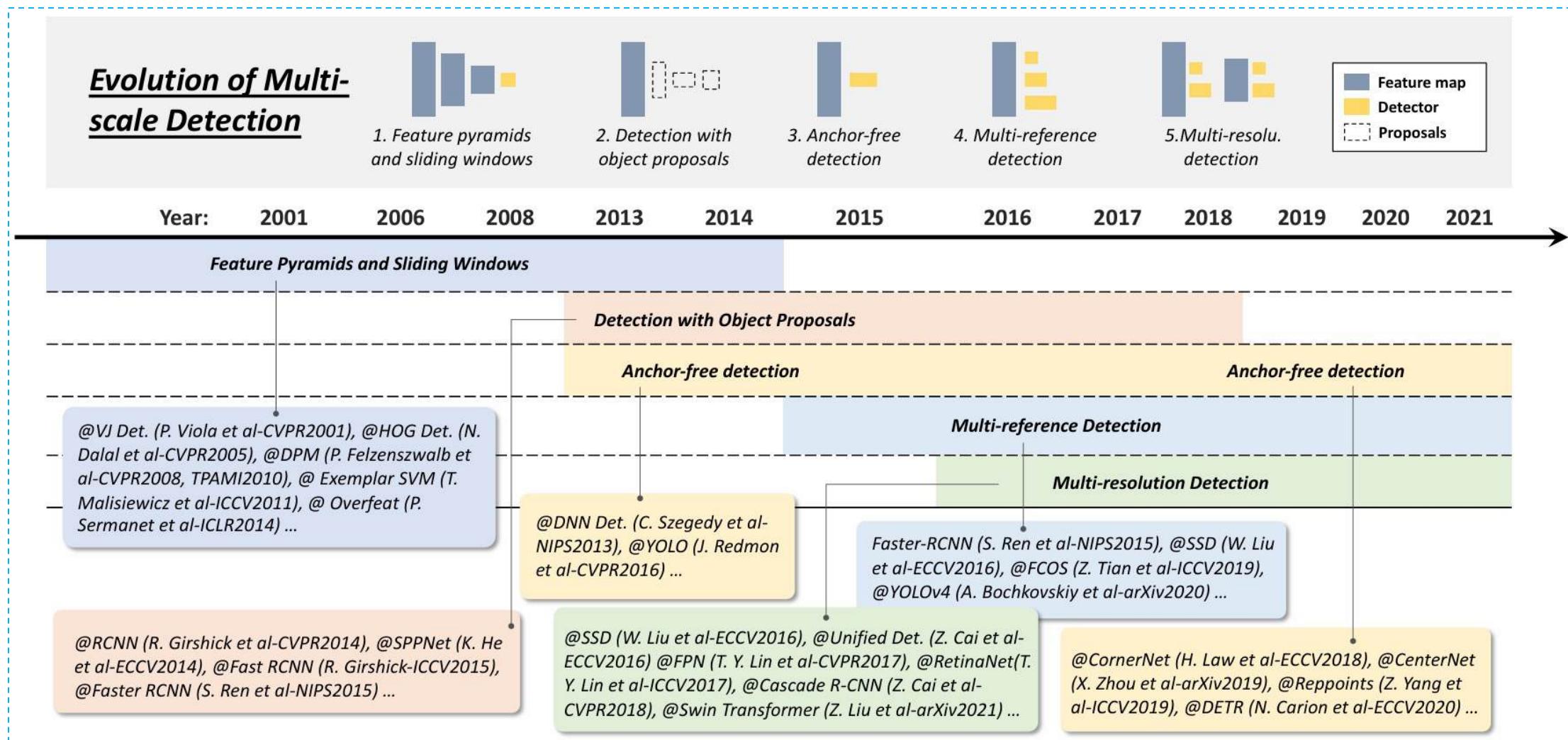


Shaoxi Li et al. "Survey on Deep Learning-Based Marine Object Detection", 2021

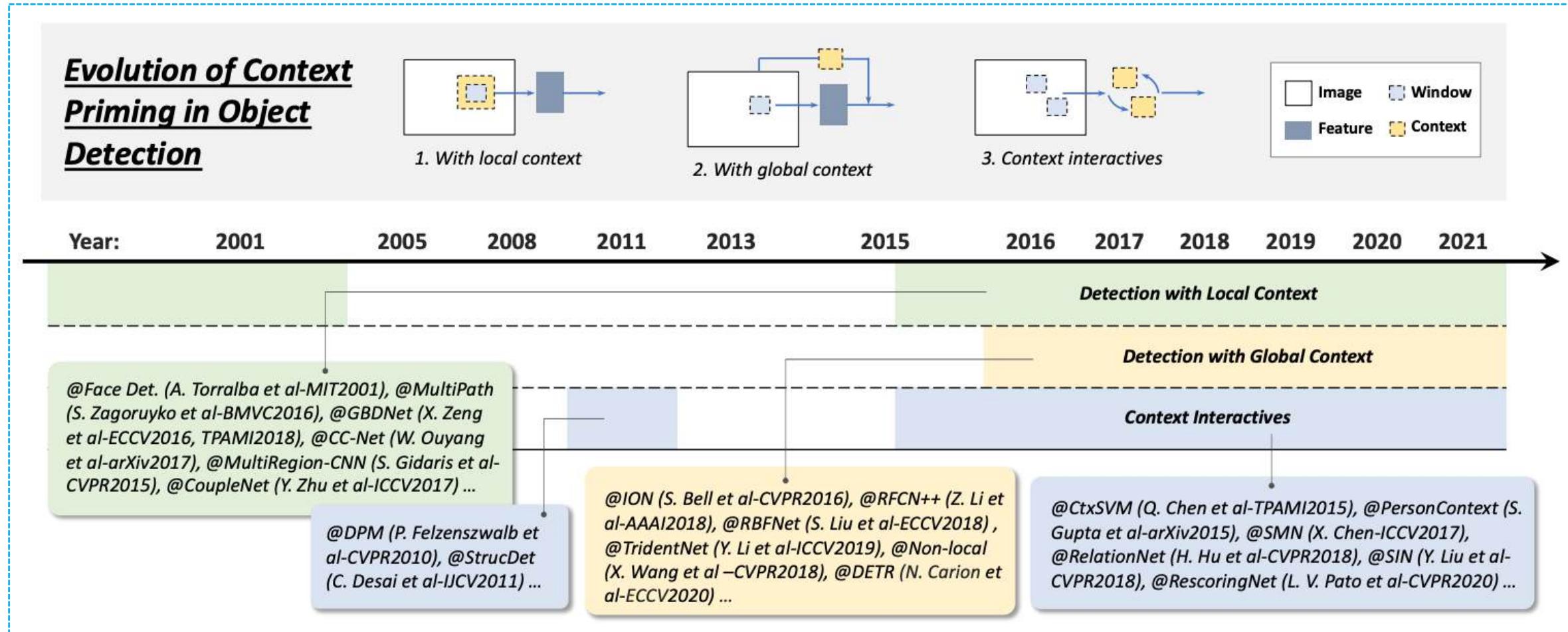
Object Detection Milestones



Object Detection Milestones



Object Detection Milestones



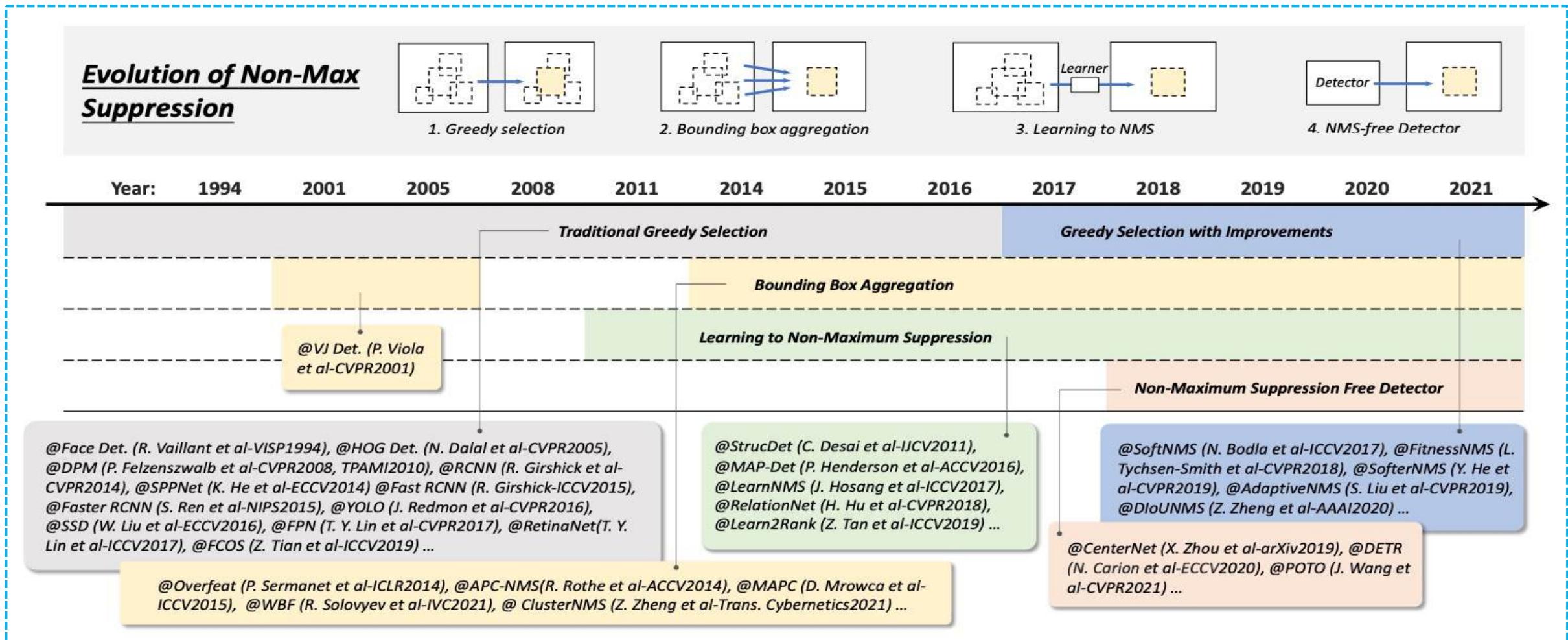
Object Detection Milestones

Evolution of Hard Negative Mining

Year:	1994	2001	2005	2008	2014	2015	2016	2017	2018	2019	2020	2021
Method	Bootstrap				Without Hard Negative Mining			Bootstrap + New Loss Functions				
Remarks	Bootstrap was widely used to deal with the insufficient computing resources of early time				By simply balancing the weights between object and background classes			Focusing on hard examples Computing power is no longer a problem				
@Face Det. (H. A. Rowley et al-CMUTechRep1995), @Haar Det. (C. P. Papageorgiou et al-ICCV1998), @VJ Det. (P. Viola et al-CVPR2001), @HOG Det. (N. Dalal et al-CVPR2005), @DPM (P. Felzenszwalb et al-CVPR2008, TPAMI2010) ...				@RCNN (R. Girshick et al-CVPR2014), @SPPNet (K. He et al-ECCV2014), @Fast RCNN (R. Girshick-ICCV2015), @Faster RCNN (S. Ren et al-NIPS2015), @YOLO (J. Redmon et al-CVPR2016) ...				@SSD (W. Liu et al-ECCV2016), @FasterPed (L. Zhang et al-ECCV2016), @OHEM (A. Shrivastava et al-CVPR2016), @RetinaNet (T. Y. Lin et al-ICCV2017), @RefineDet (Zhang et al-CVPR18), @FCOS (Z. Tian et al-ICCV2019), @YOLOv4 (A. Bochkovskiy et al-arXiv2020) ...				

The training of a detector is essentially an imbalanced learning problem. In the case of sliding window based detectors, the imbalance between backgrounds and objects could be as extreme as 107 :1

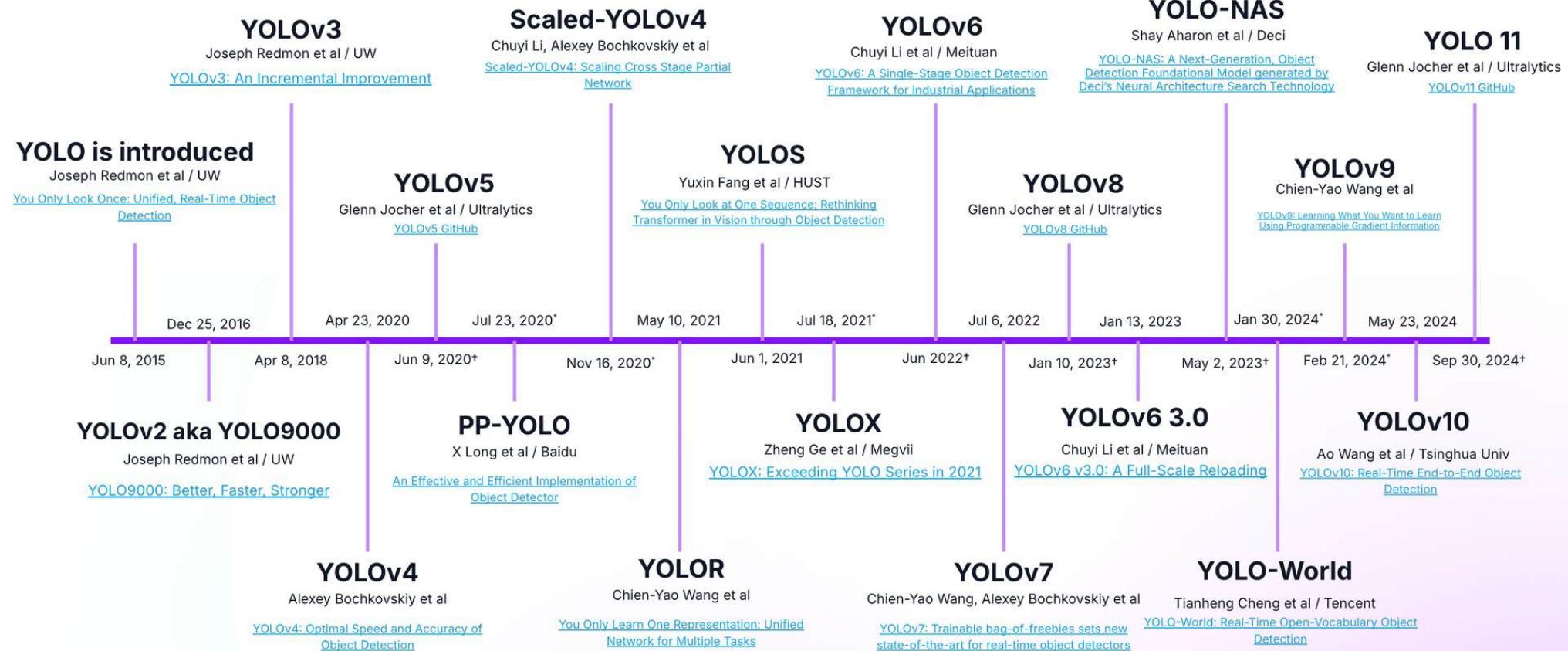
Object Detection Milestones





Object Detection Milestones

YOLO Greatest Hits

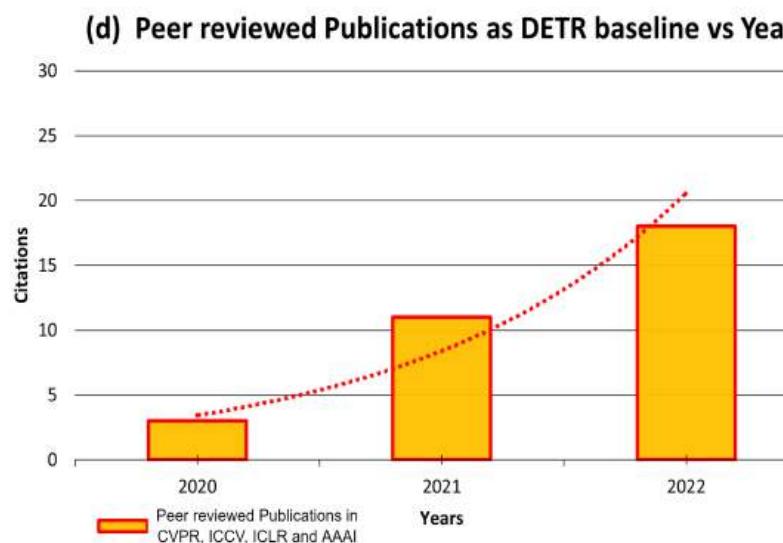
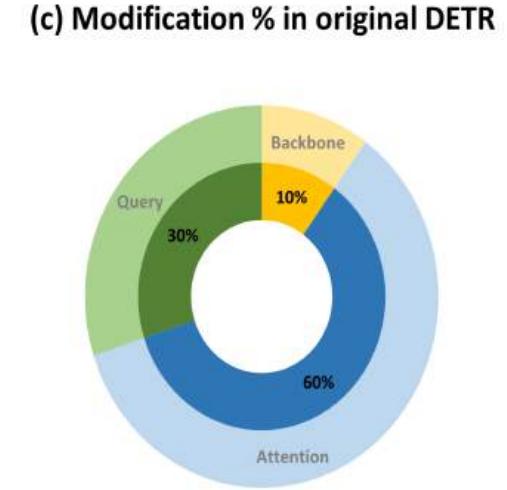
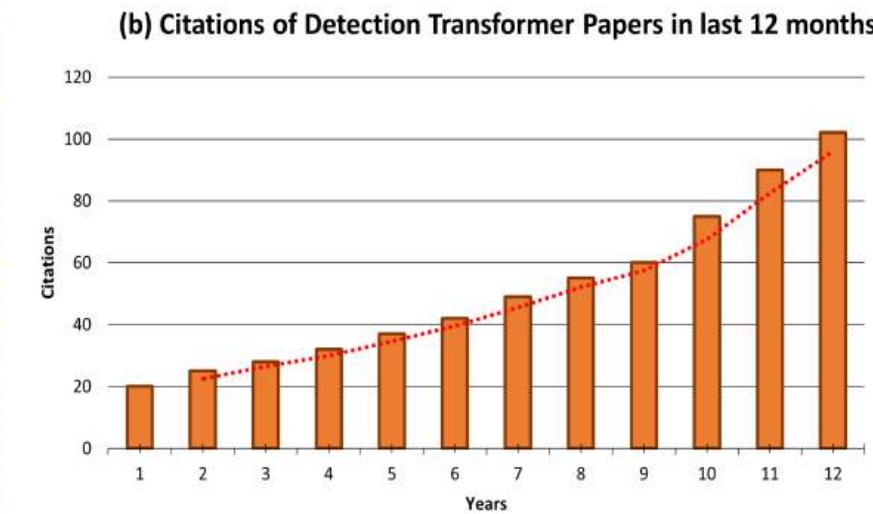
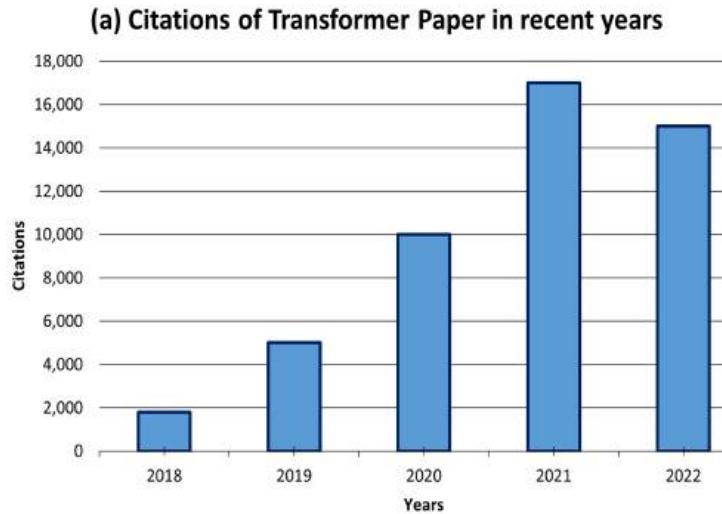


* Denotes paper updated after first publication date

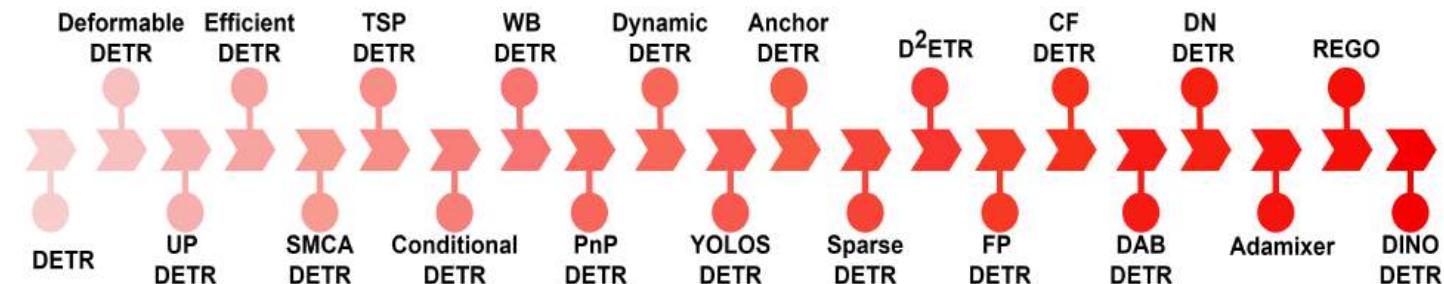
* Denotes repository predates paper publication date



Object Detection: Transformer



(e) Timeline of important developments in DEtection TRansformers (DETR)





Lightweight object detection models

No.	Authors	Detector	Type	Input image	Published in	URL link
1	Li et al. (2017)	YOLOv3-DarkNet53	Anchor-based	320*320	arXiv 2018	https://github.com/westerndigitalcorporation/YOLOv3-in-PyTorch
2	Howard et al. (2017)	MobileNet-SSD	Anchor-based	300*300	arXiv 2017	https://github.com/chuangqi305/MobileNet-SSD
3	Sandler et al. (2018)	MobileNetv2-SSDLite	Anchor-based	320*320	CVPR 2018	https://github.com/tranleanh/mobilenets-ssd-pytorch
4	Li et al. (2018)	Tiny-DSOD	Anchor-based	300*300	arXiv 2018	https://github.com/lyxok1/Tiny-DSOD
5	Wang et al. (2018)	Pelee	Anchor-based	304*304	NeurIPS 2018	https://github.com/Robert-JunWang/Pelee
6	Qin et al. (2019), Huang et al. (2018)	YOLO-LITE	Anchor-based	416*416	ICBD 2018	https://github.com/reu2018DL/YOLO-LITE
7	Qin et al. (2019)	ThunderNet	Anchor-based	320*320	ICCV 2019	https://github.com/DayBreak-u/Thundernet_Pytorch
8	Tan et al. (2019)	MnasNet-A1 + SSDLite	Anchor-based	320*320	CVPR 2019	https://github.com/tensorflow/tpu/tree/master/models/official/mnasnet
9	Tang et al. (2020a, 2020b)	LightDet	Anchor-based	320*320	ICASSP 2020	Not available
10	Yi et al. (2019)	YOLOV3-Tiny	Anchor-based	416*416	ICACCS 2020	https://github.com/pjreddie/darknet/blob/master/cfg/yolov3-tiny.cfg
11	Long et al. (2020a, 2020b)	PP-YOLO	Anchor-based	608*608	CVPR 2020	https://github.com/PaddlePaddle/PaddleDetection
12	Long et al. (2020a, 2020b)	YOLOv4-Tiny	Anchor-based	416*416	arXiv 2020	https://github.com/truong2710-cyber/Mask-Detection-YOLOv4-tiny-Kaggle-Dataset
13	Tan et al. (2020)	EfficientDet	Anchor-based	512*512	CVPR 2020	https://github.com/xuannianz/EfficientDet
14	Huang et al. (2021)	PP-YOLOv2	Anchor-based	640*640	arXiv 2021	https://github.com/PaddlePaddle/PaddleDetection
15	Ge et al. (2021)	YOLOX-Nano	Anchor-free	416*416	ICSP 2022	https://github.com/Megvii-BaseDetection/YOLOX
16	Ge et al. (2021)	YOLOX-Tiny	Anchor-free	416*416	IJCINI 2022	https://github.com/TexasInstruments/edgeai-yolox/blob/main/exp/defalut/yolox_tiny.py
17	Cai et al. (2021)	YOLOBile	Anchor-based	320*320	AAAI 2021	https://github.com/nightsnack/YOLOBile
18	Wang et al. (2021)	Scaled YOLOv4	Anchor-based	608*608	CVPR 2021	https://github.com/WongKinYiu/ScaledYOLOv4
19	Wang et al. (2021)	Trident YOLO	Anchor-based	416*416	Wiley IET	Not available
20	Li et al. (2021a, 2021b, 2021c)	NanoDet	Anchor-free	320*320	Journals of Radar	https://github.com/RangiLyu/nanodet
21	Yu et al. (2021)	PP-PicoDet	Anchor-free	416*416	arXiv 2021	https://github.com/PaddlePaddle/PaddleDetection
22	Ding et al. (2022)	Slim YOLOv4	Anchor-free	416*416	JRIP	Not available
23	Liu et al. (2022a, 2022b)	Mini YOLO	Anchor-free	320*320	Wiley Journal	Not available
24	Xu et al. (2022)	PP-YOLOE-S	Anchor-free	640*640	arXiv 2022	https://github.com/PaddlePaddle/PaddleDetection
25	Wang et al. (2022a, 2022b, 2022c, 2022d)	YOLOv7-X	Anchor-free	640*640	arXiv 2022	https://github.com/WongKinYiu/yolov7
26	Li et al. (2022a, 2022b)	L-DETR	Anchor-free	800*1333	IEEE Access	https://github.com/wangjian123799/L-DETR.git



Lightweight object detection models

No.	Detector	Backbone	Loss function	AP	Proposal
1	YOLOv3-DarkNet53	DarkNet53	Logistic regression	38.1	Network structure makes greater use of the GPU, making it faster to evaluate than Darknet-19
2	MobileNet-SSD	MobileNet	Smooth L1 loss	19.3	Lightweight deep network is constructed using depth-wise separable convolutions
3	MobileNetv2-SSDLite	MobileNetv2	Smooth L1 loss	22.1	With fewer parameters and less computational complexity, gets competitive accuracy
4	Tiny-DSOD	DDB-Net	Smooth L1 loss	23.2	For resource-constrained uses based on DDB and D-FPN blocks
5	Pelee	PeleeNet	Smooth L1 loss	22.4	Variant of DenseNet, built with conventional convolution
6	YOLO-LITE	Darknet-53	L1 loss	12.2	A real-time detection model developed to run on portable devices lacking a GPU
7	ThunderNet	SNet535	Sigmoid	28.1	Embedded context enhancement and spatial attention module
8	MnasNet-A1 + SSDLite	MnasNet-A1	Smooth L1 loss	23.0	Directly measures real-world inference latency by executing the model on edge devices
9	LightDet	Modified ShuffleV2	Smooth L1 loss	24.0	Introduce an efficient feature-preserving and refinement module
10	YOLOv3-Tiny	Reduced Darknet-53	Logistic regression	16.6	Lightweight model of YOLOv3, which takes reduced training time
11	PP-YOLO	MobileNetV3	IoU aware loss	45.9	Balanced efficiency, directly applied in real application scenarios
12	YOLOv4-Tiny	CSP-ResNet	Clou loss	28.7	Reduced parameters, makes it suitable for edge devices
13	EfficientDet	EfficientNet	Focal loss	34.6	Proposed a weighted bi-directional FPN for feature fusion
14	PP-YOLOv2	ResNet101	IoU aware loss	49.5	Increasing the input size and follow the design of PAN to aggregate the top-down information
15	YOLOX-Nano	DarkNet53	GloU loss	25.8	Dynamic label assignment strategy SimOTA
16	YOLOX-Tiny	CBAM	GloU loss	32.8	Fuses convolutional attention and mixup data enhancement strategy
17	YOLObile	CSP-DarkNet53	GloU loss	31.6	Offers mobile acceleration and block-punched pruning with a mobile GPU-CPU collaborative strategy
18	Scaled YOLOv4	CSPNet-15	Clou loss	28.7	Propose a network scaling approach that modifies the width, and resolution of network
19	TridentYOLO	CSP-RFBs	Focal loss	40.3	Propose a trident feature pyramid network
20	NanoDet	ShuffleNetV2	Focal loss	20.6	Based on visual saliency and perform feature learning on samples added with saliency maps
21	PP-PicoDet	Enhanced ShuffleNet	GloU Loss	30.6	Improved detection One-Shot NAS pipeline
22	Slim YOLOv4	MobileNetV2	Clou loss	29.2	Efficient network computing methods for convolution
23	MiniYOLO	DSLightNet	Clou	21.4	Adopted depthwise separable convolution
24	PP-YOLOE-S	ResNet50-vd	IoU aware loss	43.1	Scalable backbone-neck architecture, and refined loss function
25	YOLOv7-X	RepCSPResNet	Assistant loss	53.1	Propose the trainable bag-of-freebies method to enhance accuracy
26	L-DETR	PP-LCNet	H-sigmoid function	-	Embedded group normalization



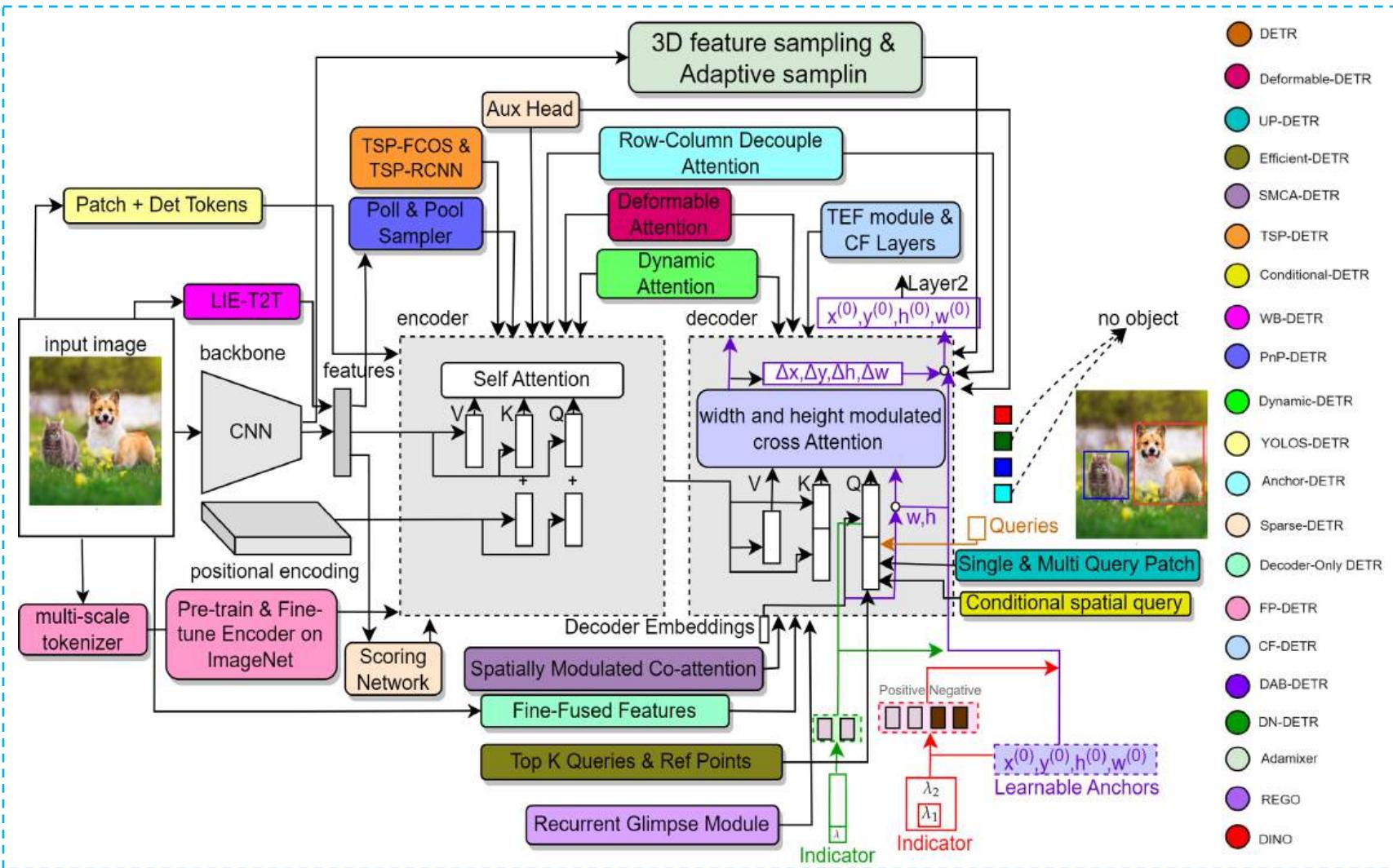
Lightweight object detection models

No.	Light-weight object detector	Backbone	FLOPs	Inference time (ms)	FPS	Parameters (M)	Real-time applications
1	YOLOv3-DarkNet53	DarkNet53	1453B	22	171	–	✓
2	MobileNet-SSD	MobileNet	1.2G	–	59.3	4.31	*
3	MobileNetv2-SSDLite	MobileNetv2	0.8G	–	–	3.38	*
4	Tiny-DSOD	DDB-Net	1.12G	–	105	0.95	*
5	Pelee	PeleeNet	1.21B	–	205	5.98	*
6	YOLO-LITE	Darknet-53	0.48G	–	21	–	*
7	ThunderNet	SNet535	0.47	–	248	–	✓
8	MnasNet-A1 + SSDLite	MnasNet-A1	0.8B	203	–	4.9	*
9	LightDet	Modified ShuffleV2	0.50G	–	250	–	✓
10	YOLOv3-Tiny	Reduced Darknet-53	5.56 B	4.5	368	8.86	*
11	PP-YOLO	MobileNetV3	1.02G	10.48	73	1.08	✓
12	YOLOv4-Tiny	CSP-ResNet	–	–	371	6.06	✓
13	EfficientDet	EfficientNet	2.5B	10.20	98	3.9	✓
14	PP-YOLOv2	ResNet101	0.115G	14.50	68.9	1.08	✓
15	YOLOX-Nano	DarkNet53	1.08G	19.23	90.1	0.91	✓
16	YOLOX-Tiny	CBAM	6.48G	32.77	–	5.06	✓
17	YOLOBile	CSP-DarkNet53	3.95G	–	17	4.59	✓
18	Scaled YOLOv4	CSPNet-15	6.3B	–	62	53	✓
19	TridentYOLO	CSP-RFBs	5.19B	–	29.3	–	✓
20	NanoDet	ShuffleNetV2	1.2G	13.35	–	0.95	*
21	PP-PicoDet	Enhanced ShuffleNet	0.73G	8.13	–	0.99	✓
22	Slim YOLOv4	MobileNetV2	–	–	60.19	–	✓
23	MiniYOLO	DSLightNet	–	–	–	2.06	*
24	PP-YOLOE-S	ResNet50-vd	110.7G	12.8	208.3	52.20	✓
25	YOLOv7-X	RepCSPResNet	189.9G	–	114	71.3	✓
26	L-DETR	PP-LCNet	–	–	–	–	✓

Mittal, P. A comprehensive survey of deep learning-based lightweight object detection models for edge devices. *Artif Intell Rev* 57, 242 (2024).

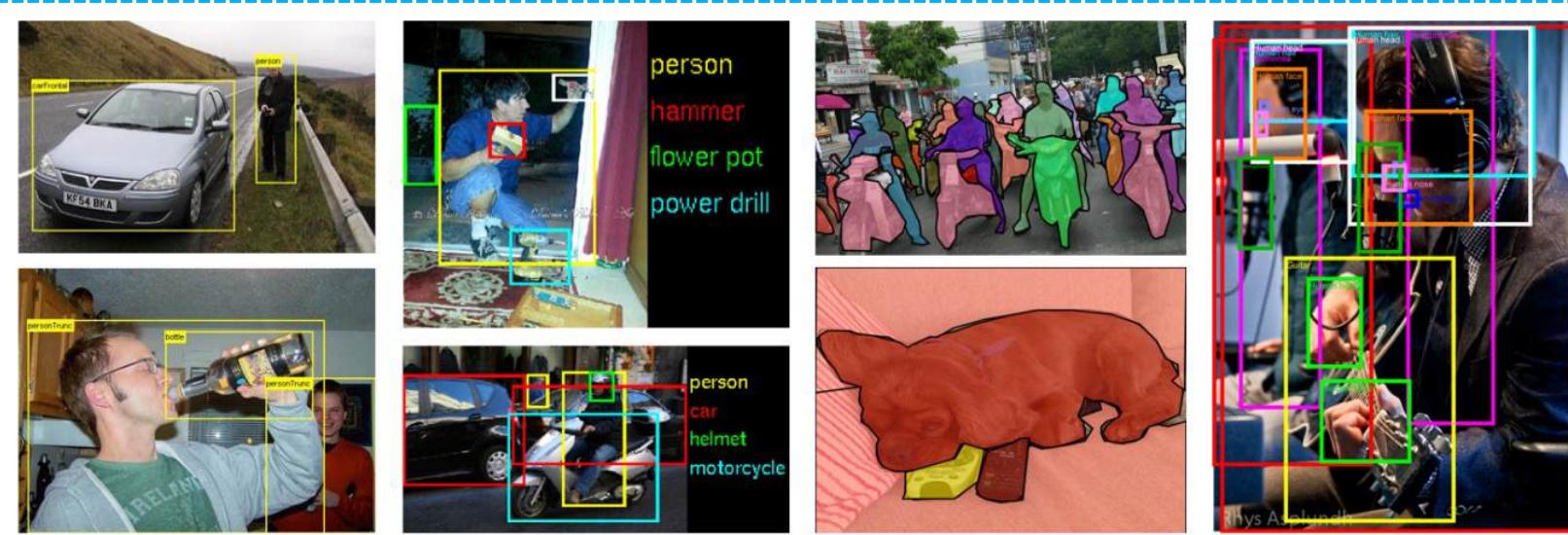
<https://doi.org/10.1007/s10462-024-10877-1>

Object Detection: Transformer



An overview of the Detection Transformer (DETR) and its modifications proposed by recent methods to improve performance and training convergence.

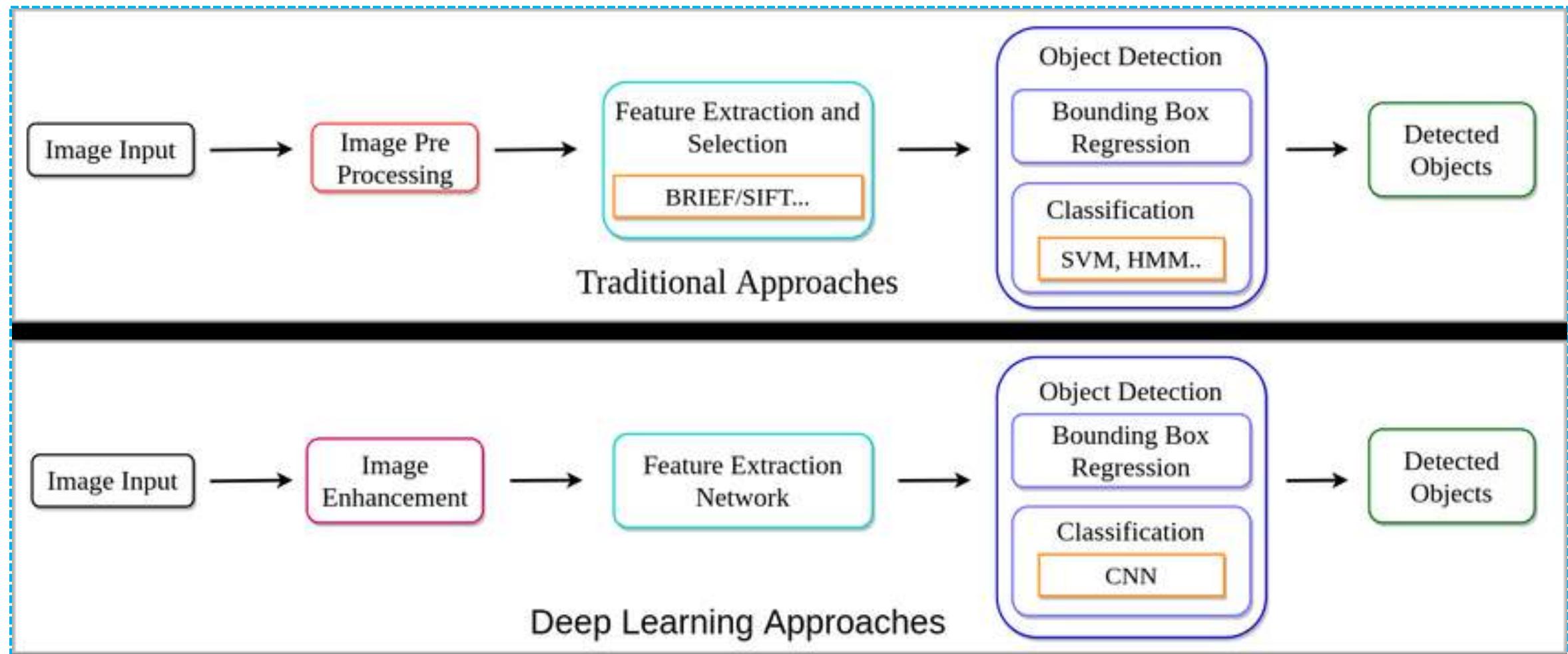
Well-known Object Detection Datasets



Z. Zou, K. Chen, Z. Shi, Y. Guo and J. Ye, "Object Detection in 20 Years: A Survey," in Proceedings of the IEEE, vol. 111, no. 3, pp. 257-276, March 2023, doi: 10.1109/JPROC.2023.3238524.

Dataset	train		validation		trainval		test	
	images	objects	images	objects	images	objects	images	objects
VOC-2007	2,501	6,301	2,510	6,307	5,011	12,608	4,952	14,976
VOC-2012	5,717	13,609	5,823	13,841	11,540	27,450	10,991	-
ILSVRC-2014	456,567	478,807	20,121	55,502	476,688	534,309	40,152	-
ILSVRC-2017	456,567	478,807	20,121	55,502	476,688	534,309	65,500	-
MS-COCO-2015	82,783	604,907	40,504	291,875	123,287	896,782	81,434	-
MS-COCO-2017	118,287	860,001	5,000	36,781	123,287	896,782	40,670	-
Objects365-2019	600,000	9,623,000	38,000	479,000	638,000	10,102,000	100,000	1,700,00
OID-2020	1,743,042	14,610,229	41,620	303,980	1,784,662	14,914,209	125,436	937,327

Traditional Vs. Deep Learning





Object Detection Milestones

Milestones: Traditional Detectors

Viola Jones Detectors, SVM + HOG & DPM

Milestones: CNN based Two-stage Detectors

RCNN, SPPNet, Fast RCNN, Faster RCNN,..

Milestones: CNN based One-stage Detectors

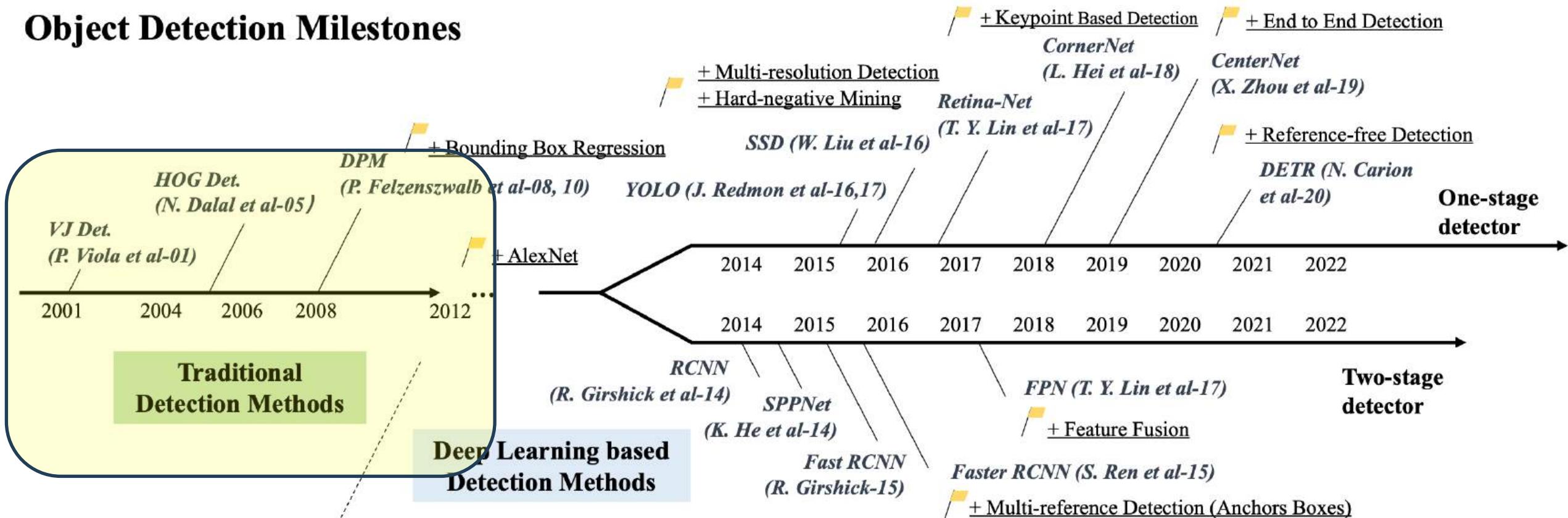
YOLO, SSD, RetinaNet, CornerNet, Center Net,..

Milestones: Transformer for OD

DETR, D-DETR, DINO,...

Object Detection Milestones

Object Detection Milestones



Outline

Introduction to Object Detection

Applications of Object Detection

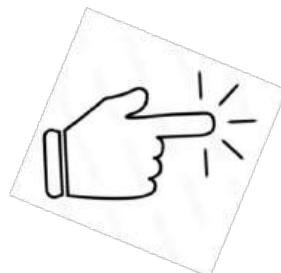
Object Detection Milestones

Traditional Object Detectors

CNNs for Image Classification

Image Classifier to Object Detectors with CNN

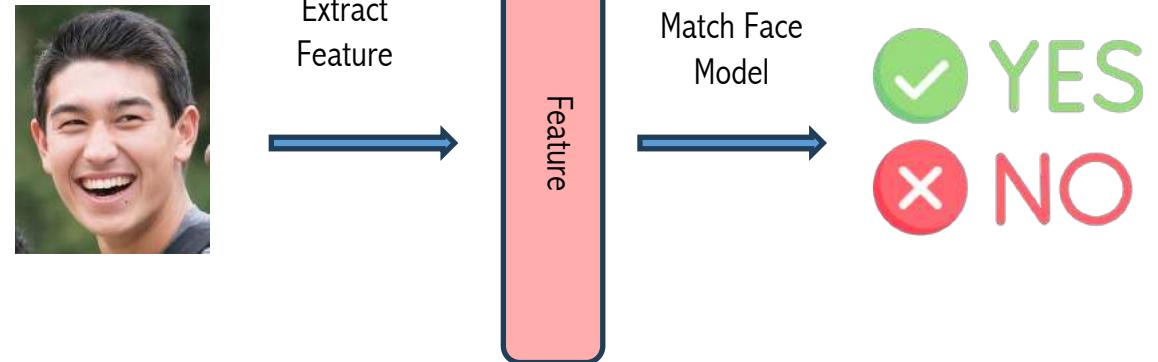
CNN Limitations and Spatial Outputs



Traditional Detectors



Slides windows of different sizes across image.
At each location match window to face model



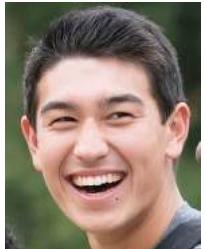
Features:

- How to extract feature?
- Which features represent face well?

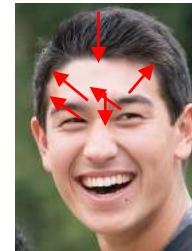
Classifier:

- How to build a model and classify features as face or not?

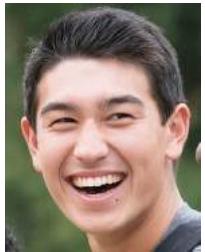
What are good features?



Interest Point (Edge, Corners, SIFT,..)



- Extreme Fast to Compute
- Millions of windows in an image



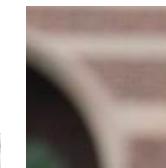
Facial Components (Templates)?



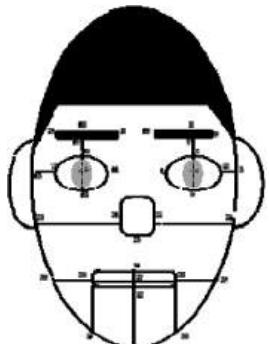
The key aspect in face recognition is detecting relevant features in human face like eyes, eyebrows, nose, lips. So how do we detect these features in real time/in an image ?



Discriminative Face / None Face

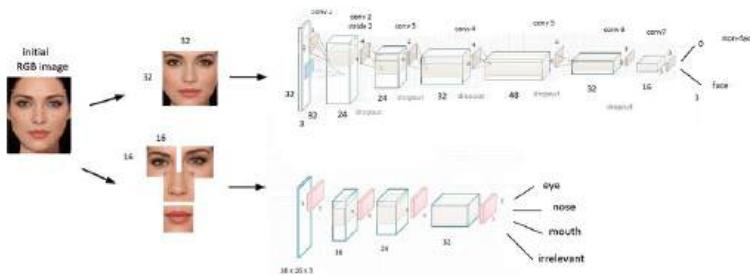


Methods for Face Detection



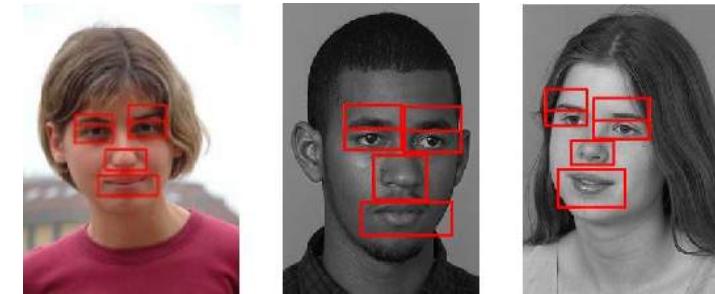
Knowledge Based

- Rule based (Ex: X must have eyes, x must have a nose)
- Too many rules and variables with this method



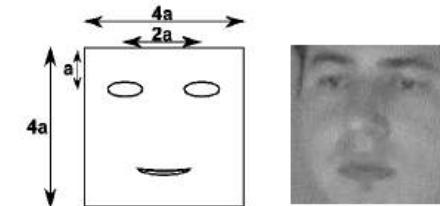
Appearance Based

Learn the characteristics of a face. Example: CNN's
Accuracy depends on training data (which can be scarce)



Feature Based

Locate and extract structural features in the face
Find a differential between facial and non facial regions in an image

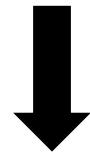


Template

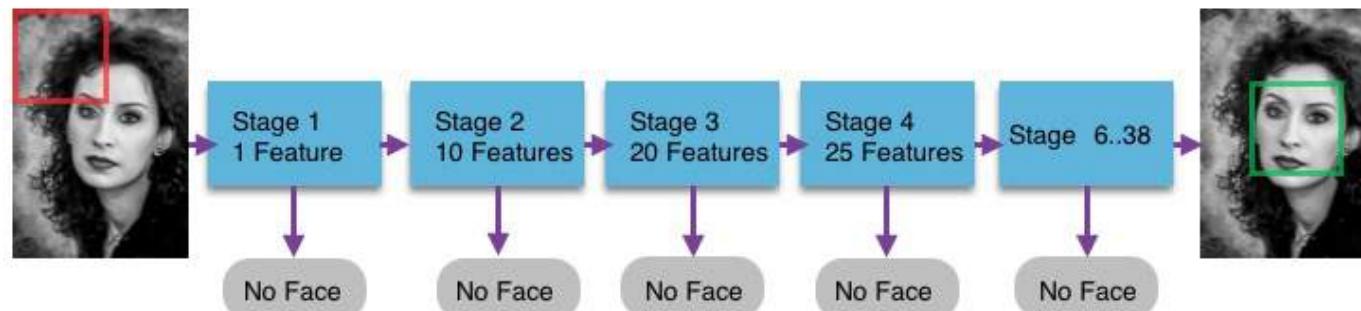
Using predefined templates for edge detection. Quick and easy
A trade off for speed over accuracy

The key aspect in face recognition is detecting relevant features in human face like eyes, eyebrows, nose, lips. So how do we detect these features in real time/in an image ?

The answer is **Haar Wavelets or Haar Features**



The algorithm used is called as Viola-Jones Algorithm



Viola Jones Detectors : Haar Feature

Although there are thousands of possible feature shapes that can be created, the two most common are **Edge Features** and **Line Features**.

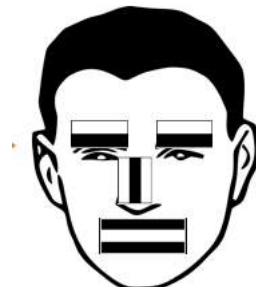
Detect part of a face, **eyebrow**, naturally the shade of the pixels of on an eyebrow in an image will be darker and abruptly gets lighter (skin).



Edge Features

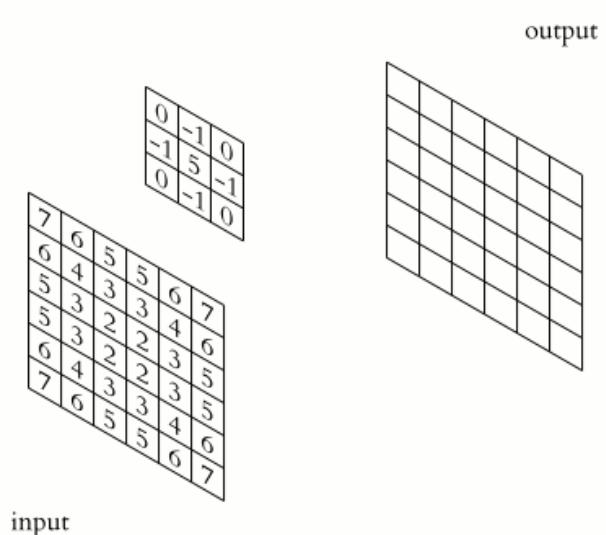
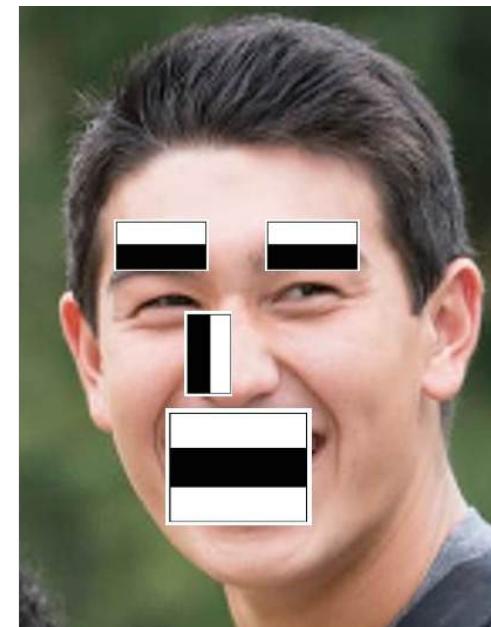
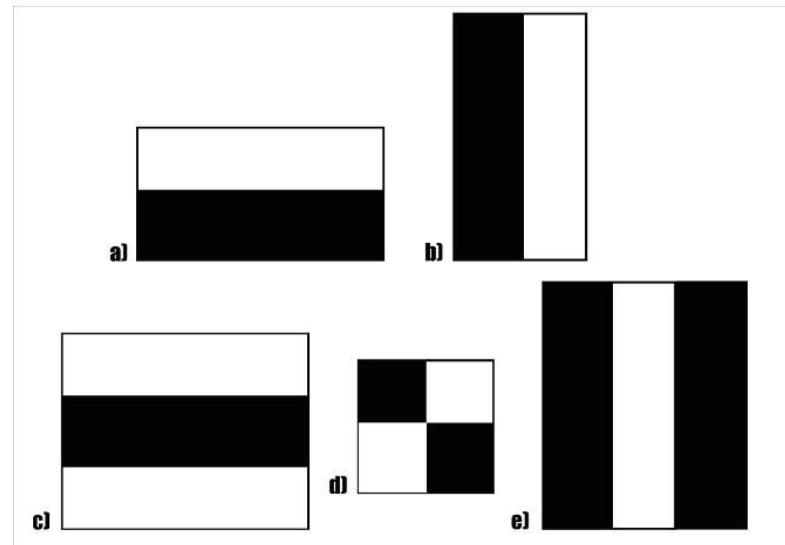


Detect a **mouth**: naturally the shape of the lips region on your face go from light to dark to light again. For this, Line features prove to be the best.



Viola Jones Detectors : Haar Feature

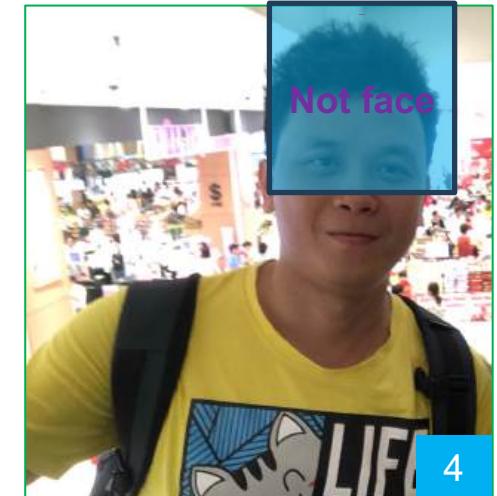
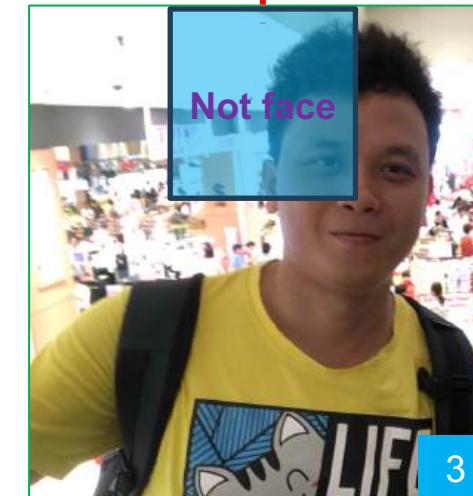
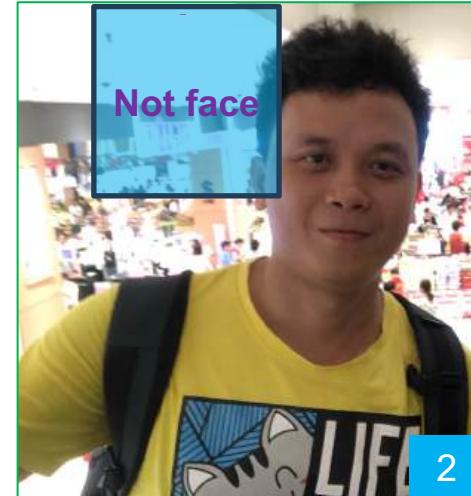
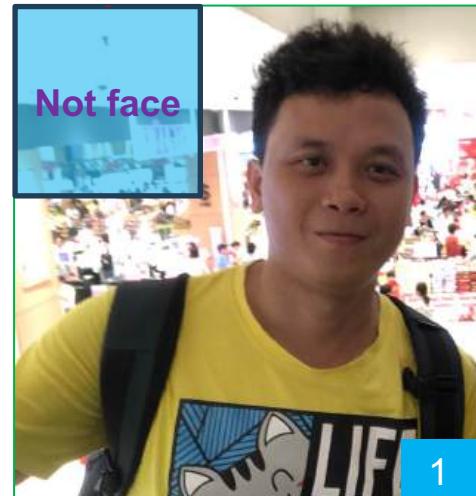
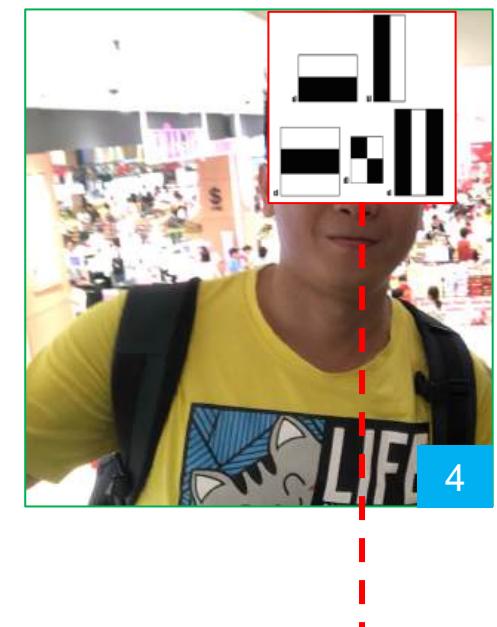
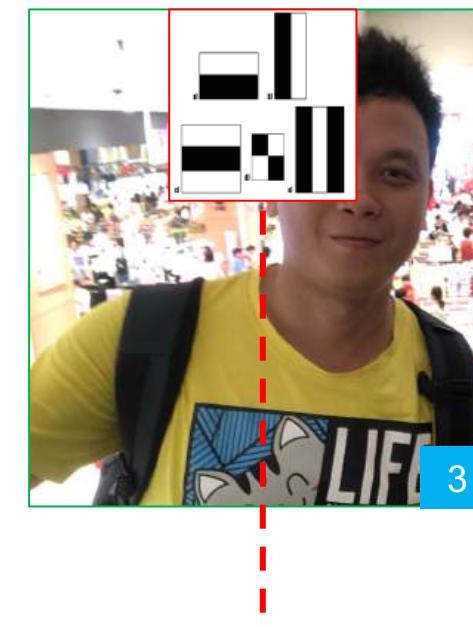
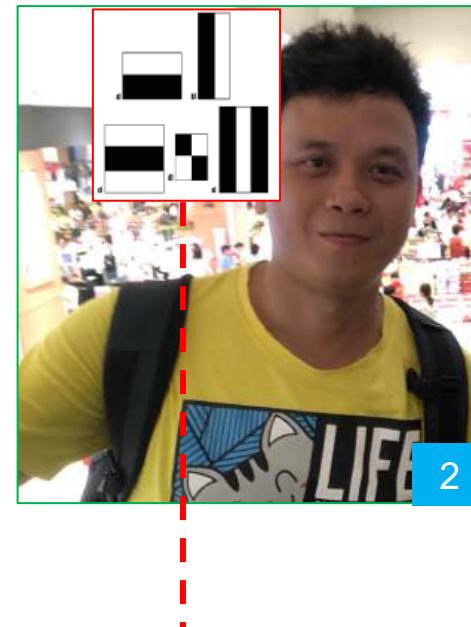
Running on a 700MHz Pentium III CPU, the detector was tens or even hundreds of times faster than other algorithms in its time under comparable detection accuracy



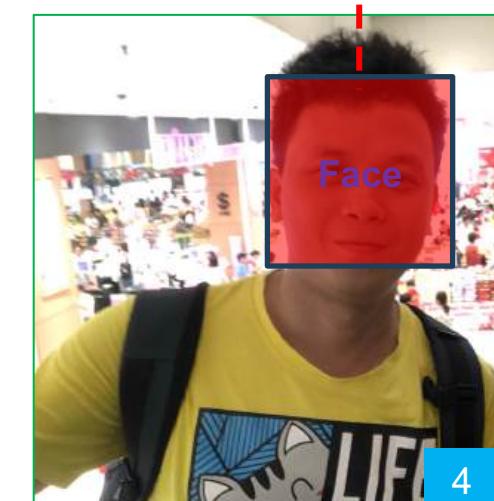
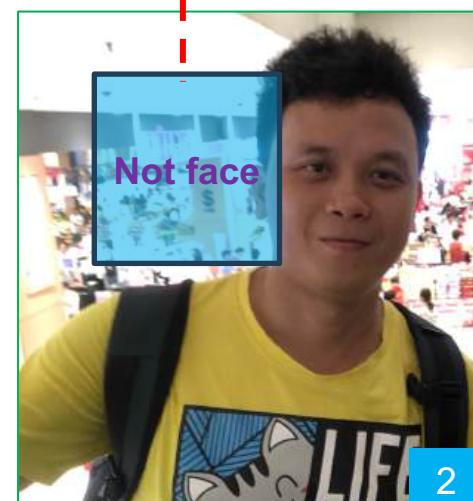
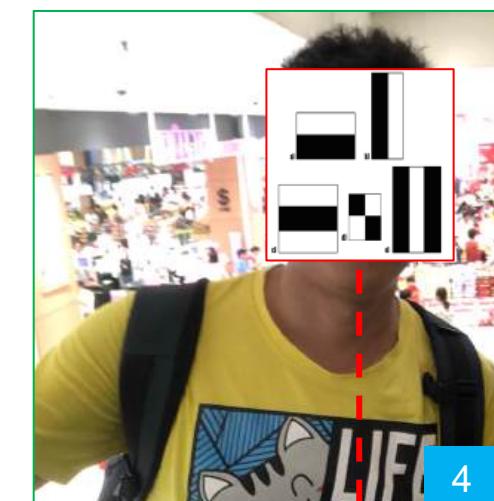
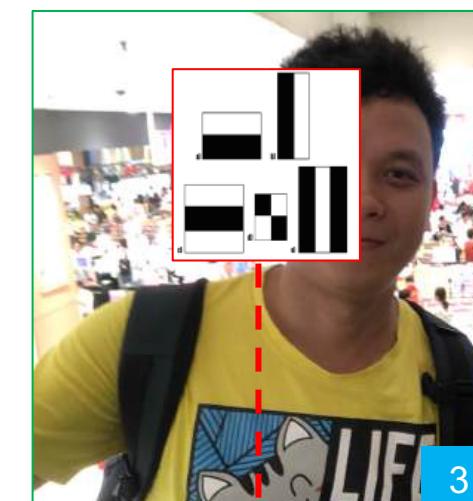
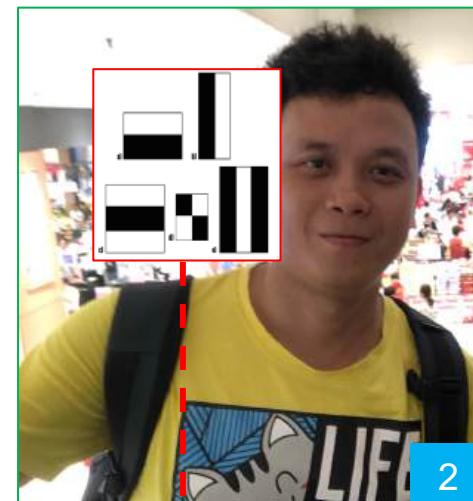
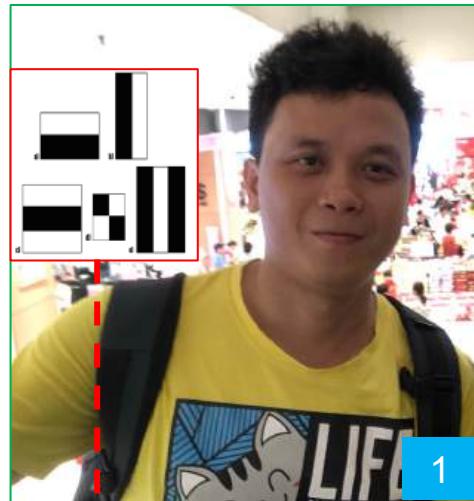
Viola-Jones uses 24*24 as base window size and calculates the above features all over the image shifting by 1 PX (Template + Feature). There are over 160,000 possible feature combinations that can fit into a 24x24 pixel image, and over 250,000 for a 28x28 image

P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.

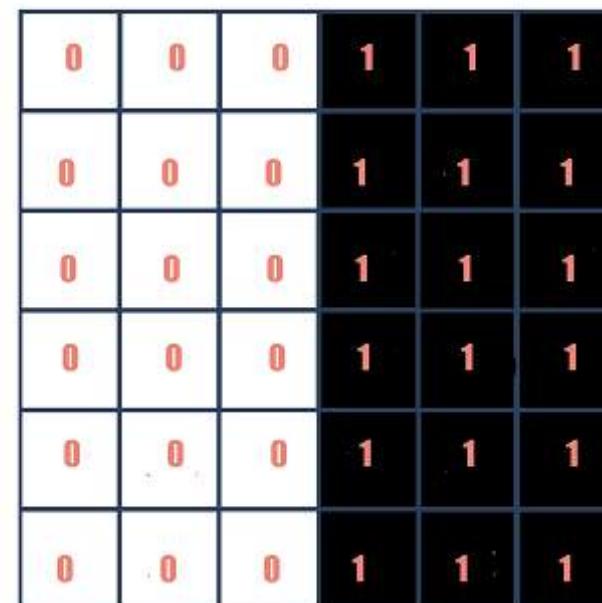
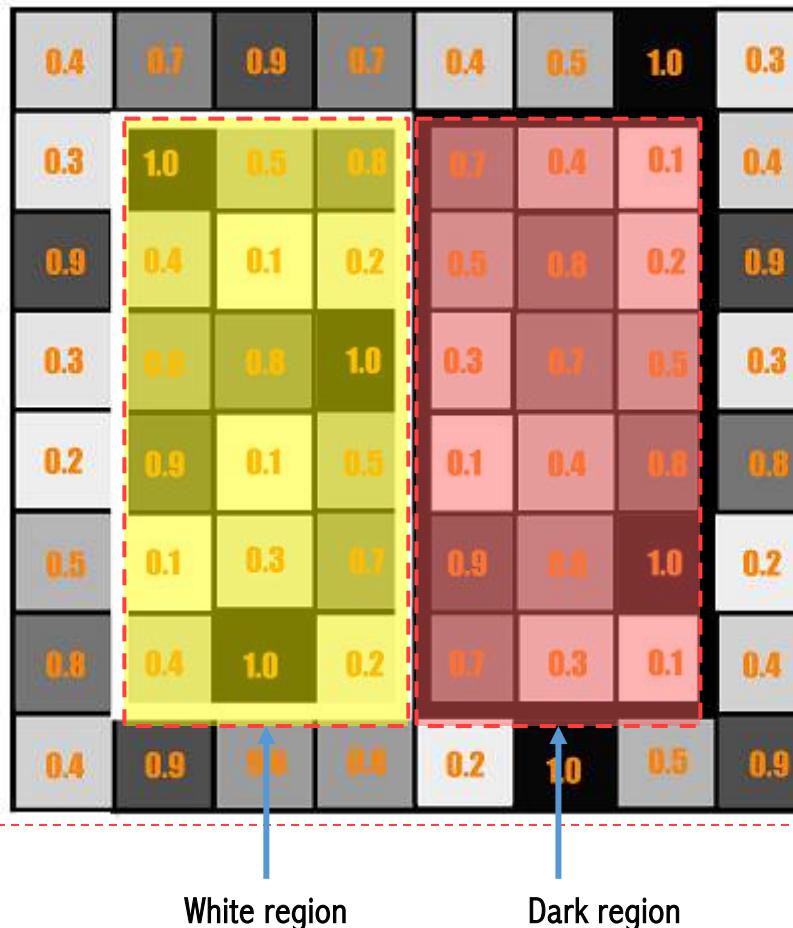
Viola Jones Detectors : Haar Feature



Viola Jones Detectors : Haar Feature



Viola Jones Detectors : Haar Feature



=

$$\frac{\text{SUM OF THE DARK PIXELS}}{\text{NUMBER OF DARK PIXELS}} - \frac{\text{SUM OF THE LIGHT PIXELS}}{\text{NUMBER OF THE LIGHT PIXELS}}$$

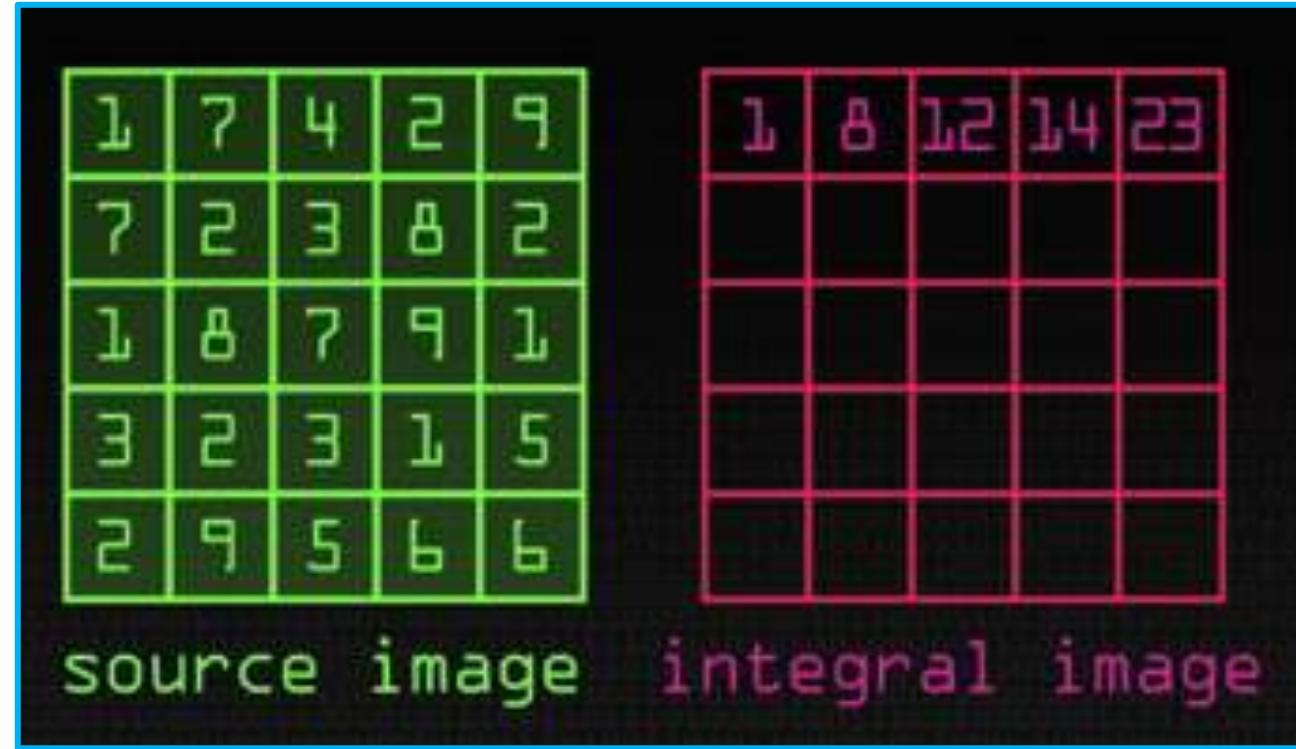
$$(0.7 + 0.4 + 0.1 + 0.5 + 0.8 + 0.2 + 0.3 + 0.7 + 0.5 + 0.1 + 0.4 + 0.8 + 0.9 + 0.6 + 1.0 + 0.7 + 0.3 + 0.1)/18$$

$$(1.0 + 0.5 + 0.8 + 0.4 + 0.1 + 0.2 + 0.6 + 0.8 + 1.0 + 0.9 + 0.1 + 0.5 + 0.1 + 0.3 + 0.7 + 0.4 + 1.0 + 0.2)/18$$

$$0.51 - 0.53 = -0.02$$

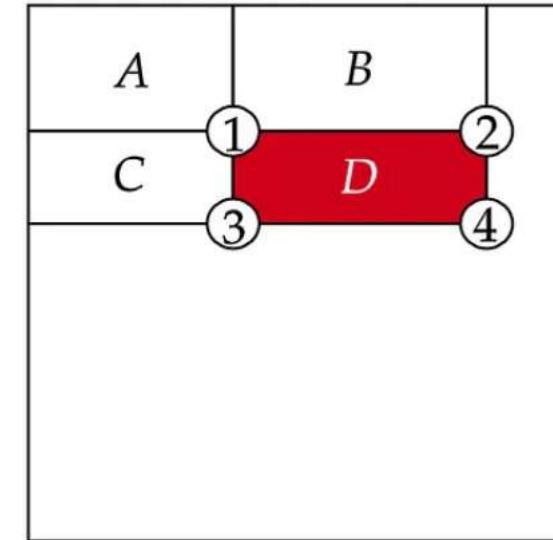
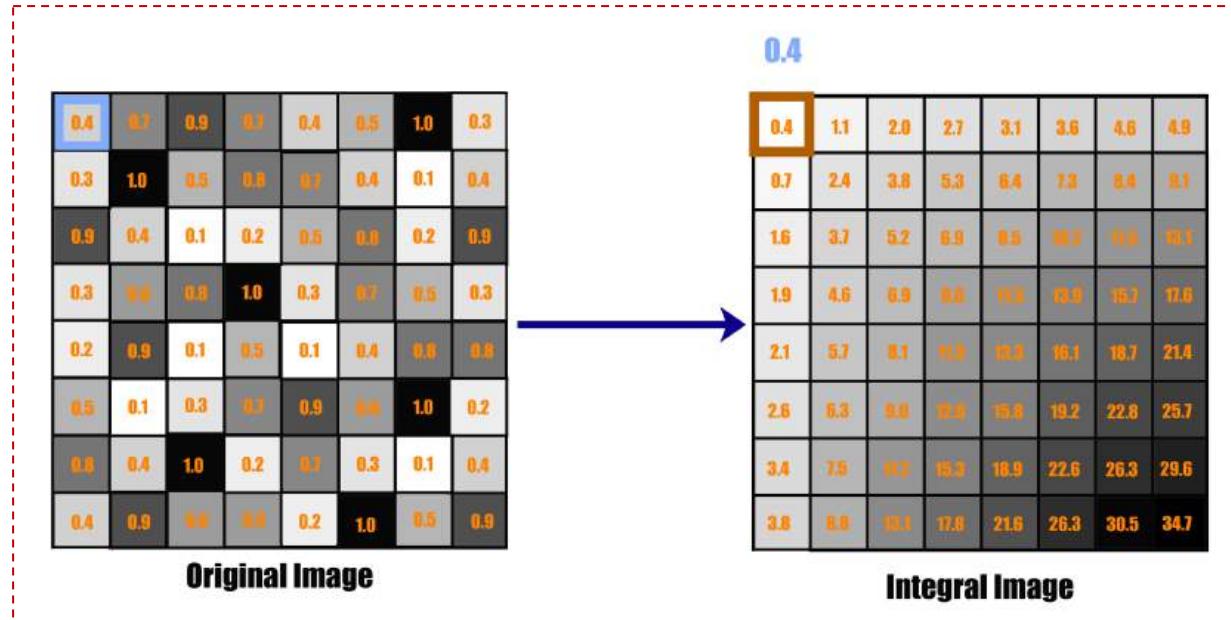
Summing up pixel values for all feature types in all images in your dataset can be very **computationally expensive**, especially depending on the resolution of your images.

Viola Jones Detectors : Integral Image

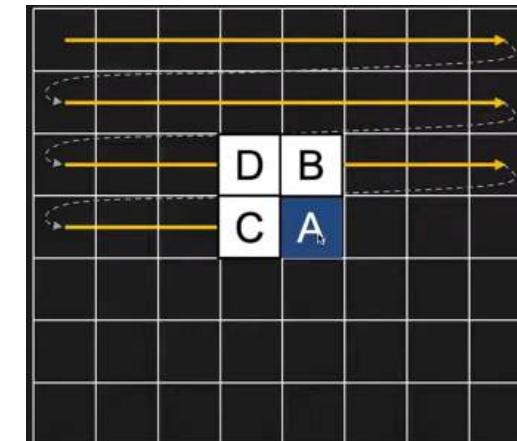


Each point in the **integral image** is a sum of the pixels **above and left** of the corresponding pixel in the source image

Viola Jones Detectors : Integral Image



- The value of the integral image at point 1 is the sum of the pixels in rectangle A
- The value at point 2 is A + B
- The value at point 3 is A + C
- The value at point 4 is A + B + C + D.
- Therefore, the sum of pixels in region D can simply be computed as : $4+1-(2+3)$.



Raster Scanning

Viola Jones Detectors : Integral Image

Source Image Pixel Values									Computed Integral Image							
1	2	4	6	7	9	10	11		1	3	7	13	20	29	39	50
21	2	2	4	3	2	1	1		22	26	32	42	52	63	74	86
1	2	1	3	4	5	9	10		23	29	36	49	63	79	99	121
1	2	7	8	9	2	27	1		24	32	46	67	90	108	155	178
2	3	7	8	9	1	1	1		26	38	59	88	120	141	183	207
1	3	2	2	2	7	22	21		27	42	65	96	128	159	190	218
21	7	3	9	10	11	17	18		48	70	96	136	184	224	264	302
1	1	1	4	9	8	7	2		49	72	101	145	193	234	274	312

#Long way

$$1 + 2 + 4 + 6 + 7 + 21 + 2 + 2 + 4 + 3 + 1 + 2 + 1 + \\ 3 + 4 + 1 + 2 + 7 + 8 + 9 = 90$$

$$9 + 10 + 11 + 2 + 1 + 1 + 5 + 9 + 10 + 2 + 27 + 1 = 88$$

$$\Delta = 90 - 88 = 2$$

#Short Way

$$178 - 90 = 88$$

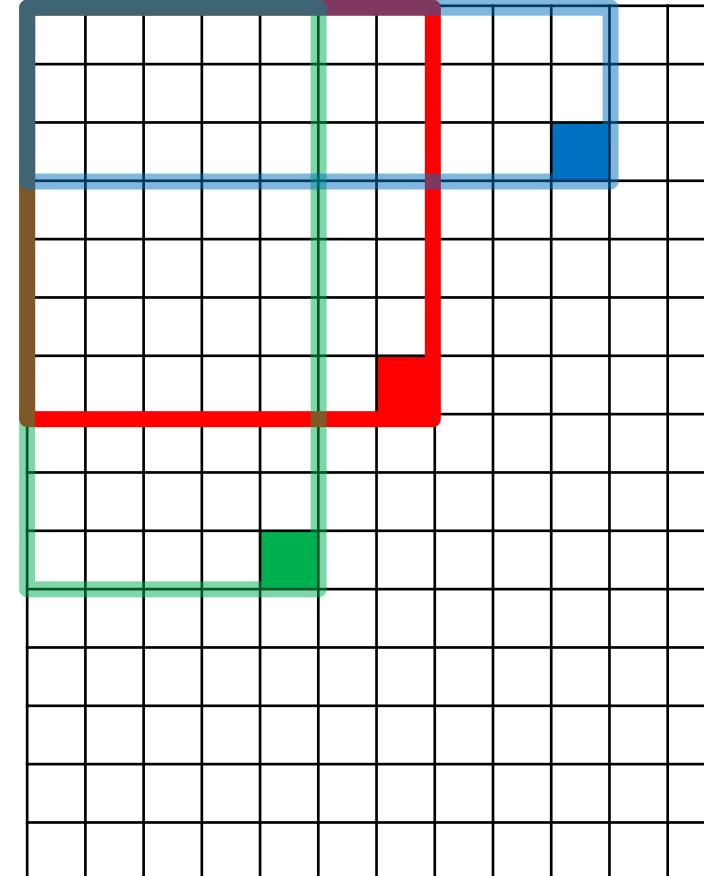
$$\Delta = 90 - 88 = 2$$

Viola Jones Detectors : Integral Image

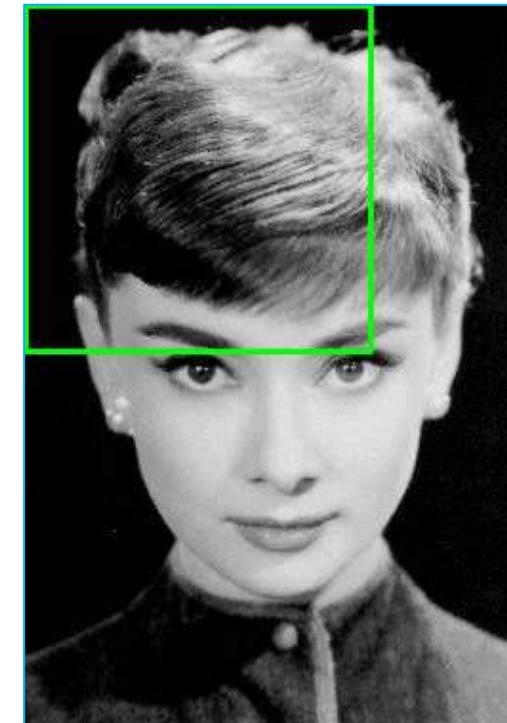
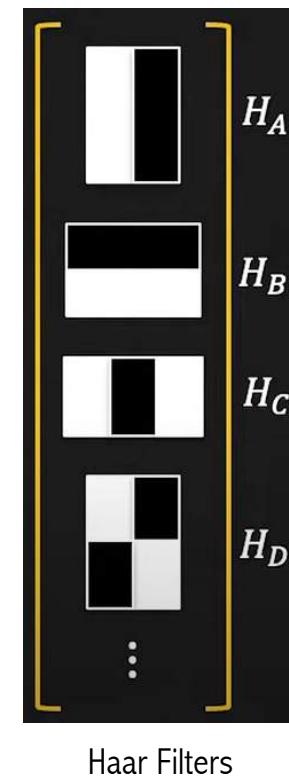
The trick is to compute an “integral image.” Every pixel is the sum of its neighbors to the upper left.

Sequentially compute using:

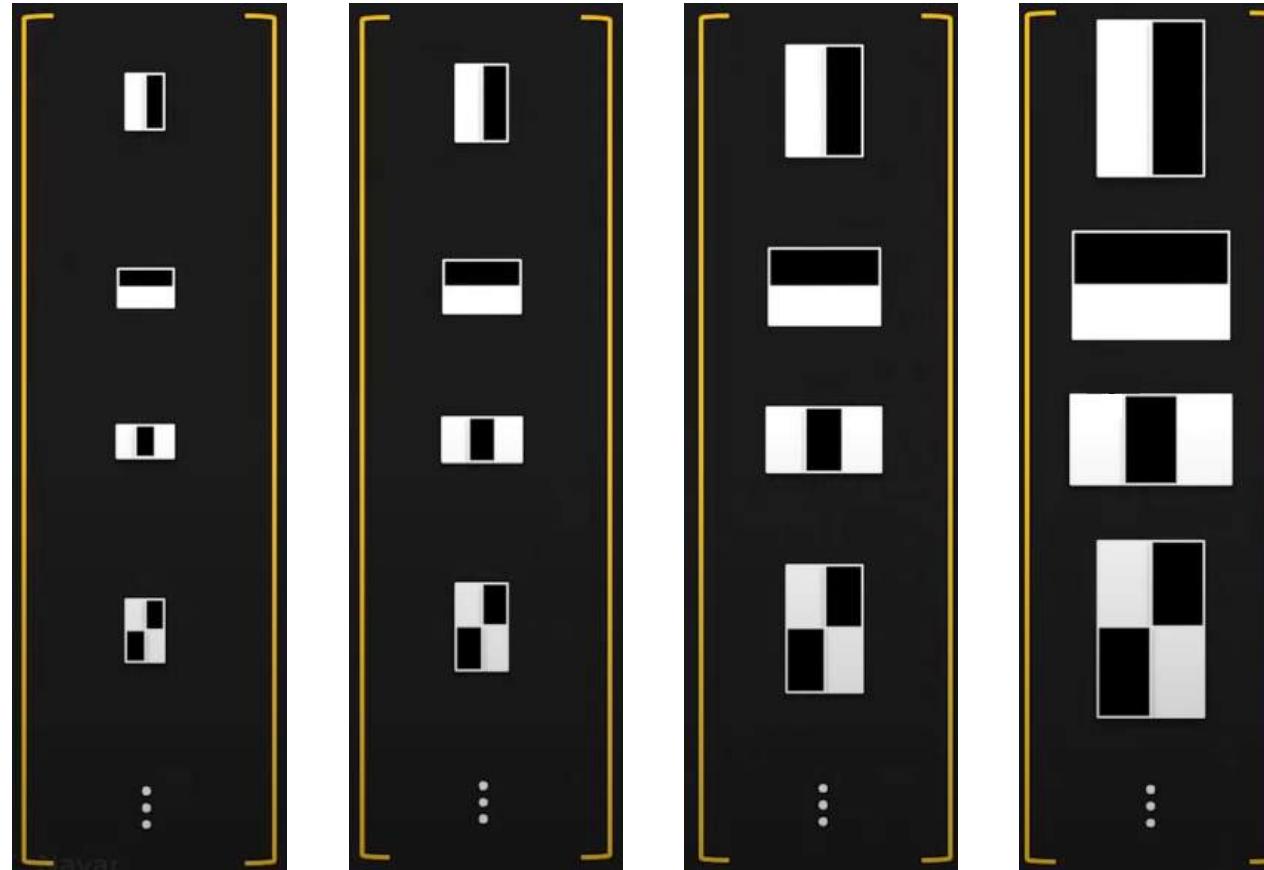
$$\begin{aligned} I(x, y) = & I(x, y) + \\ & I(x - 1, y) + I(x, y - 1) - \\ & I(x - 1, y - 1) \end{aligned}$$



Viola Jones Detectors : Harr Feature



Haar Features for Multiple Scales

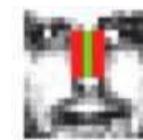


Can we create a good classifier using just a small subset of all possible features?

$$V_A[i,j] = \sum_m \sum_n I[m - i, n - j] H_A[m, n]$$

Compute Haar Features at different scales to detect faces of different sizes

As stated previously there can be approximately 160,000 + feature values within a detector at 24*24 base resolution which need to be calculated. But it is to be understood that only a few set of features will be useful among all these features to identify a face.



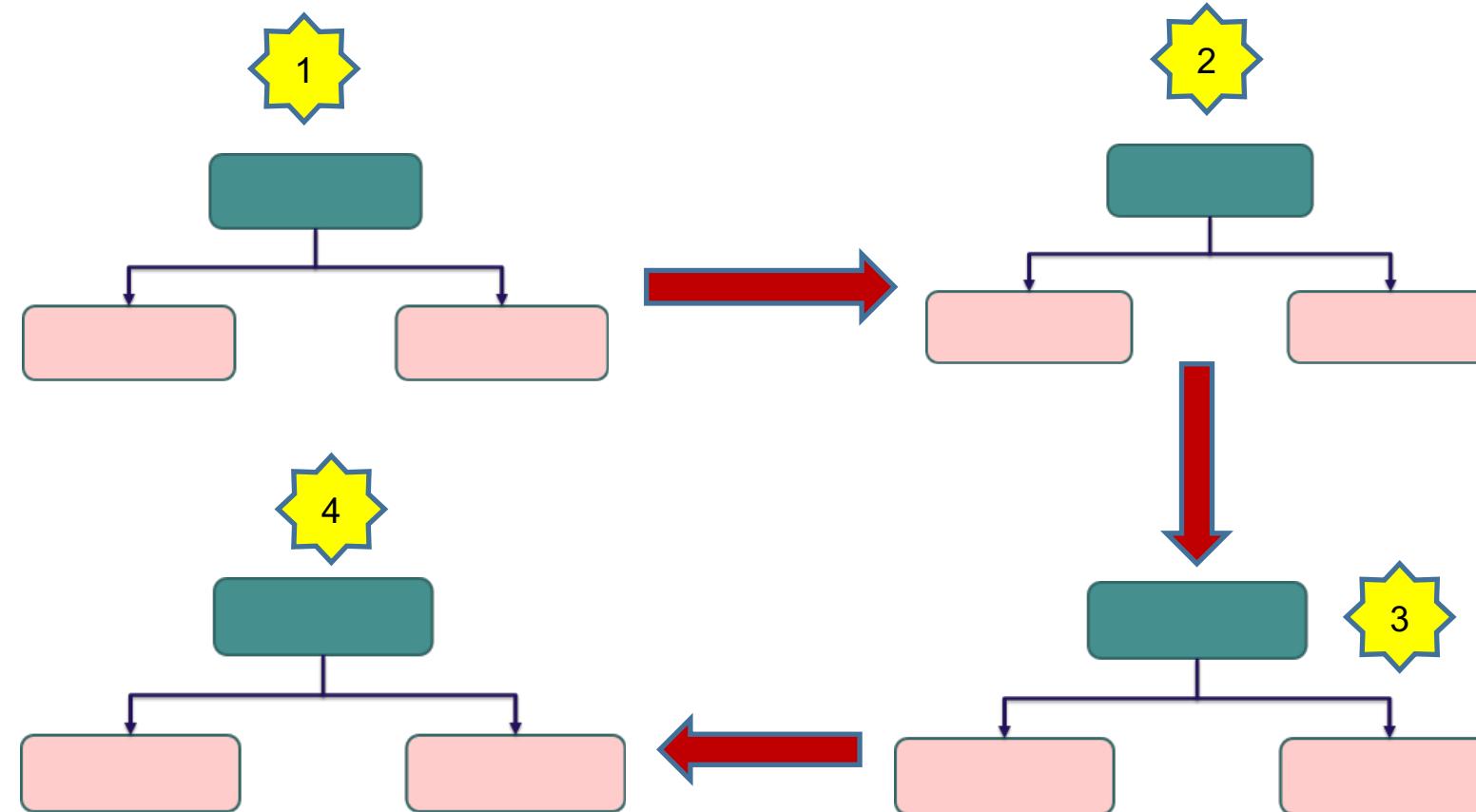
Relevant feature



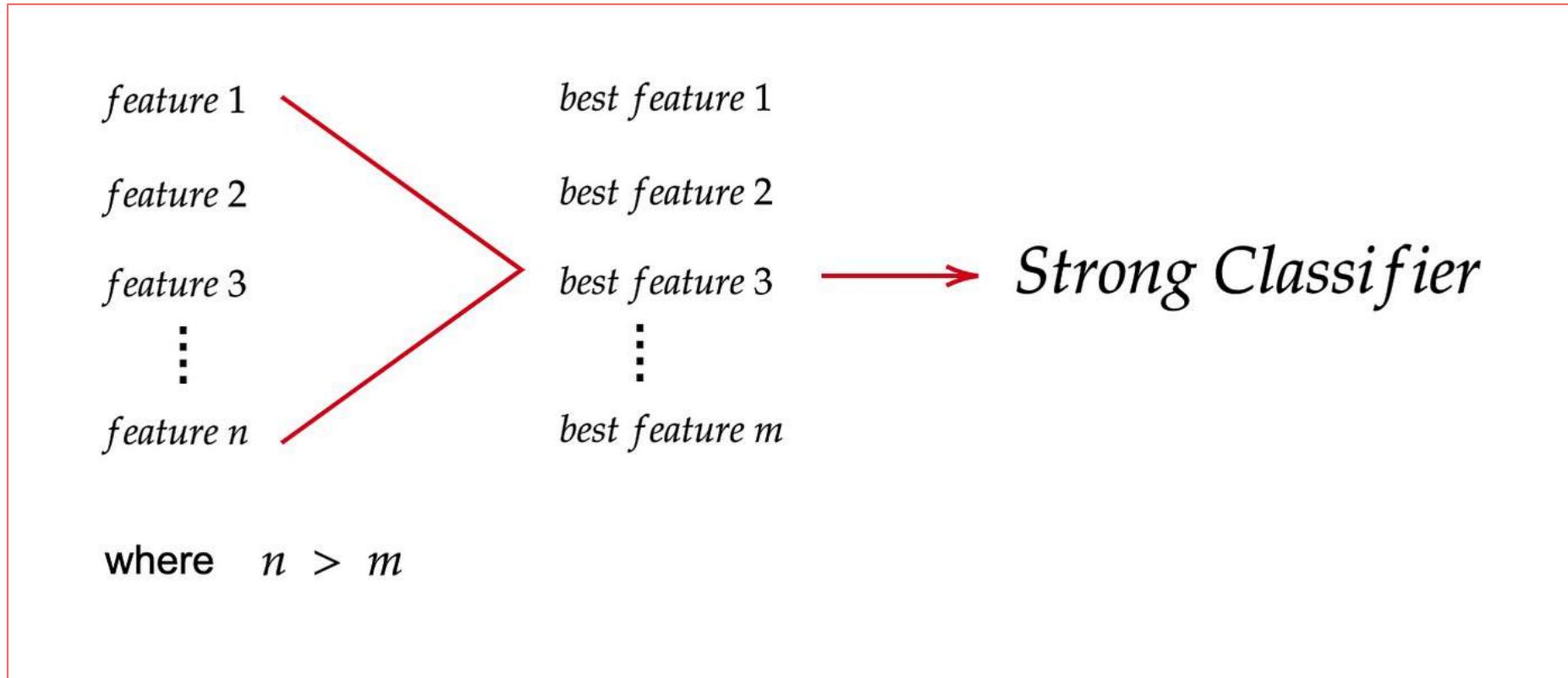
Irrelevant feature

Feature detecting a vertical edge is useful detecting a nose but irrelevant detecting a lip.

AdaBoost: FOREST OF STUMP

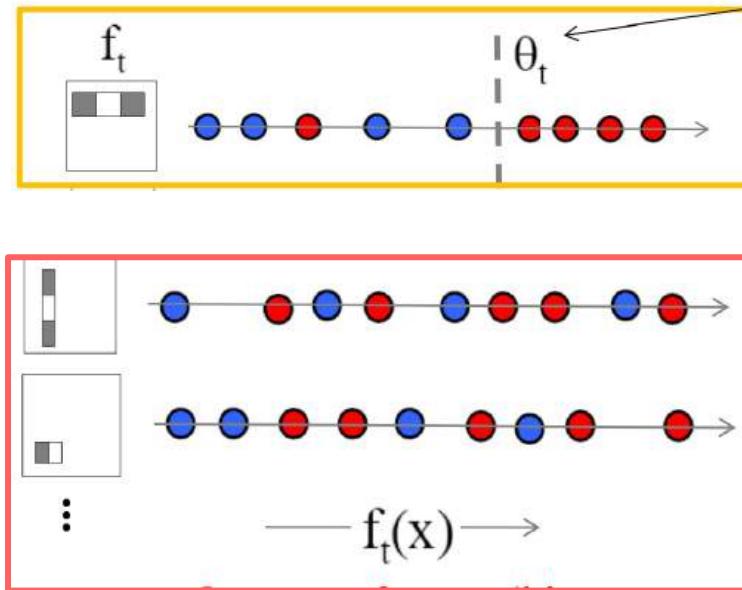


Adaboost for Face Detection



The goal of using the AdaBoost algorithm is to extract the best features from n features

AdaBoost for Feature+Classifier Selection



θ^t is a threshold for classifier h^t

Resulting weak classifier:

$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ 0 & \text{otherwise} \end{cases}$$

For next round, reweight the examples according to errors, choose another filter/threshold combo.

Want to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of weighted error

- Each weak classifier works on exactly one rectangle feature.
- Each weak classifier has 3 associated variables
 - 1. its threshold θ
 - 2. its polarity p
 - 3. its weight α
- The polarity can be 0 or 1
- The weak classifier computes its one feature f
 - When the polarity is 1, we want $f > \theta$ for face
 - When the polarity is 0, we want $f < \theta$ for face
- The weight will be used in the final classification by AdaBoost.

$h(x) = 1 \text{ if } p \cdot f(x) < p\theta, \text{ else } 0$
used for the combination step

The code does not actually compute h .

1. Initialized weights for each training example
2. Normalized all the weights
3. Then we selected the best weak classifier based on the weighted error of the training examples
4. Then updated the weights according to the error of the chosen best weak classifier
5. Repeated steps 2-4 N times, where N is the desired number of weak classifiers

AdaBoost Algorithm modified by Viola Jones

- Given example images $(x_1, y_1), \dots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \dots, T$:
 - Normalize the weights,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$$

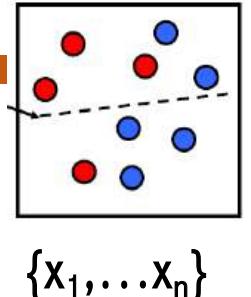
so that w_t is a probability distribution.

- For each feature, j , train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t , $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$. sum over training samples
- Choose the classifier, h_t , with the lowest error ϵ_t .
- Update the weights:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

NOTE: Our code
uses equal weights
for all samples



For T rounds:

meaning we will
construct T weak
classifiers

Normalize weights

Find the best threshold and polarity for each
feature, and return error.

Re-weight the examples:
Incorrectly classified -> more weight
Correctly classified -> less weight

AdaBoost for Feature+Classifier Selection

- Given example images $(x_1, y_1), \dots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \dots, T$:

- Normalize the weights,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$$

so that w_t is a probability distribution.

- For each feature, j , train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t , $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
- Choose the classifier, h_t , with the lowest error ϵ_t .
- Update the weights:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-\epsilon_t}$$

where $\epsilon_t = 0$ if example x_i is classified correctly, $\epsilon_t = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

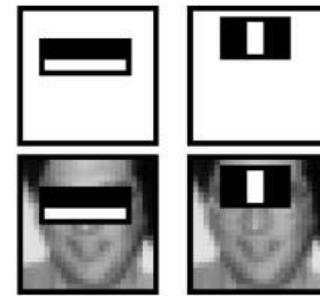
- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

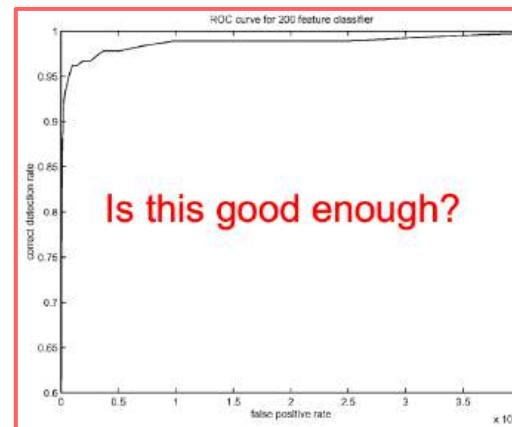
$$\text{where } \alpha_t = \log \frac{1}{\beta_t}$$

The AdaBoost algorithm for classifier learning. Each round t of boosting selects one feature from the 180,000 potential feature

First two features selected by boosting:



This feature combination can yield 100% detection rate and 50% false positive rate

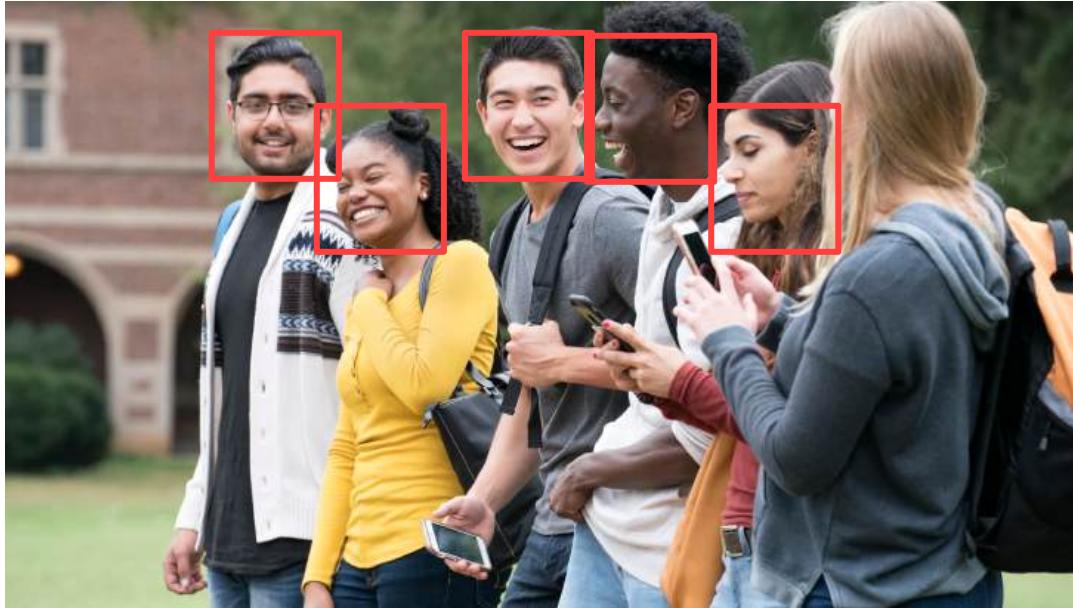


LIMITATION



A 200-feature classifier can yield 95% detection rate and a false positive rate of 1 in 14084

The Cascade Classifier

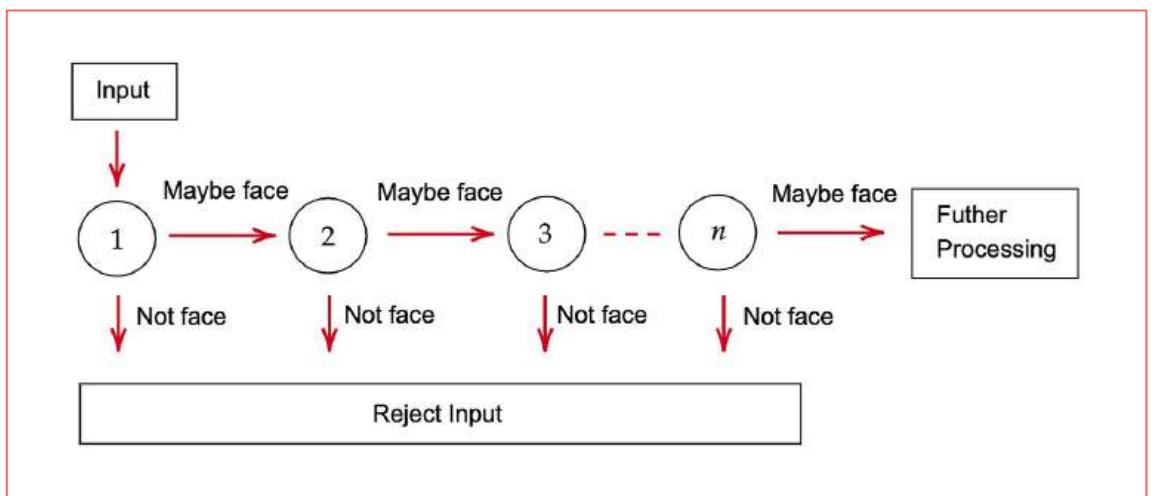


A series of classifiers are applied to every sub-window. These classifiers are simple decision trees :

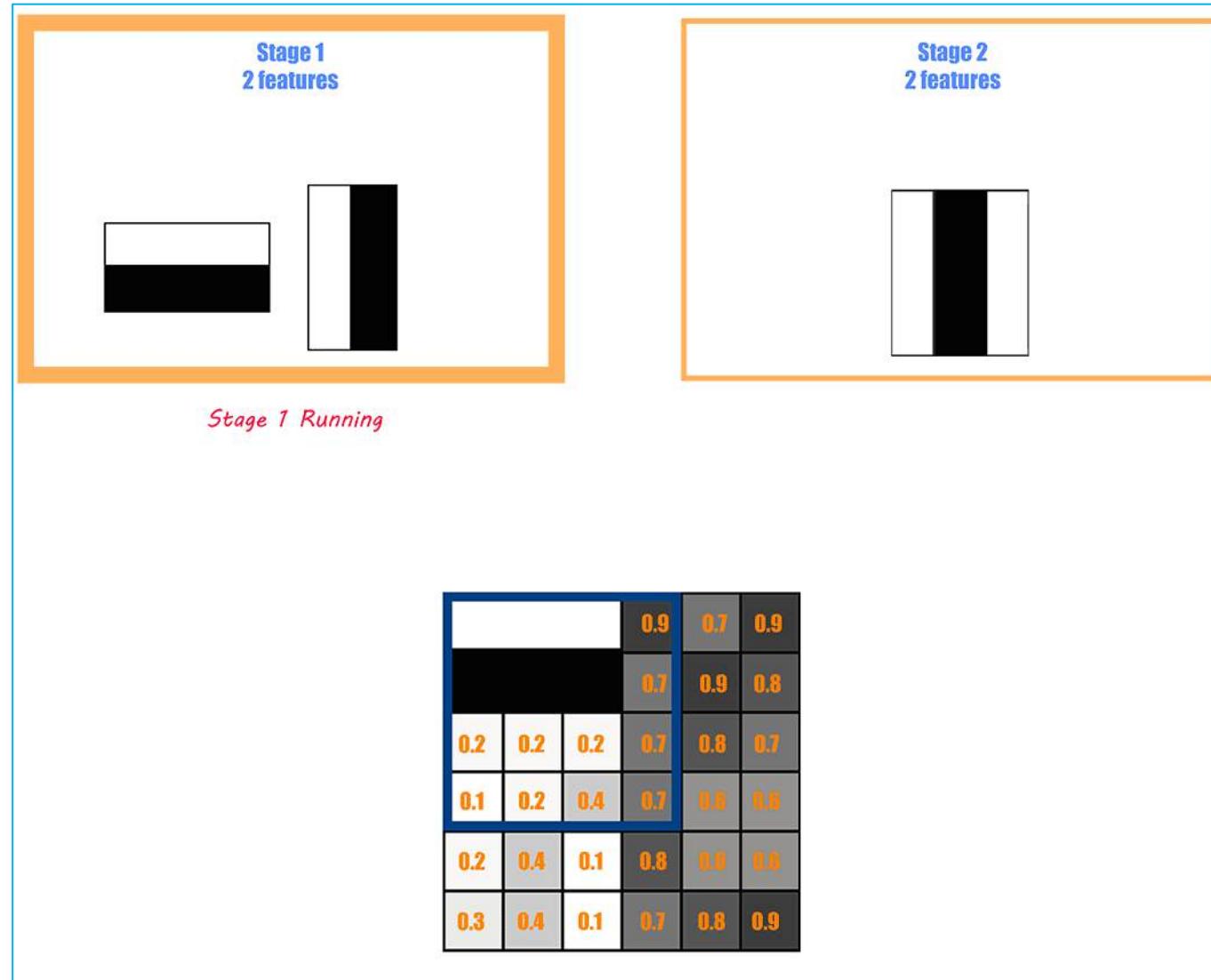
- if the first classifier is positive, we move on to the second
- if the second classifier is positive, we move on to the third.
- Any negative result at some point leads to a rejection of the sub-window as potentially containing a face

In an image, most of the image is a non-face region

The key idea is to reject sub-windows that do not contain faces while identifying regions that do. Since the task is to identify properly the face, we want to minimize the false negative rate, i.e the sub-windows that contain a face and have not been identified as such.



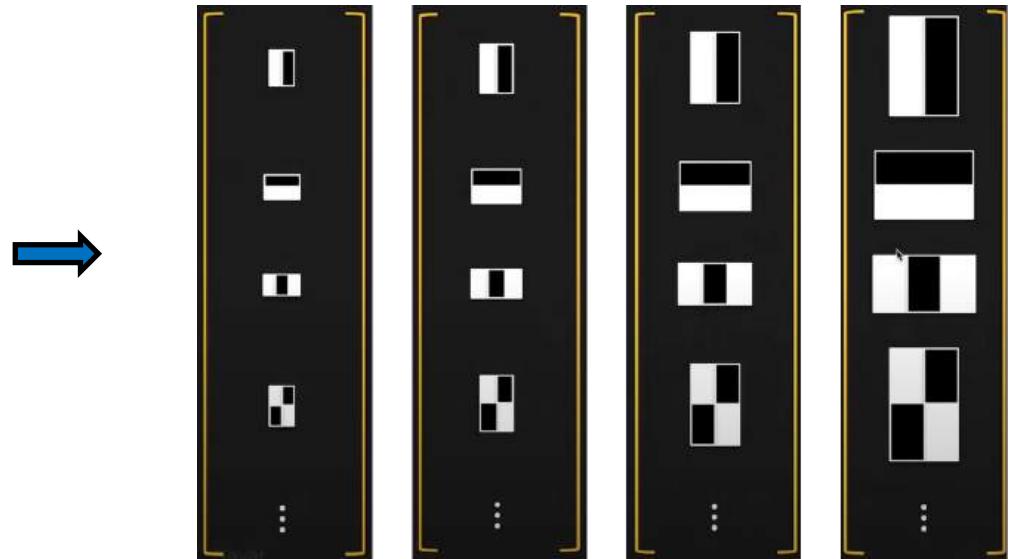
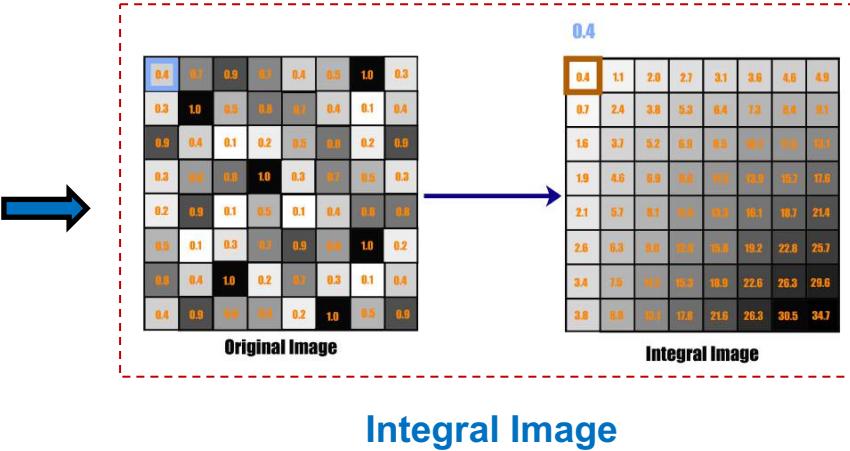
The Cascade Classifier



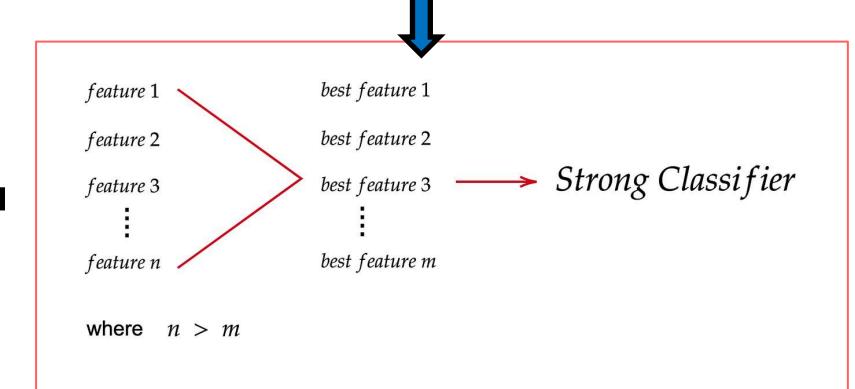
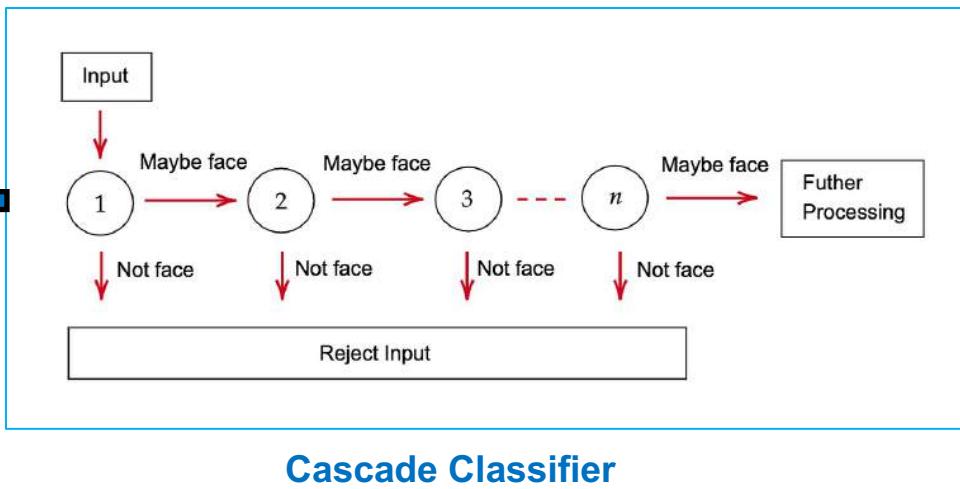
Summary



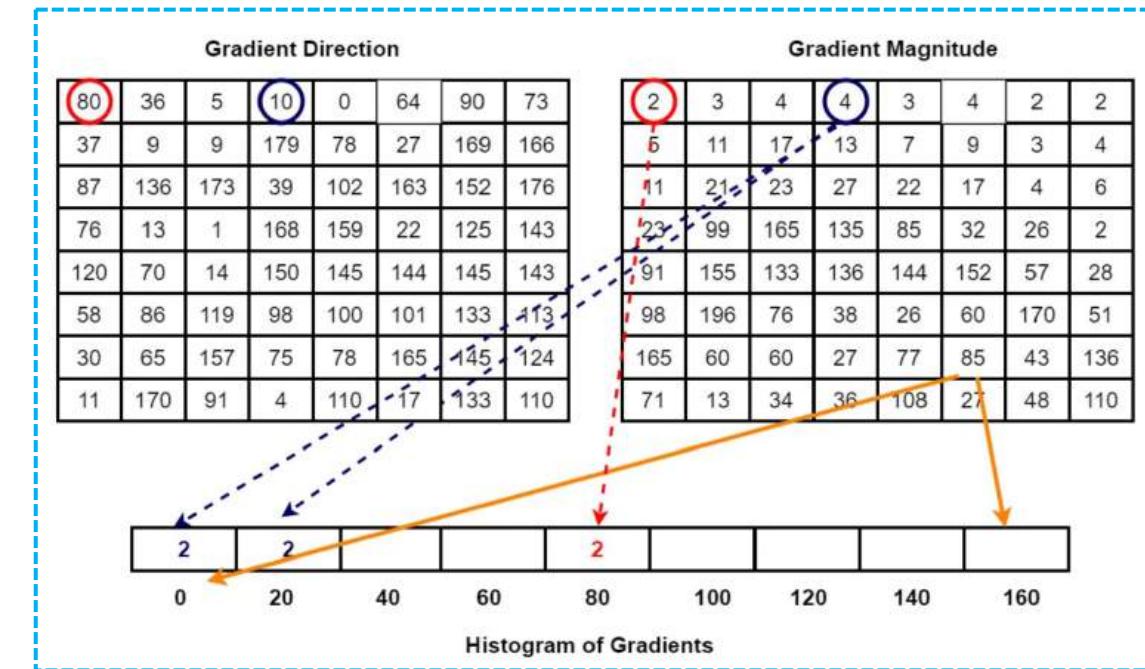
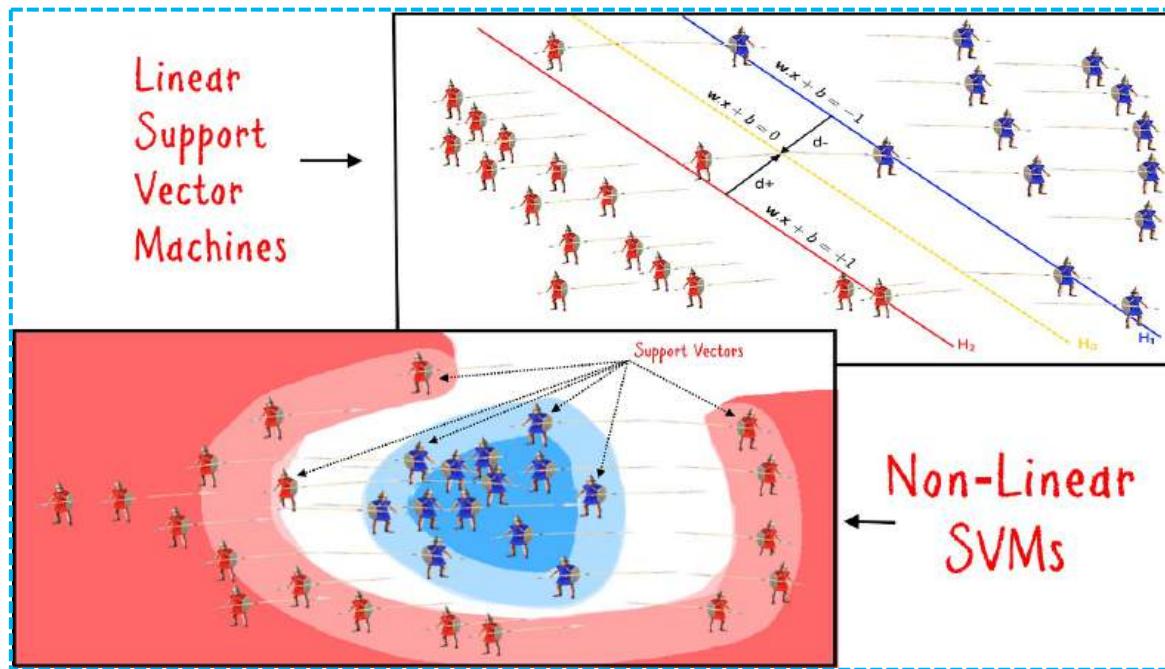
Input Image



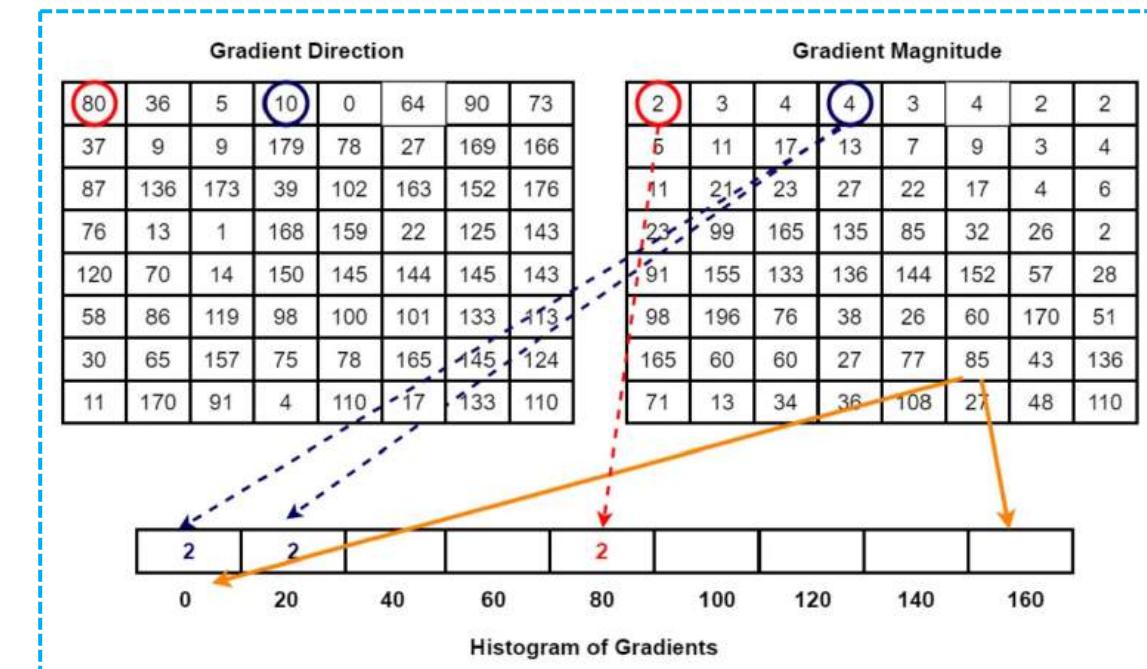
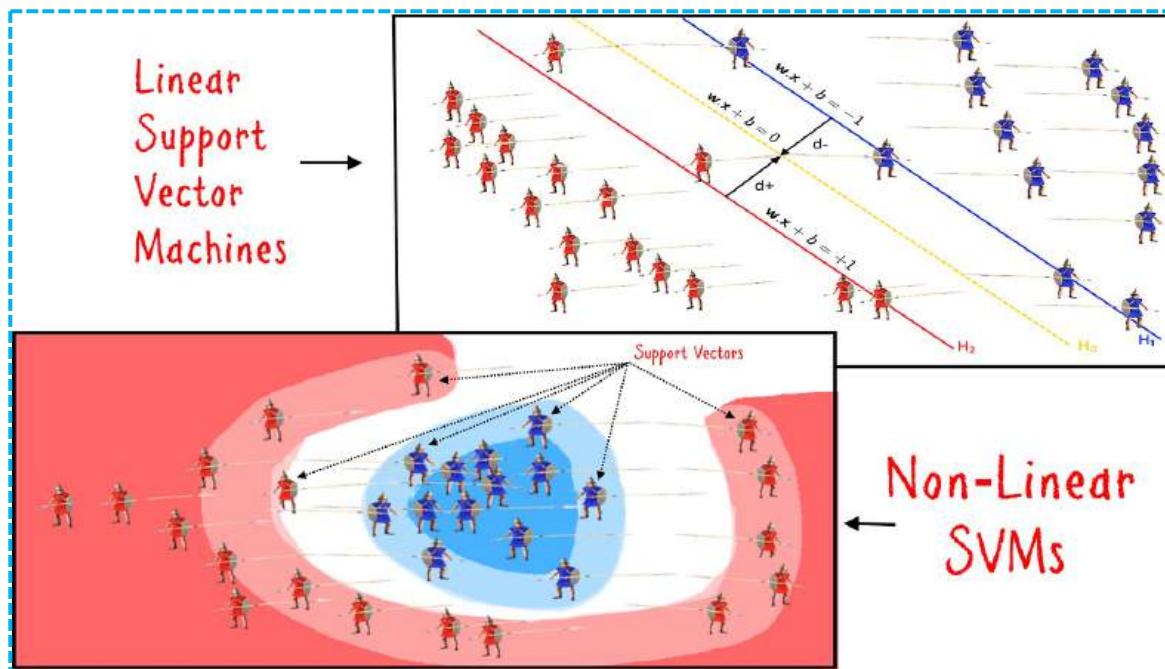
Output



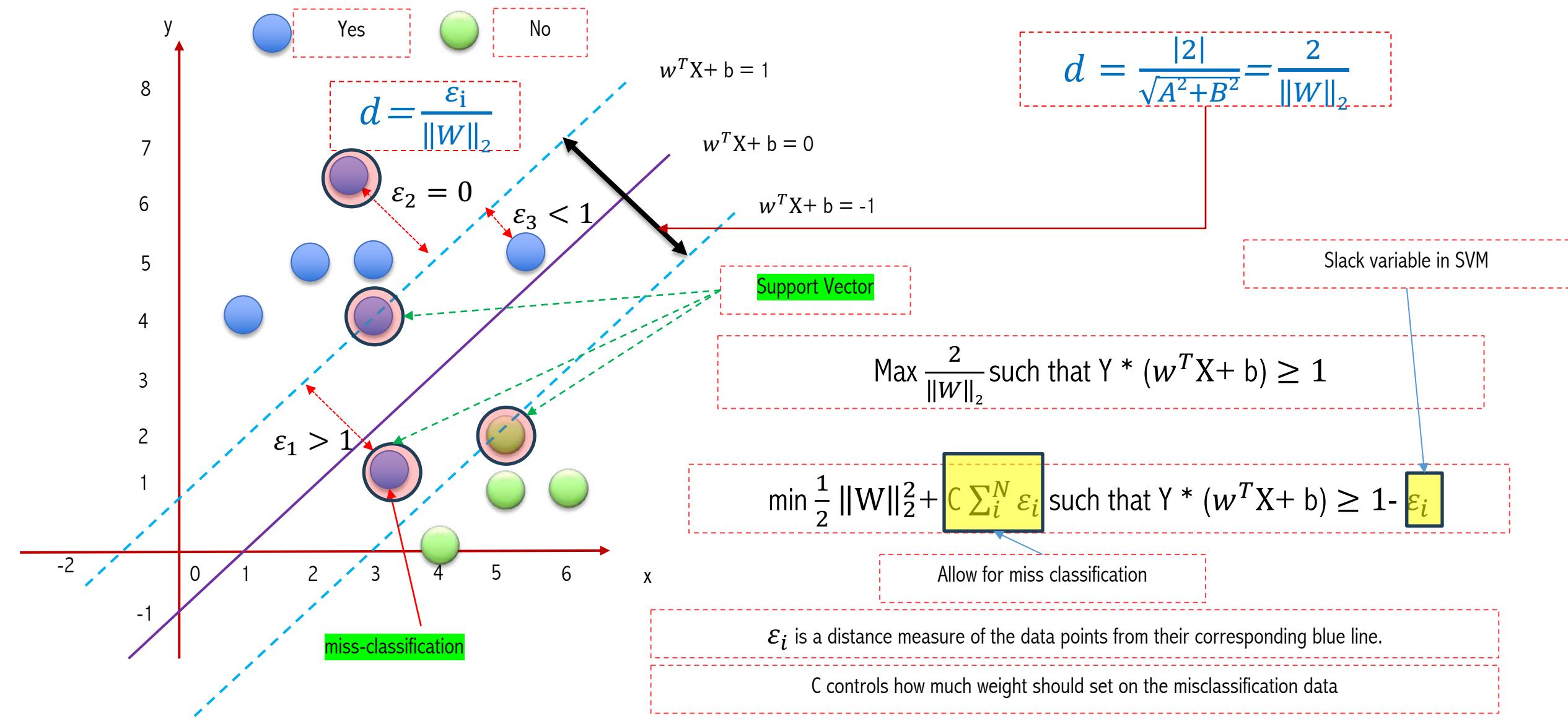
SVM and HOG for Object Detection



SVM and HOG for Object Detection



Support Vector Machine

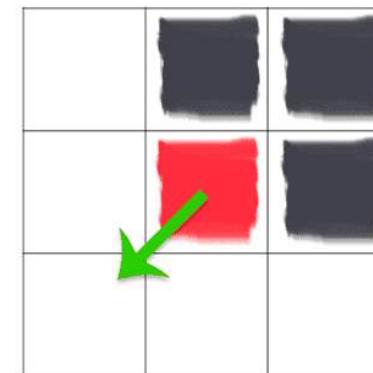


Histogram of Oriented Gradient

An image gradient is a directional change in the intensity or color in an image. The gradient of the image is one of the fundamental building blocks in image processing. For example, the Canny edge detector uses image gradient for edge detection.

131	162	232	84	91	207
104	93	139	101	237	109
243	26	252	196	135	126
185	135	230	48	61	225
157	124	25	14	102	108
5	155	116	218	232	249

255	0	0
255	255	0
255	255	255

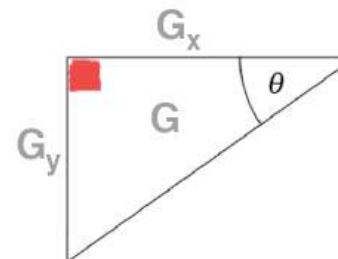


$$\theta = \arctan2(G_y, G_x) \times \left(\frac{180}{\pi} \right)$$

$$G_x = I(x+1, y) - I(x-1, y)$$

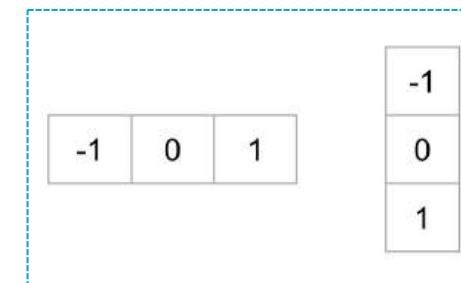
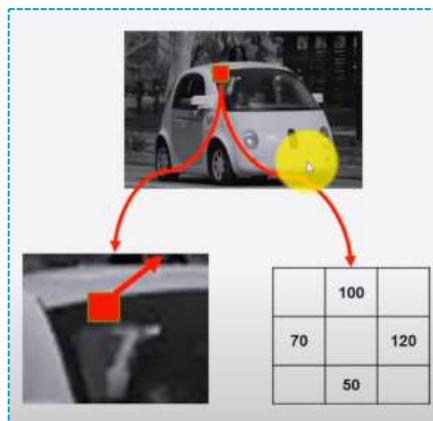
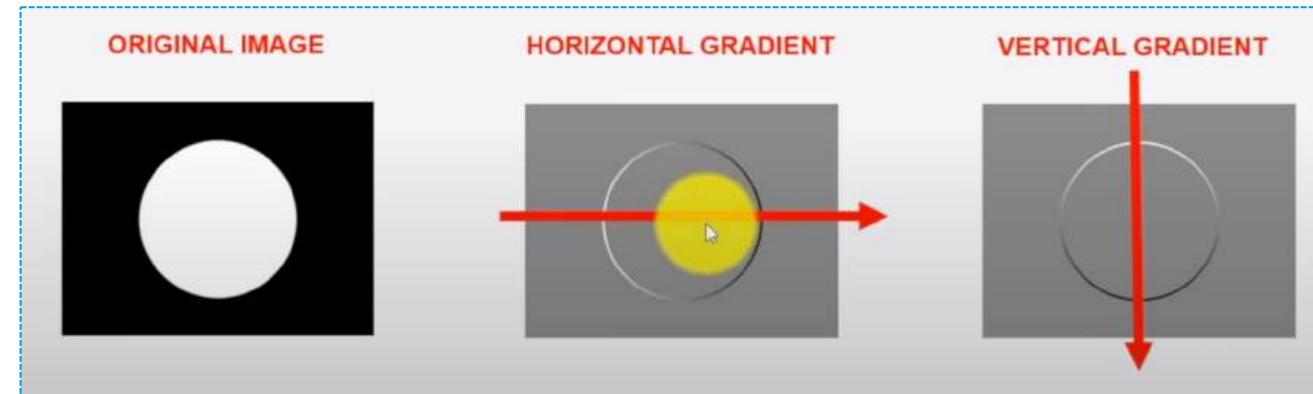
$$G_y = I(x, y+1) - I(x, y-1)$$

$$G = \sqrt{G_x^2 + G_y^2}$$



Histogram of Oriented Gradient

An image gradient is a directional change in the intensity or color in an image. The gradient of the image is one of the fundamental building blocks in image processing. For example, the Canny edge detector uses image gradient for edge detection.

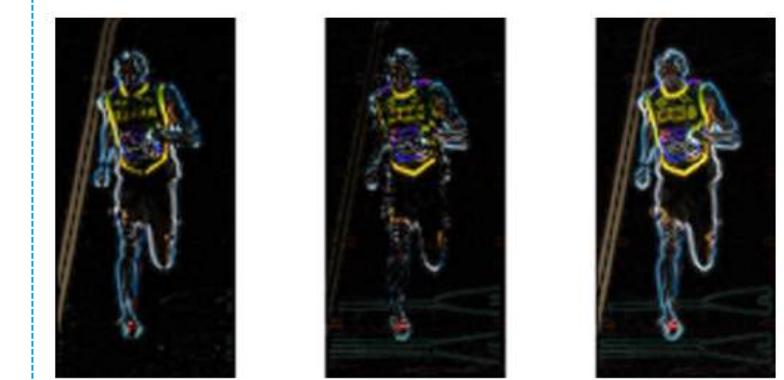
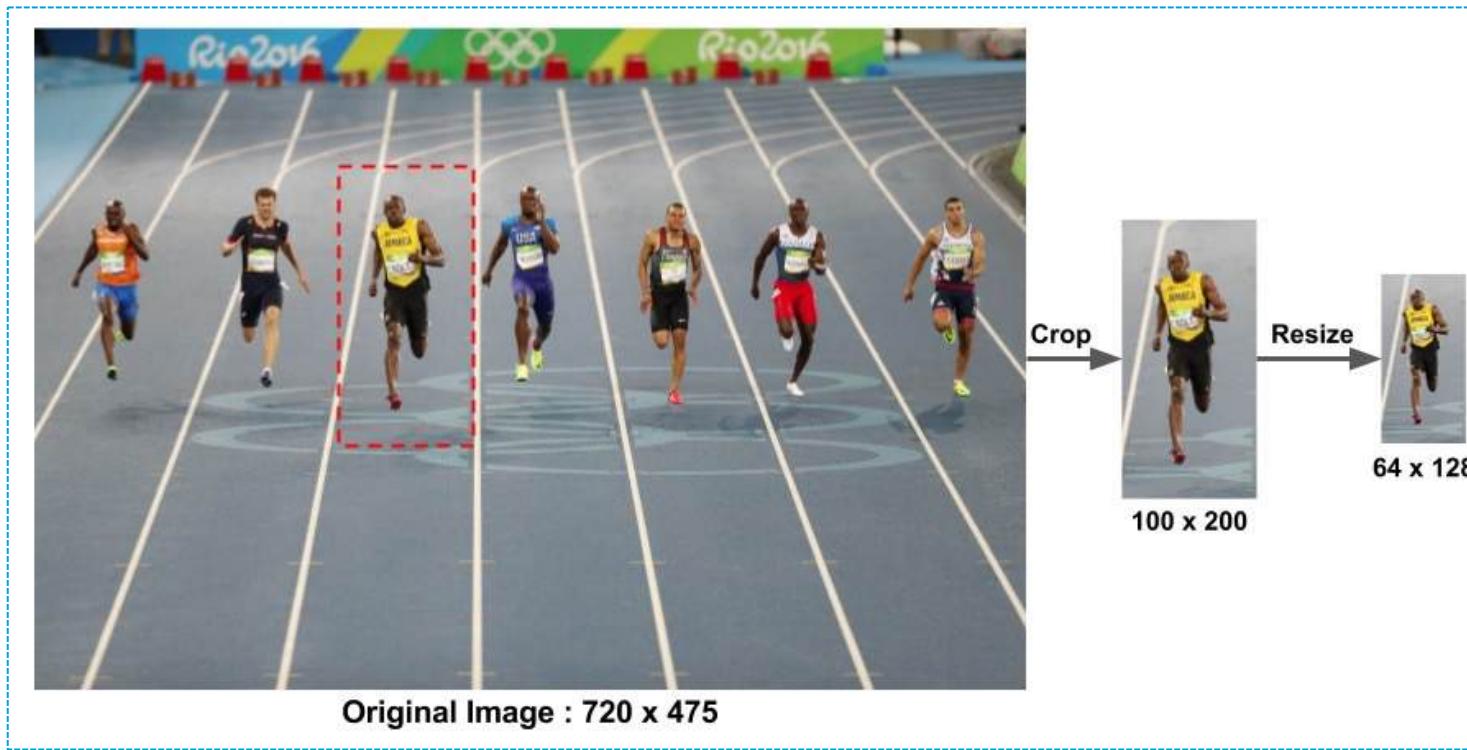


$$\text{Gradient Magnitude} = \sqrt{(50)^2 + (50)^2} = 70.1$$

$$\text{Gradient Angle} = \tan^{-1}(50/50) = 45^\circ$$

Histogram of Oriented Gradient

Step 1: preprocessing and compute gradient



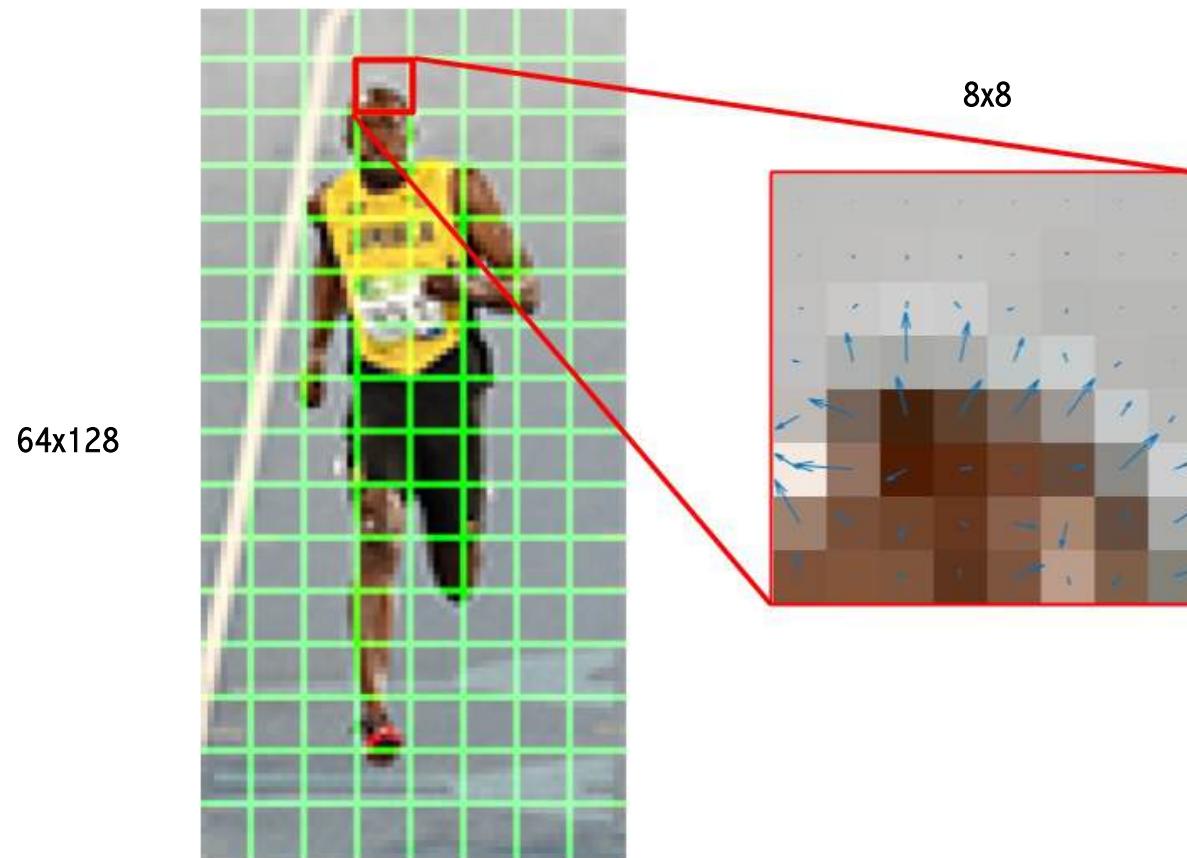
Left : Absolute value of x-gradient. Center : Absolute value of y-gradient. Right : Magnitude of gradient.

In the case of the HOG feature descriptor, the input image is of size $64 \times 128 \times 3 = 24576$ and the output feature vector is of length 3780



Histogram of Oriented Gradient

Step 2: Calculate Histogram of Gradients in 8×8 cells



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

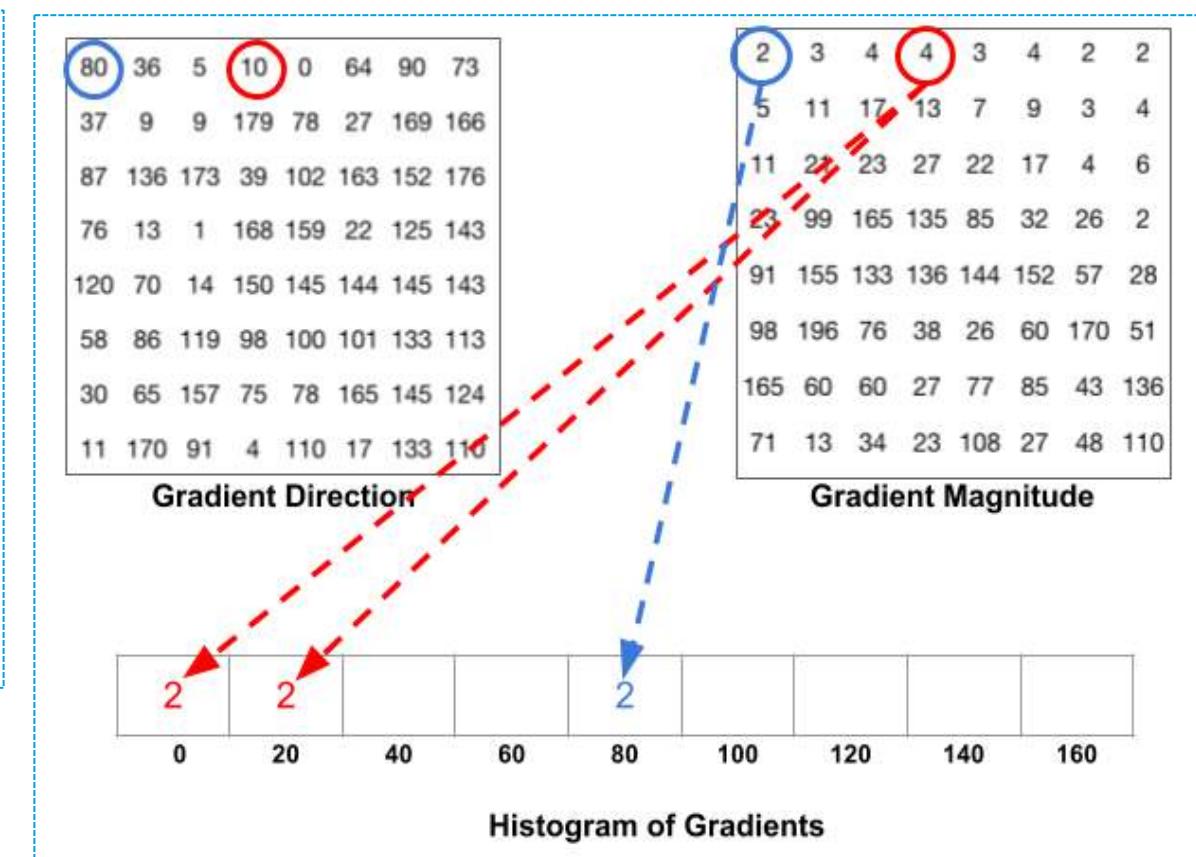
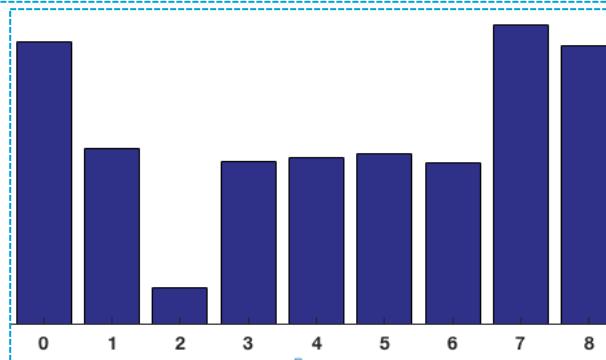
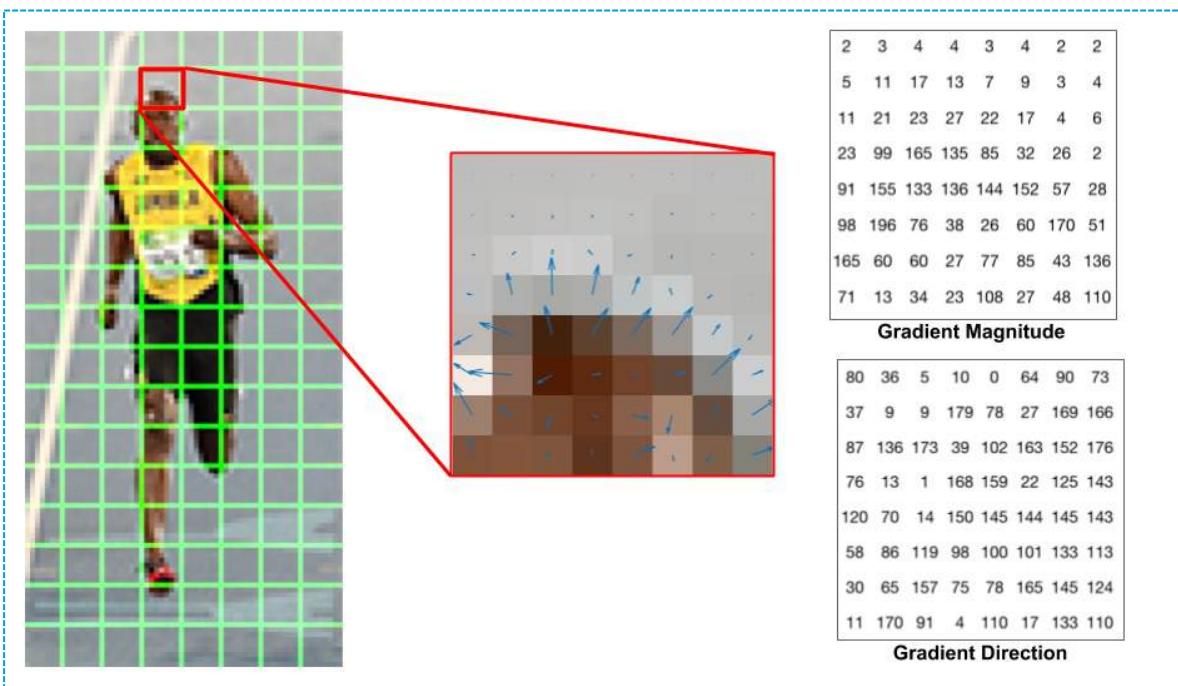
Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

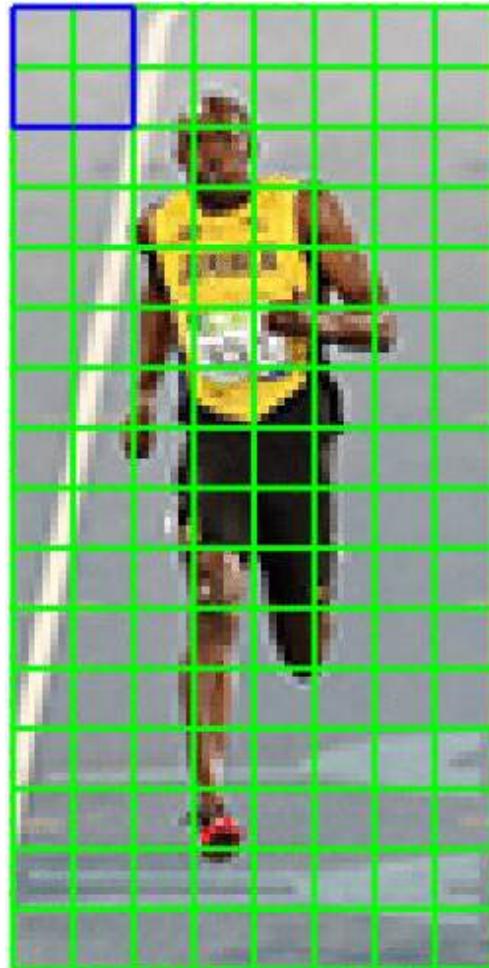
Histogram of Oriented Gradient

Step 3: Compute Histogram of Oriented Gradient



Histogram of Oriented Gradient

Step 4: 16×16 Block Normalization



Gradients for highlighted Block (having 36 features):

$$V = [a_1, a_2, a_3, \dots, a_{36}]$$

Root of the sum of squares:

$$k = \sqrt{a_1^2 + a_2^2 + a_3^2 + \dots + a_{36}^2}$$

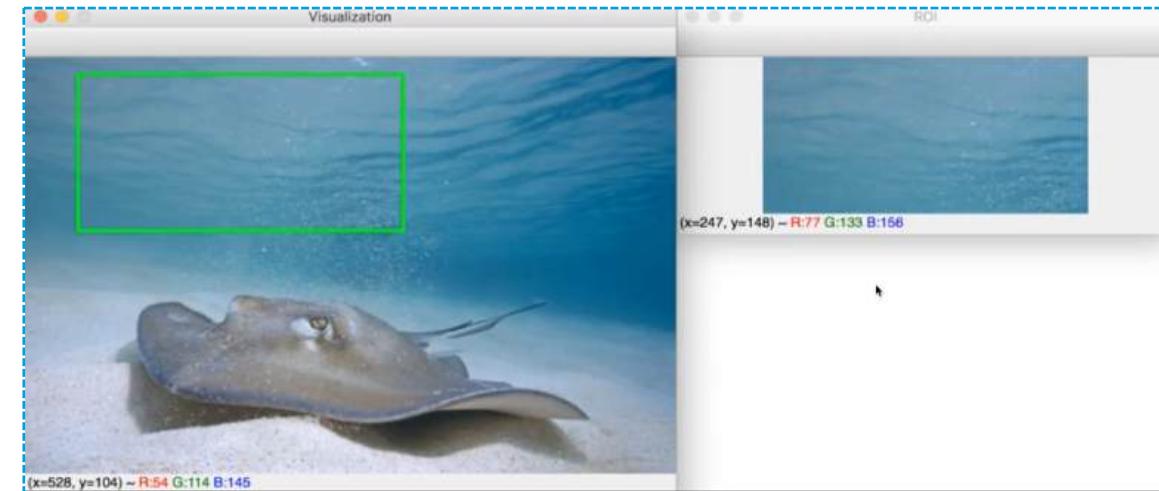
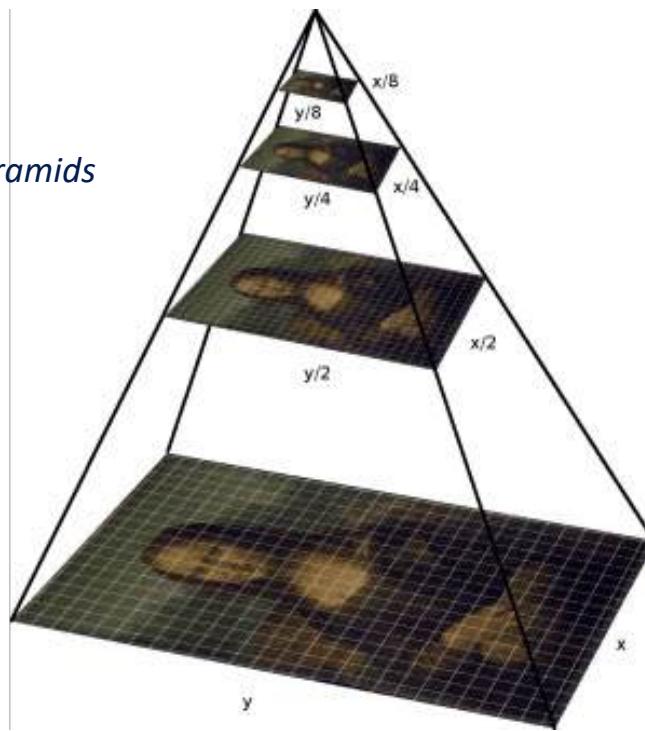
$$\text{Normalized vector} = \left(\frac{a_1}{k}, \frac{a_2}{k}, \frac{a_3}{k}, \dots, \frac{a_{36}}{k} \right)$$

Normalized vectors created for all 105 blocks, having 36 features each image Feature Descriptor: 1×3780 matrix

SVM + HOG

Advanced techniques:

Image pyramids



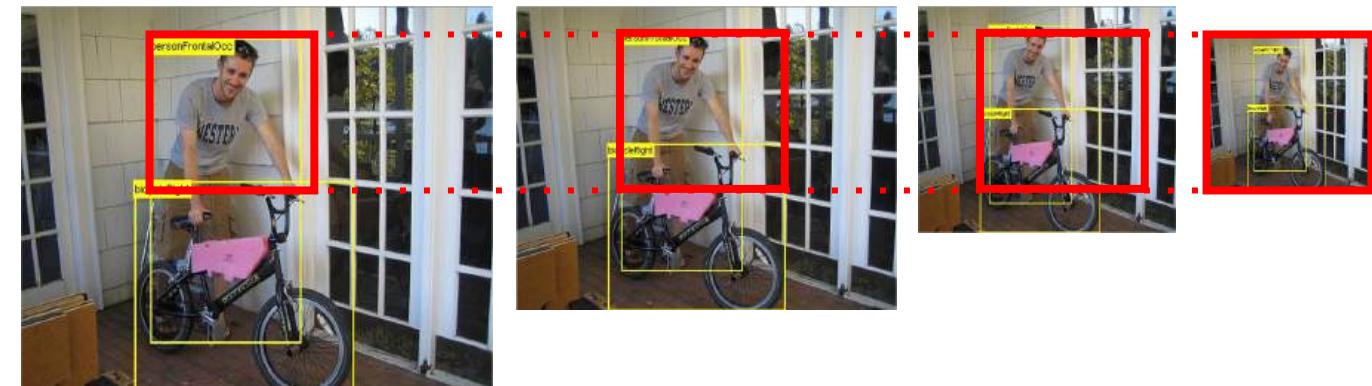
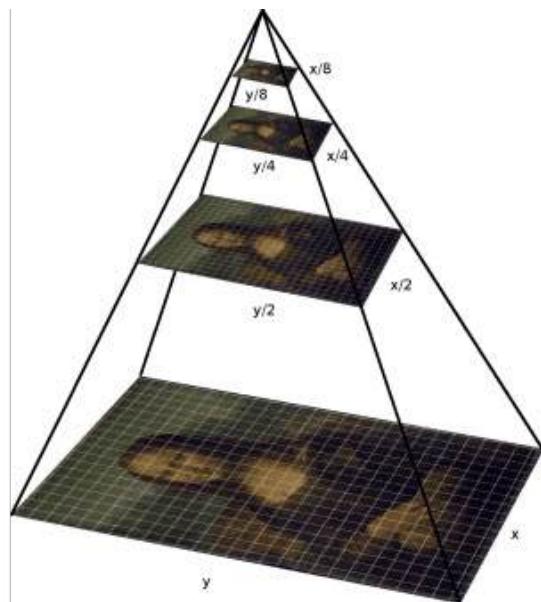
Sliding Window

non-maxima suppression



SVM + HOG

Image pyramids

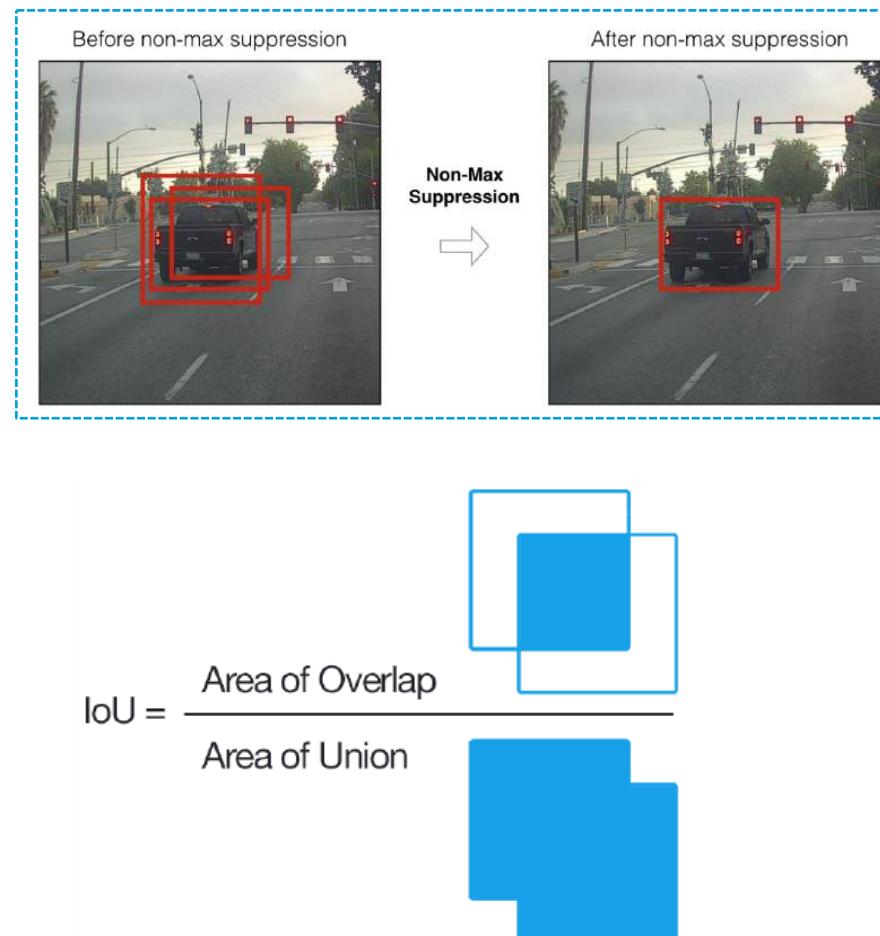


Smaller objects

Sliding Window – Location
Image Pyramid - Scale

Larger objects

SVM + HOG



Non-maxima suppression

Algorithm 1 Non-Max Suppression

```

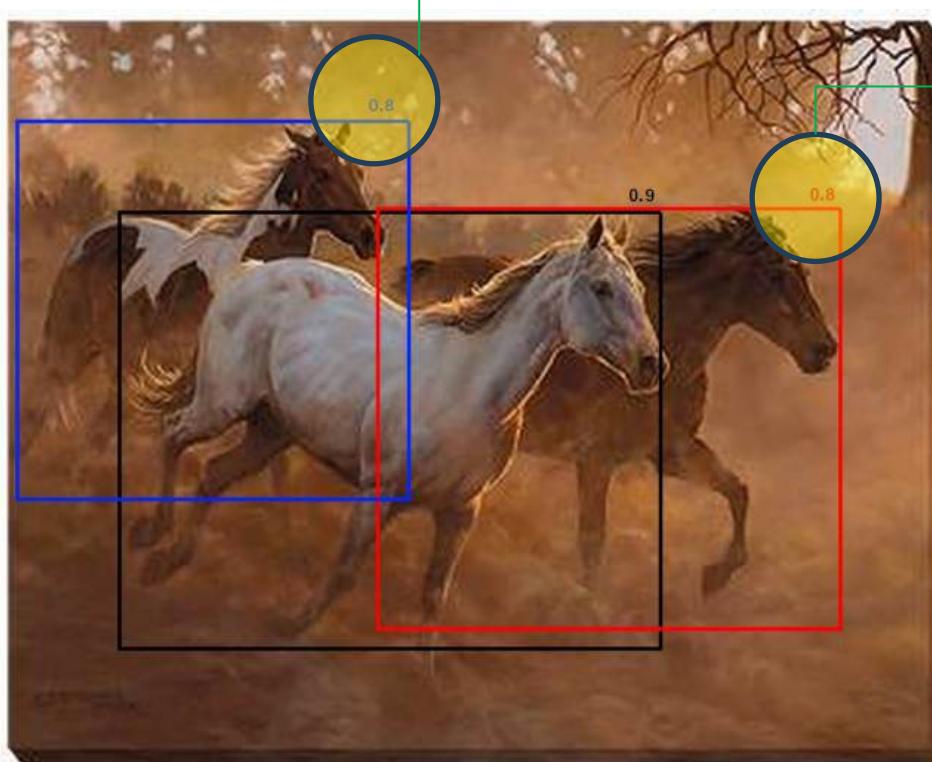
1: procedure NMS( $B, c$ )
2:    $B_{nms} \leftarrow \emptyset$  Initialize empty set
3:   for  $b_i \in B$  do  $\Rightarrow$  Iterate over all the boxes
4:      $discard \leftarrow \text{False}$  Take boolean variable and set it as false. This variable indicates whether  $b(i)$  should be kept or discarded
5:     for  $b_j \in B$  do Start another loop to compare with  $b(i)$ 
6:       if same( $b_i, b_j$ )  $> \lambda_{nms}$  then If both boxes having same IOU
7:         if score( $c, b_j$ )  $>$  score( $c, b_i$ ) then Compare the scores. If score of  $b(i)$  is less than that of  $b(j)$ ,  $b(i)$  should be discarded, so set the flag to True.
8:            $discard \leftarrow \text{True}$ 
9:         if not  $discard$  then Once  $b(i)$  is compared with all other boxes and still the discarded flag is False, then  $b(i)$  should be considered. So
10:             $B_{nms} \leftarrow B_{nms} \cup b_i$  add it to the final list.
11:   return  $B_{nms}$  Do the same procedure for remaining boxes and return the final list

```

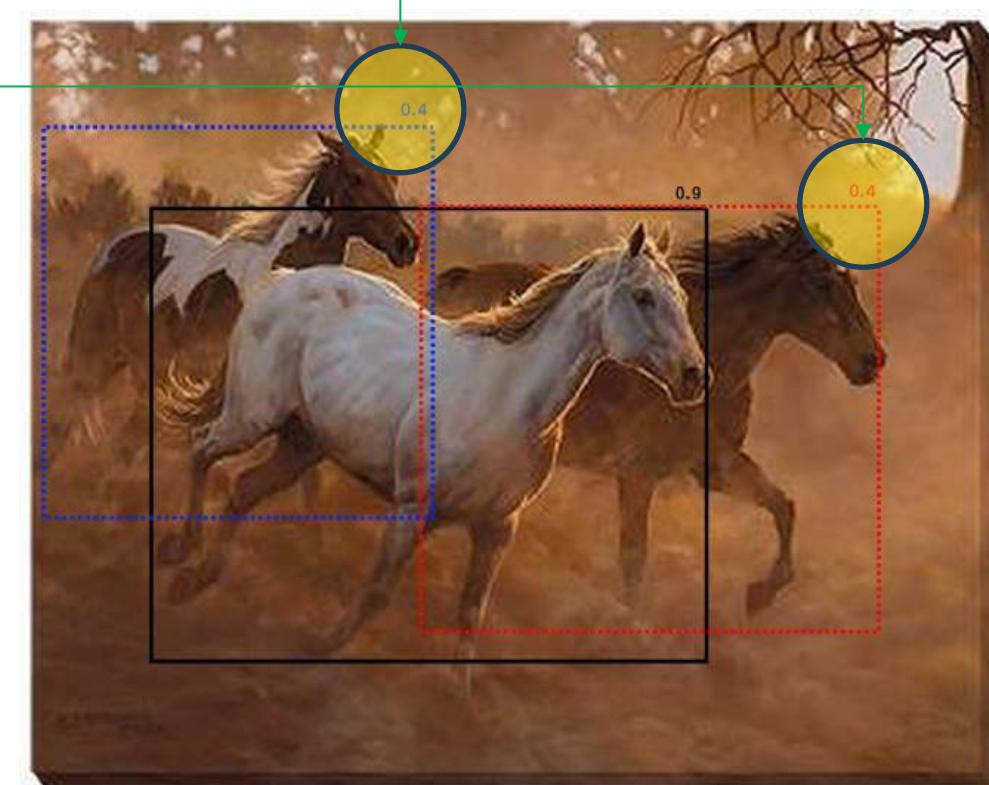
SVM + HOG

Non-maxima suppression

Hard NMS



Soft NMS



SVM + HOG

Non-maxima suppression

Hard NMS

Algorithm 1 Non-Max Suppression

```

1: procedure NMS( $B, c$ )
2:    $B_{nms} \leftarrow \emptyset$  Initialize empty set
3:   for  $b_i \in B$  do  $\Rightarrow$  Iterate over all the boxes
   Take boolean variable and set it as false. This variable indicates whether  $b(i)$ 
4:      $discard \leftarrow \text{False}$  should be kept or discarded
5:     for  $b_j \in B$  do Start another loop to compare with  $b(i)$ 
6:       if  $\text{same}(b_i, b_j) > \lambda_{\text{nms}}$  then If both boxes having same IOU
7:         if  $\text{score}(c, b_j) > \text{score}(c, b_i)$  then
8:            $discard \leftarrow \text{True}$  Compare the scores. If score of  $b(i)$  is less than that
   of  $b(j)$ ,  $b(i)$  should be discarded, so set the flag to
9:           if not  $discard$  then True.
10:           $B_{nms} \leftarrow B_{nms} \cup b_i$  Once  $b(i)$  is compared with all other boxes and still the
    discarded flag is False, then  $b(i)$  should be considered. So
    add it to the final list.
11: return  $B_{nms}$  Do the same procedure for remaining boxes and return the final list

```

Soft NMS

Input : $\mathcal{B} = \{b_1, \dots, b_N\}, \mathcal{S} = \{s_1, \dots, s_N\}, N_t$
 \mathcal{B} is the list of initial detection boxes
 \mathcal{S} contains corresponding detection scores
 N_t is the NMS threshold

```

begin
   $\mathcal{D} \leftarrow \{\}$ 
  while  $\mathcal{B} \neq \text{empty}$  do
     $m \leftarrow \text{argmax } \mathcal{S}$ 
     $M \leftarrow b_m$ 
     $\mathcal{D} \leftarrow \mathcal{D} \cup M; \mathcal{B} \leftarrow \mathcal{B} - M$ 
    for  $b_i$  in  $\mathcal{B}$  do
      if  $iou(M, b_i) \geq N_t$  then
         $| \quad \mathcal{B} \leftarrow \mathcal{B} - b_i; \mathcal{S} \leftarrow \mathcal{S} - s_i$ 
      end
       $s_i \leftarrow s_i f(iou(M, b_i))$ 
    end
  end
  return  $\mathcal{D}, \mathcal{S}$ 
end

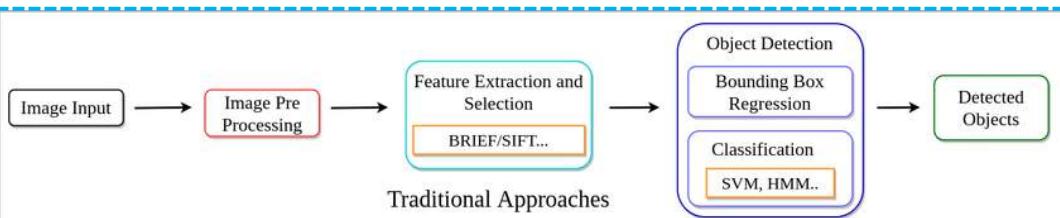
```

NMS

Soft-NMS

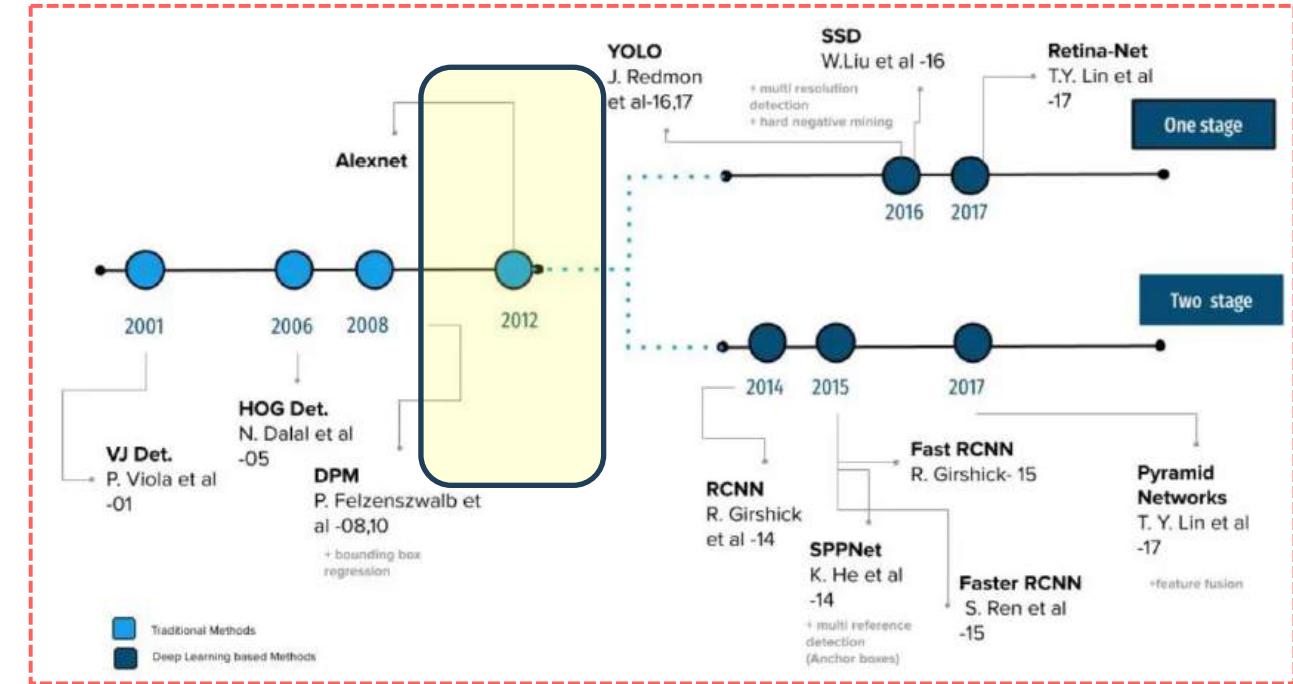
Traditional Detectors: Summary

Most traditional object detection algorithms like Viola–Jones, and Histogram of Oriented Gradients (HOG) are relied on extracting handcrafted features like edges, corners, gradients from the image and classical machine learning algorithms.



The traditional computer vision approaches were in the game until 2010.

From 2012, a new era of convolutional neural networks started when AlexNet (an image classification network) won the ImageNet Visual Recognition challenge 2012. The accuracy of 84.7% as compared to the second-best with an accuracy of 73.8%.



Outline

Introduction to Object Detection

Applications of Object Detection

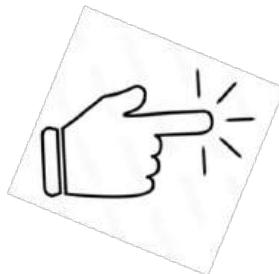
Object Detection Milestones

Traditional Object Detectors

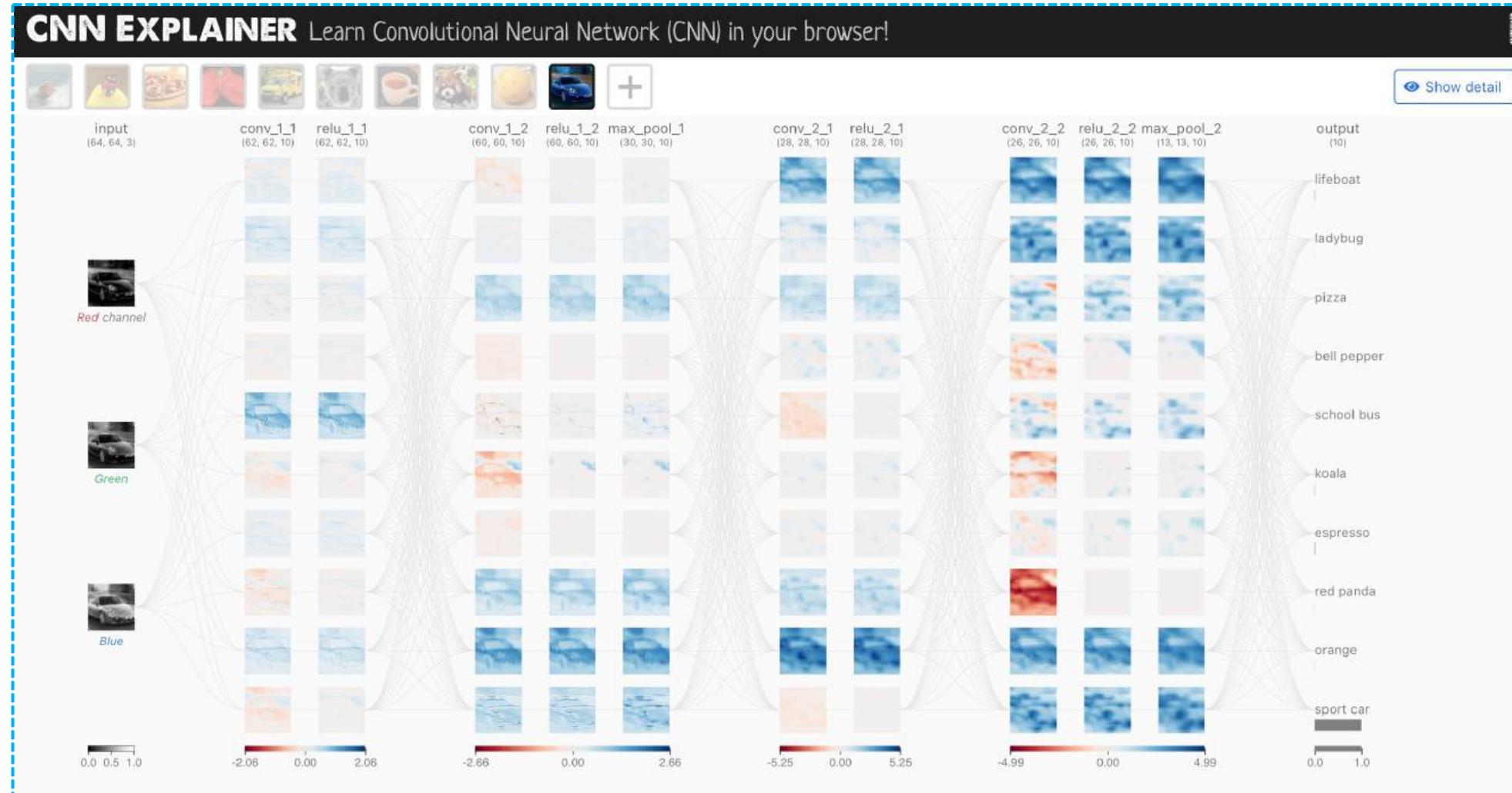
CNNs for Image Classification

Image Classifier to Object Detectors with CNN

CNN Limitations and Spatial Outputs



Convolutional Neural Network



<https://poloclub.github.io/cnn-explainer/>

Convolutional Neural Network

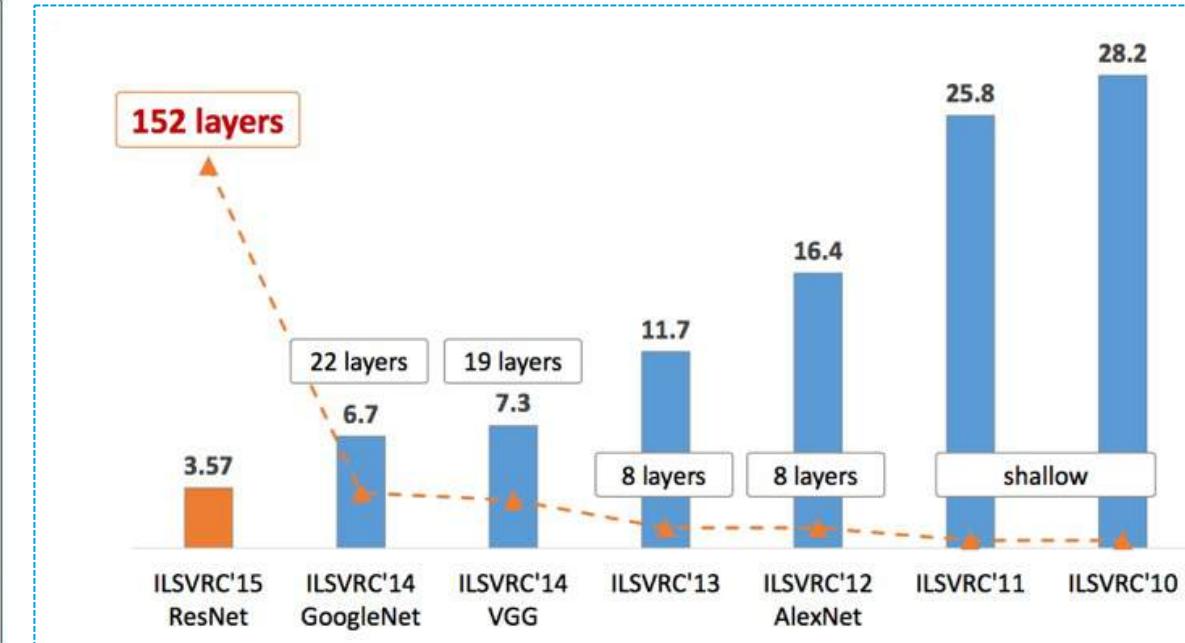
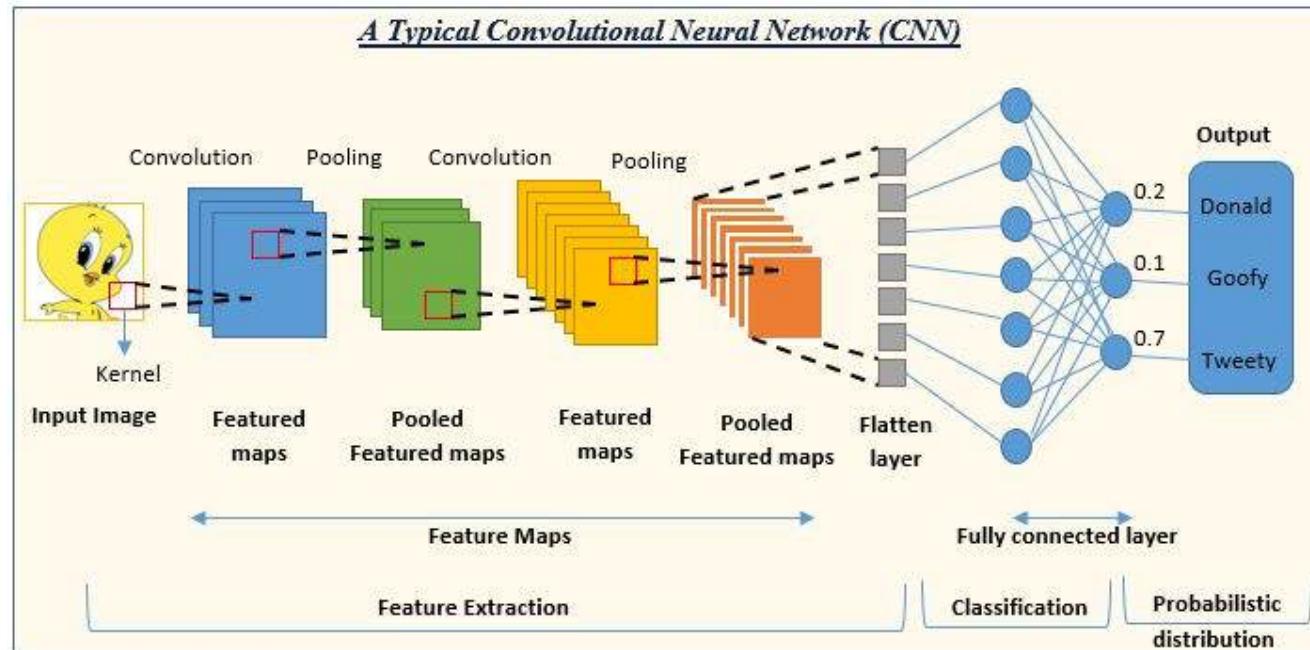
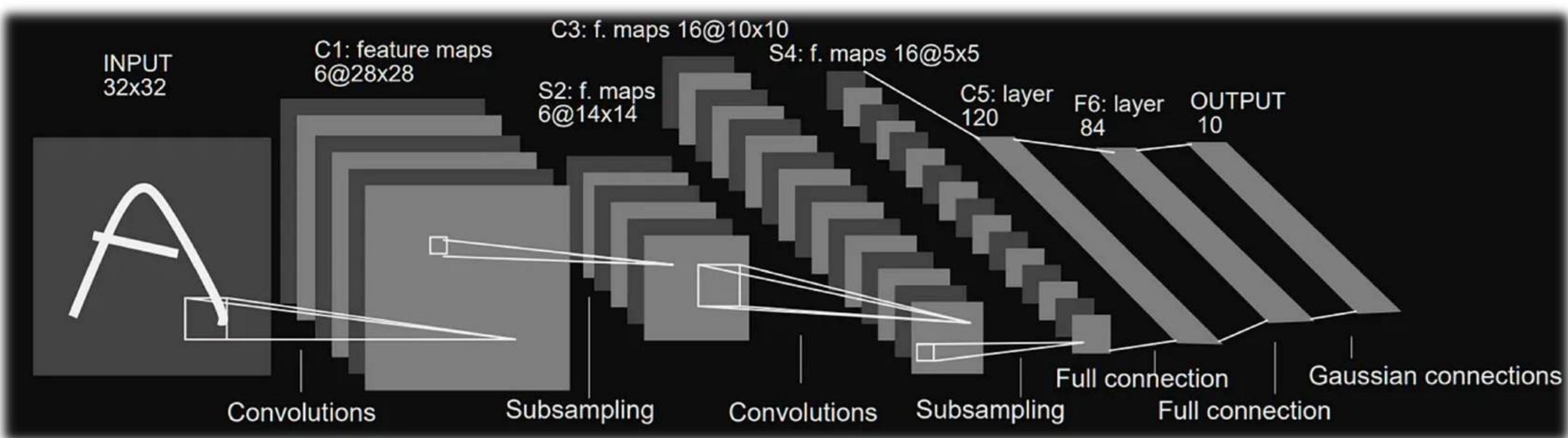


Image Classifier (IF)

LeNet for Image Classification

'Hello World' in the domain of Convolution Neural Networks

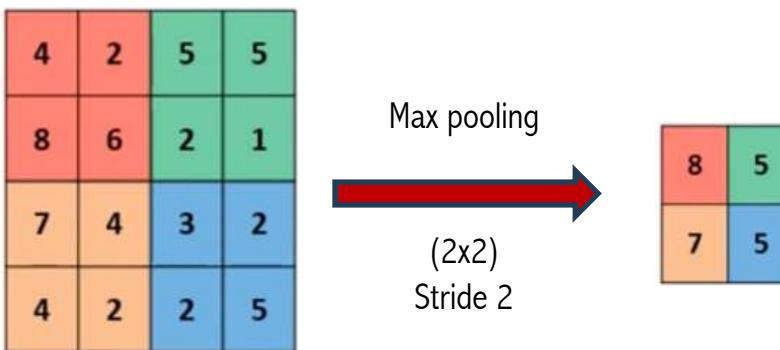
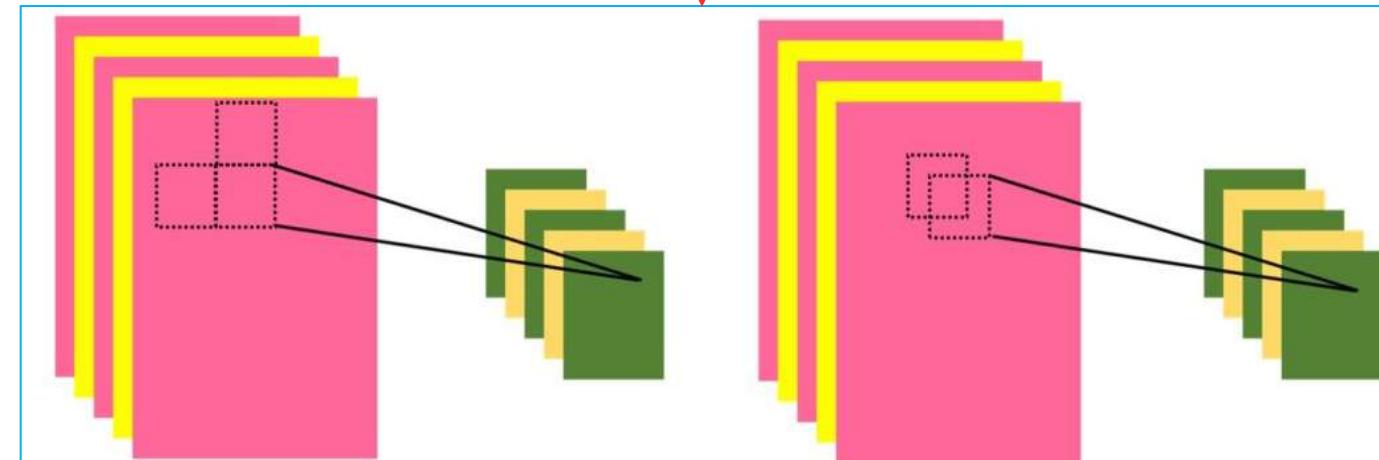
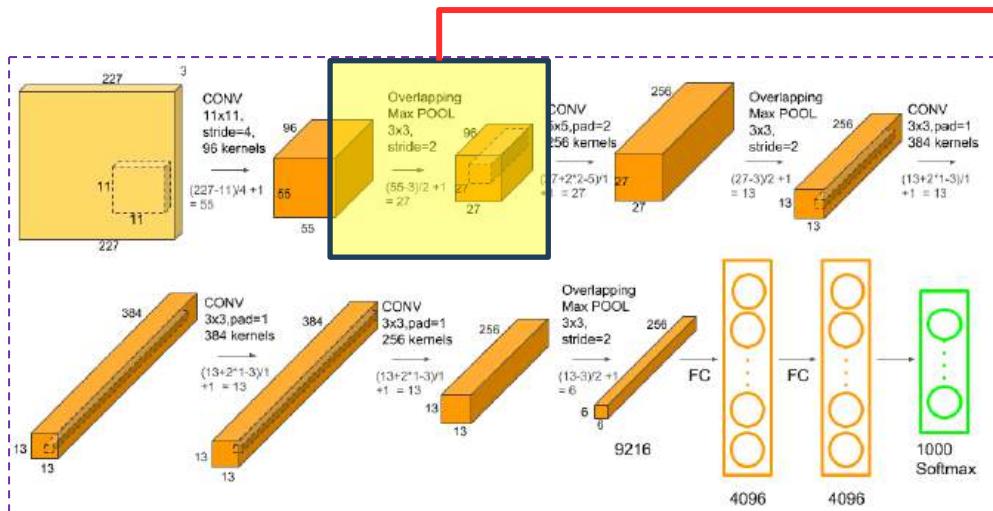


handwritten and machine-printed character recognition

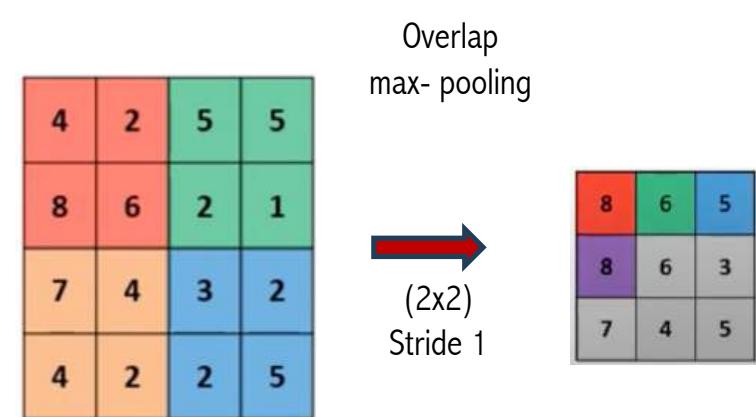
It was introduced by Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner in 1998

AlexNet for Image Classification

New: Overlapping Max Pooling

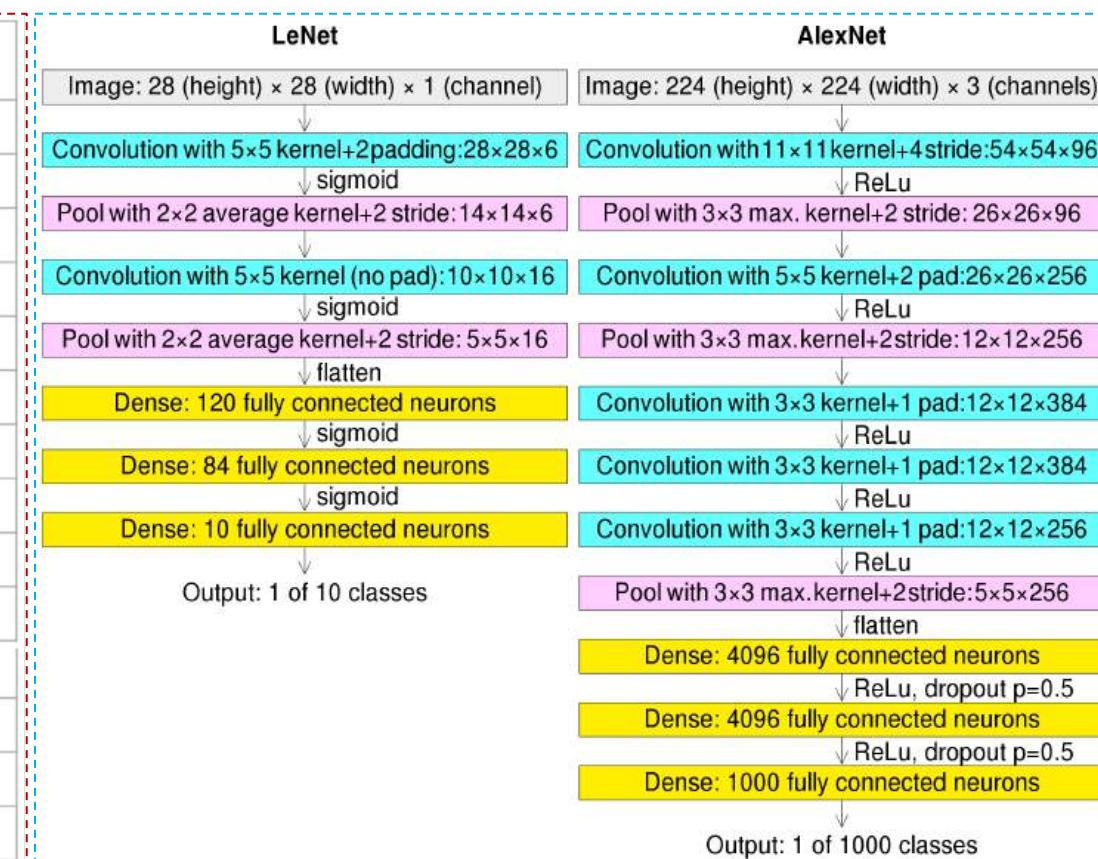


This overlapping nature of pooling helped reduce the **top-1 error rate by 0.4%** and **top-5 error rate by 0.3%** respectively when compared to using non-overlapping pooling windows of size 2×2 with a stride of 2 that would give same output dimensions.

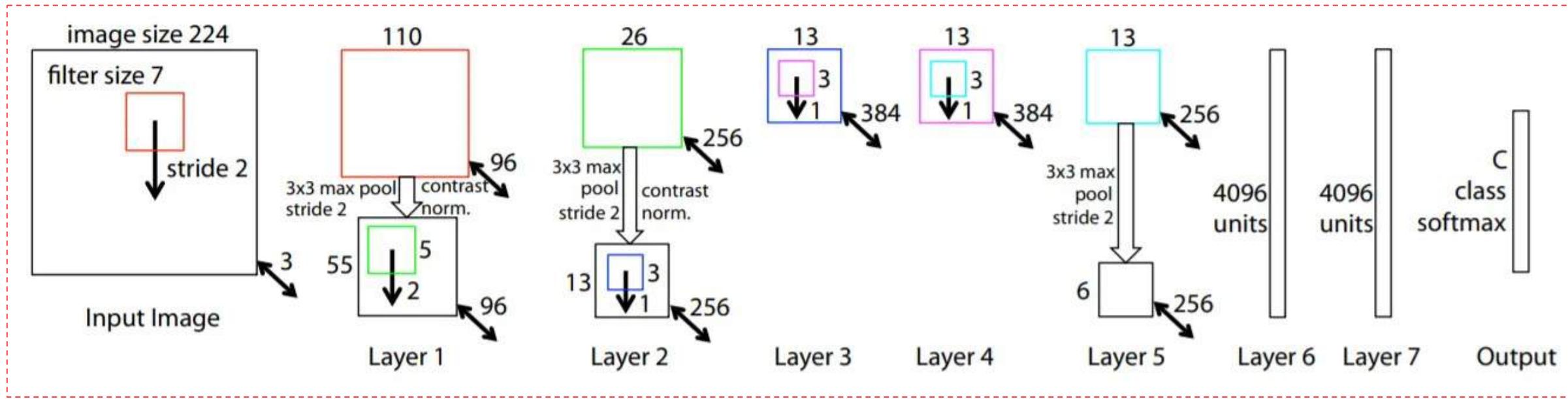


AlexNet Summary

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax



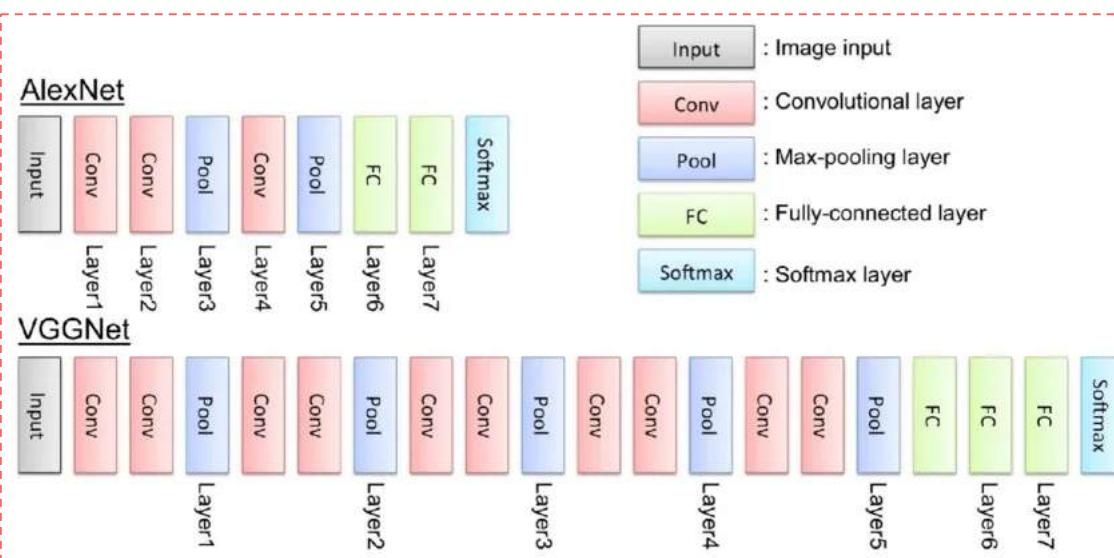
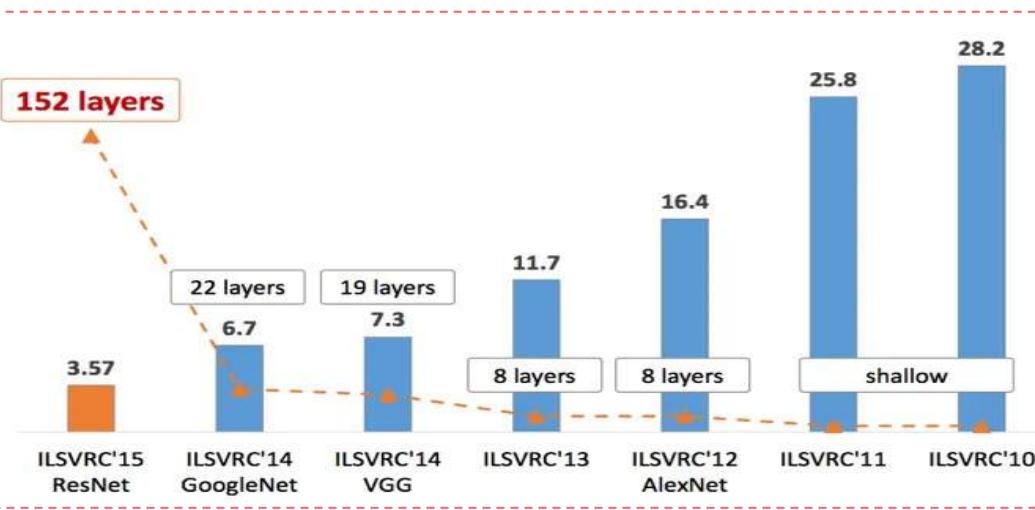
ZFNet for Image Classification



By visualizing the convolutional network, ZFNet became the Winner of ILSVLC 2013 in image classification by fine-tuning the AlexNet invented in 2012

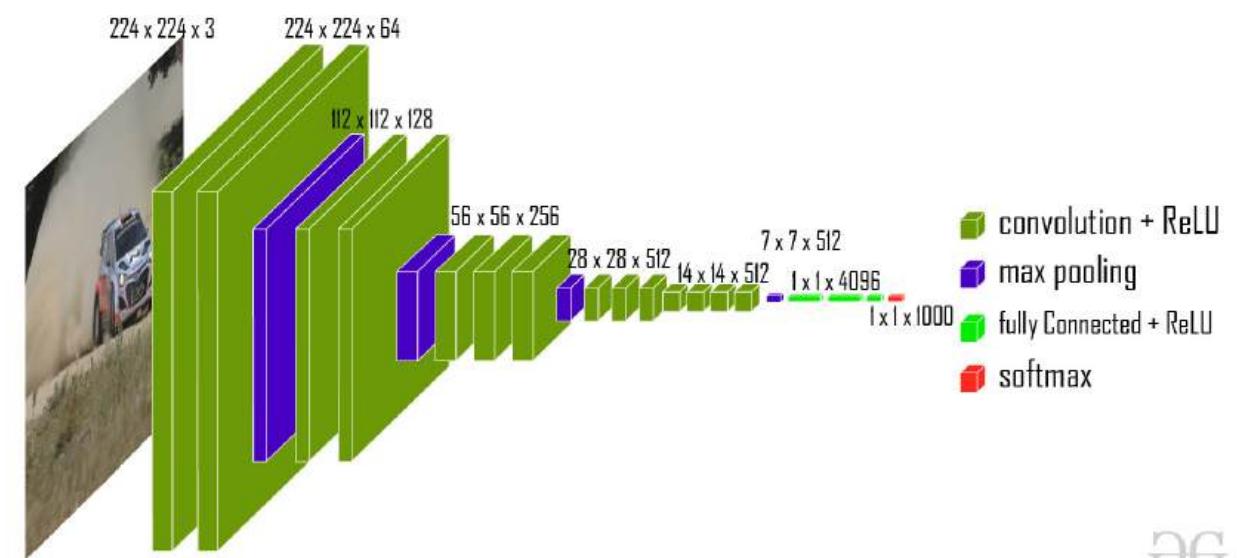
ZFNet was invented by Rob Fergus and Matthew D. Zeiler

VGGNet for Image Classification



The full name of VGG is the Visual Geometry Group, which belongs to the Department of Science and Engineering of Oxford University.

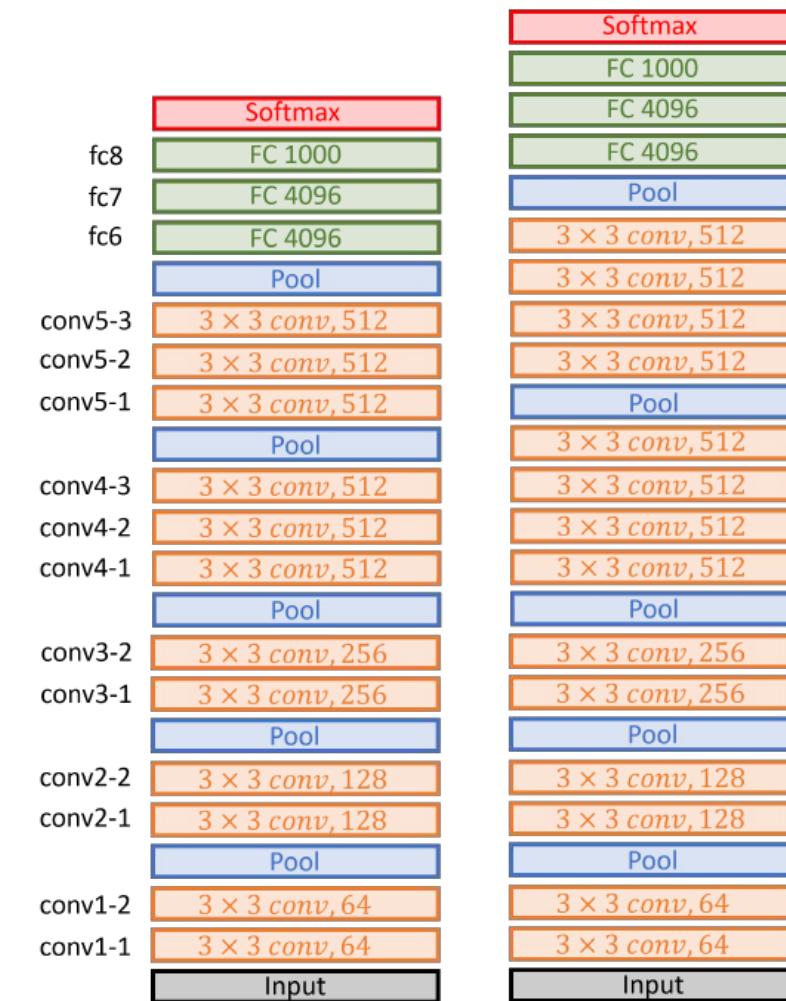
The original purpose of VGG's research on the depth of convolutional networks is to understand how the depth of convolutional networks affects the accuracy and accuracy of large-scale image classification and recognition.





VGG16 for Image Classification

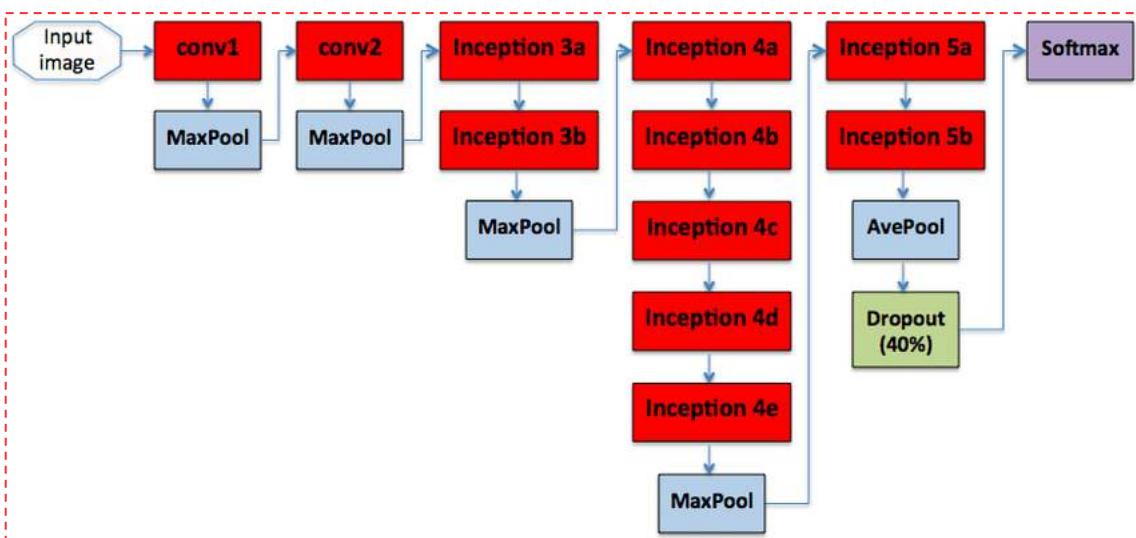
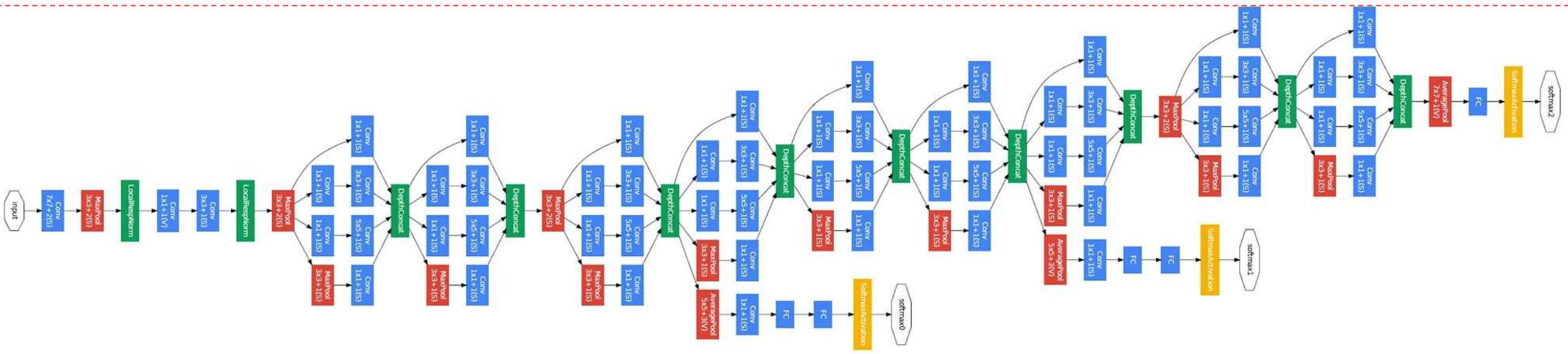
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax



VGG16

VGG19

GoogleLeNet for Image Classification

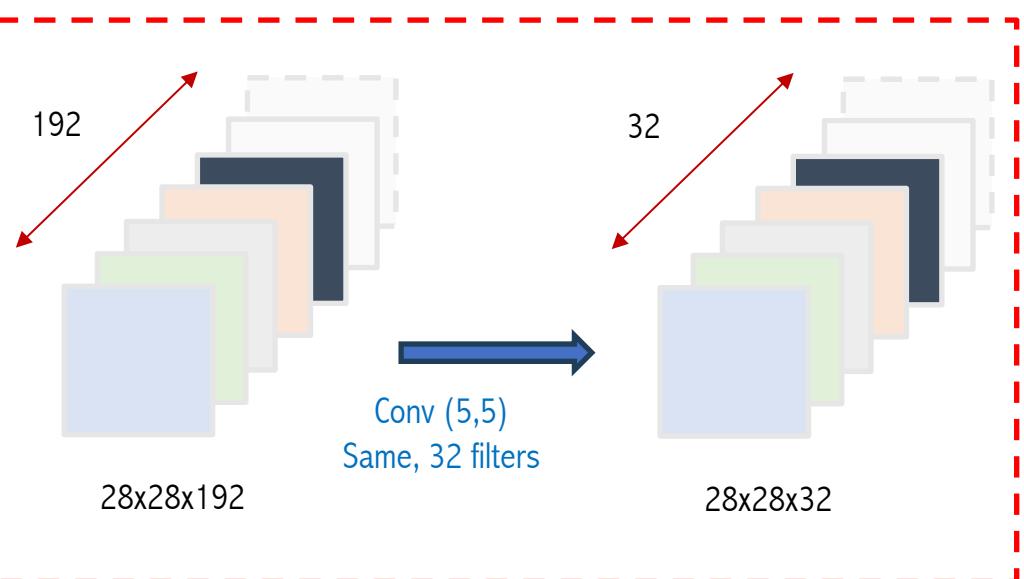


There are a total of 22 layers, It is already a very deep model compared with previous AlexNet, ZFNet, and VGGNet

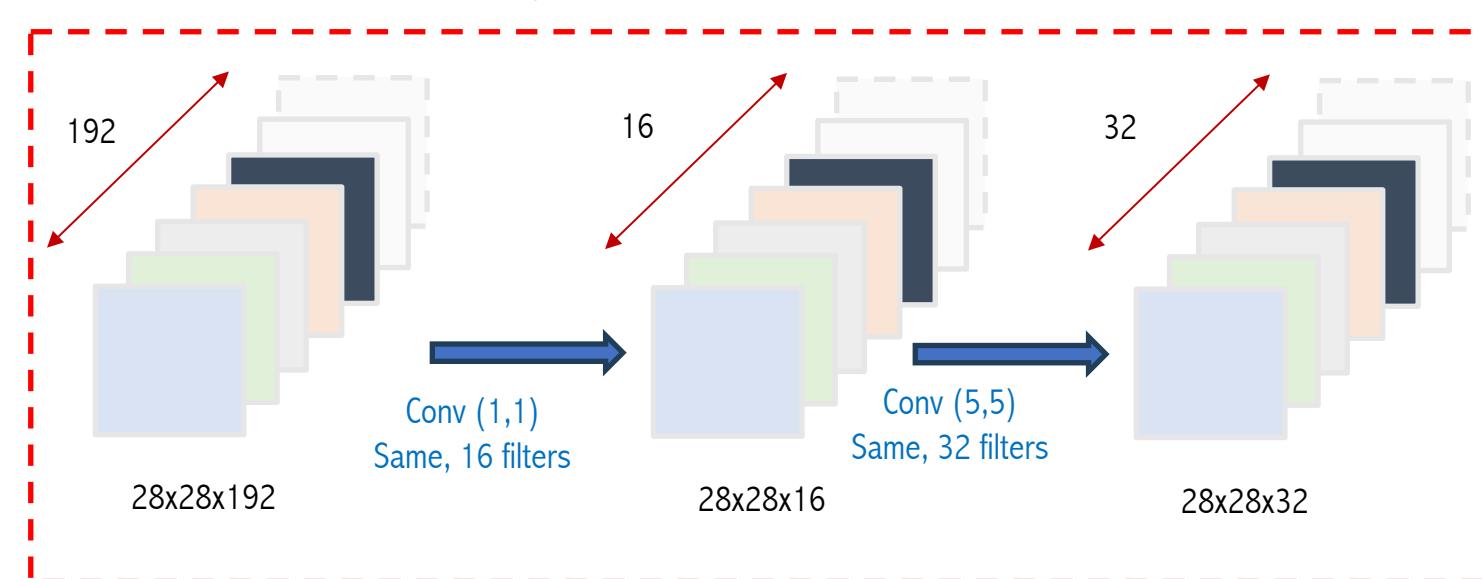
GoogleLetNet for Image Classification

New: 1×1 convolution is used as a dimension reduction module to reduce the computation

5×5 convolution without using 1×1 bottleneck convolution



5×5 convolution using 1×1 bottleneck convolution



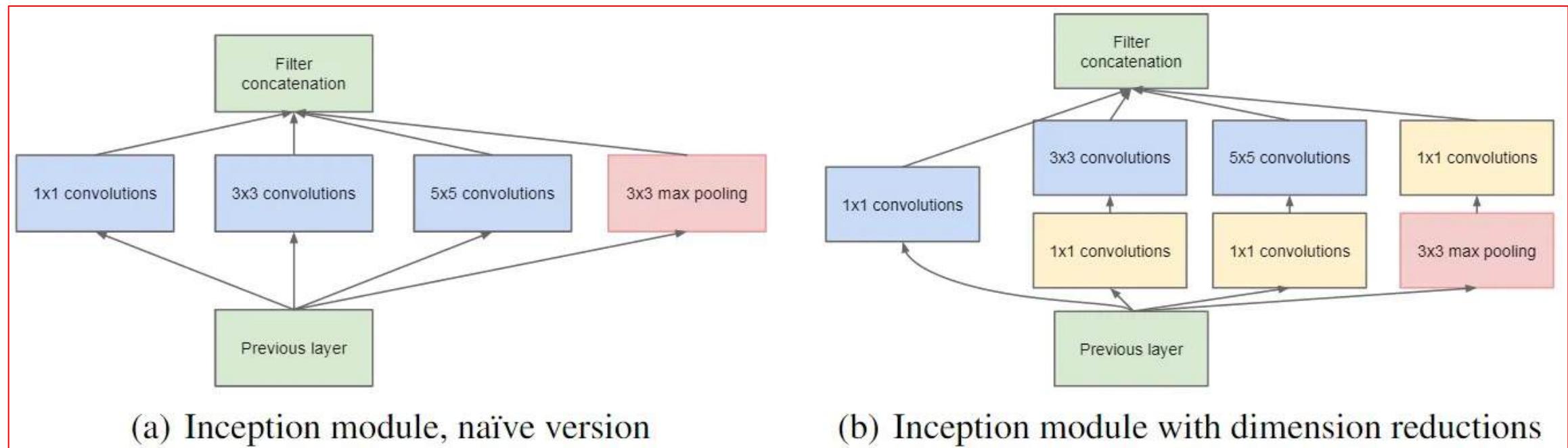
Number of parameters = $28 \times 28 \text{ (size of the input image)} \times 5 \times 5 \text{ (size of filter)} \times 192 \text{ (channels)} \times 32 \text{ (no of filters)}$
 $= 120,422,400$ operations

Number of parameters = $28 \times 28 \times 16 \times 192 + 28 \times 28 \times 32 \times 5 \times 5 \times 16 \sim 12.4M$

As the authors said that the idea of the name “Inception” comes from famous internet meme below: WE NEED TO GO DEEPER

GoogleLetNet for Image Classification

New: Inception Module



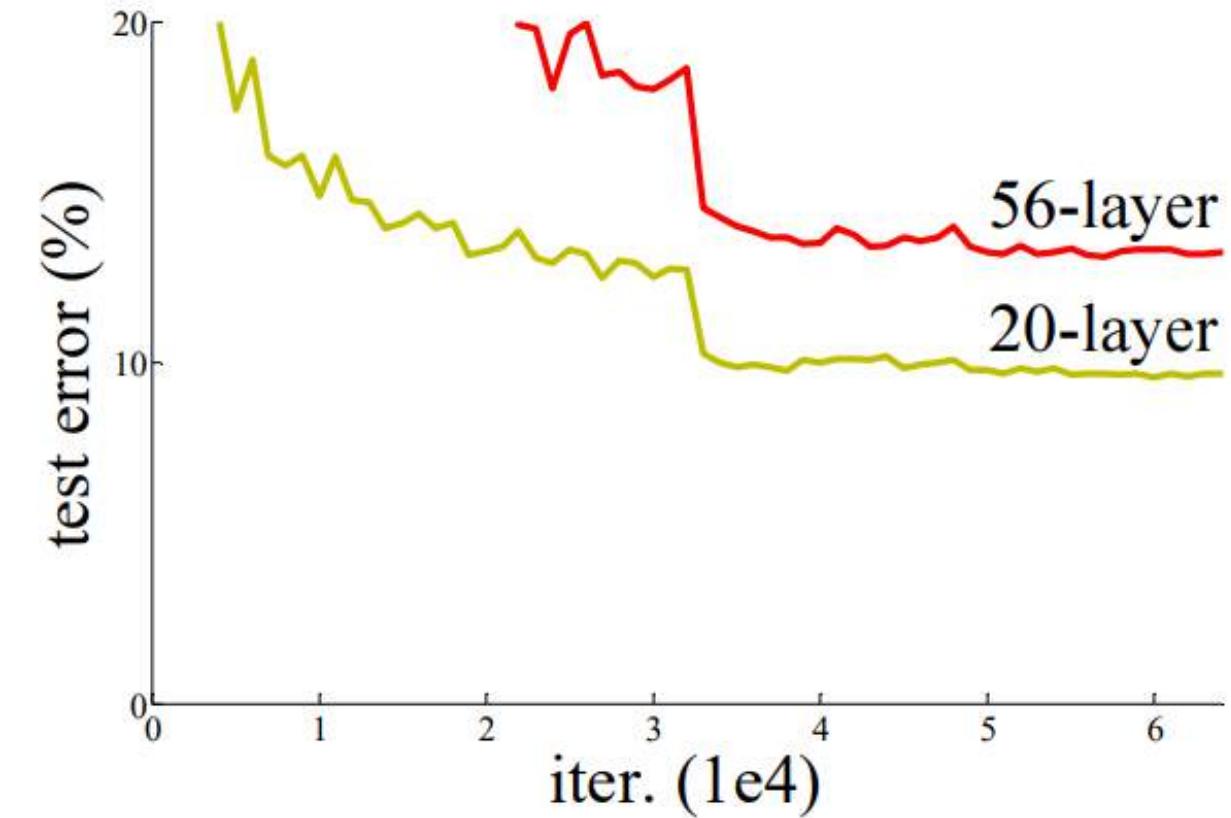
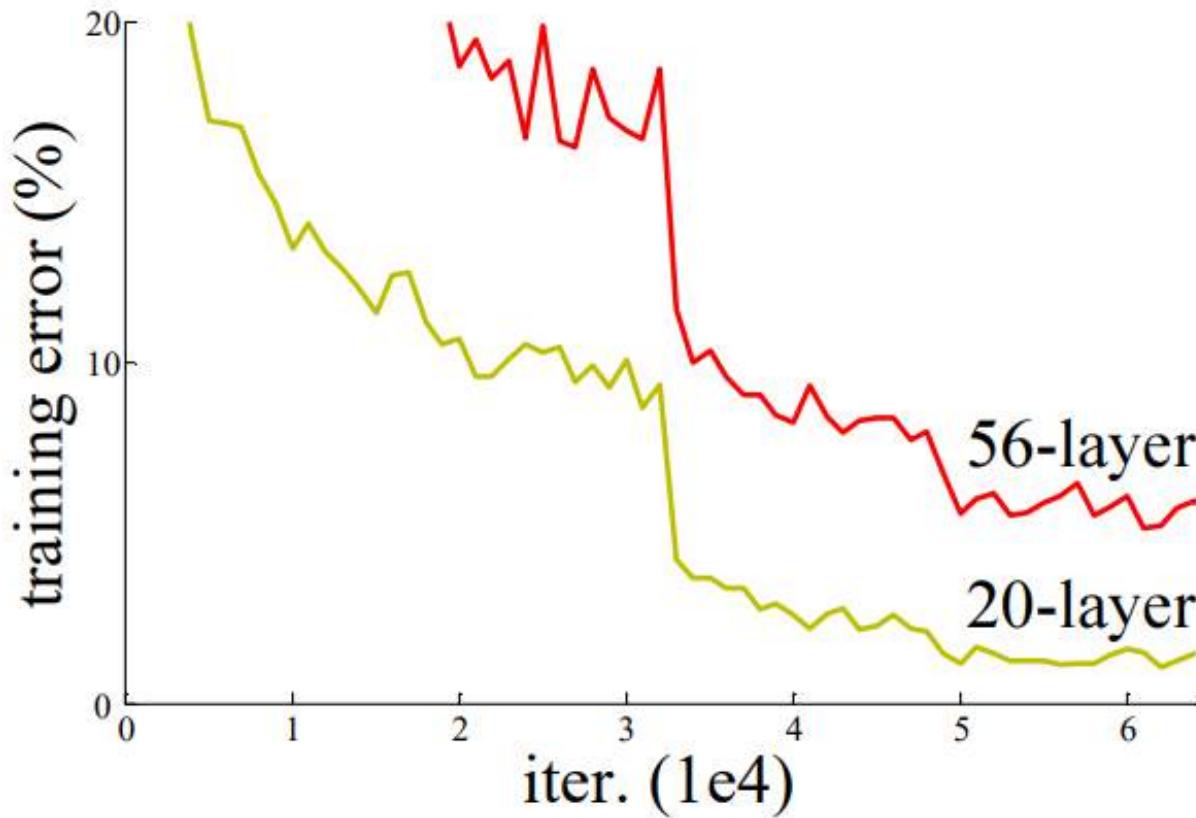
(a) Inception module, naïve version

Inception V1 Module

(b) Inception module with dimension reductions

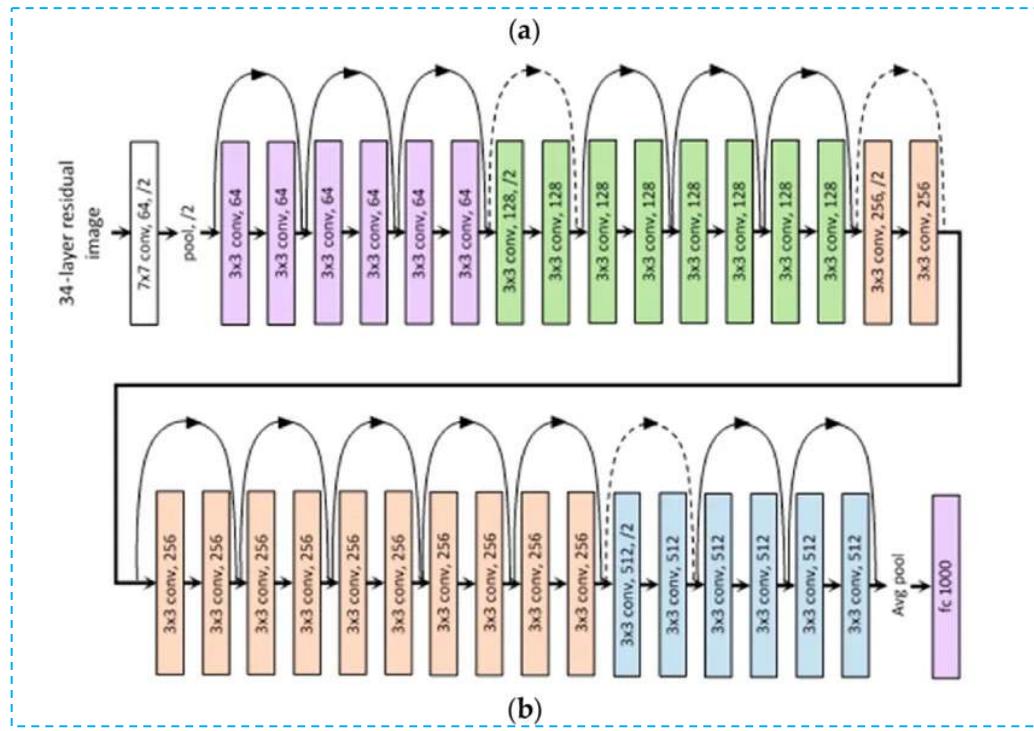
Inception V1 Optimized

ResNet for Image Classification

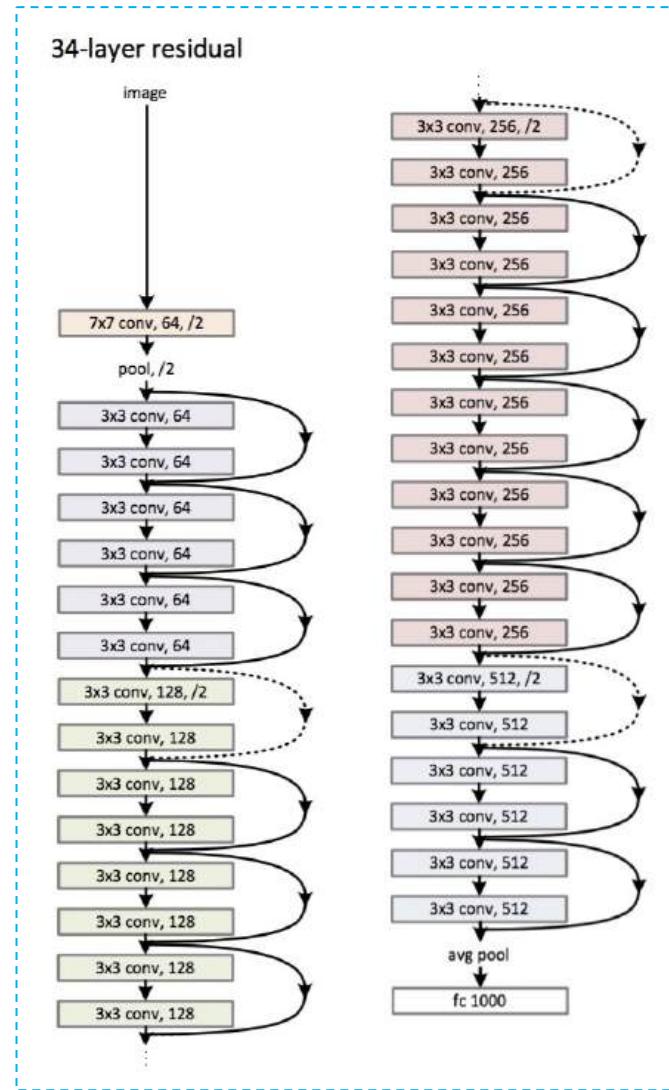


The Residual Blocks idea was created by this design to address the issue of the vanishing/exploding gradient

ResNet for Image Classification

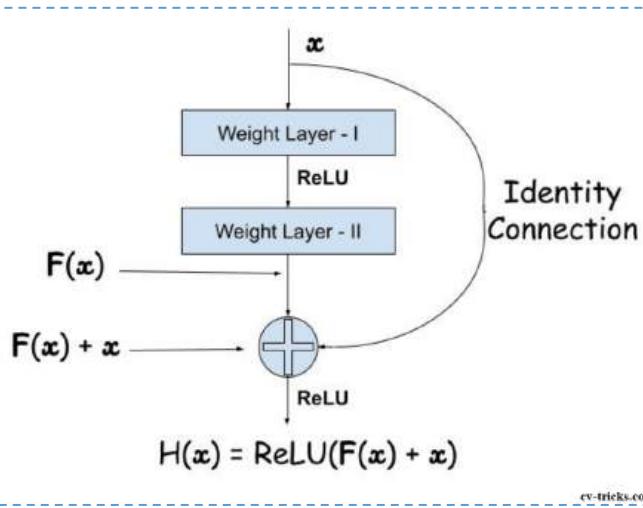


A novel architecture called **Residual Network** was launched by Microsoft Research experts in 2015 with the proposal of **ResNet**.

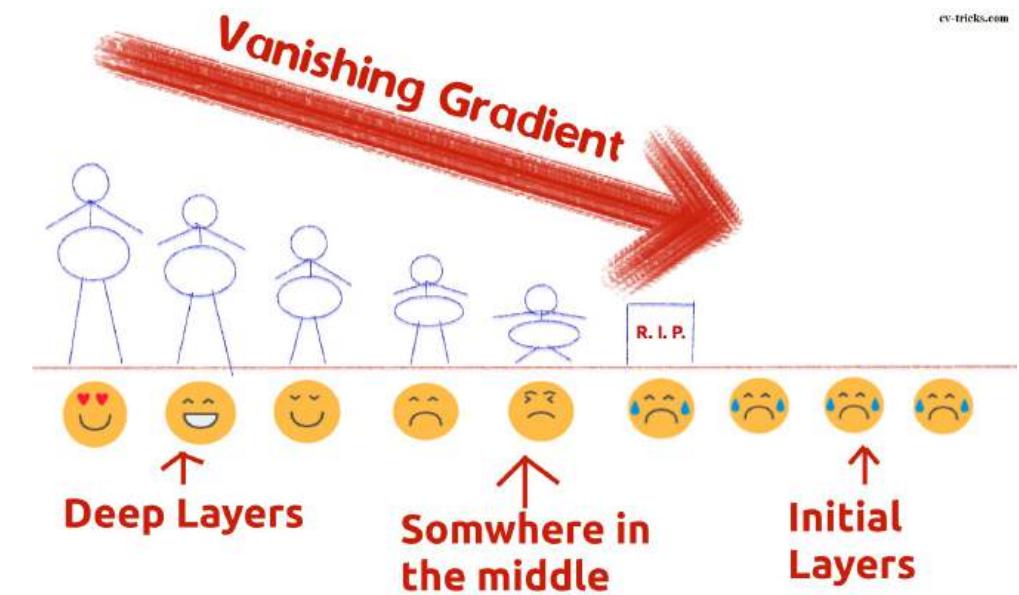
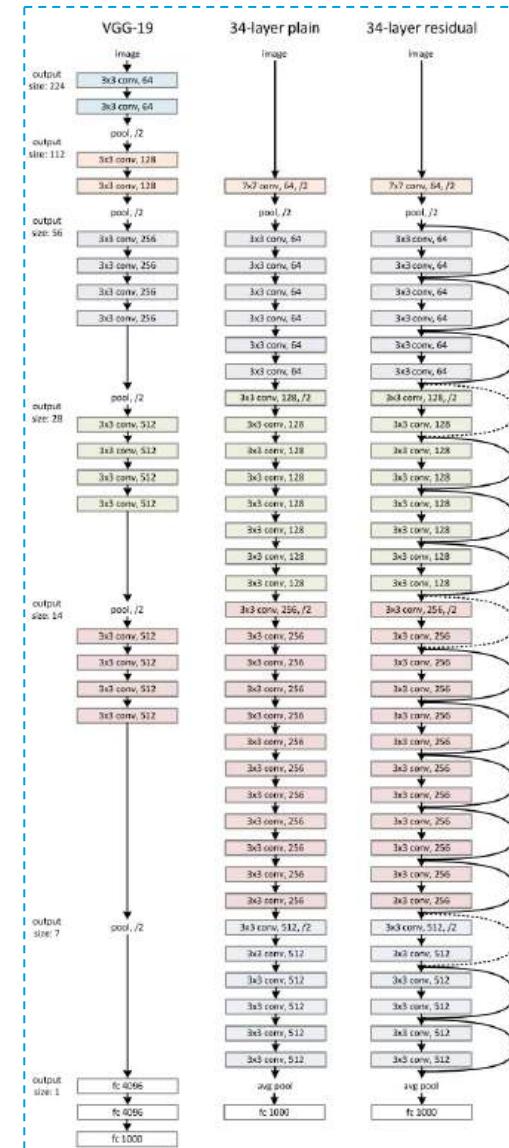


ResNet34 for Image Classification

New: Skip Connection



The intuition is that learning $f(x) = 0$ has to be easy for the network.



The VGG-19-inspired 34-layer plain network architecture used by ResNet is followed by the addition of the shortcut connection

Image Classification vs. Object Detection

Classification



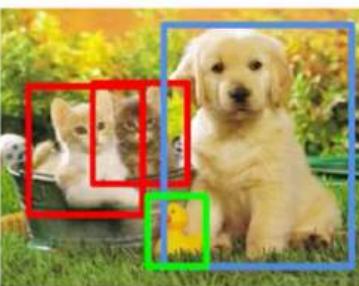
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Classification

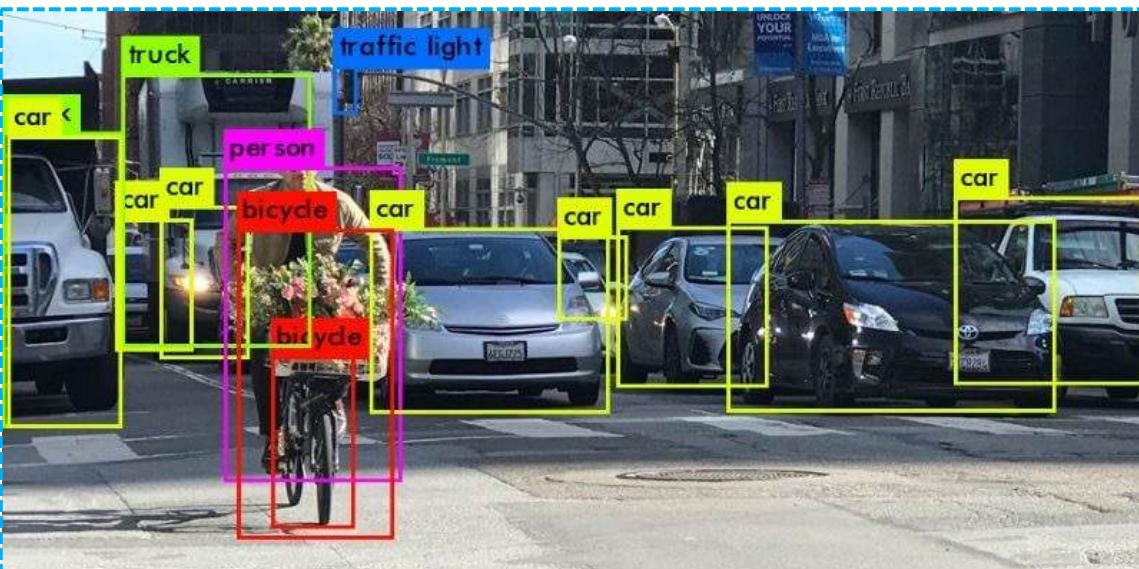


Cat

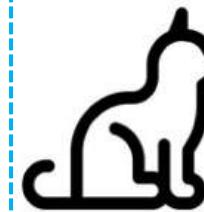
Detection



Cat, Dog, Dog

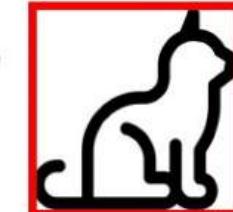


Classification



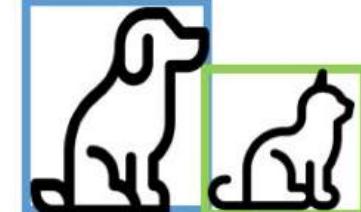
CAT

Classification + Localization



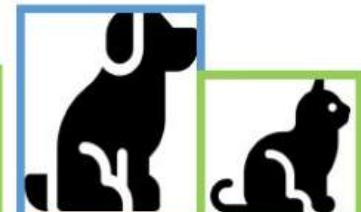
CAT

Object Detection



DOG , CAT

Instance Segmentation



DOG , CAT

Single Object

Multiple Object

Outline

Introduction to Object Detection

Applications of Object Detection

Object Detection Milestones

Traditional Object Detectors

CNNs for Image Classification

Image Classifier to Object Detectors with CNN

CNN Limitations and Spatial Outputs

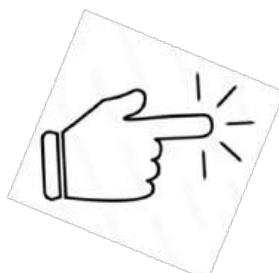
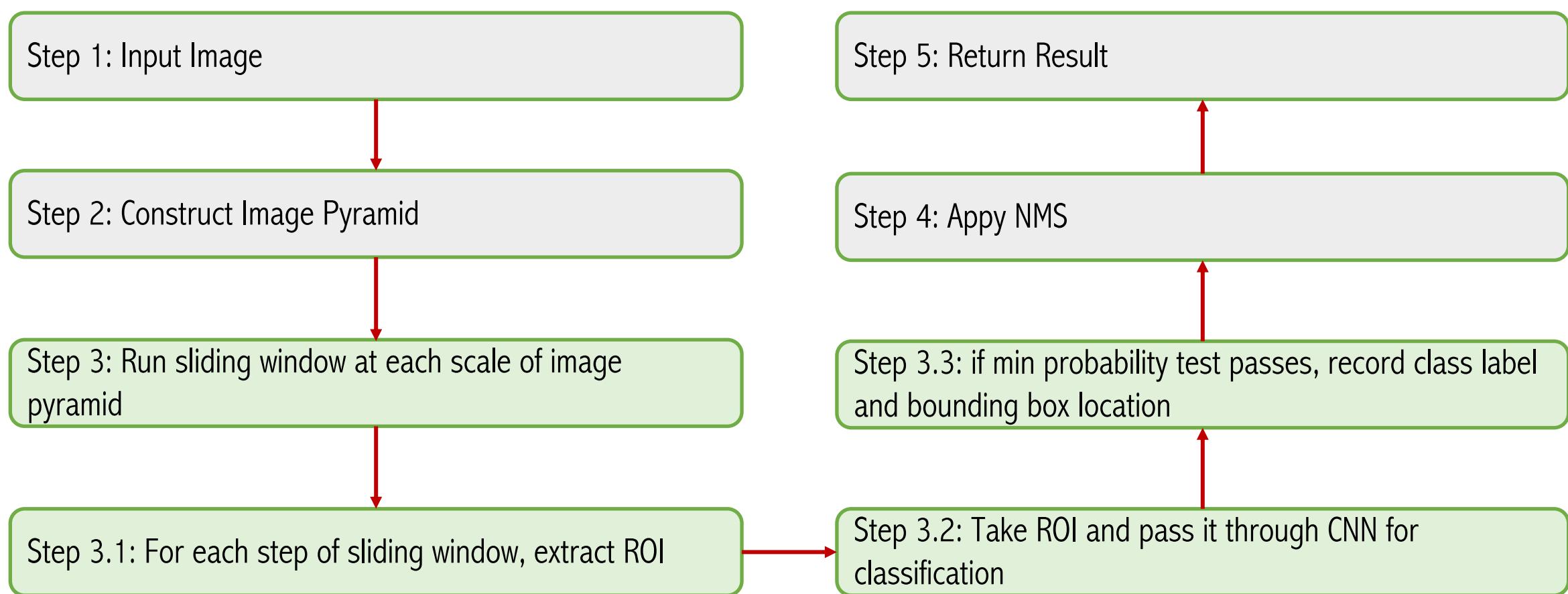


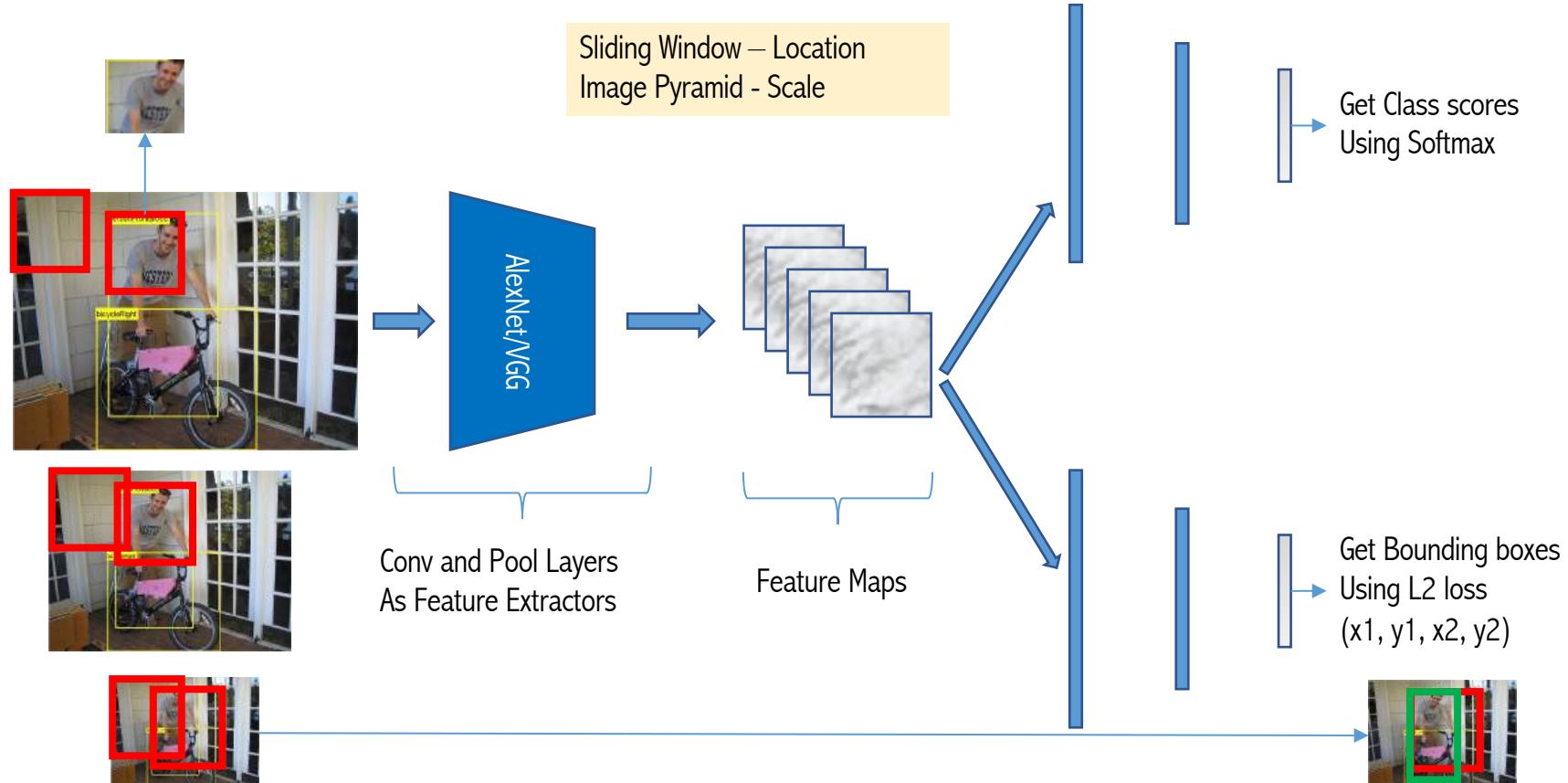
Image Classification To Object Detection



Code: <https://colab.research.google.com/drive/1Et xlG4XRgrsOF7N8ivHQ6JZvVRboQx-8?usp=sharing>

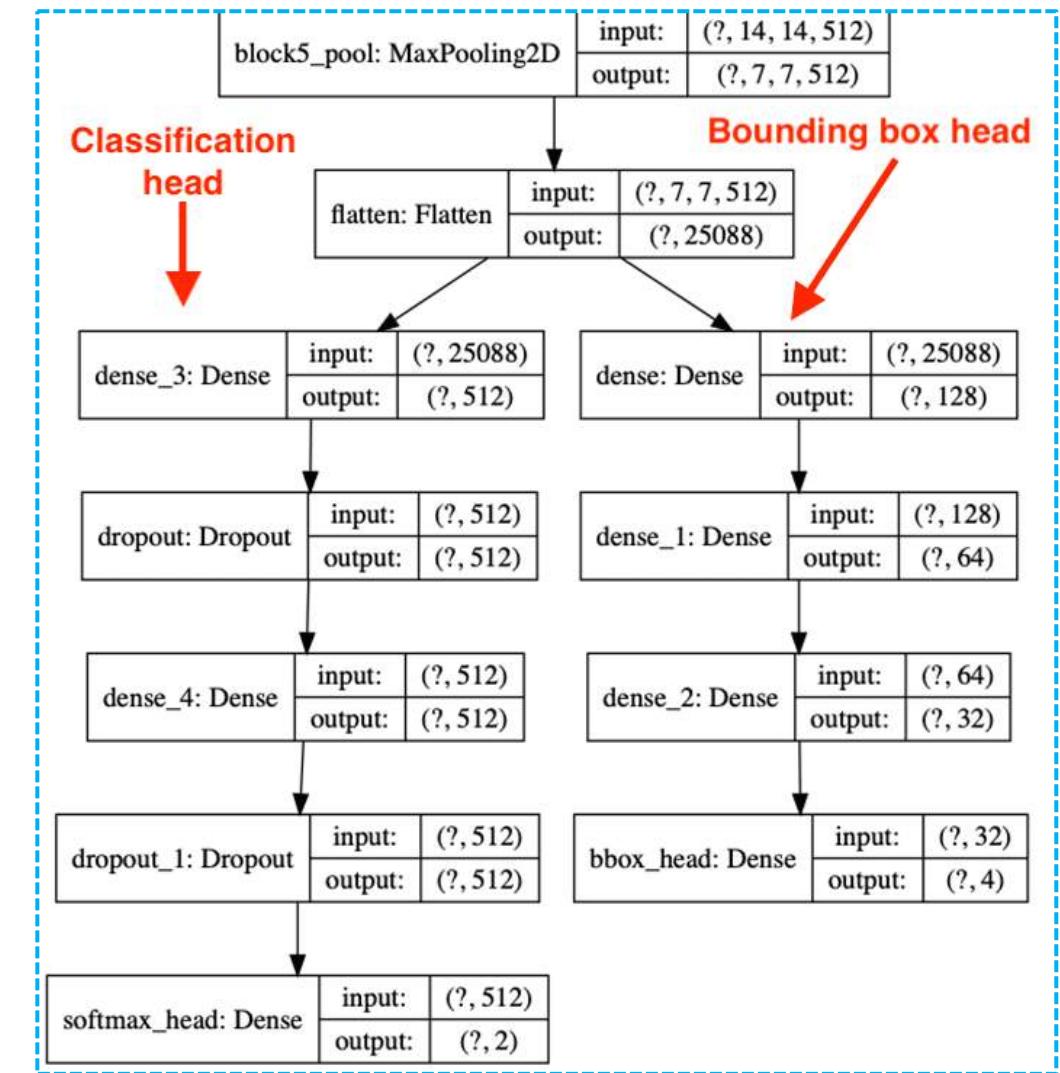
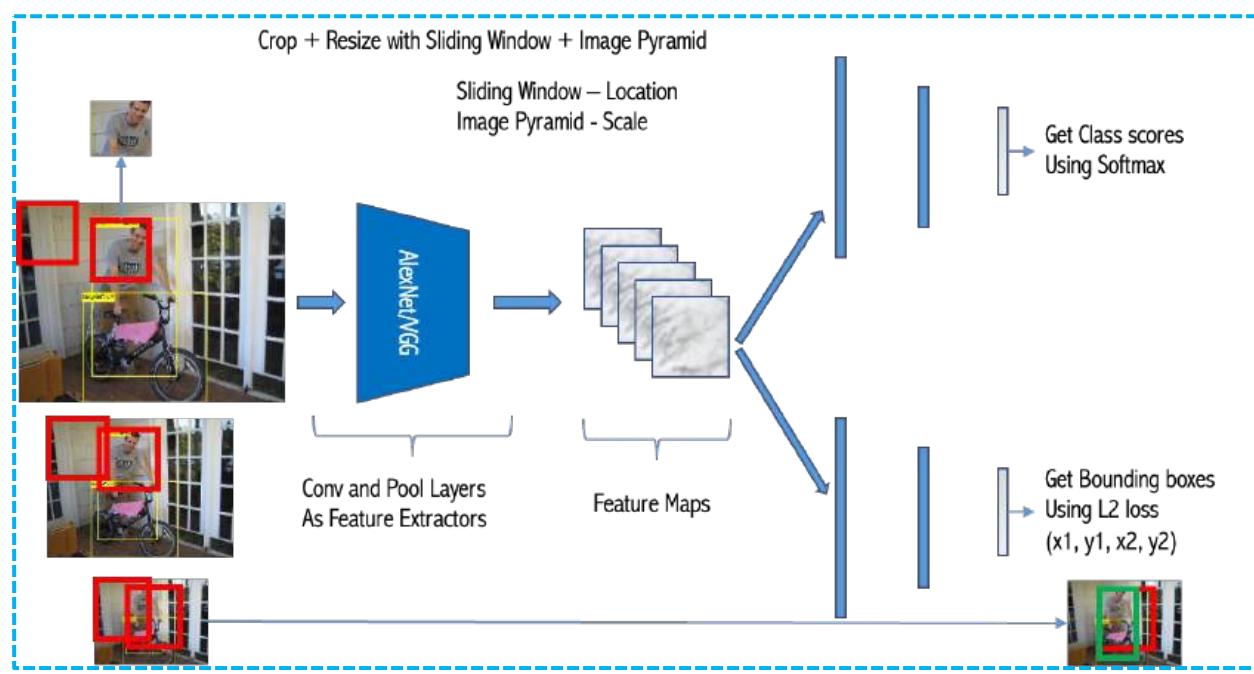
Ideas for Detection using CNN

Crop + Resize with Sliding Window + Image Pyramid



For example, to process an image of 800x800, if the sliding window size is 224, we will end up with 331,776 crops.

Ideas for Detection using CNN



Outline

Introduction to Object Detection

Applications of Object Detection

Object Detection Milestones

Traditional Object Detectors

CNNs for Image Classification

Image Classifier to Object Detectors with CNN

CNN Limitations and Spatial Outputs

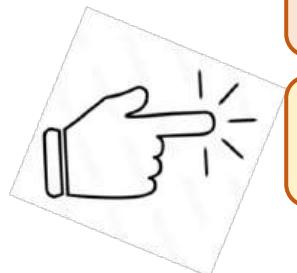
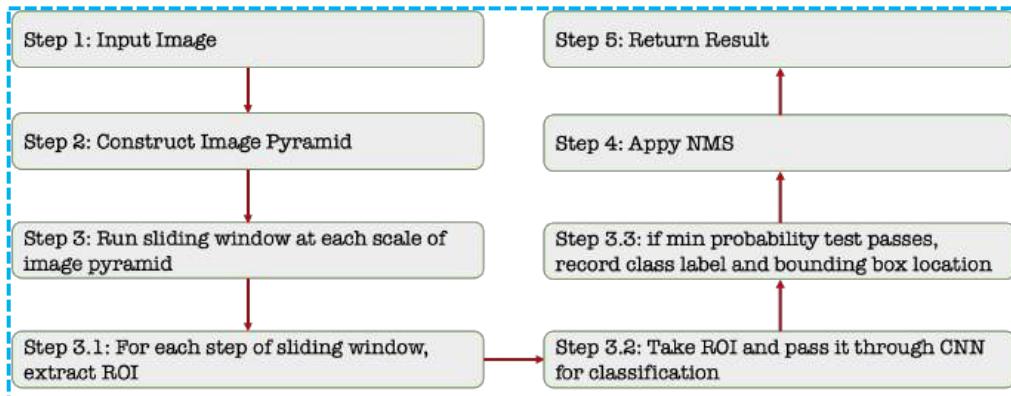
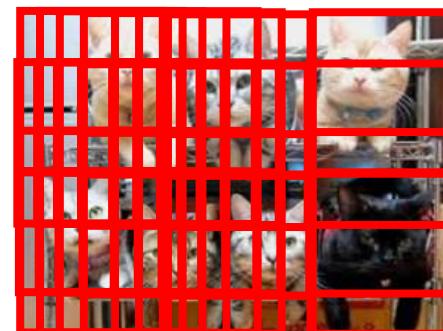
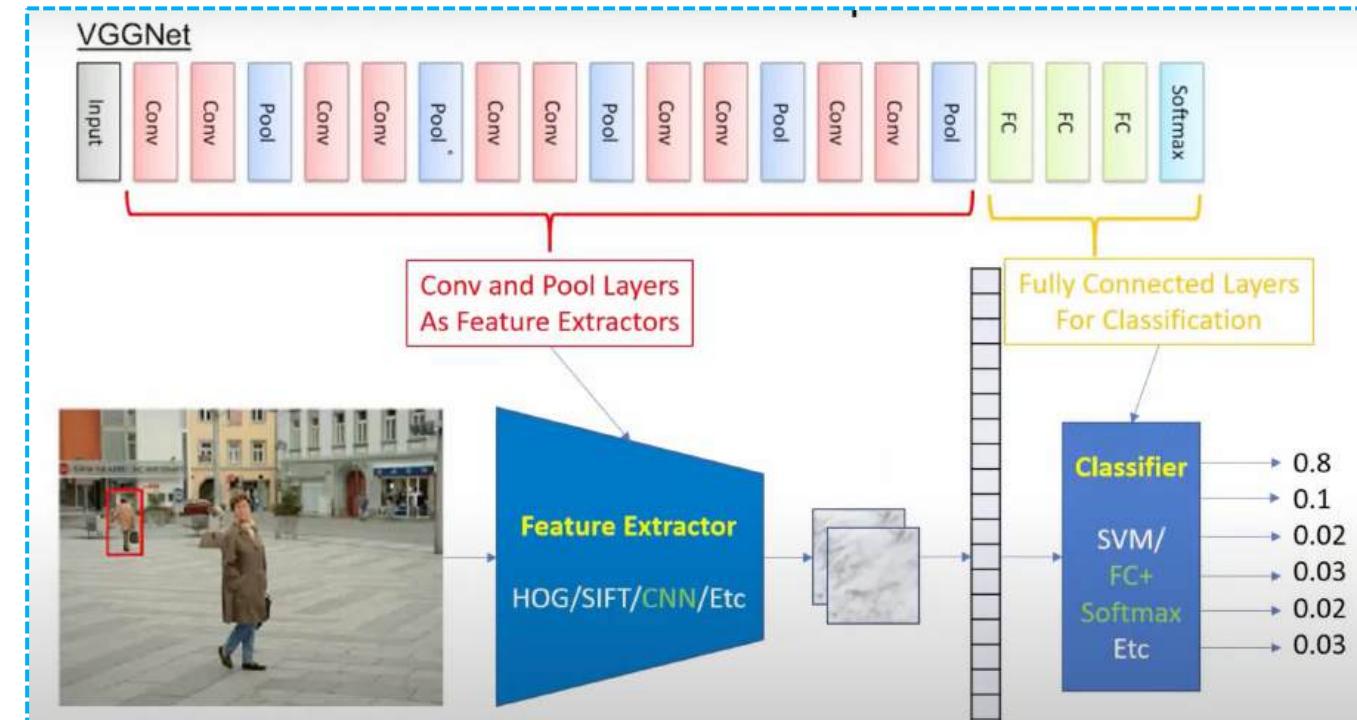


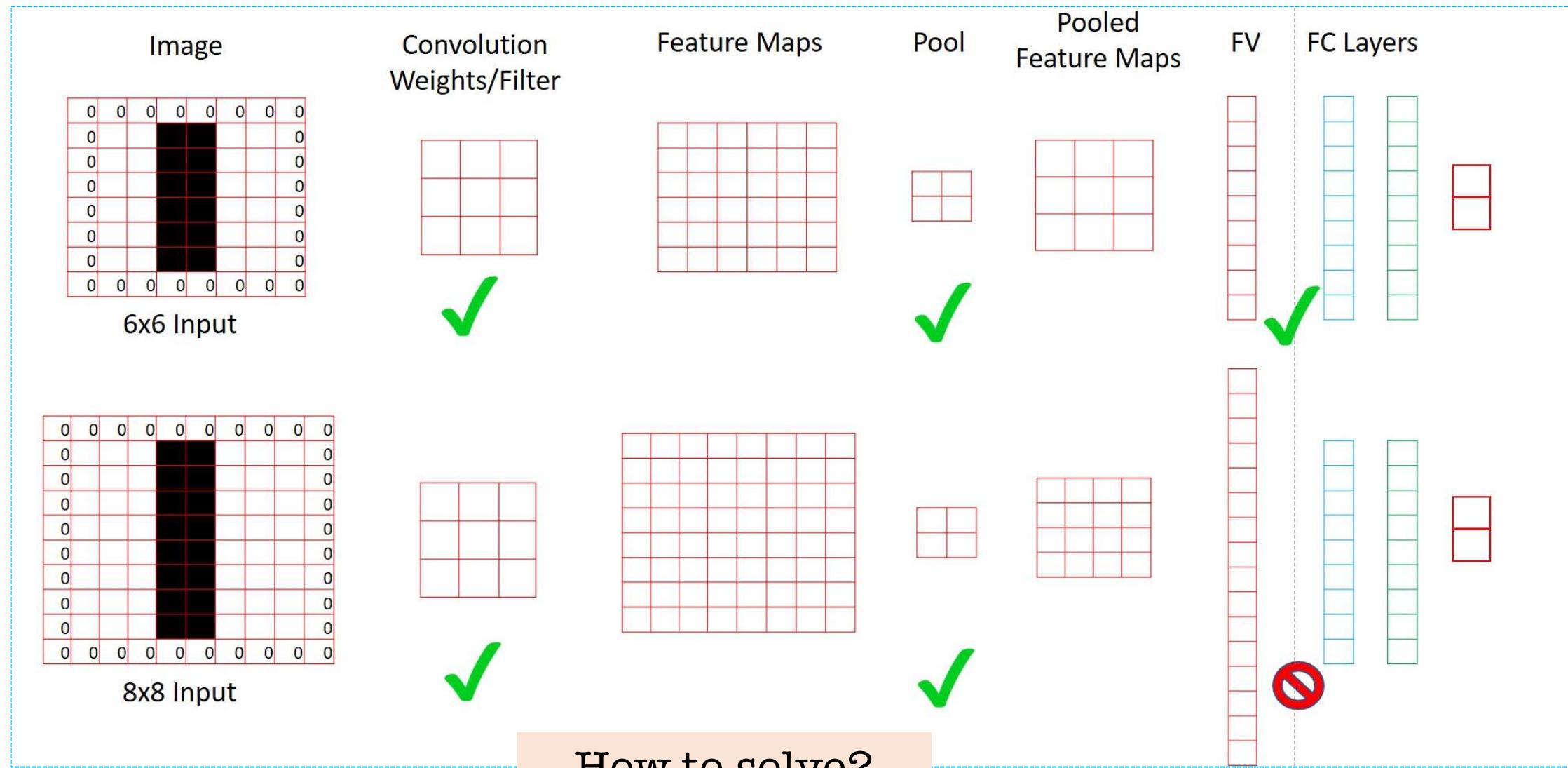
Image Classifier To Object Detector



LIMITATION

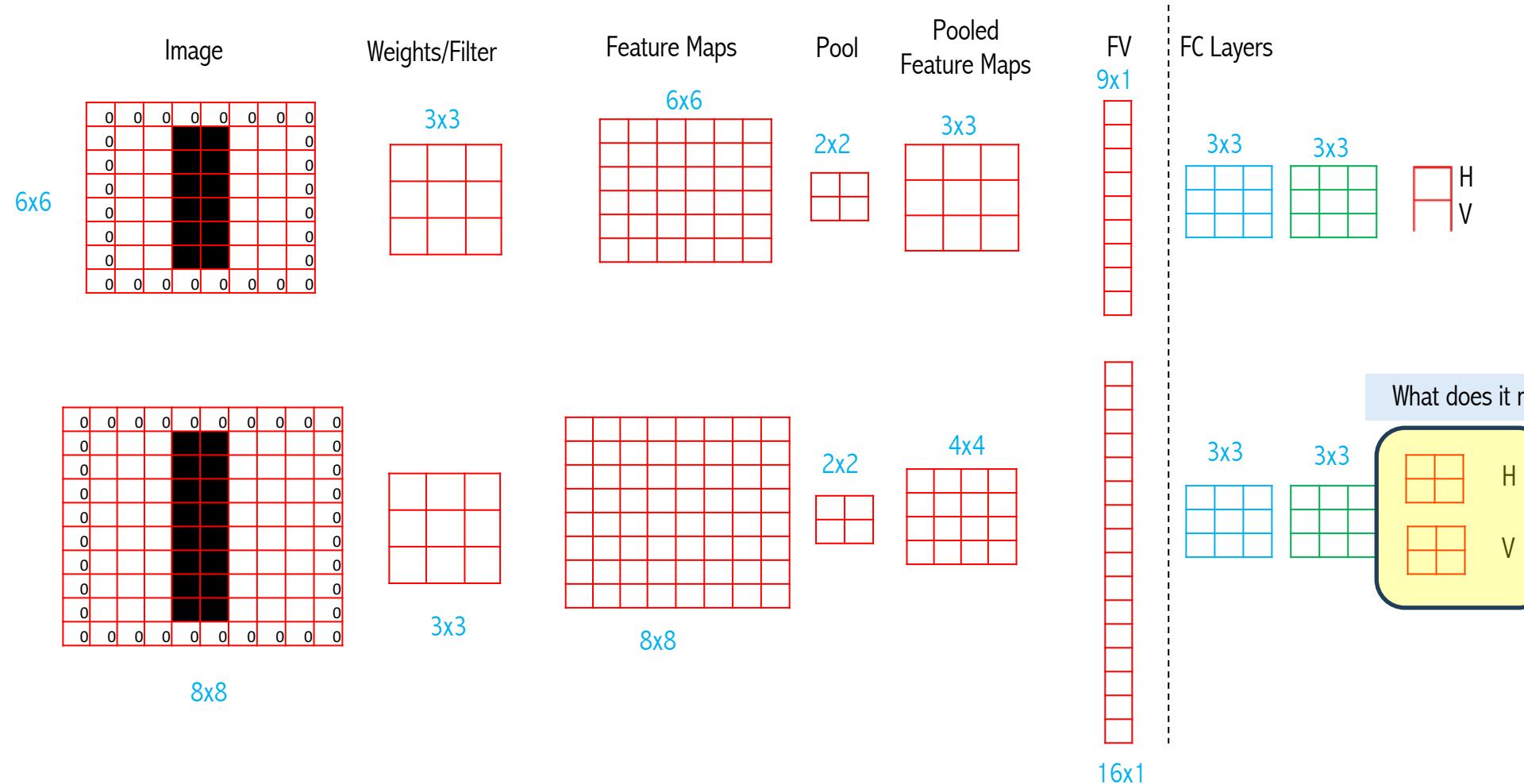


Problems: CNN Input Size Constraints



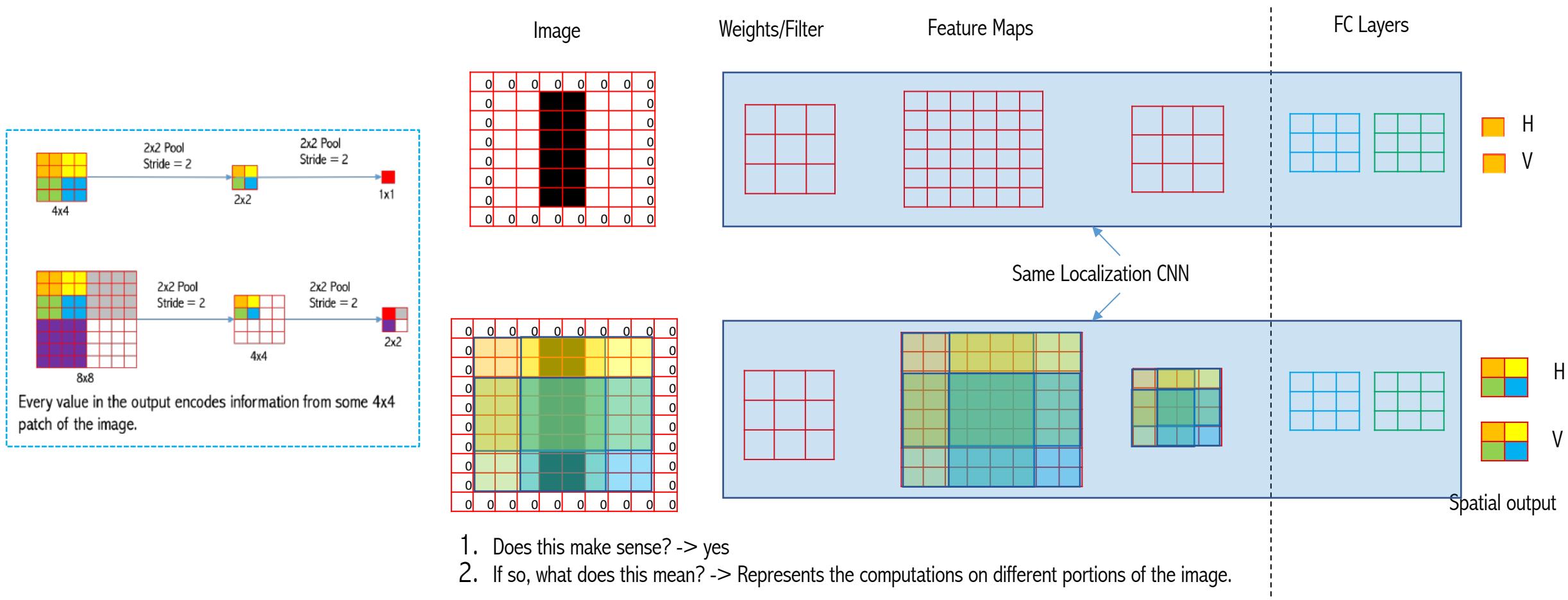
ConvNets input size constraints

❖ FC as Conv



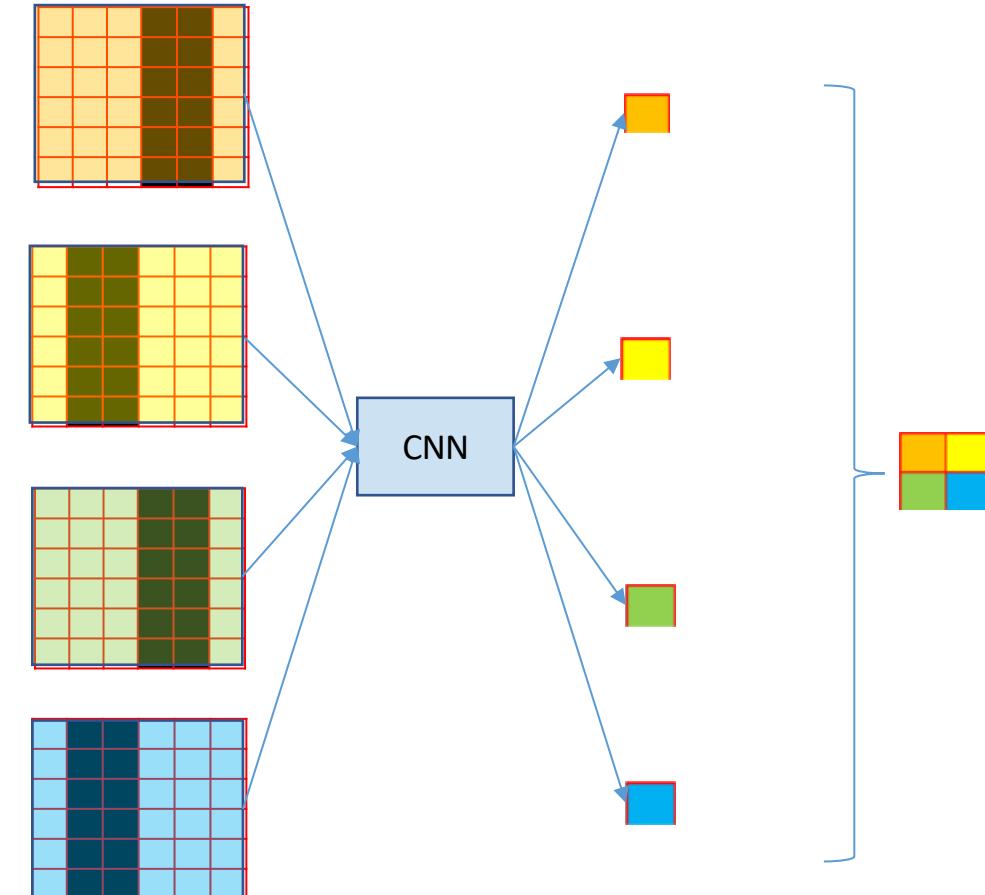
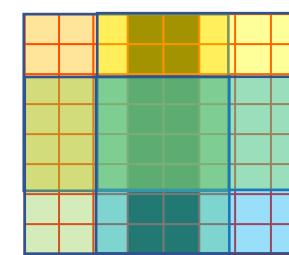
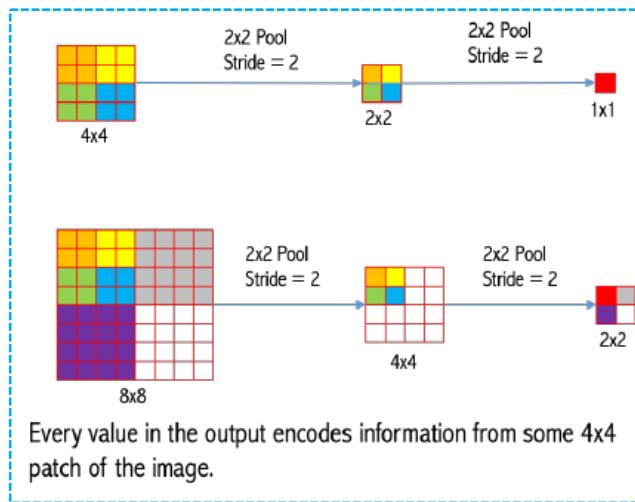
ConvNets input size constraints

❖ FC as Conv

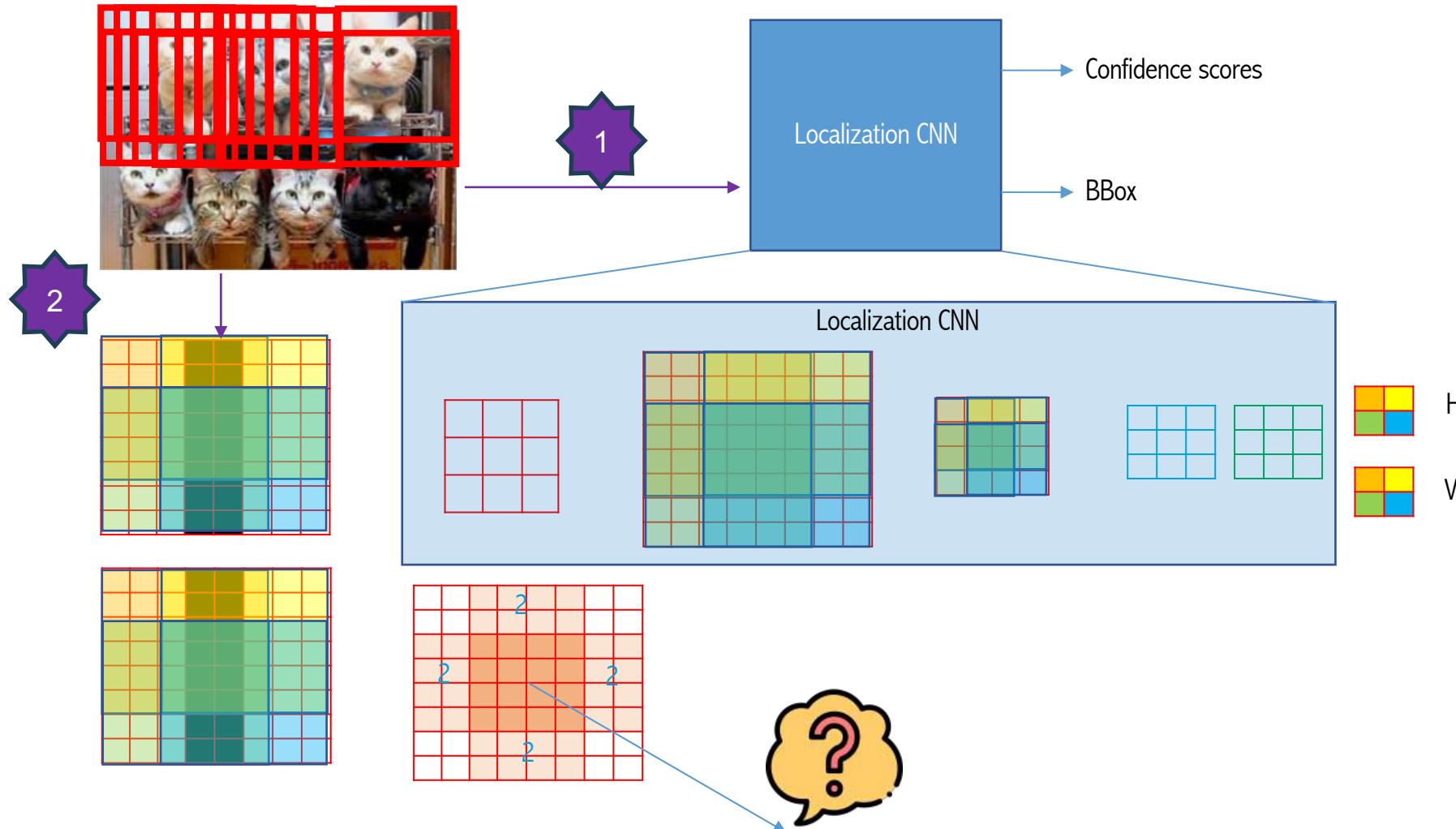


ConvNets input size constraints

❖ FC as Conv

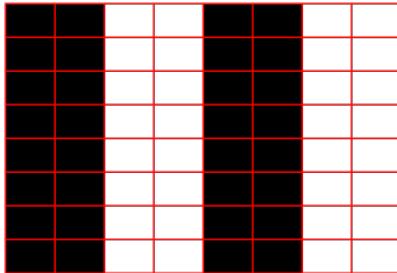


ConvNets and Sliding Window Efficiency





ConvNets and Sliding Window Efficiency



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

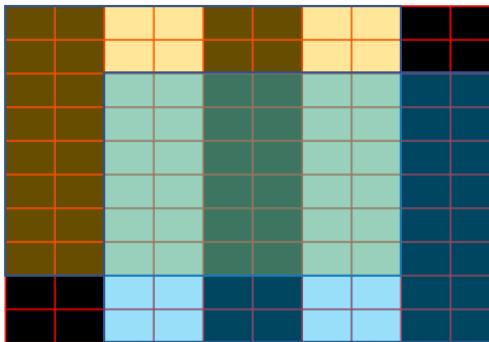
8x8

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

3x3

-765	-765	765	765	-765	-765
-765	-765	765	765	-765	-765
-765	-765	765	765	-765	-765
-765	-765	765	765	-765	-765
-765	-765	765	765	-765	-765
-765	-765	765	765	-765	-765

6x6



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

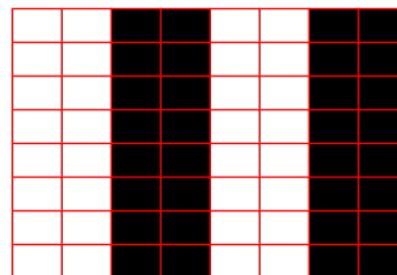
10x10

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

3x3

-765	-765	765	765	-765	-765	765	765
-765	-765	765	765	-765	-765	765	765
-765	-765	765	765	-765	-765	765	765
-765	-765	765	765	-765	-765	765	765
-765	-765	765	765	-765	-765	765	765
-765	-765	765	765	-765	-765	765	765
-765	-765	765	765	-765	-765	765	765
-765	-765	765	765	-765	-765	765	765

8x8



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

8x8

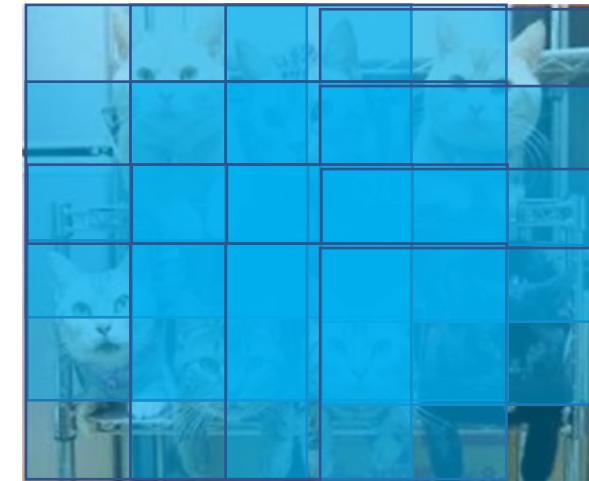
1	0	-1
1	0	-1
1	0	-1

3x3

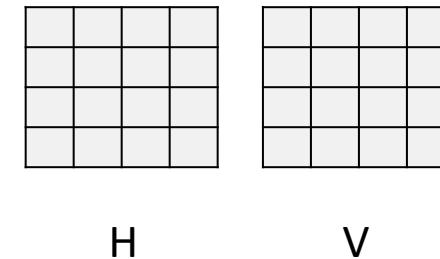
765	765	-765	-765	765	765
765	765	-765	-765	765	765
765	765	-765	-765	765	765
765	765	-765	-765	765	765
765	765	-765	-765	765	765

6x6

Spatial Output for Image Pyramids



H V



Spatial Outputs: Summary

With Spatial Outputs, we can detect different objects at different locations of the image. Below figure shows a 2x3 Spatial Output for a sample image.

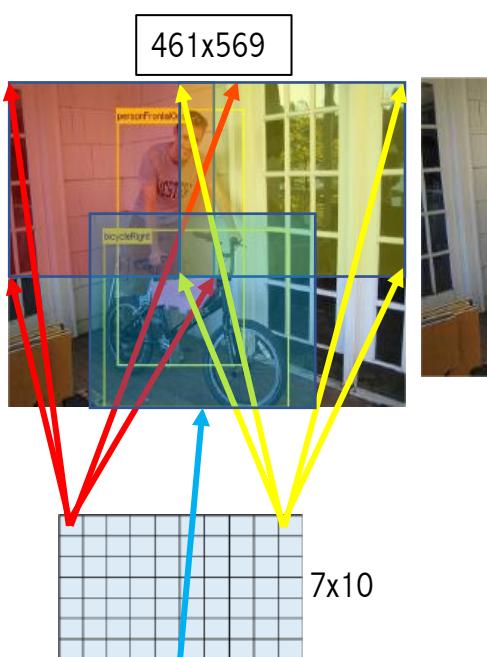


OverFeat

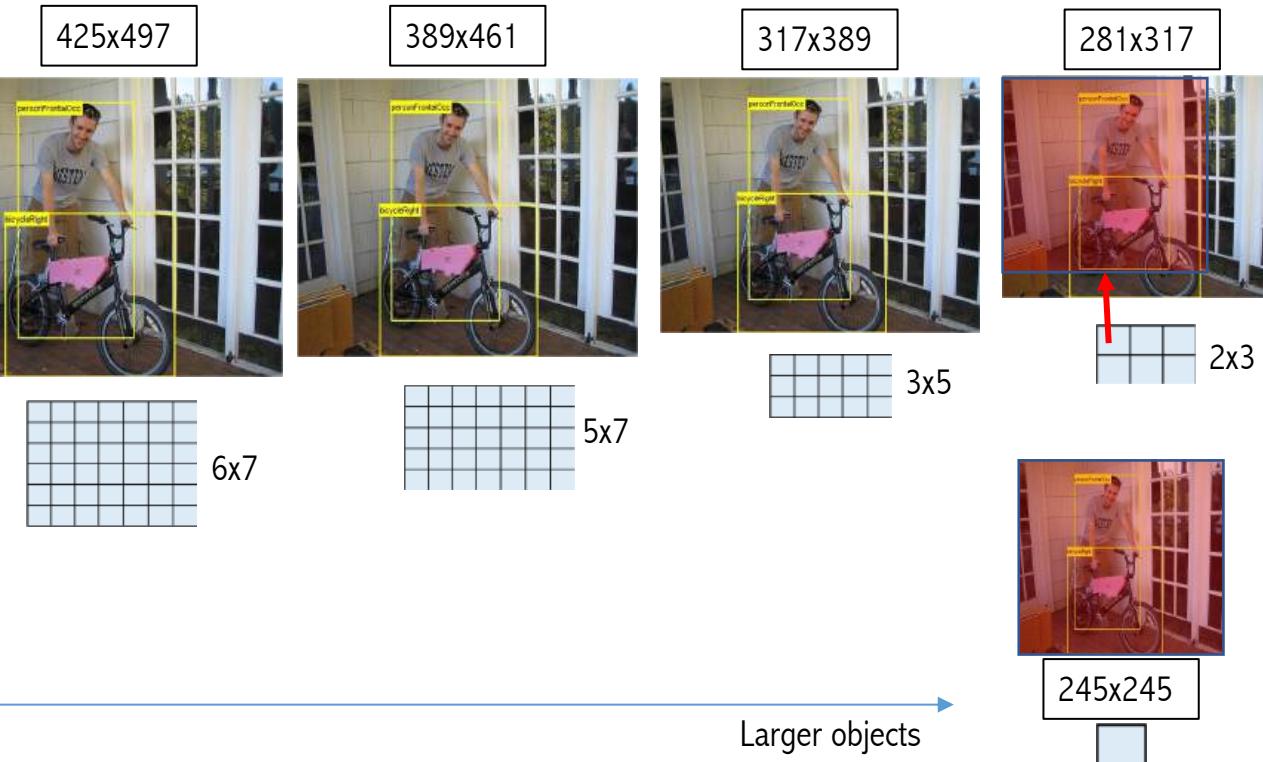
Integrated Recognition, Localization and Detection using Convolutional Networks

~~Sliding Window Crop~~ FC as Conv (No input size constraint) + Spatial Output + Image Pyramid

Resolution = 36



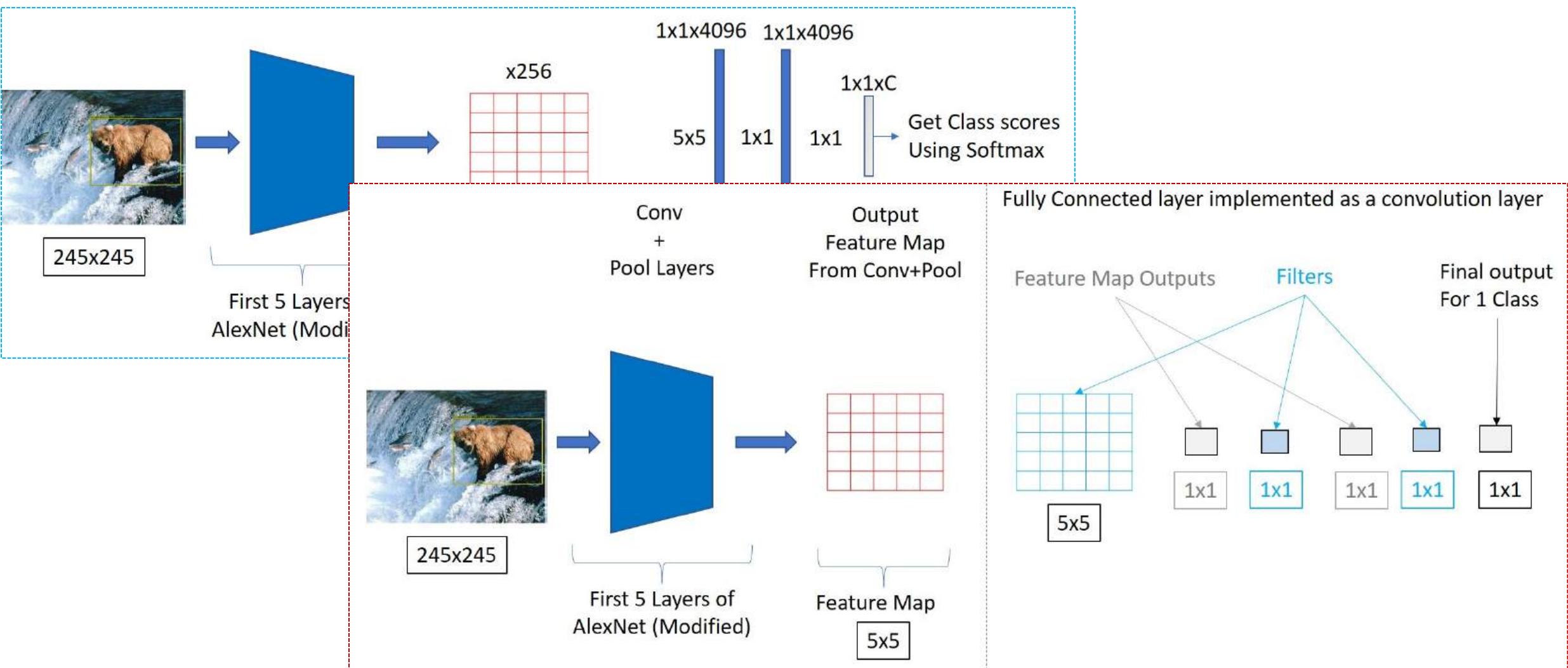
How to modify localization framework to convert FC as Conv?



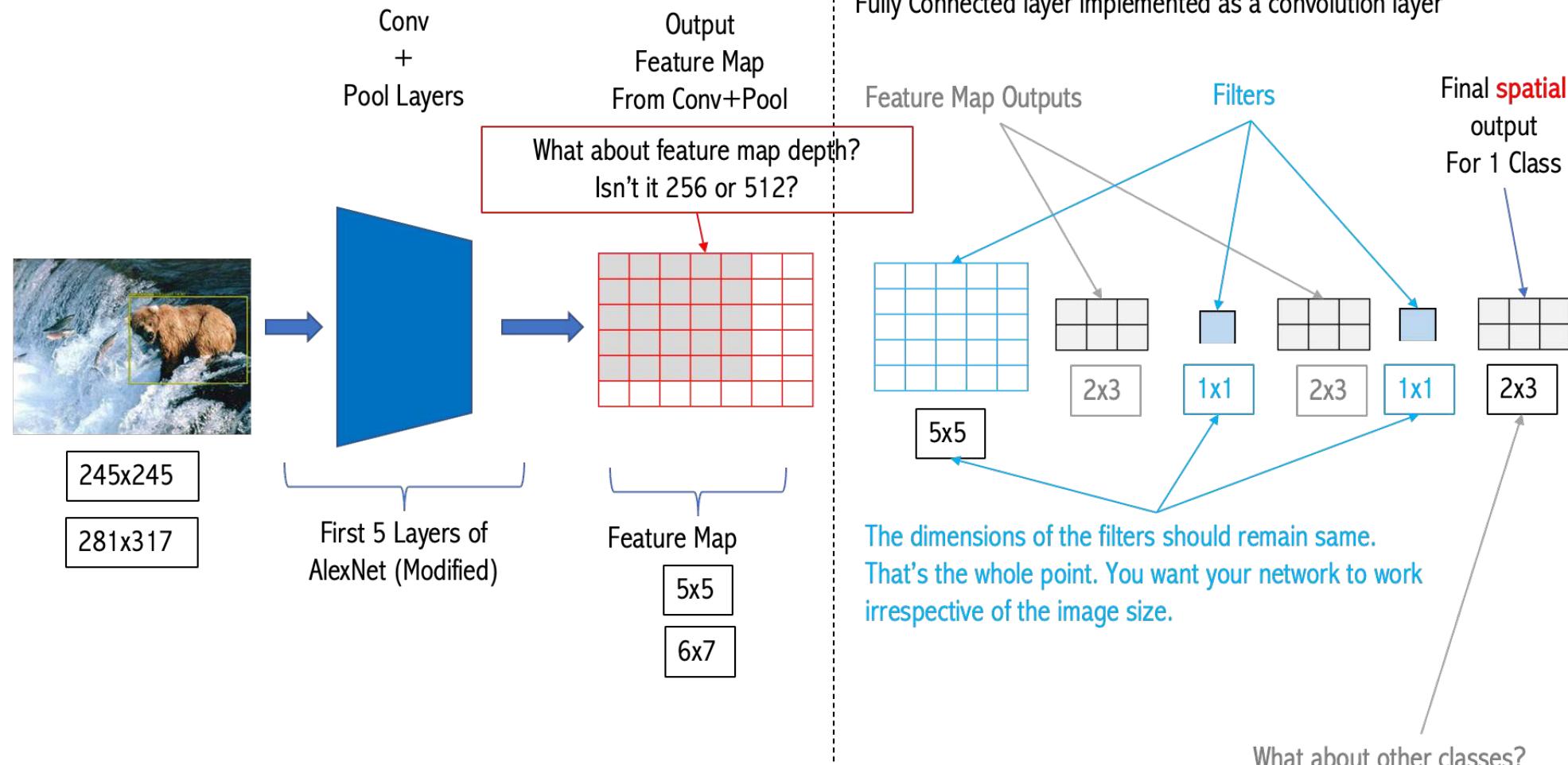
1. Use the same localization network, without using the Sliding Window crops at different locations.
2. No input size constraint, be able to use the Image pyramids.
3. Use Image Pyramids, we will get the Spatial Output, which will give us detections at different locations of the image.
4. The entire network is using Convolution operations, it is way more efficient than taking crops.

If you want to detect even smaller objects, use even bigger image pyramids. Trade-off, increase in computation

OverFeat Classification

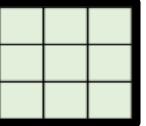
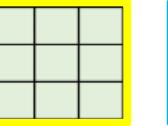
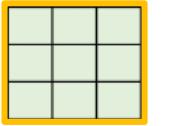
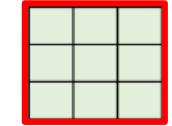
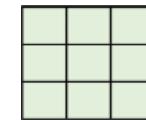


Overfeat Classification

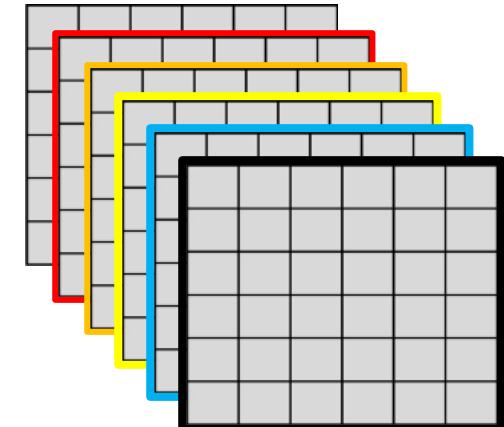
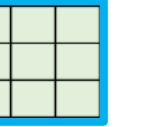
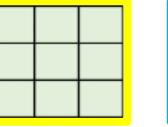
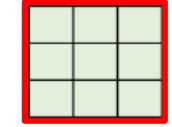
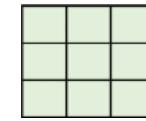


N layer Conv – M Feature Maps

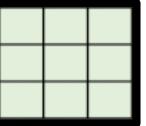
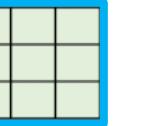
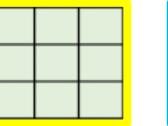
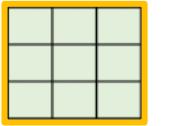
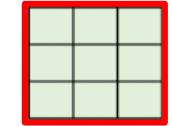
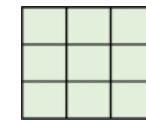
0	0	0	0	0	0	0	0
0							0
0						0	
0					0		
0				0			
0			0				
0		0					
0	0	0	0	0	0	0	0



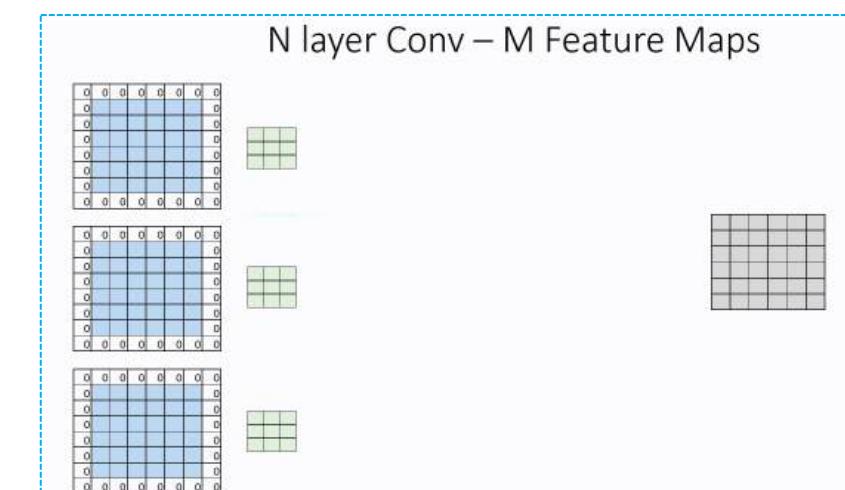
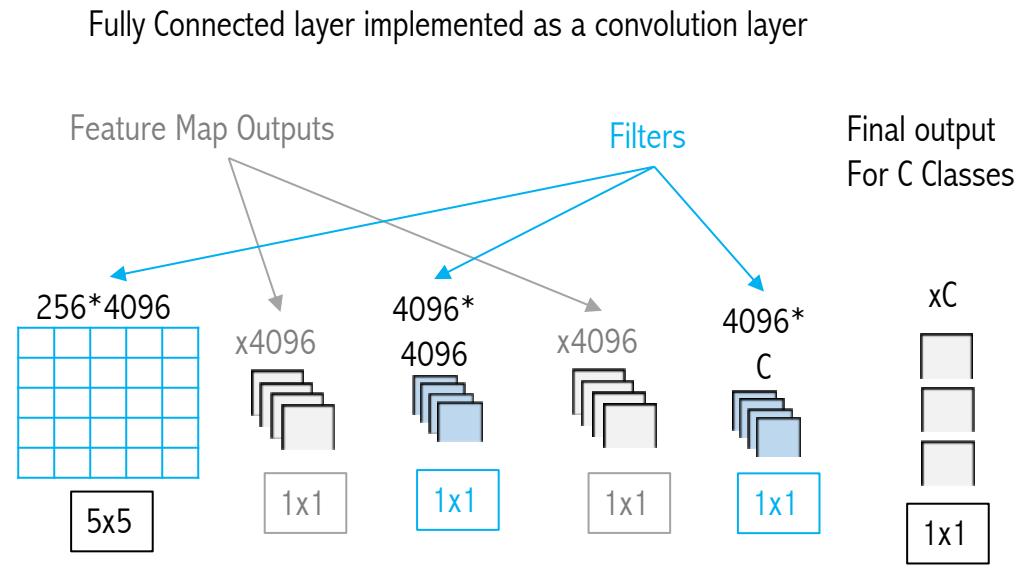
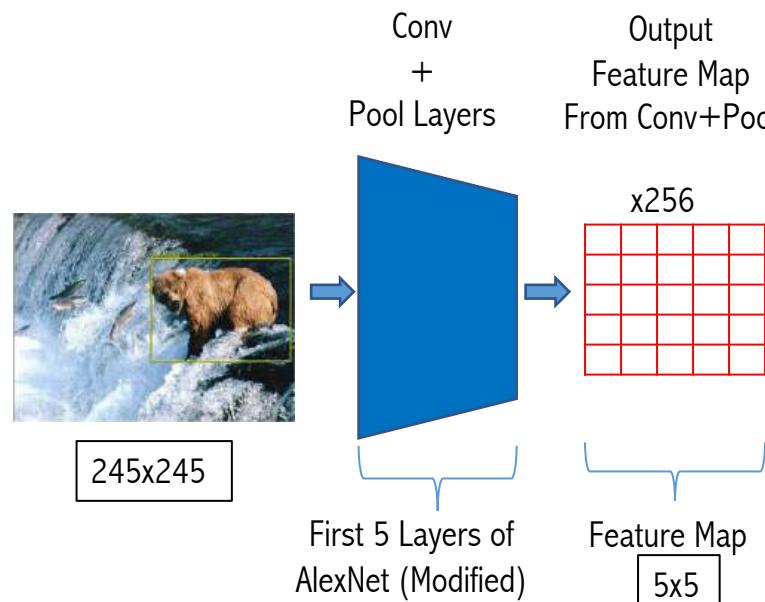
0	0	0	0	0	0	0	0
0							0
0						0	
0					0		
0			0				
0		0					
0	0	0	0	0	0	0	0



0	0	0	0	0	0	0	0
0							0
0						0	
0					0		
0			0				
0		0					
0	0	0	0	0	0	0	0



Overfeat Classification



Overfeat Classification

