



Visión por Computador II

CEAI, FIUBA

Profesores:

- Javier A. Kreiner, javkrei@gmail.com
- Andrés F. Brumovsky, abrumov@gmail.com

Cuarta clase:

- ResNet50 ejercicio de implementación
- Inception Network:
 - convoluciones 1x1
- Localización de objetos y landmarks
- Detección de objetos
- Algoritmo sliding windows
 - ejercicio de implementación
- Familias:
 - R-CNN
 - YOLO
- Algoritmo YOLO

Versiones de ResNet

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	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
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

ResNet50



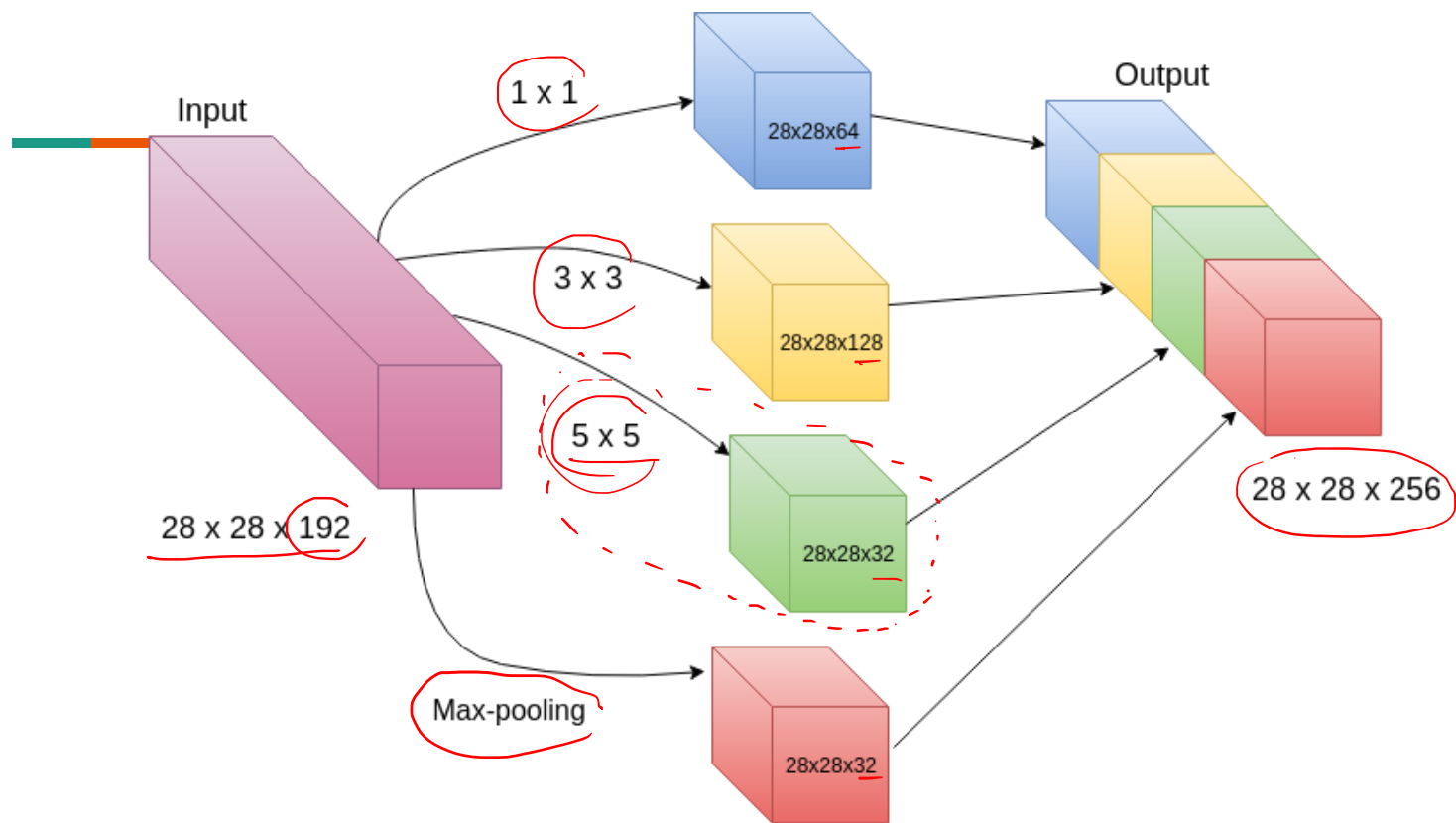
- Ejercicio de programación:
https://colab.research.google.com/drive/14aKayg8puBmOaKiLHFetrL28_CnOzsAJ?usp=sharing



Red Inception



Motivación para la red Inception

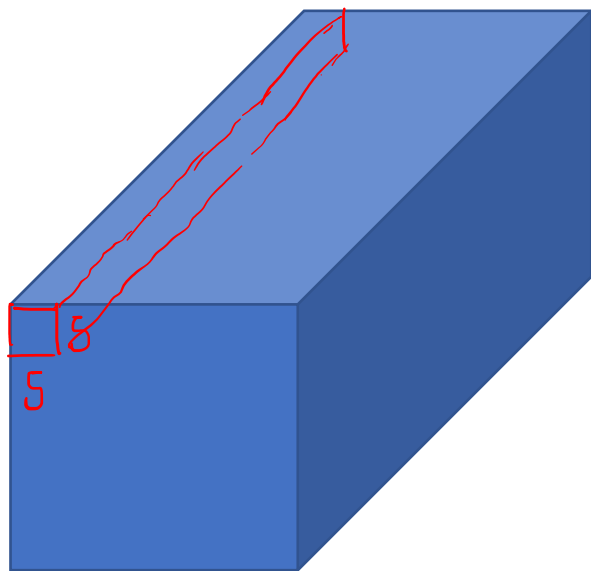


[Szegedy et al. 2014. Going deeper with convolutions]

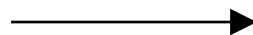
Problema de costo computacional



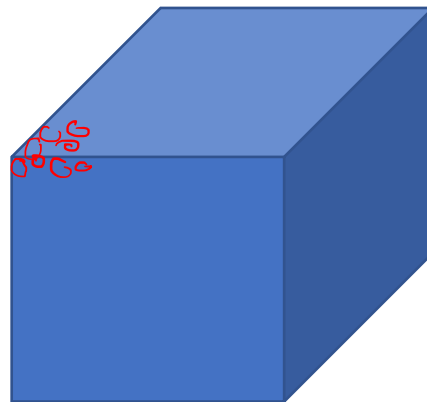
$$\underline{(5 \times 5 \times 192)} \times \underline{(28 \times 28 \times 32)} = 120.422.400$$



$$\underline{28} \times \underline{28} \times 192$$



CONV
 5×5 ,
same,
32



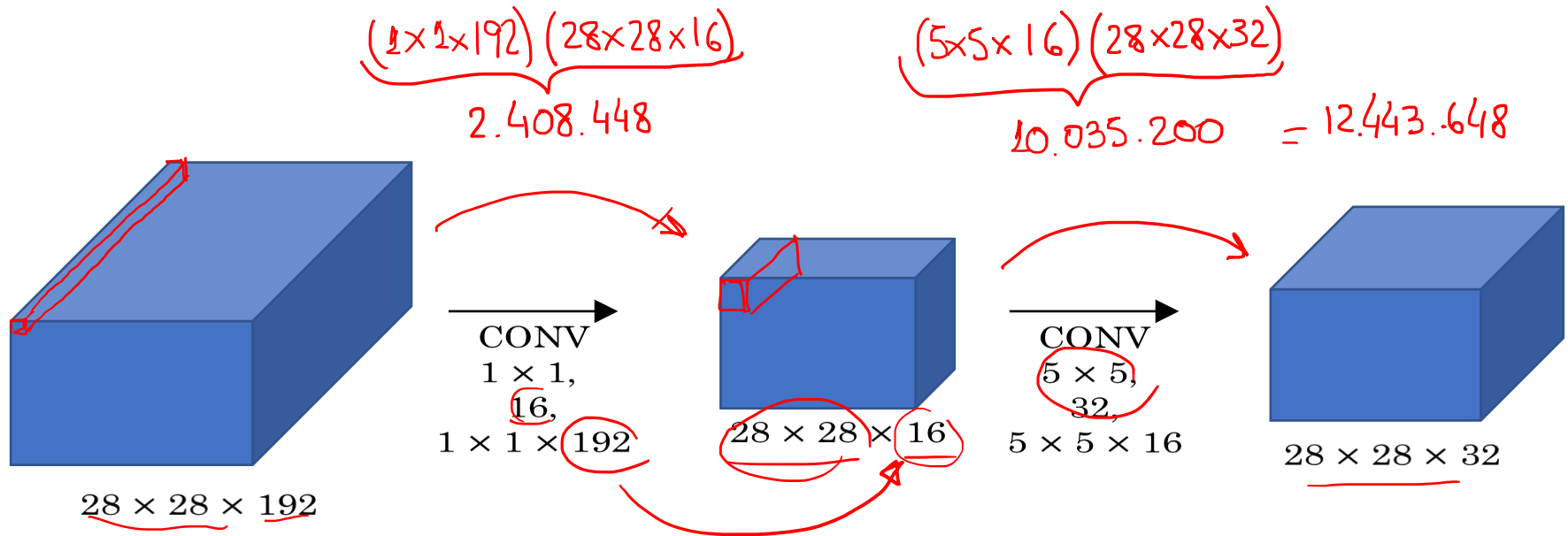
$$\underline{28} \times \underline{28} \times \underline{32}$$

Convoluciones 1x1

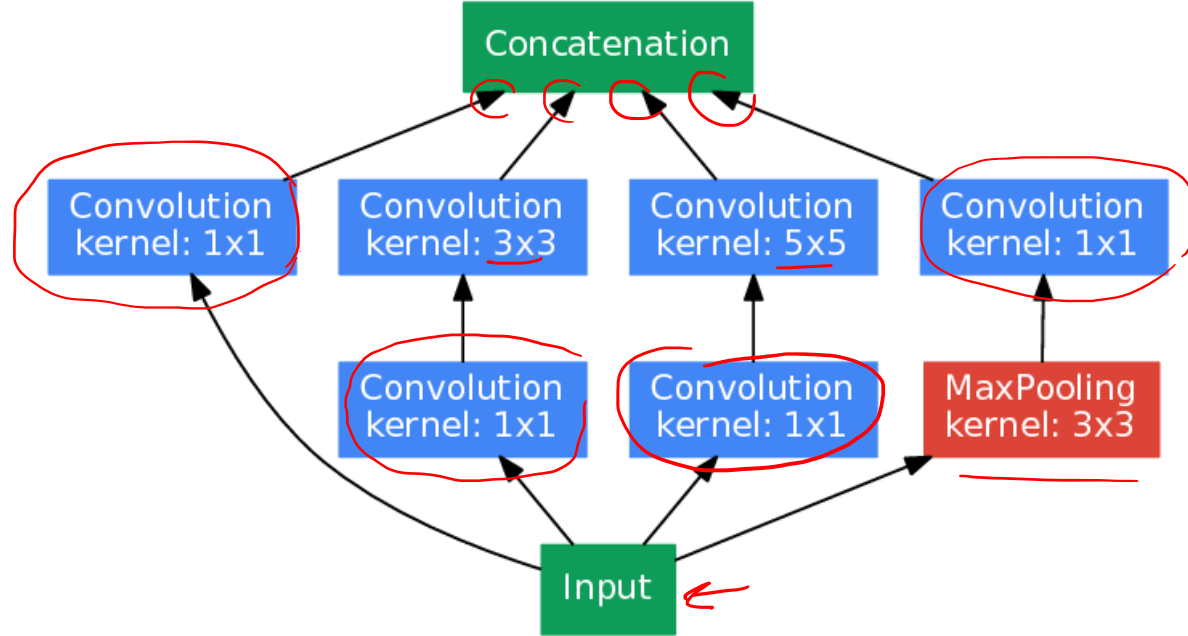


- Jamboard

Reducción de necesidad de cómputo utilizando convolución 1x1

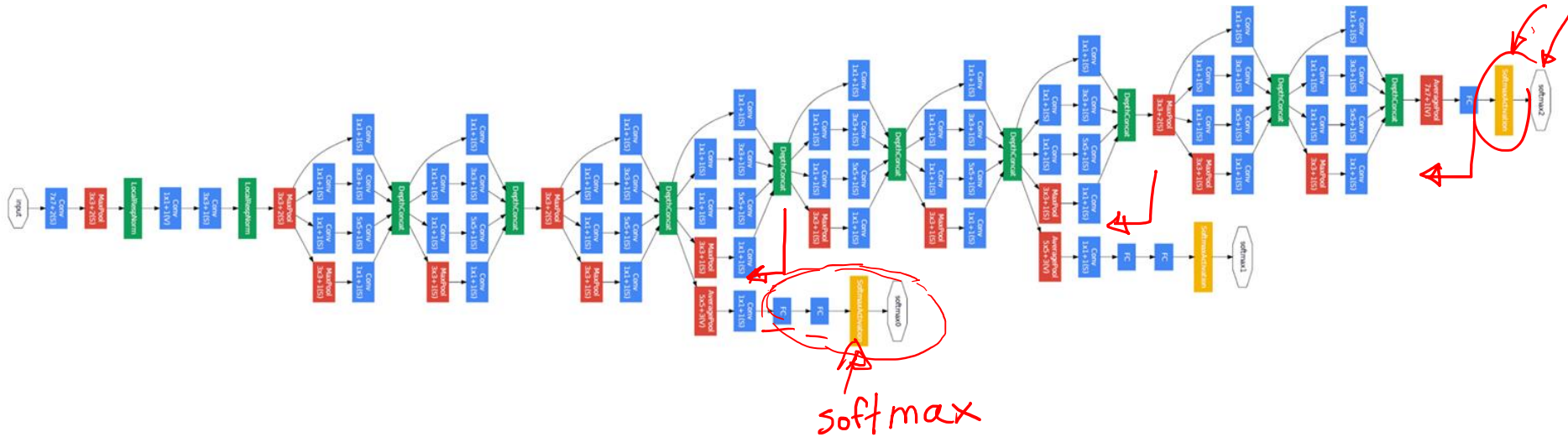


Módulo Inception



- Ejemplo de Programación
- https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deep_With_2015_CVPR_paper.html

Arquitectura Inception = GoogLeNet

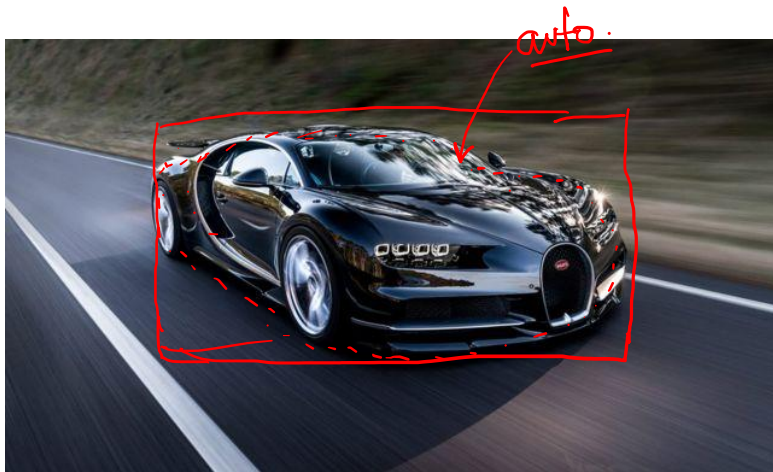


Tareas de visión por computadora



- Clasificación (en general un objeto)
- Localización (en general un objeto)
- Clasificación + Localización (en general un objeto)
- Detección (múltiples objetos de diferentes categorías)

Clasificación – Localización – Clasif.+Local. - Detección





Algunos datasets para detección

- PASCAL VOC Dataset:
<http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html>
- COCO Dataset (Common Objects in COntext), <https://cocodataset.org/>

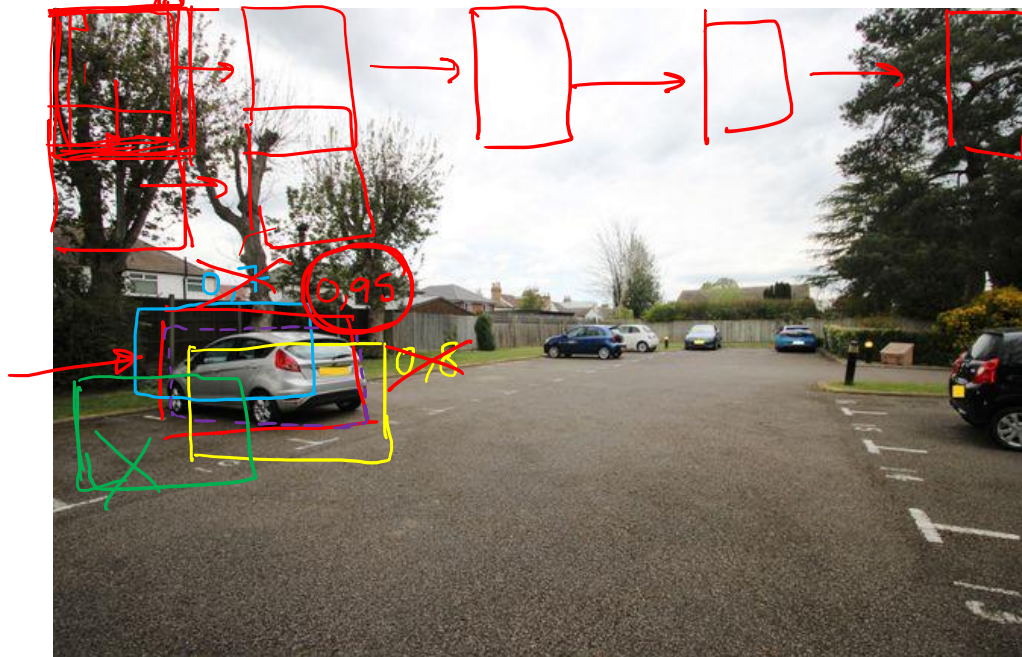
Localización



- Algoritmo de sliding windows
 - Nos apoyamos sobre un clasificador existente
 - Desventaja: gran costo computacional
 - las bounding boxes no son muy precisas
- Jamboard

Sliding Window

$I.O.U > 0,5$

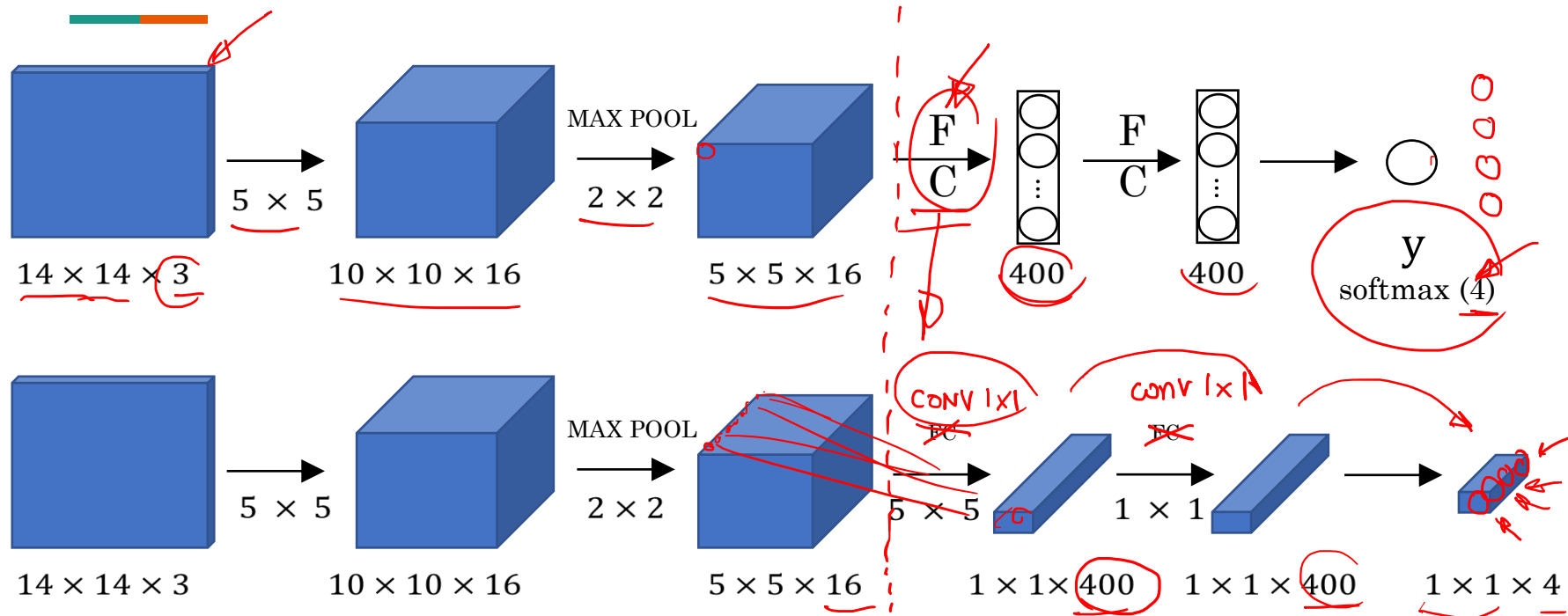


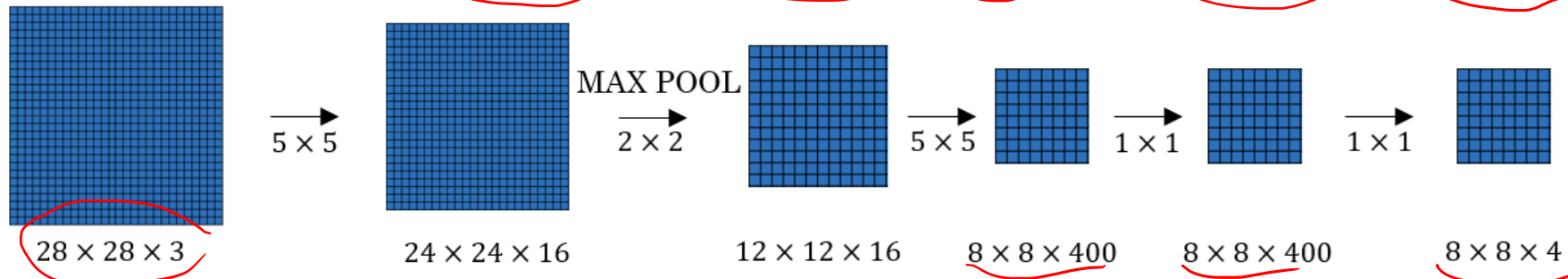
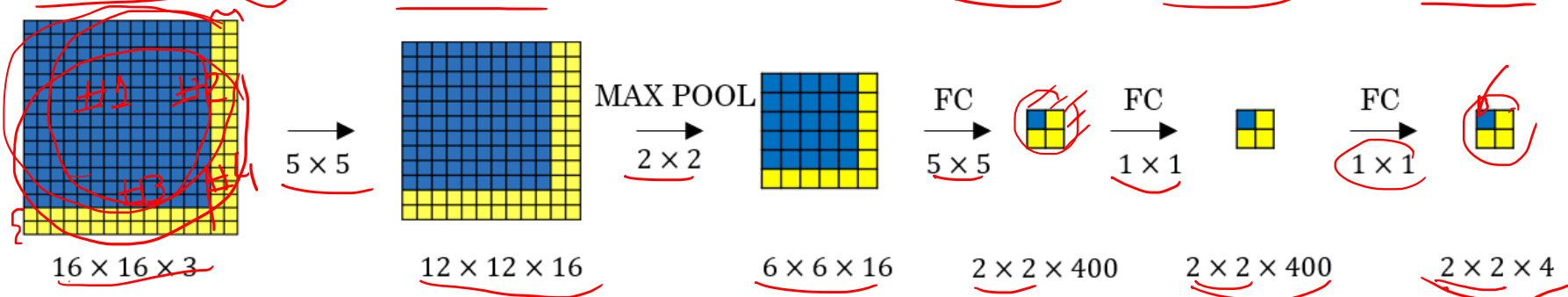
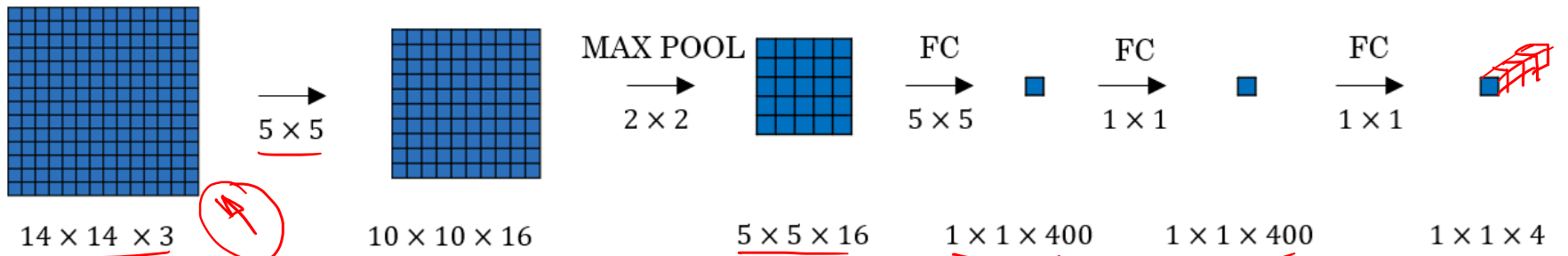
Cómo implementar sliding windows de manera convolucional

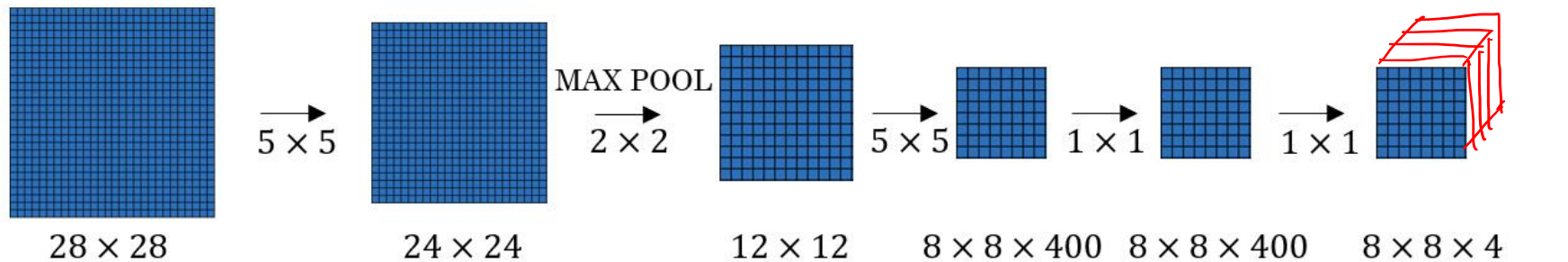


- Permite reutilizar los cálculos en el algoritmo de sliding windows
- Hace los cálculos en paralelo

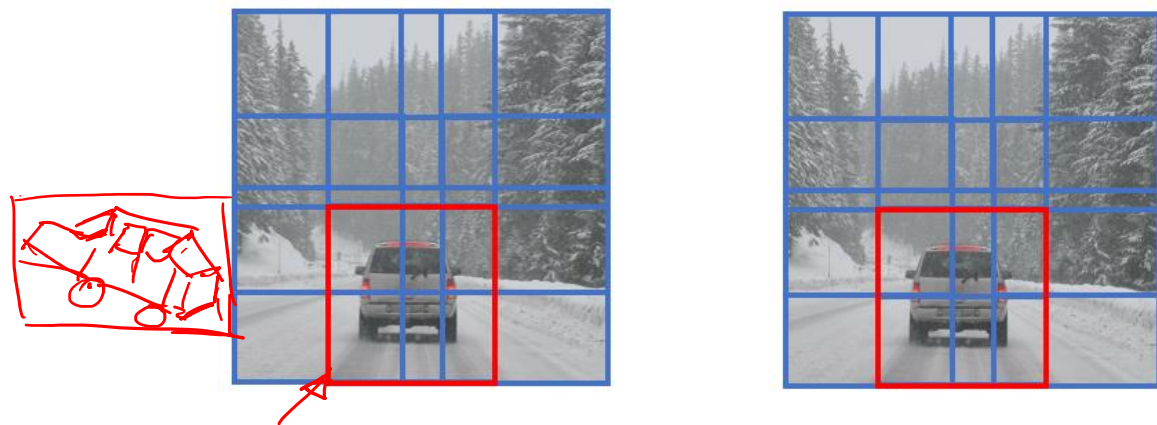
Convertir en convolución:







→
class?



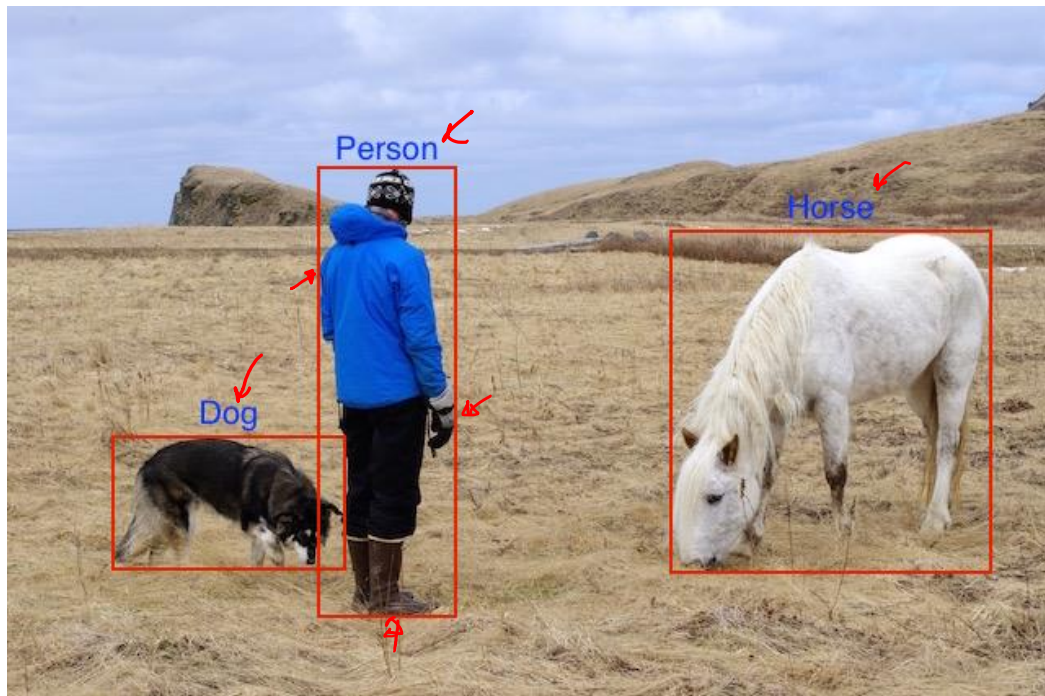
Medida de performance de los algoritmos

- mAP: mean Average Precision para detección de objetos (hay otra definición para document retrieval)
- El algoritmo debe encontrar los objetos (ubicarlos en bounding boxes) y clasificarlos
- Tenemos que evaluar cuán correctas son las bounding boxes y cuán correcta es la clasificación

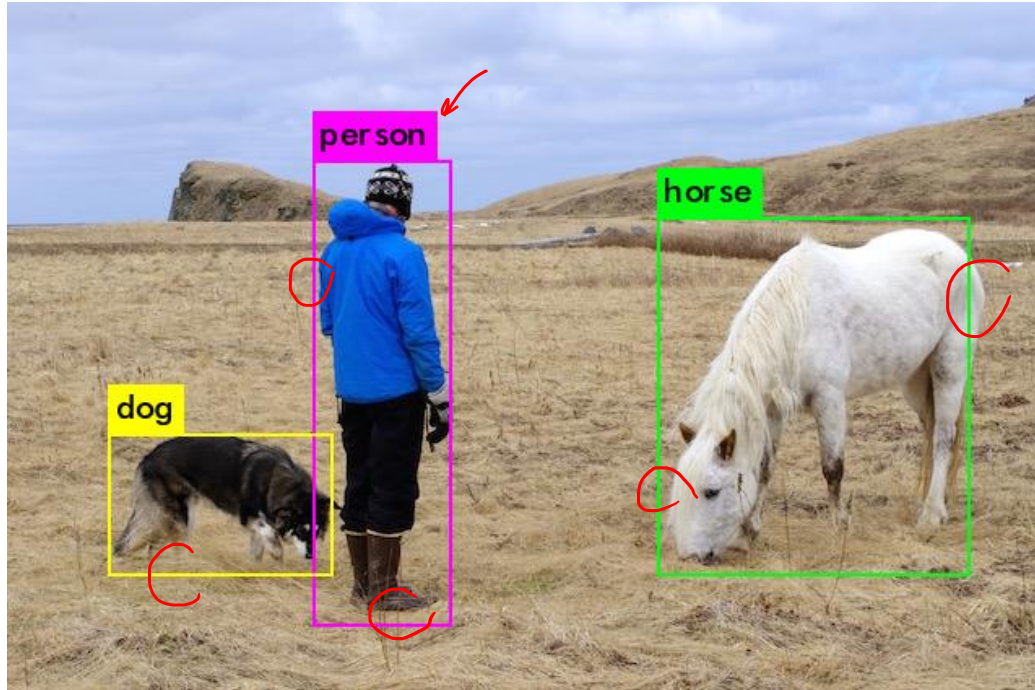
Descripción original, página 11 de:

http://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf

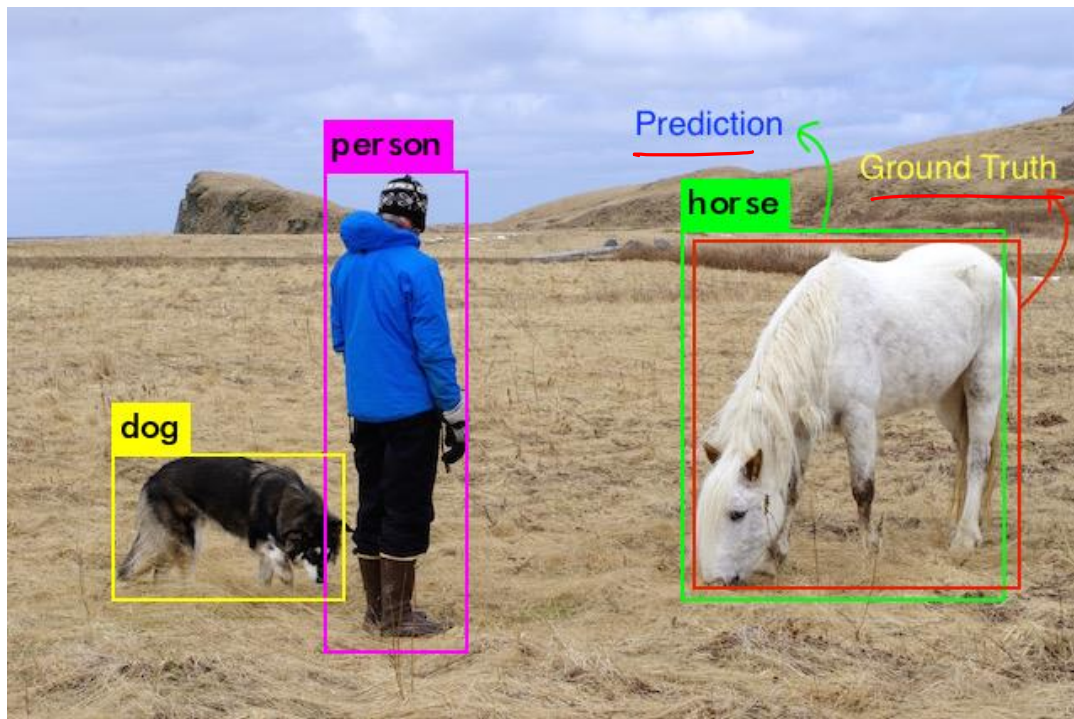
Ground Truth



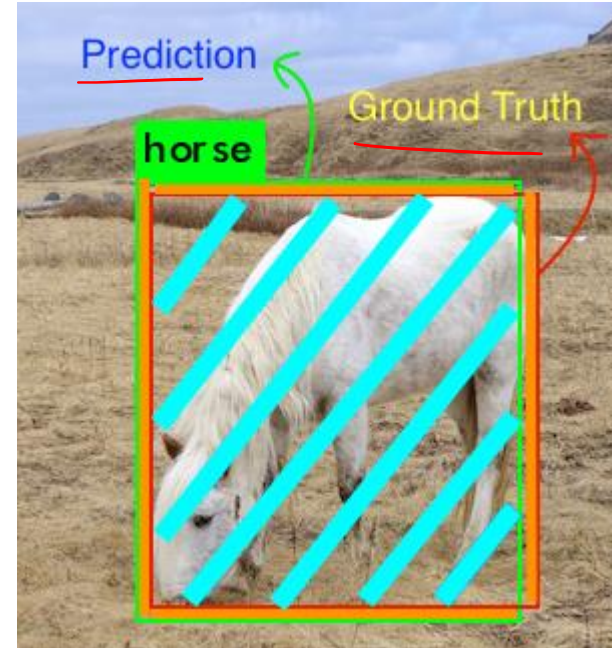
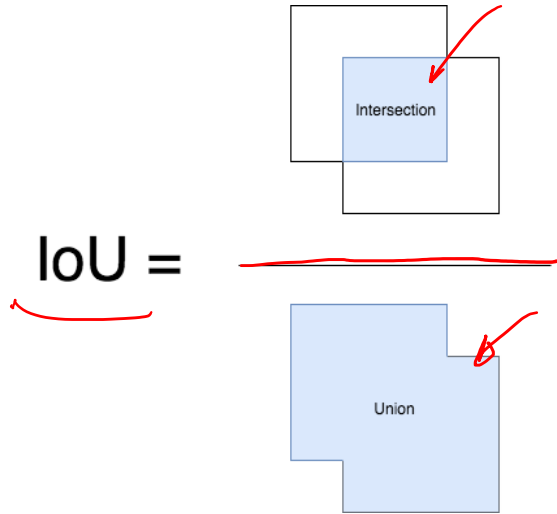
Supongamos que el algoritmo predice esto:



Comparando...



Intersection over Union (o Jaccard Index)



Cómo evaluar la salida de los modelos

- Los modelos retornan para cada imagen una lista de : una bounding box con una clase predicha y un nivel de confianza
- Las predicciones son asociadas a un objeto del ground truth si $IoU > 0.5$ (para Pascal VOC es 0.5; para COCO se usan una serie de niveles de IoU de corte)
- Múltiples detecciones de un objeto son ranqueadas según el nivel de confianza asignado
- Si hay varias predicciones para un objeto sólo una se considera correcta, las demás incorrectas (el algoritmo debería descartar múltiples detecciones)
- Para cada clase del dataset se calcula la curva de precision/recall, que especifica el nivel de precisión (definida como proporción de los ejemplos por encima de un rango que pertenecen a la clase correcta) para un dado recall (proporción de los ejemplos positivos que aparecen por encima de un cierto rango)
- Average precision resume la forma de la curva de precision/recall, y es definido con la precision media en un conjunto de 11 niveles de recall equiespaciados: $[0, 0.1, \dots, 1]$ (en Pascal VOC)
 - Es una medida que se evalúa por cada clase a clasificar, o sea, debe medirse a nivel del dataset
 - Es bastante buena para comparar métodos diferentes

Ejemplo

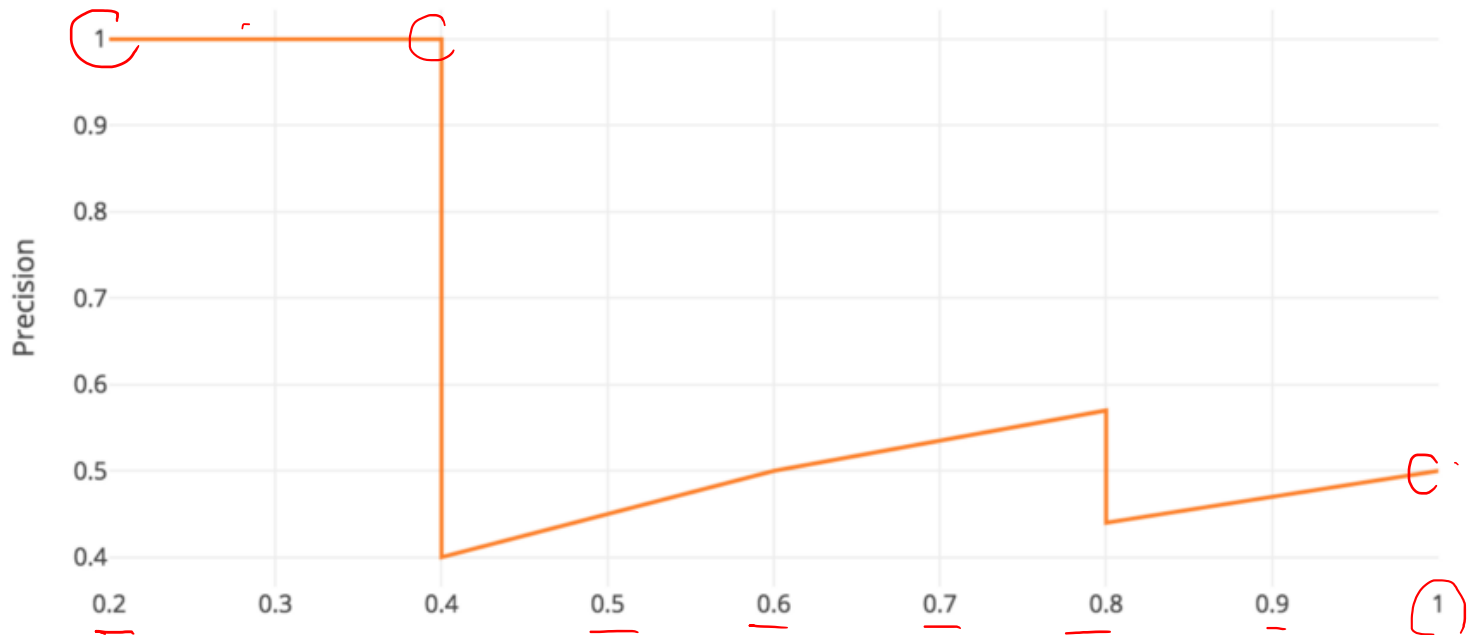
Supongamos que el dataset tiene sólo 5 manzanas y que el algoritmo retorna esto:

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0



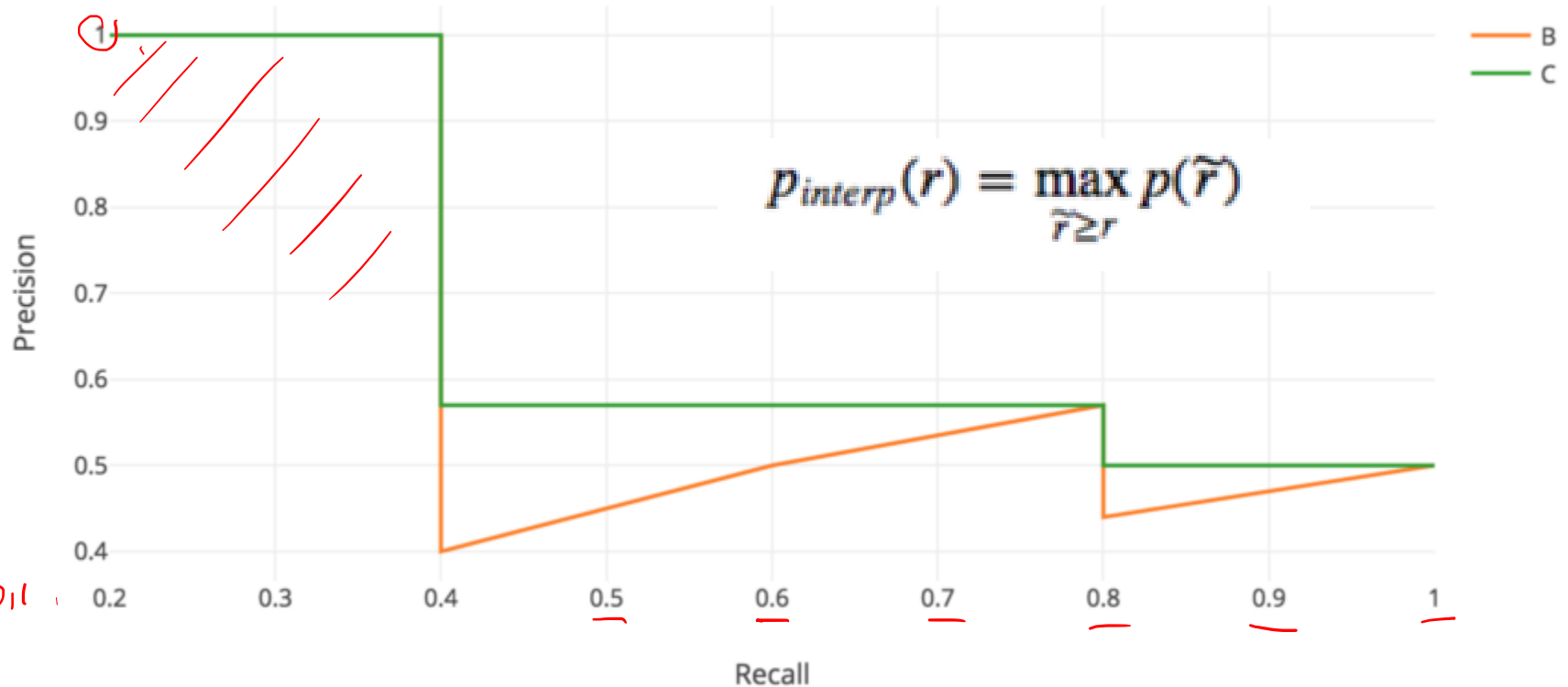
Rank	Correct?	Precision	Recall
1	True	1.0 ↑	0.2 ↑
2	True	1.0 -	0.4 ↑
3	False	0.67 ↓	0.4 -
4	False	0.5 ↓	0.4 -
5	False	0.4 ↓	0.4 -
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑

Curva de precision/recall




Average precision: $AP = \int_0^1 p(r)dr$

En Pascal VOC interpolan:



Finalmente se toman 11 intervalos y se hace el promedio


$$AP = \frac{1}{11} \times (AP_r(0) + AP_r(0.1) + \dots + AP_r(1.0))$$

Y en el ejemplo dado:

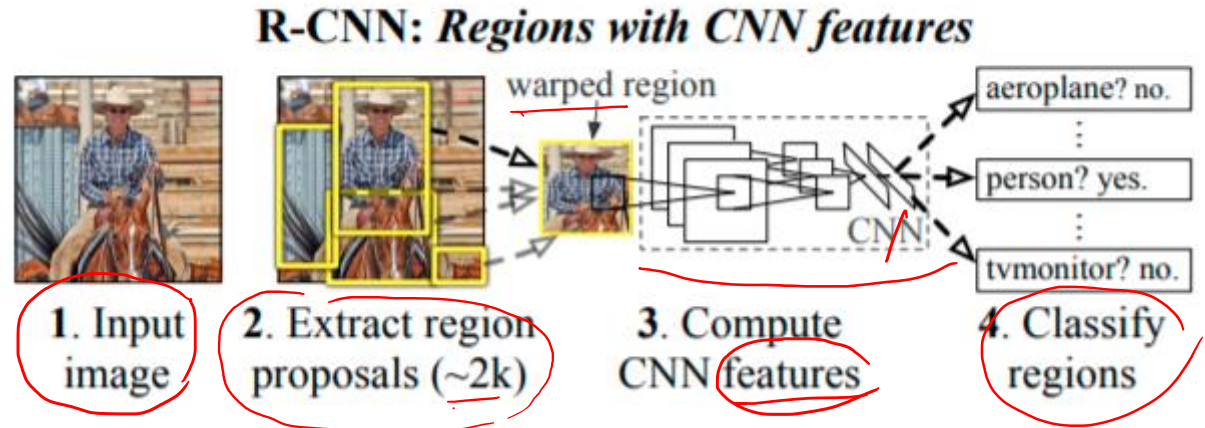
$$AP = (5 \times 1.0 + 4 \times 0.57 + 2 \times 0.5) / 11 = \underline{\underline{0.7527}}$$



Algoritmos

- Métodos de dos etapas:
 - R-CNN (region proposals CNN)
 - Fast R-CNN
 - Faster R-CNN
 - otras variaciones
- Métodos de una etapa:
 - YOLO (You Only Look Once)
 - Otros: RetinaNet, SSD (Single-shot detection)

R-CNN



- En la primera etapa, búsqueda selectiva (selective search):
 - Se genera una sub-segmentación inicial, se generan muchas regiones candidatas
 - Se usa un algoritmo greedy para combinar recursivamente regiones similares en regiones mayores
 - Se usan las regiones generadas para producir las propuestas de regiones candidatas (region proposals), alrededor de 2000 por imagen
 - <http://www.huppelen.nl/publications/selectiveSearchDraft.pdf>
- Segunda etapa:
 - Cada una de las regiones es pasada por una red convolucional que extrae features
 - En la capa final se utiliza una Support Vector Machine para clasificar cada región propuesta y asignarle un nivel de confianza a cada región
 - Se utiliza non-max suppression para cada clase para determinar las regiones que sobreviven
- paper original: Rich feature hierarchies for accurate object detection and semantic segmentation
<https://arxiv.org/pdf/1311.2524.pdf>

Non-max suppression

T

- Primero descartamos las regiones que tengan confianza $< \text{umbral1}$ (e.g. 0.4)
- Consideremos una clase y tomemos todas las regiones clasificadas como de esa clase para una imagen
- Se ordenan las regiones por nivel de confianza,
 - se toma la que tiene mayor nivel de confianza, se eligen todas las que tienen IoU $> \text{umbral2}$ (e.g. 0.5) y son descartadas
 - se toma la proxima region por nivel de confianza
 - hasta terminar la lista



Algoritmo YOLO

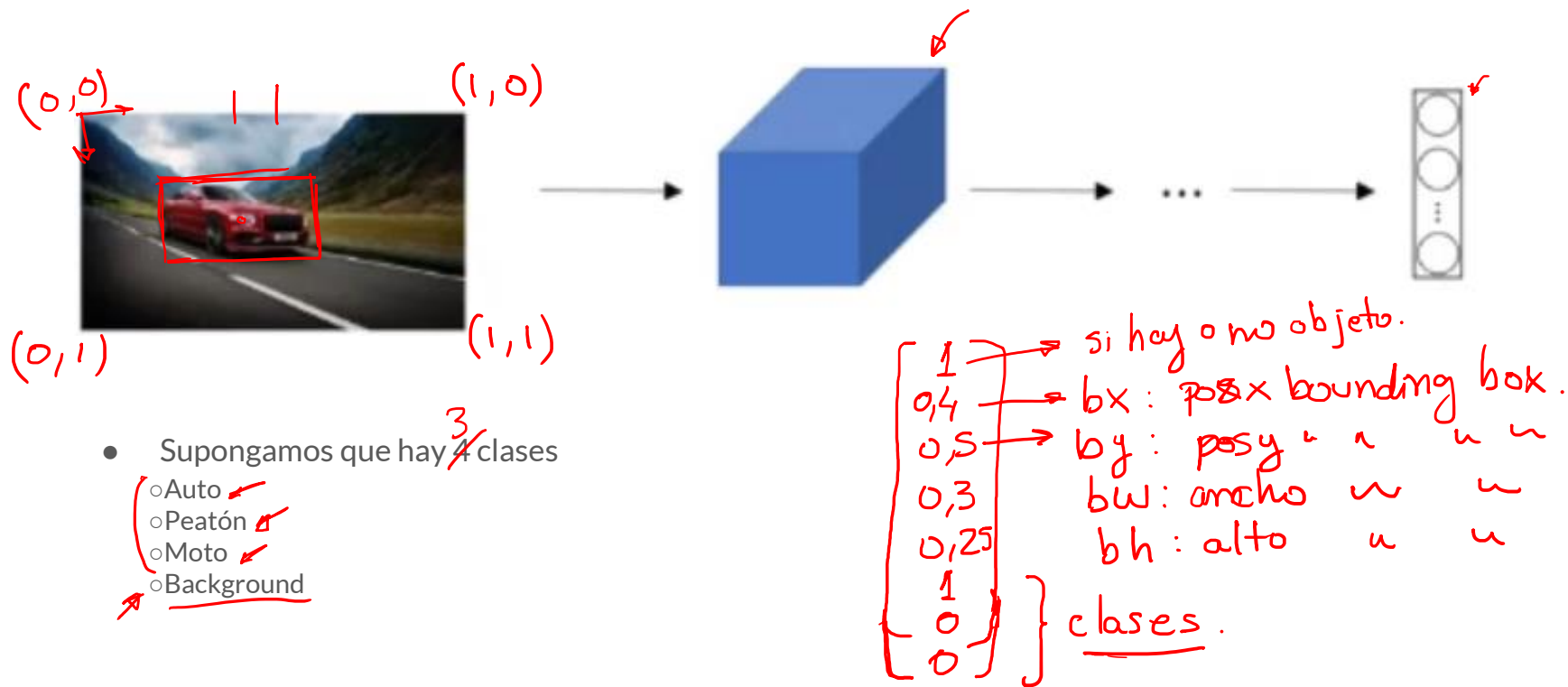
- Algoritmo de una sola etapa
- ~~Muy veloz~~
- Varias versiones
- YOLO watches youtube: <https://www.youtube.com/watch?v=U9c1gXO8xEU>
- YOLO watches nature: <https://www.youtube.com/watch?v=dTcfAuCEV3A>
- artículo original: You Only Look Once: Unified, Real-Time Object Detection:
<https://arxiv.org/pdf/1506.02640.pdf>

Clasificación + Localización

- Ejemplo en Jamboard de cómo ‘construir’ el dataset
- ¿Cómo definir la función de pérdida?

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

Clasificación + Localización



Ejemplos



Vector:

$\begin{bmatrix} 1 \\ 0,4 \\ 0,5 \\ 0,3 \\ 0,25 \\ 1 \\ 0 \\ 0 \end{bmatrix}$

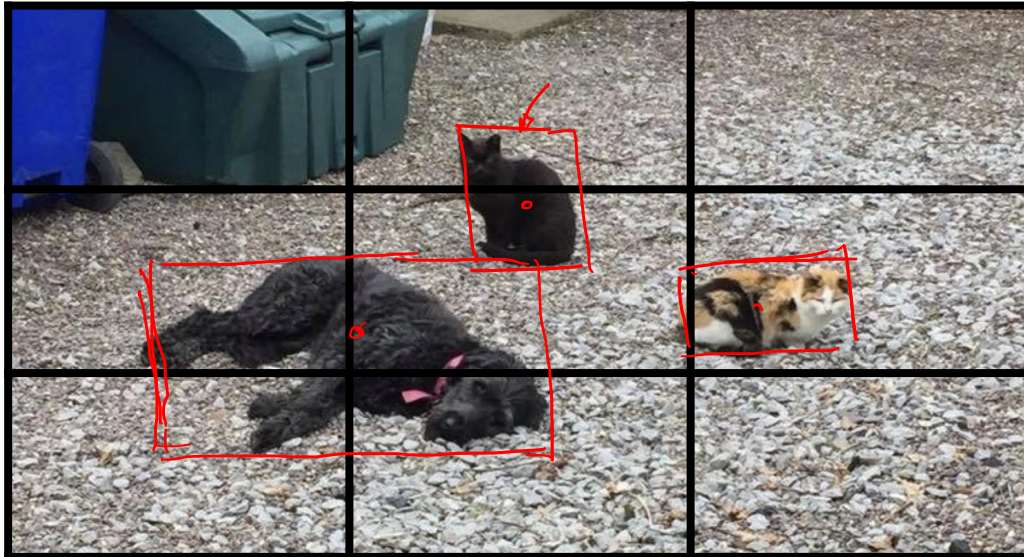


Vector:

$\begin{bmatrix} 0 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$

Grilla sobre la imagen, vamos a clasificar y localizar en cada celda

- 1. Perro
- 2. Gato
- 3. Conejo



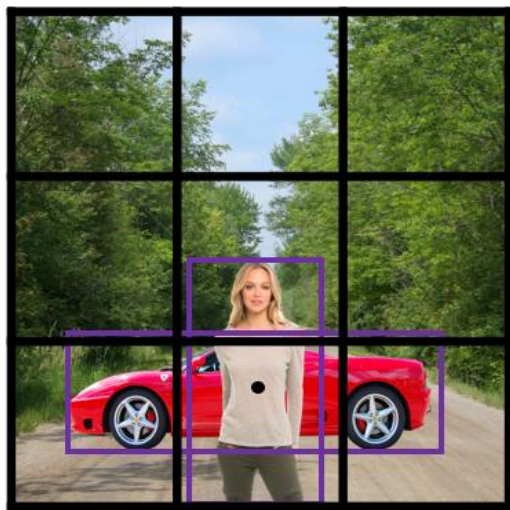
Salida del algoritmo:

- Cada celda de la grilla se representa por un vector
- Un objeto es asignado a una celda si su centro está dentro de ella
- Volumen de salida: $\#celdas_w \times \#celdas_h \times (1 + 4 + \#clases)$
- Comparado a sliding windows:
 - permite determinar una bounding box mucho más precisa
 - es convolucional



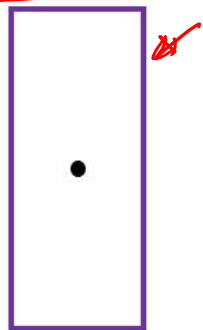
Otra idea: anchor boxes

- ¿Cómo tener varios objetos en la misma celda?

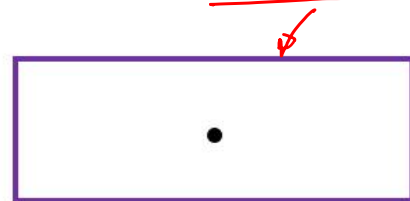



$$\textcircled{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Anchor box 1:



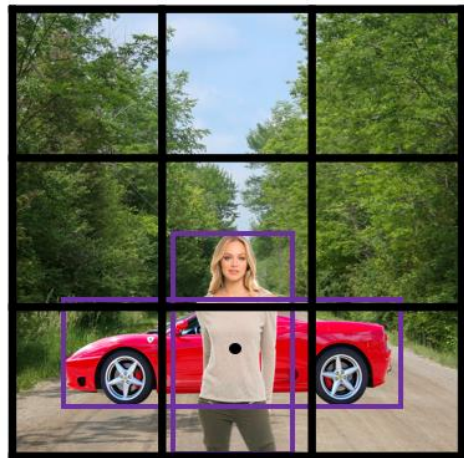
Anchor box 2:



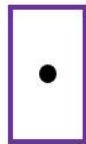


Antes: cada objeto en la imagen de entrenamiento es asignado a una celda de la grilla que contiene el punto medio del objeto.

Ahora: Cada objeto en la imagen de entrenamiento es asignado a la celda de la grilla que contiene el centro del objeto y a la anchor box con la cual tiene mayor IoU.



Anchor box 1:



Anchor box 2:



$$\underline{\underline{y}} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Kahoot de detección de objetos



POSIBLES Trabajo final



- Competencias en kaggle:
 - <https://www.kaggle.com/c/siim-covid19-detection/overview>
 - <https://www.kaggle.com/c/imaterialist-fashion-2020-fgvc7/overview>
 - <https://www.kaggle.com/c/flower-classification-with-tpus>
 - <https://www.kaggle.com/c/bengaliai-cv19/leaderboard>
 - <https://www.kaggle.com/c/Kannada-MNIST>
 - https://www.kaggle.com/c/understanding_cloud_organization/data
 - <https://www.kaggle.com/c/severstal-steel-defect-detection/overview>