

# Detección de Objetos

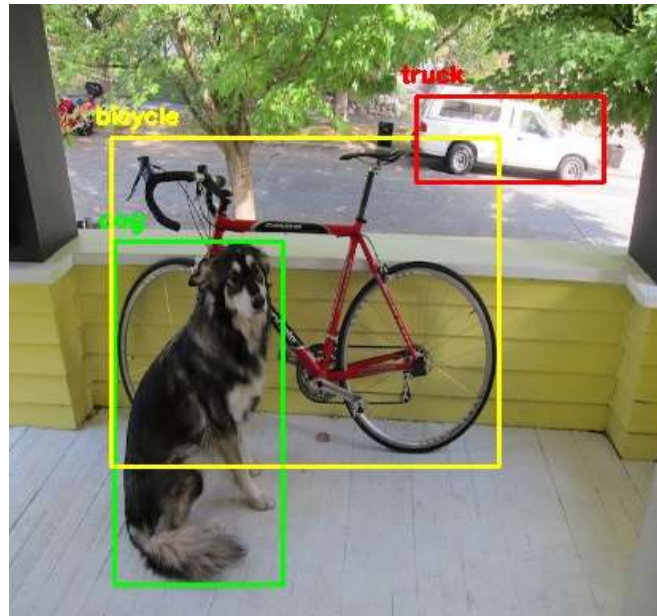


Dr. Iván Sipirán Mendoza

# Objetivo

---

Determinar la ubicación de los objetos en la imagen



# Datos y Evaluación



# Datos

---

Es necesario contar con datasets para esta tarea

- Suficientemente grandes
- Etiquetados



Pascal VOC



COCO dataset

# Evaluación de detección

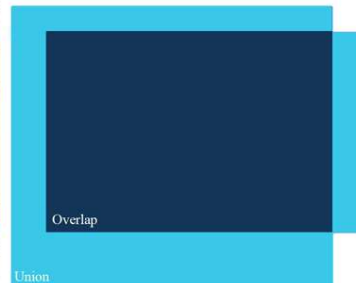
Dado un groundtruth y las detecciones de un algoritmo

- Se considera una detección exitosa si hay alto overlap entre los bounding boxes



□ Ground truth  
□ Prediction


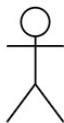




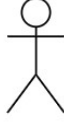



$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$

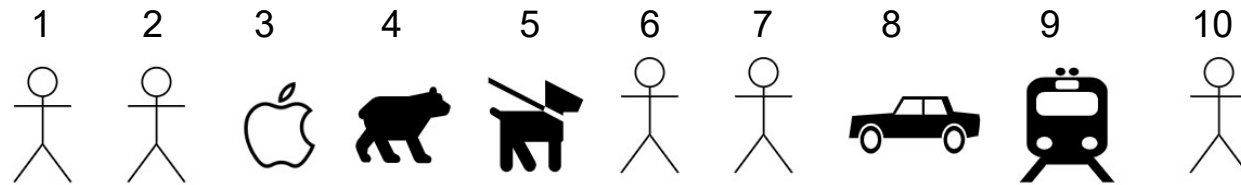


$IoU \geq 0.5$

# Evaluación de detección

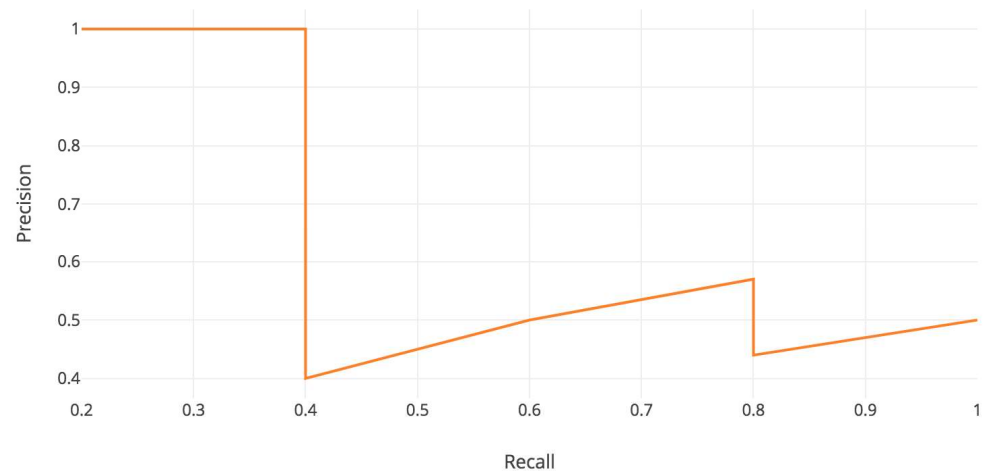
- Medida más común es el Average Precision (AP), que se calcula por cada clase de objeto.
- Colectamos todas las predicciones en donde el algoritmo nos diga que hay un objeto de una clase y se ordenan por valor de confianza.
- Ejemplo con clase “persona”

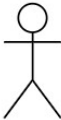
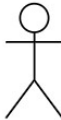








	1	2	3	4	5	6	7	8	9	10
										
P.	1	1	0.67	0.5	0.4	0.5	0.57	0.5	0.44	0.5
R.	0.2	0.4	0.4	0.4	0.4	0.6	0.8	0.8	0.8	1.0



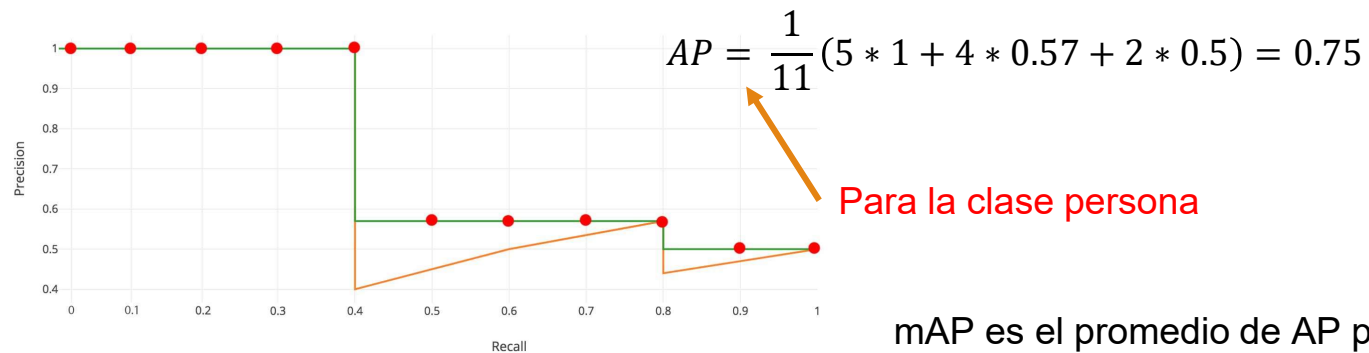
P.	1	1	0.67	0.5	0.4	0.5	0.57	0.5	0.44	0.5
R.	0.2	0.4	0.4	0.4	0.4	0.6	0.8	0.8	0.8	1.0

Si dibujamos la curva de precisión: AP es el área bajo la curva



	1	2	3	4	5	6	7	8	9	10
										
P.	1	1	0.67	0.5	0.4	0.5	0.57	0.5	0.44	0.5
R.	0.2	0.4	0.4	0.4	0.4	0.6	0.8	0.8	0.8	1.0

En la práctica se toman valores interpolados de recall y se suman los valores monótonicamente decrecientes de precisión



mAP es el promedio de AP para todas las clases



# Region-based Convolutional Network (RCNN)

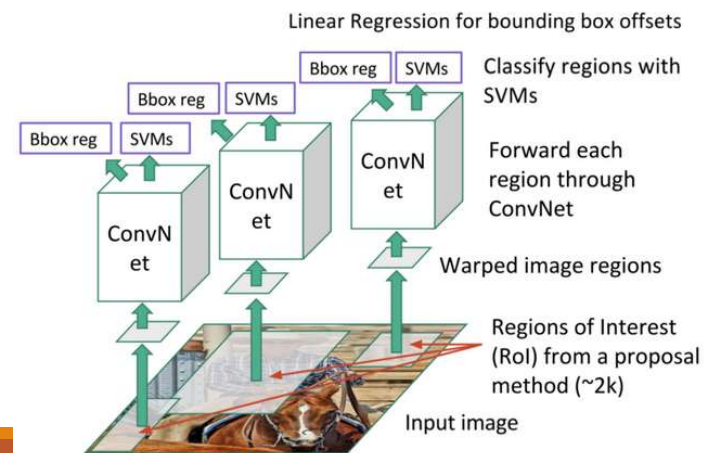
---



# Region-based Convolutional Network (RCNN)

Algoritmo de tres etapas

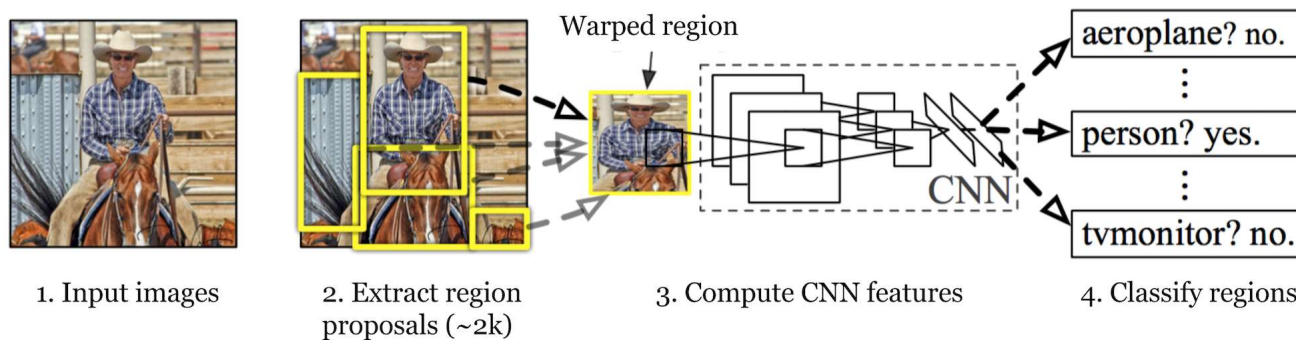
- Usar un algoritmo para generar regiones candidatas en imágenes (Selective Search)
- Extraer feature de región candidata con CNN pre-entrenada
- Clasificación (objeto) + Regresión (bounding box): SVM + regresor mínimos cuadrados



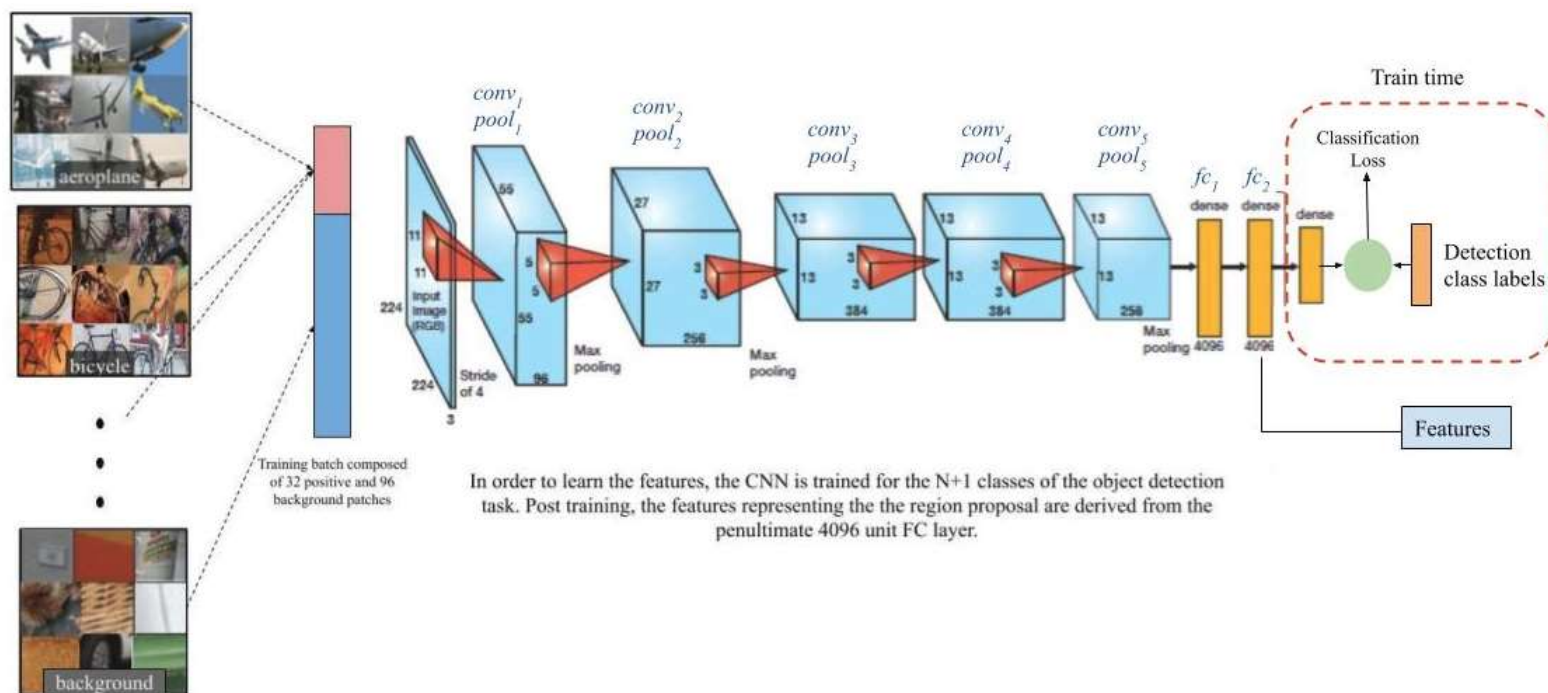
# Region-based Convolutional Network (RCNN)

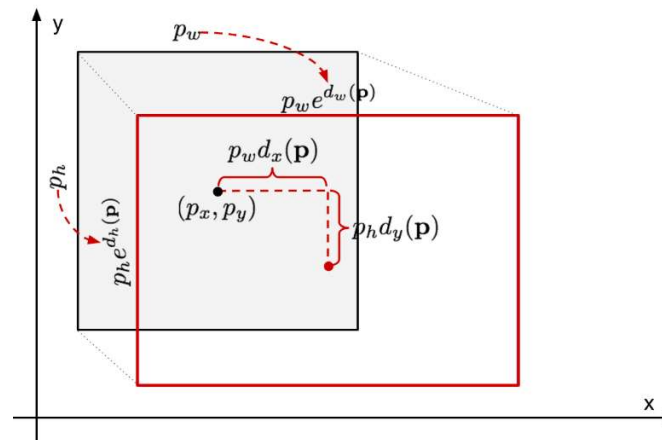
Algoritmo de tres etapas

- Usar un algoritmo para generar regiones candidatas en imágenes (Selective Search)
- Extraer feature de región candidata con CNN pre-entrenada
- Clasificación (objeto) + Regresión (bounding box): SVM + regresor mínimos cuadrados



# Region-based Convolutional Network (RCNN)





# Region-based Convolutional Network (RCNN)

---

Regresión de bounding boxes

- Optimización

$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t_\star^i - \hat{\mathbf{w}}_\star^\top \phi_5(P^i))^2 + \lambda \|\hat{\mathbf{w}}_\star\|^2.$$

$$t_x = (g_x - p_x)/p_w$$

$$t_y = (g_y - p_y)/p_h$$

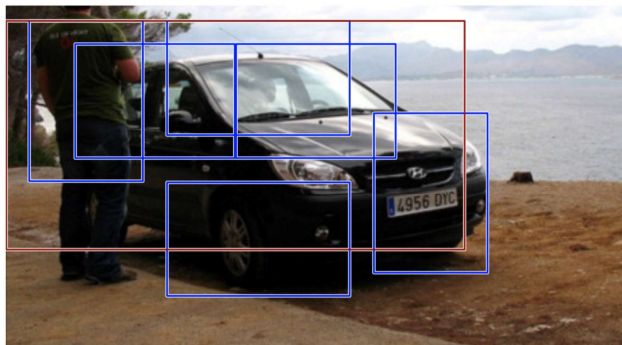
$$t_w = \log(g_w/p_w)$$

$$t_h = \log(g_h/p_h)$$

# Region-based Convolutional Network (RCNN)

## Post-procesamiento

- Supresión de no máximos: podrían generarse muchos bounding boxes asociados al mismo objeto.
- **Estrategia:**
  - Ordenar bounding boxes por valor de confianza. Escoger el de valor más alto.
  - Remover todos los bounding boxes con  $\text{IOU} \geq 0.5$  y misma clase que el escogido.
  - Repetir hasta no poder eliminar más bounding boxes.



Before non-max suppression



After non-max suppression

# Fast RCNN

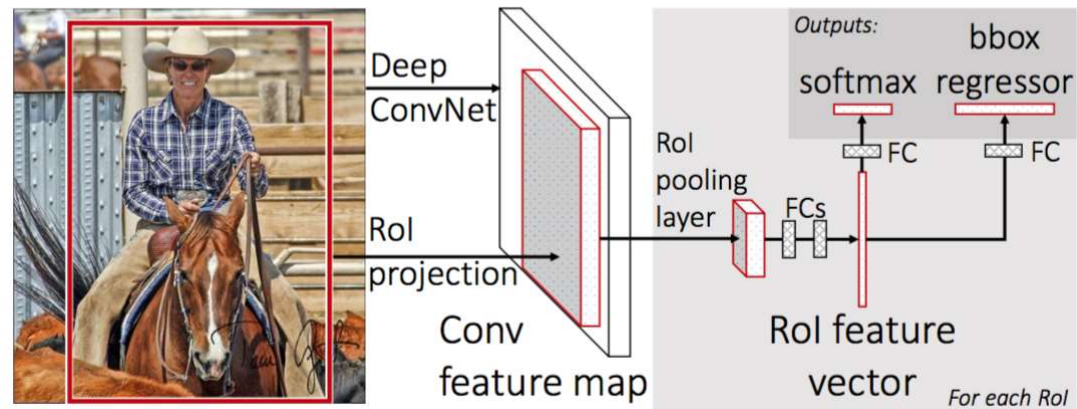
---





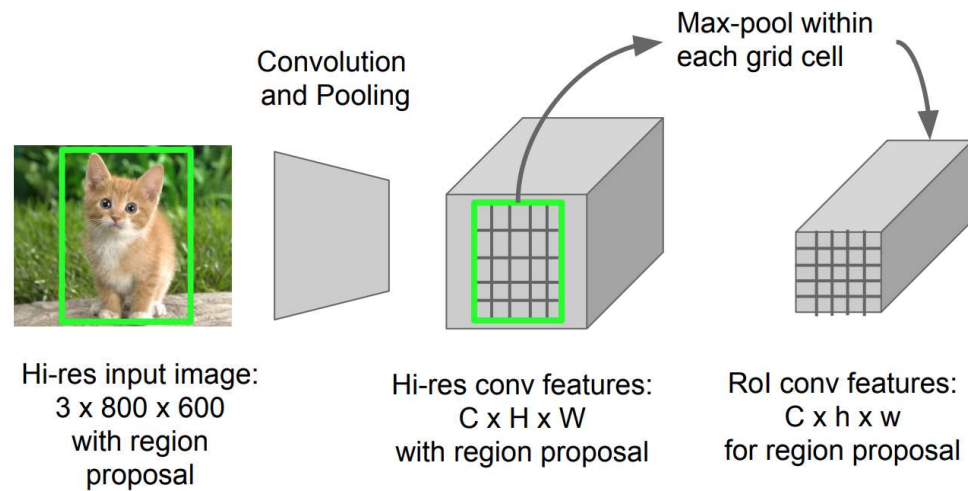
# Fast R-CNN

- Principal problema de RCNN es la caracterización de cada región propuesta con una CNN pre-entrenada
- Solución: dejar que la caracterización se haga en una sola pasada por la red



# Fast R-CNN

- Toda la imagen entra a una CNN, en donde cada región propuesta se proyecta en el feature map de salida.
- Se usa ROI Pooling para generar volúmenes del mismo tamaño por cada región



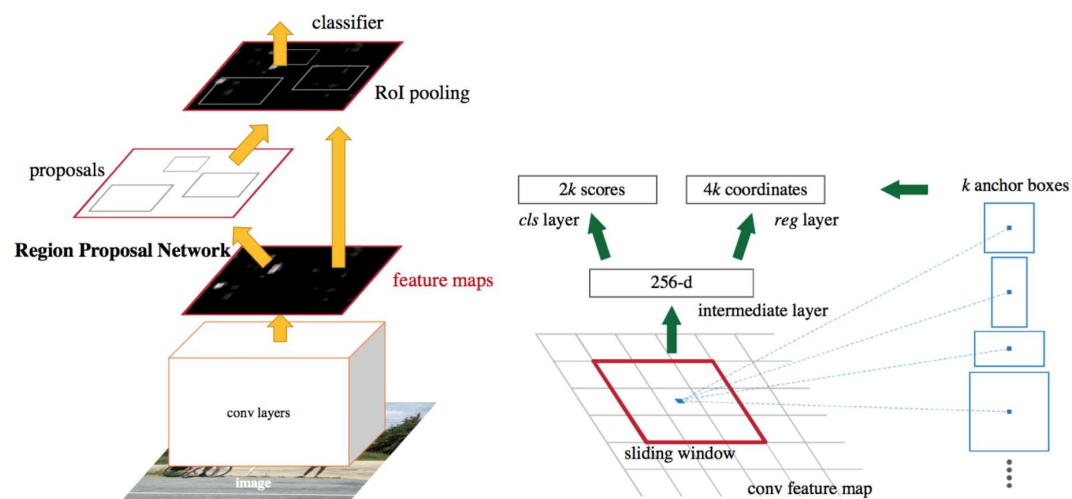
# Faster RCNN

---



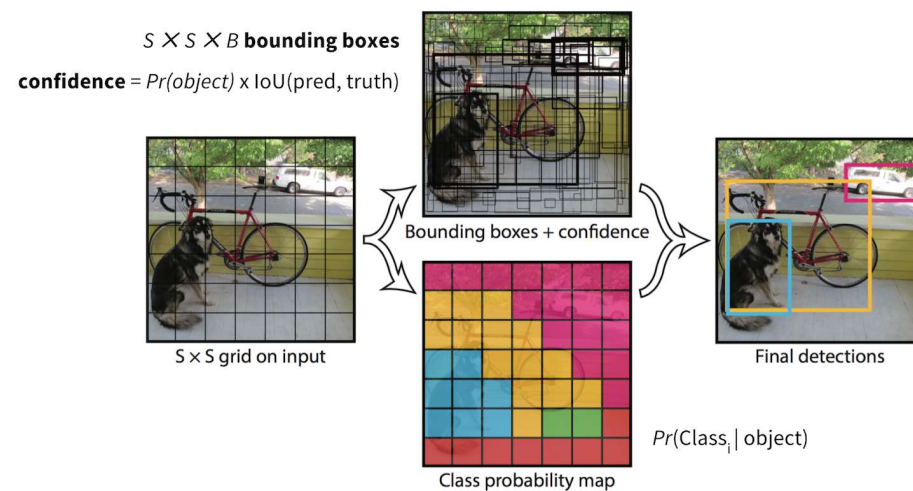
# Faster R-CNN

- En este modelo, se busca integrar el algoritmo de propuesta de regiones en un solo modelo.



# Yolo: You Only Look Once

- Modelo de una sólo pasada. La red realiza la predicción directamente desde la imagen de entrada. No propuestas de regiones.
- Estrategia:
  - Pre-entrenar una CNN para clasificación
  - Partir imagen en  $S \times S$  celdas. Si el centro del un objeto cae en una celda, esa celda es responsable de detectar a ese objeto. Cada celda predice: ubicación de  $B$  bounding boxes, un score de confianza, distribución de probabilidades para clasificación.



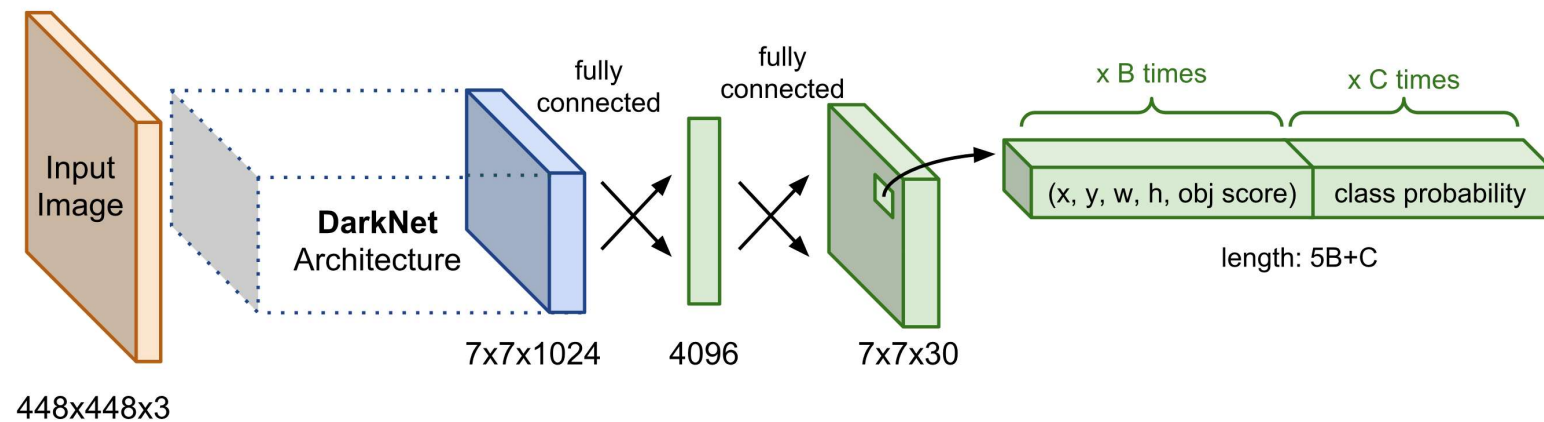
YOLO (You only look once)

---

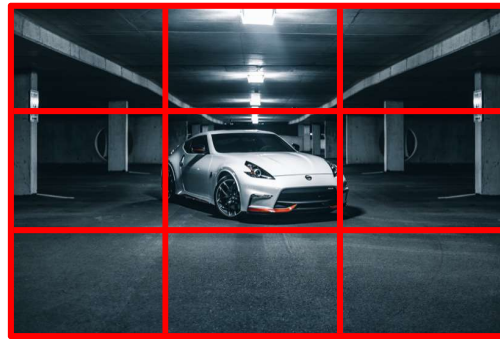


# Yolo: You Only Look Once

- Arquitectura de red
  - Modelo inicial: GoogleNet, módulo Inception modificado (Conv1x1 y Conv3x3)
  - Predicción final: 2 fully connected después del último feature map



# YOLO - Ejemplo



$y$  tiene tamaño  $3 \times 3 \times 2 \times 8$

Cada celda tiene  $B$  anchors ->

Clases

1. Persona
2. Carro
3. Moto

$y =$

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

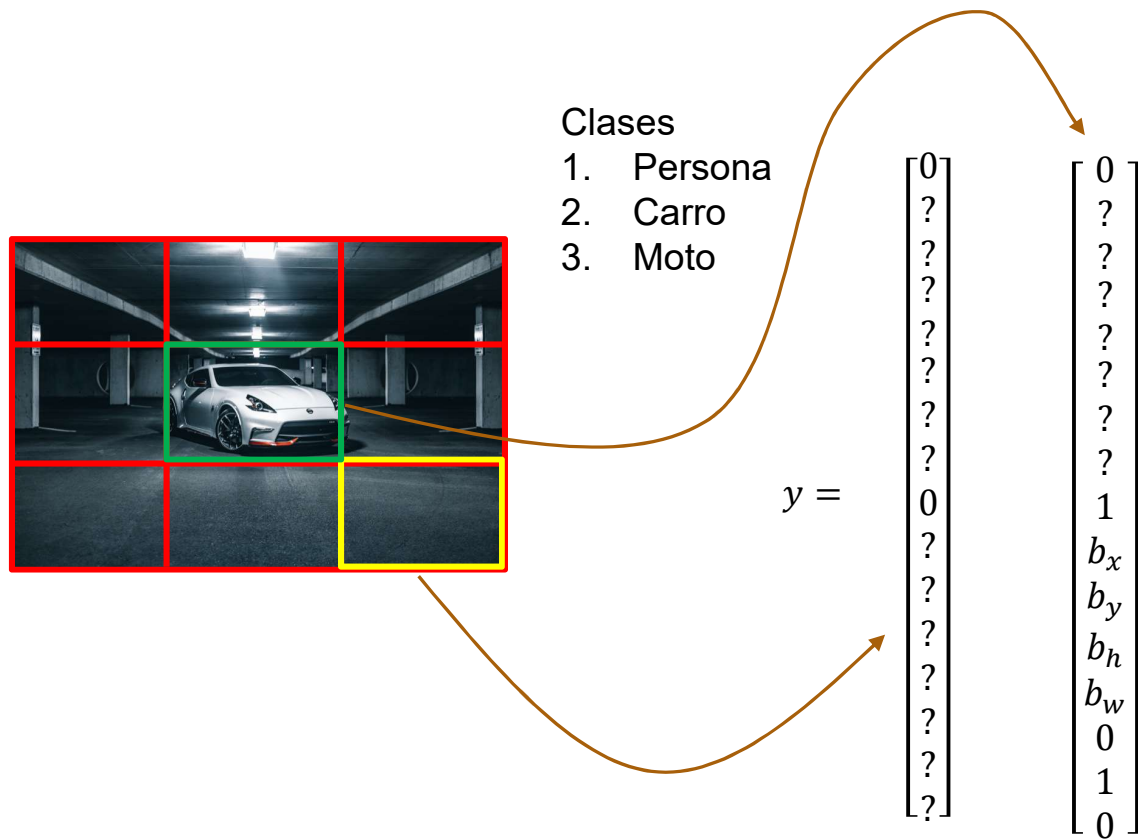
← Hay objeto?

Bounding box

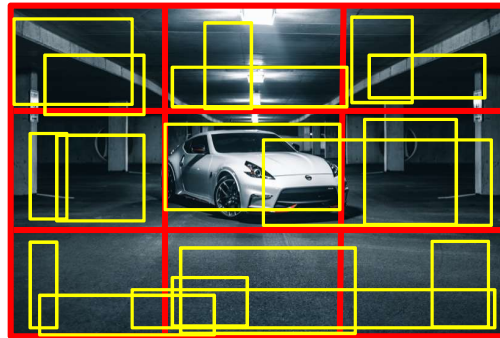
Prob. clases



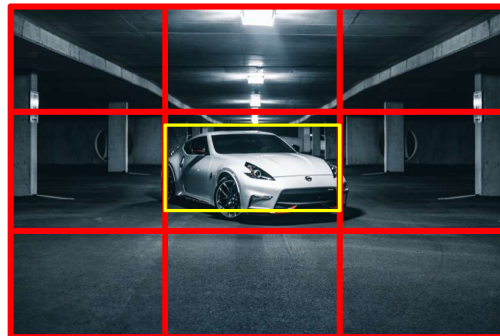
# YOLO - Ejemplo



# YOLO - Postprocesamiento



- Para cada celda, obtener B bounding boxes



- Supresión de no máximos

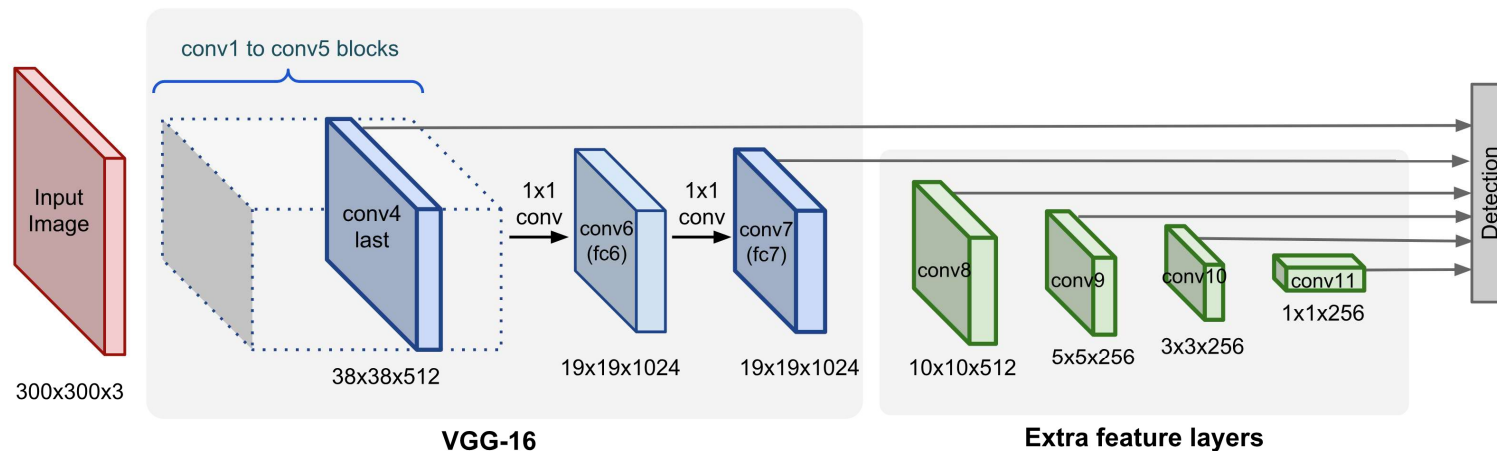
# Single Shot Detection

---



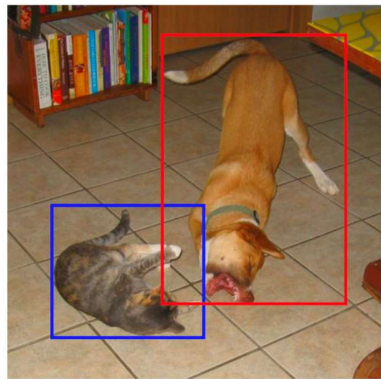
# Single Shot MultiBox Detector (SSD)

- Uno de los principales problemas de métodos de detección es lidiar con objetos de múltiples tamaños (problemas de escala).
- SSD es el primer algoritmo en buscar solución a este problema

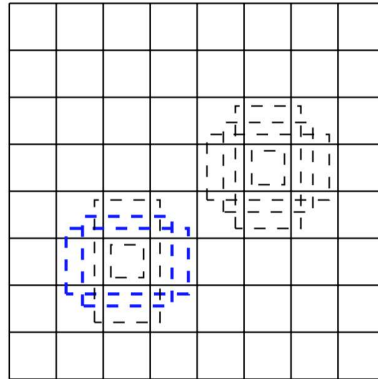


# Single Shot MultiBox Detector (SSD)

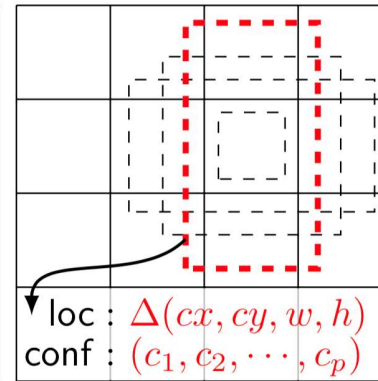
- Cada feature map se hace cargo de detectar objetos de un tamaño respectivo
- Objetos pequeños en primeras capas y objetos grandes en capas profundas



(a) Image with GT boxes



(b)  $8 \times 8$  feature map



loc :  $\Delta(cx, cy, w, h)$   
conf :  $(c_1, c_2, \dots, c_p)$

(c)  $4 \times 4$  feature map

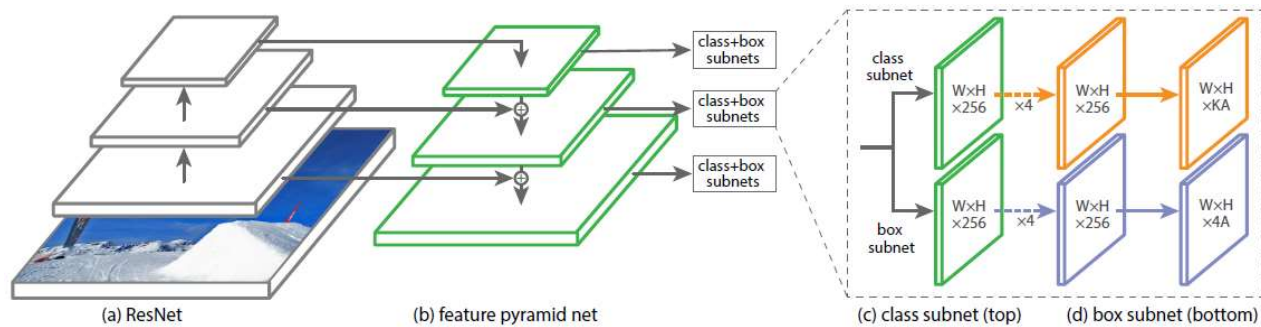
# RetinaNet (Feature Pyramid Network + Focal Loss)

---



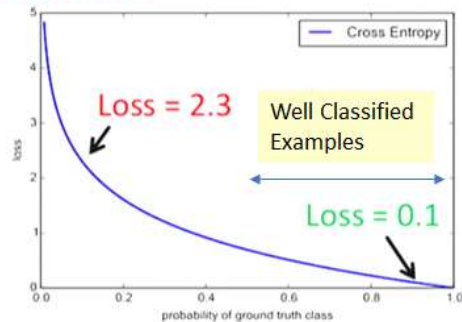
# RetinaNet

- Dos propuestas originales
  - Feature Pyramid Net (FPN)
  - Focal Loss para imbalance de clases



# RetinaNet

- En detectores de una sola pasada, se generan muchas anchors candidatos. Si quieres una detección densa, tienes que generar muchos anchors: la mayoría de ellos con contenido que no son objetos.
- Existe un desbalance de ejemplares negativos con respecto a positivos.
  - 100000 easy : 100 hard examples
  - 40x bigger loss from easy examples



- Si son 100,000 ejemplos fáciles: loss = 10,000
- Si son 100 ejemplos difíciles: loss = 230
- El loss es dominado por los ejemplos fáciles, perjudicando el aprendizaje de ejemplos difíciles

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t).$$



# Comparación

---

Model	PASCAL VOC 2007	PASCAL VOC 2010	PASCAL VOC 2012	COCO 2015 (IoU=0.5)	COCO 2015 (IoU=0.75)	COCO 2015 (Official Metric)	COCO 2016 (IoU=0.5)	COCO 2016 (IoU=0.75)	COCO 2016 (Official Metric)	Real Time Speed
R-CNN	x	62.4%	x	x	x	x	x	x	x	No
Fast R- CNN	70.0%	68.8%	68.4%	x	x	x	x	x	x	No
Faster R-CNN	78.8%	x	75.9%	x	x	x	x	x	x	No
R-FCN	82.0%	x	x	53.2%	x	31.5%	x	x	x	No
YOLO	63.7%	x	57.9%	x	x	x	x	x	x	Yes
SSD	83.2%	x	82.2%	48.5%	30.3%	31.5%	x	x	x	No
YOLOv2	78.6%	x	x	44.0%	19.2%	21.6%	x	x	x	Yes

# Instance Segmentation

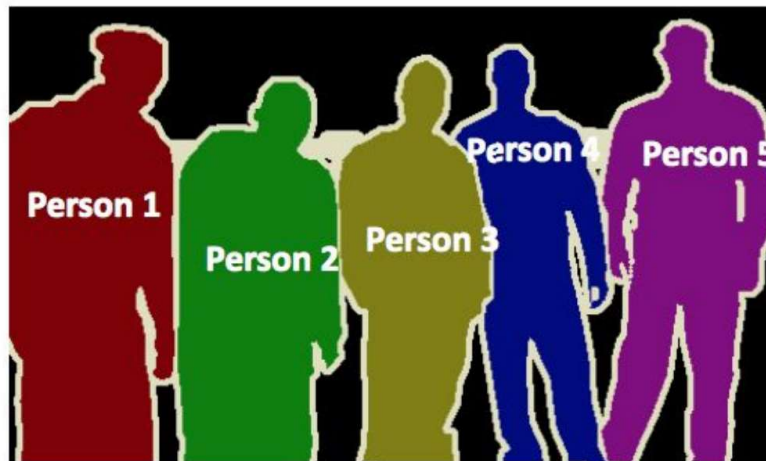
---



# Objetivo

---

- Etiquetar cada píxel con su correspondiente objeto detectado
- Combinación de Detección + Segmentación



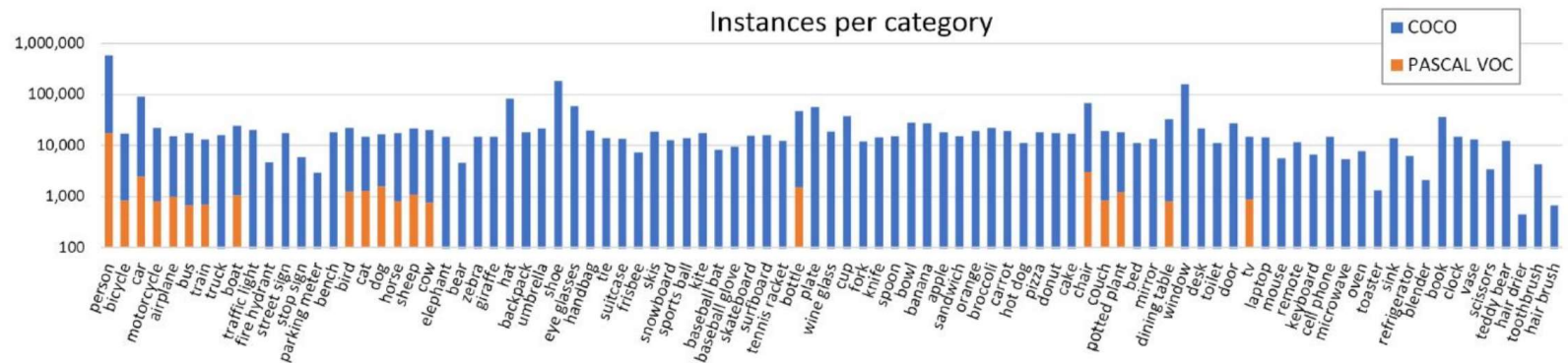
**Instance Segmentation**

# COCO Dataset

- Clases: 91
- +300,000 imágenes
- +2.5M de instancias etiquetadas

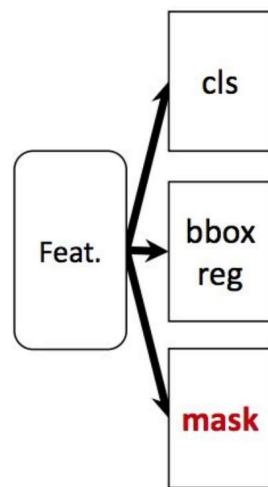


# COCO dataset

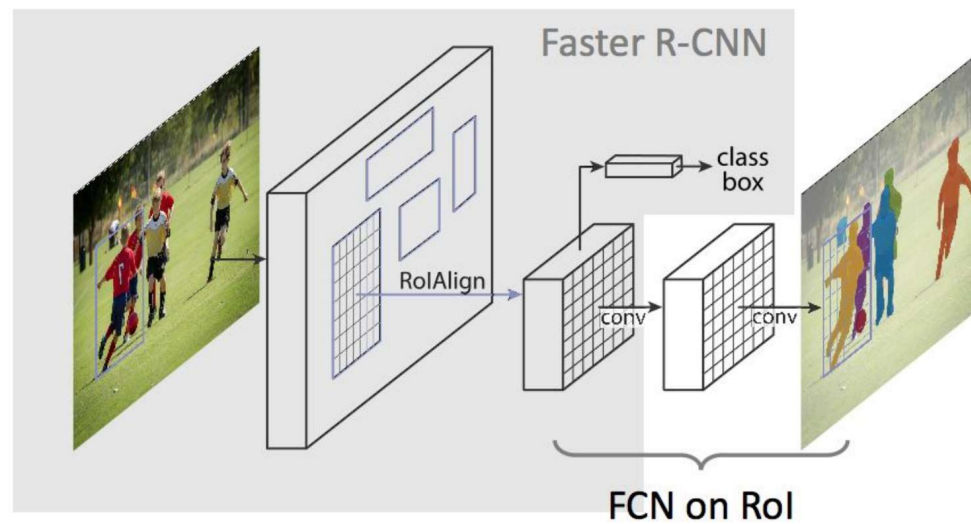


# Mask-RCNN

- Es una extensión simple a Faster RCNN.
- Se infiere máscaras para cada objeto.



Mask R-CNN



# Ejemplos

---

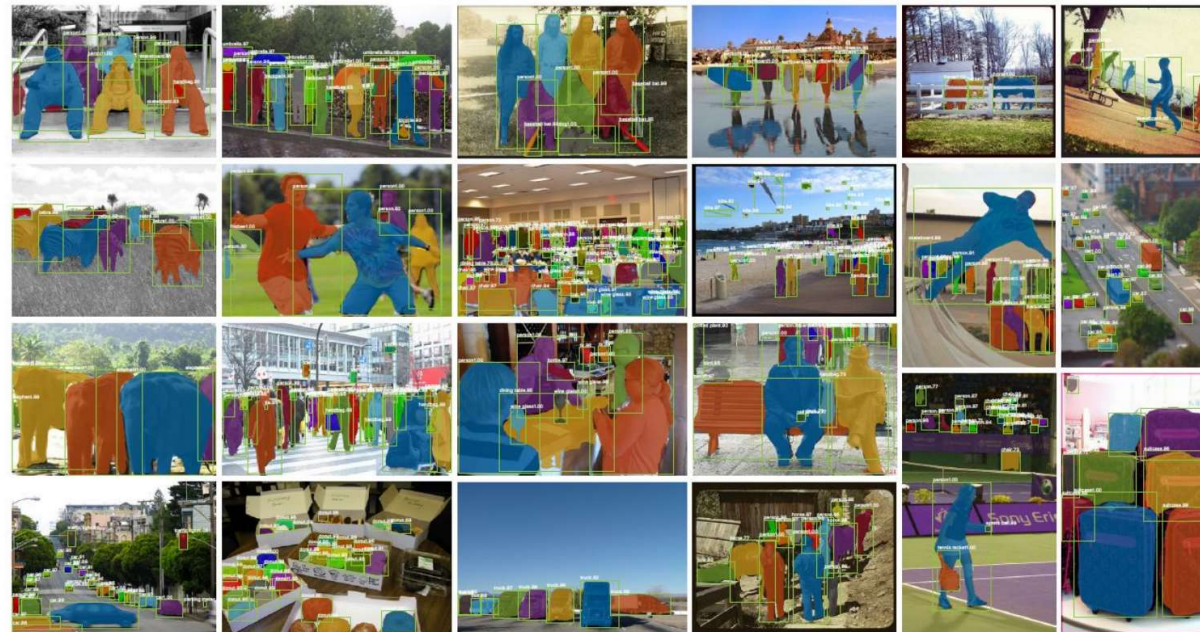


Figure 5. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).



# Aplicaciones

---

## Detección de pose humana





# Qué hay de nuevo en este tema?

---

- Facebook liberó su sistema de detección

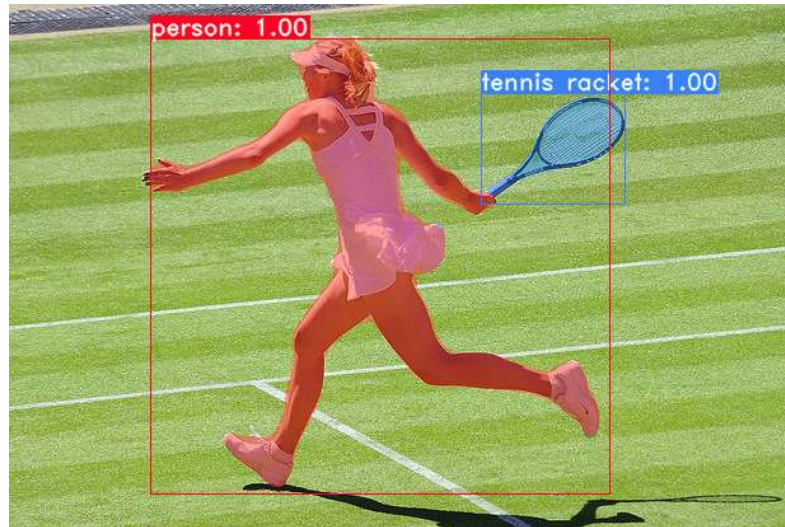


<https://github.com/facebookresearch/detectron>

# Qué hay de nuevo en este tema?

---

- Instance Segmentation en tiempo real: YOLACT



<https://github.com/dbolya/yolact>

# Qué hay de nuevo en este tema?

---

- Detección de poses en humanos: HRNet



<https://github.com/leoxiaobin/deep-high-resolution-net.pytorch>

# Qué hay de nuevo en este tema?

---

- Detección de poses en humanos: DensePose



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



# Qué hay de nuevo en este tema?

---

- Reconstrucción 3D de poses humanas: VIBE

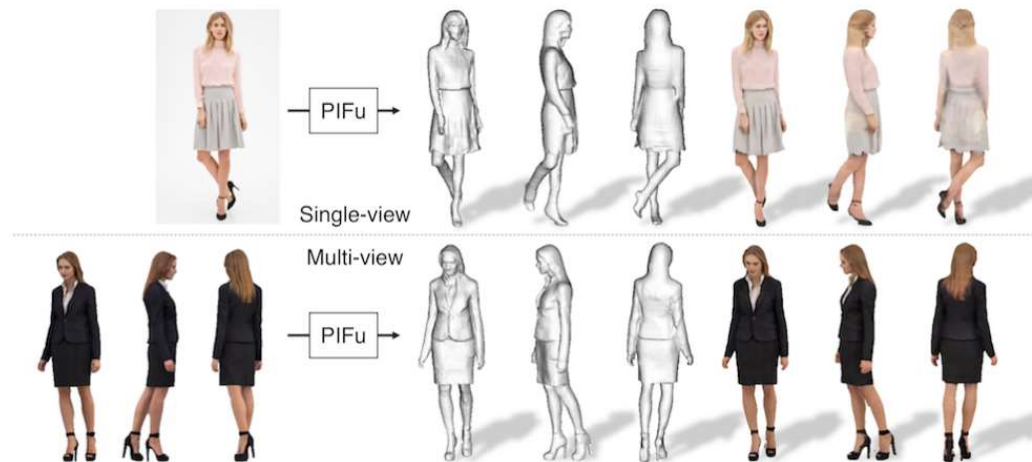


<https://github.com/mkocabas/VIBE>

# Qué hay de nuevo en este tema?

---

- Reconstrucción 3D de poses humanas: PiFu



<https://shunsukesaito.github.io/PIFu/>