

Redes de detección

Índice

1. Antecedentes	2
1.1. Localización como regresión	2
1.2. Detección como clasificación	2
1.3. Clasificación con preselección de regiones prometedoras	3
1.4. Medidas de rendimiento	5
2. You Only Look Once (Yolo)	7
3. Single Shot Detection (SSD)	10
4. Ejercicios prácticos	12

Referencia principal

☞ Z.Q. Zhao, S.T. Xu, X. Wu: *Object Detection with Deep Learning: A Review*

Tiempo estimado = 1 sesión de 2 horas (1 semana)

1. Antecedentes

La tarea supervisada por excelencia es la clasificación, que en el contexto de visión artificial se suele denominar “reconocimiento”. Sin embargo, reconocer algo en una imagen puede ser insuficiente según el problema al que nos enfrentemos. Por ejemplo, podemos reconocer a un delincuente en una foto con mucha gente, pero ¿cuál de todos ellos es?

La visión artificial da una vuelta de tuerca al problema del reconocimiento y se plantea dos tareas adicionales: la localización y la detección.

1.1. Localización como regresión

Localizar es devolver la posición (x, y) , y el ancho y alto (w, h) del *bounding box* (bbox de aquí en adelante) del objetivo, si este aparece en pantalla.

Este problema se puede replantear como uno de regresión. Dado un conjunto de imágenes, tal que en todas ellas aparece el objetivo, y todas ellas etiquetadas con una 4-tupla (x, y, w, h) que representa el bbox del objetivo, podemos construir una red convolucional con un cabezal regresor de 4 salidas para estimar el dicho bbox. Este proceso se muestra en la Figura 1.

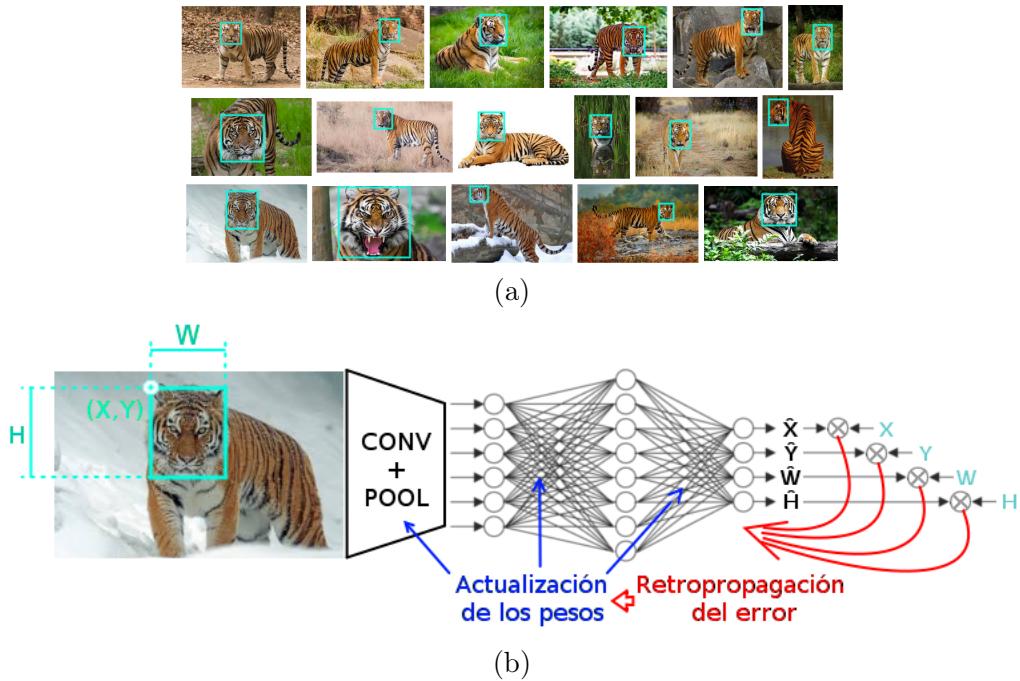


Figura 1: (a) Conjunto de entrenamiento para localización. (b) Aprendizaje en localización

1.2. Detección como clasificación

La localización tiene una limitación: sólo es aplicable cuando hay un único objetivo en la imagen. La generalización a N objetivos diferentes es la detección. En otras palabras, la detección es la tarea que devuelve el bbox de cada uno de los objetos reconocidos en la imagen.

Como primera aproximación, podemos pensar en abordar este problema como una clasificación. Ya que las redes convolucionales demuestran ser muy eficaces en reconocimiento, podríamos utilizar una ventana deslizante multi-resolución. Cada una de estas ventanas es un bbox candidato, del cual la red devuelve una clase con una cierta probabilidad. Al terminar el barrido nos encontramos con una multitud de bboxes, que debemos filtrar mediante umbralizado y clustering.

Esta es, en esencia, la solución del método Overfeat, propuesta por Sermanet et al. en 2013 ↗.

En ese trabajo la red que se utiliza es una AlexNet para la extracción de características, y dos FCNNs; una con cabezal clasificador para estimar la clase del bbox, y la otra con cabezal regresor para refinar el bbox dentro de la ventana deslizante actuando como una red de localización (es decir actúa como una red de localización pero sólo dentro de la ventana deslizante). Overfeat se entrenó con ImageNet 2012, que contiene 1.2 millones de imágenes de 1000 clases diferentes). Además se añade una clase “fondo”. Para reducir el coste computacional, la ventana es siempre de la misma resolución, y se utilizan 6 tamaños diferentes de la misma imagen.

En la figura 2 se muestra un detalle del proceso de entrenamiento, donde se han producido dos clases diferentes (amarilla y roja) en algunas de las ventanas.

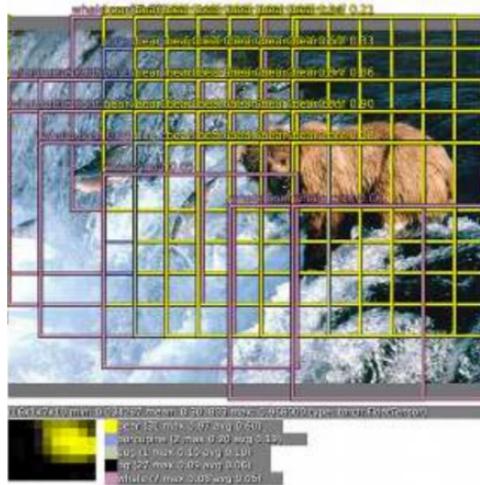


Figura 2: Paso de entrenamiento en Overfeat.

Overfeat tuvo buenos resultados en el *ImageNet Large Scale Visual Recognition Competition* (ILSVRC), pero actualmente es un proyecto abandonado porque no funciona en tiempo real.

1.3. Clasificación con preselección de regiones prometedoras

Overfeat tiene dos problemas principales:

1. El coste computacional de utilizar una ventana deslizante.
2. Supongamos que la red de clasificación acierta 99 de cada 100 veces, lo cual está, a priori, muy bien. Si tenemos una imagen de 500×500 y utilizamos una ventana de 50×50 con un *stride* de 5, entonces tenemos que clasificar, aproximadamente, 10.000 ventanas. Por la precisión del clasificador, es probable que obtengamos 100 fallos. Esto significa que necesitaremos un proceso de filtrado y corrección posterior que será, de nuevo, costoso.

Sin abandonar la detección como un problema de clasificación, para aliviar este problema necesitamos reducir el número de ventanas a clasificar. Esto se puede lograr haciendo una propuesta de regiones de interés, en forma de bbox, previa a la clasificación. A partir de esta idea surge la solución R-CNN, propuesta por Girshick et al. en 2013 ↗.

R-CNN propone hacer una preselección de ventanas mediante “Busqueda Selectiva”, método propuesto por Uijlings et al. en 2012 ↗ que esencialmente consiste en hacer una segmentación rápida de la imagen agrupando los píxeles atendiendo al color, textura, forma, etc. del cluster resultante. El proceso itera varias veces de manera que obtenemos una jerarquía de regiones. De esta manera se obtienen muchas propuestas, aunque muchas menos que con Overfeat, pero con mucha probabilidad de contener “algo”. Es decir tendremos muchos Falsos Positivos, pero un Recall muy alto; o sea que la probabilidad de NO detectar los objetos de interés es muy baja.

Este proceso se muestra en la Figura 3.

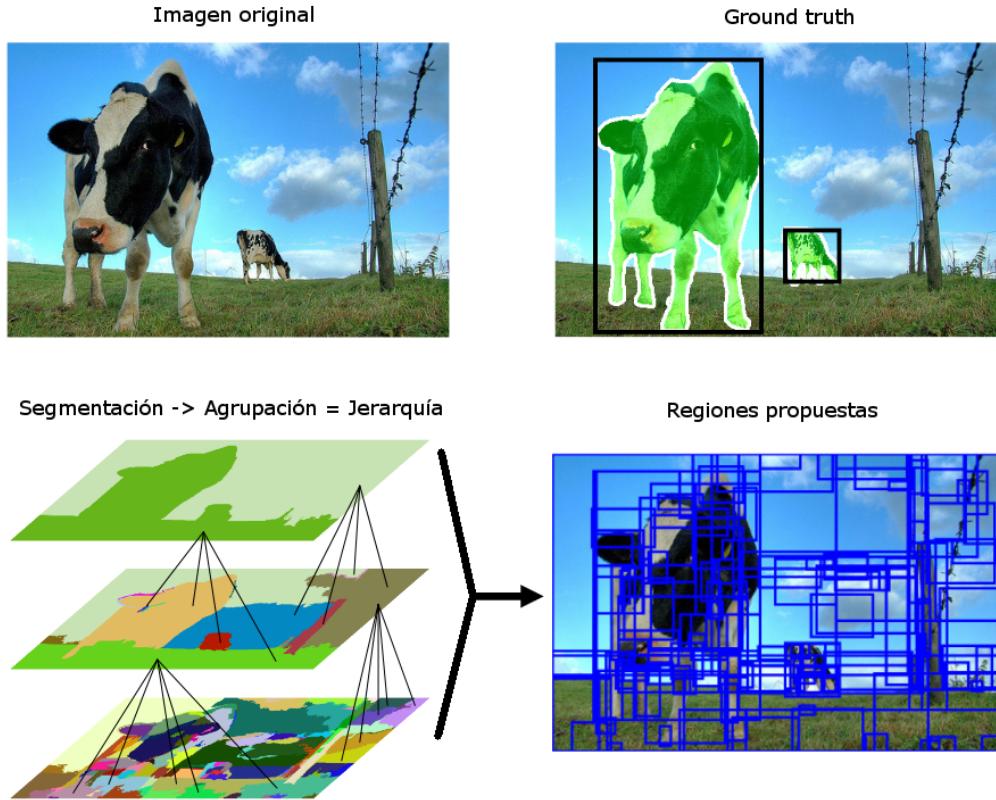


Figura 3: Obtención de las regiones para R-CNN

A partir de ese punto, se retoma la idea de utilizar una CNN seguida de una FCNN pero ya de un sólo cabezal clasificador, puesto que las regiones llevan el bbox ajustado, tal y como se muestra en la Figura 4.

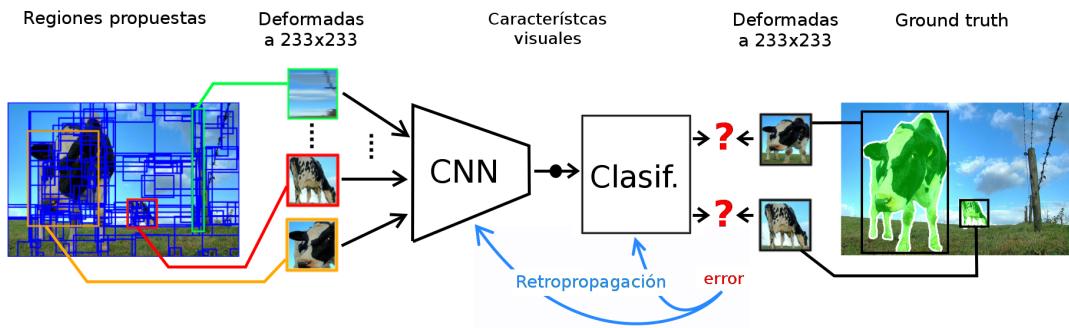


Figura 4: Aprendizaje de una R-CNN

Respecto de Overfeat, R-CNN tiene dos ventajas. Dado que evalúa menos ventanas, aumenta su rapidez de respuesta, aunque sigue sin llegar a ser tiempo real. Además, el bbox se ajusta mejor al objeto sin necesidad de un cabezal regresor.

Por estos motivos, la propuesta ha tenido más recorrido en el tiempo y mejoras continuas, logrando reducir su tiempo de ejecución.

Fast-RCNN es la primera de estas propuestas, presentada en el ICCV de 2015 por Girshick . En esta red la imagen entra directamente a una CNN que genera un mapa de características, sobre el cual se ejecuta el algoritmo que propone regiones. Después cada una de estas regiones

es reescalada y clasificada. Es decir que esencialmente se han intercambiado las posiciones de la propuesta de regiones y la CNN, respecto a R-CNN. Esta solución es mucho 25 veces más rápida que R-CNN. El motivo es que las regiones se proponen sobre el mapa de características, que suele ser más pequeño que la imagen original.

Faster-RCNN Ren et al. (2015) ↗ es la segunda mejora, que a su vez es superior también a Fast-RCNN. La diferencia es que la propuesta de regiones la hace otra red (*Region Proposal Network*, RPN), mucho más rápida que el algoritmo utilizado antes. RPN trabaja sobre 9 regiones prefijadas de antemano en tamaño y aspecto por cada píxel de la imagen seleccionado denominadas “anclas”. Por ejemplo, si en una imagen de 600×800 tomamos 1 píxel cada 16, tanto en vertical como en horizontal, tenemos $39 \times 51 = 1989$ píxeles, que dan lugar a $1989 \times 9 = 17901$ anclas. Para cada ancla nos preguntamos: ¿contiene algún objeto relevante?, ¿de qué manera deberíamos modificarla para que se ajuste mejor al objeto relevante? La primera pregunta se puede responder con una red de clasificación, y la segunda con una de regresión; y la combinación de ambas respuestas es un conjunto de regiones prometedoras. Es decir que utilizando una red con dos cabezales (la RPN) podemos convertir las anclas en regiones prometedoras sobre las que actuar del mismo modo que se hacía con la Fast-RCNN. En definitiva, hemos logrado construir un sistema de detección basado únicamente en redes.

La Faster-RCNN no sólo es la primera arquitectura enteramente neuronal propuesta para detección, también logra buenos resultados reduciendo el tiempo de respuesta a décimas de segundo.

1.4. Medidas de rendimiento

En la tarea de detección, como en cualquier otra, podemos acertar o fallar en nuestras predicciones. Podemos fallar de dos maneras:

1. Devolviendo un bbox de una clase pero o bien no hay nada o es de otra clase, es decir generando bbox donde no debería, y por tanto dando Falsos Positivos (FP)
2. No detectando un objeto que sí debería, es decir perdiendo detecciones, y por tanto dando Falsos Negativos (NP)

Sin embargo sólo podemos acertar de una manera: generando el bbox correcto para un objeto de la imagen, o sea Verdaderos Positivos (*True Positives*, TP).

Por tanto no podemos usar aquellas métricas derivadas que involucren los Verdaderos Negativos (*True Negatives*, TN). La Figura 5 muestra las permitidas y las que no.

Pero ¿cómo calificaríamos un bbox que acierta la posición pero se queda un poco corto o un poco largo al estimar el tamaño? ¿Es un fallo o un acierto? Por este motivo, en detección se mide el cociente entre la intersección del bbox con el ground truth y la unión de ambas, denominado IoU (*intersection over union*). En la figura 6(a) se muestra esta medida.

La decisión sobre si una detección es un TP se toma umbralizando su IoU. Habitualmente, si $\text{IoU} > 0.5$ entonces se considera que dicha detección es un TP; y en caso contrario es un FP.

Por tanto, en detección hay dos umbrales que dan lugar al recuento de TP, FP y FN: el del IoU y el umbral con el que decidimos si un bbox candidato es una detección o no, es decir la probabilidad de detección que arroja la red neuronal. Este segundo es muy dependiente del modelo, es decir que una probabilidad del 0.8 puede ser muy alta en algunos casos y muy baja en otros.

Sin embargo, podemos buscar qué probabilidad hace falta para tener un Recall de 0, de 0.1, de 0.2, etc. Después, con esos valores de probabilidad que hemos obtenido, podemos calcular el Precision. Así, se define el *mean Average Precision* (mAP) como el promedio de los valores de precisión obtenidos de esta manera, y es la medida más frecuente para compararse en problemas

de detección. La Figura 6(b) se muestra una comparativa de las diferentes propuestas de RCNN respecto del tiempo de ejecución, el speedup y el mAP en el conjunto de datos VOC 2007.

		Condition (as determined by "Gold standard")			
		Total population	Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$
Test outcome	Test outcome positive	True positive	False positive (Type I error)	Positive predictive value (PPV, Precision) = $\frac{\sum \text{True positive}}{\sum \text{Test outcome positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Test outcome positive}}$
	Test outcome negative	False negative (Type II error)	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Test outcome negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Test outcome negative}}$
	Positive likelihood ratio ($LR^+ = TPR/FPR$)	True positive rate (TPR, Sensitivity, Recall) = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR, Fall-out) = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$	
	Negative likelihood ratio ($LR^- = FNR/TNR$)	False negative rate (FNR) = $\frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	True negative rate (TNR, Specificity, SPC) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$		
	Diagnostic odds ratio ($DOR = LR^+/LR^-$)				

Figura 5: Métricas derivadas de la matriz de confusión que se pueden utilizar en detección

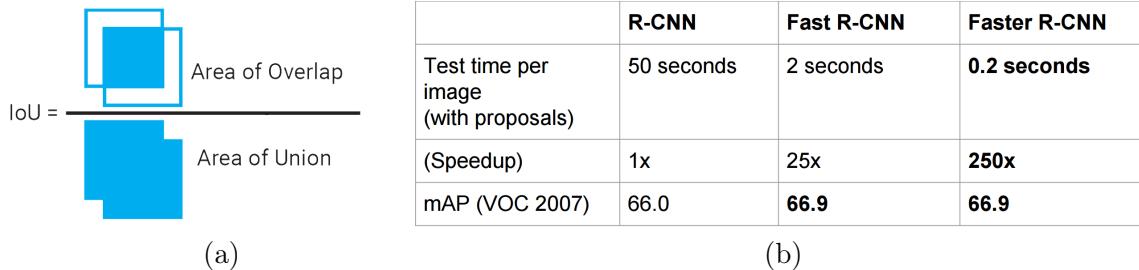


Figura 6: (a) Medida IoU para estimar la precisión de una detección. (b) Comparativa de R-CNNs

2. You Only Look Once (Yolo)

Yolo (J. Redmon *et al.*, et al, CVPR'16 [\[2\]](#)) tuvo un gran impacto en tareas de detección porque presentaba una aproximación muy diferente a lo realizado hasta el momento, logrando ser la primera propuesta basada en redes profundas que funcionó en tiempo real.

En Yolo, la imagen original se divide en un cierto número de celdas. Cada una de estas celdas se introduce en la red profunda, que predice la clase a la que pertenece dicha región y el bbox del objeto completo al que pertenece. Por tanto esta red profunda aúna tareas de regresión y clasificación. En cierto sentido es como volver a la solución del problema de localización, pero en vez de utilizar toda la imagen utilizamos las celdas creadas, por lo que no estamos restringidos a un único objeto en la imagen. Finalmente los bbox obtenidos de esta manera se fusionan.

En la Figura 7 se muestra este proceso en dos etapas. En la de arriba se puede que la imagen se divide en 7×7 celdas. La red de regresión toma cada celda y devuelve el bbox estimado del objeto completo junto con la confianza en dicha predicción. Si por ejemplo hace 2 predicciones por cada bbox, el resultado son 98 bbox. En la de abajo se observa como cada celda produce, además una distribución de masa de probabilidad sobre las clases posibles.

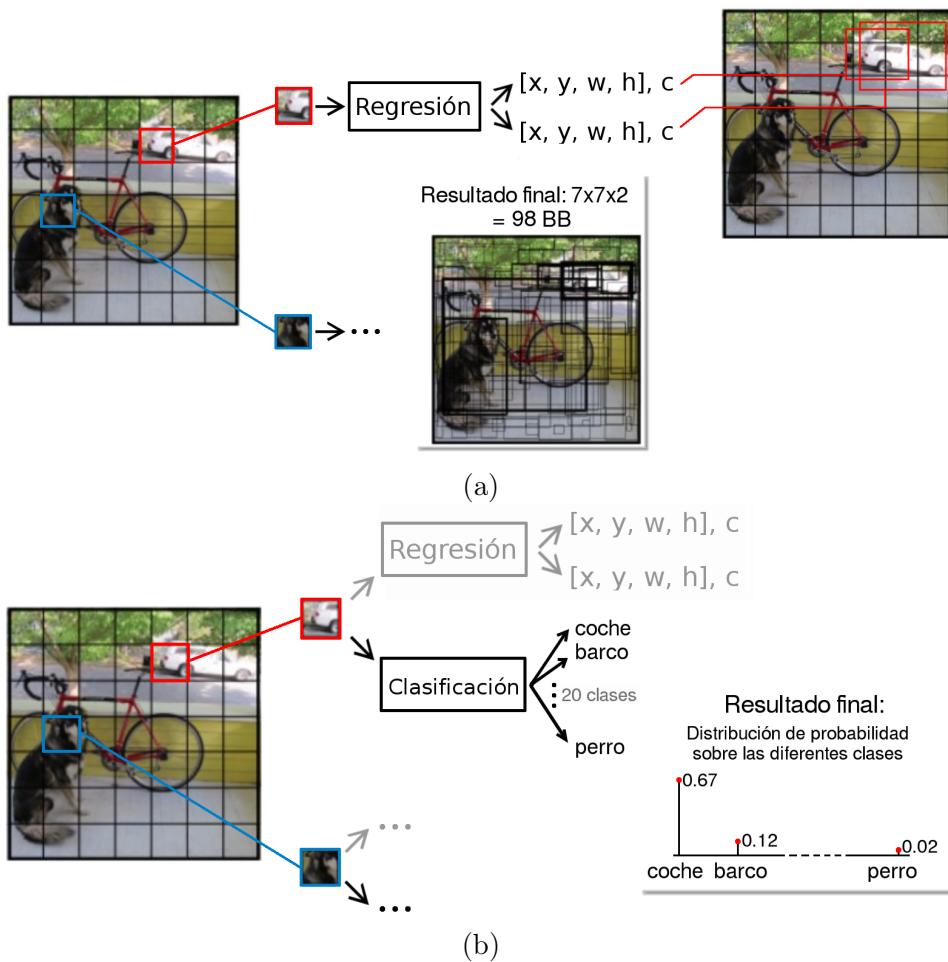


Figura 7: (a)Red de regresión de Yolo. (b)Red de clasificación de Yolo.

Por tanto, cada celda genera un vector de $5 \cdot B + C$ elementos, donde B es el número de bbox que predice la celda (2 en el ejemplo de la Figura 7) y C es el número de clases; de manera que la imagen entera genera un tensor de salida de dimensiones $S \times S \times (5B + C)$, donde $S \times S$ es el número de celdas, en nuestro ejemplo 7×7 . En la Figura 8 se muestra el tensor de salida de la red que estamos usando de ejemplo.



Figura 8: Tensor de salida de Yolo

La arquitectura de Yolo no es tan determinante como lo que acabamos de explicar. En la Figura 9 se puede ver que la arquitectura original no tiene demasiadas complicaciones.

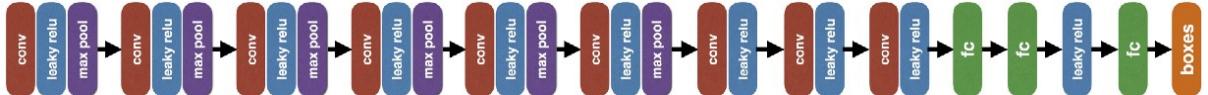


Figura 9: Arquitectura de la red profunda que implementa Yolo

Aunque la confianza c es producida por la red de regresión, formalmente la podemos definir como

$$c = \Pr(\exists \text{Objeto}) \cdot IoU,$$

es decir que si en la celda en cuestión no hay ningún objeto la confianza de detección debería ser cero, y si hay un objeto dependerá de lo precisa que sea la celda propuesta respecto de su bbox.

Multiplicando la confianza por la distribución generada por la celda, obtenemos una medida que combina la probabilidad de cada clase en ese bbox cuando este contiene efectivamente un objeto, y la exactitud del bbox respecto al *ground truth*:

$$\Pr(Clase|\exists \text{Objeto}) \cdot c = \Pr(Clase|\exists \text{Objeto}) \cdot (\Pr(\exists \text{Objeto},) \cdot IoU) = \Pr(Clase, \exists \text{Objeto}) \cdot IoU,$$

pero si no hay objeto, $c = 0$, por lo que el último término se puede reducir, en la práctica, a $\Pr(Clase) \cdot IoU$. De esta manera, para cada celda tenemos un array de C elementos con la medida

$$\Pr(Clase) \cdot IoU.$$

Aquellas que no superen un umbral (por ej. 0.2) se igualan a cero; y después se eliminan aquellas que sean redundantes de acuerdo al algoritmo NMS (*Non-Max Supression*). En la Figura 10 se muestra un resultado.

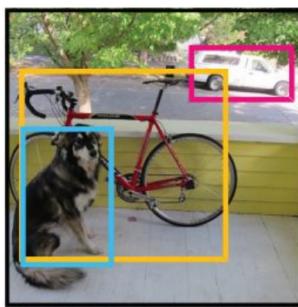


Figura 10: Resultado de Yolo

La función de coste utilizada por Redmon et al. incorpora varios términos que, para simplificar su explicación, se muestran desglosados en la Figura 11. El primer término mide el error en la posición del bbox, el segundo el error en las dimensiones del bbox, el tercero la confianza arrojada por ese bbox cuando efectivamente hay un objeto, el cuarto es similar al tercero, pero cuando no hay un objeto, y el quinto es la diferencia entre la distribución de masa de probabilidad sobre las clases¹.

¹Para comparar distribuciones de probabilidad hay pérdidas más apropiadas, pero esta es la que se utilizaba en el artículo original.

Para facilitar la comprensión de la Figura 11 se ha utilizado el siguiente código de colores:

- Peso del componente
- Componentes x, y, w, h, c de cada $bbox$ predicho
- Componentes x, y, w, h, c del ground truth
- probabilidad de la clase asignada a cada celda en el groun truth
- probabilidad de la clase estimada a cada celda

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

Figura 11: Función de pérdida de Yolo

Yolo sacrifica precisión para ganar rapidez, funciona en tiempo real, 8x respecto a Faster R-CNN. El motivo de la falta de precisión es que falla cuando dos objetos tienen su centro en la misma celda. Esto ocurre cuando tienen tamaños y orientaciones diferentes, de manera que son visualmente detectables (es decir que no oculta el de delante al de detrás). Por ejemplo en la Figura 12 o detecta el saxo o al saxofonista, y lo mismo con la guitarra y el guitarrista.

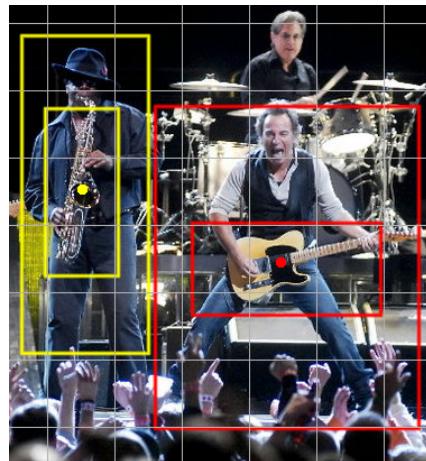


Figura 12: Imagen con 4 objetos, de los cuales Yolo sólo podría detectar 2.

3. Single Shot Detection (SSD)

Las limitaciones de la primera versión de Yolo se han resuelto en las versiones posteriores utilizando anclas. Esta idea, introducida en Faster-RCNN también está en la concepción de *Single Shot Detection* (SSD). Esta aproximación, además tiene otras contribuciones importantes:

- No se divide la imagen original, sino los mapas de características que se van obteniendo tras etapas convolucionales → se trabaja a diferentes resoluciones.
- Cada celda puede predecir un cierto número de cajas llamadas “priors”, similares a las anclas en Faster R-CNN.
- Las etiquetas del *ground truth* deben incluir la representación *one-hot* de la clase a la que pertenece cada bbox.

Por ejemplo, en la Figura 12, y suponiendo que cada celda tiene 2 priors asociados, la SSD produciría el resultado que se muestra en la Figura 13

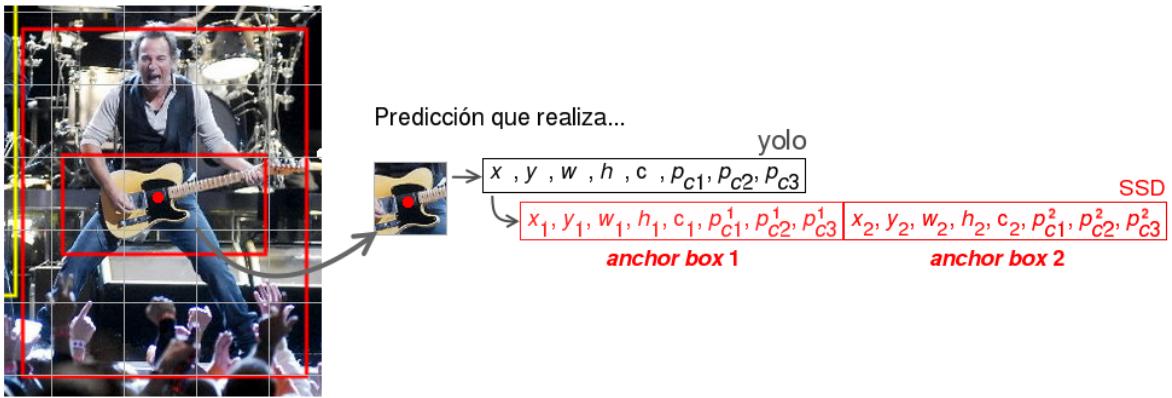


Figura 13: Comparativa de la predicción de Yolo frente a SSD

Vamos a suponer que los objetos a detectar pueden ser de 3 clases: $\{c_1=\text{persona}, c_2=\text{saxo}, c_3=\text{guitarra}\}$. Para la celda seleccionada en la Figura 13, el bbox de cada objeto es:

- $[x_1, y_1, w_1, h_1, c_1, p_{c1} = 1, p_{c2} = 0, p_{c3} = 0]$
- $[x_2, y_2, w_2, h_2, c_2, p_{c1} = 0, p_{c2} = 0, p_{c3} = 1]$

Por tanto la etiqueta, en el *ground truth*, sería

$$[x_1, y_1, w_1, h_1, c_1, p_{c1} = 1, p_{c1} = 0, p_{c1} = 0 ; x_2, y_2, w_2, h_2, c_2, p_{c1} = 0, p_{c2} = 0, p_{c3} = 1].$$

Si en una celda sólo hay un objeto los valores (x, y, w, h, c) de las demás anclas son irrelevantes. Por ejemplo, si la celda está asociada a una persona y hay dos anclas por celda, la etiqueta del *ground truth* sería:

$$[x_1, y_1, w_1, h_1, c_1, p_{c1} = 1, p_{c1} = 0, p_{c1} = 0 ; *, *, *, *, *, p_{c1} = 0, p_{c2} = 0, p_{c3} = 0]$$

El aprendizaje se realiza con una CNN de varias etapas convolucionales. Recordar que cada etapa da como resultado un mapa de características, de una cierta resolución y profundidad. Cuanto más cercana esté la etapa a la imagen original, mayor será su resolución. Es decir que 1 píxel en un mapa de características muy lejano se corresponderá con una vecindad $n \times n$ en la imagen.

Si lanzamos el algoritmo de detección en los mapas de características en vez de la imagen original estaremos haciendo detección a diferentes escalas. Para ello, cada celda de un mapa de características predice k bbox **predefinidas** (“priors”) de diferentes tamaños y proporciones.

Si un prior es una 4-tupla $[x, y, w, h]$ y 1 mapa tiene $m \times n$ celdas, entonces cada celda del mapa predice k priors y la distribución sobre p clases posibles, entonces la predicción que genera dicho mapa es un tensor $m \times n \times k(4 + p)$.

Por ejemplo, la Figura 14 se muestran algunos priors en dos mapas de características, el primero de resolución 8×8 y el segundo 4×4 . El tamaño de las celdas de cada mapa es relativo a la imagen original y lo mismo con los priors que proponen. De esta manera, los priors azules, en el mapa 8×8 son los mejores para detectar al gato, mientras que los rojos, en el mapa 4×4 son los mejores para detectar al perro.

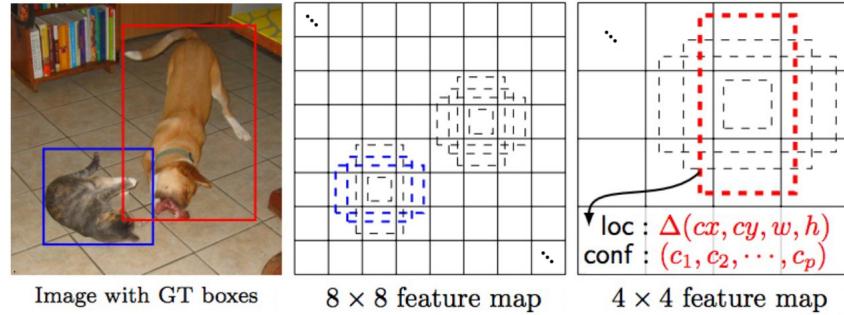


Figura 14: Priors a diferente resolución en SSD

Por último, la Figura 15 muestra la arquitectura de la red SSD, donde se puede apreciar la detección multiresolución en las diferentes etapas convolucionales.

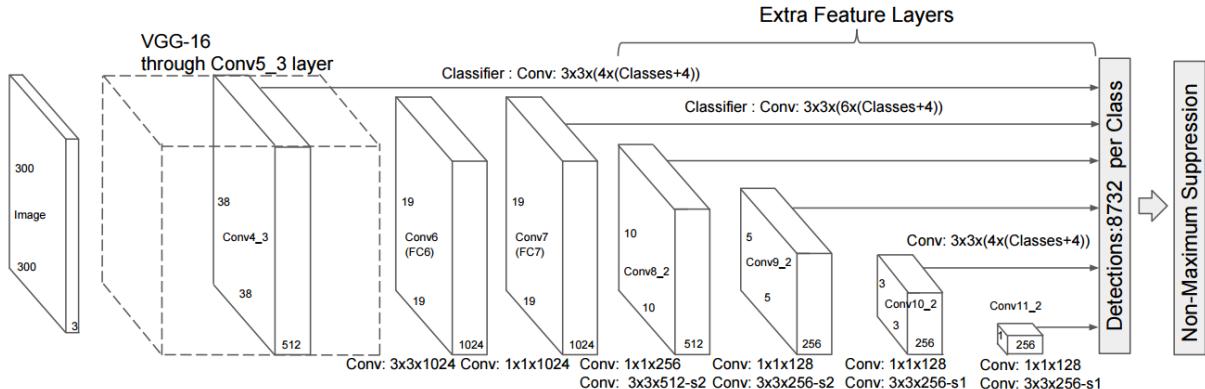


Figura 15: Arquitectura de SSD

Por ejemplo, la primera etapa consiste en una VGG-16 que genera un 512 mapas de características de 38×38 . Estos mapas de características son la entrada de un clasificador-regresor construido únicamente con capas convolucionales, de nucleo 3×3 , que predice 21 clases más los 4 atributos del bbx de cada una de los 4 priors.

4. Ejercicios prácticos

- Construir una red de detección para el conjunto de datos BCCD (*Blood Cell Detection*, disponible en GitHub ↗).
- Descargar una red de detección preentrenada y adaptarla utilizando transfer learning para el mismo conjunto de datos.