Implementation of an object detection algorithm with YOLO

Laura Camila Duque Vinasco - 816020

Análisis Numérico
Universidad Nacional de Colombia - Sede Manizales
Octubre 2020

***Resumen*—En este documento se pretende registrar el proceso de búsqueda, selección e implementación de un proyecto de detección y rastreo de objetos que se encuentre abierto al público con el fin de replicar sus resultados a corto-medio plazo en una aplicación de tiempo real, además de plantear la posibilidad de añadirle un sistema que permita estimar las distancias espaciales entre las detecciones para monitorear que se cumpla el distanciamiento social.**

I. Introducción

Los seres humanos tenemos la capacidad de ver una imagen e identificar casi instantáneamente qué objetos hay en la imagen, dónde están y cómo interactúan. Ahora, ¿qué pasaría si las computadoras pudieran replicar esta capacidad humana? Pues algoritmos rápidos y precisos para la detección de objetos permitirían a las computadoras conducir automóviles sin necesidad de sensores, permiten que los dispositivos de asistencia transmitan información de escenas en tiempo real a los usuarios humanos y desbloquean el potencial de los sistemas robóticos receptivos de propósito general. Esto es la visión por computador.

La visión por computador busca cómo hacer que las computadoras adquieran un alto entendimiento de imágenes o videos mediante estos mismos, de modo que se automatizan tareas que el sistema visual humano normalmente ejecuta.

Los problemas fundamentales que circundan la visión por computador son los de *detección*, *localización* y *seguimiento* de objetos. A partir de la solución de estos podremos abordar otros como el conteo de objetos o la medición de la distancia entre ellos. Una tarea particularmente importante es el seguimiento de objetos, que involucra la localización y la detección, por su gran potencial práctico en situaciones de vigilancia, optimización de producción, transporte, logística, cuidado y monitoreo de niños y/o algo tan necesario en tiempos de pandemia como el monitoreo del correcto distanciamiento social.

II. Marco teórico

II-A. Planteamiento del problema

Dado un video se busca obtener un modelo que detecte y siga un conjunto de n objetos (en este caso personas) por cada frame del video, donde cada objeto es representado a través de un vector $y_i \in \mathbb{R}^6$:

$$y_i = \{x_i, y_i, w_i, h_i, p_r, i_d\} \tag{1}$$

donde x_i, y_i representan la posición, w_i ancho y h_i la altura del objeto; estas cuatro características dan información del recuadro que encierra al objeto detectado (ver figura 1). p_r representa el nivel de certeza del modelo de que lo que se ha detectado sí es un objeto e i_d es la variable que permite hacer seguimiento del objeto en cuestión.

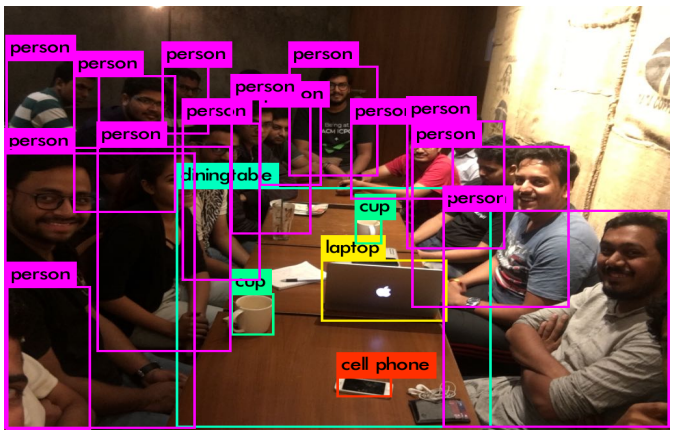

Figura 1. Representación visual de x_i, y_i, w_i, h_i

II-B. Fundamentos matemáticos para resolver el problema

Como ya se mencionó, este problema es uno de visión por computador. Actualmente la mayoría de este tipo de problemas se solucionan mediante Deep Learning, es un subcampo del aprendizaje autónomo que se ocupa de los algoritmos inspirados en la estructura y función del cerebro llamados redes neuronales artificiales (ANN); Deep Learning no es mas que redes neuronales enormes.

Lo que diferencia Deep Learning de las ANN tradicionales es que para el primero tiene un rendimiento proporcional a la cantidad de datos que se tengan, por lo que también se requiere de computadoras con hardware potente y de alta velocidad.

Comencemos por entender las redes neuronales tradicionales.

II-B1. Redes Neuronales: Para entender el funcionamiento de una red neuronal primero se debe entender su componente fundamental: la neurona o perceptrón. En pocas palabras, una neurona se entiende como la suma ponderada de varias entradas cuyo resultado es evaluado por una función.

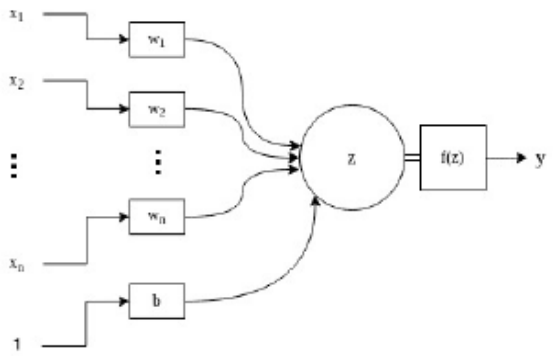

Figura 2. Diagrama de un perceptrón con cinco señales de entrada.

Donde

$$y = f(z) \qquad z = \sum_{i}^{n} w_i x_i + b$$

Cada entrada del perceptrón se puede interpretar como un atributo o característica que describe una instancia. La entrada puesta en 1 y el término b se pueden ver como un bias que le da más flexibilidad al modelo al momento de ajustar la salida. La función que evalúa dicha suma ponderada se le conoce como función de activación.

La función de activación se encarga de devolver una salida a partir de un valor de entrada, normalmente el conjunto de valores de salida en un rango determinado como (0,1) o (-1,1). Se buscan funciones que las derivadas sean simples, para minimizar con ello el coste computacional. Esta función se escoje dependiendo de la aplicación. Algunas de las más utilizadas son la **sigmoidea, tangente hiperbólica, ReLU** y **softmax**.

En la figura 2 se muestra lo que se conoce como el perceptrón simple, que es básicamente una red neuronal de una sola capa. Este solo sirve para clasificar problemas linealmente separables, cosa que ya se podía hacer mediante métodos estadísticos antes de que este apareciera, y de una forma mucho más eficiente, por lo que se volvió inútil rápidamente.

Pero, ¿qué tal si se utilizan múltiples secuencias de neuronas? Así aparece el perceptrón multicapa. El perceptrón multicapa es una red neuronal artificial formada por múltiples capas, de tal manera que tiene capacidad para resolver problemas que no son linealmente separables, lo cual ya mencionamos es la principal limitación del perceptrón (también llamado perceptrón simple).

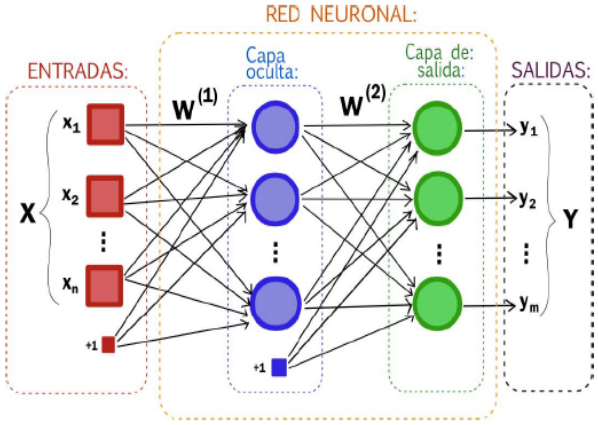

Figura 3. Esquema de una red neuronal

Como se puede apreciar en la figura 3, las neuronas que componen la red están interconectadas a través de varias capas. Estas capas son la de entrada, salida y las ocultas. La capa oculta es la que abstrae los datos que provee la capa de entrada mientras que la capa de salida codifica esa información para producir una salida. Pueden haber varias capas ocultas pero solo hay una capa de salida y una de entrada. En este caso las matrices W_1 y W_2 representan la ponderación o el peso que se le da a cada interconexión entre capa y capa. La ecuación que describe el modelo de una red neuronal de L capas es:

$$Y = f^L(W^L f^{L-1}(W^{L-1}...f^1(XW^1+b^1)...+b^{L-1}+b^L)$$

Donde

- Y es la salida de la red, cada columna corresponde a la salida de una neurona de la capa final y cada fila se interpreta como la salida correspondiente a una instancia de entrada.
- X es el set de datos de entrada en la red, las columnas de X son las características que representan una instancia, mientras que cada fila viene a ser la instancia en sí misma.
- W^i representa las conexiones que hay entre las neuronas de la capa i y las neuronas de la capa $i-1$.
- b^i representa la conexión de offset de cada capa
- f^i es la función de activación que tienen las neuronas pertenecientes a la capa i.

II-B2. Aprendiendo con el gradiente descendiente: algoritmo backpropagation: El algoritmo de **backpropagation** es probablemente la parte más fundamental en una red neuronal. Fue introducido por primera vez en la década de 1960 y casi 30 años después (1989) popularizado por Rumelhart, Hinton y Williams en un artículo titulado "Aprendizaje de representaciones mediante retropropagación de errores".

El algoritmo se utiliza para entrenar eficazmente una red neuronal a través de un método llamado regla de cadena. En términos simples, después de cada paso hacia adelante a través de una red, la propagación hacia atrás realiza un paso hacia atrás mientras ajusta los parámetros del modelo (pesos y bias).

El paso final en un paso hacia adelante es evaluar la salida predicha s contra una salida esperada y. La salida y es parte del conjunto de datos de entrenamiento (x, y) donde x es la entrada (como vimos en la sección anterior).

La evaluación entre s y y sucede a través de una función de costo. Esto puede ser tan simple como MSE (error cuadrático medio) o más complejo como la entropía cruzada. Llamamos a esta función de costo C y la denotamos de la siguiente manera:

$$C = cost(s, y)$$

donde el costo puede ser igual a MSE, entropía cruzada o cualquier otra función de costo.Basado en el valor de C, el modelo "sabe" cuánto ajustar sus parámetros para acercarse a la salida esperada y. Esto sucede usando el algoritmo de backpropagation.

Backpropagation tiene como objetivo minimizar la función de costo ajustando los pesos y sesgos de la red. El nivel de ajuste está determinado por los gradientes de la función de costo con respecto a esos parámetros.

Nuestro objetivo es reducir la pérdida cambiando los pesos de manera que la pérdida converja al valor más bajo posible. Intentamos reducir la pérdida de forma controlada, dando pequeños pasos hacia la pérdida mínima. Este proceso se llama Gradient Descent (GD). Mientras realizamos GD, necesitamos saber la dirección en la que deben moverse los pesos. En otras palabras, debemos decidir si aumentar o disminuir los pesos. Para conocer esta dirección, debemos tomar la derivada de nuestra función de pérdida. Esto nos da la dirección del cambio de nuestra función.

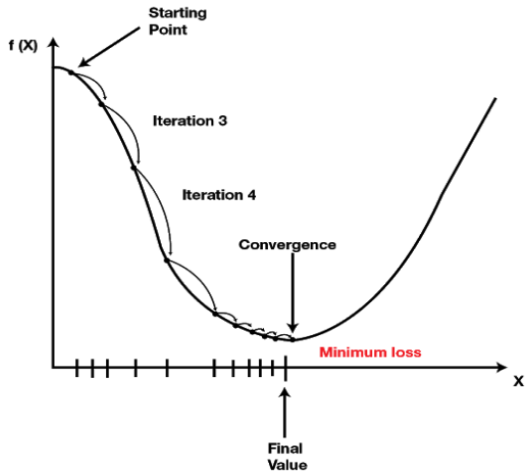

Figura 4. Ejemplo de función de costo.

A continuación se muestra una ecuación que muestra cómo actualizar los pesos usando Gradient Descent.

$$w = w - \alpha \frac{\partial C}{\partial w}$$

Aquí, el término α se conoce como la tasa de aprendizaje y se multiplica por la derivada de nuestra función de pérdida C.

Al realizar la propagación hacia atrás, debemos encontrar la derivada de nuestra función de pérdida con respecto a nuestros pesos. En otras palabras, nos preguntamos "¿Cómo cambia nuestra función de pérdida cuando cambiamos nuestros pesos en una unidad?". Luego multiplicamos esto por la tasa de aprendizaje, alfa. La tasa de aprendizaje controla el tamaño del paso del movimiento hacia los mínimos. Intuitivamente, si tenemos una gran tasa de aprendizaje vamos a dar

grandes pasos. Por el contrario, si tenemos una tasa de aprendizaje pequeña, vamos a dar pequeños pasos. Así, la tasa de aprendizaje multiplicada por la derivada puede considerarse como pasos que se están dando sobre el dominio de nuestra función de pérdida. Una vez que damos este paso, actualizamos nuestros pesos. Y este proceso se repite para cada función.

III. Bibliografía

1. Aurélien Gerón. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition), Part 2: Neural Networks and Deep Lear- ning. 2017

2. Joseph Redmon. You Only look Once: Unified, Real-Time Object detec- tion. 2016

3. A. Bewley, G. Zongyuan, F. Ramos, and B. Upcroft. Simple online and realtime tracking. inICIP, 2016.

4. N. Wojke, A. Bewley and D. Paulus. Simple online and realtime tracking with a deep association metric. 2017 IEEE International Conference on Image Processing (ICIP), Beijing, 2017.