UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA - PROCESAMIENTO DIGITAL DE SEÑALES

Tutorial:
Entrenamiento de la Red Neuronal Convolucional YOLO para objetos propios

EDGAR ROBERTO SILVA GUZMÁN

CIUDAD UNIVERSITARIA, CD. MX. MARZO 2020

Tutorial basado en la tesis de Maestría:
Detección de Objetos con Redes Neuronales Profundas para un Robot de Servicio [1]

Capítulo 1

Introducción al aprendizaje profundo

El *aprendizaje automatizado* o *aprendizaje máquina* (del inglés, *machine learning*) es una rama de la inteligencia artificial que tiene como objetivo, desarrollar algoritmos computacionales que doten la capacidad de "aprender" a las computadoras. Se dice que un sistema puede aprender si mejora su desempeño con base en experiencia, es decir, con base en ejemplos que se le presentan al sistema.

En las últimas décadas, el *aprendizaje profundo* (del inglés, *deep learning*) ha tenido gran impacto en el desarrollo de diversos sistemas, como la clasificación de imágenes, segmentación de imágenes, reconocimiento de voz, la traducción automática, entre otros.

El *aprendizaje profundo* es un tipo de *aprendizaje automatizado*, en el cual se utilizan redes neuronales artificiales con una jerarquía establecida, donde en cada nivel se extraen las características significativas del nivel anterior. A esta jerarquía se le conoce como *modelo neuronal*. Como menciona Goodfellow [2], es difícil para una computadora darle significado a un conjunto de datos sin procesar, por ejemplo, darle una descripción a un conjunto de pixeles de una imagen de entrada. El aprendizaje profundo resuelve este tipo de problemas, utilizando cada una de las capas del modelo neuronal para generar patrones representativos de los datos de entrada.

En la figura 1.1 se muestra un modelo neuronal que da la representación de una persona combinando simples conceptos, como esquinas, bordes y contornos. En la primera capa, la *capa visible* (nombrada así por que las variables que contiene son observables) se presentan los datos de entrada, en este caso es un conjunto de pixeles de una imagen. Después, se presenta una serie de *capas ocultas* que extraen características abstractas de la imagen. Reciben el nombre de *capas ocultas*, porque sus variables no son observables

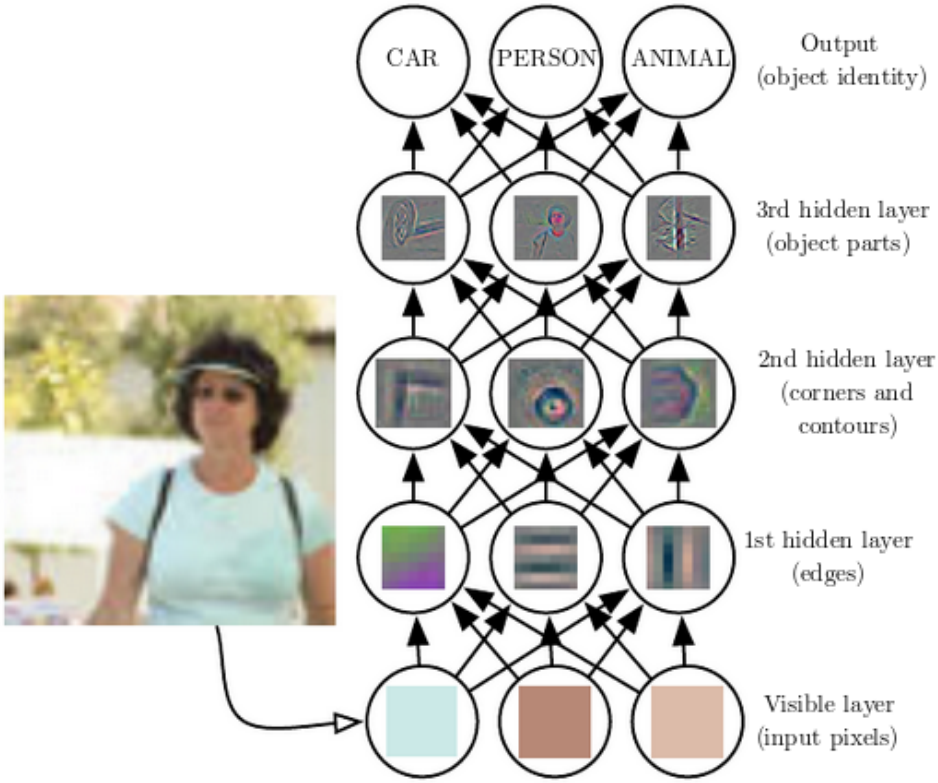


Figura 1.1: Modelo de aprendizaje profundo[2].

por el desarrollador. Goodfellow [2] menciona que si se visualiza el contenido de las capas ocultas podemos encontrar distintos patrones. En la primera capa oculta se encuentran los bordes de la imagen, en la segunda capa oculta se encuentran esquinas y contornos, y en la tercera capa oculta se pueden encontrar objetos específicos, formados por conjuntos de bordes y esquinas representativos. Por último, en la *capa de salida* se obtiene una descripción del objeto en términos de los patrones que contiene.

1.1. Redes neuronales artificiales

Las redes neuronales artificiales son un modelo computacional que trata de emular los procesos biológicos del sistema nervioso, como la forma en la que el cerebro humano almacena información, aprende y reconoce patrones. Están formadas por un un conjunto de nodos conectados entre sí, cuyo principal objetivo es almacenar información, procesarla, y dependiendo de su estado

de activación, transmitir o no señales a otros nodos.

La estructura general de una red neuronal artificial viene dada por conjuntos de capas de nodos, donde cada nodo representa una neurona artificial (figura 1.2). Cada capa tiene las siguientes características:

- En la capa de entrada, se adquieren las señales de entrada, es decir, la información proveniente de sensores o de algún sistema de procesamiento de bajo nivel.
- En las capas ocultas, se extraen características abstractas de la señal de entrada y se obtienen los descriptores o los patrones más significativos de ésta.
- En la capa de salida, se obtiene una clasificación, detección o interpretación de la señal de entrada.

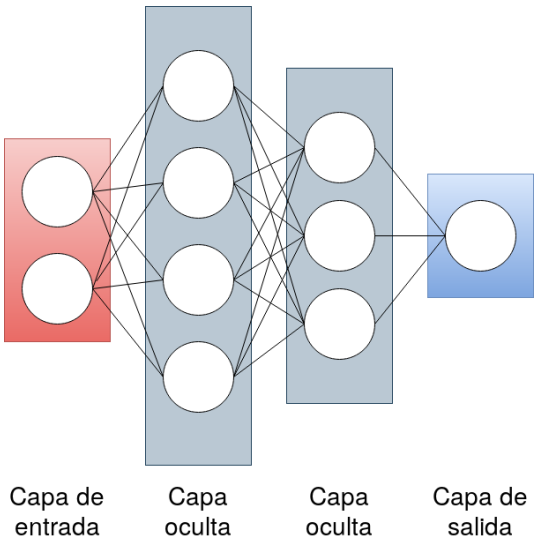


Figura 1.2: Estructura general de una red neuronal artificial.

En el año de 1957, Frank Rosenblatt, basado en la *teoría Hebbiana* de la plasticidad sináptica (el comportamiento de las neuronas biológicas durante el proceso de aprendizaje), crea el *perceptrón* como el modelo más simple de una neurona biológica. Como menciona Rojas en [3], el principal aporte de Rosenblatt fue la introducción de variables numéricas llamadas *pesos*, y una *función de activación* para cada neurona, la cual indica si la neurona transmitirá o no su información a la siguiente capa neuronal.

El modelo del perceptrón de Rosenblatt (figura 1.3) está compuesto por un vector de entradas $\vec{x} = (x_1, x_2, x_3, ..., x_n)$ asociado a un vector de pesos

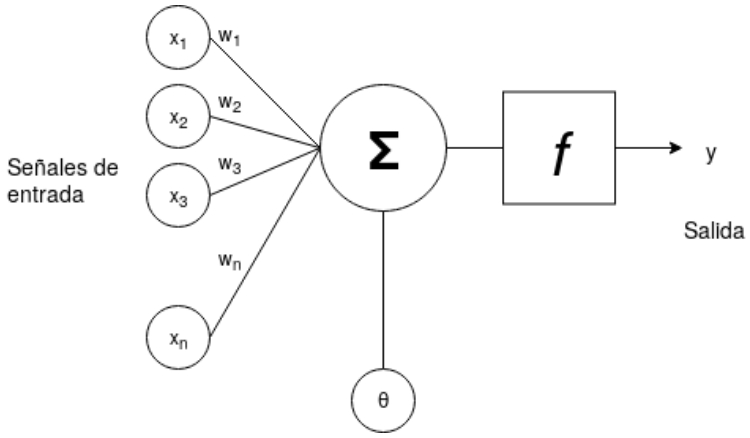


Figura 1.3: Modelo de un perceptrón.

$\vec{w} = (w_1, w_2, w_3, ..., w_n)$, y produce una salida binaria, determinada por la suma ponderada $\sum w_j x_j$, un sesgo Θ (en inglés *bias*) y una función de activación, que en este caso es la función escalón de Heaviside, definida por la siguiente ecuación:

$$f(x) = \begin{cases} 1 & \text{si} \quad \sum_i^n w_i x_i + \Theta > 0 \\ 0 & \text{en otro caso} \end{cases}$$

En algunas aplicaciones del perceptrón, como la detección de bordes, los valores del vector de pesos $\vec{w}$ ya están previamente establecidos. Para encontrar los valores $\vec{w}$ de manera automática en otro tipo de aplicaciones, se hace uso de los *algoritmos de aprendizaje*.

1.2. Algoritmos de Aprendizaje Automatizado

Un algoritmo de aprendizaje automatizado es un algoritmo que puede aprender con base en los datos que se le presentan[2]. En otras palabras, es un algoritmo adaptativo mediante el cual, una red neuronal artificial modifica sus parámetros para lograr un comportamiento deseado. Esto se logra presentando una serie de ejemplos mapeados de entrada a salida, y ejecutando iterativamente un procedimiento de corrección de errores, hasta que la red neuronal dé la respuesta deseada[3].

La mayoría de los algoritmos de aprendizaje automatizado se pueden dividir en la categoría de *aprendizaje supervisado* y *aprendizaje no supervisado*.

Los algoritmos de **aprendizaje supervisado** utilizan un conjunto de datos de entrenamiento, donde cada ejemplo está asociado con una etiqueta u objetivo, el cual se espera como resultado del ejemplo dado. Los parámetros de la red se corrigen de acuerdo con la magnitud del error, definido por el algoritmo de aprendizaje. El aprendizaje supervisado se utiliza en sistemas de clasificación y sistemas de regresión (como la predicción de datos).

Los algoritmos de **aprendizaje no supervisado** utilizan un conjunto de datos de entrenamiento que, a diferencia del aprendizaje supervisado, no tienen una etiqueta u objetivo establecido. La finalidad de este tipo de aprendizaje, es encontrar un modelo de distribución de probabilidad de los ejemplos u observaciones dados[2]. Generalmente se utiliza en tareas de agrupamiento de datos, donde se divide la base de datos en conjuntos de datos con características en común.

1.3. Redes neuronales convolucionales

Las Redes neuronales convolucionales (en inglés, *Convolutional Neural Network*), son una versión *regularizada* del perceptrón multicapa, usadas principalmente en sistemas de detección y clasificación de objetos, segmentación de imágenes, procesamiento de lenguaje natural, entre otras disciplinas, debido a su buen desempeño y a que necesitan poco pre-procesamiento de datos. Las redes neuronales convolucionales son redes neuronales que utilizan la operación de *convolución* en lugar de una matriz de multiplicación en al menos en alguna de sus capas[2].

Una convolución es una operación matemática que transforma dos funciones en una tercera, la cual representa la magnitud de cuánto se superpone una función sobre la otra. Siendo f la función de entrada y g el núcleo, la convolución de f y g (el mapa de características), denotada como $f * g$, se define como la integral del producto de ambas funciones, después de desplazar una de ellas una distancia t:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\eta) g(t - \eta) d\eta$$

En el caso discreto, se utiliza una forma discreta de la convolución:

$$s(t) = (f * g)(t) = \sum_{n} f(n) g(t - n)$$

En las aplicaciones de aprendizaje profundo, la función de entrada es un arreglo multidimensional de datos y el núcleo es un arreglo multidimensional de parámetros que se adaptan mediante un *algoritmo de aprendizaje*[2]. Por ejemplo, si se tiene una imagen bidimensional I como entrada, se tendrá un núcleo bidimensional K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

De igual forma, en muchas bibliotecas de redes neuronales se implementa una función similar llamada **correlación cruzada**, que actúa igual que la convolución, pero sin voltear la matriz del núcleo:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Cabe destacar que en algunos libros de aprendizaje profundo se implementa la función de correlación cruzada, pero la llaman convolución.

Las redes neuronales convolucionales están compuestas principalmente de capas convolucionales, capas de reducción (en inglés, *pooling layer*), y capas totalmente-conectadas (en inglés, *fully-conected*).

1.3.1. Capas convolucionales

La característica principal de una capa convolucional, es que puede extraer las características espaciales y temporales de una imagen mediante la aplicación de filtros (la matriz del núcleo) en cada una de sus capas. El objetivo es reducir el tamaño de las imágenes sin perder sus características más significativas. Entonces, el resultado de la convolución es una matriz de menor tamaño definida por el parámetro *paso* (en inglés, *stride*), que indica cómo el filtro convoluciona a través de la imagen. Por último, la salida de cada neurona es definida por una *función de activación*. Generalmente en las capas convolucionales, se utiliza la función de activación **ReLU** (siglas del inglés, *Rectified Linear Unit*).

En la figura 1.4a, se muestra como ejemplo una imagen de una sola capa, de dimensiones 5x5x1 y un núcleo de 3x3x1. El resultado de la operación convolución con *paso=1* en la primera iteración se muestra en la figura 1.4b. El resultado de la siguiente iteración se muestra en la figura 1.4c. Y el resultado de la última iteración se muestra en la figura 1.4d. En la figura 1.4 se aprecia cómo el núcleo se recorre a través de la imagen para hacer la operación de convolución.

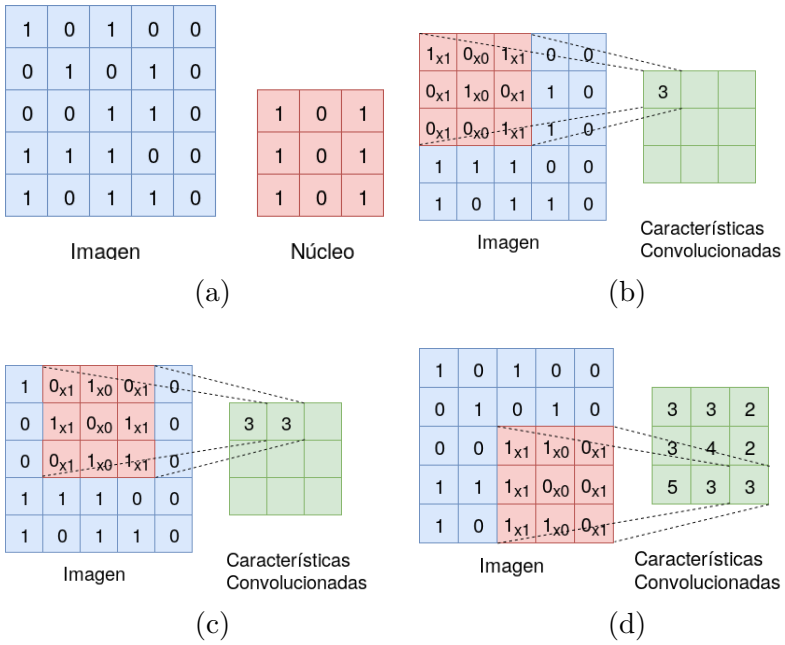


Figura 1.4: Operación Convolución.

Funciones de activación

La función de activación, es una transformación no lineal que define la salida de una neurona artificial en términos de la entrada dada. La salida será enviada a la entrada de la siguiente capa neuronal.

Existen diferentes tipos de funciones de activación, siendo la más usada la función ReLU. En la figura 1.5, se muestran las funciones de activación más comunes.

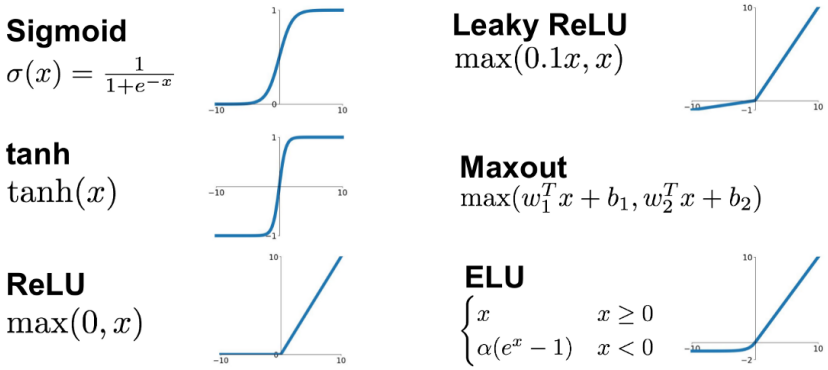


Figura 1.5: Funciones de activación[4].

1.3.2. Capas de reducción

La capa de reducción (en inglés, *pooling layer*) tiene como objetivo, reducir el tamaño espacial de las características convolucionadas para, de esta manera, reducir también el tiempo de cómputo requerido para el procesamiento de los datos. Además, se extraen características dominantes que son invariantes a rotaciones y traslaciones [5].

Hay dos tipos de capas de reducción, la capa de reducción-máxima (en inglés, *max-pooling*) y la capa de reducción-promedio (en inglés, *average-pooling*). La capa de reducción-máxima regresa el valor máximo de la parte de la imagen cubierta por el núcleo. La capa de reducción-promedio regresa el promedio de todos los valores de la parte de la imagen cubierta por el núcleo. En la figura 1.6 se muestra el resultado de la operación en ambas capas.

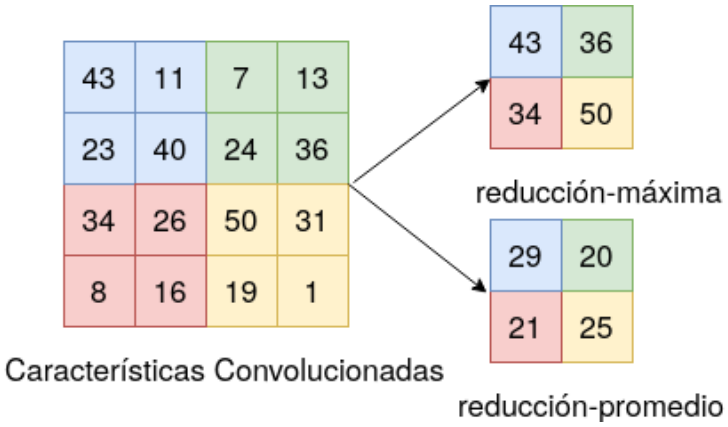


Figura 1.6: Capas de Reducción con *paso*=2.

1.3.3. Capas totalmente-conectadas

Generalmente en las arquitecturas de redes neuronales artificiales, se utilizan una o más capas totalmente-conectadas para la etapa de clasificación. A la salida de las múltiples capas convolucionales y capas de reducción, se tienen las características de alto nivel, que pueden ser fácilmente clasificadas por un perceptrón multicapa. La clasificación se logra transformando las matrices multidimensionales en un vector columna (que es la entrada del perceptrón multicapa) para, posteriormente utilizar el método de *retropropagación* (en inglés, *backpropagation*) y así ajustar los parámetros de la red neuronal artificial (hacer el entrenamiento de la red). Por último se tienen diferentes técnicas de clasificación, como la función *softmax*.

La función Softmax es la función más utilizada a la salida de las capas de clasificación. Ésta representa la distribución de probabilidad sobre n diferentes clases[2], en otras palabras, la función regresa un vector de n probabilidades, dónde la suma de todos sus elementos es 1.

La función Softmax viene dada por la siguiente ecuación:

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

1.3.4. Capas residuales (ResNet)

En la mayor parte de las arquitecturas neuronales, cada capa alimenta a la siguiente capa, es decir, cada una de las capas se conecta únicamente con la siguiente capa neuronal. A diferencia de esto, en una red residual (en inglés,

Residual Network o *ResNet*), en cada bloque residual, cada capa alimenta a la siguiente capa y a una capa que está de dos a tres capas de distancia. En la figura 1.7 se muestra el diagrama de un bloque residual.

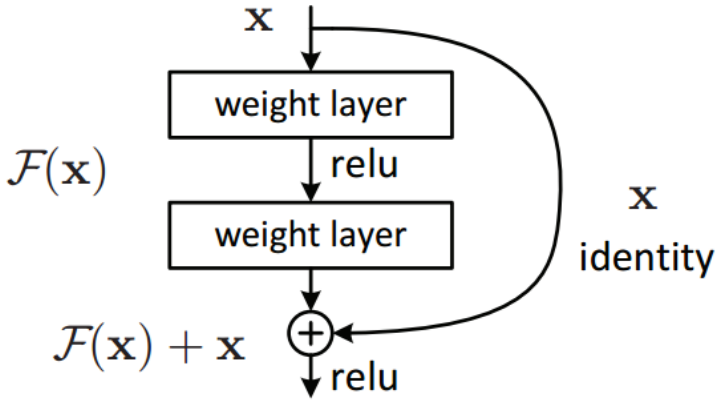


Figura 1.7: Capa Residual[6].

La razón principal de utilizar bloques residuales, es evitar el problema del desvanecimiento del gradiente (en inglés, *vanishing gradient*), lo que significa que no puede encontrar un gradiente óptimo al agregar demasiadas capas a la arquitectura neuronal. Con los bloques residuales, se pueden propagar los gradientes más grandes a las capas iniciales, para que de esta manera, las capas iniciales también puedan aprender de manera rápida al igual que las capas finales [6].

1.4. Retro-propagación

Cuando se utiliza una red pre-alimentada (en inglés, *feedforward network*) se recibe una entrada x y se produce una salida y. Para esto, la información fluye hacia adelante, propagándose por todas las capas ocultas a través de la red para, finalmente, producir una salida. Este procedimiento es llamado *propagación hacia adelante* (del inglés, *fordward propagation*). Una vez calculada la salida de la red, es posible calcular el error de la propagación hacia adelante respecto a los *datos reales* (en inglés, *ground truth*), es decir, la información de la base de datos del entrenamiento.

El algoritmo de retro-propagación (en inglés, *back-propagation*), es un algoritmo de aprendizaje supervisado que permite que la información del error se propague hacia atrás en la red. Su objetivo es minimizar la *función*

de costo ajustando los parámetros de pesos y sesgos de la red, utilizando el *algoritmo del descenso del gradiente*.

Una función de costo, (en inglés, *cost function*) es una medida del desempeño de un modelo de aprendizaje máquina, que cuantifica el error entre las predicciones hechas y los datos reales del modelo. El valor obtenido por esta función es llamado *costo*, *error* o *pérdida*. La finalidad de este método, es encontrar los parámetros que minimicen la función de costo [7].

Existen diversas funciones de costo, cada una se utiliza dependiendo del problema de aprendizaje:
Error absoluto medio:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\widehat{y_i} - x_i|$$

Error Cuadrático Medio:

$$MSE = \frac{1}{2}\sum_{i=1}^{n}(\widehat{y_i} - x_i)^2$$

Entropía Cruzada Binaria:

$$CE = -\frac{1}{n}\sum_{i=1}^{n}(x * log(\widehat{y_i}) + (1-x) * log(1-\widehat{y_i}))$$

Entropía Cruzada Categórica:

$$CE = -\frac{1}{n}\sum_{i=1}^{n} x_i * log(\widehat{y_i})$$

Donde:

- i es el índice de la muestra.
- $\widehat{y}$ es valor predicho.
- x es el valor esperado.
- n es el número de clases en el conjunto de datos.

1.4.1. Algoritmo del descenso del gradiente

El descenso del gradiente es un algoritmo de optimización utilizado para minimizar el valor de una función de costo. Se calcula iterativamente en dirección del menor valor en cada paso, definido por el negativo del gradiente

$(-\nabla J)$. Para definir la relación entre cualquier peso w_{ij} y la función de costo J, se hace uso de las derivadas parciales y la *regla de la cadena*:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

Dónde

- J es la función de costo.
- w_{ij} es el peso i de la capa j.
- a_j es la salida de la neurona después de la función de activación.
- z_j es la combinación lineal resultado de $\sum \vec{x}\vec{w}$.

De esta manera, el nuevo valor para el peso w_{ij} viene dado por:

$$w_{ij} = w_{ij} - \alpha\frac{\partial J}{\partial w_{ij}}$$

Donde α es la tasa de aprendizaje (en inglés, *learning rate*).

1.4.2. Transferencia de conocimiento

La transferencia de conocimiento (en inglés, *Transfer Learning*) es una técnica del aprendizaje máquina que utiliza, como punto de partida, un modelo preentrenado de una red neuronal artificial, en los casos donde se tiene como objetivo, entrenar una conjunto de datos de imágenes pequeño con muchas características similares al modelo anterior.

En esta metodología se utiliza un modelo neuronal entrenado para un conjunto de datos robusto, como ImageNet[8] o COCO[9], y se hace el reentranamiento o *ajuste fino* (en inglés, *fine-tuning*) de las últimas capas, las capas de clasificación. De esta manera, se tiene un nuevo modelo que aprovecha las características extraídas del modelo anterior, para clasificar únicamente las nuevas clases de interés para el sistema.

1.4.3. Sobre-Ajuste y Sub-Ajuste

Uno de los desafíos más importantes de un modelo neuronal, es que debe funcionar con datos de entrada nuevos, es decir, datos que no sean parte del conjunto de datos de entrenamiento de la red. A esta capacidad del modelo se le llama *generalización*.

Los términos sobre-ajuste (del inglés, *overfitting*) y sub-ajuste (del inglés, *underfitting*) hacen referencia a las deficiencias de un modelo para poder desempeñarse de manera correcta en presencia de datos nuevos.

Cuando en el conjunto de datos de entrenamiento se introducen datos irrelevantes, con ruido, distorsiones y muestras que no son suficientemente representativas para el entrenamiento, el modelo los considerará como ejemplos válidos, por lo cual, la red sufrirá de un sobre-ajuste. Esto quiere decir que el modelo predecirá correctamente solo los datos idénticos al conjunto de datos de entrenamiento, y los datos que salen de los rangos establecidos serán descartados.

Por el contrario, el sub-ajuste ocurre cuando el modelo no tiene suficientes datos de entrenamiento, lo cual produce una baja generalización y el modelo será incapaz de hacer predicciones correctas.

En la figura 1.8 se muestra la curva de desempeño de un modelo entrenado con sobre-ajuste, sub-ajuste y un modelo correctamente entrenado.

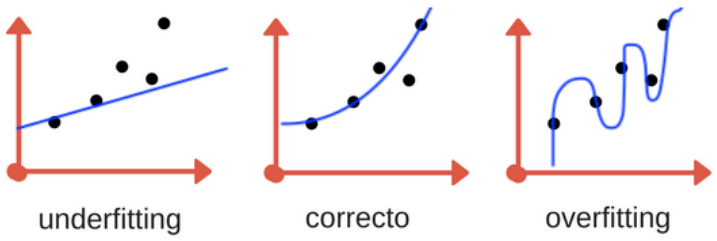


Figura 1.8: Sub-Ajuste y Sobre-Ajuste[10].

Para evitar el sobre-ajuste y el sub-ajuste, se deben tener en cuenta los siguientes puntos:

- Hacer uso de técnicas de aumento de datos (en inglés, *data augmentation*)
- Tener una cantidad suficiente de muestras por cada clase de los datos de entrenamiento.
- Las clases deben ser variadas y equilibradas en cantidad.
- Dividir los datos en un conjunto de entrenamiento y un conjunto de validación.
- Evitar la cantidad excesiva de capas neuronales en la arquitectura neuronal.

1.4.4. Algunas terminologías del aprendizaje profundo

Época. Se llama época (del inglés, *epoch*) al procedimiento donde todo el conjunto de entrenamiento se propaga hacia adelante y se retro-propaga por toda la arquitectura de la red neuronal una sola vez.

Tamaño de lote. El tamaño de lote (del inglés, *batch size*) es el número de ejemplos que se le presentan a la red para estimar el error del gradiente.

Iteraciones. Las iteraciones (del inglés, *iterations*), es el número de lotes (del inglés, *batches*) que se procesan durante el entrenamiento de una red neuronal.

Capítulo 2

YOLO

YOLO (por sus siglas en inglés, *You Only Look Once*) es un sistema del *estado del arte*, que utiliza una red neuronal convolucional para la detección de objetos en tiempo real[11]. El sistema aplica una única red neuronal a la imagen completa, razón por la cual es muy rápido. Esta red divide la imagen en regiones y predice múltiples cajas delimitadoras (en inglés, *bounding box*), y la probabilidad de detección de las clases de entrenamiento para cada caja delimitadora. Por último, utiliza un método de supresión de no-máximos para eliminar múltiples detecciones del mismo objeto. Como resultado, el sistema imprime los objetos que detectó, su confidencia y el tiempo de ejecución. En la figura 2.1 se muestra una ejecución del sistema sobre una imagen de entrada.

En el presente proyecto de tesis, se utilizó la versión 3 de YOLO (YOLOv3). Para explicar el funcionamiento del YOLOv3, se abordarán brevemente las partes que componen el sistema.

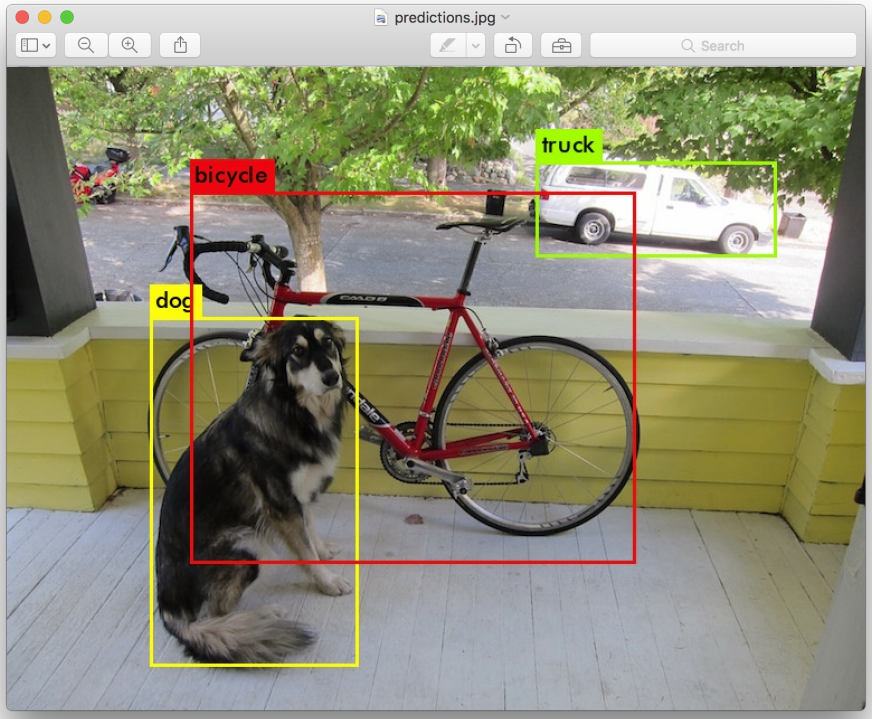


Figura 2.1: Detección de objetos con YOLO[11].

2.0.1. Arquitectura neuronal

La arquitectura de YOLOv3 está compuesta por 53 capas convolucionales, razón por la cual recibe el nombre de *Darknet-53* (figura 2.2). Cada capa convolucional es seguida de una normalización de lote (en inglés, *batch normalization*) y la función de activación **Leaky ReLU**. No se utiliza ninguna capa de reducción, en su lugar, se utilizan capas convolucionales con paso=2. Con esto, se reduce la dimensionalidad del mapa de características, evitando perder las características de bajo nivel que se le atribuyen a las capas de reducción-máxima.

Por otro lado, se plantea otra arquitectura sin capas residuales y con menos capas convolucionales (13) llamada *YOLOv3_tiny*. Como resultado se tiene una red neuronal más rápida, pero con menos confidencia en la detección y clasificación. En la figura 2.3 se muestra la arquitectura de *YOLOv3_tiny*

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 2.2: Arquitectura de Darknet-53[11].

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Figura 2.3: Arquitectura de YOLOv3-tiny.

2.0.2. Procesamiento

La imagen de entrada del sistema YOLOv3 se puede configurar de diferentes dimensiones en cada proceso de entrenamiento, como único requisito es que sean imágenes cuadradas de un tamaño múltiplo de 32, por ejemplo 608x608 ó 416x416. De lo contrario, *Darknet* redimensiona las imágenes al tamaño establecido en el archivo de configuración.

La entrada viene dada por:

$$Entrada(m, h, w, d)$$

Donde:

- m es el tamaño del lote de entrenamiento (en inglés, *batch*).
- h es la altura de la imagen (en inglés, *height*).
- w es el ancho de la imagen (en inglés, *width*).
- d son los canales de la imagen de entrada (en inglés, *depth*).

Dependiendo de la tarjeta gráfica, el tamaño del lote de entrenamiento puede ser subdividido. De esta manera, si se tiene un lote de tamaño 64 (64 imágenes) y subdivisiones de 16, entonces se procesarán 4 imágenes en paralelo en cada iteración de la red neuronal convolucional. Siguiendo este ejemplo y utilizando una imagen de 3 canales (RGB) de 416x416, a la entrada de YOLOv3 tendremos:

$$Entrada(64, 416, 416, 3)$$

YOLOv3 hace el proceso de detección en tres diferentes escalas, redimensionando la imagen de entrada por un factor de 32, 16 y 8 respectivamente. Por ejemplo, si se tiene como entrada una imagen de 416x416 y reduce por un factor de 32, el tamaño del mapa de características será de de 13x13 (figura 2.4a). A cada elemento del mapa se le conoce como *celda*[12]. En el mapa de características, cada celda predice un número fijo de cajas delimitadoras.

En cada celda de un mapa de características se tienen los siguientes atributos: Bx(5+C). B representa el número de cuadros delimitadores que cada celda puede predecir. Según [11], cada uno de estos cuadros delimitadores B puede especializarse en la detección de cierto tipo de objeto. Cada uno de los cuadros delimitadores tiene atributos 5+C, que describen las coordenadas del centro, las dimensiones, la puntuación de objetividad y las confidencias de clase C para cada cuadro delimitador. En este proyecto de tesis, YOLOv3 predice 3 cuadros delimitadores para cada celda.

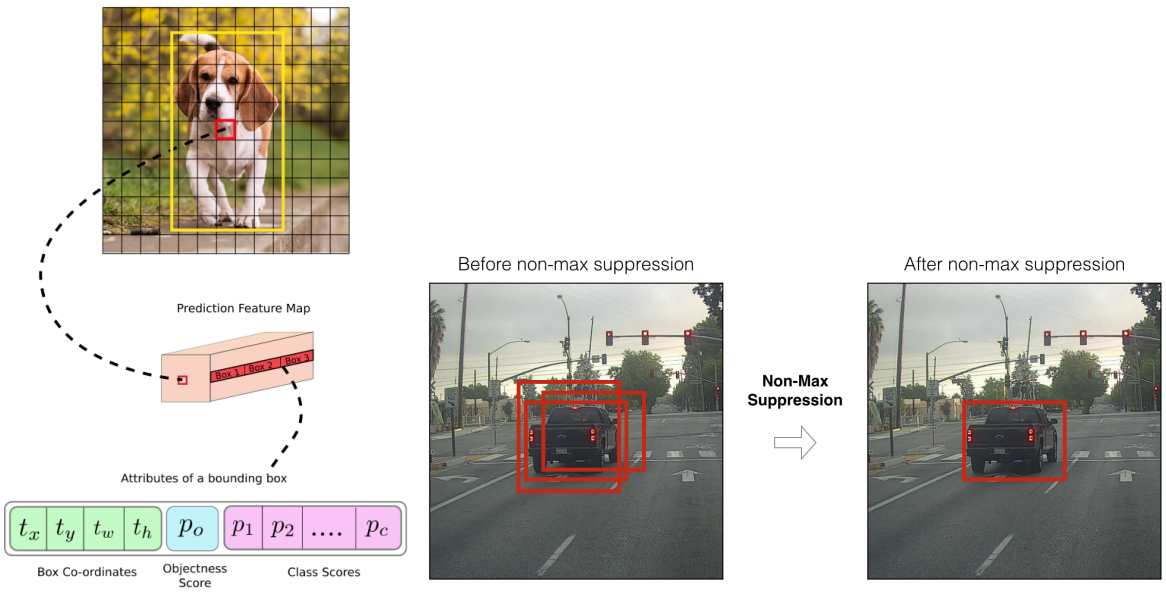
Por ejemplo, si en la arquitectura principal de YOLOv3 (DarkNet-53) se tiene una imagen de entrada de 416x416, se tendrán como resultado 3 mapas de características con 10,647 cajas delimitadoras predichas:

$$((52 * 52) + (26 * 26) + (13 * 13)) * 3 = 10,647 \tag{2.1}$$

Para reducir este número significativo de cajas delimitadoras se utilizan dos filtros.

- Se utiliza un umbral mínimo en la probabilidad de detección de un objeto en cada caja delimitadora, si no se supera dicho umbral, la caja delimitadora será descartada.

- Se utiliza un método de supresión de no-máximos con el objetivo de evitar múltiples detecciones del mismo objeto. Para esto se utiliza la métrica IoU (siglas del inglés *Intersection Over Union*) de la siguiente manera:
  Se elige la caja delimitadora con mayor probabilidad de detección y se compara con las demás cajas delimitadoras. Si el resultado de IoU es superior a un umbral, la caja delimitadora con menor probabilidad de detección es descartada (figura 2.4b).



(a) Mapa de características (b) Método de supresión de no-máximos

Figura 2.4: Procesamiento de YOLOv3 [12].

Capítulo 3

Ejecución del sistema de detección de YOLO entrenado con la base de datos COCO

3.1. Página Oficial

Página Oficial:
`https://pjreddie.com/darknet/yolo/`

En la página oficial se puede encontrar información detallada del framework DarkNet (framework utilizado para la implementación de YOLO).

3.2. Compilación

3.2.1. Descargar el repositorio

```
https://github.com/pjreddie/darknet
```

3.2.2. Clonar el repositorio (GitHub):

```
>git clone https://github.com/pjreddie/darknet
```

3.2.3. Compilación

Complilación (CPU)

```
>cd darknet/
>make
```

Complilación (GPU) (OpenCV) (CUDA & CUDNN) [1]

```
>cd darknet/
```

Editar el archivo Makefile (figura 3.1):
GPU=1
CUDNN=1
OPENCV=1

Elegir la arquitectura de la tarjeta gráfica para compilar con CUDA (CUDA arch y CUDA gencode)
Compilar:

```
>make
```

[1] Es necesario tener: drivers de Nvidia (depende la tarjeta gráfica), OpenCV, CUDA y CUDNN.

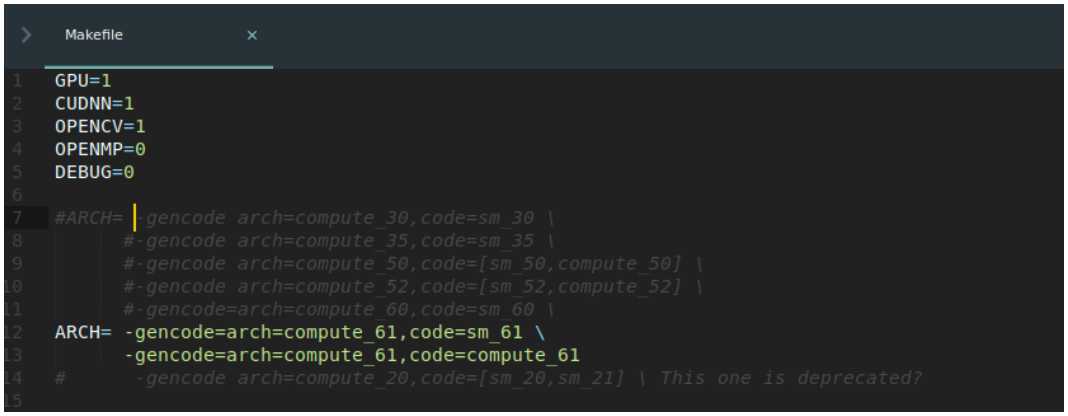


Figura 3.1: Modificación del archivo MakeFile

3.3. Configuración

3.3.1. Descargar los pesos

Descargar los pesos del modelo neuronal yolov3 en la carpeta cfg/:

```
>cd darknet/cfg/
>wget https://pjreddie.com/media/files/yolov3.weights
```

Editar el archivo cfg/yolov3.cfg (figura 3.2):

Linea 2:#Testing
Linea 3: batch=1
Linea 4: subdivisions=1
Linea 5:#Training
Linea 6:#batch=64
Linea 7:#subdivisions=16

```
yolov3.cfg
1  [net]
2  # Testing
3  batch=1
4  subdivisions=1
5  # Training
6  # batch=64
7  # subdivisions=16
```

Figura 3.2: Modificación del archivo yolov3.cfg

3.4. Detección de objetos en una imagen

Ejecutar el sistema de detección

```
>./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

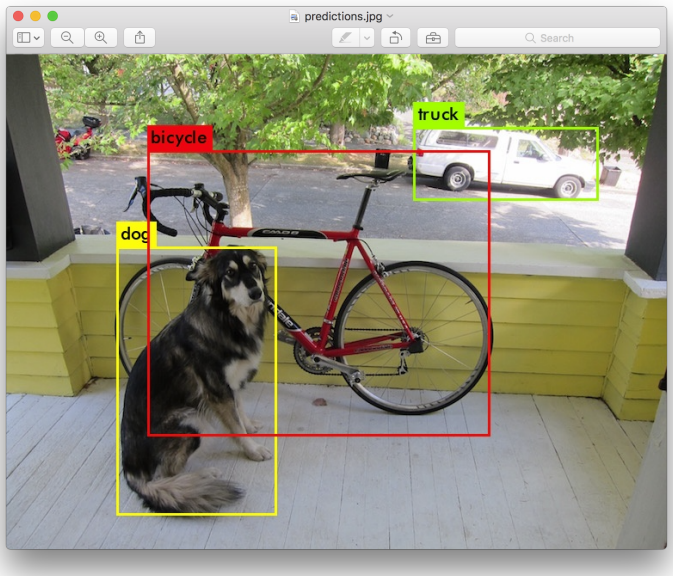


Figura 3.3: Imagen resultado del proceso de detección

3.5. Detección en un archivo de video

Ejecutar el sistema de detección

```
>./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights
```

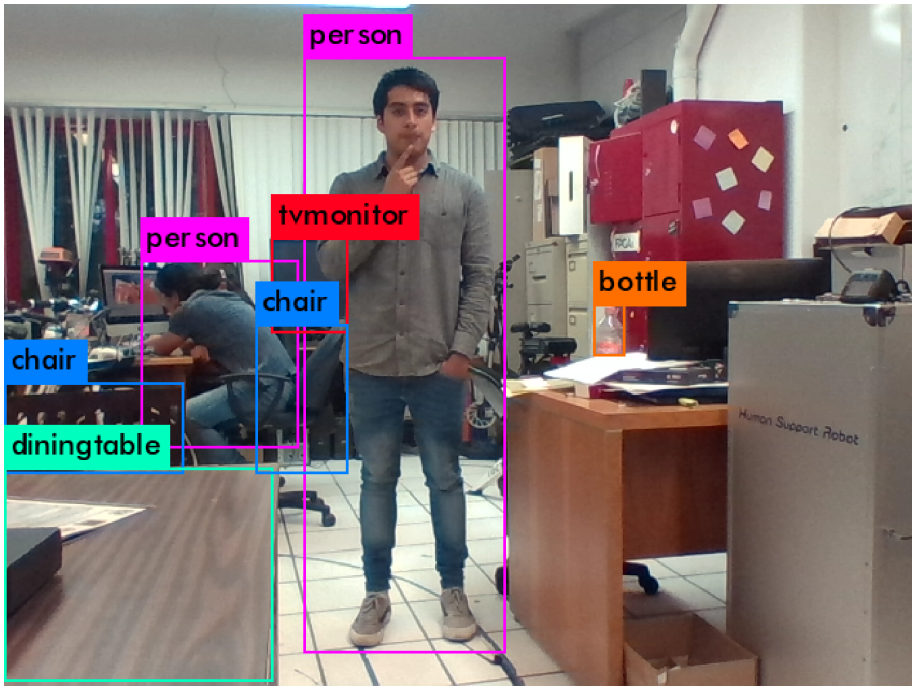


Figura 3.4: Imagen resultado del proceso de detección en una webcam

Capítulo 4

Entrenamiento de YOLO con objetos propios (yolo_tiny)

4.1. Etiquetado del dataset

Cada imagen debe tener un archivo de texto (.txt) asociado con la información de cada objeto de interés.

La sintaxis es la siguiente:

```
<object-class><x><y><width><height>
```

4.2. YoloMark

YoloMark es un software libre utilizado para etiquetar un dataset de imágenes. Es de fácil uso y genera los archivos de datos del etiquetado (archivos .txt) con la sintaxis compatible con el entrenamiento de la red YOLO.

Repositorio:

`https://github.com/AlexeyAB/Yolo_mark`

Clonar el repositorio (GitHub):

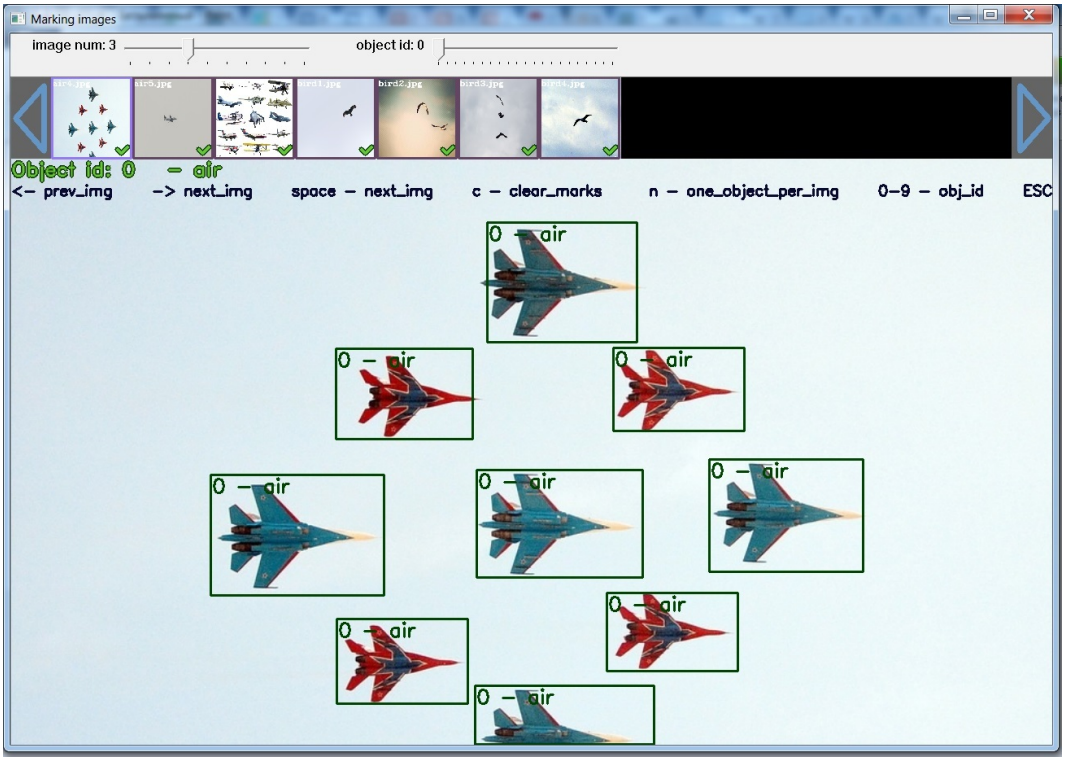
```
>git clone https://github.com/AlexeyAB/Yolo_mark.git
```

Complilación

```
>cd Yolo_mark/
>cmake .
>make
```

Ejecución

```
>./linux_mark.sh
```



4.3. Archivos necesarios

Archivos necesarios para realizar el entrenamiento de yolov3_tiny:
`yolo_v3.names`
`yolo_v3.data`
`yolo_v3.cfg`
`train.txt`
`valid.txt`
`yolov3_tiny.conv.15` #Pesos convolucionales de yolov3_tiny preentrenados

Enlace para descargar los archivos necesarios:
`https://mega.nz/#!LktE2aqI!rRp_WEHixG-bsUe-yJ2xgS3Dcx_bZSHpJPNXpAZMGDU`

Carpetas necesarias para realizar el entrenamiento de yolov3_tiny:
`dataset/` #Dataset de imágenes etiquetadas
`backup/`
Enlace para descargar el dataset:
`https://mega.nz/#!qw1mTY5R!Y_9f2BssbNBLuqBviTSQt8D5W5Vnulrz3M5Cg9rFMnY`

4.4. Configuración del archivo yolov3-tiny.names

En este archivo se coloca el nombre de cada una de las clases de los objetos a entrenar.
yolov3_tiny.names:
`apple_juice`
`blue_bowl`
`blue_lego`
`blue_mug`
`blue_spoon`
`chocolate_cookies`
`green_mug`
`orange_juice`
`pineapple_cookies`
`purple_mug`
`purple_plate`
`red_mug`
`strawberry_cookies`
`yellow_cup`
`yellow_lego`

4.5. Configuración del archivo yolov3_tiny.data

Es necesario crear el archvio train.txt y el archivo valid.txt
El programa de python *train_valid.py* genera los archivos train.txt y valid.txt con una distribución de 90-10 respectivamente del total de imágenes del dataset. La sintaxis de ejecución es la siguiente:

```
python train_valid.py /home/edd/yolov3-tiny/dataset/
```

El archivo train.txt contiene la ruta de todas las imágenes del dataset de entrenamiento.
El archivo valid.txt contiene la ruta de todas las imágenes del dataset de validación.

El archivo yolov3-tiny.data configura la ruta de los archivos necesarios para el entrenamiento
yolov3_tiny.data:
`classes= 15` #Numero de clases
`train = /home/edd/yolov3_tiny/train.txt` #Ruta del archivo train.txt
`valid = /home/edd/yolov3_tiny/valid.txt` #Ruta del archivo valid.txt
`names = /home/edd/yolov3_tiny/yolov3_tiny.names` #Ruta del archivo .names
`backup = /home/edd/yolov3_tiny/backup/` #Ruta de la carpeta para guardar los pesos del modelo

4.6. Configuración del archivo yolov3_tiny.cfg

En este archivo se configura la arquitectura neuronal de yolov3_tiny
yolov3_tiny.cfg:
Linea 3: `batch=24` #Configurar dependiendo de la tarjeta gráfica
Linea 4: `subdivisions=8` #Configurar dependiendo de la tarjeta gráfica
Linea 127: `filters=60` #(clases + 5)*3
Linea 135: `classes=15` #Numero de clases de entrenamiento
Linea 171: `filters=60` #(clases + 5)*3
Linea 177: `classes=15` #Numero de clases de entrenamiento

4.7. Ejecución del entrenamiento

Comando para iniciar el entrenamiento:

```
>cd darknet
```

```
>./darknet detector train home/edd/yolov3_tiny/yolov3_tiny.data
home/edd/yolov3_tiny/yolov3_tiny.cfg home/edd/yolov3_tiny/yolov3_tiny.conv.15
```

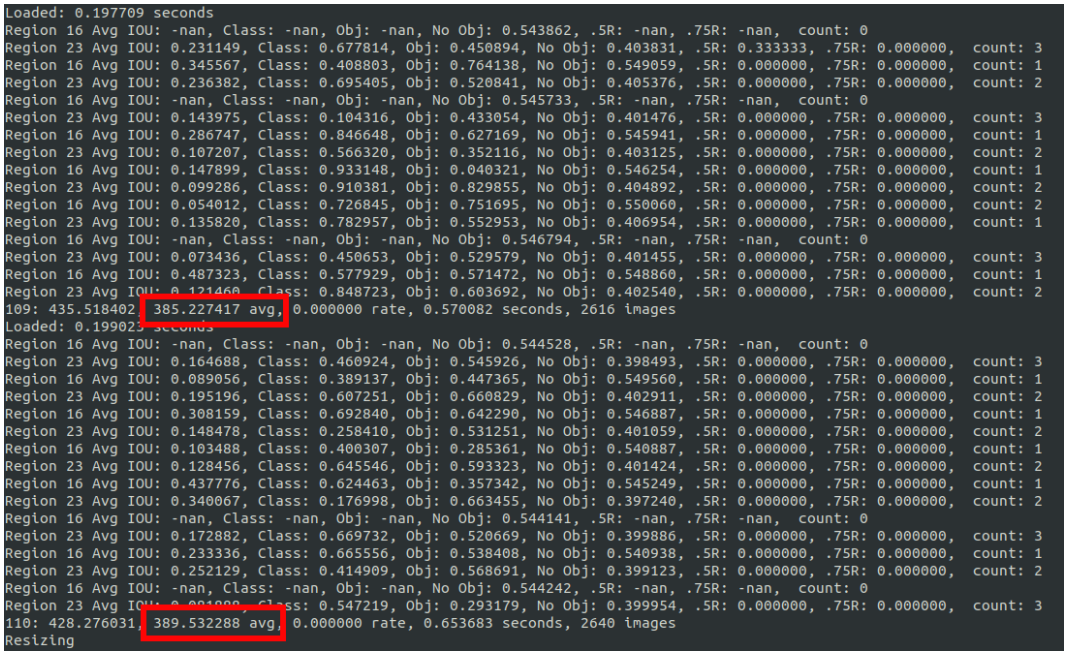


Figura 4.2: Entrenamiento de YOLO

El entrenamiento termina a las 500200 iteraciones (se puede modificar en el archivo .cfg). Si el error promedio es bajo (menor a 5.0) y no cambia durante un periodo de tiempo, el entrenamiento se puede parar (control + c) (figura 4.2).

Bibliografía

[1] E. Silva. "Detección de Objetos con Redes Neuronales Profundas para un Robot de Servicio". Tesis de mtría. Facultad de Ingeniería, Despartamento de Procesamiento Digital de Señales. Universidad Nacional Autónoma de Méxcio, 2020.

[2] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[3] Raúl Rojas. *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996. ISBN: 3-540-60505-3.

[4] Pawan Jain. *Complete Guide of Activation Functions*. 2019. URL: `https://towardsdatascience.com/complete-guide-of-activation-functions` (visitado 08-08-2019).

[5] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks*. 2018. URL: `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way` (visitado 08-08-2019).

[6] Sabyasachi Sahoo. *Residual blocks — Building blocks of ResNet*. 2018. URL: `https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet` (visitado 08-08-2019).

[7] Simeon Kostadinov. *Understanding Backpropagation Algorithm*. 2019. URL: `https://towardsdatascience.com/understanding-backpropagation-algorithm` (visitado 08-08-2019).

[8] Standford Vision Lab. *ImageNet*. 2016. URL: `http://image-net.org` (visitado 08-08-2018).

[9] Tsung-Yi Lin y col. "Microsoft COCO: Common Objects in Context". En: *European Conference on Computer Vision (ECCV)*. Oral. Zürich, 1 de ene. de 2014. URL: `/se3/wp-content/uploads/2014/09/coco_eccv.pdf,http://mscoco.org`.

[10] Na8. *Qué es overfitting y underfitting y cómo solucionarlo*. 2019. URL: https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/ (visitado 08-08-2019).

[11] Joseph Redmon y Ali Farhadi. "YOLOv3: An Incremental Improvement". En: *arXiv* (2018).

[12] Python Lessons. *YOLOv3 theory explained*. 2018. URL: https://medium.com/@pythonlessons0/yolo-v3-theory-explained (visitado 08-08-2019).

[13] Rafael C. Gonzalez y Richard E. Woods. *Digital image processing*. Prentice Hall, 2008.

[14] RoboCup Federation. *RoboCup@Home*. 2016. URL: https://athome.robocup.org/ (visitado 08-08-2018).

[15] Redmon Joseph. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016.

[16] Ui Yamaguchi y col. "HSR, Human Support Robot as Research and Development Platform". En: *The Abstracts of the international conference on advanced mechatronics : toward evolutionary fusion of IT and mechatronics : ICAM* 2015.6 (dic. de 2015), págs. 39-40. DOI: 10.1299/jsmeicam.2015.6.39.

[17] Matthew D. Zeiler y Rob Fergus. "Visualizing and Understanding Convolutional Networks". En: *CoRR* abs/1311.2901 (2013). arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901.

[18] Alexey A.B. *Yolo Mark*. https://github.com/AlexeyAB/Yolo_mark. 2019.

[19] Asus. *Xtion-PRO LIVE*. 2019. URL: https://www.asus.com/3D-Sensor/XtionPRO (visitado 08-08-2019).

[20] Sony. *Sony DSC-RX100M5*. 2019. URL: https://www.sony.com.mx (visitado 08-08-2019).

[21] Takeshi Team. *HSR-Challenge 3. Facultad de Ingeniería de la UNAM*. 2019. URL: https://www.youtube.com/watch?v=et75sn1KkZE (visitado 13-04-2019).

[22] Ramesh Jain, Rangachar Kasturi y Brian Schunck. *Machine Vision*. Ene. de 1995. ISBN: 978-0-07-032018-5.

[23] RoboCup Federation. *RoboCup*. 2016. URL: https://www.robocup.org/ (visitado 08-08-2018).

[24] Bjelonic Marko. *YOLO ROS: Real-Time Object Detection for ROS*. https://github.com/leggedrobotics/darknet_ros. 2016–2018.

[25] Federación Mexicana de Robótica. *Torneo Mexicano de Robótica*. 2016. URL: https://www.femexrobotica.org/tmr2019/ (visitado 08-08-2018).

[26] World Robot Summit. *WRS*. 2016. URL: https://worldrobotsummit.org/en/ (visitado 08-08-2018).

[27] Savage Carmona Jesús. *Laboratorio de Bio-Robótica de la Facultad de Ingeniería de la UNAM*. 2016. URL: https://biorobotics.fi-p.unam.mx/ (visitado 08-08-2018).

[28] Olga Russakovsky y col. *Imagenet large scale visual recognition challenge 2013 (ilsvrc2013)*. 2013. URL: http://www.image-net.org/challenges/LSVRC/2013/. (visitado 08-08-2018).

[29] iRobot. *iRobot Vacuum Cleaning*. 2018. URL: https://www.irobot.mx/ (visitado 08-08-2018).

[30] ROS. *ROS (Robot Operating System*. 2018. URL: https://www.ros.org/ (visitado 08-08-2018).

[31] OpenCV. *Open Source Computer Vision Library*. 2019. URL: https://opencv.org/ (visitado 08-08-2019).

[32] Molinero Díez Gregorio. "Segmentación de imágenes en color basada en el crecimiento de regiones." Tesis doct. Universidad de Sevilla, Departamento de ingeniería, 2010.

[33] George Stockman y Linda G. Shapiro. *Computer Vision*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN: 0130307963.

[34] Cruz Navarro Jesús. "Detección y reconocimiento de objetos usando imágenes RGB y nubes de puntos organizadas". Tesis doct. Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS). Universidad Nacional Autónomas de México, 2016.

[35] Jungong Han y col. "Enhanced Computer Vision with Microsoft Kinect Sensor: A Review". En: *IEEE Trans. Cybernetics* 43 (2013). URL: https://www.microsoft.com/en-us/research/publication/enhanced-computer-vision-with-microsoft-kinect-sensor-a-review/.

[36] David A. Forsyth y Jean Ponce. *Computer Vision - A Modern Approach, Second Edition*. Pitman, 2012, págs. 1-791. ISBN: 978-0-273-76414-4.

[37] Lih-Jen Kau y Tien-Lin Lee. "An Efficient and Self-Adapted Approach to the Sharpening of Color Images". En: *TheScientificWorldJournal* 2013 (nov. de 2013), pág. 105945. DOI: 10.1155/2013/105945.

[38] Kurt Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". En: *Neural Netw.* 4.2 (mar. de 1991), págs. 251-257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: http://dx.doi.org/10.1016/0893-6080(91)90009-T.

[39] Zachary Chase Lipton. "A Critical Review of Recurrent Neural Networks for Sequence Learning". En: *CoRR* abs/1506.00019 (2015). arXiv: 1506.00019. URL: http://arxiv.org/abs/1506.00019.

[40] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". En: *Psychological Review* (1958), págs. 65-386.

[41] Nicholas T. Carnevale y Michael L. Hines. *The NEURON Book*. New York, NY, USA: Cambridge University Press, 2006. ISBN: 0521843219.

[42] Kaiming He y col. "Mask R-CNN". En: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), págs. 2980-2988.

[43] Razavian Ali Sharif y col. "CNN Features off-the-shelf: an Astounding Baseline for Recognition". En: *CoRR* abs/1403.6382 (2014). URL: http://arxiv.org/abs/1403.6382.

[44] Pierre Sermanet y col. "Overfeat: Integrated recognition, localization and detection using convolutional networks". English (US). En: *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*. 2014.

[45] J Canny. "A Computational Approach to Edge Detection". En: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (jun. de 1986), págs. 679-698. ISSN: 0162-8828.