Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

UNIDAD ZACATENCO
DEPARTAMENTO DE COMPUTACIÓN

Reconocimiento de marcadores con redes profundas

Tesis que presenta

Gonzalo Adán Chávez Fragoso

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. Luis Gerardo de la Fraga

Ciudad de México, México Septiembre, 2020.

Resumen

El reconocimiento de marcadores fiduciales es una parte importante de varias tareas dentro del área de visión por computadora, ente las que se encuentran la realidad aumentada, la localización y la navegación de robots. Aunque los métodos de reconocimiento de marcadores basados en técnicas clásicas de procesamiento de imágenes y reconocimiento de patrones muestran una gran exactitud, aún existen problemas abiertos por resolver, tales como la robustez frente a oclusión o la capacidad de reconocer múltiples marcadores en la misma escena. Esto sin olvidar el requerimiento de procesamiento en tiempo real. Recientemente las redes neuronales profundas han mostrado una gran superioridad, respecto a métodos clásicos, en tareas de visión como clasificación, detección y segmentación de objetos.

El propósito de esta tesis es usar redes neuronales profundas para reconocer marcadores fiduciales y probar su efectividad para resolver los problemas descritos anteriormente. Nuestro trabajo comprende la comparación de redes neuronales convolucionales profundas para la detección de objetos que ofrecen un nivel competitivo de precisión y adicionalmente una ejecución en tiempo real. Se realizan pruebas de varias arquitecturas de redes neuronales convolucionales profundas tipo YOLO usando pesos pre-entrenadas, midiendo su tiempo de ejecución en un procesador utilizando un solo núcleo. Construimos conjuntos de entrenamiento para probar su desempeño en el reconocimiento de un marcador fiducial captado mediante la cámara web PlayStation Eye. Se muestra la necesidad del uso de una GPU para acelerar el entrenamiento de las redes profundas dado que el tiempo de entrenamiento usando únicamente CPU no es manejable. Finalmente se proponen varias modificaciones a las arquitecturas de redes neuronales profundas para mejorar su tiempo de procesamiento considerando el cambio en precisión de los resultados.

Abstract

Fiducial marker recognition is an important part in several computer vision tasks, such as augmented reality, location, and robot navigation. Although markers' recognition methods based on classical image processing and pattern recognition techniques show great accuracy, there are still open problems to solve, such as robustness to occlusion or the ability to recognize multiple markers at the same time, within a given scene, particularly without removing the requirement of real-time processing. Recently deep neural networks have shown great superiority, compared to classical methods in vision tasks such as object classification, detection and segmentation.

The purpose of this thesis is to use deep neural networks to recognize fiducial markers and test their effectiveness in solving the problems described above. Our work comprises the comparison of deep convolutional neural networks for object detection which offer a competitive level of accuracy while allowing real-time execution. Several YOLO-type deep neural network architectures are tested using pre-trained weights, measuring their execution time on a processor using a single core. We build training sets to test its performance on the recognition of a fiducial marker captured by the PlayStation Eye webcam. The need to use a GPU to speed up the training of deep neural networks is made evident, since training using only a CPU is not feasible. Finally, several modifications to deep neural network architectures are proposed to improve their execution time, by taking into account a change of accuracy of the results.

Agradecimientos

Al CINVESTAV

Agradezco al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV), en particular al Departamento de Computación por aceptarme en el programa de maestría en ciencias en computación, lo que contribuyó en gran medida a mi desarrollo tanto profesional como personal.

Agradezco a mi asesor el Dr. Luis Gerardo de la Fraga por guiarme en la realización de este trabajo, por instruirme con firmeza y paciencia, sobre todo por ayudarme a corregir mis errores y a superar mis deficiencias.

A los profesores del departamento de computación por los conocimientos que me impartieron, así como al personal administrativo por siempre estar presente haciendo mi estancia en esta institución una gran experiencia.

A mis padres

A mi mamá Juana Fragoso y mi papá Juan Chávez, por darme las bases para forjar mi futuro, por el apoyo incondicional que siempre he recibido de su parte, quienes con sus consejos me han ayudado a tomar las decisiones correctas y han estado junto a mi, apoyándome y brindándome lo mejor de ellos.

A Luisa

Por acompañarme en esta etapa, por todos los momentos que has estado a mi lado, conservaré esos recuerdos por siempre. Gracias por el apoyo que me brindaste en todo momento, por tu cariño y confianza, eres una inspiración para ser mejor cada día.

Al CONACyT

Agradezco al Consejo Nacional de Ciencia y Tecnología por la beca que me brindó durante estos dos años de estudios de maestría.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En esta tesis se detectará la imagen de un marcador por medio de redes profundas. Primero se describirán las distintas arquitecturas de redes profundas que se han usado en el estado del arte. Posteriormente, se definirá el problema que se quiere resolver, los objetivos generales y específicos del trabajo. Finalmente, en este capítulo se describe la organización de esta tesis.

1.1. Estado del arte

El desarrollo de algoritmos para la detección de objetos se ha visto acelerado gracias a la existencia de distintos conjuntos de imágenes etiquetadas en los cuales probar el desempeño de dichos algoritmos. Aunado a esto se encuentran competencias anuales basadas en estos conjuntos de datos, en las que diversos equipos académicos presentan sus avances. Entre los más importantes se encuentran el PASCAL Visual Objects Classes (VOC) Challenge de 2005 a 2012 [2, 3], ImageNet Large Scale Visual Recognition Challenge (ILSVRC) de 2013 a 2017 [4], Microsoft Common Objects in Context (MS COCO)[5] de 2015 a la actualidad y Open Images Challenge de 2018 a la actualidad.

1.1.1. PASCAL VOC Challenge 2005-2012

En esta competencia se tiene un conjunto de imágenes con etquetas que indican los objetos de ciertas clases que están presentes y también su posición. Lo primero permite probar métodos de clasificación y lo segundo permite probar la localización o detección. Las clases tomadas en cuenta fueron 4 en 2005, 10 en 2006 y 20 a partir de 2007. La cantidad de imágenes etiquetadas fue desde 1578 en 2007 hasta 11530 en 2012 [2].

Para evaluar los resultados de distintos métodos en la tarea de detección de objetos, se utiliza la métrica *precisión promedio* (PP). Para la tarea de detección de objetos los participantes proporcionan una caja delimitadora para cada detección, con un nivel de

confianza para cada caja contenedora, es decir un valor entre 0 y 1 que indica qué tan seguro está el modelo de que el resultado es correcto. El nivel de confianza permite ordenar las detecciones de tal manera que que se puede evaluar el intercambio entre falsos positivos y falsos negativos. Se debe calcular la curva entre precisión y exhaustividad. La exhaustividad (E) se define como la proporción de los ejemplos positivos ordenados sobre una posición dada. La precisión (P) es la proporción de todos los ejemplos sobre esa posición que fueron clasificados correctamente como positivos verdaderos. Para esto, se deben introducir criterios de clasificación que se muestran en la tabla 1.1.

Tabla 1.1: Criterios para un método de clasificación binaria.

Concepto	Explicación
PV: Positivos verdaderos	Son las muestras positivas que fueron predichos como positivas
NV: Negativos verdaderos	Son las muestras negativas que fueron predichas como negativas
PF: Falsos positivos	Son las muestras negativas que fueron predichas como positivas
NF: Falsos negativos	Son las muestras positivas que fueron predichas como negativas

Para la detección de objetos, una predicción se considera positiva cierta si la intersección sobre la unión (del inglés *Intersection over Union* (IoU)) de la caja contenedora predicha y la caja contenedora de verdad fundamental es mayor que 0.5 y corresponde a la misma clase. Los negativos verdaderos no se toman en cuenta, puesto que no se lleva registro de objetos que no pertenecen a las clases preestablecidas y las predicciones solo se centran en casos positivos. Una predicción se considera positiva falsa cuando la intersección sobre la unión de la caja predicha y la caja verdadera de referencia de un objeto es menor que 0.5 aunque la clase sea correcta. Cuando la clase predicha sea incorrecta, sin importar el valor de intersección sobre la unión, será considerado como positiva falsa. Finalmente se consideran negativos falsos cuando existen objetos para los cuales no existe ninguna caja contenedora de predicción que tenga intersección con la caja verdadera de referencia. Es decir, cuando un objeto presente en una imagen no es detectado.

La intersección sobre la unión es la cantidad de superposición entre dos cajas contenedoras. En la ecuación 1.1 se la define de intersección sobre la unión de la caja contenedora B y la caja contenedora verdadera de referencia B_g. Se puede ver un diagrama en la figura 1.1. Cuando dos cajas contenedoras no se sobreponen la IsU es 0, y cuando coinciden exactamente, este valor es 1.

$$IsU(B, B_g) = \frac{área(B \cap B_g)}{área(B \cup B_g)} \tag{1.1}$$

Teniendo en cuenta los criterios de clasificación descritos, se define la exhaustividad y la precisión en las ecuaciones 1.2 y 1.3, respectivamente.

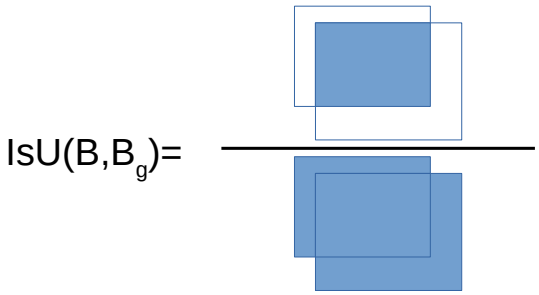


Figura 1.1: Diagrama de la intersección sobre la unión de dos cajas delimitadoras.

$$E = \frac{PV}{PV + PF} \tag{1.2}$$

$$P = \frac{PV}{PV + NF} \tag{1.3}$$

La PP resume la forma de la curva de precisión/exhaustividad, y se define como la precisión promedio en un conjunto de once niveles de exhaustividad igualmente espaciados $[0, 0.1, \cdots, 1]$:

$$PP = \frac{1}{11} \sum_{r \in \{0, 0.1, \cdots, 1\}} P_{interp}(r) \tag{1.4}$$

La precisión en cada nivel de exhaustividad r es interpolada. Es decir se toma la precisión máxima medida por un método para el cual la exhaustividad correspondiente excede r:

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}) \tag{1.5}$$

donde $p(\tilde{r})$ es la precisión medida en una exhaustividad $\tilde{r}$.

De esta manera, la PP mide la calidad de detección de los objetos de una clase dada. Ésta es cero cuando ningún objeto fue detectado y es 1 cuando todos los objetos son detectados y las cajas delimitadoras obtenidas concuerdan exactamente con las cajas contenedoras verdaderas de referencia.

La mayoría de métodos de detección de objetos entre 2005 y 2007 usaban el enfoque de "ventanas deslizante", que consiste en tomar ventanas rectangulares de la imagen, extraer sus características, y posteriormente clasificarlas como contenedoras de una instancia de una clase dada o no. Las características consistían, en su mayoría, en variaciones del método de bolsa de palabras visuales: se calculan características locales, por ejemplo descriptores invariantes a la escala SIFT (del inglés, *Scale Invariant Feature Transform*) [6], se cuantizan vectores en un vocabulario visual y, posteriormente, cada ventana es representada como un histograma de la frecuencia en que las características son asignadas

a cada palabra visual. El clasificador es típicamente una máquina de vectores de soporte con χ^2 o un núcleo con la métrica de Wasserstein. Este clasificador es después ejecutado exhaustivamente sobre la imagen variando la posición y la escala. Con el fin de lidiar con múltiples detecciones cercanas usualmente se aplica una etapa de "supresión de no-máximos".

No es posible determinar cuál de los métodos es mejor dado que, como era permitido en la competencia, se podía seleccionar un subconjunto de las 20 clases para su evaluación. El método de Oxford [7] ganó en las 6 clases en las que participó, el método UoCTTI [8] ganó en 6 clases y el método MPI_ESSOL [9] ganó en 5. El método de Oxford obtuvo los mejores puntajes en las 6 clases en las que se sometió, con un PP que excedía sustancialmente al segundo mejor con un margen de ente 4.0 % y 11.4 %. El método UoCTTI se sometió a las 20 clases, resultando primero o segundo en 14 clases. MPI_ESSOL también se sometió a las 20 clases, pero es notorio que en algunas clases el puntaje de PP fue pobre, relativo a otros métodos.

El método de referencia para detección de objetos en VOC de 2008 a 2012 fue el modelo de partes deformables (MPD). Este método está basado en un filtro "raíz" de escala gruesa usando una representación de histogramas de gradientes orientados (HGO), además de histogramas de gradientes orientados de escala más fina que pueden moverse relativos a la raíz. Este modelo es aplicado en toda la imagen usando ventanas deslizantes eficientes. Las salidas son post-procesadas involucrando regresión para predecir las cajas contenedoras desde la raíz y las posiciones de partes, supresión de máximos codiciosa y reconstrucción de cada caja contenedora con una máquina de vectores de soporte usando información contextual sobre las fuerza máxima de detecciones para todas las clases [3].

1.1.2. ImageNet Large Scale Visual Recognition Challenge

Esta competencia se ha realizado desde el año 2010, pero solo enfocada a la clasificación de imágenes. Desde el 2011 se evalúa la localización de un solo objeto y a partir de 2013 se introduce la detección de varios objetos. Consta de 200 clases para la competencia de detección de objetos, con aproximadamente 450000 imágenes de entrenamiento, 20000 imágenes de validación y 40000 para prueba [4].

En esta competencia se introduce la medida de desempeño denominada *media de la precisión promedio* (mPP), la cual está basada en la medida PP considerada en la competencia PASCAL VOC. Simplemente se calcula el promedio de PP de todas las clases tomadas en cuenta. Esta medida resume la calidad de detección de todas las clases en un solo puntaje.

En el año 2011, el método ganador en la categoría de localización de un solo objeto utilizó búsqueda selectiva para generar regiones donde se hipotetiza la existencia de objetos, seguida de un muestreo denso y cuantización de vectores de varias características SIFT de color, agrupamiento con correspondencia espacial en pirámide y clasificación con una máquina de vectores de soporte; es decir con métodos parecidos a los vistos en PASCAL VOC. La edición de 2012 fue crucial para el reconocimiento de objetos a gran escala pues fue cuando las redes neuronales profundas mostraron una gran superioridad frente a los métodos anteriores. El ganador tanto en la clasificación de imágenes como en la localización de un solo objeto fue el equipo SuperVision [10]. Este equipo entrenó una red neuronal convolucional profunda con valores RGB, con 60 millones de parámetros usando una implementación eficiente en GPU. El tamaño de esta red fue de seis capas convolucionales.

Para 2013, la mayoría de métodos usaban redes neuronales convolucionales profundas. La red profunda OverFeat [11] participó tanto en clasificación, localización de un solo objeto y detección, siendo la mejor en localizar un solo objeto. Utilizaba una red con cinco capas convolucionales con un enfoque de ventanas deslizantes en varias escalas. El ganador en la categoría de detección de objetos también utilizó una red convolucional de clasificación de imágenes para reconstruir detecciones.

En el año 2014 se notó un gran progreso: el error en clasificación casi se redujo a la mitad y la mPP en detección de objetos casi se duplicó comparados con el año anterior. En 2014 surgieron varias de las arquitecturas de red convolucional más influyentes, que posteriormente servirían como base para el desarrollo de distintos métodos. Además de la modalidad de entrenamiento con el conjunto de imágenes proporcionado por la competencia, las cuales que pertenecen a ImageNet, se agregó una nueva modalidad que permite utilizar imágenes externas. Así se calificaron seis categorías: clasificación, localización de un solo objeto y detección, cada una de las cuales puede ser con datos proporcionados o con datos externos. La arquitectura GoogLeNet [12] obtuvo los mejores resultados en las categorías de clasificación utilizando datos proporcionados y detección de objetos con datos externos.

GoogLeNet es una arquitectura de red convolucional mucho más grande que las anteriores, consistiendo de 22 capas. Una red con gran cantidad de parámetros es propensa a sobreajuste, además de que su carga computacional es considerablemente mayor. Por lo tanto, usaron un diseño al que llamaron *Inception*, que permitió cambiar de una arquitectura totalmente conectada a una poco densa. El uso de capas adicionales con convoluciones de 1×1 para reducción de dimensión permitió aumentar tanto la profundidad como la amplitud de la red significativamente sin incurrir en un aumento en el costo computacional.

En la categoría de localización de un solo objeto con datos proporcionados en ILSVRC

2014 el ganador fue el equipo VGG [13], que exploró el efecto de la profundidad de las redes convolucionales en su precisión, usando tres diferentes arquitecturas con hasta 19 capas de pesos con una unidad lineal rectificada. Para la detección de objetos con datos proporcionados, la mejor arquitectura fue RCNN (*Regions with CNN features*) [14].

En 2015, las arquitecturas de aprendizaje profundo residual ResNet [15] obtuvieron los mejores resultados tanto en clasificación, como en localización de un solo objeto y detección, las tres con datos proporcionados. El aprendizaje profundo residual introduce capas con atajos, es decir, que permiten la comunicación de una capa con otras capas que no son inmediatamente consecutivas. Este enfoque permite entrenar redes mucho más grandes (hasta 152 capas), además de reducir su complejidad y aumentar su exactitud. Adicionalmente ResNet obtuvo el primer lugar en las categorías de detección y de segmentación en la competencia MS COCO Challenge 2015.

En el año 2016 la arquitectura GBD-Net [16] resultó ganadora en la categoría de detección de objetos tanto con datos proporcionados como con datos externos. Utiliza información local y contextual para apoyar a la detección de objetos para los que se hayan generado cajas contenedoras relativamente pequeñas. También utiliza la técnica CRAFT para proponer cajas contenedoras. En 2017, el equipo BDAT [17] usó una combinación de métodos con los que obtuvo los mejores resultados en detección de objetos en ambas modalidades.

1.1.3. Microsoft Common Objects in Context

Esta es una competencia donde se incluyen 80 clases. En 2015 contaba con 165482 imágenes de entrenamiento, 81208 para validación y 81434 para pruebas.

En esta competencia se introduce una nueva métrica denominada PP. A diferencia de PASCAL VOC y ILSVRC, toma en cuenta diversos umbrales de intersección sobre la unión para calcular los criterios de clasificación de los que se obtendrá la PP. Para las competencias anteriores este umbral era 0.5. Para MS COCO se consideran los umbrales $\{0.5, 0.55, \cdots, 0.95\}$, es decir de 0.5 a 0.95 con un tamaño de paso de 0.05. Con esto se calculan las medidas $\mathrm{PP}_{0.5}, \mathrm{PP}_{0.55}, \cdots, \mathrm{PP}_{0.95}$, cada una correspondiendo al umbral de su subíndice, y se calcula su promedio, con lo que se obtiene $\mathrm{PP}_{\mathrm{coco}}$. De esta manera, PP de PASCAL VOC es equivalente a $\mathrm{PP}_{0.5}$. Finalmente, $\mathrm{mPP}_{\mathrm{coco}}$ es el promedio de los puntajes $\mathrm{PP}_{\mathrm{coco}}$ correspondientes a todas las clases.

En 2015, la arquitectura ganadora fue ResNet y también ganó la competencia ILSVRC en ese año, la cual ya se describió anteriormente. En 2016, el equipo G-RMI (Google Research and Machine Intelligence) [18] fue el ganador en la categoría de detección de

objetos y obtuvo el segundo lugar en segmentación. Este equipo realizó un estudio de la compensación entre precisión y velocidad de distintas arquitecturas de redes profundas tanto para detección de objetos como para segmentación semántica.

La arquitectura ganadora en el año 2017 fue MegDet [19], la cual propone un entrenamiento por lotes con tamaño de lote más grandes que en arquitecturas previas, apoyándose de normalización por lotes en varios dispositivos, lo que logra acelerar el entrenamiento. Para 2018, los mejores métodos como Megavii y MMDet [20] utilizan la combinación de múltiples arquitecturas y técnicas para mejorar el entrenamiento.

1.1.4. Compensación entre precisión y velocidad

Tabla 1.2: Comparación de desempeño de los mejores métodos por año en distintas competencias.

Método	Datos	Año	mPP
UvA	ImageNet	2013	0.23
Overfeat	ImageNet	2013	0.19
GoogLenet	ImageNet+ext	2014	0.44
R-CNN	ImageNet	2014	0.37
ResNet	ImageNet	2015	0.62
GBD-Net	ImageNet	2016	0.66
BDAT	ImageNet	2017	0.73
ION	MS COCO	2015	0.31
FAIRCNN	MS COCO	2015	0.33
ResNet	MS COCO	2015	0.37
Trimps-Soushen	MS COCO	2016	0.36
G-RMI	MS COCO	2016	0.41
MegDet	MS COCO	2017	0.53
Megvii	MS COCO	2018	0.55
MMDet	MS COCO	2018	0.56
Avengers	Open Images	2018	0.5862
PFDet	Open Images	2018	0.5863
kivajok	Open Images	2018	0.5866
Prism	Open Images	2019	0.6421
Imagesearch	Open Images	2019	0.6533
MMFruit	Open Images	2019	0.6588

En la tabla 1.2 se resumen las mejores arquitecturas de redes convolucionales para la detección de objetos, señalando el conjunto de datos usados para su entrenamiento y su

evaluación, el año en el que se presentaron y su nivel de precisión de acuerdo con la competencia.

Sin embargo, el objetivo de estas arquitecturas es obtener los puntajes más altos sin tomar en cuenta otros factores como la velocidad de procesamiento o la memoria utilizada. La primera red profunda de detección de objetos que se centró en la optimización de esos aspectos fue RCNN, la cual obtuvo los mejores resultados en 2014. Como resultado de esas optimizaciones surgieron Fast RCNN [21] y Faster RCNN [22]. Posteriormente, surgieron múltiples arquitecturas con optimizaciones que perseguían la detección de objetos con una velocidad mayor o igual que 30 marcos por segundo. Entre éstas se encuentran SPPNet[23], SSD [24], RetinaNet [25], Yolo [26], YoloV2 [27], YoloV3 [28] y YoloV4 [29].

1.2. Definición del problema

Se hipotetiza que el uso de una red profunda permitirá la detección de marcadores en imágenes, brindando precisión, robustez a la oclusión y, adicionalmente ,ejecución en tiempo real.

En este contexto, tiempo real significa que la red será capaz de procesar una imagen de entrada cada 1/30 o 1/60 de segundo.

Detectar un marcador cuando se oclusiona con un dedo o con otros objetos es un problema muy difícil de resolver con técnicas clásicas de procesamiento de imágenes, dado que se presentan múltiples casos que se deben resolver.

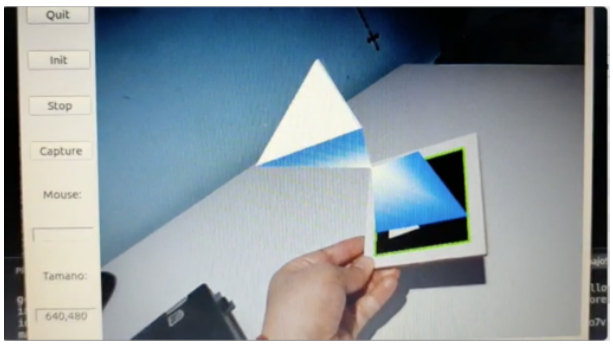


Figura 1.2: Fotograma de un sistema de realidad aumentada usando un marcador.

En la figura 1.2 se muestra un fotograma de un sistema de realidad aumentada diseñado en el curso de Visión del año 2020 [30]. En esta figura se ve un objeto virtual dibujado sobre el marcador. El objeto virtual son dos tetraedros encontrados. La detección del marcador consiste en encontrar en la imagen el objeto oscuro con cuatro esquinas y un triángulo

dentro de él. Este sistema es semejante al desarrollado en la tesis de maestría de Daybelis Jaramillo Olivares [31]. Estos sistemas usan técnicas convencionales de procesamiento de imágenes y logran una frecuencia de procesamiento de 30 cuadros (imágenes) por segundo.

1.3. Objetivos generales y específicos del proyecto

General

Uso de técnicas de aprendizaje profundo para resolver el problema de detección de marcadores fiduciales basados en tipo de orden [32, 33].

Específicos

1. Estudio de las arquitecturas de redes profundas que podrían resolver el problema.
2. Análisis del costo computacional de esas arquitecturas.
3. Elección y entrenamiento de una red profunda.
4. Pruebas de la red elegida.
5. Medición del costo computacional de los sistemas tradicionales y de la solución que usa una red profunda.

1.4. Organización de esta tesis

Esta tesis consta de cinco capítulos. En este primer capítulo se proporciona una revisión del estado del arte sobre los métodos de detección de objetos, se define el problema y se establecen los objetivos generales y específicos de esta tesis.

El segundo capítulo presenta los conceptos relacionados con el trabajo desarrollado. Se define la operación de convolución, que es el elemento principal de las redes neuronales convolucionales, de las cuales también se describen sus demás componentes. También se hace la distinción entre varias tareas de reconocimiento de objetos en imágenes como clasificación, localización de un solo objeto, detección, segmentación semántica y segmentación de instancias. Posteriormente, se describen los marcadores basados en tipo de orden y el problema de emparejamiento de puntos. Finalmente, se describen las redes profundas YOLO, las cuales fueron seleccionadas para la solución del problema de detección de marcadores fiduciales.

En el tercer capítulo se describen los pasos que se llevaron a cabo utilizando las redes YOLO para resolver el problema de interés. Se comparan resultados en términos de exactitud y velocidad de procesamiento y las modificaciones realizadas a distintas versiones de arquitecturas de redes profundas.

En el último capítulo se presentan las conclusiones de este trabajo de tesis y se describen algunas posibles sugerencias de trabajo futuro.

Capítulo 2

Antecedentes

En esta capítulo se describirán los conceptos necesarios para entender el problema que se resolverá. Se describirán la operación de convolución y las redes neuronales que la usan. También se describirán los marcadores fiduciales basados en tipo de orden, que sirven para hallar la correspondencia de puntos entre el modelo del marcador y la imagen que toma una cámara de él. Finalmente, en este capítulo se describirán las redes YOLO que fueron seleccionadas para resolver el problema de detección de marcadores de tipo de orden.

2.1. Convolución

La convolución es una operación que tiene como argumentos dos funciones con valores reales. La operación de convolución es típicamente denotada con un asterisco:

$$s(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau, \tag{2.1}$$

donde la función s es resultado de la convolución de las funciones f y g, $f, g : \mathbb{R} \longrightarrow \mathbb{R}$. En la terminología de las redes convolucionales, la primera función f en (2.1) es la entrada y la segunda función, g, es el núcleo (*kernel* en inglés). Al resultado, s, algunas veces se le llama **mapa de características**.

Frecuentemente se usa la convolución sobre más de un eje a la vez. Por ejemplo, para una imagen de dos dimensiones I, con ancho w y altura h como entrada, también se usa un núcleo K de dos dimensiones, con ancho w_k y altura h_k:

$$S(i,j) = I(i,j) * K(i,j) = \sum_{m=1}^{w_k} \sum_{n=1}^{h_k} I(m,n)K(i-m, j-n).1 \leq i \leq w, 1 \leq j \leq h. \tag{2.2}$$

La convolución es conmutativa, lo que significa que podemos escribir equivalentemente:

$$S(i,j) = I(i,j) * K(i,j) = \sum_{m=1}^{w_k} \sum_{n=1}^{h_k} I(i-m, j-n)K(m,n).1 \leq i \leq w, 1 \leq j \leq h. \quad (2.3)$$

Usualmente, la última fórmula es más sencilla de implementar en una biblioteca de aprendizaje automático, porque el tamaño donde está definido el núcleo es mucho menor que el tamaño de la entrada. En el estado del arte usualmente se usan núcleos de tamaño 3×3 [26, 13, 24].

La convolución en (2.2) gira el núcleo con respecto a la entrada, en el sentido de que mientras m se incrementa, el índice en la entrada aumenta, pero el índice en el núcleo disminuye. Generalmente, el núcleo es simétrico, por lo que muchas bibliotecas de redes neuronales implementan una función relacionada llamada correlación cruzada, la cual es igual a la convolución cuando el núcleo es simétrico (y el núcleo no se gira):

$$S(i,j) = I(i,j) * K(i,j) = \sum_{m=1}^{w_k} \sum_{n=1}^{h_k} I(i+m, j+n)K(m,n).1 \leq i \leq w, 1 \leq j \leq h. \quad (2.4)$$

En la ecuación (2.5) se muestra un ejemplo de la operación de convolución con tamaño de paso de 1 píxel entre una imagen de entrada con dimensiones de 5×6 con un núcleo de dimensiones 3×3, lo que da como resultado un mapa de características con dimensiones de 3×4. Los números en negritas en la entrada indican las posiciones por las cuales se sobrepone el centro del núcleo.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 3 & 1 \\ 0 & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{3} & 1 \\ 0 & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{3} & 1 \\ 0 & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{3} & 1 \\ 0 & 1 & 2 & 3 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 2 \end{bmatrix} \quad (2.5)$$

La primera salida 0 se calcula multiplicando los tres primeros renglones y columnas de la entrada por cada valor correspondiente del núcleo y se suman las multiplicaciones: $0 = (0)(0) + (1)(-1) + (2)(0) + (0)(-1) + (1)(4) + (2)(-1) + (0)(0) + (1)(-1) + (2)(0) = 0 - 1 + 0 + 0 + 4 - 2 + 0 - 1 + 0$. Los números de la entrada en negritas representan donde se puede calcular la salida.

Se pueden hacer dos observaciones respecto a la convolución de la ecuación (2.5). En primer lugar el núcleo se desplazó por la entrada con tamaño de paso de un píxel. Por otra parte, el mapa de características resultante tiene unas dimensiones menores a las de la entrada. Ésta es una propiedad inherente de la función de convolución: si se tiene

una entrada con dimensión m y un kernel de dimensión k (suponiendo ambos cuadrados), entonces la dimensión de la salida será de $m - k + 1$. Tal reducción en la dimensión del mapa de características limitaría la cantidad de capas a apilar para la construcción de una red neuronal convolucional pues eventualmente se llegará a un mapa de características de una dimensión.

En las imágenes a color, cada píxel contiene tres valores que corresponden a las intensidades de rojo, verde y azul. Eso también se puede ver como si una imagen a color consistiera de tres matrices, una para cada color, o en términos tensoriales, como un tensor con alto y ancho iguales a los de la imagen, pero una profundidad de 3. La aplicación de la operación de convolución correspondiente a imágenes a color consiste en utilizar un núcleo para cada uno de los canales, lo cual, en términos tensoriales, es un tensor de alto 3, ancho 3 y profundidad 3. En este caso, el resultado de la convolución será la suma del producto de los valores de la entrada en sus tres canales, con los valores de los núcleos correspondientes, es decir, la suma de los resultados de la convolución de cada canal con el filtro correspondiente. En redes convolucionales se le conoce como filtro a un conjunto de núcleos bidimensionales, donde la cantidad de núcleos corresponde a la profundidad del filtro si se le ve como un tensor. Se debe notar que la profundidad de la imagen y del filtro deben coincidir para que se pueda aplicar la operación de convolución. También se le llama filtro a la aplicación de un núcleo. Esto viene del procesamiento de imágenes y procesamiento de señales, donde la aplicación de un núcleo tiene similitudes a la aplicación de un filtro en el espacio de frecuencias [6].

2.2. Red neuronal convolucional

Las redes convolucionales son un tipo especializado de redes neuronales que procesan datos que tienen una topología tipo malla. Un ejemplo son las series de tiempo, las cuales pueden pensarse como mallas de una sola dimensión, en las que cada dato son muestras que se toman en intervalos regulares de tiempo. Las imágenes pueden pensarse también como una malla cuadrada de píxeles (en dos dimensiones). Las redes convolucionales reciben su nombre de la operación matemática llamada **convolución**. Las redes convolucionales son simplemente redes neuronales que usan la convolución en lugar de la multiplicación general de matrices en por lo menos una de sus capas [34].

Las redes convolucionales en general consisten en etapas de convolución, etapas de aplicación de funciones no lineales y etapas de agrupación. Estas capas, son comúnmente apiladas una después de la otra para formar modelos más profundos. Su principal función es la extracción de características de los datos de entrada, y estas redes tienen en su salida final una o más capas de redes neuronales comunes, llamadas capas totalmente conecta-

das, con el propósito de producir predicciones, por ejemplo de la clase a la que pertenece una imagen en un problema de clasificación. En la figura 2.1 se muestra una arquitectura típica de una red neuronal convolucional para la clasificación de imágenes.

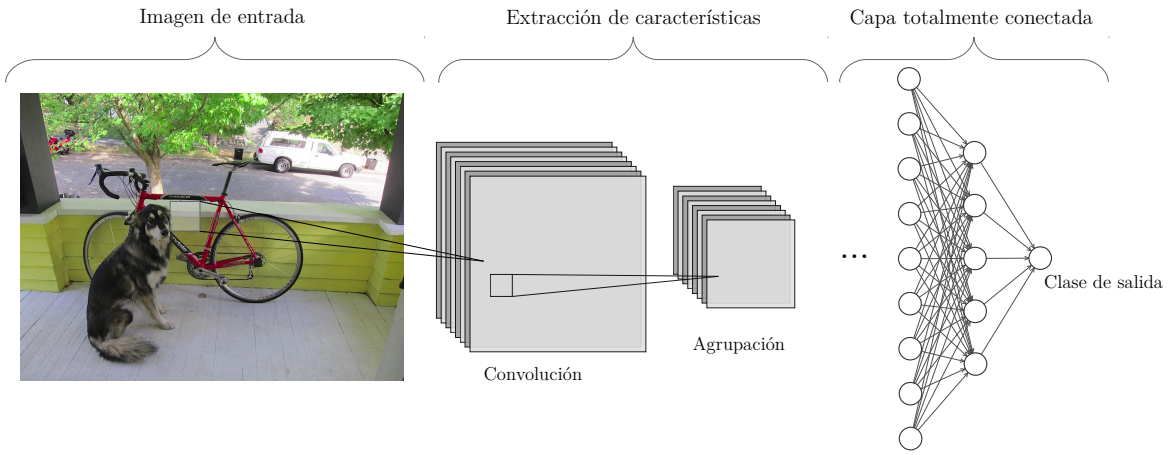


Figura 2.1: Arquitectura típica de una red neuronal convolucional para la clasificación de imágenes.

Una capa convolucional consiste en la aplicación de varios filtros a una imagen de entrada, los mapas de características resultantes se apilan, con lo que se crea un tensor que tiene una profundidad igual a la cantidad de filtros aplicados. Este tensor, que es resultado de la ejecución de una capa convolucional, sirve como entrada de la siguiente capa convolucional en la que los filtros deberán tener la profundidad adecuada para poder aplicar la operación de convolución.

En la figura 2.2, se muestra un ejemplo de los efectos en la profundidad al usar distintas capas convolucionales a una imagen de entrada de tamaño 16×16. Como entrada se tiene una imagen en escala de grises, por lo tanto con una profundidad de 1. Posteriormente, se le aplica una capa convolucional de 4 filtros de tamaño 3×3, con lo que el resultado es un mapa de características de profundidad 4 de tamaño $16 - 3 + 1 = 14$, esto es 14×14. Después se le aplica una capa de 8 filtros y al final se extrae un vector de características. El tamaño del mapa de características resultado de la aplicación de la segunda capa es $14 - 3 + 1 = 12$, esto es 12×12. El vector de salida es de $12 \times 12 \times 8 = 1152$. Comúnmente, la cantidad de filtros aumenta en cada capa aplicada, con el propósito de extraer características complejas. Sin embargo, de la manera que se muestra en la figura 2.2, al final quedaría un tensor muy profundo y del mismo tamaño que la entrada. Una función de agrupación (que se explicará en la siguiente subsección) ayudará a reducir las dimensiones de los mapas de características.

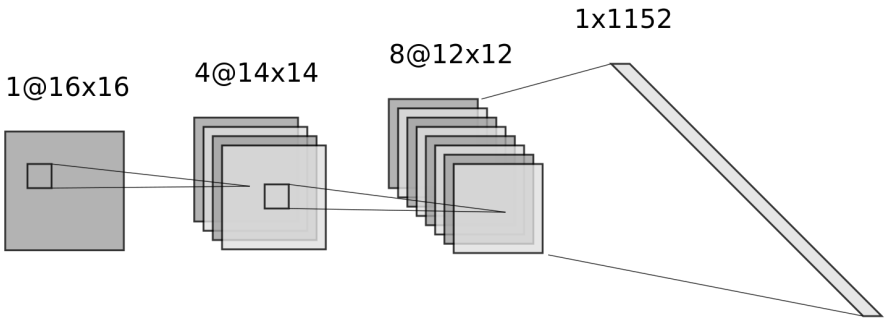


Figura 2.2: Ejemplo del cambio de profundidad tras aplicar capas convolucionales a una entrada. Los filtros aplicados son todos de tamaño 3×3.

2.2.1. Agrupación

Una función de agrupación reemplaza la salida de una red en cierta ubicación con una estadística que resume las salidas de una vecindad. Por ejemplo, la operación agrupación max reporta la salida máxima en una vecindad rectangular. Otras funciones de agrupación populares incluyen la agrupación promedio de una vecindad rectangular, la norma L^2 de una vecindad rectangular, o un promedio ponderado según la distancia al píxel central. En todos los casos, la agrupación ayuda a hacer que las representaciones se vuelvan aproximadamente invariantes a traslaciones pequeñas en la entrada.

Las funciones de agrupación más usadas son la agrupación max y la agrupación promedio. La agrupación max devuelve el máximo de una vecindad de la ubicación indicada mientras que la agrupación promedio devuelve el promedio de todos los valores en la porción de la imagen correspondiente. Además, estas funciones se usan comúnmente con un paso mayor a uno con lo que permiten que el mapa de características resultante tenga unas dimensiones menores que el de entrada. En la figura 2.3 se muestra el resultado de la aplicación de agrupación max y agrupación promedio con una ventana de 2×2 con un paso de 2 píxeles sobre la misma región.

Con las capas convolucionales y las capas de agrupamiento se pueden construir redes profundas para la extracción de características complejas de imágenes en una cantidad mucho menor a la cantidad original de píxeles que contiene cada imagen. En la figura 2.4 se muestra una arquitectura de red convolucional que usa ambos elementos para que al final se obtenga un conjunto de 2304 características. Estas características están agrupadas como vector para alimentar una capa de predicción, como puede ser una red neuronal totalmente conectada.

El primer resultado notable que se obtuvo con una red convolucional fue su aplicación para el reconocimiento de dígitos escritos a mano descrita por Le Cun en 1989 [35], usado

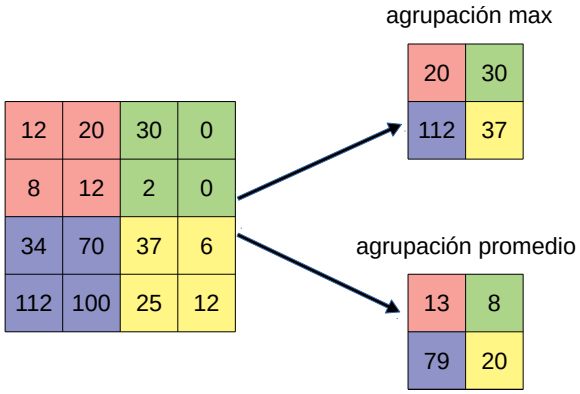


Figura 2.3: Comparación del resultado de la agrupación max y agrupación promedio.

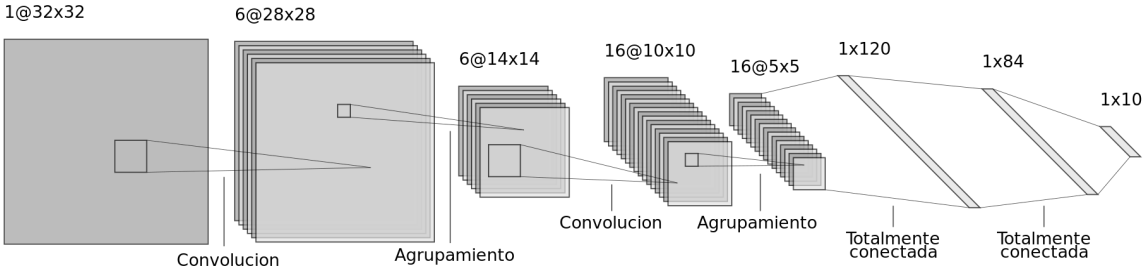


Figura 2.4: Ejemplo de la arquitectura de red convolucional LeNet-5, utilizada para el reconocimiento de dígitos escritos a mano [1].

en el servicio postal de los Estados Unidos, con lo que surgió un conjunto de datos de 70000 imágenes llamado MNIST con el que se probaban modelos en la tarea de clasificación de imágenes. Posteriormente, surgieron otros conjuntos de imágenes de libre acceso para la prueba de precisión de modelos de redes convolucionales en distintas tareas como clasificación, detección de objetos y segmentación entre otras. Entre estos se encuentran CIFAR [36], ImageNet [37], PASCAL VOC [2] y MS COCO [5].

2.3. El problema de reconocimiento

Los humanos usan sus ojos y su cerebro para ver y entender el mundo tridimensional que los rodea. La visión por computadora es una disciplina que pretende proporcionar a las computadoras capacidades de reconocimiento de imágenes similares, si no es que superiores, a las humanas. Los problemas que trata de resolver la visión por computadora incluyen: clasificación de imágenes, reconocimiento de rostros, reconocimiento de peatones, reconocimiento de categorías, reconocimiento de instancias, reconocimiento basado en características, reconocimiento basado en regiones, reconocimiento y segmentación si-

multáneos, reconocimiento de ubicación, comprensión de contexto y de escenas y reconstrucción 3D, entre otras [38, 39].

Existen diversos concursos en los que, con base en un conjunto de imágenes que contienen objetos de clases preestablecidas, se evalúa el desempeño de distintos algoritmos para la solución de problemas relacionados con el reconocimiento de objetos en imágenes, lo cual engloba:

- Clasificación de imágenes: Predecir la presencia de por lo menos un objeto de cierta clase. El resultado puede ser la predicción de una lista de objetos. En algunos casos se requiere un valor de confianza de cada predicción. Este problema se encuentra en los concursos PASCAL VOC [2], MS COCO [5] e ILSVRC [4].

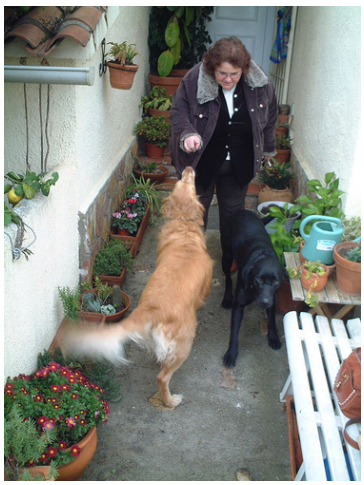
- Localización de objetos: Producir una lista de clases de objetos presentes en una imagen, acompañada de una caja delimitadora alineada a los ejes que indica la posición y escala de una instancia de cada categoría de objeto. Este problema únicamente es considerado en el ILSVRC de 2011 a 2014.

- Detección de objetos: Consiste en la localización de todas las instancias de cada clase de objeto presentes en una imagen. Es decir, predice las cajas delimitadoras de cada objeto presente en la imagen, de acuerdo a las clases tomadas en cuenta en el conjunto de entrenamiento. El resultado de la evaluación de una imagen es una lista de objetos con su clase y la caja delimitadora que lo contiene. Este problema se considera en MS COCO, ILSVRC, y en PASCAL VOC. Para este último también se requiere un valor de confianza de la predicción.

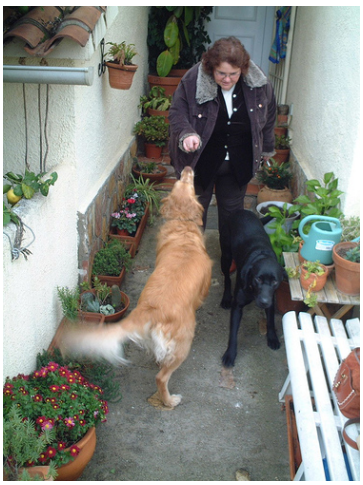
- Segmentación semántica: Consiste en predecir la clase de cada píxel de la imagen, ya sea un objeto o fondo. Las instancias de cada objeto no necesitan ser diferenciadas. Es una categoría de la competencia MS COCO y se introdujo a PASCAL VOC en 2007.

- Segmentación de instancias: Igual que el caso anterior, pero cada instancia de una clase debe tener una etiqueta diferente. Aunque ninguna competencia tiene una categoría de segmentación de instancias, para etiquetar las imágenes se entrenamiento y de prueba de la categoría de segmentación semántica se lleva a cabo la segmentación de todas las instancias de los objetos tomados en cuenta.

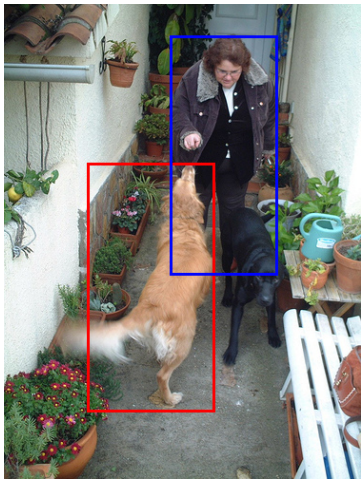
En la figura 2.5 se ejemplifica la diferencia entre los problemas de reconocimiento de objetos en imágenes. La imagen original se muestra en la figura 2.5a y en la figura 2.5b se da un posible resultado de un algoritmo de clasificación: la detección de la existencia de objetos de las clases “persona” y “perro”. La localización se puede observar en la figura 2.5c, donde solo se señala la existencia de solo un objeto de cada una de las clases válidas, a diferencia de la detección en la figura 2.5d, donde se señala una caja delimitadora



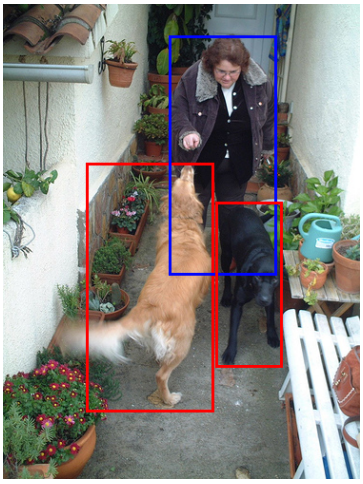
(a) Imagen original.



(b) Clasificación: Persona, Perro.



(c) Localización.



(d) Detección.

(e) Segmentación semántica.

(f) Segmentación de instancias.

Figura 2.5: Ejemplo de las tareas de reconocimiento de objetos en imágenes.

para todos los objetos de cada una de las clases. Finalmente, en las figuras 2.5e y 2.5f se muestran los resultados de la segmentación semántica y la segmentación de instancias, respectivamente. La diferencia entre las últimas recae en que en la segmentación semántica el fondo también recibe una etiqueta y que ambos perros tiene la misma etiqueta y en la segmentación de instancias el fondo no es tomado en cuenta y cada uno de los perros tiene una etiqueta distinta.

De los problemas anteriores, el que más se ajusta a los propósitos del presente trabajo, la detección de marcadores, es el problema de detección de objetos, pues se requiere determinar la localización de múltiples marcadores en una imagen. Esto se logra prediciendo cajas delimitadoras que contienen a cada uno de los objetos. Una caja delimitadora se puede caracterizar por a) las coordenadas de su esquina superior izquierda (x_0, y_0) y su esquina inferior derecha (x_1, y_1) o b) las coordenadas de su centro (x_c, y_c) y sus medidas de alto y ancho. Así, la detección de un objeto consistirá en la predicción de los valores que caracterizan a una caja delimitadora, acompañado de la predicción de la clase a la que pertenece dicho objeto, comúnmente codificado como una probabilidad por cada una de las clases presentes en el conjunto de entrenamiento. La clase que tenga la mayor probabilidad asociada se toma como la predicción final o, en otros casos, se toman en cuenta varias de las clases con mayor probabilidad [27, 28, 23, 11].

Existen arquitecturas de redes profundas para la detección de objetos que evalúan segmentos de la imagen con una arquitectura de clasificación de objetos y, si se obtiene un resultado que indique la presencia de un objeto de una clase dada, se genera una caja delimitadora correspondiente al segmento extraído y se le asigna la clase obtenida. Los segmentos pueden ser calculados mediante la fusión de ventanas que se deslizan sobre toda la imagen [11] o propuestos por algún algoritmo como la búsqueda selectiva [14]. Por otra parte, la detección se puede llevar a cabo como un problema de regresión de las cajas delimitadoras [26, 24].

En reconocimiento de patrones, una tarea común consiste en el aprendizaje de propiedades de un conjunto de datos para realizar predicciones en otro conjunto de datos del cual no se tiene conocimiento. Es decir, dado un conjunto de datos $\{x_1, x_2, \cdots, x_n\}$ del cual se conoce su categoría $y_i = y(x_i)$, se pretende aprender el mapeo y con el cual se puedan calcular las categorías de un conjunto de datos desconocidos. El aprendizaje de este mapeo se lleva a cabo mediante el entrenamiento de algún modelo de aprendizaje máquina. Cuando los datos de entrenamiento comprenden ejemplos de los cuales se conocen sus categorías objetivo se dice que éste es un problema de aprendizaje supervisado. Cuando se aspira a asignar a cada uno de los datos de entrada una categoría de un conjunto finito de categorías discretas, se dice que éste es un problema de clasificación (de donde viene el nombre de clasificación de imágenes). Si el resultado deseado consiste en una o más variables continuas, entonces se dice que tenemos un problema de regresión [40]. Así, el

problema de detección de objetos puede ser resuelto como un problema de regresión de los valores que caracterizan una caja delimitadora y las probabilidades de que el segmento de la imagen que está dentro de esa caja delimitadora pertenezca a una clase de entre las que se toman en cuenta en el problema de detección.

Otra técnica para la generación de cajas delimitadoras es el uso de cajas de anclaje. Las cajas de anclaje son cajas delimitadoras de referencia, es decir, son representantes de todas las cajas delimitadoras del conjunto de entrenamiento. Las cajas de anclaje pueden ser calculadas usando un algoritmo de agrupamiento. De esta manera, las cajas delimitadoras que corresponden a la detección de un objeto son calculadas como desplazamientos (*offset*) de los valores característicos de las cajas de anclaje. Esta técnica se caracteriza por lograr un entrenamiento más fácil, así como resultados con mayor precisión [27, 14].

2.4. Marcadores basados en tipo de orden

Los marcadores son objetos conocidos y de fácil detección por la computadora. En específico los marcadores de tipo de orden nos ayudan a resolver fácilmente el problema de la correspondencia de puntos entre el modelo del marcador y la imagen del modelo que está captando la cámara [32].

La correspondencia de puntos nos ayudaría a calcular la homografía entre el modelo y la imagen, y una vez obtenida la homografía se puede estimar la traslación y rotación que se debe aplicar al modelo para que se vea exactamente como se ve en la imagen del marcador.

En la imagen 2.6 se ven dos planos, uno es el modelo (a la derecha) y el otro la imagen (a la izquierda). Hace años usamos la esquina superior derecha del cuadrado en la imagen (la más cercana al origen de las coordenadas en la imagen [en línea punteada en la figura]), para empatar los puntos. Esta forma es sencilla pero no se pueden detectar rotaciones diferentes entre [0,90] grados.

La secuencia de problemas que se tienen que resolver para empatar los puntos es:

1. Detectar las características en la imagen (puntos, líneas, triángulos, etc.)
2. Se ajusta un modelo usando estadísticas robustas, RANSAC (o alguna otra heurística).
3. Encontrar la información en el modelo (si es un marcador).

Y así se tiene el empate de los puntos.

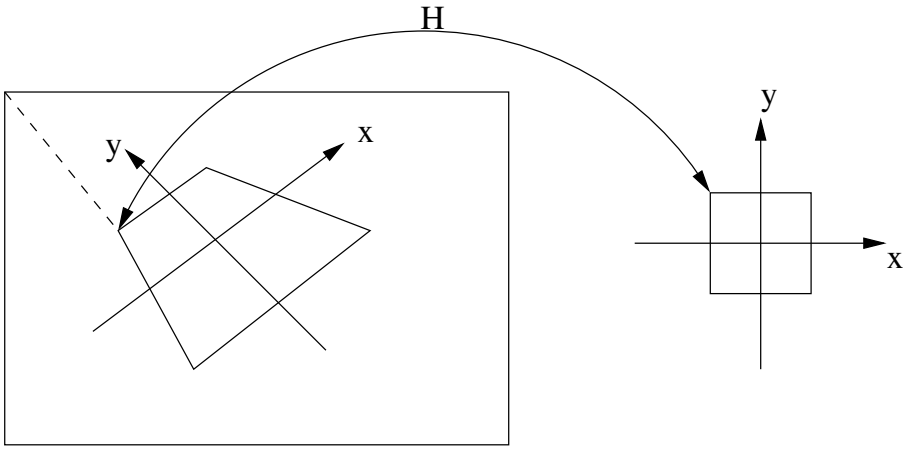


Figura 2.6: Un plano visto desde diferentes orientaciones. Ambas vistas están relacionas por la homografía H.

Se puede usar la característica que se llama *tipo de orden* para hallar el empate de dos conjuntos de puntos en el plano, aún si se han transformado según una traslación, rotación y proyección en perspectiva.

El tipo de orden es el conjuntos de orientaciones en el plano de tripletas de puntos.

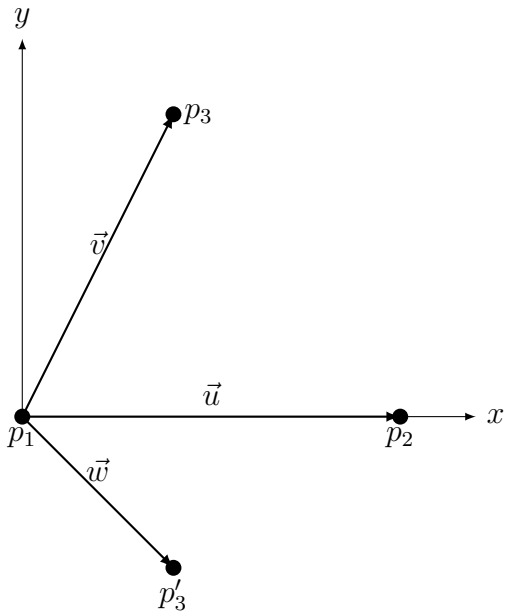


Figura 2.7: Orientación de tres puntos en el plano.

Si tenemos la secuencia p_1, p_2 y p_3 como se ve en la figura 2.7, los puntos están ordenados en el sentido contrario a las manecillas del reloj. El producto cruz tiene su componente Z

positiva:

$$((p_2 - p_1) \times (p_3 - p_1))_z = (\vec{u} \times \vec{v})_z > 0 \tag{2.6}$$

Para la secuencia de puntos p_1, p_2 y p'_3, donde p'_3 está abajo del eje de las 'x' (ver figura 2.7), la orientación de los puntos en el sentido de las manecillas de reloj y el producto cruz tiene su componente Z negativa:

$$((p_2 - p_1) \times (p'_3 - p_1))_z = (\vec{u} \times \vec{w})_z < 0 \tag{2.7}$$

Dados estos tres puntos: $p_1 = [0, 0]^T$, $p_2 = [5, 0]^T$, $p_3 = [0, 4]^T$.
Los vectores $\vec{u}$ y $\vec{v}$ tendrían los valores: $\vec{u} = [5, 0]^T$, $\vec{v} = [0, 4]^T$
Haciendo el producto cruz:

$$\begin{bmatrix} i & j & k \\ 5 & 0 & 0 \\ 0 & 4 & 0 \end{bmatrix} \begin{matrix} i & j \\ 5 & 0 \\ 0 & 4 \end{matrix} = i(0) + j(0) + k(20 - 0) = 20k \tag{2.8}$$

El producto cruz de los dos vectores $\vec{u}$ y $\vec{v}$ nos resulta en el doble del área del triángulo que forman los tres puntos. Para un triángulo, solo existe un tipo de orden como se muestra en la figura 2.8.

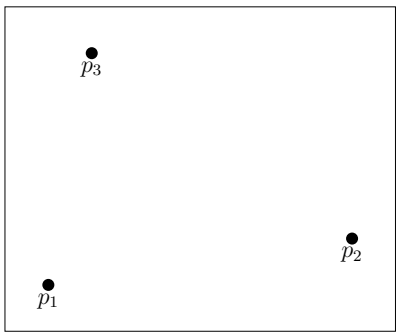


Figura 2.8: El único tipo de orden para tres puntos en el plano.

Cuatro puntos tienen dos tipos de orden como se muestra en la figura 2.9.

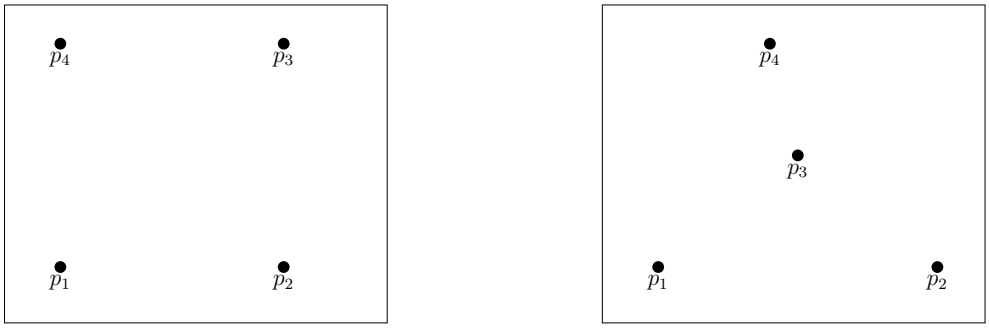


Figura 2.9: Los dos tipos de orden existentes para cuatro puntos en el plano.

La matriz lambda es una forma de representar el tipo de orden para un conjunto de puntos.

Una forma de orientar un conjunto de puntos es:

1. Calculamos los puntos que forman la cubierta convexa.
2. Cada punto de la cubierta convexa lo tomamos como el origen, y ordenamos todos los puntos en forma circular en el sentido contrario a las manecillas del reloj.
3. Calculamos la matriz lambda.
4. Tomamos de todas las las matrices lambda, la mínima lexicográfica.

Necesitamos conocer:

- Cómo encontrar la cubierta convexa (es un algoritmo de geometría computacional). Se podría utilizar el algoritmo de Graham [41].
- Un algoritmo para ordenar los puntos de forma circular (ver figura 2.10).
- Un algoritmo para calcular la matriz λ.

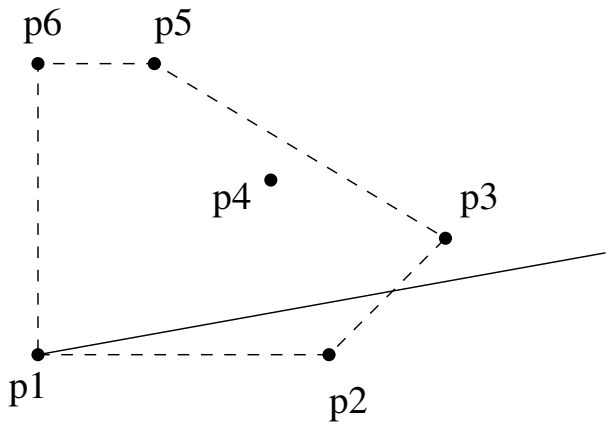


Figura 2.10: La línea punteada es la cubierta convexa. Los puntos están ordenados en forma circular alrededor del punto p_1.

Para calcular la matriz lambda podríamos usar el siguiente algoritmo:

- Tenemos la matriz de tamaño $n \times n$ para n puntos
- n columnas de los puntos $p_1, p_2, \cdots, p_n$
- n renglones de los puntos $p_1, p_2, \cdots, p_n$
- se traza una línea entre los puntos p_i y p_j $(i \neq j)$, y se cuentan los puntos que están a la izquierda de la línea.

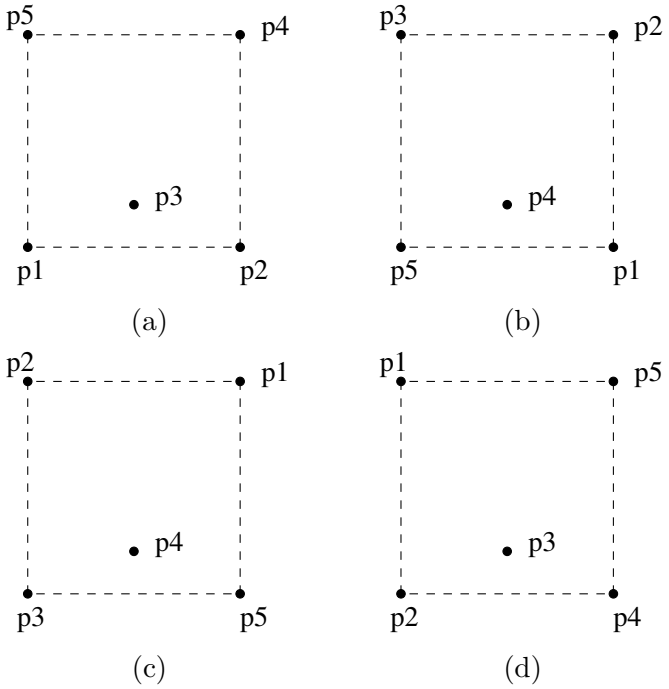


Figura 2.11: Las cuatro etiquetaciones posibles según la cubierta convexa de cinco puntos.

Para la figura 2.11, arriba a la izquierda, la matriz λ es:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} - & 3 & 2 & 1 & 0 \\ 0 & - & 1 & 3 & 2 \\ 1 & 2 & - & 2 & 1 \\ 2 & 0 & 1 & - & 3 \\ 3 & 1 & 2 & 0 & - \end{bmatrix} \end{array} \tag{2.9}$$

La relación entre $\lambda_{i,j}$, para $i < j$, y $\lambda_{j,i}$ está dada por la ecuación (2.10).

$$\lambda_{i,j} + \lambda_{j,i} = n - 2 \tag{2.10}$$

Las tres matrices λ para el resto de los puntos en la cubierta convexa en la figura 2.11 son:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} - & 3 & 2 & 1 & 0 \\ & - & 3 & 1 & 2 \\ & & - & 2 & 3 \\ & & & - & 1 \\ & & & & - \end{bmatrix} \end{array} \tag{2.11}$$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} - & 3 & 2 & 1 & 0 \\ & - & 3 & 1 & 2 \\ & & - & 2 & 3 \\ & & & - & 2 \\ & & & & - \end{bmatrix} \end{array} \tag{2.12}$$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} - & 3 & 2 & 1 & 0 \\ & - & 2 & 3 & 1 \\ & & - & 2 & 2 \\ & & & - & 3 \\ & & & & - \end{bmatrix} \end{array} \tag{2.13}$$

La matriz (2.11) corresponde a la matriz λ para la figura 2.11b, la matriz (2.11) corresponde a la figura 2.11c, y la matriz (2.13) corresponde a la figura 2.11d.

La matriz mínima lexicográfica sería λ_a:

- λ_a: 3 2 1 0 1 3 2 2 1 3
- λ_b: 3 2 1 0 3 1 2 2 3 1
- λ_c: 3 2 1 0 3 2 1 2 3 2
- λ_d: 3 2 1 0 2 3 1 2 2 3

Con los cinco puntos se puede hacer un marcador como el mostrado en la figura 2.12, con el cual se pueden detectar las cuatro esquinas del cuadrado negro y el vértice del triángulo más cercano al centro del cuadrado. Tenemos que detectar el cuadrado y el triángulo usando procesamiento de imágenes.

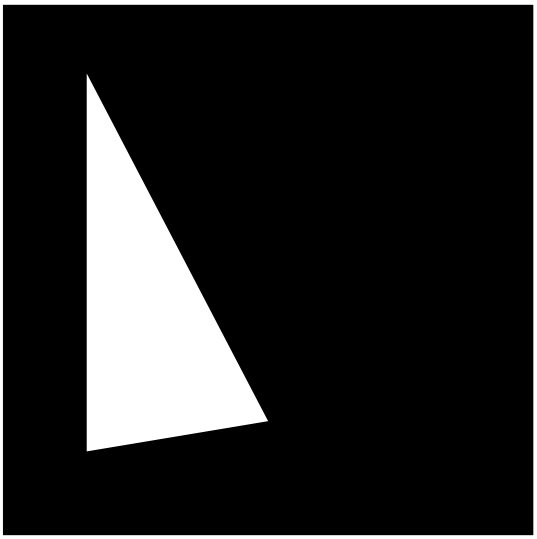


Figura 2.12: Marcador basado en tipo de orden.

Para el modelo del marcador de la figura 2.12, el sistema de coordenadas conviene que esté en el centro del marcador:

$$p1 = [-3.5, -3.5]^T, p2 = [3.5, -3.5]^T, p3 = [0.0, -2.0]^T, p4 = [3.5, 3.5]^T, p5 = [-3.5, 3.5]^T.$$

se pueden normalizar para no recalcularlos cada vez que se necesita calcular la homografía.

- En la imagen se detectan los triángulos
- Se calculan los valores de los vértices. Los valores de los vértices constituyen el conjunto de puntos que se quiere orientar.
- Se realiza el procedimiento explicado en esta sección.

Los puntos en la orientación de la matriz λ mínima lexicográfica corresponde a la misma etiquetación de arriba.

De esta forma, se resuelve el problema de emparejamiento de los puntos del modelo con los de la imagen.

2.5. Las redes profundas YOLO

Redmod *et al.* [26] propone la red neuronal profunda YOLO (You Only Look Once) para la detección de objetos en imágenes que es capaz de ejecutarse en tiempo real con

un desempeño comparable a métodos líderes. Formula la detección de objetos como un problema de regresión de cajas delimitadoras y probabilidades de clases asociadas. Una sola red neuronal predice cajas delimitadoras y probabilidades de clase directamente de imágenes completas en una evaluación. Es extremadamente rápida, procesa imágenes en tiempo real a 45 cuadros por segundo. Una versión más pequeña de la red, *Fast YOLO* procesa a 155 cuadros por segundo alcanzando el doble del rendimiento promedio que otros detectores de tiempo real. YOLO ve la imagen completa durante el entrenamiento y las pruebas, e implícitamente codifica información contextual, sobre clases así como de su apariencia.

Este sistema divide la imagen de entrada en $S \times S$ partes, cada una de las cuales se llama celda. La altura de cada celda es igual a la altura de la imagen dividida entre S y su ancho es igual al ancho de la imagen dividida entre S, por que las dimensiones de la imagen deben ser divisibles entre S. Al conjunto de todas las celdas se le conoce como malla. Si el centro de un objeto cae en una celda, esa celda es responsable de detectar ese objeto. Cada celda predice B cajas delimitadoras y puntajes de confianza para esas cajas. Esa confianza refleja qué tan seguro está el modelo de que esa caja contiene un objeto y también qué tan preciso se cree que la caja es lo que predice. Cada caja delimitadora consiste de cinco variables que deben predecirse: el centro de la caja (dos variables), el alto y ancho de la caja (otras dos variables) y el valor de la confianza de la predicción.

El centro de la caja se representa por un par de coordenadas (x, y), ambas con valores entre 0 y 1, donde $x = 0$ significa que se encuentra a la izquierda y $x = 1$ a la derecha; por su parte $y = 0$ significa arriba y $y = 1$ en el borde inferior de la celda. El ancho y la altura se predicen relativos a la imagen completa, no a la celda en la que se encuentra el centro del objeto. Dicho de otra manera, esto quiere decir que las predicciones de ancho y alto tienen valores entre 0 y 1, y representan una fracción de las dimensiones de la imagen que se está evaluando. Finalmente, la predicción de confianza representa la intersección sobre la unión (IsU) entre la caja predicha y cualquier caja verdadera de referencia (*ground truth*) de objetos del conjunto de entrenamiento. La confianza toma valores entre 0 y 1 y representa qué tan seguro está la red profunda de que el resultado que devuelve es correcto.

Cada caja también predice C probabilidades condicionales de clases $Pr(Clase_i \mid Objeto)$. Estas probabilidades son condicionadas en la celda que contiene un objeto. Solo se predice un conjunto de clases de probabilidad por celda, sin importar el valor para el número de cajas (B). Las predicciones se codifican como un tensor de $S \times S \times (5B + C)$.

El modelo YOLO se implementó como una red neuronal convolucional. Las capas convolucionales iniciales de la red extraen características de la imagen mientras que las capas totalmente conectadas predicen las probabilidades y las coordenadas de salida. Su arquitectura está inspirada en el modelo GoogLeNet [12] para la clasificación de imágenes.

La red YOLO tiene 24 capas convolucionales seguidas de 2 capas totalmente conectadas. También usa capas de reducción de 1×1 seguidas de capas convolucionales de 3×3. La versión rápida de YOLO, Fast YOLO, usa una red neuronal con 9 capas convolucionales en lugar de 24 y menos filtros en esas capas. Aparte del tamaño de la red, todos los parámetros de entrenamiento y pruebas son iguales en YOLO y en Fast YOLO.

Para su evaluación en el conjunto de datos PASCAL VOC 2012 [2], se usaron los parámetros $S = 7$, y $B = 2$, siendo la cantidad de clases $C = 20$, con lo que el tensor final de predicción tiene dimensiones de $7 \times 7 \times 30$. Obtuvo un mPP de 63.4 ejecutándose a una velocidad de 45 cuadros por segundo. Para esto, entrenaron las capas de extracción de la red en el conjunto ImageNet correspondiente a 1000 clases para la tarea de clasificación a una resolución de 224×224. Posteriormente, se agregan las capas finales para la detección, utilizando imágenes con una resolución de 448×448.

En [27] se presentan varias mejoras a la red YOLO, llamada YOLO v2, entre los que se encuentran:

- Normalización por lotes, es decir normalizar las respuestas de capas anteriores durante el entrenamiento, lo que permite mitigar problemas relacionados con la inicialización de los pesos y las diferencias en la escala de las activaciones. Permite un entrenamiento más rápido y permite una inicialización menos cuidadosa de los parámetros de la red.

- Preentrenamiento con ImageNet en una resolución más alta. Es decir que, a diferencia de la primera versión, ahora el preentrenamiento se realiza a la misma resolución que la detección, a 448×448 píxels.

- Capas convolucionales con cajas de anclaje, prediciendo variaciones con respecto de éstas en vez de coordenadas, lo que hace más fácil el entrenamiento.

- Características de grano fino: aumenta la cantidad de celdas de 7×7 a 13×13. Esto es ventajoso para la localización de objetos pequeños, además de que extrae características con una malla de 26×26 para concatenarlas con las de la malla principal.

- Entrenamiento multiescala: durante el entrenamiento se cambian aleatoriamente las dimensiones de las imágenes de entrada, con lo que la red pude hacer predicciones en resoluciones diferentes. Cada 10 épocas de entrenamiento escoge una resolución de entre $\{320, 352, \cdots, 608\}$ y continúa el entrenamiento. Esto fuerza a la red a aprender a predecir bien en una variedad de dimensiones de entrada, con lo que permite predecir a menores resoluciones y recibe una aceleración sin reducir significativamente la precisión.

- Cambio de extractor de características: pasó de usar una versión de GoogLeNet que requiere 8,520 millones de operaciones para una evaluación hacia adelante, a usar Darknet-19 la cual requiere 5,580 millones de operaciones para su evaluación y además mejora su precisión.

Darknet-19 es una arquitectura de red convolucional que, como su nombre lo indica, consta de 19 capas convolucionales y es implementada en el framework para redes profundas Darknet. Darknet [42, 43] ofrece programas para entrenar y ejecutar redes convolucionales profundas, inicialmente para acelerar la arquitectura YOLO en su versión 2, pero posteriormente incluyó soporte para múltiples extractores de características y enfoques para la detección de objetos [44]. Es de código abierto y está escrito en C y CUDA por lo que soporta cálculos en CPUs y GPUs.

Más recientemente, también se presenta YOLOv3 [28], que es una actualización a la red profunda YOLOv2. Utiliza nuevos métodos para la predicción de cajas delimitadoras, y para la predicción de clases en cada caja delimitadora, con lo que permite una detección multietiqueta. También implementa una predicción de cajas en 3 escalas diferentes, lo que le permite extraer características a esas escalas. Para la extracción de características usa una nueva red: un híbrido entre Darknet-19 usada en YOLOv2 y cosas novedosas de redes residuales, tiene capas convolucionales sucesivas de 3×3 y 1×1, cuenta con conexiones de acceso directo y ahora es más grande teniendo 53 capas. Obtiene mejores resultados que YOLOv2, pero no es más rápida. Usando el mismo hardware, YOLOv2 procesa 171 cuadros por segundo, mientras que YOLOv3 procesa 78 cuadros por segundo, es decir sigue siendo capaz de ejecutarse en tiempo real. En el sitio [45] se puede encontrar más información sobre la red YOLO y sus distintas versiones. Además, se pueden descargar los modelos entrenados para ejecutarse usando una cámara web, así como la arquitectura para entrenarla con datos propios.

Capítulo 3

Propuesta de solución

Es este capítulo se describirá cómo se utiliza una red profunda YOLO, el trabajo que se realizó para medir los tiempos de ejecución de las distintas versiones de redes YOLO, y finalmente el trabajo que se realizó con esta red para poder reconocer marcadores.

3.1. Uso de una red YOLO

Para la ejecución de una red profunda tipo YOLO se necesitan varios componentes:

- Descargar el código [43] y compilar el programa que se encargará de la ejecución, indicando en un archivo Makefile los componentes que se incluirán, tales como OpenCV, OpenMP o incluso la ejecución en GPUs y aplicar posteriormente el comando `make`.
- El archivo que describe la arquitectura de la red profunda es un archivo de texto con extensión *.cfg*.
- El archivo (binario) de los pesos, resultado del entrenamiento al que fue sometida la red profunda de acuerdo al problema de clasificación específico, así como archivos que describen el conjunto de datos, con extensión *.data* y *.names*.

El archivo de configuración contiene información como:

- El tamaño de las imágenes que procesará la red. Para mantener consistencia, todas las imágenes se escalan a este tamaño.
- El número de canales: 3 para imágenes a color y 1 para imágenes en escala de grises.
- Parámetros para el entrenamiento tales como: tamaño de lote y subdivisiones, momento, decaimiento, tasa de aprendizaje, máxima cantidad de lotes y número de pasos en los que se ajusta la tasa de aprendizaje.

- La declaración de las capas ocultas de la red, su tipo, la cantidad de filtros que contiene, el tamaño de los filtros y el tamaño de paso, si se agregan píxeles alrededor, y la función de activación.
- La declaración de la capa o capas de clasificación, las anclas que utiliza, las clases en las que clasifica, cómo se van a recortar las imágenes para el aumento de datos, entre otras.

Al ejecutar una red profunda se muestra un resumen de su arquitectura. Por ejemplo, al ejecutar la red Tiny Yolo v2 se muestra la información en la tabla 3.1.

Tabla 3.1: Resumen de la arquitectura Yolov2Tiny.

layer	filters	size/strd(dil)	input		output	
0 conv	16	$3 \times 3/1$	$416 \times 416 \times 3$	$\rightarrow$	$416 \times 416 \times 16$	0.150 BF
1 max		$2 \times 2/2$	$416 \times 416 \times 16$	$\rightarrow$	$208 \times 208 \times 16$	0.003 BF
2 conv	32	$3 \times 3/1$	$208 \times 208 \times 16$	$\rightarrow$	$208 \times 208 \times 32$	0.399 BF
3 max		$2 \times 2/2$	$208 \times 208 \times 32$	$\rightarrow$	$104 \times 104 \times 32$	0.001 BF
4 conv	64	$3 \times 3/1$	$104 \times 104 \times 32$	$\rightarrow$	$104 \times 104 \times 64$	0.399 BF
5 max		$2 \times 2/2$	$104 \times 104 \times 64$	$\rightarrow$	$52 \times 52 \times 64$	0.001 BF
6 conv	128	$3 \times 3/1$	$52 \times 52 \times 64$	$\rightarrow$	$52 \times 52 \times 128$	0.399 BF
7 max		$2 \times 2/2$	$52 \times 52 \times 128$	$\rightarrow$	$26 \times 26 \times 128$	0.000 BF
8 conv	256	$3 \times 3/1$	$26 \times 26 \times 128$	$\rightarrow$	$26 \times 26 \times 256$	0.399 BF
9 max		$2 \times 2/2$	$26 \times 26 \times 256$	$\rightarrow$	$13 \times 13 \times 256$	0.000 BF
10 conv	512	$3 \times 3/1$	$13 \times 13 \times 256$	$\rightarrow$	$13 \times 13 \times 512$	0.399 BF
11 max		$2 \times 2/1$	$13 \times 13 \times 512$	$\rightarrow$	$13 \times 13 \times 512$	0.000 BF
12 conv	1024	$3 \times 3/1$	$13 \times 13 \times 512$	$\rightarrow$	$13 \times 13 \times 1024$	1.595 BF
13 conv	1024	$3 \times 3/1$	$13 \times 13 \times 1024$	$\rightarrow$	$13 \times 13 \times 1024$	3.190 BF
14 conv	125	$1 \times 1/1$	$13 \times 13 \times 1024$	$\rightarrow$	$13 \times 13 \times 125$	0.043 BF
15 detection						

Se observa que cuenta de 16 capas, de las cuales 8 son convolucionales con filtros de tamaño 3×3 y una de tamaño 1×1, 6 son capas de reducción de valor máximo (max) y la restante es la capa que realiza la detección. En la segunda columna se observa la cantidad de filtros que tiene cada capa convolucional. Comúnmente hay pocos filtros en las primeras capas y éstos van aumentando. En la tercera columna se observa el tamaño de los filtros convolucionales así como el tamaño de paso. Para las capas de reducción de valor máximo, el tamaño siempre es 2×2 y el tamaño de paso (*strading*) es 2. Para las capas convolucionales, el paso siempre es 1 y el tamaño es de 3×3 en todas, excepto en la inmediata anterior a la capa de detección, la cual tiene tamaño de 1×1. En las siguientes columnas se observa el tamaño de los datos, tanto de entrada como de salida. En la última

columna se muestran los miles de millones (*billions*) de operaciones de punto flotante que se llevan a cabo al realizar una evaluación de cada capa.

3.2. Pruebas realizadas

En el artículo en el que se describe la primera versión de YOLO [26], se ofrece una liga [45] donde se muestran algunos resultados obtenidos con la red YOLO. Ahí se muestran algunos ejemplos del uso de algunas arquitecturas con pesos preentrenados con conjuntos de datos conocidos como Pascal VOC o COCO. También se describe cómo obtener la aplicación Darknet y cómo compilar el programa que se encarga del entrenamiento y la ejecución de redes profundas tipo YOLO. Se proporciona una liga para un proyecto en GitHub [43]. Sin embargo, su última actualización tiene fecha del 14 de septiembre de 2018. Por otra parte, existe una escisión del proyecto anterior [44] el cual aún sigue recibiendo actualizaciones y promete mejoras respecto al anterior; de aquí en adelante nos referiremos a esta versión del programa como Darknet 2.

Como primer paso se compararon los dos proyectos mencionados anteriormente. Se ejecutaron arquitecturas Yolov2, Yolov3 y sus versiones Tiny sobre imágenes de prueba proporcionadas en los sitios que contienen el código de las dos versiones de Darknet [43, 44] y otras más del sitio de Garima Nishad titulado *YOLO-Object-Detection* [46] donde se muestra el uso de Darknet en Jupyter Notebook [47] (Jupyter Notebook es una aplicación que permite ejecutar código de Python a través de un visor WEB).

En primer lugar se comparan los resultados de la ejecución de la arquitectura Yolo V3 con Darknet y Darknet 2, pues ésta promete los resultados más precisos. Los pesos usados fueron los proporcionados en la página del proyecto original [48], resultado del entrenamiento con el conjunto de datos COCO. Es decir, los pesos son el resultado de una red que detecta 80 clases de objetos. En la tabla 3.2 se muestran los tiempos de evaluación (en segundos) de las imágenes de prueba bajo la arquitectura Yolo V3 con ambos proyectos.

El equipo de cómputo usado para estas pruebas fue una computadora portátil modelo Lenovo Thinkpad T430 con procesador Intel Core i5 3320M con 2 núcleos físicos con tecnología Hyper-Threading con lo que tiene 4 hilos de procesamiento, tiene una velocidad de 2.6 GHz y velocidad Turbo Boost de 3.3 GHz, con memoria RAM DDR3 de 8 GB a 1600 MHz. El sistema operativo usado es Ubuntu 18.04 LTS de 64 bits. Las pruebas se realizaron usando solo un hilo de procesamiento.

En el último renglón de la tabla 3.2 se observa que los tiempos de ejecución de la red profunda por parte de cada proyecto evaluado tiene una desviación estándar pequeña, y en los demás renglones se observa que, en promedio, Darknet 2 es 2.45× más rápido que el proyecto original.

Tabla 3.2: Tiempos de ejecución en segundos de la arquitectura YoloV3 usando Darknet y Darknet 2.

Imagen	V3 Darknet	V3 Darknet 2
cat.jpg	26.417763	10.798221
city_scene.jpg	26.381366	10.702318
dog.jpg	26.286319	10.720821
dog2.jpg	26.351538	10.824617
Dog.png	26.199680	10.759497
eagle.jpg	26.871400	10.704488
food.jpg	26.525591	10.824194
giraffe.jpg	26.526086	10.835262
horses.jpg	26.494050	10.737496
kite.jpg	26.596260	10.788961
motorbike.jpg	26.557517	10.738273
person.jpg	26.495359	10.629286
scream.jpg	26.512240	10.729344
surf.jpg	26.411350	10.714395
wine.jpg	26.495513	10.872334
Promedio	26.474802	10.758634
Desviación std	0.153329	0.064298

En cuanto a detección de objetos, a continuación se muestran las imágenes donde hubo diferencias sobresalientes en las figuras 3.1 a 3.5. Para cada imagen, a la izquierda se muestra el resultado de Darknet y a la derecha el resultado de Darknet 2.

Los tiempos de ejecución, aún usando Darknet 2, son demasiado altos y alejados de la meta de 30 imágenes por segundo para su uso en sistemas de realidad aumentada. Por tal motivo, se hicieron pruebas con la versión 2 de la arquitectura YOLO en busca de mayor velocidad.

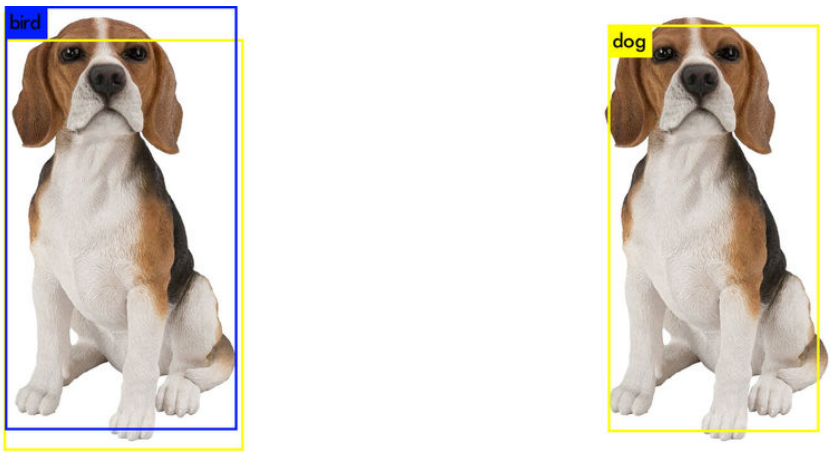


Figura 3.1: Dog.jpg

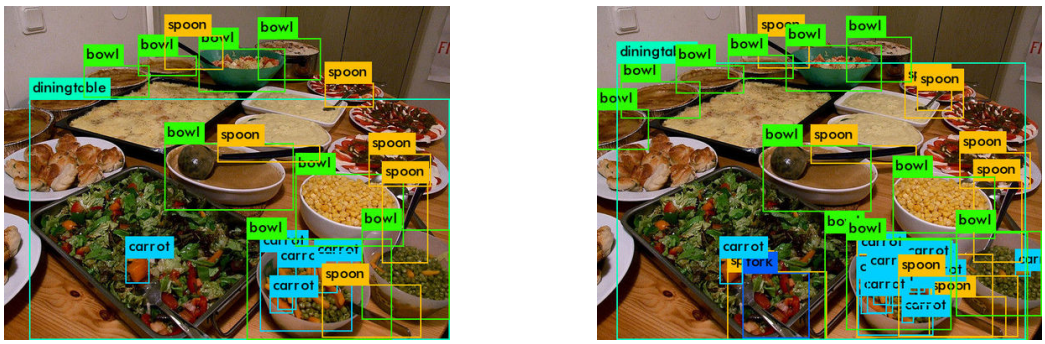


Figura 3.2: food.jpg

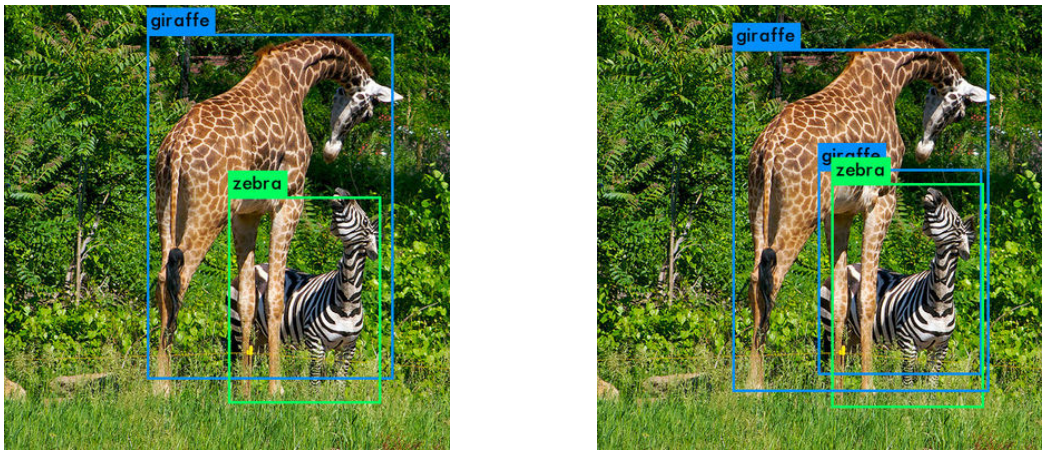


Figura 3.3: giraffe.jpg

Figura 3.4: surf.jpg

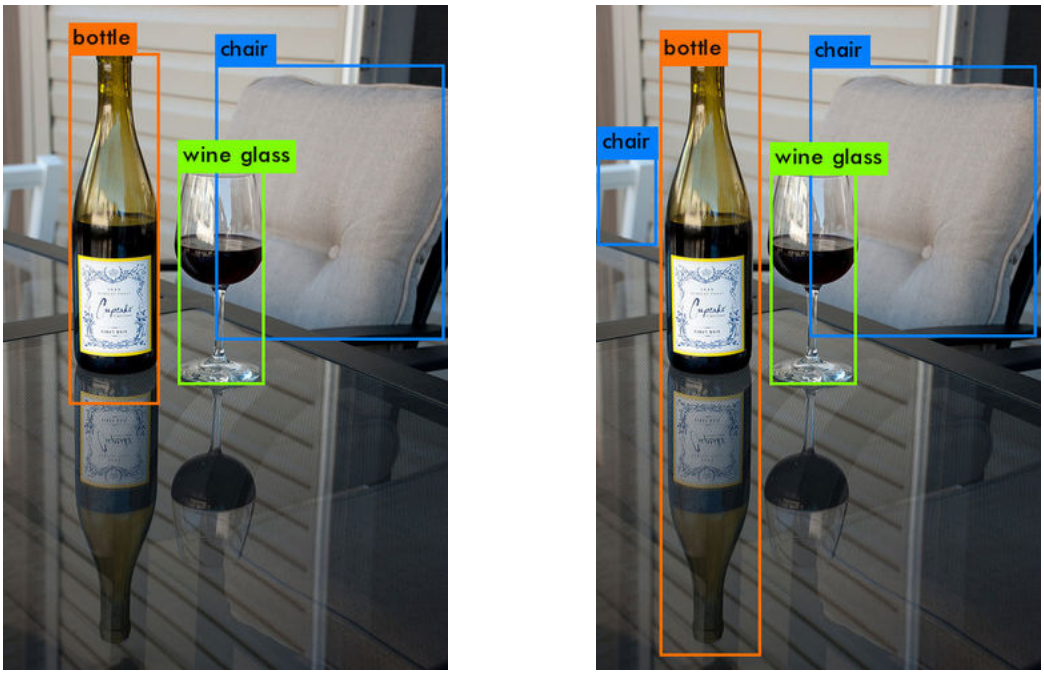


Figura 3.5: wine.jpg

En la tabla 3.3 se muestran los tiempos de evaluación (en segundos) de las imágenes de prueba bajo la arquitectura YOLO V2 con ambos proyectos. Se observa que el tiempo de ejecución es consistente en ambos casos, pues tienen una baja desviación estánda. Además, Darknet 2 muestra una aceleración de aproximadamente 2.3× con respecto al proyecto original.

Tabla 3.3: Tiempos de ejecución en segundos de la arquitectura YoloV2 usando Darknet y Darknet 2.

Imagen	V2	V2 Darknet 2
cat.jpg	12.705997	5.311514
city_scene.jpg	12.812199	5.290186
dog.jpg	12.893929	5.320891
dog2.jpg	12.878323	5.312965
Dog.png	12.368009	5.276196
eagle.jpg	12.436625	5.283967
food.jpg	12.362612	5.302146
giraffe.jpg	12.835892	5.278300
horses.jpg	12.301064	5.264640
kite.jpg	12.620270	5.294962
motorbike.jpg	12.656175	5.288004
person.jpg	12.475567	5.225981
scream.jpg	12.470502	5.258368
surf.jpg	12.314667	5.313592
wine.jpg	12.290696	5.317650
Media	12.561502	5.289291
STD	0.222432	0.026104

Por otra parte, los resultados en cuanto a detección fueron prácticamente idénticos excepto en las imágenes que se muestran en las figuras 3.6 a 3.11. Para cada figura, del lado izquierdo se muestra el resultado de Darknet y del lado derecho se muestra el resultado de Darknet 2. En éstas, se muestra que en general Darknet 2 produce más detecciones. En algunos casos, estas detecciones corresponden a aquellas que faltan en la evaluación de la red con la versión original de Darknet. Sin embargo, en otros casos se tienen detecciones erróneas. En otras palabras, aporta tanto detecciones correctas como positivos falsos.

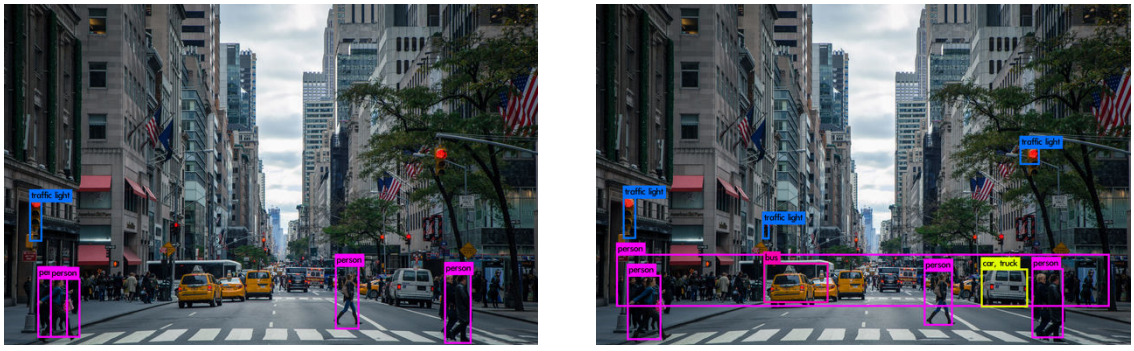


Figura 3.6: city_scene.jpg

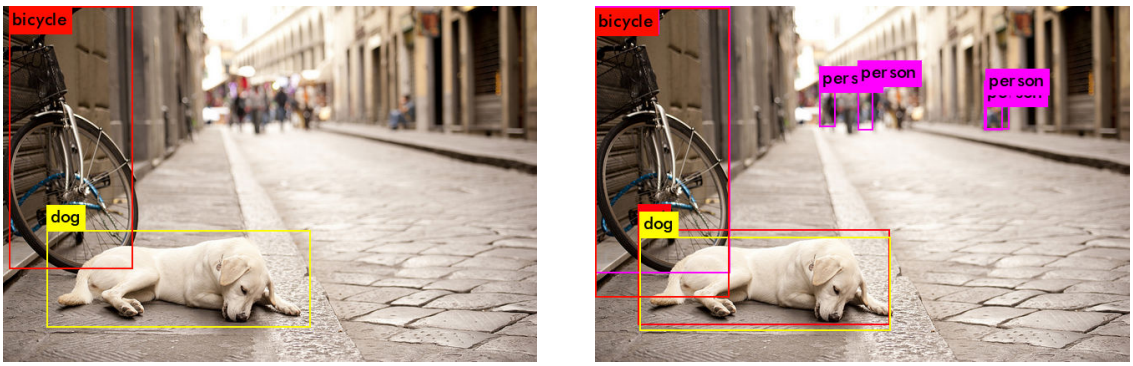


Figura 3.7: dog2.jpg

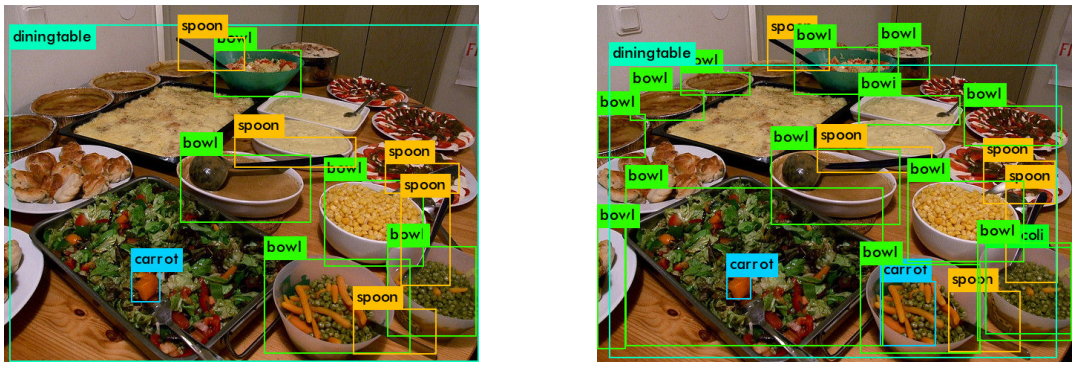


Figura 3.8: food.jpg

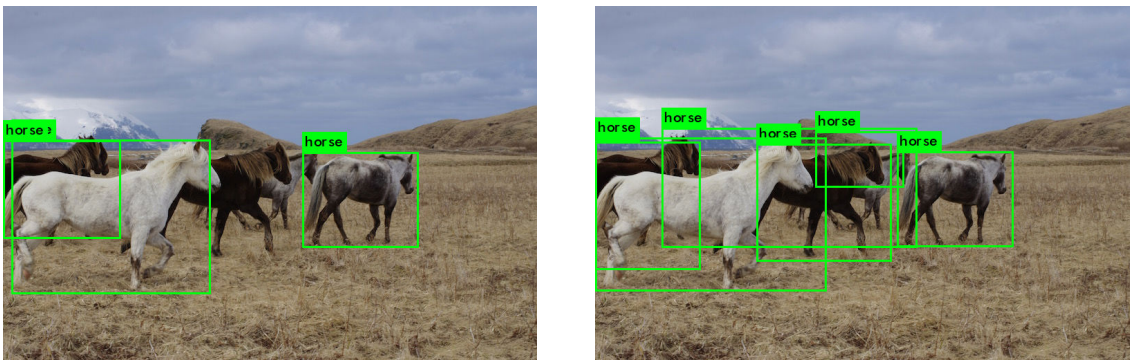


Figura 3.9: horses.jpg

Figura 3.10: kite.jpg

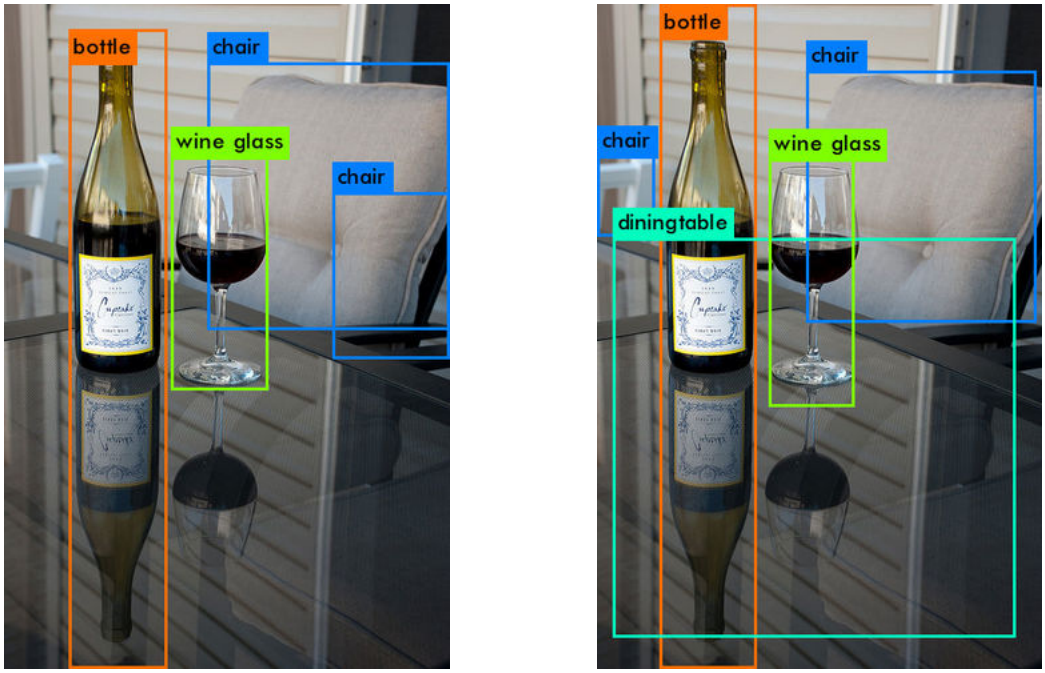


Figura 3.11: wine.jpg

Considerando la aceleración que se obtiene y la calidad de los resultados, se decidió el uso de Darknet 2 sobre el original. Sin embargo, los tiempos de ejecución de las arquitecturas YOLO V2 y YOLO V3 están muy alejados de la meta de 30 imágenes por segundo buscada para su utilización en sistemas de realidad aumentada. Por lo tanto, el siguiente paso fue medir el tiempo de ejecución de las versiones pequeñas de las arquitecturas, es decir Tiny YOLO V2 y Tiny YOLO V3, pero con Darknet 2.

En la tabla 3.4 se observan los tiempos de ejecución (en segundos) de cada imagen con las arquitecturas YOLO V2 Tiny y YOLO V3 Tiny. Al igual que en los casos anteriores, tienen una desviación estándar que indica que la ejecución de redes profundas tipo YOLO no depende de la imagen que se evalúa. Por otro lado, aunque el tiempo de ejecución

disminuyó considerablemente en comparación con las versiones completas de las arquitecturas usadas, aún no es suficiente para los objetivos anteriormente mencionados.

Tabla 3.4: Tiempo de ejecución en segundos de Yolo V2 Tiny y Yolo V3 Tiny medidos en Darknet 2.

Imagen	V2 Tiny	V3 Tiny
cat.jpg	0.955861	0.940443
city_scene.jpg	0.990452	0.961284
dog.jpg	0.975716	0.942548
dog2.jpg	0.951901	0.997000
Dog.png	0.960295	0.962537
eagle.jpg	0.959129	0.940351
food.jpg	0.976944	0.942518
giraffe.jpg	0.960785	0.944472
horses.jpg	0.960291	0.935134
kite.jpg	0.990114	0.939035
motorbike.jpg	0.969824	0.952094
person.jpg	0.956328	0.942096
scream.jpg	0.969951	0.937766
surf.jpg	0.962574	0.945324
wine.jpg	0.967018	0.947397
Media	0.967146	0.948667
STD	0.011807	0.015537

3.3. Detección de marcadores con redes tipo YOLO

Entre los trabajos basados en la red profunda YOLO se encuentra el de Huang, Pedoeem y Cuixian [49] en el cual se pretende diseñar una red profunda que se pueda ejecutar en dispositivos que carezcan de Unidades de Procesamiento Gráfico (GPUs) a una velocidad de 20 imágenes por segundo. Para tal propósito, proponen distintas arquitecturas producto de modificar la arquitectura YOLO V2 Tiny con lo que además exploran el efecto del uso de normalización por lotes en el entrenamiento.

En total, proponen 17 arquitecturas. La arquitectura con la que obtuvieron los mejores resultados obtiene una media de precisión promedio (mPP) de 33.57 en el conjunto de entrenamiento PASCAL VOC (20 clases) y una mPP de 12.26 en el conjunto COCO (80 clases) lo que nos sugiere que la cantidad de clases consideradas en un conjunto de entrenamiento es proporcional al tamaño de la arquitectura necesaria para resolverlo. En otras

palabras, que un problema en donde se requiera detectar una cantidad menor de clases podrá ser resuelto con arquitecturas más pequeñas. Dado que en este trabajo de tesis se pretende detectar marcadores y de acuerdo a lo observado, se sugiere que se obtendrán mejores resultados que los obtenidos en [49].

Para comprobar la veracidad de la afirmación anterior se realizaron pruebas con las 17 arquitecturas, reportadas en [50], con un conjunto de datos con una clase, correspondiente a un marcador. Siguiendo los pasos provistos en [51] se generó sintéticamente un conjunto de entrenamiento compuesto de 1292 imágenes con un fondo con valores en escala de grises aleatorio y un marcador. Para esto se utilizó PovRay [52], un software gratuito para la creación de escenas en tres dimensiones, usando su propio lenguaje de programación.

De acuerdo las instrucciones para la preparación del conjunto de datos propio para el entrenamiento de una arquitectura de red profunda, se llevó a cabo lo siguiente:

1. Crear el archivo `synthetic.names` en el directorio `build\darknet\x64\data\`, con los nombres de los objetos, uno en cada línea. Para nuestro propósito, solo fue una clase con el nombre de *marker*.

2. Crear el archivo `synthetic.data` en el directorio `build\darknet\x64\data\`, que contenga:

   ```
   classes= n
   train  = data/synthetic_train.txt
   valid  = data/synthetic_test.txt
   names = data/synthetic.names
   backup = backup/
   ```

3. Poner archivos de imágenes (.jpg) de tus objetos en el directorio `build\darknet\x64\data\synthetic\`

4. Se deben etiquetar cada objeto en las imágenes de tu conjunto de datos. Se deben tener archivos `.txt` para cada imagen `.jpg`, en el mismo directorio y con el mismo nombre, con número de objeto y las coordenadas de los objetos en la imagen, con cada objeto en una línea:

   ```
   <object-class> <x_center> <y_center> <width> <height>
   ```

   donde:

   - $< object-class >$ es un valor entero, es el número del objeto de 0 a $(clases-1)$,

- $< x\_center > < y\_center > < width > < height >$, son números reales de punto flotante correspondientes al ancho y al alto de la caja contenedora que rodea al objeto, están en el intervalo $(0.0, 1.0]$.

5. Crear el archivo train.txt en el directorio `build\darknet\x64\data\`, con los nombres de los archivos de las imágenes, cada nombre en una línea, con la dirección relativa al archivo ejecutable Darknet. Por ejemplo:

```
data/synthetic/img1.jpg
data/synthetic/img2.jpg
data/synthetic/img3.jpg
```

Se puede observar que el paso que requiere mayor cantidad de trabajo es el 4, pues se debe realizar para cada una de las imágenes. Sin embargo ya se contaba con los dos sistemas clásicos que realizan esta tarea [31], lo que facilita este proceso en mayor medida. Se implementó un programa que, utilizando las funciones del segundo sistema clásico, se ejecuta por línea de comandos, recibe como argumento una imagen y en caso de detectar un marcador, genera un archivo con el formato solicitado.

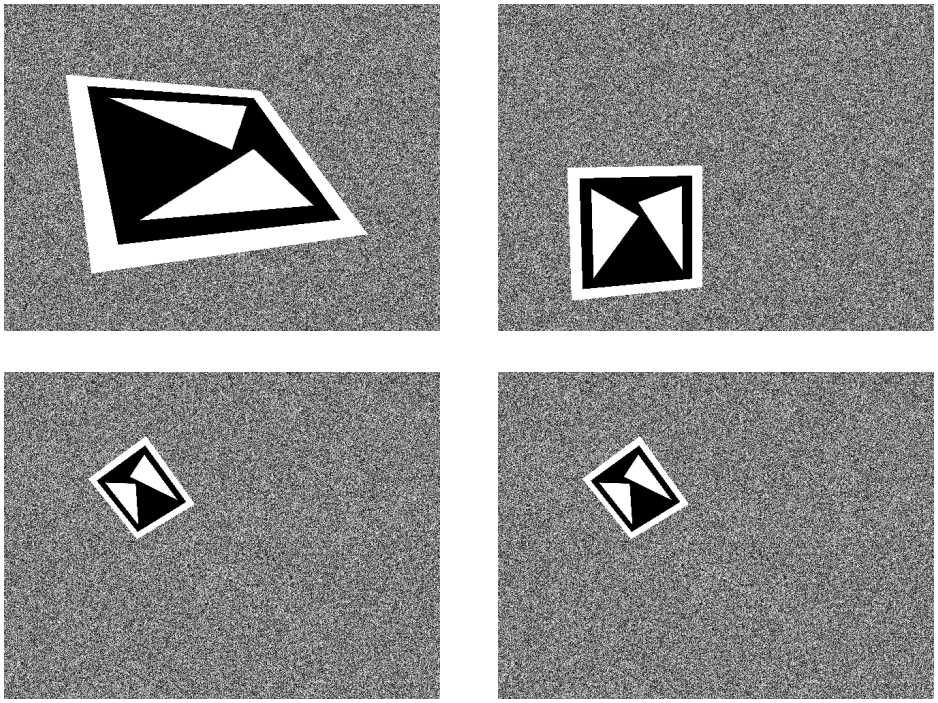


Figura 3.12: Ejemplos de imágenes sintéticas

En total, se generaron 2000 imágenes, con posiciones y rotaciones aleatorias en un rango determinado experimentalmente de acuerdo con las detecciones satisfactorias que

entregaba el segundo sistema clásico. Las detecciones estuvieron en un rango de 11.5 cm a 133 cm de distancia, a 11.5 cm de distancia el plano dentro del cual se puede ver que el marcador mide 13.6 cm de ancho y 10 cm de alto. A su vez, a 133 cm de distancia las medidas del plano donde se puede detectar el marcador son de 157 cm de ancho y 117 cm de alto. Las rotaciones están entre $-45°$ a $45°$ para los ejes X y Y y entre $-180°$ a $180°$ para el eje Z.

Para generar las imágenes se usó la función de PovRay que genera animaciones. PovRay le llama "animaciones" pero en realidad genera una secuencia de imágenes. Con este conjunto de imágenes se podría generar un video o un archivo GIF animado (con otro programa que no es parte de PovRay). El conjunto generado de imágenes tiene un cierto orden y no se generaron aleatoriamente, como enseguida se explicará. De acuerdo al número de imagen, se determina la distancia en el eje X, y posteriormente se calcula la posición de las coordenadas Y y Z de acuerdo al tamaño del plano en el que es visible el marcador a esa distancia, con lo que las imágenes se alejan de acuerdo a la posición que ocupen en la animación, siendo las primeras cercanas con un plano pequeño y las últimas lejanas con el plano más grande. El cambio de distancia se realizó linealmente basado en la posición en la animación, utilizando la variable *clock*, la cual avanza de 0 a 1 tomando valores decimales. Con esto se la distancia se calcula como:

$$\texttt{D(clock)} = \texttt{min} + \texttt{clock(max} - \texttt{min)} = \texttt{0.115} + \texttt{clock} \times \texttt{1.215} \tag{3.1}$$

De todas las imágenes generadas, el programa basado en las funciones del segundo sistema clásico realizó la detección del marcador en 1642 de ellas, las que no fueron detectadas fueron aquellas en las que el marcador estaba cercano a los bordes, con una rotación que impedía su detección, por lo que una de sus esquinas tocaba el borde, con lo que el sistema clásico la descalificaba automáticamente. Del total de imágenes etiquetadas, se tomaron 1292 para el conjunto de entrenamiento y 350 para el conjunto de pruebas.

Por otra parte, para cada uno de los archivos de configuración de las arquitecturas se debe hacer lo siguiente:

1. Se debe duplicar el archivo de configuración yolo.cfg y modificarlo para permitir el entrenamiento, de acuerdo a la nueva cantidad de clases.

   - batch=64,
   - subdivisions=8,
   - *max_batches*=clases×2000, pero no menos de 4000,
   - *steps* = 80 % y 90 % de *max_batches*,

- cambiar la línea classes=80 en las 3 capas [yolo],
- cambiar filters=255 a filters=(clases + 5)x3 en las 3 capas [convolutional] antes de cada capa [yolo],

2. Descargar los pesos preentrenados para las capas convolucionales, por ejemplo para la arquitectura Yolo V3, los pesos entrenados se consiguen en darknet53.conv.74, y es necesario colocarlos en el directorio `build\darknet\x64`.
3. Comenzar el entrenamiento usando el comando:

```
./darknet detector train data/synthetic.data yolo-synthetic.cfg darknet53.conv.74
```

4. Cuando se haya completado el entrenamiento obtener el archivo de pesos `yolo-synthetic_final.weights` del directorio `build\darknet\x64\backup\`, aunque esto se puede modificar en el archivo synthetic.data.

En el caso de las arquitecturas que se pretendían probar, no existían los pesos preentrenados, por lo que el entrenamiento se inicia con pesos aleatorios. Por otro lado, los pesos se guardan automáticamente a cada 1000 iteraciones de entrenamiento. Otra de las características que ofrece Darknet 2 sobre el original es que permite que durante el entrenamiento se realice un conjunto de pruebas para evaluar su evolución, con lo que es posible guardar un archivo con los mejores pesos vistos durante el entrenamiento, los cuales se encuentran en el archivo `synthetic_best.weights`. Con esto, resulta es más fácil determinar los pesos a utilizar y saber cómo se comporta el entrenamiento. Esto es porque, como se menciona en la página de internet de ambos proyectos, pueden ocurrir fenómenos de sobreajuste, lo que provoca que el entrenamiento se degrade en lugar de mejorar, mientras se realizan más iteraciones.

Para entrenar la primera arquitectura provista en [50], con nombre **tiny-yolov2-trial1.cfg**, después de realizar los cambios necesarios se debe ejecutar el comando:

```
./darknet detector train data/synthetic.data tiny-yolov2-trial1.cfg -map
```

En la computadora donde se realizaron las pruebas anteriores, la ejecución de dos iteraciones tomó 1760 segundos. Esto quiere decir que para llevar a cabo las 2000 iteraciones indicadas anteriormente tomaría alrededor de 1760000 segundos, lo que equivale a 20.37 días, lo cual no es manejable.

Por lo anterior, se probó entrenar la arquitectura de [49] con la que se obtuvieron mejores resultados. Es decir se utilizó **tiny-yolov2-trial3-noBatch.cfg**, ejecutando el comando:

```
./darknet detector train data/synthetic.data tiny-yolov2-trial3-noBatch.cfg -map
```

Para esta arquitectura se ejecutaron 103 iteraciones de entrenamiento en 1225 segundos. Por lo tanto, las 2000 iteraciones recomendadas tomarían un tiempo de 24768 segundos, lo que equivale a 6.6 horas.

Los resultados anteriores llevaron a intentar el entrenamiento mediante GPU, función también ofrecida en el proyecto utilizado. Para esto se usó otro equipo de cómputo que consta de una tarjeta madre Gigabyte GA-B75M-D3H, procesador Intel Core i7 3770 con 4 núcleos físicos, con tecnología Hyper-Threading con lo que tiene 8 hilos de procesamiento, tiene una velocidad de 3.4 GHz y velocidad Turbo Boost de 3.9 GHz, con memoria RAM DDR3 de 16 GB a 1600 MHz, y una tarjeta gráfica Nvidia RTX 2060 con 6 GB de memoria. El sistema operativo es Ubuntu 18.04 LTS de 64 bits. Fue necesario instalar diversos paquetes como CUDA y cuDNN en sus últimas versiones.

El entrenamiento mediante GPU de la arquitectura **tiny-yolov2-trial1.cfg** tomó 1515 segundos y para la arquitectura **tiny-yolov2-trial3-noBatch.cfg** tomó 169 segundos lo que equivale a una aceleración de 1161× y 146×, respectivamente. Posteriormente, todos los entrenamientos se hicieron utilizando el equipo de cómputo con GPUs y las pruebas de desempeño en la computadora portátil ThinkPad T430.

Los resultados obtenidos del entrenamiento de las 17 arquitecturas propuestas en [49] se muestran en la tabla 3.5. Las 9 arquitecturas restantes que no aparecen no convergieron o la detección de marcadores que reportan es cero, por lo que no fueron tomados en cuenta. Las arquitecturas 3, 6 y 7 son las que permiten una detección a mayor velocidad, según se muestra en la columna de cuadros por segundo (FPS), pero también ofrecen un nivel de desempeño alto de acuerdo con la columna de la media de la precisión promedio (mPP) y de intersección sobre la unión (IsU). Por este motivo, las pruebas posteriores se realizaron sobre estas tres arquitecturas.

Tabla 3.5: Resultados de las arquitecturas entrenadas con datos propios sintéticos. Los tiempos están medidos en segundos.

Arquitectura	mPP	IsU	Entrenamiento	Tiempo	FPS
T2	94.66	60.28	425.60	114.78	3.05
T3	99.32	71.88	53.93	27.68	12.64
T5	27.52	13.78	341.53	39.08	8.96
T6	99.69	72.07	144.00	27.65	12.66
T7	98.95	61.31	180.57	34.36	10.19
T12	88.45	50.92	154.28	46.14	7.59
T13	99.71	81.26	287.43	153.45	2.28

Aunque la detección de imágenes sintéticas con las arquitecturas anteriores tiene un alto grado de exactitud, al utilizarlas en la detección en imágenes captadas por una cámara web, los resultados no son favorables. La teoría principal fue que dado que el fondo estaba compuesto de píxeles aleatorios, las redes profundas no son capaces de diferenciar los marcadores de otras formas y texturas presentes en imágenes de escenas capturadas por una cámara en un ambiente real. Por tal motivo, el siguiente paso es entrenar las arquitecturas que entregaron los mejores resultados con un nuevo conjunto de imágenes que tome en cuenta los factores mencionados.

3.4. Reducción del número de capas

Para crear un conjunto de datos de entrenamiento que contenga imágenes de escenas reales con marcadores, resulta poco práctico capturar manualmente las fotografías además que sería difícil encontrar la variedad buscada. Por lo tanto se recurrió a utilizar de nuevo la herramienta PovRay, pero en este caso el fondo a utilizar debía consistir en imágenes de escenas reales. Aprovechando que existen conjuntos de imágenes libres para su descarga, principalmente de desafíos de problemas de visión, como clasificación de imágenes, detección de objetos, segmentación de objetos, entre otras.

Se usó PASCAL VOC 2007, que consta de 9963 imágenes donde aparecen objetos de las 20 clases: persona, pájaro, gato, vaca, perro, caballo, oveja, avión, bicicleta, bote, autobús, carro, motocicleta, tren, botella, silla, comedor, planta en maceta, sofá y tv/monitor. En la figura 3.13 se muestran dos ejemplos de las imágenes generadas.

Figura 3.13: Ejemplos de imágenes sintéticas usando el conjunto de imágenes PASCAL VOC.

El posicionamiento de marcadores en las imágenes usando PovRay se modificó ligeramente:

- El rango en el que aparecen las imágenes se disminuyó, en los ejes X y Y, con el propósito de evitar que los marcadores toquen los bordes y por lo tanto sean

descartadas al tratar de hacer la detección usando el programa basado en el segundo sistema clásico. Anteriormente, el centro del marcador se podía localizar a una distancia de la mitad de su ancho. Ahora, esa distancia se aumenta a la mitad de la dimensión de la diagonal; esto es porque en el caso anterior el marcador seguía tocando el borde cuando estaba rotado en el eje Z. Con esta modificación se logra disminuir la aparición de marcadores que toquen el borde, aunque no se evita por completo pues no se consideran las rotaciones en los otros ejes.

- La distancia de los marcadores a la cámara (eje Z), se modificó de tal manera que aparezcan más marcadores lejanos que cercanos. Esto se logró modificando la interpolación de la siguiente manera: $\texttt{D(clock)} = \texttt{min} + \sqrt{\texttt{clock}}(\texttt{max} - \texttt{min})$, así, como se puede observar en la figura 3.14, la distancia se incrementa rápidamente, con lo que hay menos imágenes cerca, y posteriormente disminuye el ritmo de crecimiento.

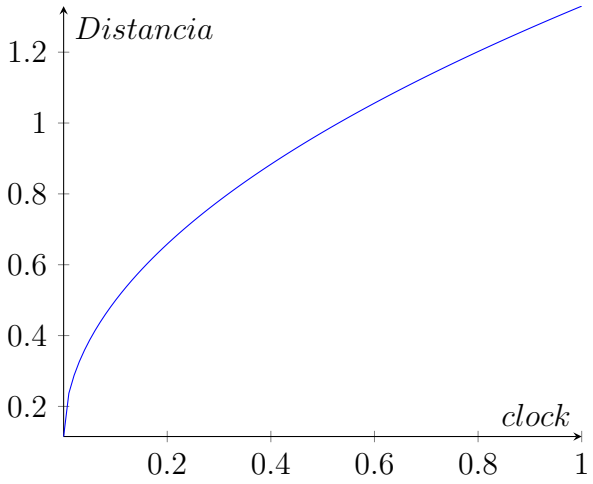


Figura 3.14: Gráfica de la distancia en función de *clock*

Con los cambios descritos se generó el conjunto de entrenamiento que consiste en las 9963 imágenes del conjunto de datos PASCAL VOC 2007 con marcadores agregados. Las imágenes generadas se etiquetaron usando el programa de detección del marcador basado en el segundo sistema clásico, el cual devolvió detecciones para 5553, y no tuvo ninguna detección para 4410 imágenes. De acuerdo a los datos de detección arrojados se analizaron los valores del ancho y alto de las cajas delimitadoras en búsqueda de valores atípicos, lo que en la mayoría de los casos se debe a una detección errónea. También se calculó la diferencia entre ambas medidas y el aspecto de los cajas, con lo que también se lograron identificar detecciones erróneas. Finalmente, se revisaron visualmente todas las detecciones, con lo que se encontraron algunas erróneas que no presentaban ninguno de los comportamientos anteriores. Se habían detectado rectángulos pero que no correspondían a marcadores, sino a objetos en las imágenes.

Después del proceso anterior se descartaron en total 671 imágenes, con lo que el total de imágenes etiquetadas correctamente se redujo a 4882, de las cuales 4382 se usaron para el conjunto de entrenamiento y las 500 restantes para el conjunto de pruebas. Se entrenaron las arquitecturas 3, 6 y 7 (de la tabla 3.5) con el nuevo conjunto de datos, introduciendo modificaciones de resolución, aspecto y colores así como en la cantidad de iteraciones de entrenamiento, la cual se aumentó a 8000. Los resultados se muestran en la tabla 3.6.

Tabla 3.6: Resultados del entrenamiento con el conjunto de datos basado en VOC.

Arquitectura	Medidas	Tiempo	mPP	IsU	FPS	mPP_d	IsU_d	FPS_a
VOC-Tiny3	224×224	44.81	99.55	77.96	11.16	0	0.00	1
VOC-Tiny6	224×224	44.04	99.57	77.29	11.35	0	0.00	1
VOC-Tiny7	224×224	57.66	99.34	75.66	8.67	0	0.00	1
VOC-Gray3	224×224	40.31	99.35	74.21	12.40	-0.20	-3.75	1.11
VOC-Gray6	224×224	39.99	99.34	76.39	12.50	-0.23	-0.90	1.10
VOC-Gray7	224×224	50.38	98.91	74.57	9.92	-0.43	-1.09	1.14
VOC-SqMedGray3	320×320	86.16	99.80	80.93	5.80	0.25	2.97	0.52
VOC-SqMedGray6	320×320	85.51	99.80	67.81	5.85	0.23	-9.48	0.52
VOC-SqMedGray7	320×320	108.90	99.78	75.91	4.59	0.44	0.25	0.53
VOC-MedGray3	320×240	63.69	99.36	65.82	7.85	-0.19	-12.14	0.70
VOC-MedGray6	320×240	62.13	90.18	61.61	8.05	-9.39	-15.68	0.71
VOC-MedGray7	320×240	78.33	92.84	61.23	6.38	-6.50	-14.43	0.74
VOC-BigGray3	640×480	281.89	99.71	73.54	1.77	0.16	-4.42	0.16
VOC-BigGray6	640×480	281.25	99.77	73.46	1.78	0.20	-3.83	0.16
VOC-BigGray7	640×480	367.79	99.17	68.65	1.36	-0.17	-7.01	0.16

Primero se muestra la versión original de las arquitecturas, es decir con tres canales de color y las dimensiones de imagen con las que fueron creadas y solo con las modificaciones necesarias para permitir el entrenamiento con una clase y con el conjunto de datos creado. Posteriormente, se encuentran las arquitecturas a las cuales se asignó 1 a la cantidad de canales, es decir, las imágenes se procesan en escala de grises. En este caso, se obtiene una aceleración de entre 10 % y 14 % respecto a las versiones originales. Esto con un costo de mPP no mayor de 0.5, y una pérdida en IsU de no más de 3.75. Al aumentar la resolución a 320×320, lo que cambia también el aspecto para igualar el de las imágenes utilizadas (4:3), el mPP aumenta hasta 99.80 en las arquitecturas 3 y 6 y 99.78 en la arquitectura 7. El IsU aumentó 2.97 hasta 80.93 en la arquitectura 3 y 0.25 para llegar a 75.91 en la 7, aunque sufrió una disminución muy grande de 9.48 para la arquitectura 6. La velocidad de procesamiento fue de entre 52 y 53 % con respecto a las versiones originales. Al aumentar la resolución a 640×480, se obtiene una ganancia de mPP en las arquitecturas 3 y 6, pero pérdida en la 7. Sin embargo, hay pérdidas de IsU en todas y la velocidad se reduce

hasta el 16 % de la de las versiones originales. En resumen, se observa una relación inversa entre la resolución de las imágenes y la velocidad de ejecución. Sin embargo, no es clara la relación con las medidas de desempeño.

Posteriormente, se realizaron pruebas modificando las arquitecturas en cuanto a las capas, eliminando las últimas. Adicionalmente, se variaron la resolución y el número de canales y se comparó la diferencia tanto en velocidad como en desempeño. Los resultados se muestran en la tabla 3.7. Al igual que en las pruebas anteriores, se muestran los resultados de las arquitecturas originales como referencia.

Las arquitecturas se nombraron por las características que fueron modificadas: la M seguida de un número se refiere a *modificación*: 1 indica que se eliminó la última capa y 2 que se eliminaron las 2 últimas. La B significa que se usó la resolución 640×480, y una G significa que se utilizó un canal de color. Todas comienzan con V para indicar que se utilizó el conjunto de datos basado en PASCAL VOC. Se observa que al eliminar la última capa se obtiene ganancia tanto en mPP como en IsU en M1T3 y M1T7, aunque en M1T6 hubo pérdida en ambas medidas. En los tres casos, el aumento en velocidad es de 7 a 8 %. Cuando se usa la resolución 640×480 en escala de grises, hay pérdida de las dos medidas en las tres arquitecturas. Sin embargo, contrario a lo anterior, la arquitectura que presenta menor pérdida es la M1BG6. La ganancia de desempeño en comparación a las arquitecturas similares pero con todas las capas (BigGray de la tabla 3.6) es de apenas 1 %. Cuando se eliminan las 2 últimas capas, conservando la resolución original de las arquitecturas, tienen una aceleración de 20 % a 22 %. Todas presentan pérdida de desempeño, pero para la arquitectura M2T3 es muy baja considerando la aceleración obtenida. Finalmente, en ausencia de las 2 últimas capas y usando la resolución 640×480 se pierde desempeño en las dos medidas con un aumento de velocidad solamente de entre 1 y 3 %.

La siguiente modificación probada consistió en reducir la cantidad de filtros de cada una de las capas convolucionales pero sin modificar su posición dentro de la arquitectura. En la tabla 3.8 se muestran los resultados obtenidos de las modificaciones descritas. Se ponen como referencia las arquitecturas con sus parámetros originales, y se compara la ganancia en velocidad y los cambios en las medidas de desempeño mPP e IsU. Con esta modificación se logra obtener tiempo real en ejecución de las redes sobre CPU.

Tabla 3.7: Resultados de modificación de número de capas.

Arquitectura	Dimensiones	mPP	IsU	FPS	mPP_d	IsU_d	FPS_a
V-Tiny3	224×224	99.55	77.96	11.16	0	0	1
V-Tiny6	224×224	99.57	77.29	11.35	0	0	1
V-Tiny7	224×224	99.34	75.66	8.67	0	0	1
V-M1T3	224×224	99.80	80.80	12.07	0.25	2.84	1.08
V-M1T6	224×224	97.54	68.68	12.23	-2.03	-8.61	1.08
V-M1T7	224×224	99.80	80.58	9.27	0.46	4.92	1.07
V-M1BG3	640×480	99.06	64.91	1.93	-0.49	-13.05	0.17
V-M1BG6	640×480	99.28	73.56	1.93	-0.29	-3.73	0.17
V-M1BG7	640×480	98.88	63.34	1.46	-0.46	-12.32	0.17
V-M2T3	224×224	99.33	77.58	13.34	-0.22	-0.38	1.20
V-M2T6	224×224	98.66	59.60	13.83	-0.91	-17.69	1.22
V-M2T7	224×224	98.40	69.62	10.44	-0.94	-6.04	1.20
V-M2BG3	640×480	94.21	57.05	2.14	-5.34	-20.91	0.19
V-M2BG6	640×480	92.90	53.79	2.09	-6.67	-23.50	0.18
V-M2BG7	640×480	88.56	53.33	1.73	-10.78	-22.33	0.20

Arq.	Final	F1	F2	F3	F4	F5	F6
V-Tiny3	$7 \times 7 \times 256$	16	32	64	128	128	256
V-Tiny6	$7 \times 7 \times 256$	16	32	64	128	128	256
V-Tiny7	$7 \times 7 \times 1024$	32	34	64	128	256	1024
V-M1T3	$14 \times 14 \times 128$	16	32	64	128	128	
V-M1T6	$7 \times 7 \times 128$	16	32	64	128	128	
V-M1T7	$14 \times 14 \times 256$	32	34	64	128	256	
V-M1BG3	$20 \times 15 \times 128$	16	32	64	128	128	
V-M1BG6	$40 \times 30 \times 128$	16	32	64	128	128	
V-M1BG7	$20 \times 15 \times 256$	32	34	64	128	256	
V-M2T3	$28 \times 28 \times 128$	16	32	64	128		
V-M2T6	$14 \times 14 \times 128$	16	32	64	128		
V-M2T7	$28 \times 28 \times 128$	32	34	64	128		
V-M2BG3	$40 \times 30 \times 128$	16	32	64	128		
V-M2BG6	$80 \times 60 \times 128$	16	32	64	128		
V-M2BG7	$40 \times 30 \times 128$	32	34	64	128		

Tabla 3.8: Resultado de reducir cantidad de filtros en capas convolucionales.

Arq.	Medidas	mPP	IsU	FPS	mPP_d	IsU_d	FPS_a	
V-Tiny3	224×224	99.55	77.96	11.16	0	0	1	
V-Tiny6	224×224	99.57	77.29	11.35	0	0	1	
V-Tiny7	224×224	99.34	75.66	8.67	0	0	1	
V-M3T3	224×224	99.63	73.65	34.97	0.08	-4.31	3.13	
V-M3T6	224×224	99.36	72.45	35.14	-0.21	-4.84	3.09	
V-M3T7	224×224	99.69	73.12	25.43	0.35	-2.54	2.93	
V-M4T3	224×224	97.56	70.42	83.89	-1.99	-7.54	7.52	
V-M4T6	224×224	96.51	58.37	81.57	-3.06	-18.92	7.18	
V-M4T7	224×224	97.81	66.89	60.46	-1.53	-8.77	6.97	
V-M4GT3	224×224	98.32	70.47	113.64	-1.23	-7.49	10.18	
V-M4GT6	224×224	97.54	65.51	108.23	-2.03	-11.78	9.53	
V-M4GT7	224×224	97.95	68.14	85.32	-1.39	-7.52	9.84	
V-M4GMT3	256×192	99.33	69.11	116.55	-0.22	-8.85	10.45	
V-M4GBT3	512×384	97.05	51.28	32.96	-2.50	-26.68	2.95	

Arq.	Finales	F1	F2	F3	F4	F5	F6	F7
V-Tiny3	$7 \times 7 \times 256$	16	32	64	128	128	256	
V-Tiny6	$7 \times 7 \times 256$	16	32	64	128	128	256	
V-Tiny7	$7 \times 7 \times 1024$	32	34	64	128	256	1024	
V-M3T3	$7 \times 7 \times 128$	8	16	32	64	64	128	
V-M3T6	$7 \times 7 \times 128$	8	16	32	64	64	128	
V-M3T7	$7 \times 7 \times 512$	16	17	32	64	128	512	
V-M4T3	$7 \times 7 \times 64$	4	6	16	32	32	64	
V-M4T6	$7 \times 7 \times 64$	4	8	16	32	32	64	
V-M4T7	$7 \times 7 \times 256$	8	7	16	32	64	256	
V-M4GT3	$7 \times 7 \times 64$	4	6	16	32	32	64	
V-M4GT6	$7 \times 7 \times 64$	4	8	16	32	32	64	
V-M4GT7	$7 \times 7 \times 256$	8	7	16	32	64	256	
V-M4GMT3	$8 \times 6 \times 64$	4	6	16	32	32	64	
V-M4GBT3	$8 \times 6 \times 128$	4	6	16	32	32	64	128

Capítulo 4

Conclusiones

Se realizó una investigación acerca de los distintos métodos de aprendizaje profundo que pudieran ser aplicados para el reconocimiento de marcadores fiduciales en tiempo real. Las redes neuronales convolucionales profundas resultaron ser el método más recomendado y usado para tareas de reconocimiento de patrones y en general, tareas relacionadas que involucran imágenes como datos de entrada. De entre los métodos de aprendizaje profundo con mayor desarrollo sobresalen las redes neuronales convolucionales profundas para la clasificación de objetos, localización de un solo objeto, detección de varios objetos, segmentación semántica y segmentación de instancias. Lo anterior, en virtud de que existen diversas competencias que evalúan el desempeño de algoritmos propuestos por grupos académicos de todo el mundo para la solución de las tareas mencionadas.

La tarea de detección de varios objetos es en la que se encuentran más semejanzas con el problema de reconocimiento de marcadores fiduciales, dado que se pretende conocer la ubicación de múltiples objetos en una imagen. Además, existen propuestas que pretenden ejecutarse en tiempo real.

Las arquitecturas de redes neuronales convolucionales profundas para la detección de varios objetos de la familia YOLO ofrecen un procesamiento en tiempo real con un grado de precisión competitivo. Además, se han mantenido en constante actualización los últimos años con respecto a mejoras en diseño y a facilidad de uso gracias al marco de desarrollo Darknet. Por lo anterior, las redes de la familia YOLO fueron seleccionadas para realizar pruebas con el propósito de la solución del problema planteado en esta tesis.

Se evaluaron distintas arquitecturas de la familia YOLO para la detección de objetos en tiempo real. Para esto fue necesario construir conjuntos de imágenes para entrenar dichas redes. Estos conjuntos de imágenes de entrenamiento consistían en imágenes generadas sintéticamente que contenían un marcador fiducial con posición y rotación aleatorias y que fueran completamente visibles en la imagen. Esto sobre un fondo en escala de grises

generado aleatoriamente píxel a píxel. Cada imagen fue etiquetada para la detección de objetos. Esto consiste en indicar cajas delimitadoras para cada marcador dentro de todas las imágenes. Es importante señalar aquí que ya se contaba con un sistema de detección de marcadores que usa técnicas clásicas de procesamiento de imágenes. Esto facilitó el trabajo, pues las cajas delimitadoras se encontraban de forma automática con este sistema clásico. El proceso de entrenamiento de las primeras redes de prueba, aunque eran las más ligeras disponibles, requerían una cantidad de tiempo significativa cuando se utilizaba una CPU para tal propósito.

Por lo tanto, se tomó la decisión de utilizar una GPU para el entrenamiento de estas redes, con lo que se obtuvo una aceleración sustancial, y el tiempo de entrenamiento resultó más manejable. Al finalizar las pruebas a las distintas arquitecturas entrenadas con el conjunto de entrenamiento construido, la versión YOLO-Lite era la que más se acercaba a un procesamiento que cumpliera las expectativas de velocidad utilizando solamente el CPU. Sin embargo, el desempeño observado en su ejecución sobre imágenes alimentadas en tiempo real por una cámara web no era satisfactorio, lo que se atribuyó a la aleatoriedad del fondo en el conjunto de entrenamiento.

Con el propósito de resolver el problema anterior se construyó otro conjunto de imágenes de entrenamiento en el que el fondo de las imágenes consistía en fotografías naturales, es decir contenían escenas cotidianas variadas, tanto de interiores como de exteriores. Con dicho conjunto se buscaba cubrir la mayor parte de las situaciones a las que se podría enfrentar un sistema de reconocimiento de marcadores fiduciales. Lo anterior fue posible gracias a la existencia de conjuntos de datos libres que son utilizados en las competencias relacionadas con tareas de reconocimiento de imágenes. Se observó una mejora en la detección de marcadores utilizando una cámara web, con respecto al conjunto de entrenamiento anterior.

Posteriormente, se realizaron varios experimentos modificando algunas arquitecturas de redes profundas tipo YOLO con el propósito de evaluar su balance entre precisión y velocidad de procesamiento. Los resultados indican una ganancia significativa en velocidad a un costo relativamente bajo en precisión al reducir sistemáticamente la cantidad de filtros convolucionales en cada una de las capas de las redes. Lo anterior resulta prometedor para su uso en sistemas con pocos recursos de cómputo.

Todos los programas usados en esta tesis se pueden encontrar en el repositorio [53]. Se incluyen:

- Programas de Darknet y Darknet 2, así como las imágenes de prueba presentadas en la sección 3.2. Se ofrecen archivos de procesamiento por lotes para su fácil ejecución.
- Programa de Python para generar imágenes con píxeles aleatorios en escala de grises.

- Programa de PovRay para incluir el marcador con posiciones aleatorias sobre los fondos aleatorios descritos en el punto anterior. En la figura 3.12 se muestran cuatro ejemplos de las imágenes generadas con estos programas.
- Programa del segundo sistema clásico para el reconocimiento de un marcador fiducial basado en tipo de orden. Éste utiliza la cámara web como dispositivo de entrada de imágenes en las que lleva a cabo el reconocimiento.
- Modificación al programa anterior, permite la detección del marcador en imágenes indicadas mediante línea de comandos y produce el archivo de detección de cajas contenedoras según el formato necesario para entrenar las redes tipo YOLO.
- Programa que, dado un archivo de imagen con su archivo de texto correspondiente a cajas contenedoras de objetos, muestra la imagen con las cajas contenedoras dibujadas. Es útil para comprobar las detecciones.
- Archivos necesarios para el entrenamiento con GPU de arquitecturas de redes profundas usando las imágenes sintéticas con fondo aleatorio. Se pueden encontrar archivos de procesamiento por lotes con los comandos necesarios tanto para el entrenamiento como la prueba de rendimiento de las redes con los pesos resultantes del entrenamiento.
- Programa de PovRay para generar las imágenes de entrenamiento que utilizan el conjunto de datos PASCAL VOC 2007.
- Archivos necesarios para el entrenamiento de diversas arquitecturas de red con el conjunto de datos mencionado en el punto anterior. Se ofrecen distintos archivos `.cfg` y los archivos de pesos (`.weigths`) de las arquitecturas con resultados más notables.

4.1. Trabajo futuro

A continuación se enumeran distintos puntos que se proponen para trabajos futuros relacionados con las redes profundas usadas en esta tesis:

- Incorporar el sistema realizado de redes profundas dentro de un sistema funcional de realidad aumentada.
- Medir si existen diferencias de desempeño al ejecutar las redes profundas en paralelo en computadoras con procesadores de varios núcleos.
- Incorporar rutinas que faciliten la inclusión de otros marcadores para su reconocimiento con redes convolucionales profundas. Es decir, para construir un conjunto de entrenamiento y a su vez entrenar las redes.

- Realizar pruebas sobre la cantidad máxima de marcadores distintos que pueden ser detectados y también su capacidad de diferenciarlos en distintas clases.
- Experimentación con las arquitecturas de redes profundas convolucionales para la detección de objetos ganadoras de las competencias del año en curso.
- Cambiar la aritmética que usan las redes a aritmética entera. Esto aceleraría 8 veces la ejecución de la red, pero se tiene que analizar sus efectos en el desempeño de la red.

Bibliografía

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[3] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[4] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[6] R.C. Gonzalez and R.E. Woods. *Digital Image Processing, 4th edition.* Pearson, 2018.

[7] Ondrej Chum and Andrew Zisserman. An exemplar model for learning object classes. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[8] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

[9] Christoph H Lampert, Matthew B Blaschko, and Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Xingyu Zeng, Wanli Ouyang, Junjie Yan, Hongsheng Li, Tong Xiao, Kun Wang, Yu Liu, Yucong Zhou, Bin Yang, Zhe Wang, et al. Crafting gbd-net for object detection. *IEEE transactions on pattern analysis and machine intelligence*, 40(9):2109–2123, 2017.

[17] Jiankang Deng. Página web: Bdat object detection. `http://image-net.org/challenges/talks_2017/ILSVRC_Detection.pptx`, 2017. Última fecha de acceso: 18 agosto, 2020.

[18] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[19] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6181–6189, 2018.

[20] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[21] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[24] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[25] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[26] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[27] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[28] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[29] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[30] Luis Gerardo de la Fraga. Notas el curso de visión, 2020. `https://delta.cs.cinvestav.mx/~fraga/Cursos/Vision/2020/`.

[31] Daybelis Jaramillo Olivares. Prototipo de un sistema de ojo de halcón. Master's thesis, Cinvestav, Departamento de Computación, agosto 2018. `https://www.cs.cinvestav.mx/tesisgraduados/2018/resumenDaybelisJaramillo.html`.

[32] Heriberto Cruz-Hernández and Luis Gerardo de la Fraga. A fiducial tag invariant to rotation, translation, and perspective transformations. *Pattern Recognition*, 81:213 – 223, 2018.

[33] Luis Gerardo de la Fraga and H. Cruz-Hernandez. Point set matching with order type. In *Mexican Conference on Pattern Recognition*, pages 229–237, 2018.

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

[35] Yann Le Cun, Lionel D Jackel, Brian Boser, John S Denker, Henry P Graf, Isabelle Guyon, Don Henderson, Richard E Howard, and William Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46, 1989.

[36] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, Department of Computer Science, April 2009.

[37] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[38] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[39] Simon JD Prince. *Computer vision: models, learning, and inference*. Cambridge University Press, 2012.

[40] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[41] Mark de Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry algorithms and applications*. Springer, 2000.

[42] Joseph Redmon. Página web: Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016. Última fecha de acceso: 18 agosto, 2020.

[43] Joseph Redmon. Repositorio github: Darknet. `https://github.com/pjreddie/darknet`. Última fecha de acceso: 18 agosto, 2020.

[44] Alexey Bochkovskiy. Repositorio github: Darknet. `https://github.com/AlexeyAB/darknet`. Última fecha de acceso: 18 agosto, 2020.

[45] Joseph Redmon. Página web: Yolo: Real-time object detection. `https://pjreddie.com/darknet/yolov1/`, 2016–2018. Última fecha de acceso: 18 agosto, 2020.

[46] Garima Nishad. Repositorio github: Yolo-object-detection. `https://github.com/Garima13a/YOLO-Object-Detection`. Última fecha de acceso: 18 agosto, 2020.

[47] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[48] Joseph Redmon. Página web: Yolo: Real-time object detection. `https://pjreddie.com/darknet/yolov2/`, 2016–2018. Última fecha de acceso: 18 agosto, 2020.

[49] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2503–2510. IEEE, 2018.

[50] Rachel Huang and Jonathan Pedoeem. Repositorio github: Yolo-lite. `https://github.com/reu2018DL/YOLO-LITE`. Última fecha de acceso: 18 agosto, 2020.

[51] Alexey Bochkovskiy. Repositorio github: Darknet, training. `https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects`. Última fecha de acceso: 18 agosto, 2020.

[52] Persistence of Vision Raytracer Pty. Ltd. Pov-ray - the persistence of vision raytracer. `http://www.povray.org/`. Última fecha de acceso: 18 agosto, 2020.

[53] Gonzalo Chávez. Repositorio gitlab: markersdeeplearning. `https://gitlab.com/GonzoCS/markersdeeplearning`. Última fecha de acceso: 15 septiembre, 2020.