Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Detección de marcadores en imágenes utilizando redes neuronales convolucionales

TESIS

Que presenta

Christian Ruiz Hernández

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. Luis Gerardo de la Fraga

Ciudad de México **Agosto, 2024**

Resumen

Actualmente, las redes neuronales convolucionales se han vuelto más populares con la introducción de bibliotecas que facilitan el uso de los modelos, así como su implementación y entrenamiento sobre unidades de procesamiento gráfico (GPU). Gracias a esto, estos modelos han adquirido una mayor complejidad y han sido aplicados en diferentes ámbitos en dentro del campo de la visión por computadora, como lo es la clasificación, detección y segmentación de imágenes. Es por eso que es importante estudiar formas de aprovechar el conocimiento de los modelos preentrenados que son útiles para adaptar una red a nuevos problemas y aplicaciones.
En este trabajo realiza una revisión de los modelos de redes neuronales convolucionales más comunes, (Single Shot Multibox Detector) y (You Only Look Once version 5), para la tarea de detección de objetos. Se realizan pruebas y entrenamientos de ambos modelos con un conjunto de datos personalizado con marcadores de tipo de orden. Este conjunto de datos se formó únicamente utilizando imágenes sintéticas que en conjunto con el modelo de la cámara oscura y de ruido tipo Perlin como fondo, facilita la creación de grandes cantidades de datos de entrenamiento y de prueba para entrenar los modelos de las dos redes profundas seleccionadas. De igual forma se utilizaron técnicas de transferencia de aprendizaje, comenzando por los modelos preentrenados de ambas arquitecturas con el conjunto de datos COCO. Se determinó que parte de estas redes se pueden congelar los pesos y el sesgo de diferentes capas para el reentrenamiento de los modelos, con el fin de ahorrar tiempos de entrenamiento aprovechando el conocimiento adquirido previamente y sin inicializar los pesos de las redes profundas con valores aleatorios. Fue posible entrenar la red YOLOv5 para detectar correctamente uno, dos y ocho marcadores diferentes.

Abstract

Currently, the convolutional neural networks have become increasingly popular with the introduction of computational libraries that facilitate the use of models, as well as their implementation and training on graphical processing units (GPU). Because of this, these models have increased their complexity and have been applied in different areas within the field of computer vision, such as image classification, detection and segmentation. That is why it is important to study ways to find ways to take advantage of the knowledge of the pre-trained models that are provided and adapt to new problems and applications.

In this work, a review is made of the most common convolutional neural network models Single (Shot Multibox Detector) and (You Only Look Once (YOLO) version 5), for the task of object detection. We test both models with a custom dataset with fiducial markers. This dataset was created using only synthetic images with the pinhole camera model and Perlin noise as background. This facilitates the creation of large amounts of training and test data for the Single Shot Multibox Detector and for the You Only Look Once version 5 models. Likewise, transfer learning techniques were used, starting by the use of the pre-trained models of both architectures with the COCO dataset. It was studied. Through several trainings, we were able to show that the weights and bias of different layers can be frozen for saving training time by taking advantage of the knowledge previously acquired instead of just initializing all deep network weights with random values. It was possible to train the YOLOv5 network to detect one, two and eight different fiducial markers successfully.

Agradecimientos

Al CONAHCYT por el apoyo económico brindado para dedicarme de tiempo completo a la culminación de la maestría. Al CINVESTAV por prestarme las instalaciones y facilitarme todos los recursos necesarios para concluir mi maestría. También a los Doctores y Doctoras que, con paciencia, fueron una guía indispensable para el cultivo de mi conocimiento. En especial a mi asesor el Dr. Luis Gerardo de la Fraga y sinodales que me ayudaron a superar los obstáculos que se presentaron durante la realización de este trabajo. A mi familia por siempre estar conmigo, en las buenas y en las malas, apoyándome y creyendo en mí. Mi mamá Mtra. María de los Angeles Hernández Maldonado, que me enseño el camino al mundo académico y me empujo a seguir en ese rumbo. A mí hermana Jessica Ruiz Hernández, que siempre estuvo motivándome a seguir adelante cuando más lo necesitaba. A mis amigos, que estuvieron conmigo ayudándome y acompañándome, dúrente este periodo de mi vida.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Definición del problema

Actualmente, para el problema de detectar objetos, es decir encerrar los objetos en un rectángulo dentro de la imagen donde aparecen, así como también clasificarlos o identificarlos, se utilizan diferentes técnicas de visión por computadora y aprendizaje automático. Más recientemente se están utilizando con más frecuencia técnicas de aprendizaje profundo es decir reden neuronales y redes convolucionales para dichos propósitos. De tal forma se aprovecha el poder de abstracción de las redes neuronales convolucionales, para extraer características de las imágenes, así como utilizando las redes profundas para la parte de clasificación.

Teniendo en cuenta esto, un gran problema al utilizar estos modelos de redes neuronales convolucionales es el gran tamaño y consumo de recursos, ya que las técnicas utilizadas en este tipo de redes para la detección hacen que la misma red sea mucho más robusta que las empleadas solo para clasificar. La idea de este trabajo es verificar si para un problema más sencillo, pueda de igual forma simplificarse la red, ya sea reduciendo capas, o fijando alguno de sus parámetros de algunas capas para mejorar el rendimiento, manteniendo la exactitud en la medida de lo posible. Para este fin, se considerarán problemas menos complejos, con marcadores más fácilmente distinguibles y menos objetos por identificar, a fin de reducir el número de clases.

Las redes convolucionales han adquirido mucha fama, por lo cual contamos con varias arquitecturas de clasificación como ResNet, EfficientNet, GoogleNet etc. Para el problema de detección de estos objetos, las más conocidas y que adoptan diferentes técnicas son la SSD(Single Shot Multibox Detector) [10] y Yolo(You Only Look Once) [11]. Esta es una de las principales razones por la que se propusieron estas dos arquitecturas para trabajar, debido a su amplia documentación así como la facilidad de acceso a los modelos ya pre-entrenados de ambas arquitecturas por parte de PyTorch. Cuenta con conjuntos de datos con varias clases y miles de imágenes como lo son ImageNet y Coco, pudiendo así utilizar estos y ser re-entrenados, aplicando posteriormente técnicas de transferencia de aprendizaje.

1.2. Objetivos generales y específicos

1.2.1. General

Detectar objetos en una imagen, encerrarlos en cajas y clasificarlos, utilizando redes neuronales convolucionales realizando mejoras a nivel rendimiento en tiempo tanto de entrenamiento como de ejecución, entrenándolas para encontrar o distinguir un solo tipo de objeto.

1.2.2. Particulares

1. Estudiar y comprender las arquitecturas actuales YoloV5 y SSD.
2. Obtener un conjunto de datos con marcadores para entrenar las redes neuronales.
3. Verificar el rendimiento de los modelos entrenados, así como su precisión.
4. Trabajar con Pytorch en GPUs, usando las arquitecturas YoloV5 y SSD.
5. Entender cómo trabajan las arquitecturas de redes neuronales en Pytorch así como el uso de los modelos que genera.
6. Utilizar las redes Yolov5 y SSD pre-entrenadas para aprovechar el conocimiento haciendo transferencia de conocimiento, cambiando la capa totalmente conectada.

1.3. Organización de la tesis

Esta tesis consta de cinco cuyo contenido es el siguiente:

1. El primer capítulo consta de los objetivos y el planteamiento del problema.
2. El segundo capítulo habla de todos los aspectos que son clave para comprender bien cómo trabajan en general las arquitecturas y de esta manera entender de igual forma el problema que se pretende solucionar.
3. El tercer capítulo abordará la propuesta de solución, así como todos los pasos y experimentos realizados.
4. En el cuarto capítulo analizaremos todos los resultados arrojados por cada uno de los experimentos realizados.
5. El quinto capítulo consta de las conclusiones y posibles líneas de trabajo futuro.

Capítulo 2

Marco Teórico

2.1. Problemas que pueden resolverse con redes convolucionales

En el mundo real existen decenas de problemas que pueden resolverse con redes neuronales convolucionales tales como: conducción autónoma, selección de cultivos, reconocimiento de caras, reconocimiento de expresiones faciales, reconocimiento de lenguaje de señas, etc. Sin embargo, en el campo del aprendizaje profundo son más específicos los problemas que se pueden resolver y a su vez más generales en aplicación, como por ejemplo:

- Reconocimiento de imágenes: Este es el problema de clasificación sobre imágenes que consiste en definir grados de pertenencia a una clase dada una imagen de entrada, identificándose a cuál de las clases pertenece.

- Detección de objetos: Este problema es el que se aborda en este trabajo que consiste en localizar objetos sobre una imagen y después clasificar dichos elementos localizados en alguna de las clases. Por lo tanto, la salida no solo nos mostrará los grados de pertenencia a cada una de las clases, sino también las coordenadas de localización del objeto, usualmente encerradas en una caja de cuatro lados. Para especificar la caja usualmente se utiliza el formato de las coordenadas del centro de la caja centro, más su ancho y su alto. Sin embargo, también existe la posibilidad de utilizar los dos puntos (el máximo y el mínimo de la caja) o los cuatro puntos de los vértices de la caja, pero estos dos últimos formatos son poco utilizados.

- Segmentación de imágenes: Este problema es muy parecido al anterior, pero ahora no solo son cajas lo que nos devolverá, sino que nos devolverá una región de píxeles de cualquier forma, ya que ahora no se utiliza una forma regular para encerrar el objeto. En este caso, son los mismos píxeles los que son marcados con una clase u otra. Para esto, usualmente en el entrenamiento se utilizan máscaras

que, dada la imagen de entrada, definen sobre los píxeles la pertenencia a las diferentes clases.

- Reconocimiento de texto: Las redes convolucionales son capaces de reconocer textos debido a que, son capaces de reconocer patrones de letras con diferentes escrituras al ser correctamente entrenadas.

- Procesamiento de video: Esto no se refiere más que al procesamiento de imágenes simultáneas, ya sea un video grabado o en tiempo real. Esta tarea la pueden realizar las redes CNN, ya que además de ser redes robustas, con el fortalecimiento del cómputo en paralelo por GPU pueden ser ejecutadas de forma continua, ya que el tiempo de ejecución (una vez paralelizadas) suelen ser en orden de los milisegundos.

- Generación de imágenes: Esto se refiere al problema de generar imágenes de cierto tipo de clase, es decir dado que las redes CNN son capaces de aprender los filtros necesarios para distinguir diferentes características de cada clase, los modelos preentrenados de redes neuronales convolucionales se aprovechan, para que, desde un conjunto de píxeles aleatorios se maximicen las características deseadas para la clase que se quiera generar y de esta forma se crea una imagen que el modelo considera corresponde a la clase para la que fue entrenada.

Como podemos ver, son varios los problemas que se pueden resolver en el campo de la visión por computadora, por medio del uso de redes neuronales convolucionales así que es un tema muy importante en la actualidad seguir estudiando estas redes para encontrar arquitecturas y técnicas que mejoren la eficiencia de las mismas.

2.2. Marcadores de tipo de orden

Los marcadores de tipo de orden son objetos cuyas características los hacen fácilmente reconocibles por algoritmos de visión por computadora. El uso más común de este tipo de marcadores es para ayudar a sistemas de visión a rastrear la posición y orientación de un objeto sobre un entorno tridimensional, a partir de datos bidimensionales, como lo son los datos que devuelven las imágenes de una cámara. Haciendo uso de patrones fácilmente reconocibles y, por lo regular, utilizando el color blanco y negro. Estos valores de los píxeles que representan la sobresaturación de los mismos (0 y 1, o 0 y 255), haciendo que el uso de estos colores facilite la detección de los patrones del marcador sobre las imágenes.

Actualmente, se hicieron famosos los códigos QR que son un tipo de marcador que se aprovecha del hecho de que son distinguibles los marcadores unos de otros, desarrollando una especie de etiquetado codificando la información sobre el mismo "marcador´´ usando de una matriz de puntos. También se utilizan diferentes tipos de marcadores en la actualidad para operaciones de realidad aumentada posicionando

elementos tridimensionales sobre los objetos bidimensionales (que también son marcadores) debido a las propiedades de los marcadores anteriormente descritas.

Los marcadores de tipo de orden están compuestos por tres elementos fundamentales, el área sin ruido, el área de datos y los puntos de la etiqueta, es decir, los puntos que forman la figura blanca representativa [12].

- Área sin ruido: Esto es el espacio en blanco que encierra todos los elementos del marcador y sirve como separador del marcador con respecto del resto de la imagen.
- Área de datos: Se compone por el área negra del marcador. Su propósito es contener un contraste con respecto a los puntos que ser requieren detectar.
- Puntos que definen la etiqueta: Son el conjunto de puntos que conformarán los vértices de la figura que contendrá el marcador.

Figura 2.1: Ejemplo de marcador de tipo de orden.

2.3. Redes neuronales convolucionales

Las redes neuronales convolucionales o redes CNN son un tipo de red del campo del aprendizaje profundo que aprovecha la operación de convolución como parte de la extracción de características. Usualmente, se utiliza en problemas de visión por computadora por su relación directa, con estos problemas.

En general, las redes CNN se componen de tres fases comunes:

- Convolución (extracción de características): En esta parte, los píxeles de la imagen de entrada se convierten poco a poco en mapas de características cada vez más abstractas y regularmente reduciendo la dimensionalidad espacial de la imagen, pero incrementando la profundidad que componen los mapas de características.

- Agrupación: En esta etapa se toman todos los mapas de características extraídas y se busca reducir el número de características, pero manteniendo siempre aquellos valores de las características que influirán más en el modelo. Usualmente, aquí se utilizan las operaciones llamadas de agrupación (pooling en inglés) para posteriormente pasar a una fase de aplanamiento convirtiendo todos estos valores en un solo vector unidimensional. Exista la agrupación del máximo, la agrupación del promedio y la agrupación del mínimo.

- Clasificación: En esta etapa se pasan los valores de la capa anterior aplanada a una red neuronal directa totalmente conectada, que se encargara de entrenar la parte de clasificación y su vez, mostrara los grados de pertenencia a cada una de las clases.

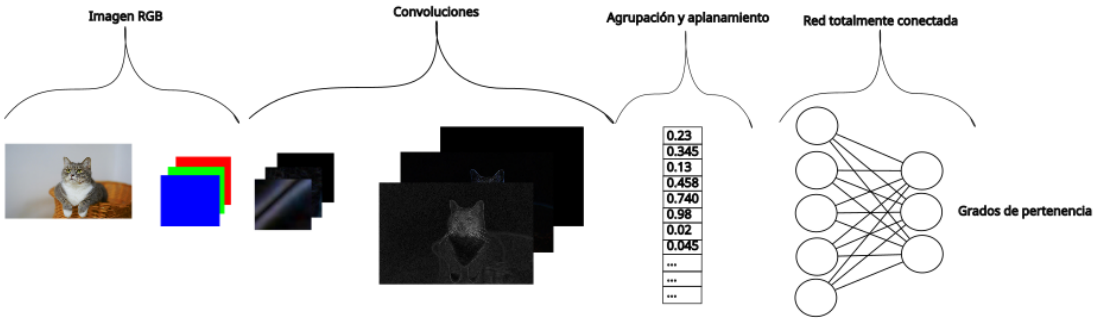


Figura 2.2: Ejemplo de red CNN.

2.3.1. Operaciones comunes utilizadas en redes convolucionales

Convolución

La convolución es la operación fundamental de una red CNN y es la primera de las operaciones que se efectúan sobre la imagen de entrada en redes CNN para clasificación y detección de imágenes. Esta es una operación en la que se involucran dos partes esenciales: 1) la imagen de entrada que es representada inicialmente como los valores de entrada de los píxeles para los diferentes canales y 2) el kernel o filtro que es una matriz de menor dimensión (regularmente de 3×3) en la cual el propósito de esta operación es lograr una extracción de características de la imagen. Cuando se entrena una red CNN los valores que se desea buscar, al igual que los pesos de la red totalmente conectada, son los valores del kernel o filtro que faciliten la distinción de las características de las diferentes clases de imágenes con las que está siendo entrenada.

En el ejemplo en la figura 2.3, se muestra la operación de convolución que consiste en multiplicar de manera directa cada uno de los valores de la imagen con los valores del filtro y después sumar cada uno de estos valores devolviendo un escalar que será el valor numérico que corresponderá al valor numérico del mapa de características en esa sección. En este ejemplo se muestran 2 cuadros, uno verde y uno azul. El verde corresponde al filtro en una posición inicial y el azul al filtro en una posición final una vez completadas las operaciones correspondientes. También para fines prácticos se tomó una imagen de 6×6 y valores aleatorios de la imagen del 0 al 9 así como un paso o Stride de uno. Es decir, que el filtro efectuará pasos de 1 pixel en 1 pixel. Sin embargo, estos valores pueden ser diferentes.

$1(1)+7(0)+8(-1)+6(1)+2(0)+8(-1)+2(1)+3(0)+9(-1) = 1-8+6-8+2-9$

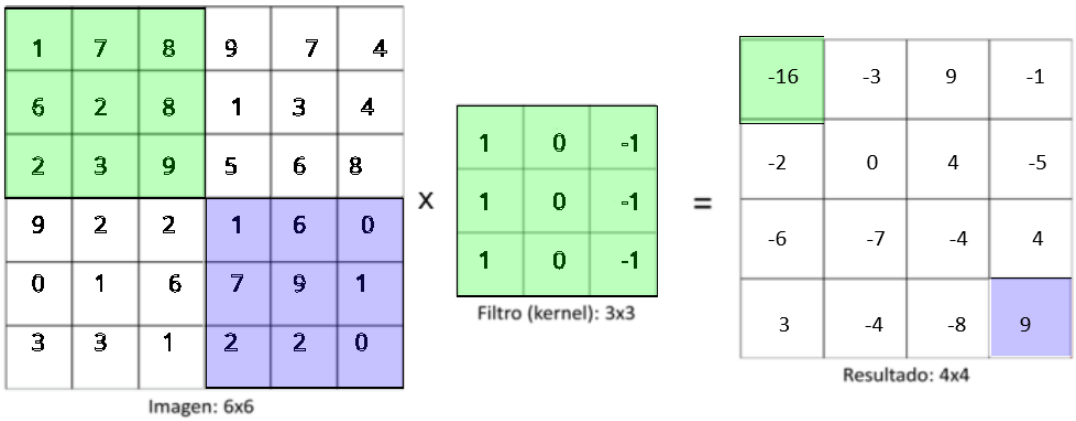


Figura 2.3: Explicación de operación de convolución.

Agrupación (pooling).

Dentro de las redes convolucionales como vimos anteriormente hay una fase de agrupación y aplanamiento. En este caso regularmente la fase de agrupación es una operación muy sencilla de implementar, ya que es parecida a la operación de convolución al necesitar de una ventana. Sin embargo, aquí no existe un filtro como tal. El tamaño de la ventana dado es aplicado sobre la misma matriz que entre mapas de características. Esto sirve para resaltar ciertas características sobre el mapa de características y a su vez reducir el tamaño de las mismas. De estas operaciones, las 3 más populares son las siguientes:

- Agrupación del valor mínimo: Esta operación consiste en tomar una ventana de $n \times n$ lados (usualmente de 2×2) sobre la matriz del mapa de características y sacar el valor mínimo que se encuentre dentro de dicha matriz. Este valor corresponderá a esa posición del mapa de características. $y = min(x)$
- Agrupación del valor máximo: Esta operación consiste en tomar una ventana de $n \times n$ lados (usualmente de 2×2) sobre la matriz del mapa de características y sacar el valor máximo que se encuentre dentro de dicha matriz. este valor será corresponderá a esa posición del mapa de características. $y = max(x)$
- Agrupación del valor promedio: Consiste de obtener la media entre los valores de la ventana y se colocará en la misma posición del mapa de características $y = \frac{1}{N} \sum_i x_i$

En la figura 2.4 se muestran las tres diferentes operaciones básicas de agrupamiento. Se tomó en cuenta un tamaño de filtro de 3×3 y un mapa de características inicial de 6×6 dándonos un mapa de características de salida de 2×2.

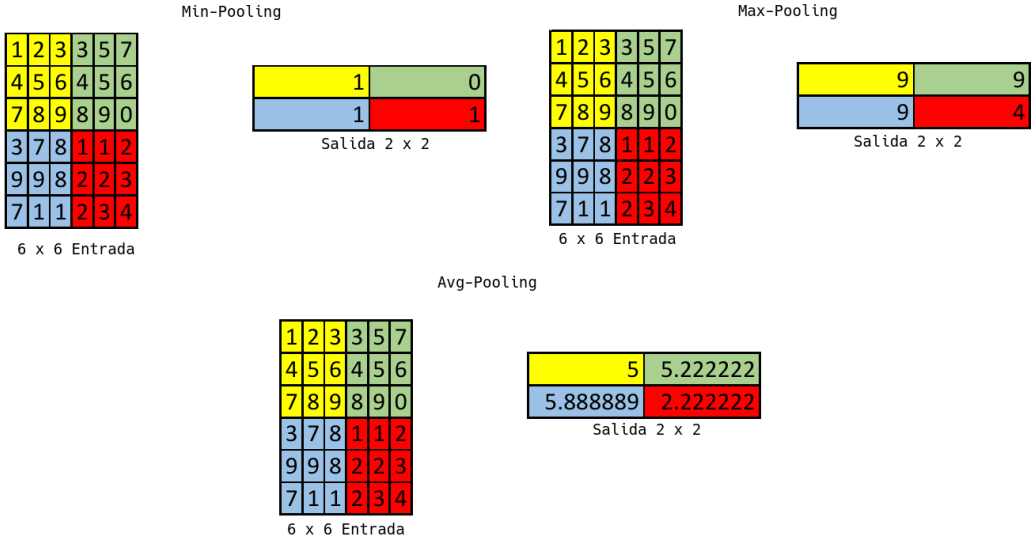


Figura 2.4: Ejemplos de pooling.

Relleno (padding).

Una operación que se integra entre las convoluciones y el pooling es la parte del relleno de los espacios en blanco cuando se pretende mantener el tamaño del mapa de características. Esto es llamado padding y usualmente consiste en rellenar los espacios en blanco con ceros, pero puede haber diferentes tipos de padding.

Tomando el ejemplo 2.4 de Max-pooling, se crea el ejemplo 2.5 suponiendo que necesitamos un tamaño de 4×4 a la salida del proceso, En este caso, hacemos un padding de 1 tomando en cuenta que será el incremento a realizar a las orillas.

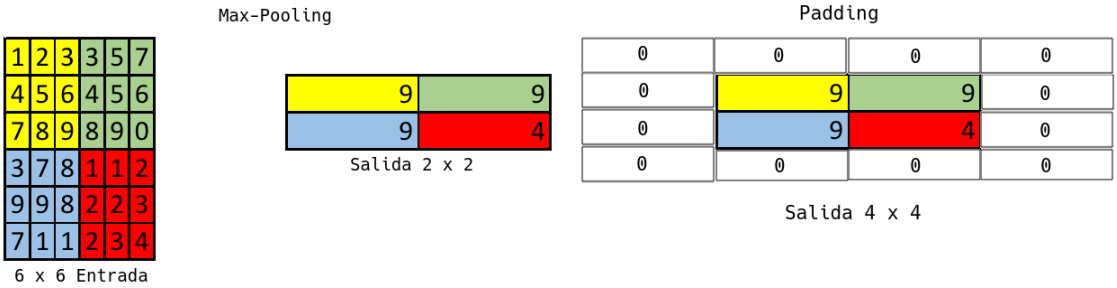


Figura 2.5: Ejemplo de relleno de valor 1 después de aplicar el agrupamiento del valor máximo.

Aplanamiento (flatten)

Por último una de las operaciones que se realizan sobre el mapa de características antes de entrar a la red totalmente conectada o red neuronal profunda clásica para clasificación, es la fase de aplanamiento o flatten. Ésta consiste en convertir todo el conjunto de mapas de características en un solo vector de una dimensión.

En la figura 2.6 podemos observar la operación flatten (aplanamiento) en un mapa de características de 3×3 en el que se resaltan con colores las respectivas posiciones de los números en vector de salida de una sola dimensión.

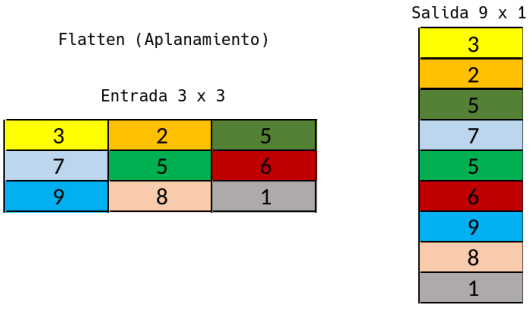


Figura 2.6: Ejemplo de flatten (aplanamiento).

2.3.2. Clasificación y detección

Como vimos en el primer capítulo, las redes CNN nos sirven para resolver varios de los problemas de visión por computadora. Dos de estos problemas son la clasificación y la detección. En esta sección abordaremos las diferencias entre estos dos problemas, así como la forma en la que una red CNN es capaz de resolverlos. Veremos también de forma genérica la estructura y los datos que devuelven las redes neuronales totalmente conectadas.

Clasificación

Este problema consiste en determinar el tipo de objeto al que corresponde cada uno de los objetos analizados dentro del mismo conjunto de datos. En este caso, como abordamos el problema desde una perspectiva de visión por computadora, el reto es determinar qué tipo de objeto aparece en una imagen o conjunto de píxeles.

Las redes CNN que resuelven este tipo de problemas, de manera general en el entrenamiento, tienen como entrada la imagen o los píxeles que la conforman y un vector que determina el grado de pertenencia a las diferentes clases. Éste usualmente está codificado con valores del 0 y 1, ya que la misma salida a la hora de la evaluación del modelo nos regresará un vector parecido, pero ahora con los diferentes grados de pertenencia del 0 al 1.

En la figura 2.7 se muestra: la fase de entrenamiento con un conjunto de datos que contiene tres clases diferentes (Gato, Perro, Auto). Podemos ver las imágenes de entrada de la red CNN así como el vector en el cual se codifica las clases, quedando de la siguiente forma gato [1,0,0], perro [0,1,0] y auto [0,0,1]. De igual forma, en la misma figura 2.7 se representa la parte del uso del modelo ya entrenado la cual nos devuelve la probabilidad de pertenencia a cada clase.

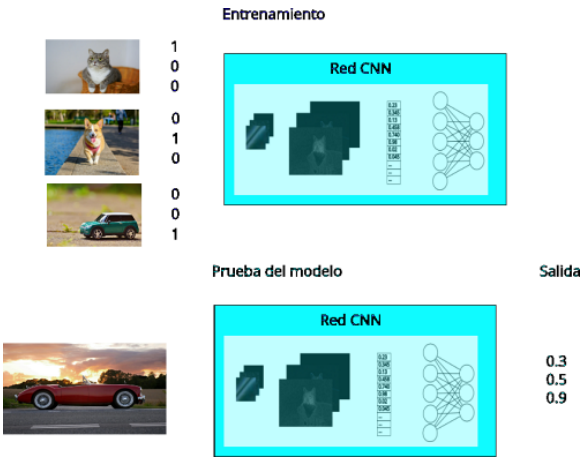


Figura 2.7: Ejemplo de una red CNN en un problema de clasificación.

Detección

En este punto el siguiente paso es la detección, es decir, ahora no basta solo con identificar el objeto, es decir clasificarlo, sino que ahora es necesario ubicarlo espacialmente sobre la imagen y encerrarlo para delimitar el objeto dentro de un cuadrado o rectángulo según sea el caso. Esto nos permitirá ahora clasificar más de un solo objeto en una misma imagen.

Para esto, es necesario agregar algunos cambios a los datos de entrada y la salida será, de igual forma, algo diferente. El primer cambio es que ahora, aparte de codificar la clase a la que pertenece, se tiene que codificar también su ubicación y para esto se utilizan por lo regular tres formatos:

- Centro, ancho y alto: Este es el formato que utiliza la red YOLO y es el más comúnmente utilizado debido a la popularidad de la misma red. Este formato se compone de las coordenadas del centro del objeto, el alto y el ancho quedando de la siguiente manera: [200, 200, 100, 100] suponiendo que el centro se ubica en las coordenadas $x = 200, y = 200$ y el ancho y el alto son de 100×100 píxeles.
- Mínimo y máximo: Este formato fue inicialmente utilizado por las primeras redes de detección FastRCNN o la misma SSD en sus primeras versiones. Se compone de un punto máximo y uno mínimo y de esta forma se dibuja el cuadro que encerrara el objeto. En la figura 2.9 podemos observar el objeto encerrado con los puntos máximos y mínimos en las siguientes coordenadas: mínimo en $min(100, 60)$ y máximo en $max(520, 910)$ (recuerde que en una imagen el origen de coordenadas está en el vértice superior izquierdo).
- Cuatro puntos: Por último. Este es el formato menos utilizado, ya que se compone directamente de los 4 puntos que encierran el objeto. Esto es mucho más ineficiente, por lo tanto, ninguna de las redes CNN para detección utiliza este formato en la actualidad.

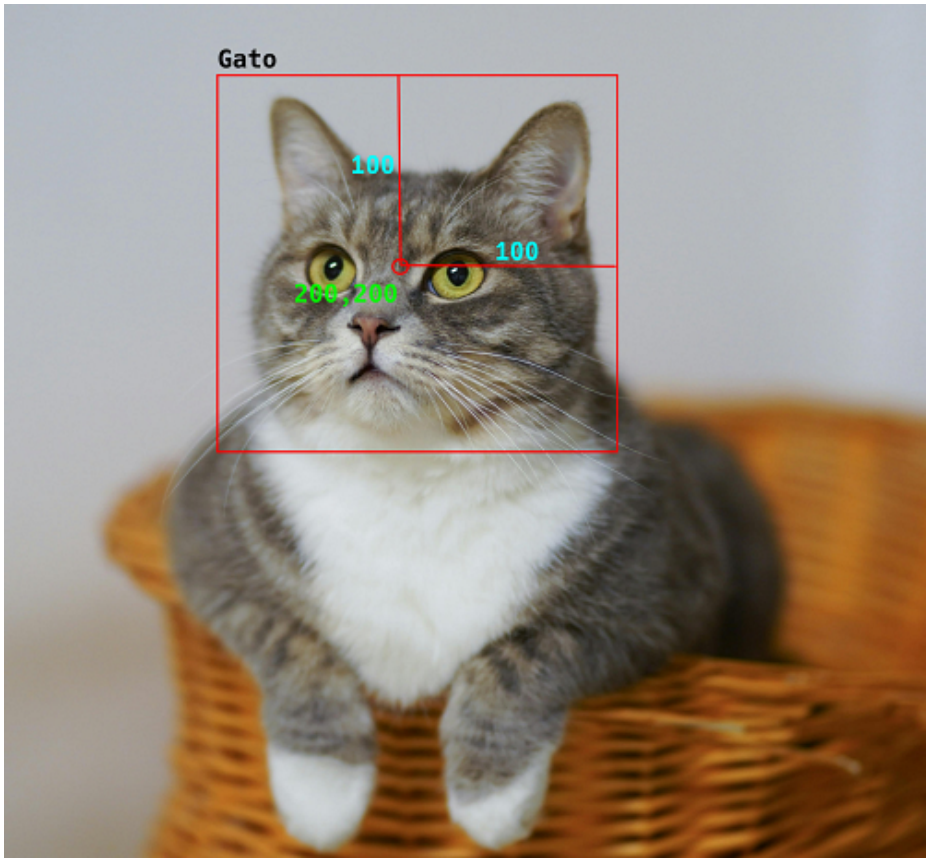


Figura 2.8: Ejemplo demostrativo del formato centro ancho y alto.

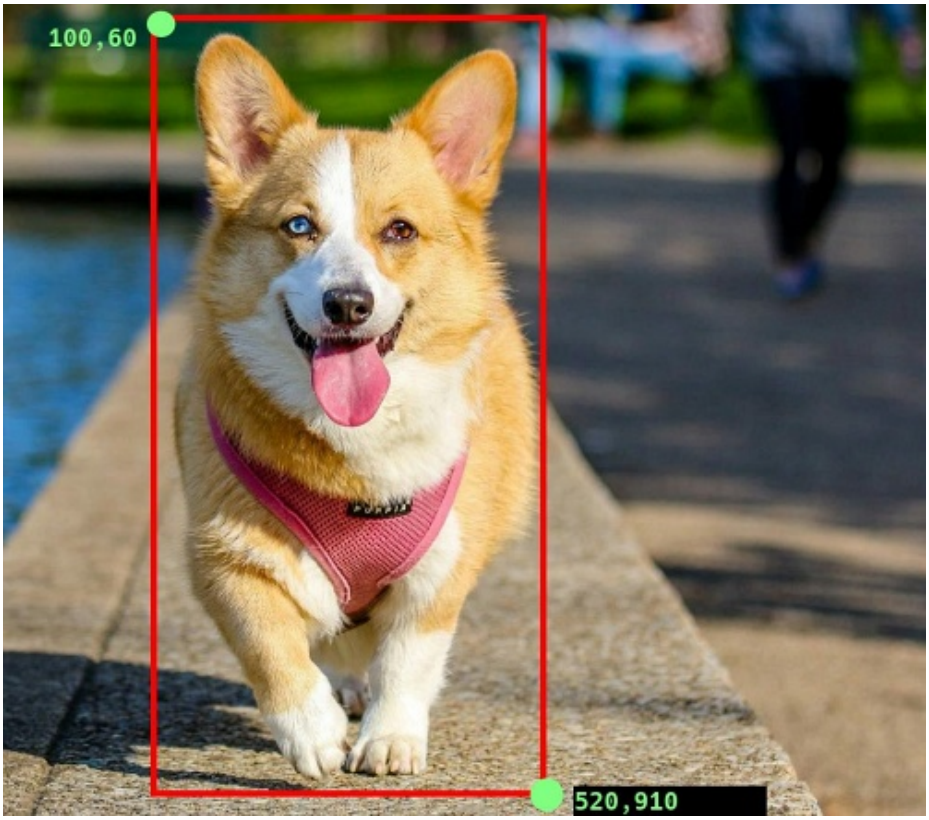


Figura 2.9: Ejemplo demostrativo del formato puntos máximos y mínimos.

Cabe mencionar que se puede cambiar o adaptar entre cualquiera de estos formatos y transformarlos de uno a otro. Esto sirve porque algunas operaciones internas que se realizan en las redes CNN para detección están optimizadas para usar uno u otro formato.

Teniendo esto en cuenta podemos observar el siguiente ejemplo al igual que el anterior de clasificación pero ahora sobre una red CNN para detección.

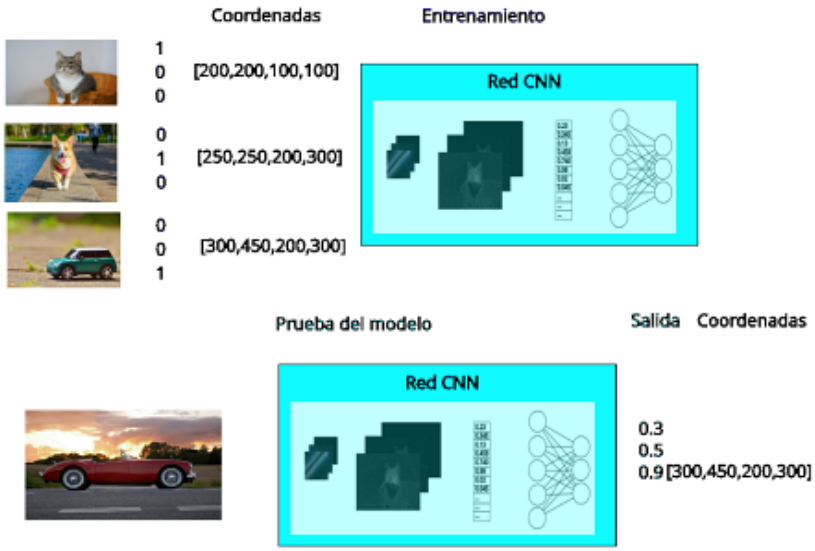


Figura 2.10: Ejemplo de una red CNN en un problema de detección.

Como se puede ver en la figura 2.10, ahora no solo se ingresa el dato sobre la clase a la que pertenece un objeto, sino también sus coordenadas y a la salida obtenemos las coordenadas y su clasificación. Por tanto, para la salida no tenemos un solo resultado sino más bien un tensor que contiene todas las detecciones es decir los N elementos que se hayan considerado como objetos detectables más sus respectivos grados de pertenencia.

Es fácil intuir, entonces, que la característica principal entre este tipo de redes CNN para detección de objetos es la forma la en que tratan las cajas y las técnicas utilizadas para proponerlas en posibles regiones, así como para decidir que caja se selecciona como parte de un objeto.

2.3.3. Red convolucional YoloV5

La red convolucional YoloV5 es la quinta de las versiones de este tipo de redes que ha mostrado ser de las mejores redes en cuanto a su relación velocidad/exactitud, ya que usualmente si se mejora uno de los aspectos, el otro tiende a empeorar bastante como podemos ver en una de sus gráficas reportadas en [1].

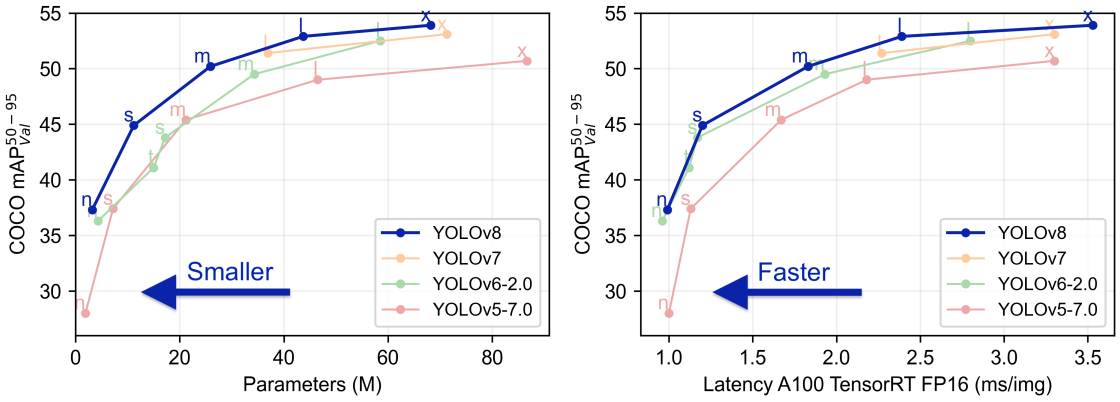


Figura 2.11: Gráfica que muestra el tamaño de los parámetros entrenables contra la exactitud media. Del lado derecho gráfico de velocidad en milisegundos contra exactitud media. Estas gráficas fueron obtenidas de [1].

En la figura 2.11 podemos observar los resultados obtenidos con las diferentes versiones de la red Yolo, así como los diferentes tamaños de las mismas, ya que éstos están clasificados como:

- Nano : representado como n
- Small : representado como s
- Medium : representado como m
- Large : representado como l
- Extra Large : representado como x

El gráfico de la izquierda de la figura 2.11 también muestra el aproximado de la cantidad de parámetros en un orden de millones sobre el eje de las x y sobre el eje de las y la exactitud media del modelo entrenado con la red COCO.
Y del lado derecho de la figura 2.11 se muestra de igual forma sobre el eje de las y la precisión media. Sin embargo, sobre el eje de las y ahora se muestra el tiempo de procesamiento por imagen en milisegundos sobre la plataforma de NVIDIA A100 para redes neuronales.

Como podemos ver la mejora no es muy significativa de la versión 5 a la 8 en sus tamaños pequeños del small al medium, en cuanto a la relación exactitud/tamaño.

Además, se integraron muchas otras técnicas creciendo en complejidad de implementación, también tomando en cuenta que esta red es muy reciente y que es mantenida por la comunidad. Versiones posteriores a la 5 de la red Yolo no tienen artículos publicados a la fecha. Es por eso que se decidió trabajar con la red YoloV5, pues además se cuenta con un fácil acceso al modelo preentrenado para esta red.

De igual forma, en la figura 2.12 podemos observar la comparación de los distintos tamaños de red y su rendimiento, así como una comparación directa con la red EfficentNet proporcionada en [1].

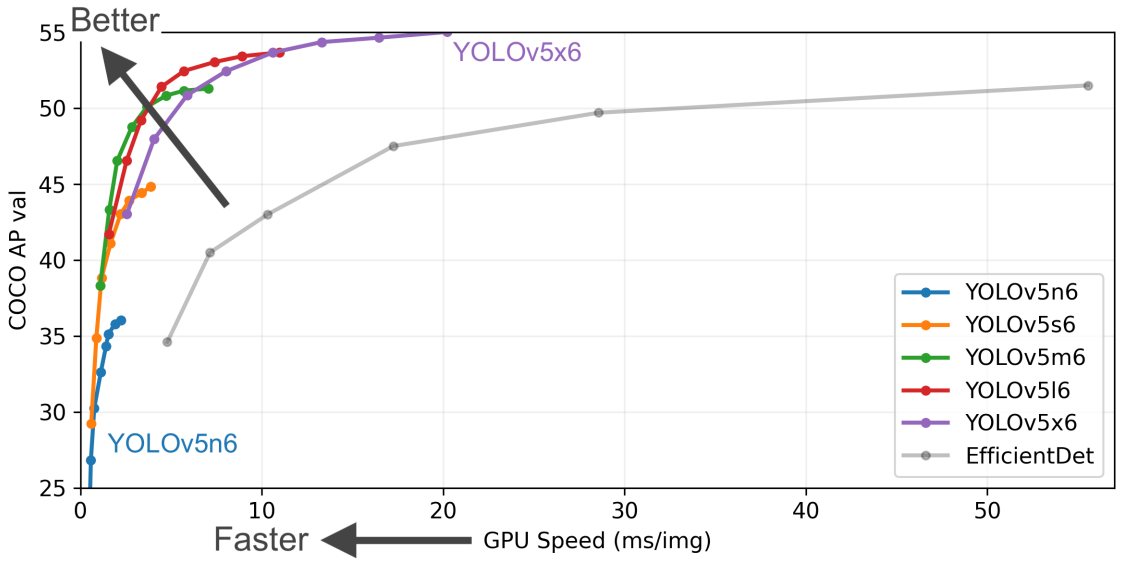


Figura 2.12: Gráfica de rendimiento YoloV5. Está gráfica fue obtenida de [1].

En el mismo repositorio proporcionado por pytorch [1] se muestra información acerca del tamaño y la entrada de cada una de las versiones de YoloV5.

Tabla 2.1: Comparación entre distintas versiones de la arquitectura YoloV5 en sus versiones de entrada de 640×640.

Nombre	Cantidad de parámetros (en millones)	Tamaño de imagen de entrada
YOLOv5n	1.9	640×640
YOLOv5s	7.2	640×640
YOLOv5m	21.2	640×640
YOLOv5l	46.5	640×640
YOLOv5x	86.7	640×640

Arquitectura

La arquitectura YoloV5 se compone de 24 capas como la red Yolo original y por lo regular todas las versiones tratan de mantenerlas. Sin embargo, como veremos a continuación en la imagen 2.15, se agrupan las capas por diferentes "técnicas" utilizadas para mejorar la red. Es decir, no son como tal 24 capas convolucionales solamente, ya que cada una de las capas se consideran con un conjunto de capas convolucionales y de agrupación, las cuales se justifican como diversas técnicas de mejora de extracción de características. Dado esto, la variación real de la cantidad de capas convolucionales radica en el tamaño elegido como vimos en la tabla 2.1, está dado por los tamaños, nano, small, medium, large y extra large. Por tal motivo, se cambia la cantidad de parámetros entrenables pero siempre conservando la arquitectura mostrada en el diagrama de la figura 2.15.

Otra forma en la que comúnmente se representa la arquitectura Yolo es dividiendo las etapas convolucionales en secciones, de la siguiente manera:

- Red troncal: Es una modificación de la Darknet53, la red convolucional que se utilizó en anteriores versiones de la red Yolo.
- Cuello: Esta parte es la que conecta la red troncal con la cabeza, en este caso utilizando SPPF(Spatial Pyramid Pooling) y PAN-Net.
- Cabeza Esta parte se encarga de dar el resultado final de la detección, por lo tanto, éste incluye la red totalmente conectada, es decir, la red de clasificación.

Esto puede verse en el diagrama 2.13 reproducido del artículo [2].

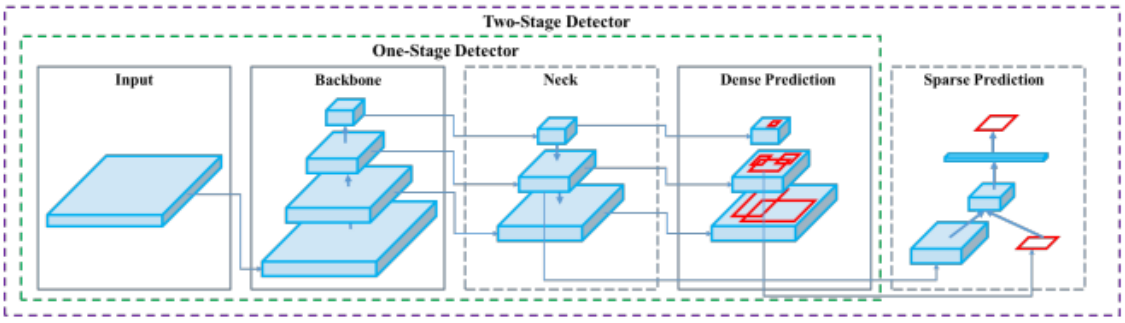


Figura 2.13: Diagrama de la arquitectura Yolov4 representando otra forma en que se dividen las capas de las arquitecturas Yolo modernas (este diagrama fue tomado del articulo [2]).

.

Tomando todo esto en cuenta, podemos empezar por entender que existen las denominadas "capas de la arquitectura YoloV5" llamadas C3 y éstas internamente se componen de varias capas convolucionales como podemos ver en la figura 2.14.

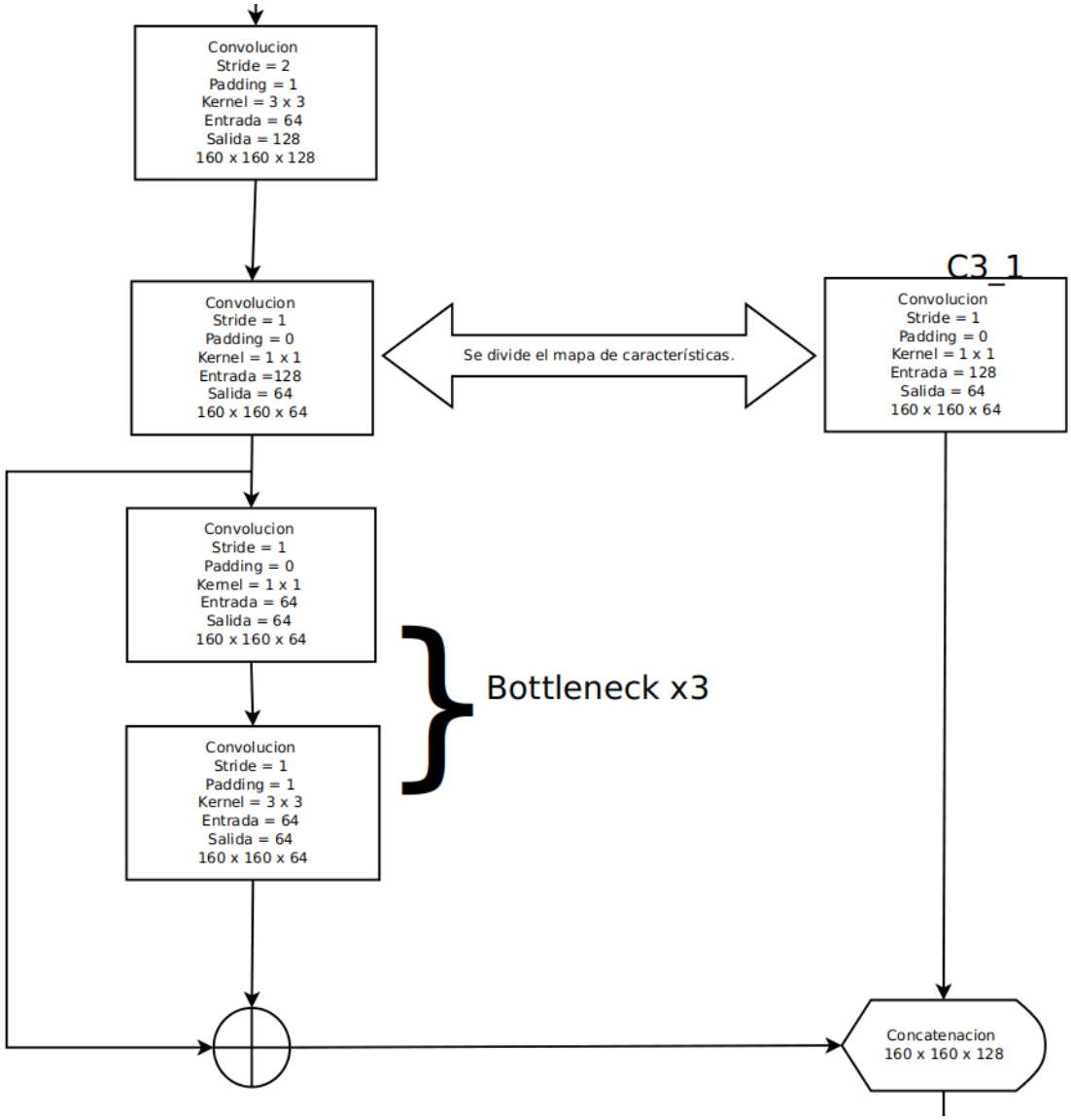


Figura 2.14: Ejemplo de capa C3

.

En la figura 2.14 podemos observar cómo el número de mapas de características de entrada se divide en dos, manteniendo la mitad intacta, mientras que la otra mitad del mapa de características que mandan a otra subcapa llamada bottleneck o cuello de botella que se compone de 2 convoluciones, esta subcapa se repetirá N veces. En el diagrama están simplificadas las partes más redundantes, por lo tanto, eso se repetirá 3 veces para este ejemplo. La salida del resultado de aplicar el cuello de botella 3 veces se sumará a la mitad intacta de entrada y este valor, a su vez, se concatenará

con la otra mitad de los mapas de características. En este ejemplo, la entrada del mapa de características es de 128 dividiéndolo así en 2 partes de 64 y después de todo el proceso anteriormente mencionado se vuelven a concatenar ambos mapas de características volviendo a una salida de 128.

Después de cada operación C3, sigue una reducción (usualmente a la mitad) utilizando filtros de 3×3 para después aplicar de nuevo a los nuevos mapas de características reducidos. Esto se repetirá 4 veces incrementando al doble el número de iteraciones sobre el cuello de botella excepto por la última operación C3 antes de la operación SPPF. Toda esta región, incluyendo la operación SPPF comprende la región troncal y cuello que en la versión 4 de la red Yolo se consideraban como dos regiones. En la versión 5 de la red Yolo se denomina solo como región troncal.

2.3.4. Operaciones especiales de la arquitectura YoloV5

Como vimos, la arquitectura YoloV5 se basa principalmente en un conjunto de diferentes operaciones y técnicas utilizadas que por sí mismas se utilizaron con diversos fines. En esta sección abordaremos dichas operaciones y su fundamento.

Agrupamiento piramidal espacial (spatial pyramid pooling)

Esta es una técnica que fue investigada en el artículo [3] para usar tamaños de entrada dinámicos, es decir, que las convoluciones no se apliquen sobre regiones del mapa de características fijo, sino que éstas puedan aplicarse sobre regiones de interés.

Esta se aprovecha de la operación de agrupación (pooling) y de ahí su nombre para agrupar las regiones de interés. Como vimos anteriormente, la agrupación permite que nos quedemos con las características más significativas o de mayor valor, reduciendo así el tamaño del mapa de características. mientras que con otras técnicas las regiones de interés son de tamaño fijo. En este caso se proponen dichas regiones como podemos ver en la figura 2.16 [3].

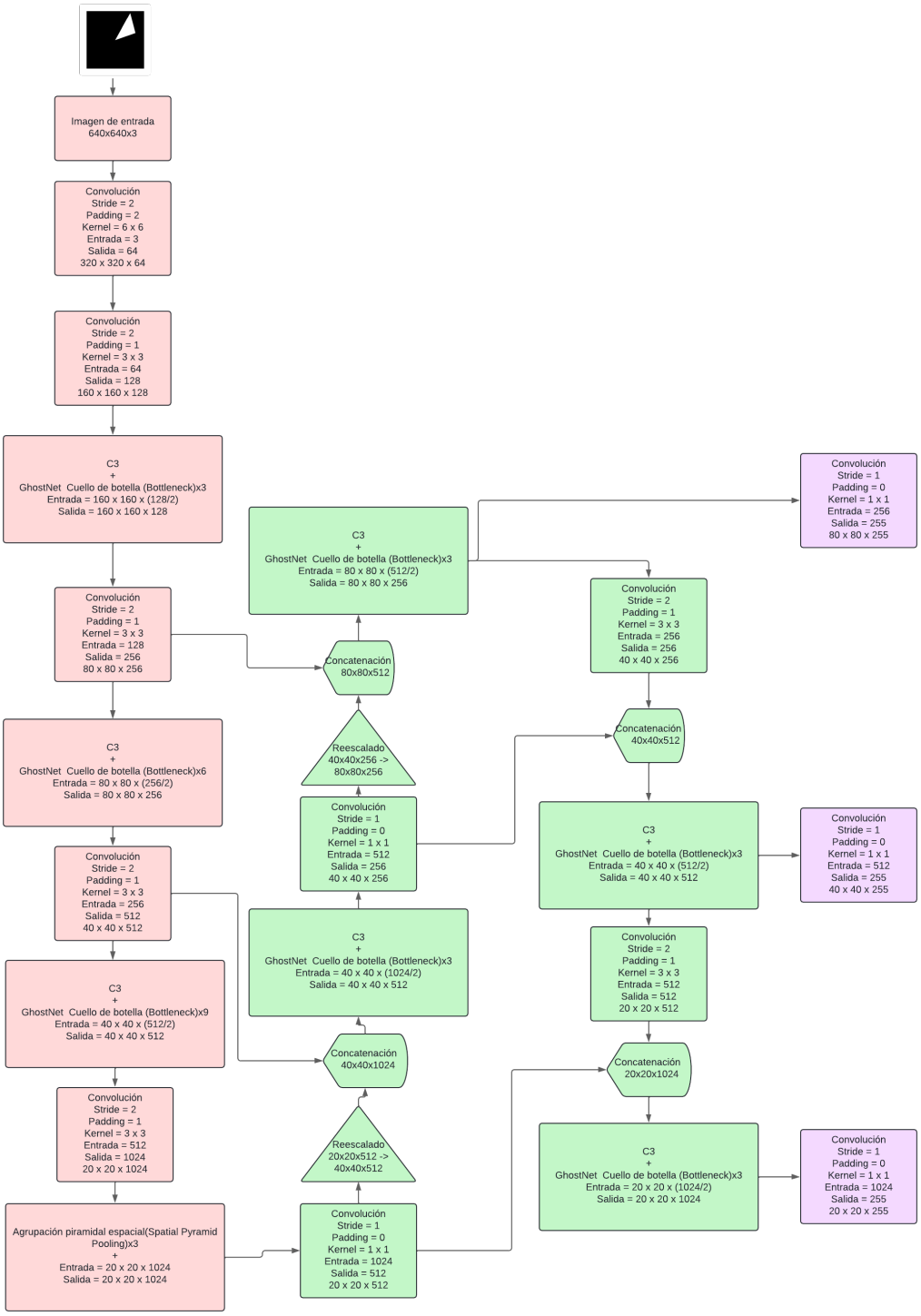


Figura 2.15: Diagrama de la arquitectura Yolov5 subdividido en regiones: rojo, región troncal (backbone); verde: cuello (neck); y morado: cabeza (head).

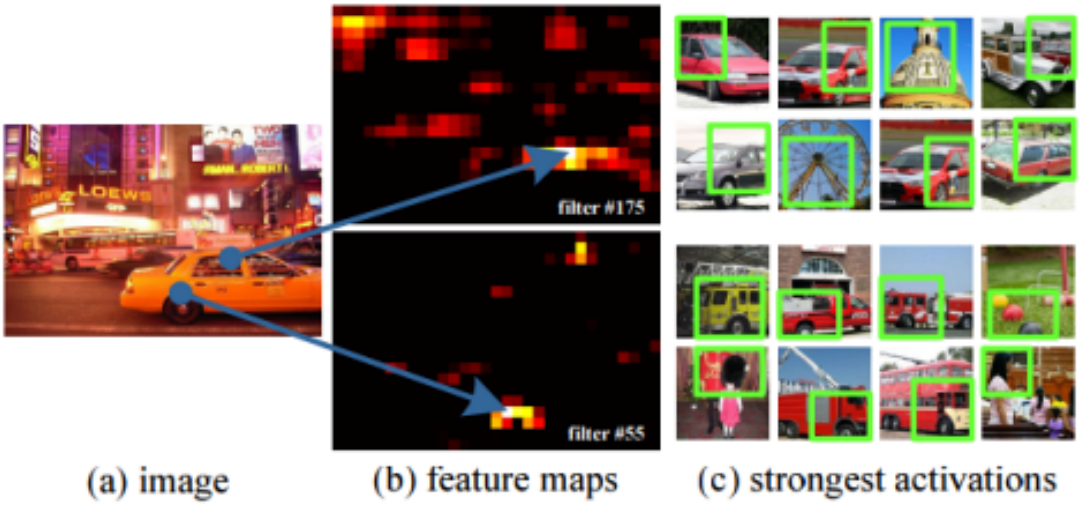


Figura 2.16: Ejemplo de cómo se proponen las regiones de interés dependiendo de la región con mayor activación [3].

Como se puede observar al utilizar la operación de agrupación con diferentes tamaños de filtro, se pueden acentuar mejor las características, denotando así las regiones de interés, las cuales posteriormente se pasarán a las capas totalmente conectadas o la parte de clasificación. En la figura 2.17 podemos ver la arquitectura propuesta, delimitada por la región rectangular. Se puede observar la parte del SPP que consiste en aplicar tres diferentes operaciones de agrupamiento del máximo con diferentes tamaños de filtro después de hacer la parte de extracción de características con las convoluciones anteriores. En el mismo artículo [3] se menciona que se probó esta técnica en la red convolucional de detección RCNN (Regional Proposal Convolutional Neural Network). Con el conjunto de datos PASCAL VOC 2007 mostró ser un 4 % más exacta la red con SPP, pasando de un 55.2 % a un 59.2 %.

Tabla 2.2: Descripción de la arquitectura propuesta de APE en el artículo [3].

Agrupación (pooling)	Tamaño	Paso (stride)
1	5×5	4
2	7×7	6
3	13×13	13

De forma resumida podemos decir que la técnica de agrupamiento piramidal espacial nos permite trabajar con regiones de interés de tamaño variable aprovechando la operación de agrupación (Max-Pooling) dejándonos con los mapas de características más representativos al utilizar los tres tamaños diferentes de ventana como se muestra

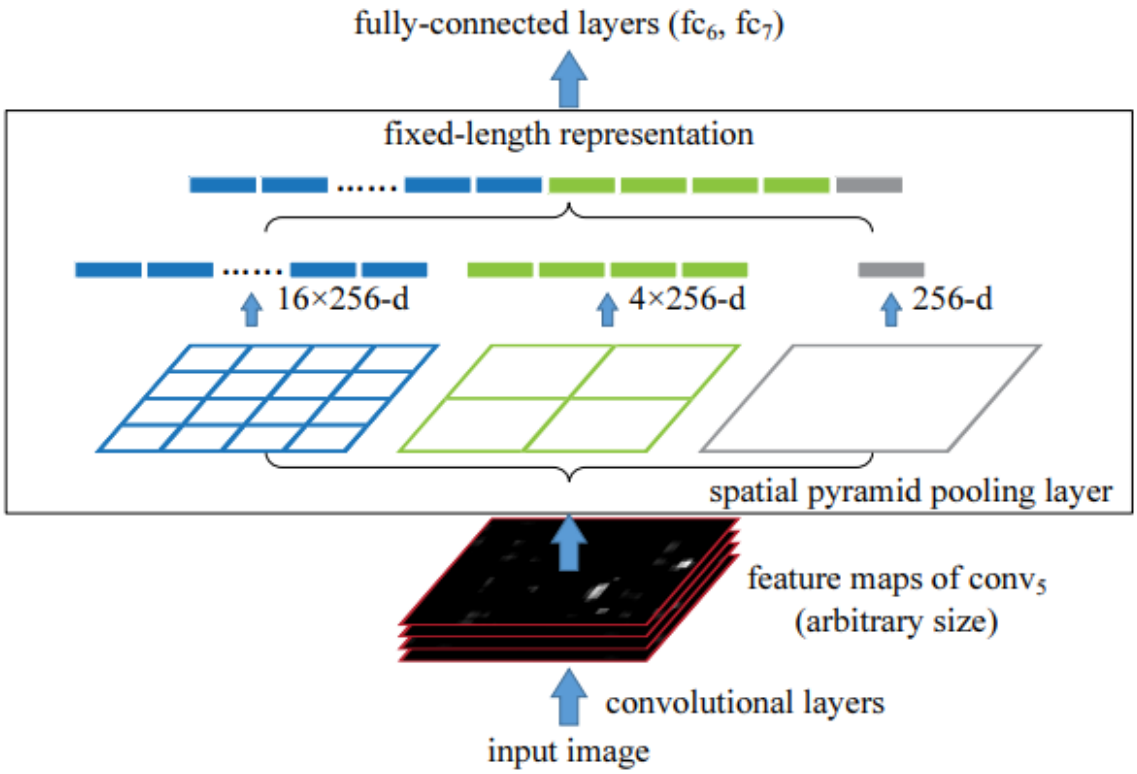


Figura 2.17: Arquitectura propuesta de APE (Agrupación Piramidal Espacial). Este diagrama fue extraído de [3].

en la tabla 2.2 y agrupándolas posteriormente, de esta forma ahorrándonos el proceso de segmentar la imagen.

GhostNet

Esta red según el artículo [4], más que una técnica es una red como tal. Sin embargo, dentro de esta red convolucional lo que la hace especial es una sección llamada GhostNet bottleneck (cuello de botella) que lo que trata de hacer es obtener más características de una misma entrada haciendo varias iteraciones sobre sí misma y sumando dichas características haciendo que resalten más regiones de activación del mapa de características. Como podemos observar en la figura 2.18 esta es la misma

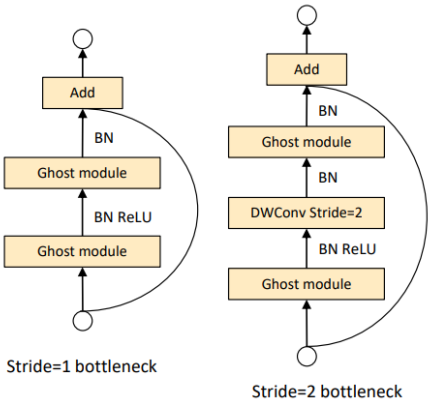


Figura 2.18: Arquitectura de módulo Ghostnet Bottleneck (cuello de botella) [4].

técnica utilizada en el bottleneck de las operaciones C3 de la red YoloV5.

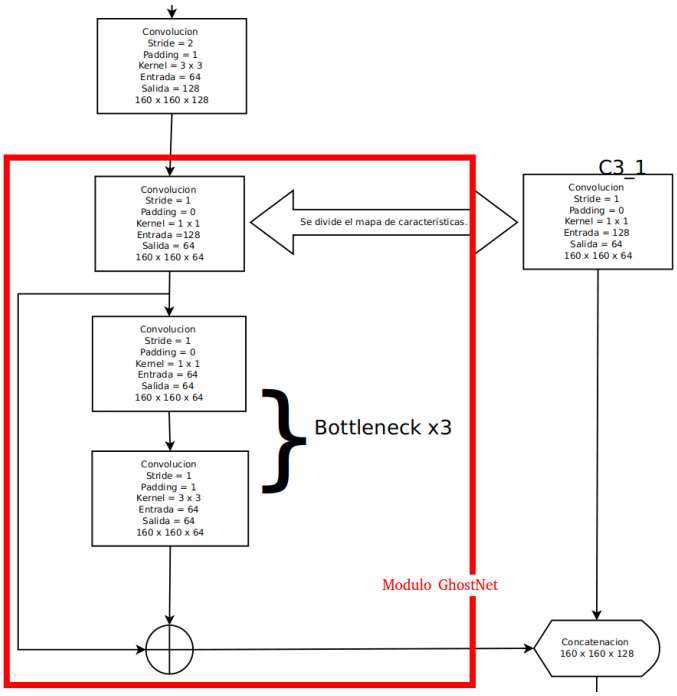


Figura 2.19: Módulo cuello de botella de GhostNet en la red YoloV5

El uso de esta técnica mostró que es capaz de remarcar o generar mapas de características más contrastadas, y extrayendo más características al concatenar los mapas de características de entrada con los procesados. Esta técnica fue aplicada sobre la arquitectura VGG-16 y ResNet-56 para clasificación, mostrando que mantiene la exactitud de la red original, pero con menos cálculos de punto flotante y, por tanto, siendo una buena opción, puesto que las operaciones de punto flotante son más costosas computacionalmente.

Tabla 2.3: Comparación de arquitecturas VGG-16 y ResNet-56 con sus variantes con la técnica GhostNet implementada [4].

Modelo	Operaciones de punto flotante (FLOPs)	Exactitud
VGG-16	313M	93.6
VGG-16 GhostNet	158M	93.7
ResNet-56	125M	93.0
ResNet-56 GhostNet	63M	92.7

En el artículo [4] se muestra una representación de los mapas de características dada la misma imagen de entrada para observar de manera más visual el efecto que produce sobre los mapas de características. En esta técnica se puso la misma imagen de entrada en la red ya entrenada VGG-16 y se comparó con la red VGG-16 GhostNet y se observaron los mapas de características de la segunda capa de convolución de la

red mostrando así los mapas de características extras que se obtuvieron después de aplicar esta técnica.

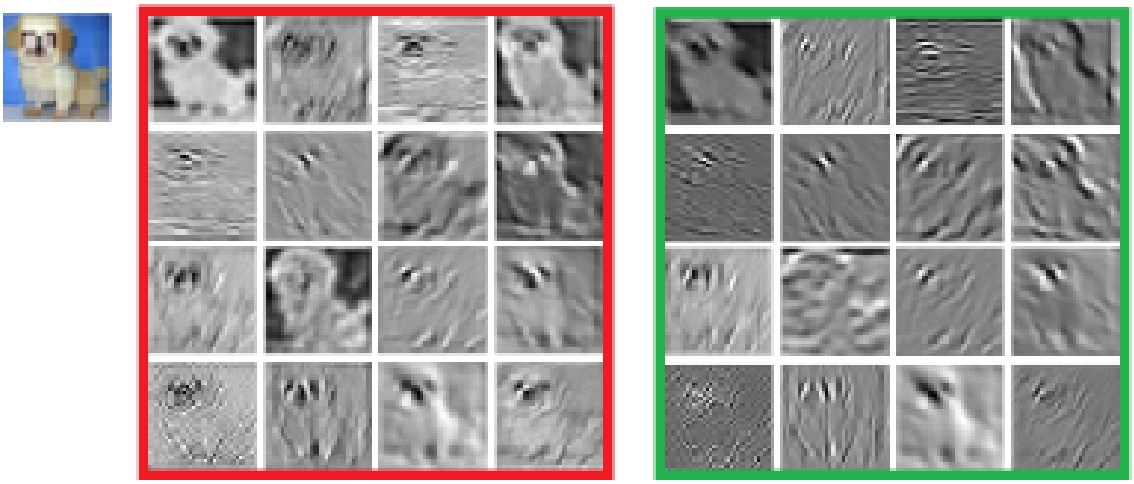


Figura 2.20: Comparación del mapa de características original en el recuadro rojo y en el recuadro verde, el mapa de características obtenido con la técnica GhostNet, Este ejemplo fue extraído del artículo [4].

Red piramidal de características (feature pyramid network)

En el área de visión, específicamente en el problema de detección utilizando redes neuronales convolucionales se han llevado a cabo diferentes técnicas para segmentar o proponer regiones de interés, es decir, regiones donde se encuentra un objeto para posteriormente ser clasificado. Una de estas técnicas ampliamente usada consiste en crear una pirámide de mapas de características que nos permite hacer detecciones a diferentes escalas o relaciones de aspecto. A diferencia de una red convolución normal de clasificación donde se realiza una predicción al final de todas las convoluciones, en una red piramidal de características se hacen predicciones a diferentes tamaños de mapas de características de entrada.

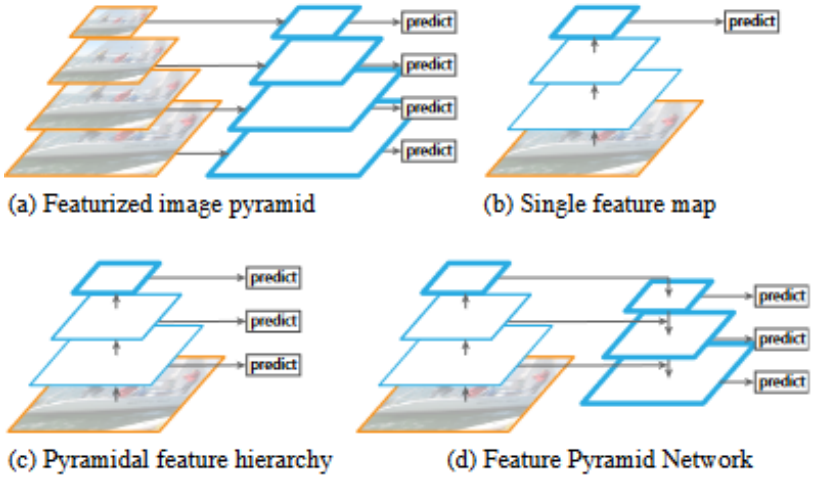


Figura 2.21: Ejemplos de las diferentes técnicas de extracción de características piramidal para detección. [5].

Lo que podemos observar en la figura 2.21 son los siguientes ejemplos:

a. Pirámide de caracterización de imagen: En este caso se entrena la red con imágenes de diferentes tamaños, ya sea teniendo la entrada original y escalando la imagen con algún algoritmo de escalado tradicional. Extrayendo los mapas de características de cada una de las imágenes, esto es el equivalente a tener varias redes entrenadas con diferentes tamaños corriendo en paralelo.

b. Único mapa de características: Es la forma más sencilla y básica de una red convolucional esto es ampliamente utilizado por redes convolucionales de clasificación donde entra una imagen de un tamaño determinado y es a esta a la que se le aplican todas las convoluciones y no es hasta el final de estas donde se realiza la clasificación.

c. Jerarquía piramidal de características: Es muy parecida a la anterior donde entra una sola imagen y a esta se le aplican convoluciones con diferentes filtros reduciendo las dimensiones del mapa de características, sin embargo, en esta no

es necesario esperar a que terminen todas las convoluciones para empezar a hacer predicciones, sino que se aprovechan los diferentes mapas de características de diferentes tamaños para clasificar esta técnica es utilizada por redes de detección como la SSD y la Yolo en su primera versión es normalmente representada como una rejilla de diferentes dimensiones sobre la imagen, donde el tamaño de la rejilla está dada por el tamaño del mapa de características en cada capa.

d. Red piramidal de características: En este caso de igual forma se parte de un solo tamaño de imagen y se va reduciendo el mapa de características a través de diferentes filtros, no obstante, al llegar al tamaño mínimo del mapa de características empieza una expansión de la misma concatenando el mapa de características de las primeras etapas teniendo así mapas de características más abstractos así como menos abstractos de las mismas dimensiones en un solo tensor por capa, haciendo para cada una de estas la clasificación, enriqueciendo así la profundidad del mapa de características. Esta es la arquitectura aprovechada por la red YoloV5 y propuesta en el artículo [5].

Esta técnica ofreció una capacidad única para mejorar la detección a diferentes escalas sin tener que añadir algoritmos más complejos para proponer tamaños de rejilla o cajas, reduciendo así el costo computacional obteniendo así una mejor precisión en la detección de objetos de diferentes dimensiones gracias la múltiple detección de diferentes características a diferentes escalas del mapa de características. Esto se logra atreves del uso de dos técnicas llamadas de abajo hacia arriba (Bottom-up) y arriba hacia abajo (top-down), donde la primera es la clásica estructura de red convolucional donde se parte de un tamaño arbitrario y se va reduciendo las dimensiones del mapa de características, pero aumentando la profundidad por el número de filtros aplicados. Mientras que la segunda toma el mapa de características más abstracto o de menor resolución y lo combina con secciones de mayor resolución, haciendo mapas de características más enriquecidos.

Aumentación de datos (data augmentation)

Esta es una técnica implementada solo para el entrenamiento y no afecta a la arquitectura de la red convolucional, esta técnica está más relacionada con los datos de entrenamiento. Que trata de partir de los datos de entrenamiento, extraer características de los mismos datos y crear datos de entrenamiento sintéticos extras en tiempo de entrenamiento. Teniendo más datos con los que el modelo puede entrenar y reduciendo así el almacenamiento requerido de los datos, pero aumentando un poco el tiempo de ejecución al implementar estas técnicas.

Al estar tratando con imágenes, en este caso las técnicas de aumentación utilizadas tienen que ver con los píxeles de las figuras encerradas de los datos de prueba, en este caso la implementación de YoloV5 de [1] tiene seis técnicas de aumentación de datos.

1. Aumentación de datos mosaico: Consiste en tomar cuatro imágenes del conjunto de datos de entrenamiento y crear una nueva imagen, dejando espacios aleatorios

entre las cuatro imágenes y reescalando las imágenes dentro de la nueva imagen creada por el conjunto de las imágenes seleccionadas.

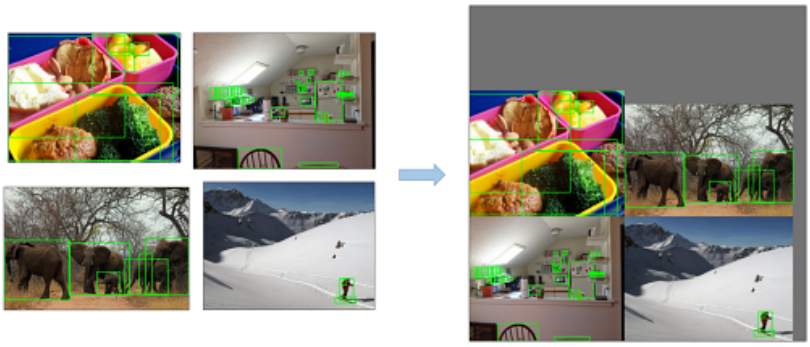


Figura 2.22: Ejemplo de aumentación de datos mosaico YoloV5 [1].

2. Aumentación de datos, copiar y pegar: consiste en tomar el objeto encerrado de una de las imágenes del conjunto de entrenamiento y pegarlo sobre otra imagen del conjunto de entrenamiento en una posición al azar.

Figura 2.23: Ejemplo de aumentación de datos copiar y pegar [1].

3. Transformaciones aleatorias: esto es parecido a la técnica mosaico, sin embargo, este incluye poner las imágenes rotadas, giradas e incompletas.

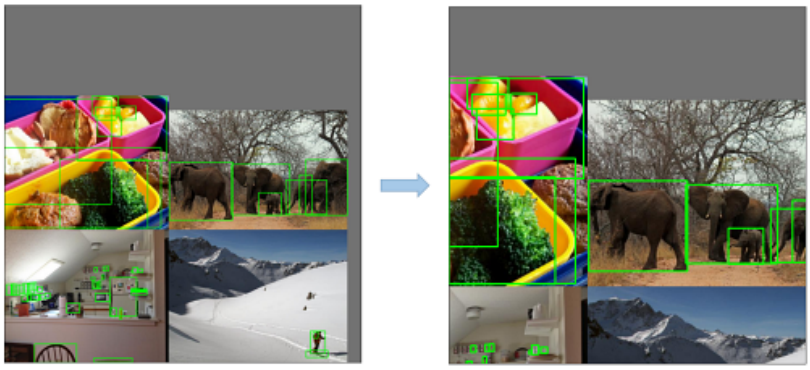


Figura 2.24: Ejemplo de aumentación de transformaciones aleatorias [1].

4. Aumentación de datos por mezcla: consiste en tomar de igual forma cuatro imágenes del conjunto de entrenamiento y superponerlas en posiciones aleatorias, haciendo transparente cada una de estas y variando el canal alpha para lograrlo, creando de esta forma la nueva imagen.

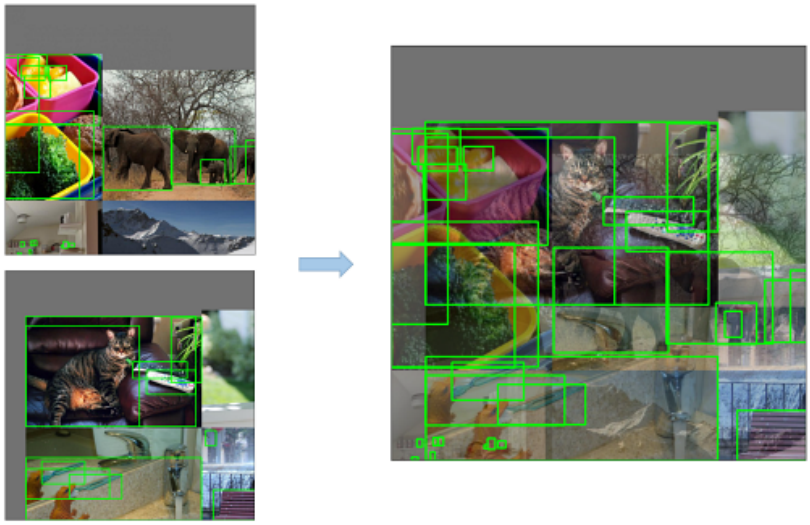


Figura 2.25: Ejemplo de aumentación de datos por mezcla Yolov5 [1].

5. Aumentación de datos matiz, saturación y valor (HVS): Este consiste en variar ligeramente los colores de la imagen, ya sea saturándolos o cargando los valores hacia diferentes canales para generar una imagen nueva.

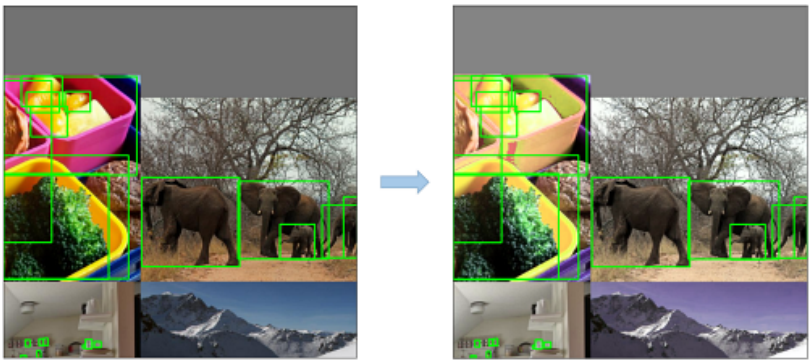


Figura 2.26: Ejemplo de aumentación de datos matiz, saturación y valor (HVS) Yolov5 [1].

6. Aumentación de datos orientación aleatoria: este método consiste en girar horizontalmente de igual forma las cuatro imágenes seleccionadas para crear la nueva imagen.

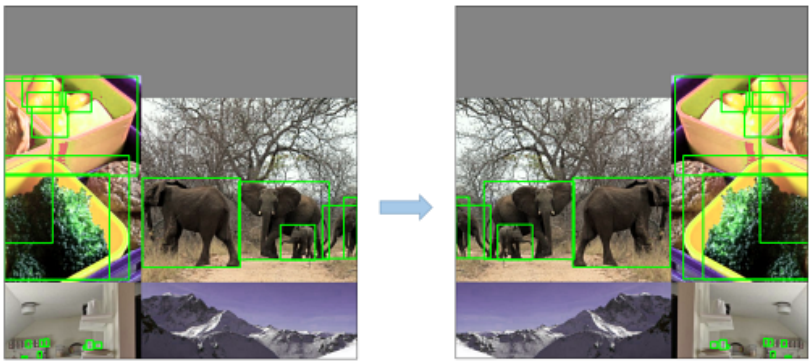


Figura 2.27: Ejemplo de aumentación de datos orientación aleatoria Yolov5 [1].

2.3.5. Red convolucional SSD (single shot multibox detector)

La red convolucional SSD fue contemporánea a la versión 1 y 2 de la red Yolo y fue presentad por ingenieros de Google y la universidad de Michigan en el artículo [6]. Esta red utiliza una sola red convolucional y predice cuadros delimitadores de diferentes tamaños, esto en una sola ejecución como bien lo dice en su mismo nombre "un solo disparo" (single shot).

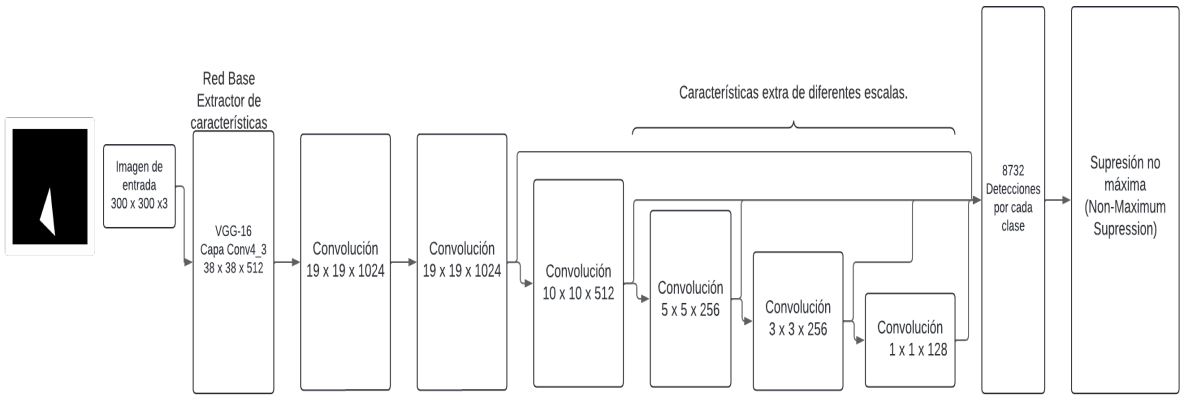


Figura 2.28: Arquitectura de red convolucional SSD [6].

Como podemos ver en la figura 2.28, la arquitectura propuesta en el artículo [6] se compone de tres diferentes partes, la primera es una arquitectura convolucional de clasificación, en este caso se propone utilizar la red VGG-16, sin embargo, no se utiliza toda esta red. En el artículo [6] se hicieron varias pruebas y se determinó que la forma más eficiente es utilizar hasta la capa Conv4_3 que da una salida de $38 \times 38 \times 512$ siendo esta capa el primer tamaño de ventana por el cual se detectara después se pasa por una serie de convoluciones extra que nos reducirán las dimensiones, la cual nos darán unas salidas para cada detección de los siguientes tamaños.

- $10 \times 10 \times 512$
- $5 \times 5 \times 256$
- $3 \times 3 \times 256$
- $1 \times 1 \times 128$

Donde según la arquitectura propuesta del artículo [6] da una salida fija de 8732 detecciones por clase, es decir, son el número de ventanas que propone por cada clase sobre la imagen, cada una de estas ventanas es clasificada, y posteriormente con

la operación de supresión no máxima (Non-Maximum Supression) se seleccionan las ventanas mejor clasificadas y detectadas es decir en esta parte con esta operación se determina la ubicación del objeto al eliminar las ventanas que se acerquen menos a las coordenadas reales dadas en el entrenamiento.

Operaciones y técnicas especiales de la arquitectura

Como podemos observar de igual forma que la red YoloV5 la red convolucional SSD utiliza un conjunto de varias técnicas previamente creadas que en conjunto permiten generar una red convolucional más robusta que permita hacer diferentes cosas para las que fueron creadas originalmente.

Indice Jaccard

El índice Jaccard o intersección sobre la unión que mide el grado de similitud dados 2 conjuntos siguiendo la siguiente expresión $J\left(A,B\right)=\frac{A\cap B}{A\cup B}$, esto es usualmente utilizado para medir el grado de similitud entre las cajas que predice el modelo con las imágenes de pruebas y las etiquetas reales de las cajas con las que se crearon.

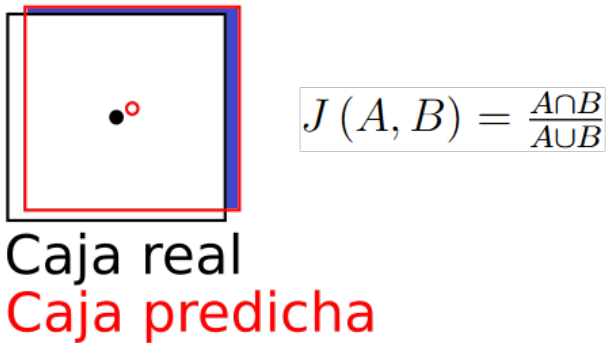


Figura 2.29: Ejemplo de uso de índice Jaccard en redes neuronales convolucionales.

En la figura 2.29 podemos ver que el área azul pertenece a la intersección, mientras que la unión pertenecería al área total de ambos conjuntos.

Detección por mapas de características de múltiples escalas

Como podemos observar hasta ahora la operación de detectar conlleva un proceso de segmentación de la imagen, el cual delimitara la posición de los objetos a detectar sobre la imagen, por lo tanto, no es de extrañarse que lo que caracterice una red convolucional de detección de otra es la forma en la que se segmentan o se crean estas regiones de interés sobre la imagen. En el caso de la red SSD a diferencia de la YoloV5 que se aprovechaba de la técnica de agrupación piramidal espacial (APE) y red de

características piramidal para generar estas regiones de interés, la red SSD utiliza una técnica similar en concepto, pero distinta en técnica, como vimos en la figura 2.21 hay diferentes formas de aprovechar los diferentes tamaños de los mapas de características conforme se aplican las convoluciones para el problema de detección en este caso la forma más similar como trabaja la red SSD siguiendo el mismo diagrama sería la llamada jerarquía de características piramidal (pyramid feature hierarchy). Donde

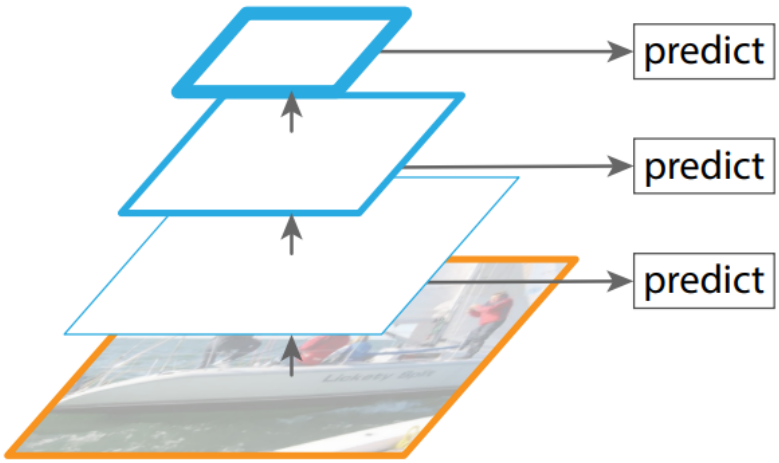


Figura 2.30: Arquitectura jerarquía de características piramidal [5].

como podemos observar en la figura 2.30 se aprovecha la reducción del mapa de características haciendo predicciones para cada uno de los tamaños de las convoluciones, directamente entonces según la arquitectura SSD propuesta el número de predicciones. La red SSD utiliza cuadros predeterminados llamados anclas (anchor) parecidos a los utilizados por la red RCNN (Region proposal network) que básicamente se usan como referencias iniciales para las predicciones estas son las que añaden los diferentes tamaños y escalas en la tarea de defección así como proporcionan la rejilla inicial de predicciones.

En el entrenamiento lo que sucede es una estrategia de ajuste donde las regiones iniciales junto con sus predicciones se tratan de ajustar a algo que en el artículo [6] denominan región verdadera (ground truth) utilizando como guía de proximidad el índice Jaccard o unión sobre intersección. De esta forma se discriminan las cajas o detecciones que obtengan menor índice Jaccard y así se aproxima a la ubicación real del objeto.

Supresión del no máximo

Una de las últimas etapas de la red SSD es utilizar la operación de supresión del no máximo, que según explica el artículo [7] sirve principalmente para disminuir el número de cajas delimitadoras que apuntan al mismo objeto en el problema de detección. La versión clásica del algoritmo publicado en el artículo [13] que habla de

Figura 2.31: Ejemplo de la operación supresión no máxima del artículo [7].

supresión no máxima funciona de la siguiente manera:

- Ordenamos las cajas por confianza, una vez obtenidos los grados de pertenencia de cada una de las cajas se ordenan de mayor a menor según la confianza que represente para cada clase.
- Seleccionamos la caja con mayor confianza.
- Para las cajas restantes aplicamos el índice Jaccard.
- Eliminamos todas las cajas cuyo índice Jaccard sea menor al umbral definido que según el índice Jaccard si es menor a 0.5 se considera normalmente como un conjunto totalmente diferente.
- Se repite el proceso hasta que al repetir el proceso el resultado del número de cajas sea igual al anterior proceso realizado.

Capítulo 3

Desarrollo

3.1. Propuesta de solucion

La propuesta de solución de forma general se compone de 3 etapas:

- Creación de conjunto de datos sintéticos: En este punto se establecerá el formato del conjunto de datos, se utilizará el modelo de la cámara oscura para la creación de las imágenes sintéticas, así como se establecerá un fondo que permita diferenciar el marcador y facilite el entrenamiento de la red.
- Entrenamiento usando la red YoloV5: Se utilizará el modelo YoloV5 preentrenado, con la biblioteca de Python PyTorch.
- Entrenamiento usando la red SSD: Se utilizará el modelo SSD preentrenado, con la biblioteca de Python PyTorch.

Utilizar el mismo hardware para el entrenamiento, así como para la prueba de los modelos entrenados.

Tabla 3.1: Especificación de hardware

Etapa	RAM	CPU	GPU	S.O
Entrenamiento	252GB	Xeon 24 núcleos 48 hilos	RTX3070 8GB	Fedora
Prueba en tiempo real	32GB	Ryzen 5 5600x	RTX 2060 6GB	Manjaro

Se establece como base de parámetros utilizar para ambas arquitecturas un entrenamiento de 300 épocas. para medir los tiempos de entrenamiento posteriormente al utilizar transferencia de aprendizaje. También se propone utilizar la misma cantidad de imágenes de entrenamiento por clase en ambas arquitecturas para comparación, sin embargo, si la arquitectura requiere más imágenes de entrenamiento se aumentarán, pero siempre respetando que para validación se tomara el 10 % del número de imágenes.

3.2. Creación de conjunto de datos sintéticos

Como vimos en el capítulo anterior del marco teórico donde se explican ambas arquitecturas y funcionamiento, podemos observar que es necesario primeramente contar con un conjunto de datos, es decir, imágenes así como los datos de las coordenadas y clase a la que pertenecen codificados para cada imagen. En este caso se decidió tomar el formato de la red YoloV5 para codificar dichos datos en archivos independientes con extensión CVS el cual separa cada uno de los datos por un espacio, así como normalizar los datos con valores de 0 a 1, quedando como se muestra en la figura 3.1.

Figura 3.1: Formato de codificación de cajas delimitadoras elegido para ambas arquitecturas

Donde, como se muestra en la figura 3.1 el formato elegido, el número coloreado de color azur representa la clase, tomando en cuenta que la forma como estará codificada cada clase será [0,1,2...n] para cada una de las clases. Seguidamente, los próximos 2 números coloreados en rojo son las coordenadas del centro del objeto. Por último, los números coloreados de color verde pertenecen al ancho y al alto de la caja.

Siguiendo las actividades del cronograma previsto en el protocolo de investigación. Se comenzó por ver la parte de la creación de un programa que nos proporcione N cantidad de datos generados automáticamente, en este caso de marcadores únicamente.

Para la realización del programa que permitiera la creación de datos sintéticos, se consideraron los siguientes aspectos:

1. Puntos del que conforman el marcador.
2. Modelo de la cámara oscura.
3. El fondo

3.2.1. Marcadores

Como vimos en el capítulo anterior, los marcadores de tipo de orden son objetos que son fácilmente distinguibles y que se conforman por diferentes áreas, área sin ruido, el área de datos y los puntos que definen la etiqueta.

Debido a esto lo primero, fue fijar los marcadores, en este caso fijar el tamaño de los marcadores, así como los puntos característicos sobre la imagen que nos permitirán diferenciar dicho marcador de otro. En este caso se propuso utilizar primeramente los marcadores definidos en la tesis de maestría de Andres Cureño Ramirez [14].

Tabla 3.2: Puntos que conforman el marcador 1

x	y
0	0
0	255
255	0
255	255
145	18
92	54

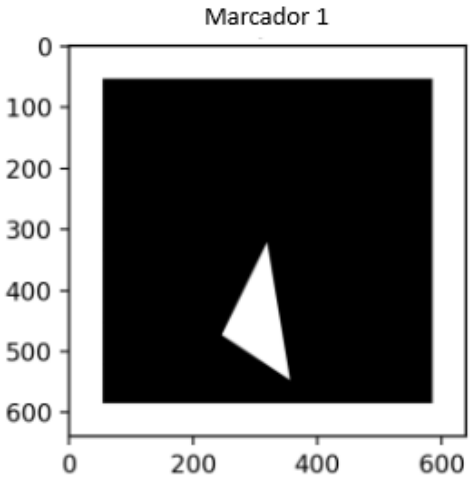


Figura 3.2: Marcador 1

Para el marcador 1 (mostrado en la figura3.2), las 4 coordenadas pertenecen al recuadro negro, es decir, el área de datos y los otros 2 puntos pertenecen a los que conformaran el triángulo que se forma cuando se unen con entre sí y al punto central de la imagen.

Tomando como referencia esta imagen, sé procedió a programar el modelo de la cámara donde se toman como parámetros de entrada los puntos correspondientes a la cámara, los puntos del plano original y los puntos del plano transformado.

3.2.2. Modelo de la cámara oscura

El modelo de la cámara oscura nos describe una proyección de un objeto de tres dimensiones sobre el plano en dos dimensiones, siendo el plano 2D una forma de representar un sensor de "Cámara" imaginaria es decir el plano en el cual se le proyectarán los puntos pertenecientes al objeto tridimensional. Por lo tanto, nosotros aprovecharemos este modelo, ya que se pueden extraer los puntos pertenecientes a estas transformaciones dándonos como resultado la proyección final. En el proceso, se emplean las coordenadas de la ubicación del objeto que necesitamos para el entrenamiento de los modelos. Este modelo está representado por la siguiente expresión matemática:

$$\lambda p = K[R|t]P \tag{3.1}$$

Donde p es el punto bidimensional de destino de la proyección, mientras que P es el punto tridimensional del objeto que deseamos proyectar, K son los parámetros de la "Cámara" imaginaria que definen las propiedades del plano el cual contiene los valores del foco, R es una matriz de rotación, y t un vector de translación.
Ya que la resolución de entrada de las imágenes para la red neuronal YOLOv5 es de 640×640, la matriz de píxeles comienza en 0 y dado que se tomará el centro como modelo, el modelo propuesto quedó en las coordenadas $x = 319.5, y = 319.5$. Una vez propuesto el modelo y dado que se utiliza el mismo modelo de cámara utilizado en previas ocasiones, se conoce el foco que es 525 por tanto se procedió a aplicar el modelo de la cámara oscura.

$$
\begin{aligned}
\lambda p &= K[R|t]P \\
&= KR[I, -c]P \\
&= K[I, -c]P \\
\lambda \begin{bmatrix} 319.5 \\ 319.5 \\ 1 \end{bmatrix} &= K[I, -c]P \\
&= K \begin{bmatrix} I, \begin{bmatrix} -c_x \\ -c_y \\ -c_z \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} 525 & 0 & -u_0 \\ 0 & 525 & -v_0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \end{bmatrix} \\
&= \begin{bmatrix} v_0 C_z & 0 & -u_0 \\ 0 & 525 & -v_0 \\ 0 & 0 & -1 \end{bmatrix}
\end{aligned}
\tag{3.2}
$$

Los parámetros de la cámara correspondientes $v_0 u_0$ así como la posición de la cámara

c_x, c_y, c_z serán elegidos aleatoriamente dentro de un rango de -4000 a 4000 dando así posiciones y transformaciones de perspectiva aleatorias a cada iteración generando al término de cada iteración los puntos necesarios ya transformados, para generar la imagen de salida.

Para la implementación del código que aplica el modelo de la cámara oscura, también se tomaron en cuenta ciertas restricciones que debían contener las transformaciones. Esto debido a que las cajas delimitadoras e imágenes generadas a partir de este modelo solo nos serán útiles para el entrenamiento de los modelos YoloV5 y SSD, si son imágenes cuyos puntos del modelo se encuentren dentro del lienzo de proyección de 640×640 en el caso de la red YoloV5 y 300×300 para la red SSD. Así mismo, la perspectiva no debe ser tan pronunciada. Es decir, no debe ser tan próxima a 90 grados, puesto que solo se vería una línea y estos datos no serían válidos para entrenar los modelos SSD y YoloV5 así como contribuirían a meter más ruido en el entrenamiento.

Como estos puntos ya están dados en las coordenadas de píxeles de la misma imagen, se tomaron éstas para crear los puntos de las cajas que envuelven al objeto (también llamada la caja envolvente). Para esto, se tomaron las coordenadas de los puntos menores y mayores respectivamente para así tomar las coordenadas más alejadas de todos los extremos y de esta forma se procedió a generar las coordenadas de dichas cajas.

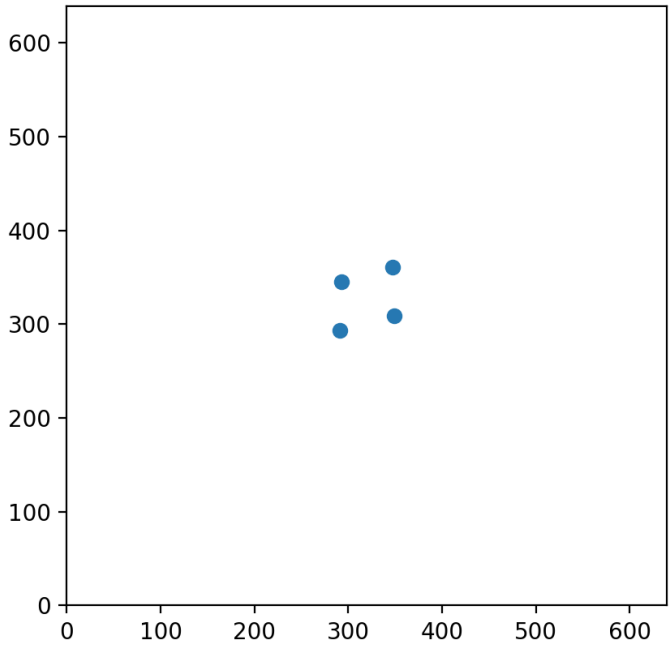


Figura 3.3: La imagen de salida al aplicar el modelo de la cámara oscura.

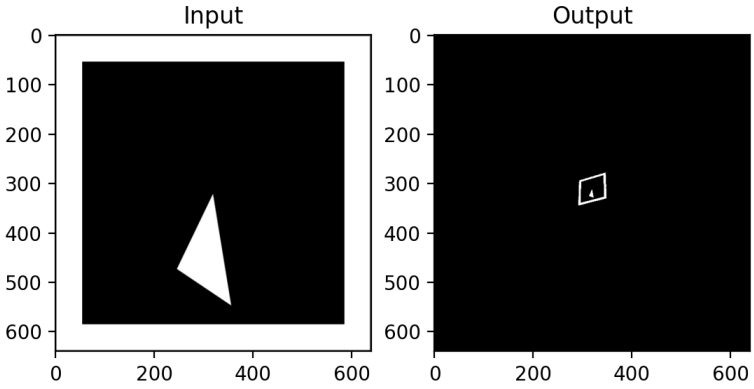


Figura 3.4: Imagen transformada.

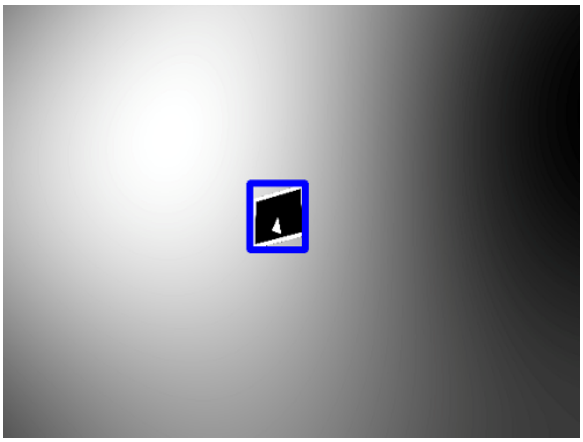


Figura 3.5: Imagen transformada con la caja envolvente al marcador

En el fondo se utilizó el algoritmo llamado ruido Perlin para generarlos, esto es, una función pseudoaleatoria que genera números entre 0 y 1 utilizada en la creación de mapas de videojuegos que necesitan generarse aleatoriamente dada una semilla, produciendo así diferentes tipos de relieves.

Para el guardado de las imágenes, los archivos contendrán el vector de entrada con los datos correspondientes a **[clase coordenadaX coordenadaY ancho largo]**, donde las coordenadas X, Y pertenecen al punto central de la caja que encierra al objeto, Esto permitirá que se dibuje la caja que lo rodea, usando el ancho y largo.

También es importante destacar que se realizó este programa que genera datos sintéticos de forma genérica. Esto significa que, cambiando solo los valores de los focos del modelo de la cámara oscura, así como las dimensiones de la imagen que se desee realizar y el rango de números aleatorios que pertenecen a las rotaciones en $[x, y, z]$ del modelo de la cámara oscura, el programa por sí solo creará las imágenes sintéticas en automático. Para ello, se usarán los parámetros dados cuidando que los 4 puntos del marcador siempre estén en la escena, es decir, que no se salgan de las dimensiones predefinidas de la imagen de salida previamente definidas. Esto hace que dicho programa sea fácilmente adaptable para crear conjuntos de entrenamiento sintéticos para ambos modelos YoloV5 [1] y SSD cuyas dimensiones de entrada varían. El modelo Yolov5 utiliza entradas de imagen de 640×640 mientras que el modelo SSD utiliza tamaños de imagen de entrada de 300×300.

3.2.3. Ruido Perlin

Se propuso utilizar el ruido Perlin como fondo debido a que genera valores aleatorios basados en gradientes. Por lo tanto, si estos valores son interpretados como intensidades de los valores de los píxeles que necesitamos como fondo, entonces esto mostrara imágenes con degradados aleatorios. Esto permite diferentes patrones más parecidos a los gradientes de luz proporcionados por una imagen obtenidos por una cámara, a diferencia de la tesis [8] donde se utilizan de fondo valores aleatorios 0 y 1

para cada pixel de la imagen, teniendo cambios bruscos en la intensidad de los píxeles.

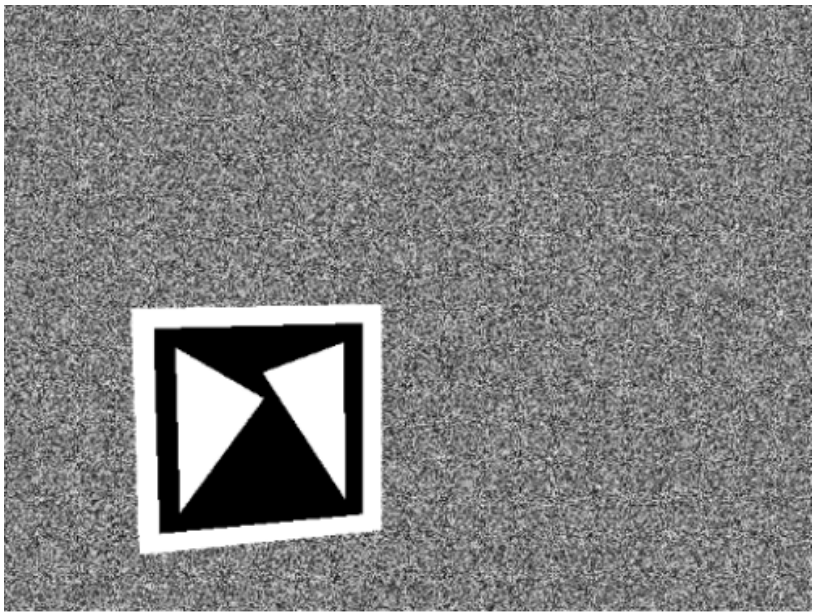


Figura 3.6: Fondo utilizado en la tesis de maestría de Gonzalo Adán Chávez Fragoso [8]

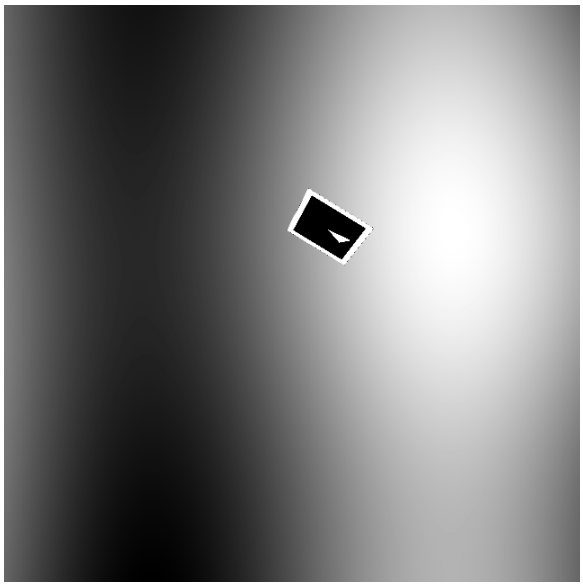


Figura 3.7: Fondo con ruido Perlin propuesto

Este tipo de ruido es famoso en el ámbito del cine y los videojuegos al crear patrones de física, como humo o polvo, así como para la creación de mapas procedurales en juegos de mundo abierto. Este tipo de ruido pseudoaleatorio se calcula por

la interpolación de varios números de gradientes precalculados, en espacios de dos dimensiones.

Se decidió utilizar la librería de Python que nos ayuda a generar ruido Perlin llamada Perlin-noise, instalada a través del gestor de paquetes pip [15]. De tal forma, simplemente pasamos a la función noise() los parámetros de tamaño de $[x, y]$ del tamaño de imagen que deseamos crear. Internamente, ya se cuenta con los gradientes correspondientes y se genera la matriz de salida con un formato correcto para imprimir con mathplotlib. De esta forma podemos rescatar y almacenar la imagen de fondo.

3.2.4. Entrenamiento YoloV5

Para comenzar se utilizó el modelo preentrenado de la red YoloV5 proporcionado por Pytorch [1]. Éste modelo se entrenó previamente con el conjunto de datos de entrenamiento COCO [9], que según el sitio oficial del conjunto de datos consta de 200 mil imágenes etiquetadas y 80 clases.

Se realizó un estudio de cómo fue entrenada la red, es decir, y como fue implementada para poder ser reentrenada con éxito. Para realizar el primer entrenamiento se crearon 2000 imágenes sintéticas de entrenamiento con el procedimiento y parámetros anteriormente mencionados, así como 200 imágenes extra de prueba para realizar las pruebas del modelo correspondientes al 10 % de las imágenes de entrenamiento.

Una vez creadas las imágenes de entrenamiento y prueba, se procedió a crear una carpeta que contendría en conjunto de datos de entrenamiento. En la documentación de la red YOLOv5 [1] se indica que ésta trabaja con un formato llamado YAML, el cual es parecido a XML o Json, en el cual se deben especificar los siguientes datos:

- Path: Ruta tomando como raíz el sitio donde se ejecuta el programa en Python.
- train: Carpeta donde se ubican los datos de entrenamiento tomando como raíz la ruta guardada en «Path» anteriormente mencionada.
- test: Carpeta donde se ubican los datos de prueba tomando como raíz la ruta guardada en «Path» anteriormente mencionada.
- names: La cantidad de clases seguida de dos puntos (:) y el nombre de la clase.

Seguido de esto se crearon los directorios correspondientes a dichas rutas y se almacenaron las imágenes para entrenamiento en la carpeta «images». Las etiquetas se almacenaron en la carpeta «labels» para entrenamiento y prueba respectivamente.

Posteriormente, se procedió a entrenar la red neuronal con los nuevos conjuntos de entrenamiento con la versión small, utilizando Pytorch y sus herramientas que facilitan el entrenamiento y re-entrenamiento de varios tipos de redes.

La documentación de la red neuronal YOLOv5 [1] menciona que la salida de la red neuronal tiene como salida un vector con los grados de pertenencia a cada clase,

así como las coordenadas con los mismos tipos da dato con los que fue entrenado, es decir **[gradosDePertenencia coordenadaX coordenadaY ancho largo]**.

Posteriormente, la documentación de Pytorch de la red Yolov5 [1] menciona que debemos ejecutar el entrenamiento del modelo con el siguiente comando de ejemplo:

```
python train.py --img 640 --epochs 3 --data coco128.yaml
--weights yolov5s.pt
```

donde los parámetros que se le pasan al archivo son los siguientes:

Tabla 3.3: Parámetros para entrenamiento según la documentación de Pytorch [1]

Parámetro	Descripción
–img	Tamaño de las imágenes de entrenamiento y validación.
–epoch	Número de épocas.
–data	Nombre o ruta del archivo previamente creado que contiene las rutas de las carpetas de los archivos de imágenes etiquetas para entrenamiento y validación, respectivamente.
–weigths	Nombre o ruta del archivo del modelo preentrenado.

Experimento con red YoloV5 entrenada

Posteriormente, se diseñó un experimento para medir las distancias y ángulos a los que se tiene un grado de pertenencia aceptable predicho con los pesos del modelo ya entrenado. El experimento consiste en medir el largo de dos mesas. En este caso, ambas mesas medían 1.20 m. Después, estas mesas fueron marcadas con cinta cada 10 cm para así mover el marcador de 10 en 10 centímetros hasta detectar una bajada menor al 60 % en el grado de pertenencia al cual corresponde al equivalente real a confundirlo con ruido en la imagen.

También se tomó en cuenta el ángulo de rotación sobre el eje X, para esto se marcó una tabla con los grados del 0 al 180 de igual forma de 10 en 10 para ir rotando el marcador sobre esta referencia y de esta forma detectar el ángulo en el que el modelo deja de mostrar un grado de pertenencia convincente.

La cámara de la prueba fue la PlayStation Eye, la cual, como se menciona en el capítulo anterior, ya tiene calculado su foco siguiendo el modelo de la cámara oscura. Con esta cámara se calcularon las rotaciones y distancias siguiendo el mismo modelo para la creación de las imágenes sintéticas. La cámara se fijó con un tripié al frente de las mesas marcadas a una altura de 12 cm sobre la mesa.

Para tener un ambiente más realista sé probo el modelo en tiempo real.

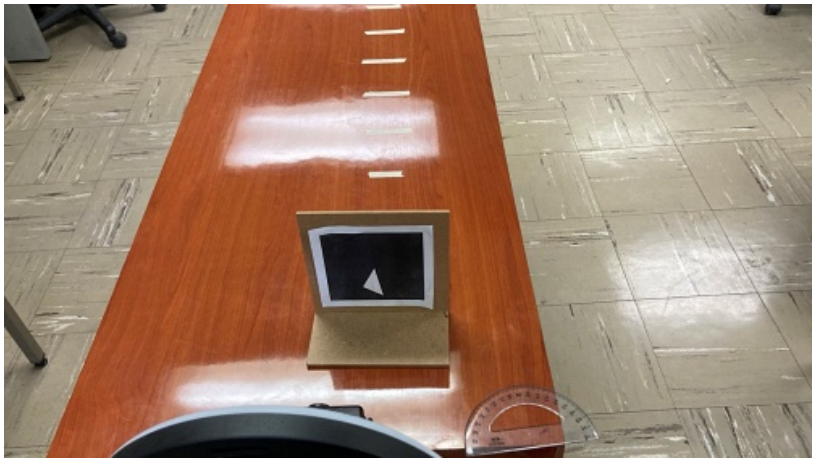


Figura 3.8: Mesa de 1.20 m marcada

Figura 3.9: Mesa marcada con cinta cada 10 cm

Figura 3.10: Tabla marcada con ángulos cada 10° de 0° a 180°

Figura 3.11: Probando Múltiples marcadores.

Como podemos observar en la figura 3.11, al utilizar múltiples marcadores se pueden detectar todos a diferentes distancias, con diferentes posiciones y diferentes ángulos de inclinación, así como con un menor grado de pertenencia los reflejos creados por éstos sobre la mesa.

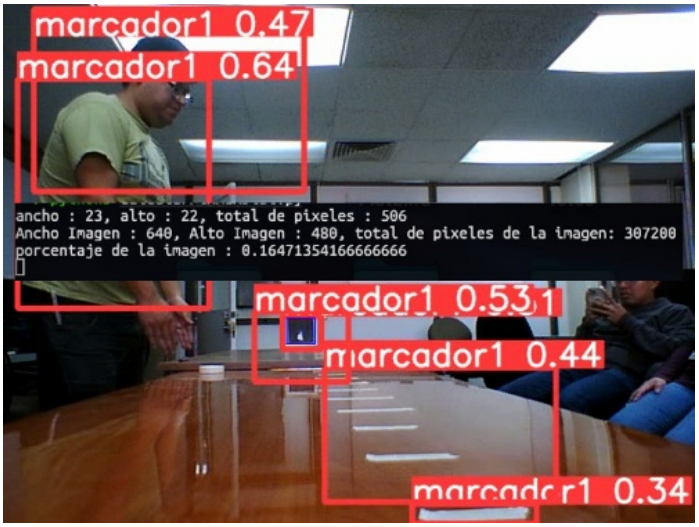


Figura 3.12: Cálculo del porcentaje de píxeles de falla del modelo.

En la figura 3.12, se muestra el cálculo del porcentaje de píxeles necesario que hace fallar el modelo mediante un script que cuenta los píxeles y obtiene el porcentaje sobre el tamaño total de la imagen de entrada.

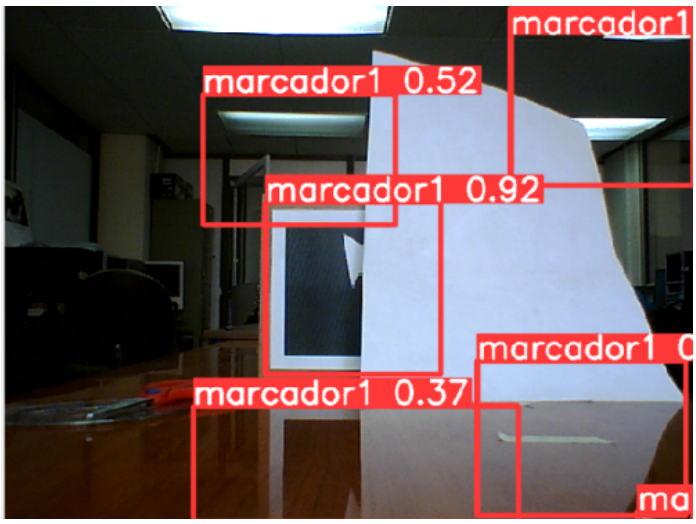


Figura 3.13: Cálculo de porcentaje de píxeles del marcador para reconocimiento.

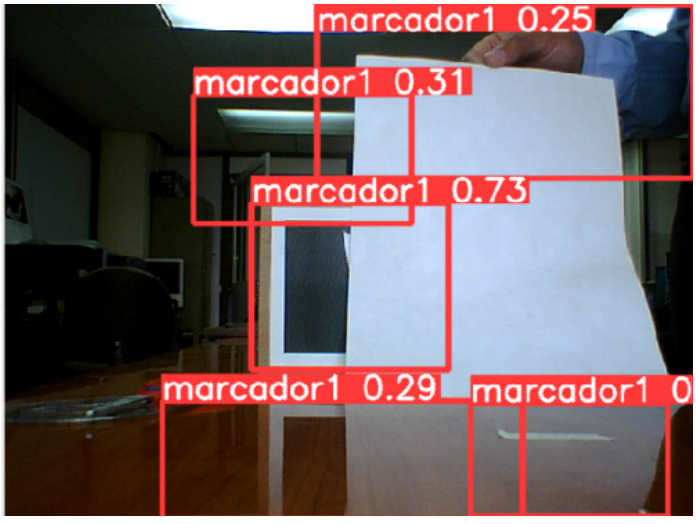


Figura 3.14: Cálculo de porcentaje de píxeles del marcador para falla.

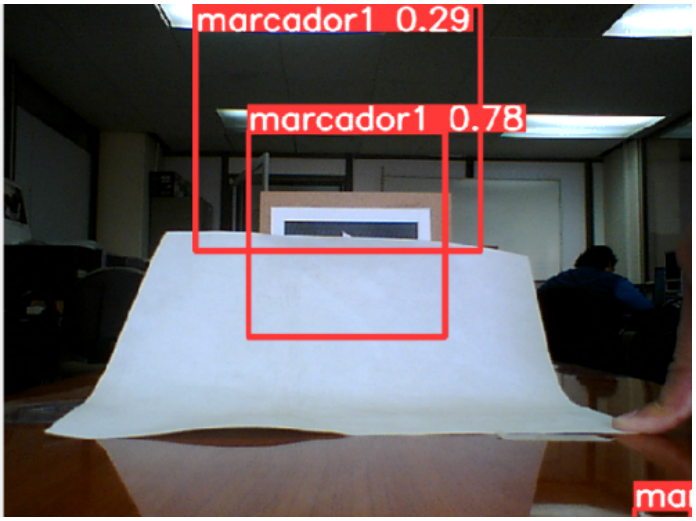


Figura 3.15: Cálculo de porcentaje de píxeles del marcador horizontal para falla.

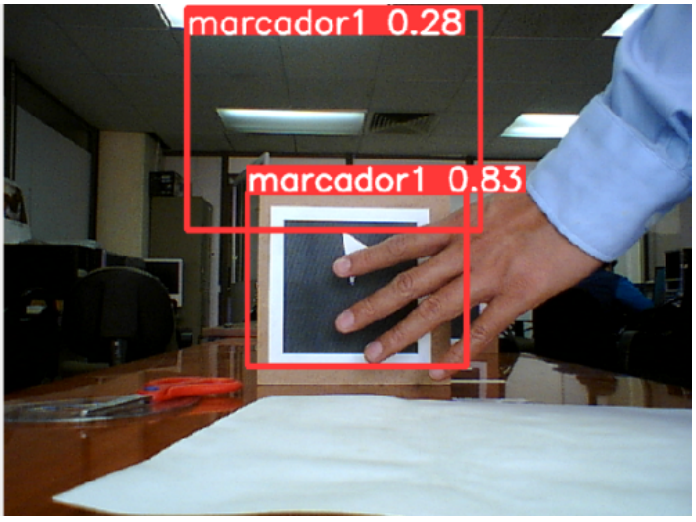


Figura 3.16: Prueba poniendo la mano encima del marcador.

En las figuras 3.13, 3.14 y 3.15 se muestran las pruebas realizadas para validar los límites del modelo entrenado al cubrir diferentes porcentajes del marcador tanto en horizontal como en vertical. En la figura 3.16 se muestra parcialmente cubierto con formas irregulares como lo son los dedos de una mano, para ver el poder de generalización al exponerlo a problemas para los que no fue preparado.

3.2.5. Experimentos con la Arquitectura YoloV5 Modificada

Para este punto, ya que hemos visto que la arquitectura YoloV5 completa funciona bastante bien para detectar un solo marcador, se planteó, según los objetivos de esta tesis, una reducción de las capas para mejorar el proceso de entrenamiento. El primer intento se realizó tomando en cuenta solo la capa totalmente conectada, es decir, la parte de clasificación. Que como se vio con anterioridad, esta capa consta de una entrada de todas las características de las etapas convolucionales a una reducción de 1280 neuronas en la primera capa que se conecta a otra capa que es la de salida que se calcula como $numero\_de\_clases + 1(la\_clase\_fondo) + 4(coordenadas\_necesarias\_para\_dibujar\_la\_caja)$. La capa totalmente conectada se encuentra directamente conectada a las últimas convoluciones de 80×80,40×40, 20×20. Cada una de ellas da un total de 255 filtros por cada salida, es decir, el número total de valores que en este punto serían:

$$\begin{aligned} t1 &= 80 \times 80 \times 255 = 1632000 \\ t2 &= 40 \times 40 \times 255 = 408000 \\ t3 &= 20 \times 20 \times 255 = 102000 \\ t1 + t2 + t3 &= 2142000 \end{aligned} \tag{3.3}$$

Todos estos valores pasan por una última convolución en el cual se ingresa el total de filtros o canales es decir $255 \times 3 = 765$ y sale una cantidad fija de 1280 al aplicar un filtro de 1×1 con pasos de 1 según el código proporcionado por Pytorch [1]. Esto para después ser agrupados con una operación de agrupación, llamado agrupación adaptativa promedio. Esta última operación aplica una eliminación (dropout) esto es una técnica que sirve para prevenir el sobreajuste dando una probabilidad a cada neurona de ser entrenada o no, durante el proceso de entrenamiento.

```
    class Classify(nn.Module):
    # YOLOv5 classification head, i.e. x(b,c1,20,20) to x(b,c2)
    def __init__(self,
                 c1,
                 c2,
```

```
            k=1,
            s=1,
            p=None,
            g=1,
            dropout_p=0.0):  # ch_in, ch_out, kernel,
            #stride, padding, groups, dropout probability
        # <<-- numero de canales = c1, numero de clases = c2
        super().__init__()
        c_ = 1280  # efficientnet_b0 size
        self.conv = Conv(c1, c_, k, s, autopad(k, p), g)
        self.pool = nn.AdaptiveAvgPool2d(1)  # to x(b,c_,1,1)
        self.drop = nn.Dropout(p=dropout_p, inplace=True)
        self.linear = nn.Linear(c_, c2)  # to x(b,c2)
```

Es aquí donde se pasan los 1280 mapas de características a una red neuronal del mismo tamaño y a la salida nos regresa al número de clases necesario ya con los grados de pertenencia.

En este nuevo intento se congelaron todos los valores de las etapas convolucionales y se reentrenó el modelo que con anterioridad ya funcionaba para detectar un solo marcador. Se esperaba que el extractor de características, es decir la capa convolucional, generalizara las características de un marcador como para reentrenar solo la última capa de clasificación previamente mencionada. De tal forma se esperaba que los pesos de los filtros, o kernerls fueran lo suficientemente robustos como para proporcionar un buen extractor de características para el problema de detección de marcadores. También se cambió el tipo de marcador y se incrementó a dos marcadores, creando así un conjunto de datos de 2000 imágenes por clase para entrenamiento y 200 por clase para validación.

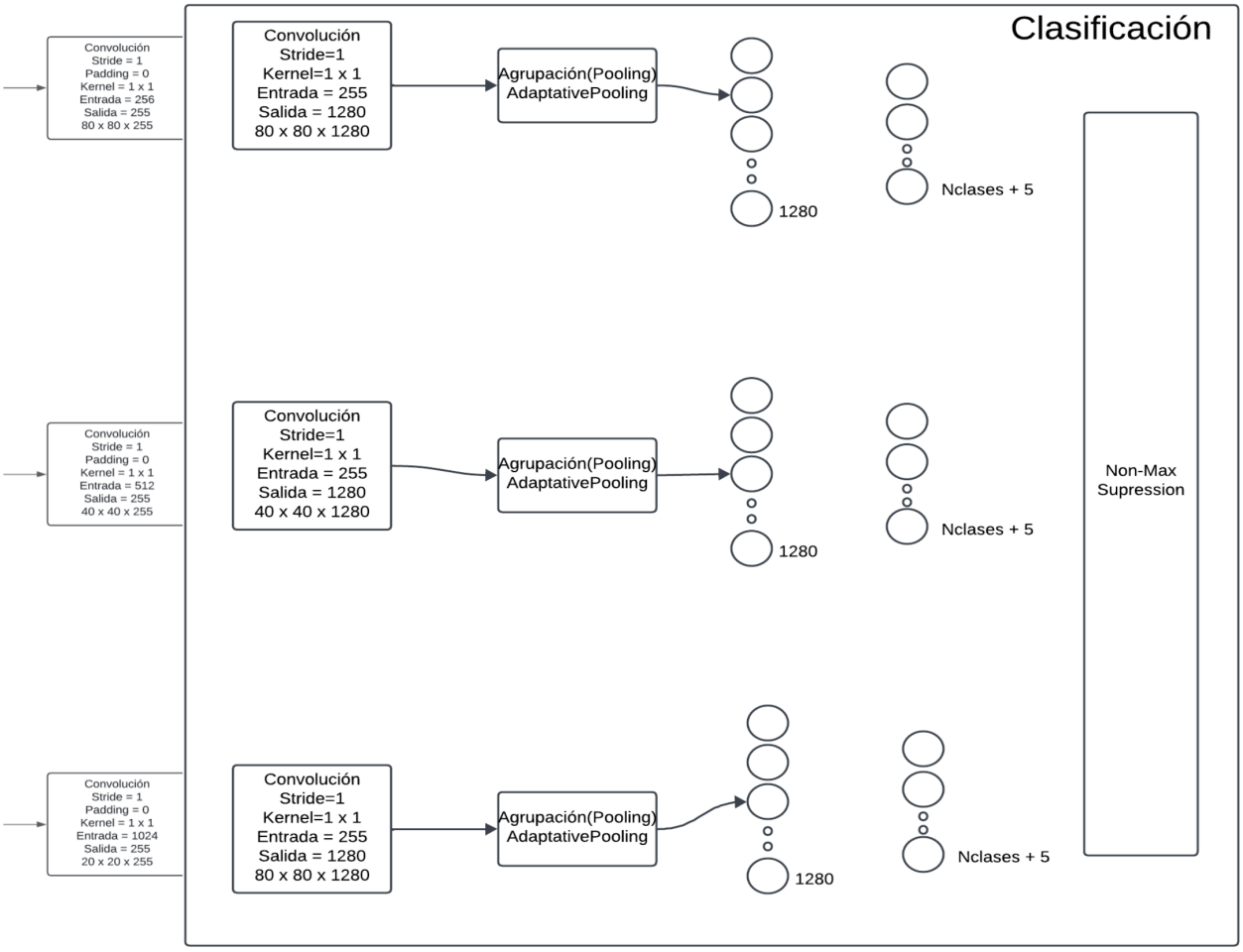
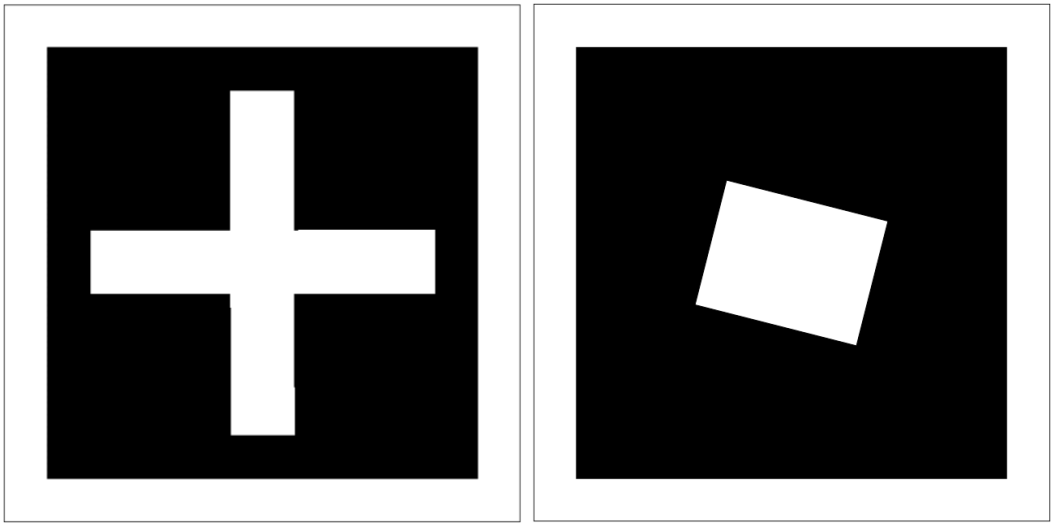


Figura 3.17: Diagrama que muestra cómo la función "classify" funciona en la última capa para hacer las detecciones, para por último utilizar el algoritmo de supresión del no máximo.



(a) Marcador de cruz (b) Marcador de rectángulo

Figura 3.18: Marcadores utilizados para crear el conjunto de entrenamiento de dos clases.

Desafortunadamente, no se obtuvieron buenos resultados, a pesar de que el tiempo de entrenamiento fue significativamente menor.

Como se puede observar en la figura 3.19, la matriz de confusión nos muestra que no es capaz de distinguir entre los marcadores y tampoco puede diferenciar entre el fondo y los marcadores.

3.2.6. Segundo intento red YoloV5 congelamiento de capas

Debido al comportamiento observado y haciendo un estudio más profundo de cómo funciona la red convolucional YoloV5 y las técnicas previamente expuestas en el marco teórico, se observó que la técnica GhostNet junto con la red piramidal de características obtiene características más abstractas de los objetos. Esto ocurre al momento de iniciar la expansión del mapa de características. Por lo tanto, se planteó como segunda propuesta para congelar el modelo, solo congelar todo hasta la parte de la agrupación espacial piramidal (SPPF) que es la parte en donde empieza la expansión de dimensiones del mapa de características, como se observa en la figura3.20.

Con esta técnica se consiguió clasificar correctamente ambos marcadores, de igual forma cómo ocurrió con el primer entrenamiento, donde se entrenó para un solo marcador. Mostrando que en este caso, el congelamiento de los parámetros no afecto negativamente la clasificación, cómo se refleja en la matriz de confusión.

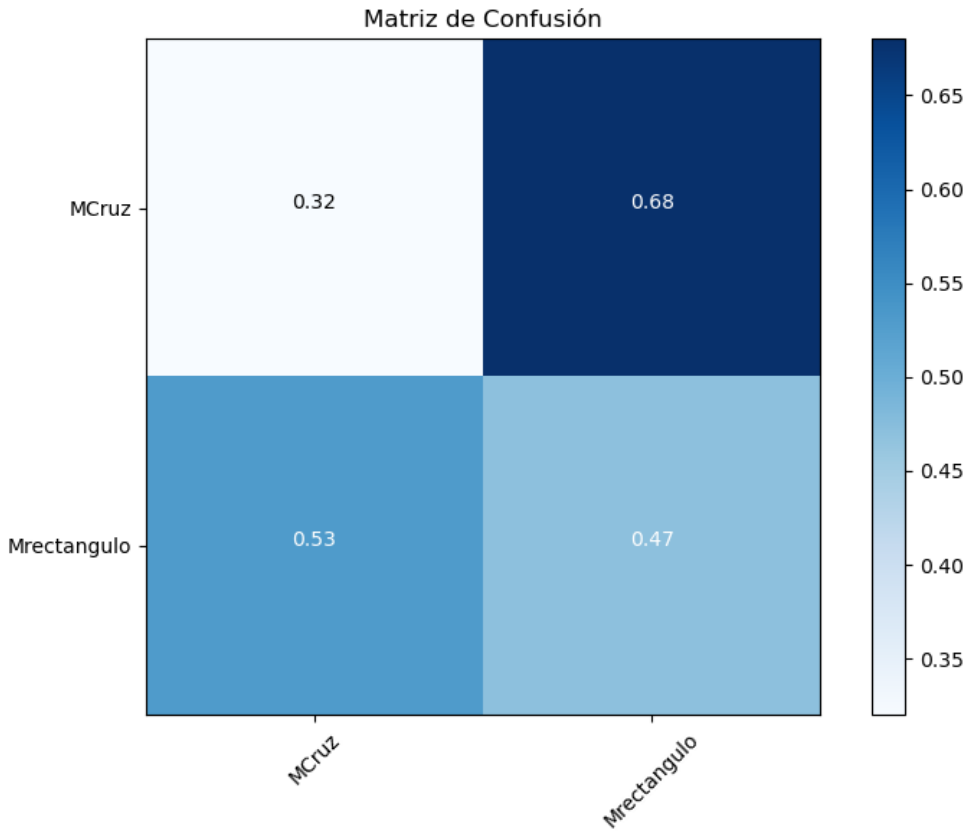


Figura 3.19: Matriz de confusión del re-entrenamiento congelando todas las convoluciones, dejando solo la capa totalmente conectada.

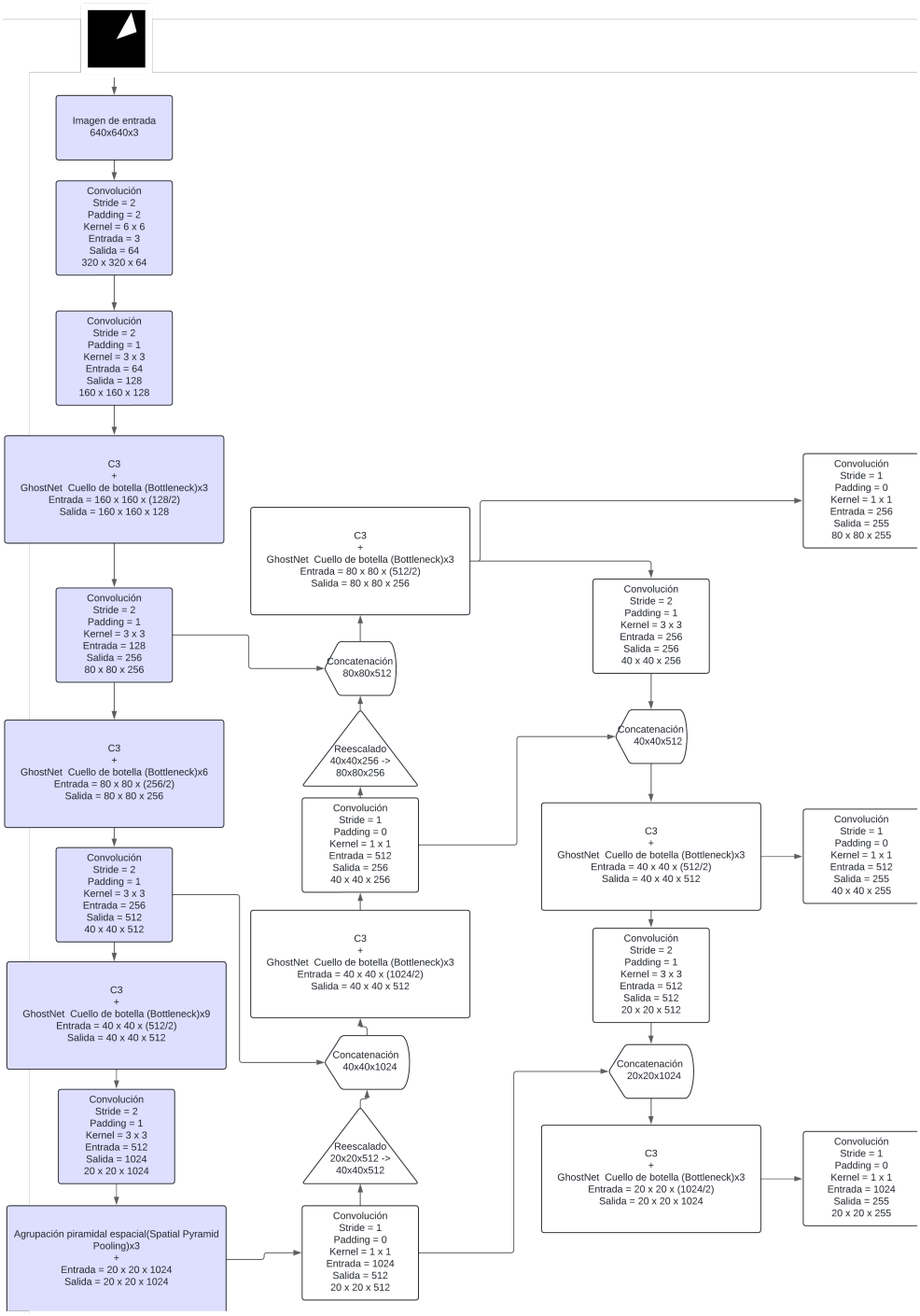


Figura 3.20: Red convolucional YoloV5 con parámetros congelados hasta la capa de agrupación espacial piramidal (SPPF).

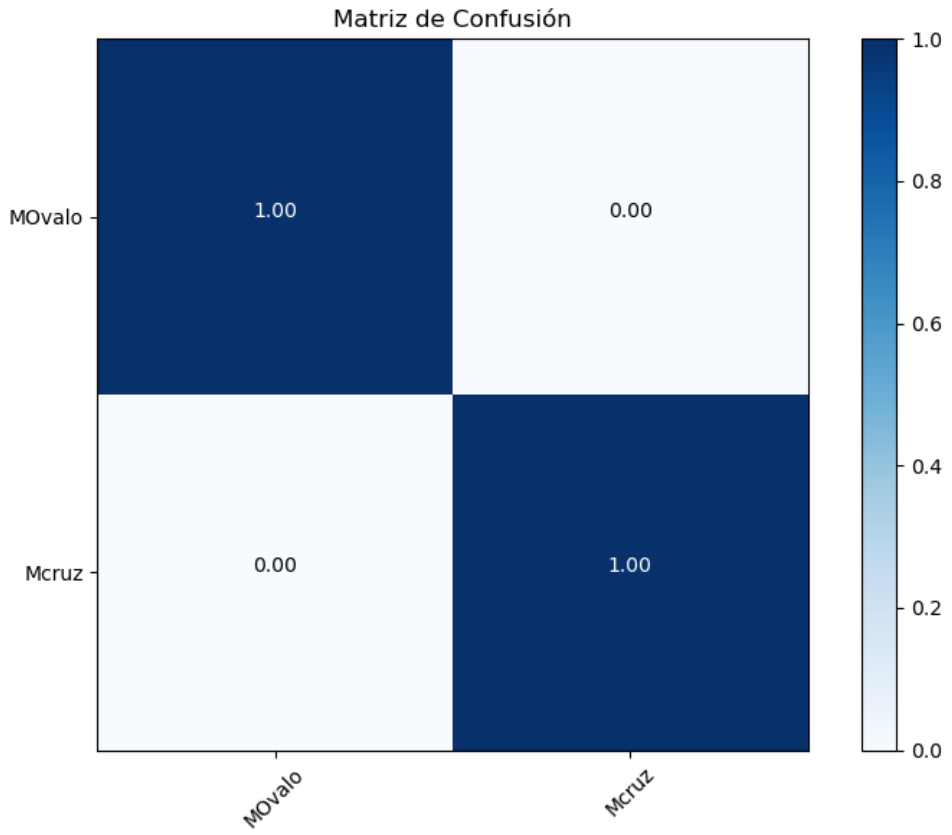


Figura 3.21: Matriz de confusión del reentrenamiento congelando convoluciones hasta la capa de agrupación espacial piramidal (SPPF).

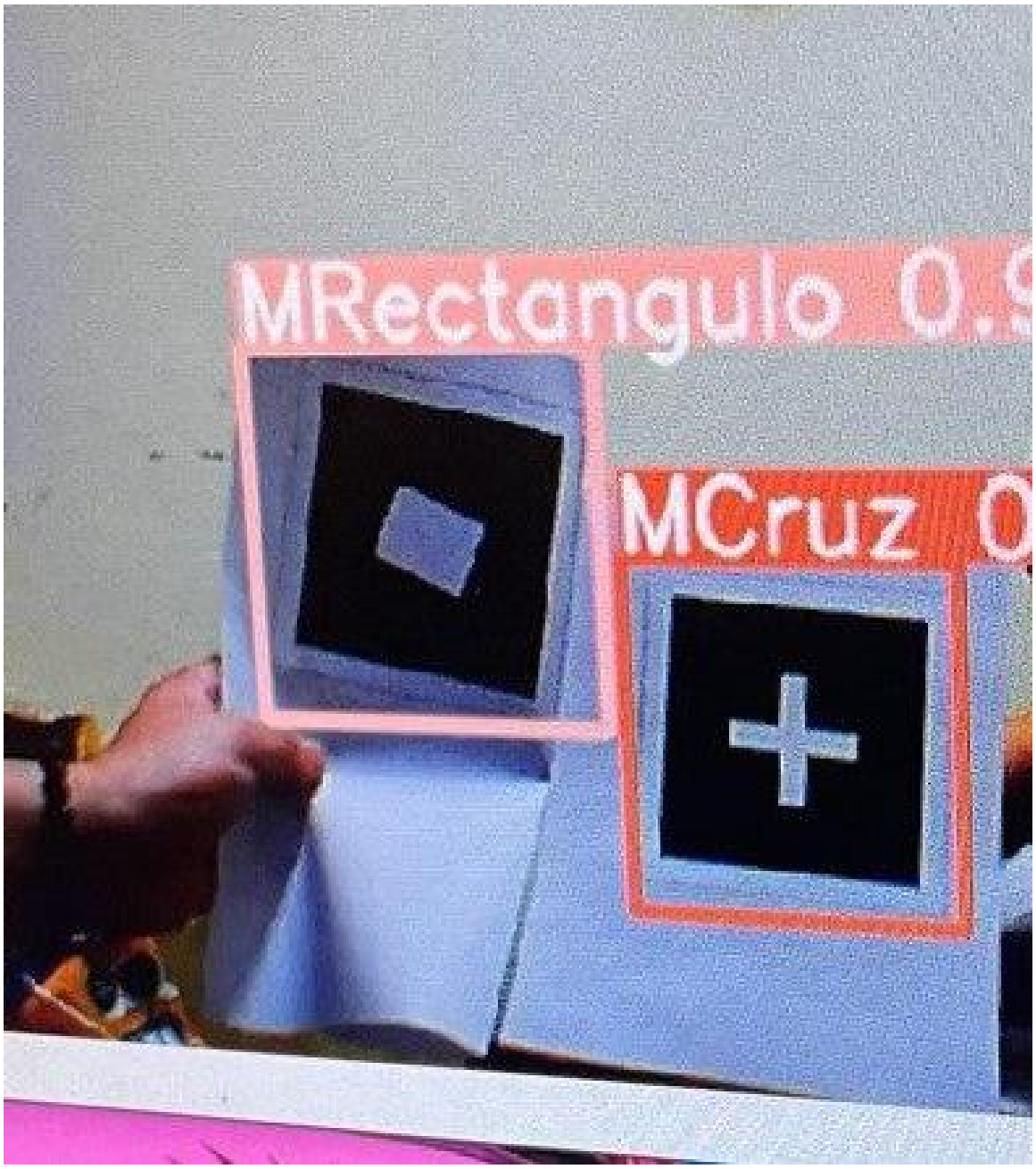
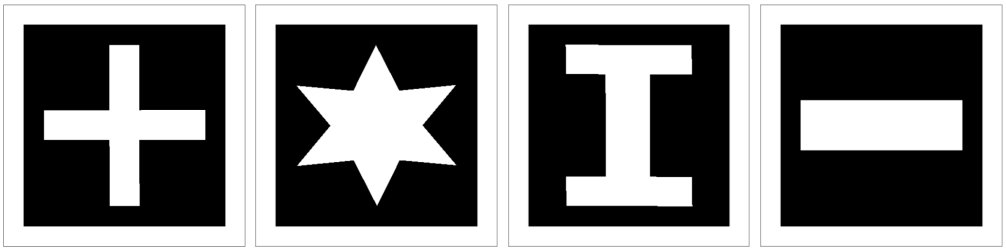


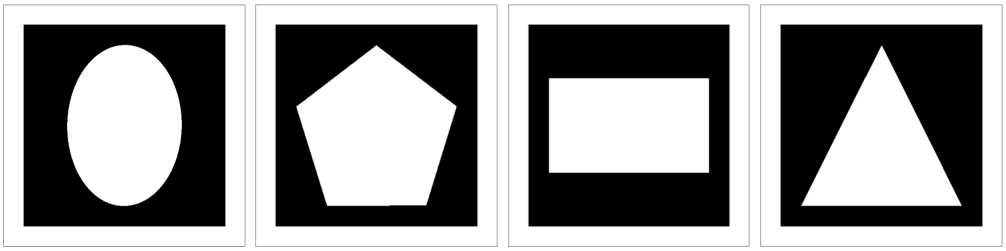
Figura 3.22: Pruebas con cámara en tiempo real de detección de dos marcadores con el modelo YoloV5 preentrenado y congelando capas hasta la capa de agrupación espacial piramidal (SPPF)

3.2.7. Problema más complejo usando la técnica de transferencia de aprendizaje para reentrenamiento y congelamiento de parámetros hasta la capa de agrupación espacial piramidal (SPPF)

Evidentemente, se tuvo éxito para detectar dos marcadores usando la técnica de transferencia de conocimiento que utiliza el congelamiento de parámetros de modelos preentrenados. Para corroborar que dicha técnica funcionaría para un problema más complejo, se decidió añadir más marcadores. Es decir, se añadieron más clases, pasando de dos clases a ocho, creando ahora un conjunto de datos de de 2000 imágenes por clase para entrenamiento y 200 imágenes por clase para validación. Esto da un total de 16000 imágenes de entrenamiento y 1600 de validación con los marcadores. Se partió del modelo preentrenado y se entrenó ahora para los ocho marcadores, congelando los parámetros de igual forma hasta la etapa de agrupación espacial piramidal por un total de 300 épocas. Esto nos dio un tiempo de entrenamiento total para las 300 épocas de 34 horas 7 minutos. También se dejaron los parámetros del tamaño del bloque (batch) a 16 sobre las, 16000 imágenes de entrenamiento.



(a) Marcador de cruz (b) Marcador de estrella (c) Marcador de forma I (d) Marcador de linea



(e) Marcador de ovalo (f) Marcador de pentagono (g) Marcador de rectangulo (h) Marcador de triángulo

Figura 3.23: Marcadores utilizados para crear el conjunto de entrenamiento de ocho clases.

Como podemos ver en la figura 3.25, Esta técnica mantuvo su efectividad frente a un problema mucho más complejo, como lo es la clasificación y detección de ocho diferentes objetos. Como se muestra en la matriz de confusión, el 100 % de las imágenes con marcadores de validación fueron clasificadas correctamente.

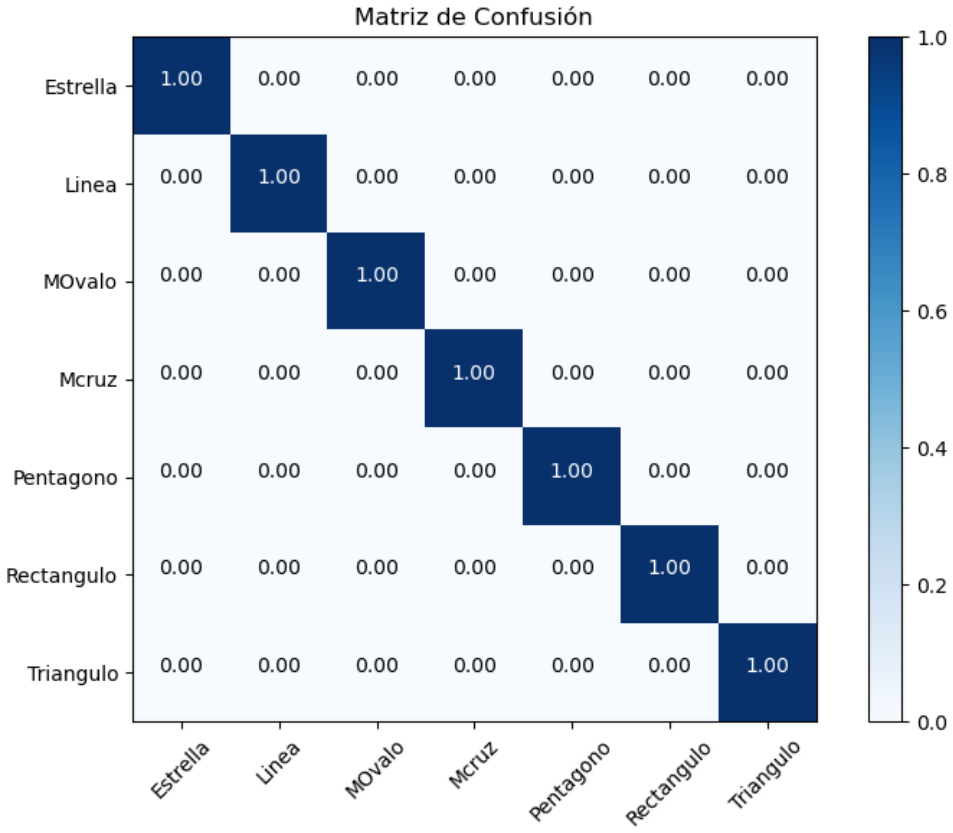


Figura 3.24: Matriz de confusión del reentrenamiento con ocho marcadores congelando convoluciones hasta la capa de agrupación espacial piramidal (SPPF).

Al probar la detección en tiempo real, funcionó bastante bien de igual forma, como se observa en la figura 3.25, donde se puede detectar al mismo tiempo las ocho clases en una misma imagen de entrada.

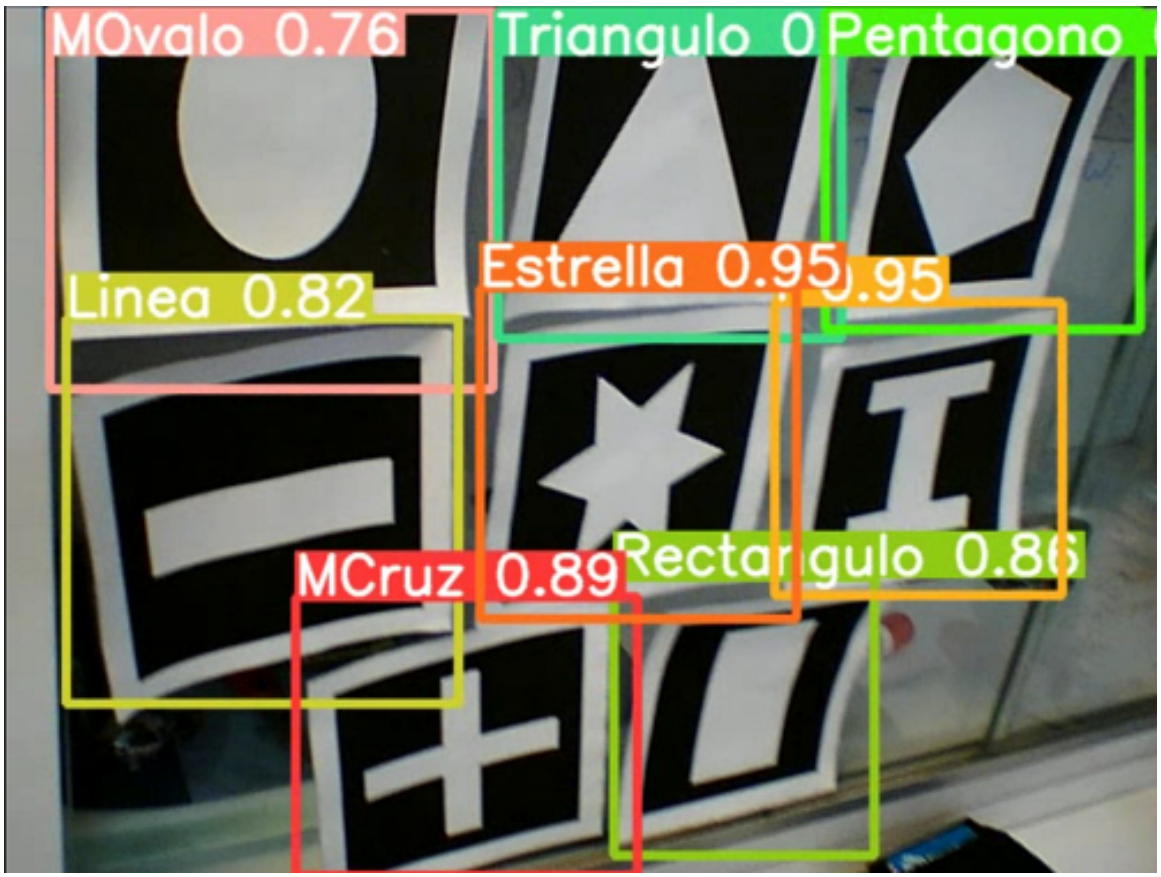


Figura 3.25: Pruebas de detección en tiempo real con cámara de modelo reentrenado con ocho clases hasta la capa de agrupación espacial piramidal (SPPF).

3.3. Red SSD (Single Shot Multibox Detector)

Como se pudo observar en el capítulo anterior, mediante el uso de la red YoloV5 aplicando correctamente la técnica de transferencia de aprendizaje, se puede llegar a obtener buenos resultados. Como se dijo en un principio, se decidió también revisar el rendimiento de la red SSD

Lo primero que se hizo, de igual forma a como se realizó con la red YoloV5, fue encontrar el modelo preentrenado de la red utilizando la biblioteca de Pytorch [16]. Este, a su vez, contiene en la documentación la forma sencilla de utilizar el modelo

preentrenado, así como el link a su GitHub donde contiene el código de la implementación realizada para el entrenamiento de dicho modelo.

Siguiendo la lógica con la que funciona la biblioteca de PyTorch al descargar el modelo preentrenado con su función "load", realmente se descarga la implementación y todo lo que conlleva del GitHub correspondiente a cada proyecto. Los modelos preentrenados y sus implementaciones así como diferentes utilidades tales como funciones de activación y operaciones específicas como la supresión de no máximos y demás operaciones que necesitan de las diferentes arquitecturas, la biblioteca de Pytorch los almacena en el siguiente directorio: **/.cache/torch/hub**. Este directorio es fundamental si se quiere comprender más a fondo cómo funcionan las implementaciones con las que fueron entrenados los modelos junto a su respectiva documentación. Como podemos observar, al hacer uso de las implementaciones YoloV5 y SSD tenemos las dos carpetas que corresponden al código implementado por la biblioteca en dicha ruta de cada uno de los modelos, respectivamente.

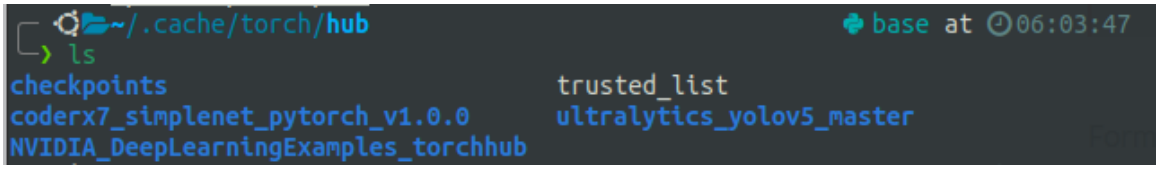


Figura 3.26: Directorio de caché de pytorch que contiene las implementaciones que se usaron para los modelos preentrenados.

En la figura 3.26 se muestra el directorio raíz del cache en donde se almacenan los modelos preentrenados así como las implementaciones que se usaron. En este caso NVIDIA_DeepLearningExamples_torchhub corresponde a varios modelos implementados por NVIDIA. Entre ellos se encuentra la red SSD, mientras que ultralytics_yolov5_master pertenece únicamente a la red YoloV5. Lo primero que se nota de acuerdo a la documentación de la SSD proporcionada por Pytorch [16], es que el modelo está diseñado para recibir un tensor compuesto por la serie de imágenes en las que se realizarán las detecciones y éste devuelve un tensor con el número total de las detecciones. En este caso es un total de 8732 de cuadros delimitadores por imagen y son éstos los que después son evaluados y discriminados por el algoritmo de supresión del no máximo anteriormente descrito. Eso quiere decir que va a variar la forma en la que se implementara el detector en tiempo real que utilice este modelo. Esto es porque a diferencia del modelo preentrenado de la red YoloV5 que utiliza una imagen de entrada y una salida con las detecciones, este utiliza un arreglo de imágenes de entrada y a este arreglo es al que se le hacen las detecciones por tanto, es necesario adaptar este funcionamiento al entorno de detección en tiempo real. Por otra parte, el modelo de igual forma está preentrenado con el conjunto de datos COCO [9] así que por esa parte se establece un punto de partida similar a la de la red YoloV5.

No obstante, el formato utilizado para dibujar las cajas delimitadoras que utiliza la biblioteca OpenCV es el de punto máximo y punto mínimo, en lugar del establecido para utilizar en esta tesis que es el de centro largo y ancho. Por lo tanto, se trabajó en un conversor o traductor de coordenadas. La transformación de coordenadas se hizo de forma sencilla tomando en cuenta que la imagen de entrada es de un tamaño fijo de 300×300. De igual manera que en la red YoloV5, los valores entran normalizados del 0 al 1 en lugar de los valores completos de cada canal de 32 bit correspondientes a los valores del 0 al 255. De la misma forma, las coordenadas pasan de estar en píxeles a valores del 0 al 1 representando el porcentaje de la imagen. Esto es importante para entender el preproceso que se hace antes de implementar la detección en tiempo real.

$$
\begin{aligned}
&\text{Revertimos valores del 0 al 1 a las dimensiones reales} \\
&centro = X \times TamImagX, Y \times TamImagY \\
&\text{Calculamos el ancho y el alto de la caja relimitadora} \\
&ancho = Xb \times TamImagX \\
&alto = Yb \times TamImagY \\
&\text{Se transforma el formato de la caja delimitadora de} \\
&\text{cento alto y ancho a punto minimo, punto maximo} \\
&Ymin = centro[y] - alto/2 \\
&Xmin = centro[x] - ancho/2 \\
&Ymax = centro[y] + alto/2 \\
&Xmax = centro[x] + ancho/2
\end{aligned}
\tag{3.4}
$$

De esta forma se consiguió establecer el sistema de coordenadas de la salida del modelo de la red neuronal convolucional al formato entendido por OpenCV que es necesario para poder trabajar con el modelo en imágenes en tiempo real. Esta forma de traducir las coordenadas se probó en el script incluido en la documentación **verImg.py**. Éste toma la imagen seleccionada en la variable nImg del conjunto de entrenamiento y aplica dicho algoritmo para posteriormente dibujar el recuadro con los datos obtenidos por el algoritmo.

Una vez entendido esto, se procedió a crear un conjunto de datos de entrenamiento con las dimensiones aceptadas por la red SSD que son de 300×300 de un solo marcador, siguiendo con el mismo procedimiento realizado por el entrenamiento de la red YoloV5. De igual forma, se crearon primeramente 2000 imágenes de entrenamiento y 200 de validación para entrenar la red SSD.
Pese a que la arquitectura SSD de la biblioteca de Pytorch fue entrenado prácticamente bajo las mismas condiciones, es decir mismo formato de números normalizados y mismo conjunto de entrenamiento COCO [9], al realizar el reentrenamiento a partir de este modelo preentrenado no se obtuvieron buenos resultados.

Primeramente, se decidió utilizar como función de pérdida el optimizador, Adam, ya que en problemas de clasificación mostraba ser bastante eficiente en otras prácticas realizadas. No obstante, en este caso fue deficiente. Después sé probo con el

optimizador mediante descenso de gradiente estocástico (SGD) el cual mostró ser un poco más eficiente. Sin embargo, este no lograba alcanzar la exactitud que mostraba el modelo YoloV5.

Gracias a lo observado se determinó que lo más probable es que este modelo, al ser relativamente más simple que el modelo YoloV5, necesita más datos de entrenamiento para lograr mejores resultados. Se decidió duplicar el número de datos de entrenamiento, quedando con 4000 imágenes de entrenamiento. Por último, se decidió duplicar de nuevo las imágenes de entrenamiento, pasando de 4000 a 8000. Con todas estas pruebas se obtuvieron los siguientes resultados.

Tabla 3.4: Resultados de los entrenamientos más relevantes de la red SSD, con el modelo preentrenado con el conjunto de datos COCO [9].

Número de imágenes	Optimizador	Exactitud	Tiempo de entrenamiento
2000	ADAM	0.62	**3 horas 22 min**
4000	SGD	0.96	**5 horas 34 min**
8000	SGD	0.98	**11 horas 15 minutos**

Este último una mejora significativa con las imágenes de validación, sin embargo, aún mostraba algunas deficiencias con la cámara en tiempo real. Hubieron problemas de varios solapamientos de las cajas delimitadoras, seguramente debido a que depende demasiado del algoritmo de supresión del no máximo. En otro tipo de problemas, tienen que ver más con la calidad de las convoluciones es decir la calidad del extractor de características, ya que aunque era capaz de distinguir el marcador real en la imagen en tiempo real así como su ubicación, pero mostraba muchos falsos positivos en posiciones con cierto patrón de oscuro con blanco entendible por el tipo de marcador, sin embargo, esto solo muestra la debilidad del extractor de características a la misma cantidad de épocas de entrenamiento frente a la red YoloV5.

Como se puede observar en la figura 3.27, el mejor modelo que fue entrenado con el optimizador SGD y las 8000 imágenes de entrenamiento, fue capaz de detectar las características generales del marcador sobre la imagen, detectando correctamente el marcador. Sin embargo, en la figura 3.28 y 3.29 podemos ver cómo confunde objetos con las mismas características sobre la imagen muy fácilmente dando falsas detecciones así como cajas delimitadoras traslapadas. Errores que con menor número de imágenes de entrenamiento el modelo YoloV5 no sufría.

Figura 3.27: Imagen que ilustra el comportamiento en tiempo real del mejor modelo entrenado con 8000 imágenes, usando el optimizador SGD, detectando el marcador con el que fue entrenado

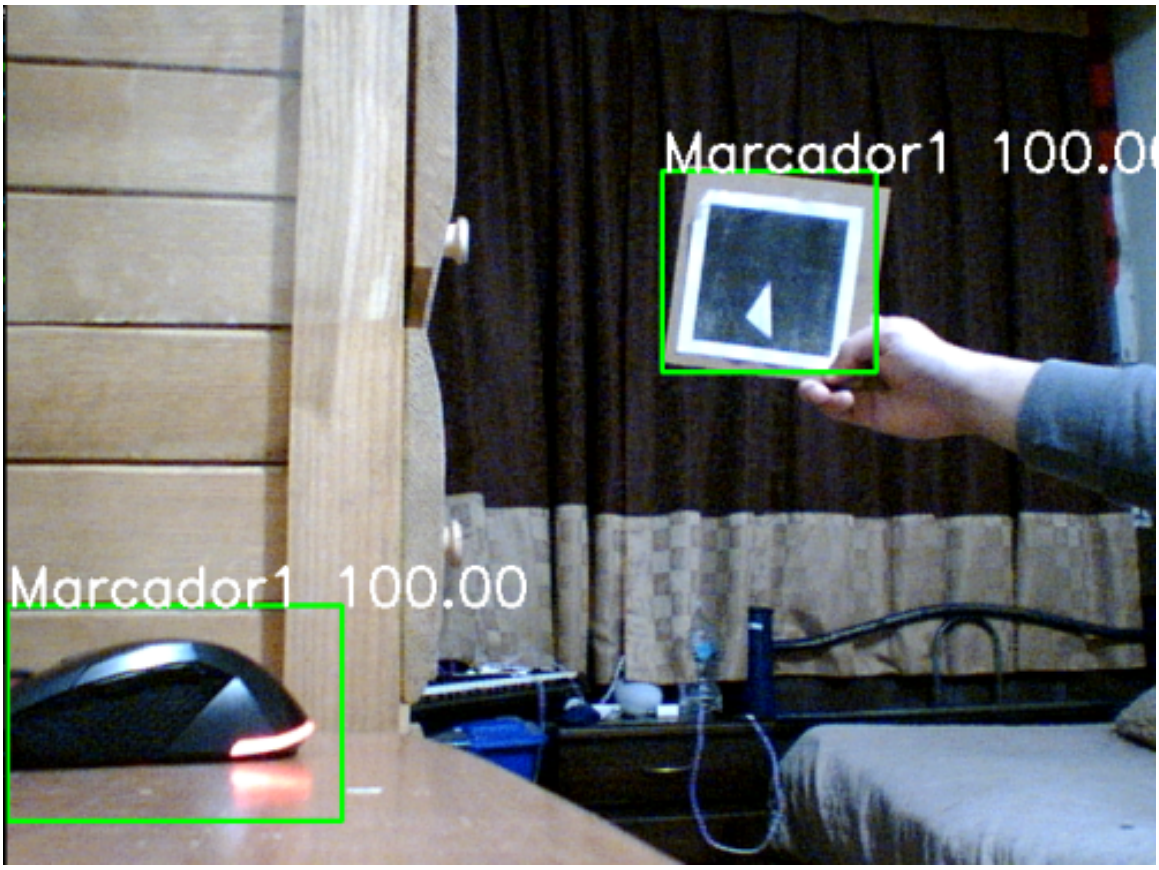


Figura 3.28: Imagen que ilustra el comportamiento en tiempo real del mejor modelo entrenado con 8000 imágenes, usando el optimizador SGD, detectando el marcador con el que fue entrenado y falso positivo que muestra el modelo en el mundo real.

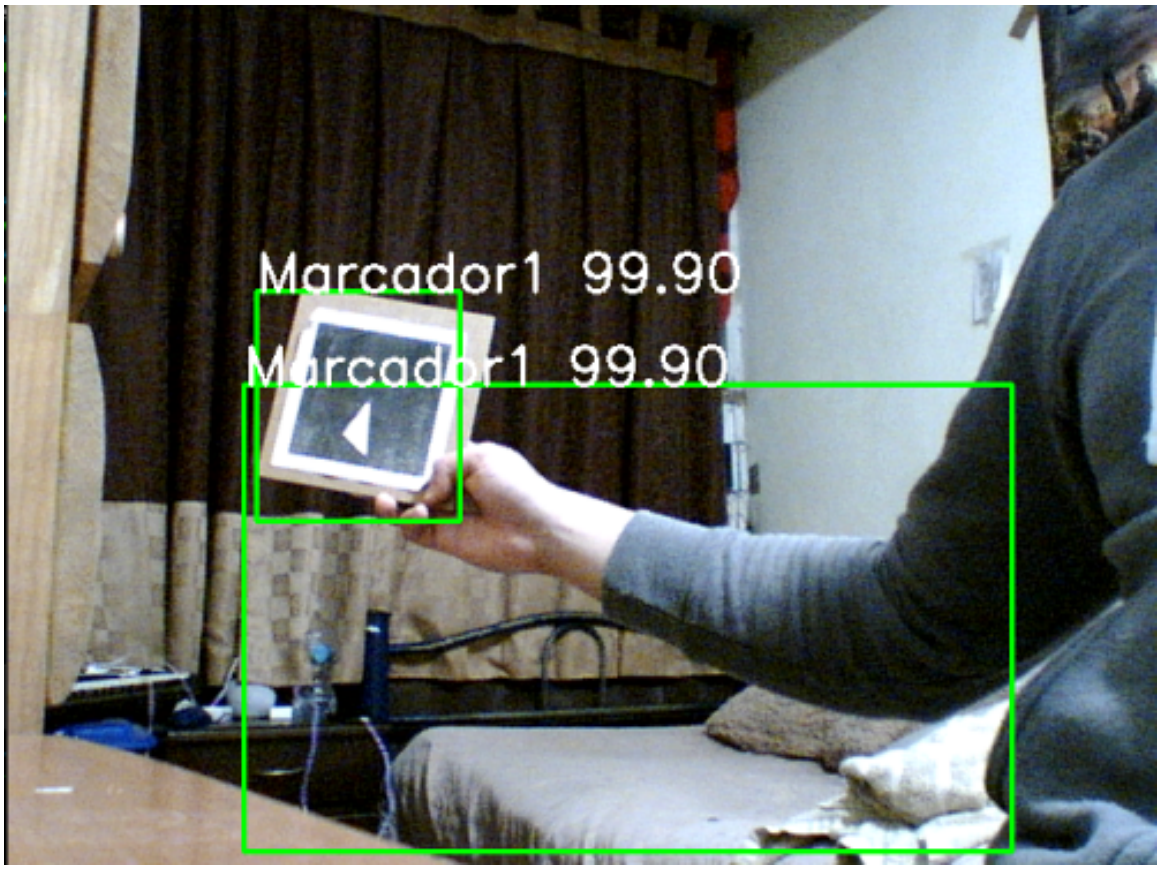


Figura 3.29: Imagen que ilustra el comportamiento en tiempo real del mejor modelo entrenado con 8000 imágenes, usando el optimizador SGD, detectando el marcador con el que fue entrenado y mostrando él traslapó de cajas delimitadoras.

Debido a la limitante del mismo modelo al compararse con el modelo YoloV5 mucho más robusto, se decidió dejar de trabajar sobre éste y quedarnos con el mejor entrenamiento que nos dio la red SSD debido a que cada vez los tiempos de entrenamiento crecían mucho más al igual que el número de imágenes sintéticas requeridas para entrenar el modelo.

Capítulo 4

Resultados

Como pudimos ver en el capítulo anterior de desarrollo, fue con la arquitectura YoloV5 que se obtuvieron mejores resultados en el capítulo de desarrollo. Sin embargo, en este capítulo daremos un vistazo más en profundidad a las estadísticas de entrenamiento, exactitud, así como tiempos de entrenamiento y rangos de detección de los diferentes modelos y variantes aplicadas a los diferentes grados de dificultad desarrollados es decir una clase, dos clases y ocho clases, respectivamente.

4.1. YoloV5

4.1.1. Un marcador

En el primer reentrenamiento del modelo total, con los pesos iniciales de modelo preentrenado con el conjunto de datos COCO se obtuvo la figura 4.1. Esta figura muestra una gráfica de exactitud para cada una de las 300 épocas, teniendo en cuenta que nos quedamos siempre con el mejor modelo.

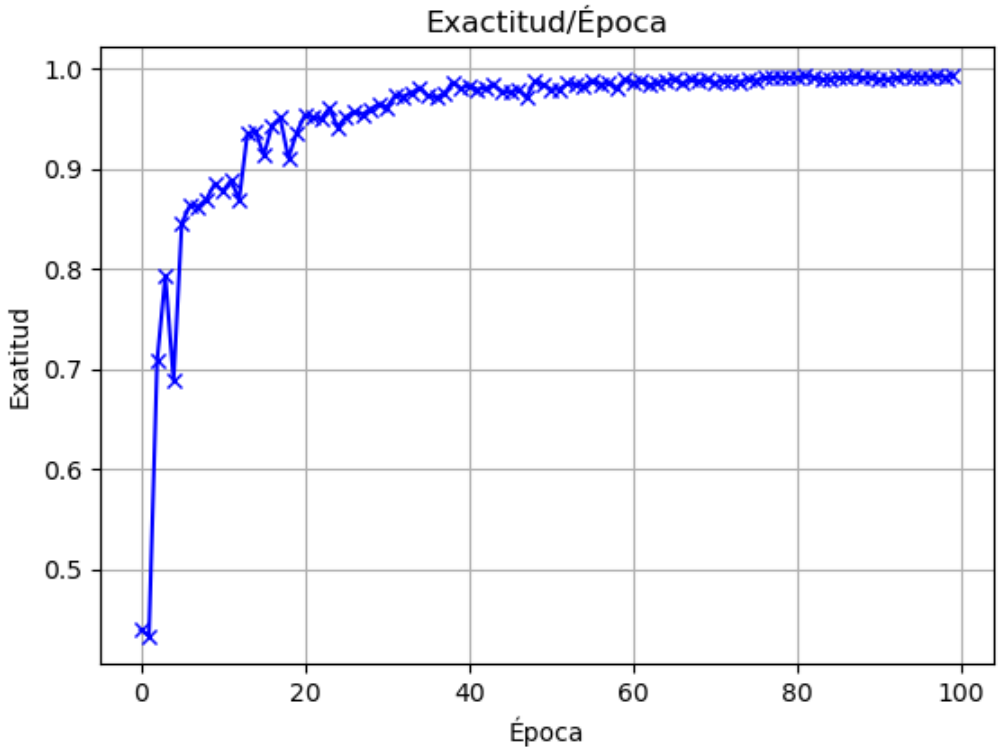


Figura 4.1: Gráfico de exactitud para cada época de entrenamiento, reentrenando para un solo marcador, con el conjunto de datos COCO.

También se midieron los ángulos de visión como se vio en el desarrollo, se realizó el experimento arrojando los siguientes resultados.

Tabla 4.1: Tabla de resultados del experimento propuesto, para modelo reentrenado para detectar un solo marcador preentrenado con el conjunto de datos COCO.

Distancia efectiva de reconocimiento	2.20 m
Número de píxeles necesarios para reconocimiento	Ventana de $23 \times 22 = 506$ píxeles
Porcentaje de marcador necesario para reconocimiento	70 %(10 % triángulo visible)
Ángulo efectivo de reconocimiento	75° a 80°
Reconocimiento de múltiples objetivos (Marcador)	Si
Tiempo de entrenamiento	**4hr 51min**

4.1.2. Dos marcadores

En este caso la exactitud mejoró hasta el punto en que en menos de 50 épocas ya conseguía una exactitud de 0.995 es por eso que se decidió realizar la gráfica con los

ejes normales y otra con una escala sobre el eje x, en este caso, el eje de las épocas.

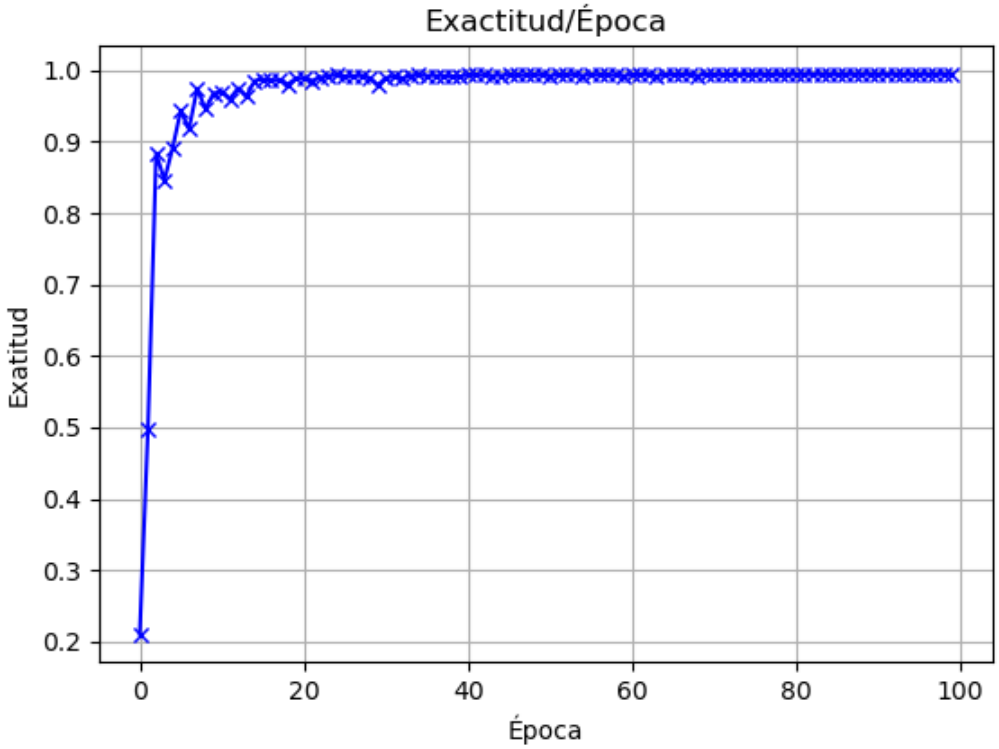


Figura 4.2: Gráfico de exactitud para cada época del modelo reentrenado para dos clases con el modelo preentrenado para detectar una sola clase y congelando hasta la capa de agrupación espacial piramidal (SPPF).

4.1.3. Ocho marcadores

De igual forma, como en el caso de detectar dos marcadores distintos, utilizamos haciendo la técnica de transferencia de conocimiento, congelando los parámetros hasta la capa de agrupación espacial piramidal (SPPF). Esto mejoró considerablemente los resultados, produciéndose una gráfica de exactitud parecida al problema de dos marcadores, habiendo encontrado el mejor modelo desde la época 14 aproximadamente con una exactitud de 0.991.

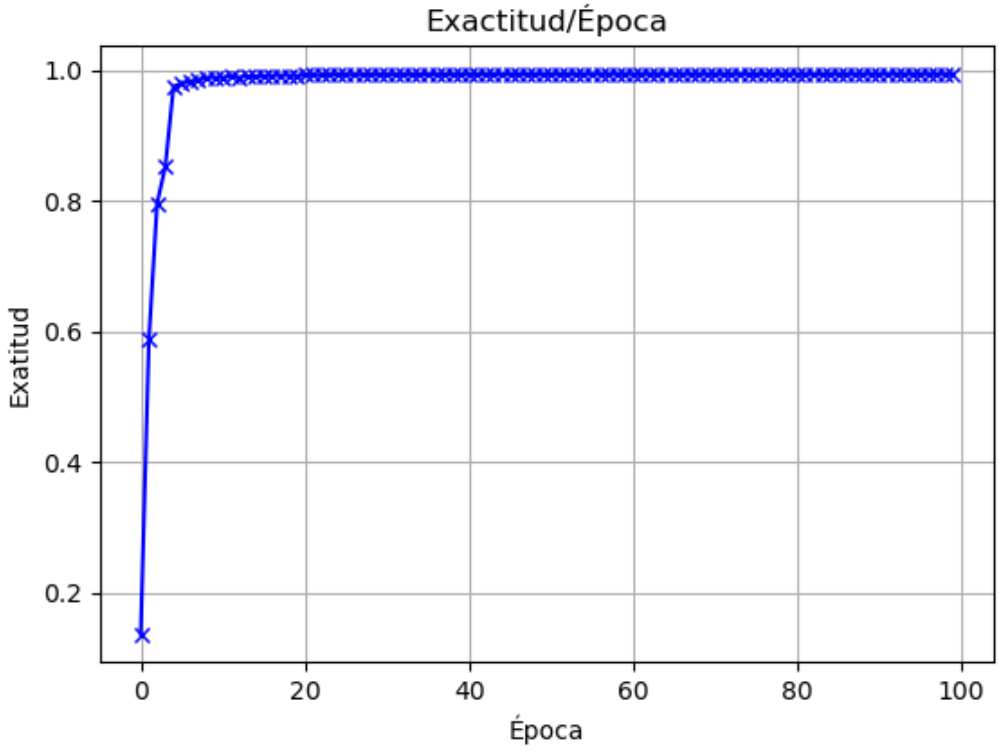


Figura 4.3: Gráficos de exactitud para cada época de modelo reentrenado para ocho clases con el modelo preentrenado para detectar una sola clase y congelando hasta la capa de agrupación espacial piramidal (SPPF).

Tabla 4.2: Tabla de resultados del experimento propuesto, para el modelo reentrenado para detectar un solo marcador y preentrenado con el conjunto de datos COCO con un **tiempo de entrenamiento de 34 horas 7 minutos.**

Figura	Distancia efectiva	Tamaño de ventana	Num píxeles	Angulo efectivo
Triángulo	1.12 - 1.16m	66×64	4224	60° a 70°
Rectángulo	1.05 - 1.10m	66×65	4290	50° a 60°
Línea	1.06 - 1.16m	67×65	4355	50° a 60°
I	1.12 - 1.17m	66×63	4158	60° a 70°
Estrella	1.11 - 1.15m	64×66	4224	60° a 70°
Óvalo	1.15 - 1.2m	66×62	4092	60° a 70°
Cruz	1.11 - 1.17m	66×68	4488	60° a 70°
Pentágono	1.2 - 1.5m	66×64	4224	60° a 70°

Capítulo 5

Conclusiones

De forma general se pueden concluir varios puntos. Uno de ellos es que el uso de técnicas transferencia de aprendizaje agiliza los procesos de entrenamiento manteniendo exactitudes similares si se sabe desde qué capas hacerlas. En este caso se utilizó la técnica de congelamiento de parámetros para ciertas capas de modelos preentrenados. También en esta tesis se observó que el uso de la arquitectura de red convolucional Single Shot Multibox Detector(SSD), requiere de una cantidad enorme de datos de entrenamiento para igualar la exactitud que demostró YoloV5. De igual forma, su extractor de características es más débil. Sin embargo, para el tiempo en que se realizó la arquitectura SSD resulto contemporánea a la red YoloV2. Por lo tanto, la parte de las convoluciones o la extracción de características están basadas en la red VGG16, que es una red de clasificación que en su momento fue muy buena. No obstante, actualmente existen redes mucho más eficientes y robustas, como lo son Eesnet, Googlenet y Efficientnet, con las que podría experimentarse para mejorar esta parte.

Se pudo estudiar la ventaja que supone tener este tipo de redes preentrenadas para el reentrenamiento, ayudando así a la detección de objetos más complejos o incluso a una mayor cantidad de clases. Esto se puede observar en el caso de la arquitectura YoloV5 donde comenzamos con el problema más sencillo de detectar un solo tipo de marcador, hasta ampliar el problema a detectar ocho marcadores, congelando parte de la red, lo que produjo ahorros en el tiempo de entrenamiento. Gracias a tener este tipo de redes preentrenadas se facilita mucho el uso de estas redes convolucionales para detecciones más complejas, permitiendo así su uso con tiempos de entrenamiento más adecuados. Por lo tanto, se puede afirmar cada vez con más seguridad que las redes neuronales convoluciones se vuelven más accesibles a comparación de los algoritmos clásicos de visión por computadora y extractores de características convencionales como lo son los momentos de Hu o el histograma de gradientes orientados (HOG).

La innovación en el equipo de cómputo de hardware, en especial en el campo de las tarjetas gráficas GPU, está suponiendo una revolución en el uso de los modelos de redes neuronales, tanto redes neuronales profundas como convolucionales. Esto ha acelerado aún más los tiempos de entrenamiento, así como el uso de estos modelos

ejecutados en tiempo real en dichas tarjetas gráficas. Asimismo, la innovación de las diferentes técnicas en las convoluciones para extraer más características o extraer mejores características con menor costo computacional como lo son Ghostnet, SPP o Feature Pyramid Network que se aplicaron en conjunción en la red YoloV5 suponen un gran salto en comparación con sus versiones anteriores así como en comparación a la red SSD.

5.1. Trabajo futuro

Como trabajo futuro se ven viables los siguientes puntos:

- Modificar la parte convolucional de extracción de características de la red SSD probándola con diferentes redes convolucionales más actuales de clasificación.
- Revisar cuál es el límite de clases que puede clasificar y detectar la red YoloV5, con la técnica de transferencia de aprendizaje aplicada, congelando los parámetros hasta la capa SPPF.
- Combinar alguna de las técnicas de extracción de características de la red YoloV5 en la red SSD para ver si ésta mejora.
- Verificar si es posible ampliar la parte de clasificación (la parte de la red totalmente conectada) tanto de la red YoloV5 como de la SSD y disminuir el número de convoluciones manteniendo la exactitud pero disminuyendo el tamaño de las redes.
- Utilizar una cámara con mayores cuadros por segundo para la red YoloV5 y verificar la cantidad máxima que es capaz de procesar en un segundo en el mundo real.

Bibliografía

[1] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, Yonghye Kwon, Kalen Michael, Jiacong Fang, Zeng Yifu, Colin Wong, Diego Montes, et al. ultralytics/yolov5: v7. 0-yolov5 sota realtime instance segmentation. *Zenodo*, 2022.

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

[3] 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.

[4] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1577–1586, 2020.

[5] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[7] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6469–6477, 2017.

[8] Gonzalo Adán Chávez Fragoso. Reconocimiento de marcadores con redes profundas. Master's thesis, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Unidad Zacatenco Departamento de Computación, September 2020.

[9] Tsung-Yi Lin. cocodataset. https://cocodataset.org/, 2017. Accessed: 2021-1-22.

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[11] 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.

[12] 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.

[13] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-nms — improving object detection with one line of code. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5562–5570, 2017.

[14] Andres Cureño Ramirez. Localización y mapeo simultáneos con marcadores y un robot móvil. Master's thesis, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Unidad Zacatenco Departamento de Computación, September 2022.

[15] salaxieb. Perlin-Noise. https://pypi.org/project/perlin-noise/, 2022. Accessed: 2022-1-22.

[16] NVIDIA. Pytorch ssd. https://pytorch.org/hub/nvidia_deeplearningexamples_ssd/, 2017. Accessed: 2021-1-22.