UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Redes neuronales profundas para la detección de peces en ambientes subacuáticos

TESIS

Que para obtener el título de

Ingeniero mecatrónico

P R E S E N T A

Ernesto Jesus Castillo Varguez

DIRECTORA DE TESIS

Dra. Nidiyare Hevia Montiel

Ciudad Universitaria, Cd. Mx., 2021

II

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El monitoreo de peces tiene un papel muy importante en el trabajo de los biólogos de la conservación, ya que es la manera en la que ellos pueden evaluar la presencia o abundancia de estos animales en cierto ecosistema. Esta actividad se ha realizado con base en las artes de pesca tradicionales, estas en su mayoría implican la extracción temporal o permanente del espécimen, esto se traduce como un impacto directo en el equilibrio de su hábitat, ya sea al dañar el entorno o al espécimen.

Este trabajo es una continuación del desarrollado por el M.C. David Espinosa en su tesis "Variaciones temporales de la comunidad de peces en un humedal costero de Yucatán, mediante imágenes subacuáticas y técnicas tradicionales"(Espinosa, 2019). En su trabajo, Espinosa se da cuenta que el uso de videocámaras tiene un gran potencial y que los resultados obtenidos son comparables y menos invasivos con respecto a los obtenidos con los métodos de conteo por captura. Sin embargo, el registro de la aparición de peces en el vídeo sigue teniendo que realizarse de manera visual, lo que implica una gran inversión de tiempo en observar la grabación realizada y la subjetividad de quien realice esta tarea.

En este trabajo se pretende automatizar gran parte del esfuerzo que realizan los biólogos en el monitoreo de las especies de un ecosistema. Delegar esta tarea a un sistema automático disminuiría el tiempo y la carga de trabajo. También otorgaría una mayor objetividad, manteniendo las ventajas que el uso de videocámaras presenta al ser poco invasivo con el ecosistema.

El sistema debe ser capaz de resolver dos de los problemas de la visión por computadora. El primero es el de la detección de objetos, que en este

caso son peces, una vez que se ha detectado, se enfrenta al segundo desafío, se deben clasificar con base en su especie.

En la literatura se pueden encontrar sistemas parecidos de detección, los cuales se mencionaran posteriormente. La principal diferencia y dificultad de este trabajo es que al ser realizado en los petenes de Yucatán, el fondo de la imagen tiene una gran cantidad de vegetación y materia orgánica suspendida en la columna de agua, lo cual se traduce en mucho ruido en la imagen, por lo que no es tan fácil detectar y clasificar incluso para el ojo entrenado. Otra dificultad a considerar es que, el petén al no ser muy profundo, en muchas ocasiones el video captura el reflejo de peces en la superficie del agua.

La clasificación de patrones a partir de imágenes es de interés general en la sociedad actual, ya que cada segundo se produce una cantidad inmensurable de información. Por lo mencionado anteriormente, es necesario contar con algoritmos capaces de identificar y clasificar esta información de manera eficiente y precisa para la solución de problemas de diferentes áreas del conocimiento.

La ventaja del uso de redes neuronales convolucionales es que son capaces de aprender ciertas características[1] del objeto de interés, invariantes a rotación y traslación (Lowe, 1999), esto permitiría que la red sea capaz de discernir entre el fondo y el objeto aún cuando no es tan sencillo para una persona.

Debido al auge en el uso de redes neuronales profundas, hoy en día es posible acceder a bases de datos gigantescas con las que se pueden preentrenar modelos de redes neuronales. Con la información aprendida, se pueden obtener sistemas especializados con gran eficiencia sin tener que invertir mucho tiempo ni recursos de cómputo una vez que los sistemas se han entrenado.

En su trabajo, Espinosa obtuvo una gran cantidad de vídeos que fueron usados como la base de datos para el entrenamiento de la red propuesta en esta tesis. Estos vídeos no se encontraban etiquetados, por lo que se tuvo que extraer imágenes de estos videos y posteriormente etiquetarlos. Para el etiquetado se hizo uso del programa *VGG Image Annotator* (VIA), con este programa se crearon los archivos de anotaciones que posteriormente se usaron para crear los conjuntos de entrenamiento, validación y prueba. Debido a que las condiciones de detección son muy ruidosas, se creó un conjunto que contuviese imágenes "fáciles", es decir en condiciones ideales para la detección de los peces. A este conjunto se le llamó *easy test* y su

[1] *Features*

finalidad fue la de proveer una idea del desempeño del modelo si se eliminan algunas de las variables que dificultan su trabajo.

Para la creación del modelo se utilizó una implementación de la arquitectura Mask R-CNN (He, Gkioxari, Dollár & Girshick, 2017a), la cual es, al momento, la última iteración de una serie de arquitecturas especializadas en el reconocimiento de objetos. Mask R-CNN es capaz de cumplir con la tarea de segmentación de instancias. Al final del trabajo se contó con 3 modelos entrenados cada vez con una mayor cantidad de datos. Para el entrenamiento se hizo uso de las técnicas de aumento de datos y transferencia de aprendizaje. El entrenamiento de la red se realizó con ayuda del *cluster* del Laboratorio Universitario de Cómputo de Alto Rendimiento (LUCAR) del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas de la UNAM.

Para comparar el desempeño de la red se hizo uso de dos métricas, la precisión y la sensibilidad. Los resultados obtenidos son más que prometedores, obteniendo precisiones y sensibilidades alrededor del 85 % en el conjunto *easy test* y del 70 % en los conjuntos de prueba en los mejores casos.

Con este trabajo se muestra que el uso de redes neuronales artificiales otorga resultados satisfactorios y que su optimización podría resolver de manera excepcional la detección y posible clasificación de peces en el petén yucateco, manteniendo todas las ventajas del uso de cámaras mencionadas por Espinosa en su trabajo.

1.1. Antecedentes

El problema de la detección y clasificación de objetos en imágenes no es un problema nuevo, este ha sido atacado desde diferentes perspectivas y métodos de visión por computadora, actualmente la manera mas popular y eficiente es con el uso de redes neuronales convolucionales profundas. En esta sección se habla en paralelo acerca de la visión por computadora y el desarrollo de las redes neuronales.

El primer modelo de red neuronal fue propuesto por McCulloch y Pitts (McCulloch & Pitts, 1988). Esta red era un clasificador binario con dos clases. La desventaja era que los pesos necesarios para realizar la clasificación eran determinados manualmente.

En los 50's Frank Rosenblatt propone lo que hoy en día es la unidad

mínima de construcción de una red neuronal, el "Perceptrón" (Rosenblatt, 1958), lo revolucionario de su modelo es que era capaz de aprender de manera automática los pesos necesarios para realizar de manera correcta su tarea, esto fue revolucionario ya que eliminaba la presencia de un humano que los modificara manualmente.

Los primeros esfuerzos para resolver los problemas de la visión por computadora se realizaron a principios de los años 60 en el Instituto de Tecnología de Massachussets (MIT), a cargo del profesor Seymourt, cuando se intentó mimetizar el sistema visual humano en una computadora. El proyecto fue llamado "Summer Vision Project"(Papert, 1966) que tuvo el objetivo de resolver el problema de la visión por computadora. El sistema debía ser capaz de construir regiones con base en una imagen, estas regiones se clasificarían como objetos, fondo y ruido, para finalmente realizar la clasificación de los objetos al compararlo con una base de datos de objetos conocidos.

Como era de esperarse el proyecto no tuvo éxito y mas de 60 años después se continúa investigando para encontrar nuevas soluciones a los problemas de la visión por computadora.

El desarrollo de técnicas basadas en el perceptrón de Rosenblatt no se detuvo en todo ese tiempo, pero fue en 1969 que Minsky y Papert publican un trabajo (Minsky & Papert, 1969) en el que demuestran que un perceptrón con una función de activación lineal no era mas que un clasificador lineal, es decir, sin importar la complejidad de la red, esta nunca seria capaz de resolver problemas no lineales. Este articulo estuvo a punto de exterminar toda investigación relacionada al trabajo con redes neuronales.

El siguiente avance significativo en el área de la visión por computadora se dio en los años 80, con la publicación del libro "Vision"por David Marr (Marr, 1982). En el se propuso la existencia de características básicas en las imágenes que pertenecían a los objetos y permiten a los humanos la identificación de estos. Estas características, que son esquinas, aristas y/o curvas, se utilizan para construir representaciones de mas alto nivel y con ello poder comprender la imagen que se observa.

A pesar que su libro fue un hito para el área, su razonamiento fue de muy alto nivel y abstracto, ya que no propuso ningún algoritmo o modelo matemático para la extracción de estas características.

Contemporáneo a Marr, comienza la convergencia de ambas ramas de investigación, Kunihiko Fukushima creo una red artificial llamada "Neocognitron"(Fukushima, 1980) capaz de reconocer patrones independientemente

si estos estaban rotados o trasladados. Lo revolucionario de esta red es que contaba con capas de convolución cuyos campos receptivos tenían ciertos pesos, conocidos como filtros.

El "Neocognitron"de Fukushima estuvo fuertemente inspirado en el trabajo de los neurofisiólogos, David Hubel y Torsten Wiesel (Hubel & Wiesel, 1959). En esta investigación descubrieron que la corteza cerebral de los gatos reacciona al movimiento de ciertos patrones, en lugar de solo a imágenes estáticas. Con base en esto, los investigadores establecieron que existen neuronas simples y complejas y que el procesamiento inicia con estructuras sencillas como líneas rectas, curvas, bordes, esquinas, entre otras.

La red neuronal desarrollada por Fukushima funcionaba con base en la misma idea, que conforme la información se propagaba a través de las capas de la red, estas se especializaban en hallar características cada vez mas complejas.

Posteriormente, Yann LeCun tuvo una idea revolucionaria en su tiempo, aplicar el algoritmo de propagación hacia atrás [2] a la red de Fukushima. Este trabajo culminó con el lanzamiento de LeNet-5 (LeCun et al., 1990), una red capaz de clasificar dígitos numéricos. A LeNet-5 se le considera la primera red convolucional moderna.

En las décadas posteriores algunos avances importantes fueron: el realizado por David Lowe, en donde describió un sistema que utiliza características invariantes a la rotación, posición y a cambios de la iluminación (Lowe, 1999). Dos años despues Paul Viola y Michael Jones presentaron el primer sistema de detección de rostros en tiempo real (Viola & Jones, 2001). A pesar de no utilizar algoritmos de aprendizaje automático, era interesante el uso de filtros que aprendían las características de la imagen. Estas características eran muy simples, por lo que era necesario calcular una cantidad enorme y propagarlas en cascada, lo que causaba que el aprendizaje sea muy costoso computacionalmente.

Con el paso de los años, se crearon diferentes bases de datos que servían como pruebas de rendimiento [3] para la comparación de los sistemas de visión por computadora. En 2010 se lanza "La Competencia de Reconocimiento Visual a Larga Escala ImageNet"[4], cuya base de datos cuenta con mas de un millón de imágenes y alrededor de mil clases de objetos.

[2] *Backpropagation*
[3] *Benchmarking*
[4] *The ImageNet Large Scale Visual Recognition Competition*

Durante los primeros dos años, la tasa de error rondaba el 26 %, pero en 2012 un equipo de la Universidad de Toronto presentó un modelo de red neuronal convolucional (AlexNet), obteniendo una tasa de error del 16.4 %.

A partir de ese momento las redes neuronales convolucionales (CNN por sus siglas en inglés[5]) han dominado las pruebas de rendimiento.

Las razones principales por las que las redes convolucionales tuvieron su auge hasta esta década se deben principalmente a la mayor potencia de cómputo disponible y a las vastas bases de datos disponibles para su entrenamiento.

En este trabajo se plantea utilizar una arquitectura de red convolucional especializada en la detección y clasificación de objetos en imágenes llamada *Mask R-CNN*, esta es producto de una serie de iteraciones realizados por varios grupos de investigación. El primer modelo propuesto es el de *R-CNN* (R. Girshick, Donahue, Darrell & Malik, 2014), realizado en la Universidad de California campus Berkeley. Este modelo utiliza un método tradicional para proponer las regiones en donde podría haber un objeto, específicamente el algoritmo de búsqueda selectiva [6], para posteriormente alimentar una red neuronal convolucional que aprenda el "mapa de características"[7].

Fast R-CNN (R. B. Girshick, 2015) fue la mejora de la arquitectura *R-CNN*, la principal diferencia es que ahora se alimenta la imagen completa a la red convolucional y se crea el mapa de características. Una vez que se tiene el mapa de características se realiza la propuesta de las "regiones de interés"[8] y se procede a la clasificación. La razón por la cual esta arquitectura es más rápida es por que no es necesario alimentar la red convolucional con todas las regiones propuestas por el algoritmo de búsqueda selectiva, sino que se alimenta una sola vez por imagen y después se proponen las regiones de interés.

Después de evaluar el desempeño de los modelos anteriores, los investigadores se dieron cuenta de la existencia de un cuello de botella en ambos, el algoritmo de búsqueda selectiva. Fue en la propuesta de *Faster R-CNN* (Ren, He, Girshick & Sun, 2015) que se eliminó y se hizo que la red fuera quien aprenda a proponer las regiones de interés. Esto mejoró de manera impresionante la eficiencia de la red, haciéndola mas rápida para predecir.

[5] *Convolutional Neural Network*
[6] *Selective search algorithm*
[7] *Feature map*
[8] *Regions of interest*

Los modelos anteriores resolvían el objetivo de la detección de objetos, es decir, son capaces de encontrar objetos en una imagen y encerrarlos en recuadros, pero son incapaces de decir que pixeles dentro de esos recuadros pertenecen al objeto y cuales no. A este nuevo objetivo se le conoce como segmentación de instancias[9] y el modelo *Mask R-CNN* (He et al., 2017a) cumple con este objetivo. Para lograr la segmentación de instancias la arquitectura *Mask R-CNN* agrega una etapa en la cual una red convolucional toma las regiones clasificadas por objeto de interés como entrada y genera máscaras para ellos, separando el objeto de interés del resto de la imagen.

1.2. Trabajos relacionados

Relacionado a la detección y clasificación de peces se han realizado varios trabajos, la mayoría de ellos utilizando métodos clásicos de visión por computadora. Como el presentado por Violetta Shevchenko (Shevchenko, Eerola & Kaarna, 2018) en la cual hacen una comparación de los métodos de sustracción de fondos, los cuales toman una imagen de entrada y crean una máscara en la cual se separa el fondo del primer plano(en donde se encuentra el pez). Posterior a esto, se toma la imagen resultante y se le aplican otra serie de procesos. Por ejemplo, el flujo óptico, en el cual se toma en cuenta que dos imágenes consecutivas (cuadros de un vídeo), no varían significativamente por lo que se puede determinar un flujo de movimiento de los pixeles. Posteriormente se hace un análisis de componentes conectadas, estas se extraen y se hace un análisis de sus propiedades, estas propiedades son área, centroide, cuadro delimitador, velocidad media y dirección del movimiento. Los resultados del trabajo realizado por Shevchenko se encuentran entre el 50 % y el 80 % de precisión.

En un trabajo parecido, Richard A. Tidd (A. Tidd & Wilder, 2001) analiza los histogramas del nivel de grises y establece un umbral en el cual se decide si algo pertenece al fondo o a un posible pez. Posteriormente utiliza un filtrado de la imagen resultante y por último impone ciertas restricciones a la relación de aspecto y el tamaño para poder decir que algo es un pez o no. Al final de su trabajo resalta que usar la relación de aspecto y el área como las características para la clasificación no es determinante para todas las especies, es capaz de clasificar una de las tres correctamente, pero las otras dos no siempre eran correctamente clasificadas.

[9] *Instance segmentation*

Otro trabajo interesante es el realizado por Concetto Spampinato (Spampinato et al., 2010), en el cual usa 50 descriptores de forma, los cuales le ayudan en la clasificación de los peces. Con esta metodología obtuvo un 92 % de precisión promedio en la clasificación de las especies que le interesaban.

Los tres trabajos anteriores hicieron uso de metodologías "tradicionales", como se puede notar hay que tomar muchas cosas en consideración para elegir correctamente el preprocesado de la imagen y los descriptores de forma mas importantes, también hay que mencionar que dependiendo de la tarea, muchas veces se debe replantear el problema y la metodología a usar.

Es en los problemas anteriores en donde las redes neuronales destacan, siendo una solución cuasigeneral. En el trabajo de Ben Tamou Abdelouahid (Tamou, Benzinou, Nasreddine & Ballihi, 2018) se pueden notar las grandes ventajas que tiene el acercamiento de las redes neuronales convolucionales con transferencia de aprendizaje. En este trabajo él utiliza una red neuronal convolucional, específicamente la AlexNet. Debido a que las bases de datos de peces no son tan extensas, el aprovecha el uso de la transferencia de aprendizaje para evitar el entrenamiento desde cero de la red. Abdelouhuadi utiliza una red preentrenada en la base de datos ImageNet y entrena la red para que se especialice en imágenes de peces obteniendo precisiones y exactitudes mayores al 90 % en la base de datos Fish4Knowledge[10] y obteniendo un sobresaliente 76 % de exactitud, comparado con otros modelos, y un 58 % de precisión en la base de datos LifeClef[11]. Con los valores que Abdelouahid presenta inmediatamente se nota un incremento significativo en la tasa de clasificación correcta, es por ello que el problema de reconocimiento de imágenes y de patrones en general ha tenido un giro hacia el uso de redes neuronales.

En el trabajo realizado por Dmitry Konovalov (Konovalov et al., 2019) se presenta una metodología de entrenamiento de redes neuronales convolucionales que permite mantener desempeños notables sobre diferentes dominios, es decir, diferentes especies de peces en diferentes ambientes (fondos de la imagen). Con este trabajo alcanzan una tasa de error de clasificación del 0.17 % de falsos positivos y del 0.61 % en falsos negativos. Es importante mencionar que en este trabajo solo se realiza la clasificación de las imágenes con base en si contiene un pez o no.

En este trabajo se continuará con el uso de redes neuronales convolucionales, ya que en la última década han mostrado grandes ventajas frente a

[10] www.fish4knowledge.eu
[11] www.imageclef.org/lifeclef/2015/fish

los métodos tradicionales de visión por computadora como los utilizados por Spampinato et al., 2010, A. Tidd y Wilder, 2001 y Shevchenko et al., 2018. Estas ventajas son en escalabilidad, capacidad de inferencia y sobre todo en desempeño.

Al utilizar una arquitectura como Mask-RCNN se tiene una mejora significativa comparada con los trabajos de Konovalov et al., 2019 y de Tamou et al., 2018 ya que se usa una arquitectura de red mas reciente. Esta tiene la capacidad de lograr la segmentación de instancias, la cual es la tarea mas difícil y completa en el reconocimiento de objetos. Ya que la arquitectura esta compuesta por redes neuronales únicamente, es completamente entrenable utilizando bibliotecas de aprendizaje profundo como TensorFlow o Pytorch.

1.3. Objetivos

Con base en las necesidades y condiciones del proyecto y en la investigación de los trabajos relacionados, se definió el objetivo general y los siguientes objetivos particulares.

1.3.1. Objetivo general

Desarrollar un sistema computacional basado en redes neuronales profundas para cumplir con la tarea de segmentación de instancias de peces en ambientes subacuáticos, entrenado a partir de imágenes digitales obtenidas en el peten yucateco.

1.3.2. Objetivos particulares

- Proponer un protocolo de selección de imágenes para la creación de una base de datos para el entrenamiento de sistemas de detección automáticos y entregar una base de datos para su uso en nuevos modelos.
- Entrenar un sistema de detección de peces usando redes neuronales convolucionales y evaluar el desempeño del sistema en diferentes conjuntos de prueba.
- Contar con la capacidad de que el sistema sea escalado, reentrenado y se le puedan agregar categorías de clasificación fácilmente una vez que

se cuente con una base de datos etiquetada.

- Comparar el desempeño del sistema contra la detección visual.

En el siguiente capitulo se profundiza en la teoría que rodea al aprendizaje automático y las técnicas que son utilizadas para el entrenamiento de sistemas de reconocimiento de objetos con redes neuronales profundas.

Capítulo 2

Marco teórico

Es importante definir algunos conceptos antes de profundizar en el desarrollo de este trabajo.

2.1. Procesamiento de imágenes

El procesamiento de imágenes no es lo mismo que la visión por computadora, este tiene como fin crear una imagen a partir de una inicial, ya sea simplificando o enriqueciendo su contenido. Es un tipo de procesamiento de señales y no tiene interés en comprender el contenido conceptual de la imagen (Tinku Acharya, 2005). Sin embargo, la visión por computadora toma prestadas muchas técnicas del procesamiento de imágenes, sobre todo para la etapa de preprocesamiento.

Algunos ejemplos de técnicas de procesamiento de imágenes son las siguientes:

- **Suavizar la imagen**: Disminuir la variación en la intensidad entre los pixeles vecinos.
- **Eliminar ruido:** Eliminar valores de pixeles que no pertenezcan a la naturaleza de la imagen, sino que se deben al proceso de captura de la imagen.
- **Detectar bordes:** Obtener una imagen en la que se aprecien únicamente los bordes de los objetos de interés en la imagen original.

Otra de las técnicas que se utilizan consiste en transformar la imagen al dominio de la frecuencia haciendo uso de la transformada de Fourier y aplicar filtros como si de un sistema dinámico se tratara.

2.2. Visión por computadora

La visión por computadora es una rama de la ciencia y de la ingeniería en computación que se dedica a crear sistemas capaces de imitar el proceso cognitivo del sistema visual humano. Esta se ayuda del procesamiento de imágenes para poder interpretar la información que se le presenta.

En su libro "Computer vision: Models, Learning and inference"Prince habla de la meta de la visión por computadora.

> "A un nivel abstracto, la meta de la visión por computadora es utilizar la información de las imágenes adquiridas para inferir algo sobre el mundo."(Prince, 2012, p. 55)

Este enunciado, a pesar de dar una noción del tema, deja un panorama muy ambiguo, ya que no menciona la importancia de que el proceso sea automático ni lo que se hace con la información inferida.

Se presenta la siguiente definición para complementar la idea:

> "La visión por computadora es el proceso automático de extracción de información a partir de imágenes. Por información se puede entender, desde modelos tridimensionales, posición de la cámara, detección de objetos, hasta reconocimiento, agrupamiento y búsqueda de contenido en las imágenes."(Solem, 2012, p. IX)

En esta definición sí se hace énfasis en la importancia de que el proceso sea automático, también se hace explícito el uso que se le da a la información obtenida de las imágenes.

Se nota rápidamente que la visión por computadora es un campo multidisciplinario, en donde conviven diferentes ramas de la ciencia y la ingeniería.

2.3. Tareas de la visión por computadora

Como se mencionó anteriormente, la visión por computadora está relacionada con una vasta gama de disciplinas, por lo que tiene una cantidad innumerable de aplicaciones en la vida moderna. Una manera de clasificarla por el tipo de aplicación es en detección, clasificación y modelado 3D a partir de imágenes.

- **Reconocimiento de objetos**: Dada una imagen, el algoritmo debe ser capaz de reconocer ciertos objetos de interés dentro de ella. Dependiendo de como lo haga puede ser clasificación, segmentación semántica, detección o segmentación de instancias.
- **Modelado 3D a partir de imágenes (Fotogrametría)**: La fotogrametría consiste en la reconstrucción de modelos tridimensionales a partir de una o varias imágenes bidimensionales de algún objeto.

El reconocimiento de objetos puede clasificarse de la siguiente manera:

- **Clasificación**: Cuando se tiene una imagen y se quiere saber que es lo que hay dentro se hace uso de la clasificación, por ejemplo, si se tiene un conjunto de imágenes y se desea determinar la presencia o falta de algún objeto, se puede utilizar un sistema de visión por computadora para que automáticamente haga la clasificación de esas imágenes.
- **Segmentación semántica**: Esta tarea de la visión por computadora permite separar por clases los objetos que se encuentran dentro de una imagen, teniendo la característica de decir que pixeles pertenecen a una clase especifica. No es capaz de discernir entre multiplicidad de objetos de la misma clase.
- **Detección**: La detección en el marco de la visión por computadora se refiere a la capacidad de encontrar objetos dentro de una imagen. Tiene la característica que es capaz de diferenciar entre objetos de la misma clase, es decir cuantos y donde se encuentran los objetos de la clase de interés delimitándolos en un recuadro y asignándoles una etiqueta.
- **Segmentación de instancias**: Esta es la tarea más completa y por lo tanto la más compleja de realizar, ya que no solo dice en donde y cuantos objetos de interés como la detección, sino que es capaz de decir que pixeles dentro del cuadro delimitador pertenecen al objeto.

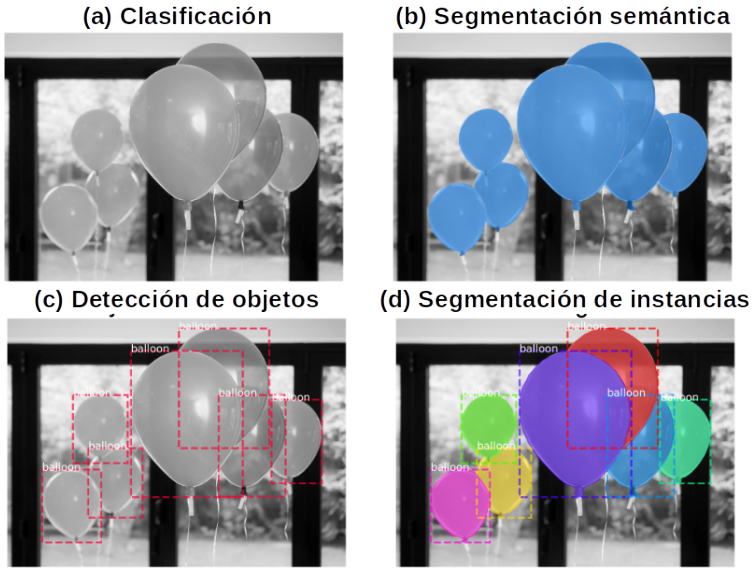


Figura 2.1: Ejemplo de las diferentes tareas del reconocimiento de objetos. [1]

En la figura 2.1 se puede observar un ejemplo de la clasificación del reconocimiento de objetos. En esta figura se observan ejemplos de la división del reconocimiento de objetos.En el inciso (a) se ejemplifica la clasificación, esta solo indica la presencia o no del globo. La segmentación semántica indica que pixeles pertenecen a la clase globo, sin diferenciar en la cantidad, esta se observa en el inciso (b). La detección de objetos, inciso (c) indica cuales son y donde están los globos, tomando en cuenta si hay más de uno. La segmentación de instancias, inciso (d), indica cuales son y donde están lo globos al igual que la detección incluyendo la diferenciación entre los pixeles dentro del cuadro delimitador.

2.4. Aprendizaje automático

Existen varias formas de intentar resolver el problema de la visión por computadora, pero hoy en día se ha demostrado en la práctica que los más eficaces son los basados en aprendizaje automático. El aprendizaje automáti-

[1] Por Abdulla, W. (2018). Tareas del reconocimiento de objetos. Recuperado de: https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46

co es un paradigma de solución de problemas en el cual no se programa explícitamente que es lo que se quiere que el sistema realice, si no que se le enseña con base en ejemplos lo que debe de hacer y después de una gran cantidad de iteraciones el sistema es capaz de reproducir lo aprendido en ejemplos no vistos con anterioridad.

El término "aprendizaje automático"[2] fue propuesto por Arthur Samuel, pero no aportó ninguna definición concreta. Es complicado proponer una definición ya que incluso en el área del aprendizaje automático no hay un consenso. Una de las definiciones mejor aceptadas es la dada por Tom Mitchell:

> "Se dice que un programa de computadora aprende de la experiencia E con respecto a una tarea T y una medida de desempeño P, si su desempeño en T, medido por P, aumenta con la experiencia E."(Mitchell, 1997)

Esta definición a pesar de su formalismo, deja muy en claro las bases de lo que se conoce como aprendizaje automático, ya que de una manera concreta explica como un programa mejora en la realización de una tarea, con base en la experiencia, es decir, en el entrenamiento. A continuación se tratará de clarificar los términos usados por Mitchell en su definición, utilizando la intuición proveída por Goodfellow en su libro (Goodfellow, Bengio & Courville, 2016):

- **Tarea, T**: Este término hace referencia a la tarea que se planea que el sistema sea capaz de resolver, por ejemplo: clasificación, regresión, detección de anomalías, traducción automática, etc. En general las tareas de un sistema de aprendizaje automático son descritas con base en el proceso que debe llevar un ejemplo, es decir, una entrada. Por ejemplo en un sistema de clasificación la entrada son los pixeles de una imagen, la tarea es clasificar con base en la presencia de algún objeto y la salida puede ser un bit que se enciende cuando se encuentra el objeto en la entrada.
- **Desempeño, P**: Para poder decir que tan bien se comporta un sistema de aprendizaje automático es necesario contar con una medida del desempeño. Estas medidas de desempeño suelen ser específicas para la tarea T que el sistema lleva a cabo. En las tareas de clasificación, generalmente se utiliza la precisión del modelo como medida de desempeño.

[2] *Machine learning*

En este caso la precisión es la razón entre ejemplos para los cuales el sistema predice la salida correcta. Es importante mencionar que no existe una sola métrica de desempeño y que muchas veces para la misma tarea dependerá de la aplicación la que se decida usar.

- **Experiencia, E**: Cuando se entrena a un algoritmo de aprendizaje automático se le permite experimentar para posteriormente corregir sus aproximaciones hasta converger y predecir correctamente nuevos ejemplos. Esta experiencia se gana al permitir al sistema probar con distintos ejemplos de una base de datos (una colección de ejemplos).

2.4.1. Clasificación de algoritmos de aprendizaje automático

Dependiendo del tipo de experiencia al que se somete el sistema de aprendizaje automático se le puede clasificar de la siguiente manera:

- **Algoritmos de aprendizaje supervisado:** En este tipo de algoritmos al sistema se le permite experimentar con una base de datos que incluye un objetivo o etiqueta asociado a cada dato. Por ejemplo, una base de datos de imágenes de perros y gatos, en la que el objetivo del sistema es aprender a clasificarlos correctamente. Cada imagen está asociada con una etiqueta que dice la clase de animal que se encuentra en ella, perro o gato. De esta manera el algoritmo tiene un punto de comparación al predecir, y con ello corregir sus valores y acercarse cada vez más a la etiqueta correcta.

- **Algoritmos de aprendizaje sin supervisión o no supervisados:** A diferencia de los algoritmos supervisados, los algoritmos no supervisados carecen de una etiqueta con la cual comparar su desempeño, por lo que se espera que sean capaces de aprender ciertas características y propiedades de la estructura de la base de datos. Un ejemplo clásico es el de agrupamiento[3], en el que se permite que el algoritmo cree grupos con base en la similaridad de sus propiedades.

Es importante mencionar que las lineas que separan esta clasificación están difusas por lo que algunos algoritmos pueden ser usados en cualquiera de las dos maneras. Existe una tercera clasificación que no se menciona explícitamente, el aprendizaje semisupervisado, este se encuentra entre los

[3] *Clustering*

dos mencionados con anterioridad, su funcionamiento es entrenar un modelo con base en una pequeña cantidad de datos y utilizarlo para etiquetar el resto de los datos.

2.4.2. Sobreajuste y subajuste.

Al trabajar con algoritmos de aprendizaje automático el mayor reto y el objetivo principal es que el sistema se desenvuelva de manera eficiente en ejemplos que se encuentran fuera de la base de datos, es decir, entradas desconocidas. A esta habilidad se le conoce como generalización.

Cuando se entrena un modelo de aprendizaje automático, generalmente se cuenta con una base de datos de entrenamiento y se puede calcular una métrica del error, a esta se le llama error de entrenamiento. Al entrenar el sistema se pretende reducir el error de entrenamiento, pero no se debe perder de vista que la finalidad es reducir el error en la predicción de nuevas entradas al sistema, a estas nuevas entradas se les conoce como base de datos de prueba. Si se asume que ambas bases de datos provienen de la misma distribución de probabilidad p_{data} se puede esperar que al reducir el error en el entrenamiento se logrará lo mismo en la base de datos de prueba.

Con base en lo mencionado anteriormente se puede decir que lo que determina que tan bien se desempeña el sistema esta dado por:

1. Disminuir el error de entrenamiento.
2. Disminuir la brecha entre el error de entrenamiento y el error de prueba.

Estos factores también son los mayores retos del aprendizaje automático: Sobreajuste[4] y subajuste[5]. El subajuste ocurre cuando el modelo no es capaz de alcanzar un error de entrenamiento lo suficientemente bajo. El sobreajuste sucede cuando la brecha entre el error de entrenamiento y el error de prueba es muy grande.

Para comprender de mejor manera la idea de subajuste y sobreajuste se puede introducir el concepto de capacidad o complejidad de un modelo de aprendizaje automático. Una manera intuitiva de entender esta idea es con el siguiente ejemplo: Se cuenta con una serie de puntos que intrínsecamente

[4] *Overfitting*
[5] *Underfitting*

siguen un comportamiento cuadrático e incluyen cierto grado de ruido por la medición. Se proponen tres modelos de ajuste, uno lineal de la forma $\hat{y} = \theta_1 x + \theta_0$, uno cuadrático de la forma $\hat{y} = \theta_2 x^2 + \theta_1 x + \theta_0$ y uno de grado ocho, cuya ecuación se omite. En donde θ_i son parámetros que serán aprendidos.

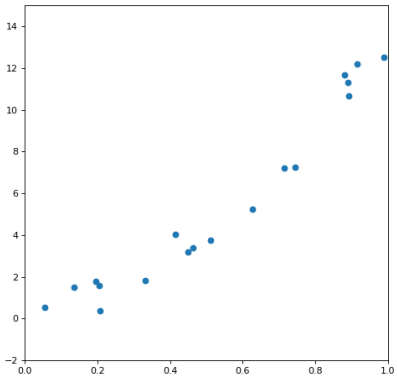
En la figura 2.2 se pueden observar los tres casos posibles. El modelo lineal no tiene la complejidad suficiente para representar el fenómeno que generó los datos, por lo que se dice que el modelo está subajustado. El modelo de octavo grado, a pesar de ajustarse exactamente a todos los puntos, no representa de manera real el fenómeno del que provienen, por lo que se dice que está sobreajustado a los datos, su complejidad es mayor de la necesaria. Finalmente, se puede notar que el modelo cuadrático, a pesar de no pasar exactamente por todos los puntos, es el que mejor representa el fenómeno del que provienen los datos.

2.4.3. El teorema "No free lunch"

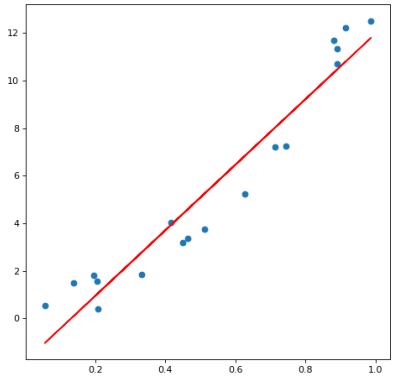
La teoría de aprendizaje propone que un sistema es capaz de generalizar a partir de una cantidad finita de ejemplos, esto es contradictorio con el razonamiento inductivo, ya que este dice que para poder inferir una regla que describa a todos los elementos de un conjunto, es necesario conocer a todos los elementos de este.

Debido a lo anterior, es importante aclarar que un sistema de aprendizaje automático no es capaz de otorgar reglas deterministas, sino reglas probabilísticas. En otras palabras, un sistema de aprendizaje automático solo es capaz de encontrar las reglas que probablemente son correctas acerca de la mayoría de los elementos de un conjunto.

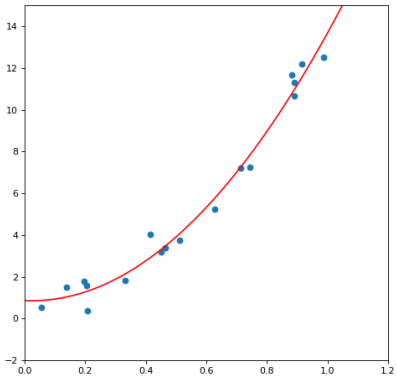
El teorema "No free lunch" (Wolpert & Macready, 1997) dice que en promedio todos los modelos de aprendizaje automático tienen el mismo desempeño al tomar en cuenta todas las distribuciones de probabilidad que generan datos. Lo importante de este teorema es que cambia el paradigma de diseño en el aprendizaje automático, haciendo que se creen sistemas especializados por dominio en lugar de un algoritmo de aprendizaje universal. Cuando se limita al modelo a aprender tareas específicas, es posible obtener buenos resultados.



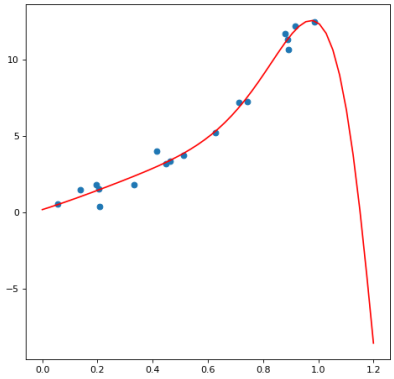
(a) Puntos generados con una ecuación cuadrática y ruido aleatorio.



(b) Regresión lineal, se observa como hay un subajuste ya que la complejidad del modelo no es suficiente para expresar la complejidad de las observaciones.



(c) Regresión cuadrática, se observa como el modelo se ajusta de manera adecuada.



(d) Regresión a un polinomio de grado ocho, se observa como el modelo se ajusta muy bien a los datos de entrenamiento pero no es capaz de expresar la naturaleza real de los puntos, este es un ejemplo de sobreajuste.

Figura 2.2: Curvas de ajuste.

2.4.4. Parámetros e hiperparámetros

Cuando se habla de aprendizaje automático, se sabe que el entrenamiento del modelo hará que este aprenda a realizar una tarea, la manera en la que el sistema aprende es ajustando el valor de ciertos parámetros. Estos son aprendidos durante el entrenamiento y aunque el encargado del sistema puede dar valores iniciales, el fin último es que estos se hallen de manera automática. Un ejemplo de ellos son los pesos de cada neurona en una red neuronal (ver sección 2.5.1). Los hiperparámetros son aquellos que son manualmente ajustados por el encargado de la creación del modelo, estos pueden ser el número de iteraciones del algoritmo, la tasa de aprendizaje en el descenso por gradiente (ver sección 2.5.5), el número de capas ocultas de una red neuronal (ver sección 2.5.3), etc.

2.4.5. Conjuntos de entrenamiento, validación y prueba

En aprendizaje automático, implícitamente se entiende que se cuenta con una base de datos. Se debe ver a este conjunto de información como el bien más preciado, es por ello que es necesario tomar decisiones adecuadas para su uso. Uno de los primeros pasos es separar la base de datos de manera adecuada, de tal forma que se puedan dar aseveraciones confiables del desempeño del modelo que se está creando. Como se mencionó en la sección 2.4.2 el primer paso es separar en dos conjuntos el de entrenamiento y el de prueba.

El de prueba es un conjunto que no se usará hasta el final del entrenamiento del modelo para poder calcular el error de generalización. El conjunto de entrenamiento es el que se usará para que el modelo encuentre los parámetros adecuados, pero como se mencionó, existen ciertos parámetros que no son ajustados por el algoritmo de entrenamiento, estos son llamados hiperparámetros. Es por lo anterior que se toma una proporción del conjunto de entrenamiento a la que se le conoce como validación, esta sirve para ajustar los hiperparámetros y entregar un estimado del error de generalización después y durante el entrenamiento. De cierta manera se puede decir que el conjunto de validación sirve para entrenar los hiperparámetros.

Cuando finalmente se ha hallado el valor adecuado de los hiperparámetros se puede evaluar el error de generalización en el conjunto de prueba. De esta manera se asegura que las observaciones en el conjunto de prueba nunca fueron utilizadas durante el entrenamiento.

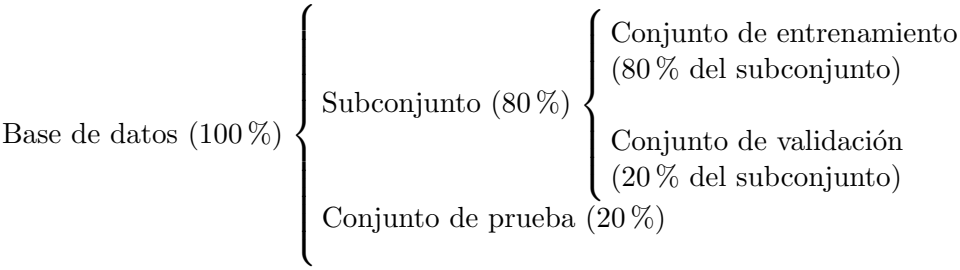


Figura 2.3: Distribución del tamaño de los conjuntos.

No existe un consenso en la proporción que se debe separar el conjunto de datos, ya que esto depende del número de observaciones con las que se cuente. Una propuesta de división se basa en el principio de Pareto (Pareto, 1896), una razón de 80/20, (véase la figura 2.3).

2.4.6. Regularización

Para evitar el sobreajuste es posible agregar ciertos términos a la función de costo que penalicen los valores grandes de los parámetros, ya que estos se asocian a una mayor complejidad y por ende es altamente probable que se esté sobreajustando a los datos. Si se supone que la función de costo es el error cuadrático medio entre el valor real y el predicho, se puede agregar a esta la norma del vector de parámetros $\boldsymbol{\theta}$. La norma usualmente usada es la L2 aunque también se suele hacer uso de la norma L1 y la L0 en menores ocasiones. La función de costo con el término de regularización queda como:

$$J(\theta)s = MSE_{train} + \Omega(\theta) \tag{2.1}$$

En donde:

> MSE_{train}: Es el término del error cuadrático medio entre la predicción y el valor real.
> $\Omega(\theta)$: Es el término de regularización, en caso de ser la norma, se calcula como: $L_j = \sum_i = 1^n \theta_i^j$. En donde j es el índice que especifica si es norma L0, L1, L2, etc.

Con este término se está penalizando el tamaño de los parámetros y se evita que el modelo se sobreajuste, es decir, que el error de generalización

disminuya. La norma solo es un ejemplo de regularización que evita específicamente el crecimiento del valor de los parámetros, pero existen otros métodos. Por ejemplo en redes neuronales (ver sección 2.5.3), es posible desactivar neuronas aleatoriamente en la red durante el entrenamiento para evitar que el desempeño de la red dependa de alguna en específico. Al método mencionado se le conoce como *Dropout*. Una definición de la regularización es la que Goodfellow propone:

> "La regularización es cualquier modificación que se haga a un algoritmo de aprendizaje con el propósito de reducir su error de generalización, pero no el de entrenamiento"(Goodfellow et al., 2016, p. 117)

2.5. Aprendizaje profundo

El aprendizaje profundo es una subrama del aprendizaje automático, un diagrama de Venn que ejemplifica la relación entre la inteligencia artificial [6], el aprendizaje automático y el aprendizaje profundo se puede observar en la figura 2.4.

Entender el concepto de aprendizaje profundo puede resultar un poco complejo ya que al ser un área de investigación relativamente nueva no hay un consenso en definiciones. Uno de los acuerdos en el área es que el elemento básico son las redes neuronales profundas, con lo anterior surge la pregunta: ¿Qué hace profunda a una red neuronal? Como se verá en el siguiente apartado, la profundidad de una red está dada por la cantidad de capas ocultas o intermedias que tiene.

2.5.1. Redes Neuronales Artificiales

Las redes neuronales artificiales son una metodología de aproximación de funciones no lineales que tienen inspiración en la manera, al menos en teoría, en la que los cerebros funcionan. Un cerebro está formado por una infinidad de células llamadas neuronas que se interconectan creando una red, esta se encarga de todos los procesos cognitivos.

[6] Dado que no existe una definición aceptada para lo que es o no inteligente, y que la discusión roza lo filosófico más que lo científico, en este trabajo no se profundizará en el concepto de inteligencia artificial.

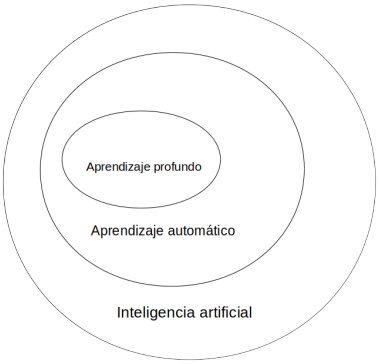


Figura 2.4: Diagrama de Venn que muestra la relación entre la inteligencia artificial, el aprendizaje automático y el aprendizaje profundo.

Visto desde un punto de vista más formal y retomando la primera oración de esta sección, las redes neuronales artificiales son modelos que tratan de aproximar una función f^*, un ejemplo puede ser una función que relaciona los pixeles de una imagen x con un vector que codifica si la imagen es de un perro o un gato y, $y = f^*(x)$. Esta función ideal o perfecta es desconocida por lo que es trabajo de la red neuronal aprenderla de alguna manera. La manera en la que la red aprende a realizar la tarea es el entrenamiento.

En términos matemáticos la función propuesta por la red neuronal es un mapeo de $\boldsymbol{x}$ a y, $y = f(\boldsymbol{x}, \boldsymbol{\theta})$, en donde $\boldsymbol{x}$ es el vector de entradas y $\boldsymbol{\theta}$ es el vector de parámetros, llamados pesos. Los pesos son modificados en el entrenamiento de la red y son los que permiten que la red aproxime a la función f^*.

Perceptrón

Fue Rosenblatt, al proponer su "Perceptrón" (Rosenblatt, 1958), quien otorga el bloque básico de construcción de las redes neuronales artificiales. En la figura 2.5 se puede observar un diagrama.

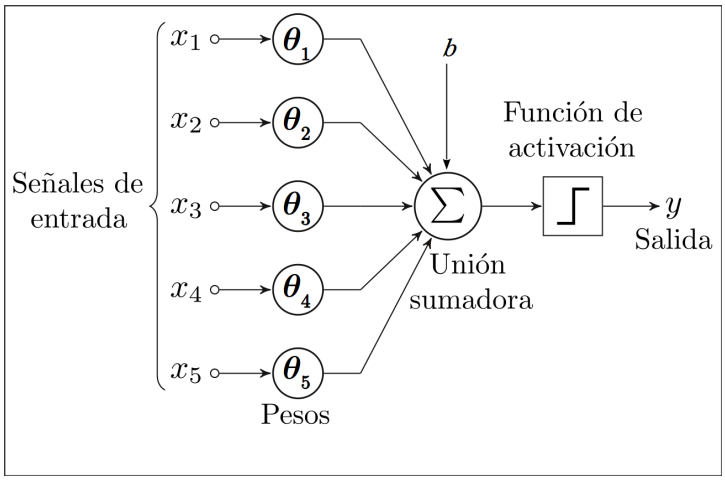


Figura 2.5: Diagrama de un perceptrón con cinco entradas y una salida. [7]

Este modelo puede expresarse de la siguiente manera:

$$y = f(\boldsymbol{x}) \tag{2.2}$$

En donde $\boldsymbol{x}$ es el vector de entradas dado de la siguiente manera:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \tag{2.3}$$

Y en donde $\boldsymbol{\theta}$ es la matriz de los coeficientes conocidos como pesos:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} \tag{2.4}$$

El término b se conoce como sesgo[8], y en muchas ocasiones para realizar una sola multiplicación de matrices se considera que es un peso θ_0 que mul-

[7] Por Alejandro Cartas - Trabajo propio, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=41534843

[8] *Bias*

tiplica a una entrada unitaria $x_0 = 1$. Por lo que las matrices $\boldsymbol{x}$ y $\boldsymbol{\theta}$ quedan de la siguiente forma:

$$\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \quad \text{y} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} \tag{2.5}$$

Se desea que cada una de las componentes del vector de entradas se multiplique por un coeficiente θ_i llamado peso, es decir que quede de la forma $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$, una vez que esto se toma el resultado y se evalúa una función de activación con él para obtener la salida del perceptrón y. Para lograr lo anterior el primer paso es transponer el vector $\boldsymbol{\theta}$ y postmultiplicarla por el vector de entradas $\boldsymbol{x}$, para después evaluar la función de activación ϕ.

$$y = \phi(\boldsymbol{\theta}^T X) \tag{2.6}$$

La utilidad de este modelo es que es capaz de aproximar clasificadores binarios, en donde se pueden ajustar los pesos θ_i para que dependiendo del valor del vector de entradas $\boldsymbol{x}$ se obtenga la salida y deseada.

2.5.2. Funciones de activación

Siguiendo con la inspiración biológica, las funciones de activación son algo comparable con los potenciales de acción en el cerebro. En la corteza cerebral las neuronas se disparan cuando se alcanza un nivel mínimo en el potencial eléctrico entre ellas, esto causa un aumento abrupto del voltaje, a este incremento se le conoce como potencial de acción (Arthur C. Guyton, 2005). En las redes neuronales las funciones de activación juegan un papel similar, dependiendo del resultado de la multiplicación $\boldsymbol{\theta}^T X$, estas pueden "activarse" o no. Como demostraron Minsky y Papert (Minsky & Papert, 1969) las funciones de activación lineales no permiten que el sistema aproxime a fenómenos no lineales, por lo que hoy en día no se usan en la práctica.

Algunas de las funciones de activación no lineales más comunes son:

- **Sigmoide:** La función sigmoide cuya gráfica se puede observar en la figura 2.6a, es especialmente útil porque provee un cambio suave en-

tre sus valores y estos están acotados entre cero y uno. También es diferenciable en todos sus puntos, lo que es especialmente importante al momento de entrenar la red (se entrará en detalle en la sección 2.5.7). La principal desventaja es que corre el riesgo de saturarse en sus extremos, es decir que la derivada sea cero, lo que dificulta el entrenamiento de las redes neuronales profundas. Su expresión matemática es la siguiente:

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

- **ReLU[9]:** Esta función tiene inspiración biológica y fue propuesta por Hahnloser (Hahnloser, Sarpeshkar, Mahowald, Douglas & Seung, 2000), al igual que la función sigmoide es no lineal por lo que es posible usarla como función de activación para modelar fenómenos no lineales. Esta función tiene la ventaja de que no se satura para valores positivos de la entrada, es decir, siempre existe un valor de la derivada diferente de cero. A pesar de contar con algunas desventajas como que no es diferenciable en cero[10], o que no tiene cota, ha demostrado ser la mejor opción para entrenar redes neuronales profundas. Su gráfica se puede observar en la figura 2.6b. Su expresión matemática es la siguiente:

$$\phi(x) = x^{+} = max(0, x) \tag{2.8}$$

Otra manera de expresarlo es:

$$\phi(x) = \begin{cases} 0 & \text{Si x} < 0 \\ x & \text{En cualquier otro caso} \end{cases} \tag{2.9}$$

- **SoftMax:** Esta función de activación es ámpliamente usada cuando se trabaja con clasificadores ya que convierte un vector de k dimensiones con valores reales arbitrarios en un vector de las mismas dimensiones, pero con valores reales entre cero y uno que pertenecen a una distribución de probabilidad.

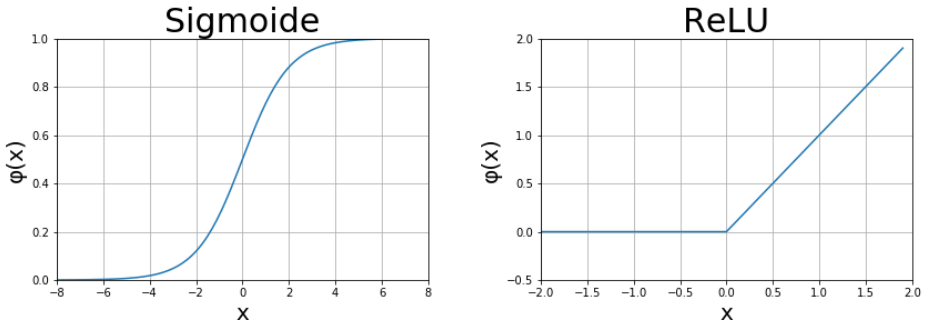
$$\phi : \mathbb{R}^K \to [0, 1]^K \tag{2.10}$$

La expresión matemática es la siguiente:

$$\phi(x)_j = \frac{e^{x_j}}{\Sigma_{k=1}^{K} e^{x_k}} \quad \text{Para } j = 1, ..., K \tag{2.11}$$

[9] *Rectified Linear Unit*

[10] Se asume en este punto, que la derivada es cero o uno.



(a) Función sigmoide. (b) Función ReLU

Figura 2.6: Gráficas de las funciones de activación.

2.5.3. Perceptrón multicapa

Si se toma la idea del perceptrón y se interconecta una serie de ellos de tal manera que se crea una red, entonces se forma una red neuronal artificial. A esta idea también se le conoce como perceptrón multicapa o red *feedforward*[11] En la figura 2.7 se puede observar como se interconectan los perceptrones o neuronas.

Siguiendo la notación del libro de Goodfellow, esta arquitectura de red puede representarse como una composición de funciones. Por ejemplo, si se tienen tres capas como en la figura 2.7, estas se pueden expresar como $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ respectivamente, por lo que la función asociada a la red es $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. A $f^{(1)}$ se le conoce como la primera capa, o capa de entrada, a $f^{(2)}$ como la capa oculta o intermedia y finalmente a $f^{(3)}$ como la capa de salida. El número de capas de la red es igual a la profundidad del modelo, a esto se debe el nombre de redes neuronales profundas.

La dimensión o el ancho de las capas ocultas está dada por el número de unidades o neuronas presentes en ella, por ejemplo, en la figura 2.7 la capa oculta tiene un ancho de 4. Mientras más profunda y ancha es la red se aumenta la capacidad del modelo y por ende se pueden aproximar funciones más complejas, claro que esto tiene un impacto directo en el costo computacional del entrenamiento.

[11] Se usa el término en inglés ya que no se encontró una traducción al español que evoque el mismo significado.

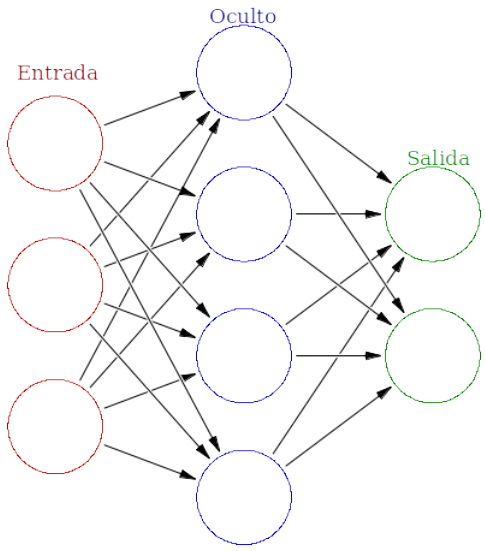


Figura 2.7: Red neuronal artificial con una capa oculta. [12]

2.5.4. Función de costo

Como se mencionó anteriormente, el objetivo de las redes neuronales es aproximar una función f^* que va de una entrada x a una salida y. De tal manera que la aproximación o predicción $\hat{y}$ de la red neuronal está dada de la manera $\hat{y} = f(x, \theta)$ en donde θ son los pesos de la red neuronal. Con la predicción $\hat{y}$ y la salida verdadera y se puede cuantificar de alguna manera que tan lejos quedó la aproximación del valor real. La manera de cuantificarlo es haciendo uso de una función de costo.

En el libro de Goodfellow (Goodfellow et al., 2016) se dice que la estimación por máxima verosimilitud ha demostrado por sus propiedades y su desempeño en la práctica ser el mejor estimador en el ámbito del aprendizaje automático.

La estimación por máxima verosimilitud permite minimizar la diferencia entre dos distribuciones de probabilidad, la p_{data} que es la del conjunto de datos de entrenamiento y la p_{model} que es la predicha por el modelo de aprendizaje. Este proceso pretende encontrar los parámetros $\boldsymbol{\theta}$ que maximizan la "similaridad" entre dos distribuciones de probabilidad.

[12] De en:User:Cburnett - File:Colored neural network uk.svg, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=78778791

La estimación por máxima verosimilitud puede expresarse como una estimación de una probabilidad condicional $P(\boldsymbol{y}|\boldsymbol{x};\theta)$ para predecir $\boldsymbol{y}$ dado $\boldsymbol{x}$. Si $\boldsymbol{X}$ representa todas las entradas y $\boldsymbol{Y}$ las salidas deseadas u observadas, entonces el estimador de máxima verosimilitud esta dado por:

$$\theta_{ML} = \underset{\theta}{argmax} P(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) \tag{2.12}$$

Si se asume que los ejemplos son independientes y pertenecen a la misma distribución de probabilidad:

$$\theta_{ML} = \underset{\theta}{argmax} \prod_{i=1}^{m} P(\boldsymbol{y}^{(i)}|\boldsymbol{x^{(i)}};\boldsymbol{\theta}) \tag{2.13}$$

Para evitar problemas de cómputo se puede expresar como una suma de logaritmos, en donde $log(f)$ es el logaritmo natural. Se puede demostrar que el argumento máximo no cambia al aplicar logaritmo sobre la función original.

$$\theta_{ML} = \underset{\theta}{argmax} \sum_{i=1}^{m} log\ P(\boldsymbol{y}^{(i)}|\boldsymbol{x^{(i)}};\boldsymbol{\theta}) \tag{2.14}$$

Finalmente se puede dividir entre m, ya que este reescalamiento no afecta el valor de la maximización de θ. Haciendo esto se puede reescribir la suma como una esperanza con respecto a la distribución de las observaciones $\hat{p}_{data}$.

$$\theta_{ML} = \underset{\theta}{argmax}\ \mathbb{E}_{\hat{p}_{data}} log\ P(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) \tag{2.15}$$

Estimación por error cuadrático medio

En muchas ocasiones se utiliza la norma L2 de la diferencia entre la predicción $\hat{y} = f(\boldsymbol{x},\boldsymbol{\theta})$ y el valor observado y , también conocida como distancia euclidiana, que se calcula de la siguiente manera:

$$d^2 = \sum_{i=1}^{m} |\hat{y}^{(i)} - y^{(i)}|^2 \tag{2.16}$$

Si se reescala entre el número de observaciones m se obtiene el error cuadrático medio[13].

$$MSE = \frac{1}{m} \sum_{i=1}^{m} |\hat{y}^{(i)} - y^{(i)}|^2 \tag{2.17}$$

[13] *Mean Squared Error* (MSE)

Se puede demostrar que la distancia euclidiana es un caso específico de la estimación por máxima verosimilitud en la cual se supone que la distribución de probabilidad de la que provienen los datos es Gaussiana.

Se utiliza la estimación por máxima verosimilitud debido a que se puede demostrar que es el mejor estimador asintótico, conforme aumenta el número de ejemplos $m \to \infty$.

Dadas las condiciones adecuadas se puede demostrar que este estimador tiene la propiedad de consistencia, es decir, que conforme aumenta el número de observaciones m, la estimación de los parámetros converge al valor real del parámetro. Las condiciones necesarias son que la distribución real de las observaciones p_{data} se encuentre dentro de la familia de distribuciones del modelo $p_{model}(\cdot, \theta)$. La segunda condición es que exista un conjunto de parámetros $\boldsymbol{\theta}$ que corresponda con la distribución real p_{data}, de otra manera se podría obtener la distribución real p_{data} pero no el conjunto de parámetros $\boldsymbol{\theta}$ usados para la generación de las observaciones.

La otra propiedad que hace del estimador por máxima verosimilitud el preferido para el aprendizaje automático es que es el que tiene la mejor eficiencia estadística, es decir, para un número fijo de observaciones m, el error de estimación es mínimo.

Verosimilitud logarítmica negativa

La máxima verosimilitud se puede expresar en términos de otra métrica, una que compara la similitud entre dos distribuciones de probabilidad, esta es llamada la divergencia de Kullback-Leibler, y se expresa de la siguiente manera:

$$D_{KL}(\hat{p}_{data}||p_{model}) = \mathbb{E}_{x\sim\hat{p}_{data}} log\, \hat{p}_{data}(x) - log\, p_{model}(x) \tag{2.18}$$

En donde:

- $\hat{p}_{data}$: Es la aproximación que da el conjunto de observaciones acerca de la distribución real p_{data} de la que se tomaron.
- p_{model}: Es la distribución de probabilidad propuesta por el modelo de aprendizaje automático.

Cuando las distribuciones de probabilidad son muy diferentes la divergencia KL es igual a un valor alto, cuando son idénticas esta es igual a cero. Por lo

anterior se desea minimizar este valor. Al minimizar la divergencia KL, la expresión se puede reducir a la entropía cruzada entre las distribuciones, ya que el primer término de la divergencia KL no depende del modelo.

$$H(\hat{p}_{data}, p_{model}) = -\mathbb{E}_{x\sim\hat{p}_{data}}[log\, p_{model}(x)] \tag{2.19}$$

Si se observa detenidamente, esta es la misma que la máxima verosimilitud pero con el signo contrario, a esta expresión se le conoce como la verosimilitud logarítmica negativa[14]. En aprendizaje automático lo más común es utilizar esta expresión como la función de costo $J(\boldsymbol{x}, \boldsymbol{\theta})$.

A partir de lo anterior se puede concluir que lo que la estimación por máxima verosimilitud trata de lograr es que la distribución de probabilidad del modelo iguale a la distribución de probabilidad de los datos $\hat{p}_{data}$. Lo ideal sería lograr alcanzar la distribución real de los datos p_{data} pero al tener un número finito de observaciones no se puede conocer por completo.

2.5.5. Optimizadores

Una vez que se ha definido la función de costo $J(\boldsymbol{x}, \boldsymbol{\theta})$, es necesario hallar el valor óptimo de los parámetros $\boldsymbol{\theta}$. Para lograr lo anterior se hace uso de métodos de optimización, los más populares en el ámbito del aprendizaje automático son los basados en el gradiente. Se sabe que el gradiente, o derivada direccional, otorga un vector que tiene la dirección con el mayor cambio en el valor de la función a optimizar, en esta aplicación, se desea minimizar el error entre la predicción y el valor observado.

Conociendo el valor del gradiente, se puede disminuir el error de predicción al mover los valores de $\boldsymbol{\theta}$ en la dirección opuesta al gradiente, a esto se le conoce como "descenso por gradiente"(Cauchy, 1847).

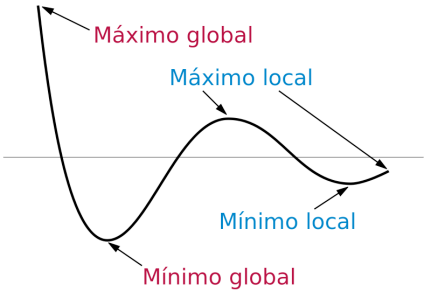
Este proceso se puede expresar con la ecuación recurrente

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{x}, \boldsymbol{\theta}) \tag{2.20}$$

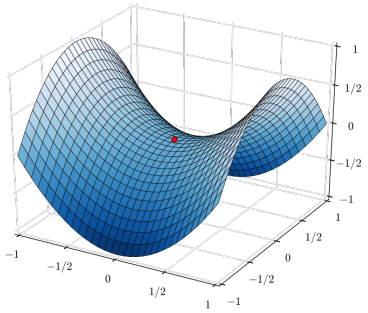
En donde:

$\boldsymbol{\theta}'$: Es el nuevo valor de los parámetros.
$\boldsymbol{\theta}$: Es el valor actual de los parámetros.
ϵ: Es la tasa de aprendizaje[15].

[14] *Negative Log Likelihood*
[15] *Learning rate*



(a) Máximos y mínimos locales y globales.[16]



(b) Punto de silla, en este caso el gradiente también es cero.[17]

Figura 2.8: Posibles casos en donde el gradiente es cero.

Cuando el gradiente es cero $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{x}, \boldsymbol{\theta})$, no se cuenta con información de la dirección en la que se debe mover, estos se conocen como puntos críticos y pueden ser uno de los siguientes tres casos:

- **Mínimo o máximo global:** Es un punto en el cual el valor de la función es mayor o menor que cualquier otro valor en su dominio.
- **Mínimo o máximo local:** Es un punto en el cual el valor de la función es mayor o menor que su valor evaluado en los puntos de su vecindad.
- **Punto de silla:** Es un punto en el cual el valor de la función es mayor que el valor de una parte de la vecindad y menor en la otra parte.

En la figura 2.8a se pueden observar los máximos y mínimos locales y globales. En la figura 2.8b se observa el punto de silla.

En el aprendizaje profundo se trabajan con funciones de dimensiones muy altas, por lo que encontrar el mínimo global es una tarea prácticamente imposible, esto hace surgir la pregunta ¿Por qué los algoritmos funcionan? La respuesta es que muchas veces el desempeño del modelo es suficientemente bueno para la aplicación aun cuando la solución a la que se llegó sea un mínimo local.

Descenso por gradiente estocástico (SGD)

En el aprendizaje profundo se utilizan bases de datos gigantescas, por lo que el cómputo del descenso por gradiente es muy costoso. Si se supone que el gradiente es una esperanza, se puede decir que el valor del gradiente calculado con todos los ejemplos de la base de datos puede aproximarse usando un subconjunto más pequeño. A este subconjunto se le llama *minibatch*, que tiene un tamaño m' y está compuesto por los ejemplos $\mathbb{B} = x^{(1)}, x^{(2)}, ..., x^{(m')}$ que fueron tomados de manera uniforme del conjunto de entrenamiento. El tamaño m' del *minibatch* es mucho más pequeño que el tamaño m.

El cálculo del gradiente estimado queda como:

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x^{(i)}}, y^{(i)}, \boldsymbol{\theta}) \tag{2.21}$$

Y el algoritmo de descenso por gradiente estocástico (SGD)[18] queda como:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g} \tag{2.22}$$

2.5.6. Otros optimizadores

Existen otros optimizadores, estos en su mayoría son modificaciones del SGD, los más importantes son los siguientes:

- **SGD con momento:** Este optimizador propuesto por Rumelhart, Hinton y Williams (Rumelhart, Hinton & Williams, 1986) presenta un nuevo concepto, el de momento. El momento al igual que en física es una propiedad del movimiento de un objeto. Imagínese una pelota que va cayendo en una colina, esta tiene cierta velocidad en cada posición en la que se encuentra, el momento del gradiente es algo equivalente, se mantiene rastro del valor del gradiente en el instante anterior. Esta modificación ayuda a que el entrenamiento converja más rápido. El algoritmo queda de la siguiente manera:

$$V_t = \beta V_{t-1} + (1 - \beta)\boldsymbol{g} \tag{2.23}$$

[17] De User:Georg-Johann - http://commons.wikimedia.org/wiki/File:Extrema_example_de.svg, Dominio público, https://commons.wikimedia.org/w/index.php?curid=15802407

[17] De Nicoguaro - Trabajo propio, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=20570051

[18] *Stochastic Gradient Descent*

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon V_t \tag{2.24}$$

En donde:

$\boldsymbol{g}$: Es el gradiente estocástico.
V_{t-1}: Es el valor de la velocidad en el paso anterior.
V_t: Es el valor de la nueva velocidad.
β: Es un parámetro que dice la proporción que es recordada de la velocidad anterior, se encuentra entre cero y uno.

- **SGD con momento de Nesterov:** Este optimizador tiene una ligera modificación respecto al momento original, en lugar de calcular el gradiente en el punto en el que se encuentra el algoritmo, se "observa" un punto en la dirección que apunta el momento y se calcula el gradiente en ese nuevo punto. En la figura 2.9 se aclara esta idea. El

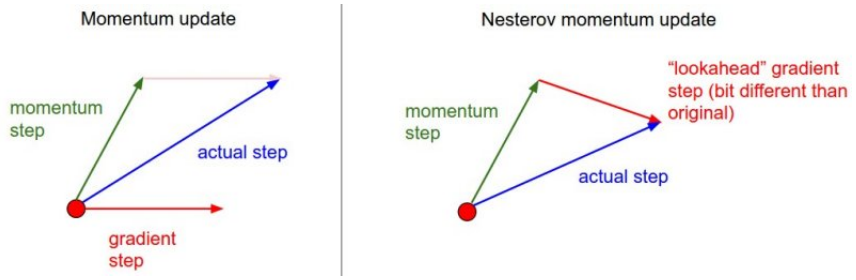


Figura 2.9: Comparación del SGD con momento contra el SGD con momento de Nesterov. [19]

algoritmo queda de la siguiente manera:

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x^{(i)}}, y^{(i)}, \boldsymbol{\theta} - \beta V_{t-1}) \tag{2.25}$$

$$V_t = \beta V_{t-1} + (1 - \beta)\boldsymbol{g} \tag{2.26}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon V_t \tag{2.27}$$

Se puede notar que la única modificación es que ahora el gradiente estocástico se calcula en el punto $\boldsymbol{\theta} - \beta V_{t-1}$.

[19] Del curso CS231n de Stanford Obtenida de: http://cs231n.github.io/neural-networks-3/#sgd

- **Adagrad:** Propuesto por Duchi (Duchi, Hazan & Singer, 2011), el algoritmo de gradiente adaptativo [20] propone una solución al problema de tener que elegir manualmente la tasa de aprendizaje ϵ. Este optimizador escala de manera dinámica el valor de la tasa de aprendizaje para cada dimensión.

- **RMSProp**: Al igual que Adagrad, este algoritmo adapta el valor de la tasa de aprendizaje dependiendo del promedio de los gradientes. Esto le permite desempeñarse adecuadamente aun en problemas ruidosos. Este algoritmo fue propuesto por Tieleman y Hilton (Tieleman & Hinton, 2012) en un curso masivo en Coursera.

- **Adam:** Propuesto por Kingma y Ba (Kingma & Ba, 2014), ha demostrado ser el mejor optimizador hasta la fecha. Además de mantener rastro del primer momento de los gradientes (el promedio) también mantiene rastro del segundo momento (la varianza).

2.5.7. Propagación hacia atrás

Como se mencionó en la sección 2.5.5 para corregir el valor de los parámetros es necesario calcular el gradiente de la función de costo respecto a cada uno de los parámetros. La manera en la que esto se logra es propagando el error a través de todas las neuronas en todas las capas de la red, el algoritmo que logra esto es conocido como propagación hacia atrás[21]. Recordando como se propuso la arquitectura de la red neuronal, se sabe que cada unidad o neurona calcula una activación y que para fines prácticos se nombrará como $a_i^{(L)}$ siempre que sea la salida de una capa oculta, el superíndice indica el número de la capa y el subíndice el número de neurona de la capa.

$$\boldsymbol{a^L} = \phi(\boldsymbol{\theta}^{(L)} a^{L-1} + b^L) \tag{2.28}$$

En donde:

$\boldsymbol{a^{(L)}}$: Es el vector de activaciones de la capa L.
ϕ: Es la función de activación.
$\boldsymbol{\theta}^{(L)}$: Es la matriz de pesos de las neuronas de la capa L.
$\boldsymbol{a}^{(L-1)}$: Es el vector de activaciones de la capa anterior $L-1$.
$\boldsymbol{b^{(L)}}$: Es el vector de sesgos de la capa L.

[20] *Adaptative Gradient*
[21] *Backpropagation*

Para calcular la derivada del costo respecto a cada uno de los parámetros se debe contar con alguna manera de reconstruir las operaciones que se hicieron y con ello encontrar la derivada $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_{jk}^{L}}$. Para eso se hace uso de un grafo que mantiene rastro de todas las operaciones realizadas, en ese diagrama cada uno de los nodos es una variable o constante, y las operaciones que se realizan son las aristas que unen a los nodos.

El algoritmo de propagación hacia atrás permite que con base en el grafo de las operaciones se pueda calcular la derivada de un nodo respecto a otro y con ellos calcular el gradiente del costo respecto a cada uno de los parámetros.

Como se mencionó en la sección 2.5.3 cada capa de la red puede verse como una función, por lo que el vector de salida $\boldsymbol{y}$ de la última capa L, es una composición de funciones, de la capa L hasta la capa de entradas $\boldsymbol{y} = f^{(L)}(f^{(L-1)}(...(f^{(1)}(\boldsymbol{x}))))$. Por lo que si se calcula el gradiente del vector de costo J con respecto a la matriz de parámetros se obtiene la matriz ∇J. Esta matriz otorga todas las derivadas necesarias para corregir los parámetros de la matriz $\boldsymbol{\Theta}$.

2.5.8. El problema del desvanecimiento y explosión del gradiente

Dada la manera en la que se propaga el gradiente a través de la red, es muy común que el valor de este tienda a cero conforme se propaga hacia atrás, esto es un serio problema ya que no habría una actualización de los valores en las primeras capas, y por ende no habría un aprendizaje. Esto ocurre cuando se obtienen gradientes menores a 1.0, al multiplicarlo recursivamente este valor tiende a cero, y como se tiene un límite en la representación de estos números en la computadora, se vuelve igual a cero.

El problema de la explosión de los gradientes es análogo, cuando se tienen gradientes con valores mayores a 1.0 estos crecen sin cota al multiplicarse en redes muy profundas, obteniendo valores que tienden a infinito y terminan desbordando la memoria de representación de la computadora, y como consecuencia tampoco se logra un buen entrenamiento. Para dar solución a estos problemas se puede hacer uso de inicializaciones específicas en el valor de los pesos en lugar de hacerlo de forma completamente aleatoria. La idea principal es suponer que el valor de los pesos proviene de una distribución de probabilidad gaussiana con media cero y varianza proporcional a $\frac{1}{n}$, en donde n es el número de entradas a la neurona. Las dos más populares son:

- Inicialización de He: Propuesta por Kaiming He (He, Zhang, Ren & Sun, 2015c), supone que se usa una función de activación ReLU, la varianza queda como: $Var(\boldsymbol{\theta}_i^L) = \frac{2}{n^{L-1}}$. En donde $\boldsymbol{\theta}_i^L$ es el vector de pesos de una neurona i en la capa L, y n^{L-1} es el número de neuronas en la capa anterior.
- Inicialización de Xavier: Propuesta por Xavier Glorot y Joshua Bengio (Glorot & Bengio, 2010), supone que si se usa una función de activación tangente hiperbólica (equivalente a una sigmoide desplazada respecto al eje de las ordenadas), la varianza queda como: $Var(\boldsymbol{\theta}_i^L) = \frac{1}{n^{L-1}}$.

2.6. Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo específico de redes neuronales que están especializadas en procesar información que se encuentra representada en forma de celdas, como son las imágenes. Reciben su nombre ya que son hasta cierto punto análogas a la operación de convolución. Gooffellow en su libro las describe como:

> "Las redes neuronales convolucionales son redes neuronales simples que usan convolución en lugar de la multiplicación general de matrices en al menos una de sus capas."(Goodfellow et al., 2016, p. 326)

2.6.1. Convolución

La convolución se define como la siguiente operación en variables continuas:

$$s(t) = \int x(a)w(t-a)da \tag{2.29}$$

La convolución generalmente se denota con un asterisco.

$$s(t) = (x * w)(t) \tag{2.30}$$

Para variables discretas queda de la siguiente manera:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.31}$$

En donde según la terminología de redes neuronales convolucionales:

x: Es conocida como la entrada.
w: Es conocido como el kernel.
$s(t)$: Es conocido como el mapa de características.

En aprendizaje automático la entrada es usualmente un arreglo multidimensional de datos, y el *kernel* es un arreglo multidimensional de parámetros que son aprendidos por el algoritmo, la red neuronal en este caso. Si se asume que estas funciones son cero en todos sus puntos excepto en la sección en donde se encuentran guardados los valores de interés, la suma infinita se vuelve una suma finita definida en las dimensiones del arreglo. Al estar trabajando con imágenes, se tiene cuando menos dos dimensiones, por lo que para una imagen I bidimensional y un *kernel* o núcleo K bidimensional, se obtiene:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.32}$$

Como la convolución es conmutativa se puede escribir:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.33}$$

Se ha definido la operación convolución pero probablemente no queda del todo claro que es lo que hace, para explicarlo de mejor manera se puede pensar en el *kernel* K como una matriz que se desplaza en las dimensiones m y n de la matriz de entrada I, al hacer esto se obtiene una matriz S llamada mapa de características que es una representación filtrada de la matriz de entrada I. En la figura 2.12 puede observarse claramente la operación convolución entre la matriz de entrada I y el filtro K.

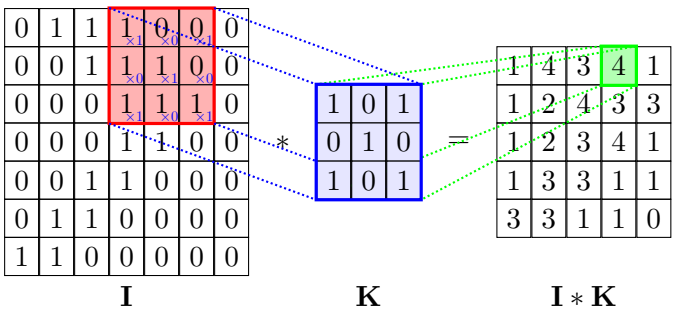


Figura 2.10: Operación convolución

2.6.2. Filtros

Como se mencionó en la sección anterior la operación convolución en procesamiento de imágenes se puede ver como una matriz que se desplaza sobre otra y genera una tercera. Esta operación ha sido ampliamente usada para generar filtros, que eliminen ruido o que hagan la imagen más nítida. En el procesamiento de imágenes estos se proponen *a priori*, dependiendo de la tarea que se desee. Algunos ejemplos se observan en la tabla 2.1.

Operación	**Filtro**	**Resultado**
Identidad	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Detección de bordes	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Enfocar	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Desenfocar	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Tabla 2.1: En esta tabla se observan algunos ejemplos de filtros, los valores son elegidos manualmente.[22]

Lo interesante es que en el contexto del aprendizaje profundo la red neuronal es la que propone el valor de cada elemento de estas matrices.

2.6.3. Capa convolucional.

Existen varias razones por las cuales es preferible usar una red neuronal convolucional en lugar de una red completamente conectada, estas son: compartición de parámetros[23], interacciones escasas[24] y representaciones equivariantes[25].

[22] De Michael Plotke - Trabajo propio, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24301122

[23] *Parameter sharing*

[24] *Sparse interactions*

[25] *Equivariant features*

En las redes neuronales completamente conectadas (FC [26] por sus siglas en íngles) todas las unidades de la capa L interactúan con las de la capa (L-1) y con la entrada X, mientras que las redes convolucionales las unidades de la capa L solo interactúa con algunos de los valores de entrada. Esto tiene una gran ventaja en costo computacional y en eficiencia estadística ya que permite a la red generalizar mejor. A esto se le conoce como interacciones escasas.

La compartición de parámetros se refiere a que al aplicar la operación convolución los parámetros del filtro son aplicados más de una vez en la imagen de entrada. Esto permite que el filtro aprenda a detectar generalidades en la imagen, como bordes.

Finalmente la representación equivariante hace referencia a que si se hace variar la entrada, la salida varía de la misma manera. Es decir si se tiene una imagen I y se le aplica una función que desplaza todos los pixeles una unidad a la derecha, $I' = g(I)$ es la nueva imagen desplazada. La función es equivariante si al aplicar la convolución sobre I' se obtiene el mismo resultado que al aplicar la convolución sobre I y posteriormente se le aplica el desplazamiento con la función g. La convolución es equivariante bajo algunas transformaciones g, esto es útil porque permite buscar características comunes que pueden aparecer en distintas partes de la imagen, como bordes.

Como se ve en las imágenes de la tabla 2.1, cada filtro otorga una nueva imagen a la salida, un mapa de características asociada al filtro.

La red convolucional se puede representar como un arreglo tridimensional, en donde cada neurona está conectada únicamente a una región local de la entrada. Al igual que las redes FC que se vieron en la sección 2.5.3 cada conexión tiene un peso o parámetro asociado, pero en lugar de que sea diferente para cada neurona, este es compartido en todas las neuronas. Para poder visualizar las conexiones en la red neuronal, se puede apoyar en la figura 2.11. En esta figura se puede observar como cada neurona solo tiene acceso a una sección de la imagen. La red convolucional se representa como un arreglo tridimensional de dimensiones $h \times w \times d$.

En esta figura la entrada es una imagen RGB de 32×32 pixeles, el que sea RGB implica que tiene tres canales, uno por cada color (rojo, verde y azul). Es por eso que sus dimensiones son $32 \times 32 \times 3$. Cada neurona se conecta con

[26] *Fully Connected*

[27] Obtenido del curso CS231n de Stanford en http://cs231n.github.io/assets/cnn/depthcol.jpeg modificado por el autor.

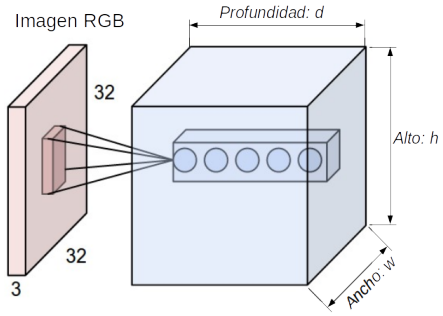


Figura 2.11: Ejemplo de la representación tridimensional de la red convolucional. [27]

solo una sección de la imagen, a esto se le conoce como campo receptivo[28], en otras palabras, las dimensiones del filtro. La red neuronal convolucional se observa como el prisma rectangular azul, en donde las dimensiones w y h coinciden con las dimensiones de los mapas de características de cada filtro, y la profundidad d es el número de filtros que se aprenderán en la red, a esto también se le conoce como volumen de salida.

Los filtros tienen cierta profundidad que necesariamente es congruente con el de la imagen de entrada, en la figura 2.12 se puede visualizar paso por paso la operación. El filtro $w1$ tiene un campo receptivo de 3, y una profundidad de 3, es decir, es un tensor de dimensiones $3 \times 3 \times 3$. En la misma figura puede verse como la operación elemento a elemento del tensor $w1$ con la imagen, más el termino de sesgo $b1$ resulta en un escalar -2 que es la entrada a una neurona del volumen de salida.

Para terminar de comprender la operación convolución y poder determinar el tamaño del volumen de salida en las redes neuronales convolucionales es necesario hablar de algunos conceptos:

- **Profundidad** d**:** Este hiperparámetro indica el número de filtros que se usarán, cada uno aprenderá a detectar diferentes características de la imagen. Es importante mantener en mente que cada filtro es un tensor de dimensiones $F \times F \times C$ en donde F es el tamaño del campo receptivo (generalmente se trabaja con filtros cuadrados) y C es el

[28] *Receptive Field*
[29] Obtenido del curso CS231n de Stanford en http://cs231n.github.io/assets/cnn/.

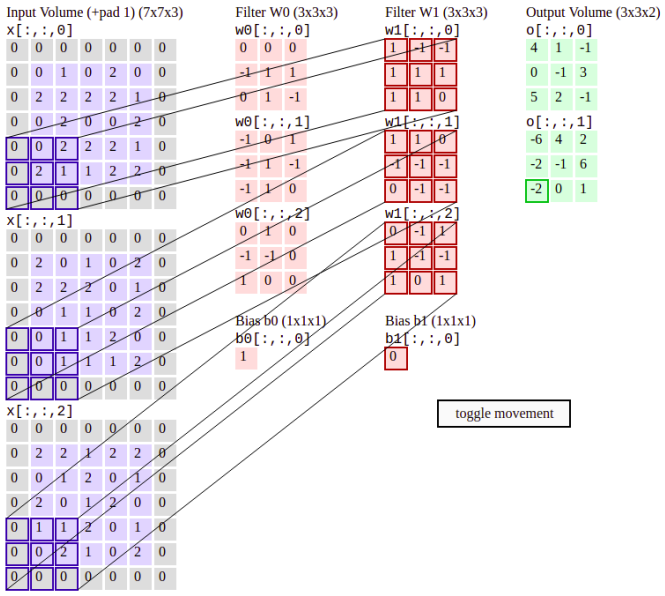


Figura 2.12: Ejemplo de la manera en la que se realiza la operación convolución. [29]

número de canales de la entrada, para una imagen RGB este es $C = 3$.

- **Paso**[30]**:** Es el número de pixeles que se desplaza el filtro, generalmente es de 1 pixel o 2 pixeles. Este hiperparámetro repercute directamente en el tamaño del volumen de salida, específicamente en el tamaño del mapa de características.

- **Relleno con ceros**[31]**:** Para poder mantener el tamaño de la imagen de entrada en el mapa de características en algunas ocasiones resulta necesario añadir elementos en los bordes de la matriz de entrada, generalmente estos elementos son ceros. A esto se le conoce como relleno de ceros.

Los conceptos definidos son los que permiten determinar el número de neuronas que estarán dentro del volumen de salida, es decir, las dimensiones del volumen de salida. La expresión que otorga estas dimensiones es la 2.34, en donde si se supone que la entrada y el filtro es cuadrado, y que el paso es el mismo a lo ancho y a lo alto de la entrada, entonces el ancho w y el alto h

[30] *Stride*
[31] *Zero padding*

del volumen de salida son iguales.

$$Dimension_{Salida} = \frac{(W - F + 2P)}{S} + 1 \tag{2.34}$$

En donde:

W: Es el tamaño de la entrada, ya sea a lo ancho o a lo alto.
F: Es el paso con el que el filtro se desplaza.
P: Es el número de ceros en los bordes. Por ejemplo para una entrada:

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Un relleno de ceros $P = 1$ equivale a:

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Un relleno de ceros $P = 2$ equivale a:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

.

2.6.4. Capa de "Pooling"

Esta capa de la red, capa *Pooling*[32]. tiene la función de disminuir la dimension de la red, con esto se logra reducir el número de parámetros y por ende el costo computacional, y con ello también evitar el sobreajuste. Esta

[32]No se realiza la traducción de este termino ya que no se encontró una que mantenga el significado original.

capa lo que hace es tomar los mapas de características del volumen de salida de la red convolucional y desplazar una ventana en ellos de manera parecida a los filtros de convolución, pero en lugar de realizar la multiplicación y luego sumarlos, aplica una operación sobre ellos. La operación más común es la de valor máximo, esta toma el elemento más grande dentro de la ventana y lo coloca como elemento de un nuevo volumen de salida. Otras operaciones son posibles como el promedio o la norma L2, pero han caído en desuso ya que la operación valor máximo ha demostrado mejores resultados en la práctica. La ecuación 2.34 sigue siendo válida para calcular el tamaño de la salida de la capa. Un ejemplo de esta operación se puede observar en la figura 2.13. En esa figura se tiene una entrada $W = 4$, un campo receptivo $F = 2$, un paso $S = 2$ y un relleno de ceros $P = 0$. Por lo que la dimensión de salida es: $W_{sal} = \frac{4-2}{2} + 1$

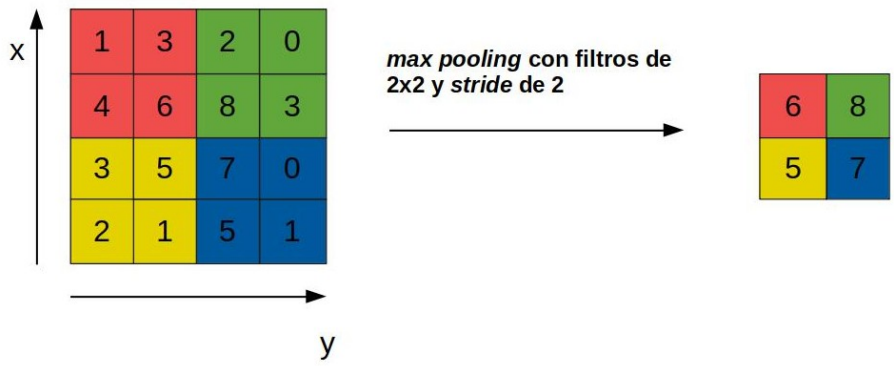


Figura 2.13: Ejemplo de la operación *max pooling*.

2.6.5. Arquitectura base

En resumen, la arquitectura básica de una red neuronal convolucional (véase la figura 2.14) está compuesta por la capa convolucional (notese que Los pesos en la capa convolucional son los mismos en cada neurona), la capa de activaciones no lineales (generalmente ReLU), la capa de *pooling* y finalmente una red FC que en el caso de clasificación, predice las etiquetas a la que pertenece la entrada.

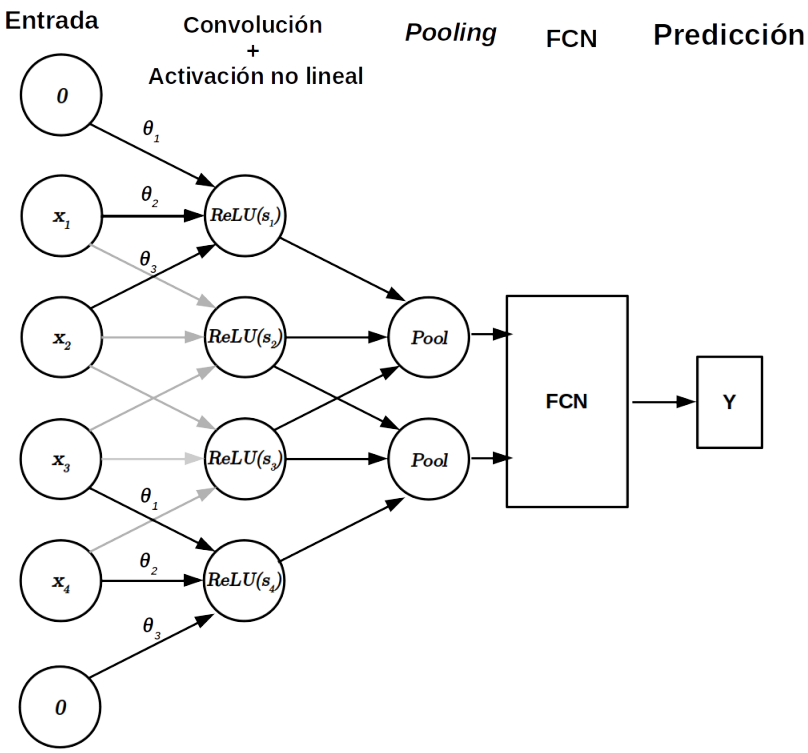


Figura 2.14: Arquitectura básica de una red convolucional.

2.7. Redes neuronales residuales (ResNet)

Una arquitectura de red ligeramente diferente a las redes convolucionales canónicas, mencionadas en la sección 2.6, son las redes neuronales residuales (He, Zhang, Ren & Sun, 2015a), estas tienen la característica de propagar la información de las primeras capas a capas más profundas saltándose capas intermedias. Esto les permite propagar la información de capas con mayor resolución, es decir con mapas de características de dimensiones mayores a capas con resoluciones menores. Con esta propuesta se puede evitar el problema del desvanecimiento del gradiente, permitiendo entrenar redes cada vez más profundas. La idea que ellos proponen es que es más fácil entrenar una red que tiene como referencia la entrada, en lugar de cero. Es decir, en lugar de tratar de encontrar la función ideal $\mathcal{H}(x)$, se propone hallar la

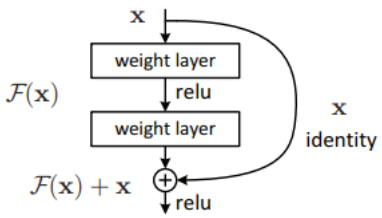


Figura 2.15: Bloque básico de construcción de las capas residuales. (He, Zhang, Ren & Sun, 2015a)

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Figura 2.16: Tasa de error en porcentaje de clasificación en la base de datos ImageNet en el conjunto de validación. (He, Zhang, Ren & Sun, 2015a)

función residual $\mathcal{F}(x)$. Esta idea se plantea en la ecuación 2.35.

$$\mathcal{F}(x) := \mathcal{H}(x) - x \tag{2.35}$$

Por lo que la función ideal queda como $\mathcal{H}(x) = \mathcal{F}(x) + x$. Esta idea está representada en la figura 2.15. Kaiming demuestra que efectivamente el entrenamiento mejora haciendo uso de esta modificación. Y se logra trabajar con redes más profundas evitando los problemas de desvanecimiento y explosión del gradiente. Estos resultados se observan en la figura 2.16.

2.8. Red piramidal de características

Una red convolucional, como se mencionó en la sección 2.6, aprende a representar filtros que le ayudan a encontrar características importantes de

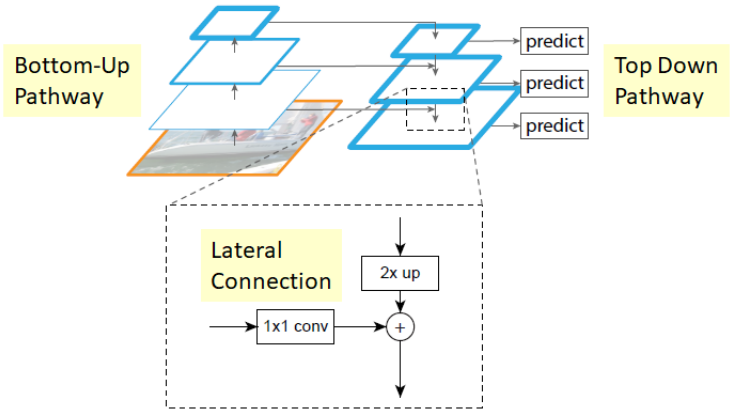


Figura 2.17: Arquitectura de la red FPN y diagrama de las operaciones entre capas. (Lin et al., 2016)

las imágenes de interés. Por la manera en la que funcionan, tienen una arquitectura piramidal intrínseca, es decir la imagen de entrada es de una resolución dada, y cada vez que se le aplique la operación convolución y el *pooling* disminuye su resolución, tomando en consideración que aumenta su profundidad (número de canales). Una propiedad de las redes convolucionales es que los primero filtros aprenden representaciones muy generales y de bajo nivel pero de alta resolución, mientras que los filtros de las últimas capas de convolución aprenden representaciones específicas de los objetos de interés pero a resoluciones bajas. Aprovechando esta característica los autores de la arquitectura de Red piramidal de características [33] (FPN) (Lin et al., 2016) proponen una manera de propagar la información de las capas más profundas a las primeras capas de alta resolución.

La manera en la que lo logran, es creando una red paralela que se encarga de la predicción, las operaciones que se llevan a cabo pueden observarse en la figura 2.17. La operación consiste en tomar una capa de la pirámide derecha, realizar una expansión [34] al doble de su resolución, haciendo uso del algoritmo más simple, vecino más cercano. Al resultado se le suma la capa equivalente de la pirámide izquierda, pero tomando en cuenta que se realiza una convolución de 1×1 a la capa, esto logra que se reduzca el número de canales a uno, haciendo la adición entre ambas capas posible. Posteriormente se repite el proceso hasta llegar a la capa de más alta resolución. Una

[33] *Feature Pyramid Network*
[34] *Upsampling*

característica de esta arquitectura es que cada capa hace una predicción, a diferencia de las redes convolucionales estándar en las que solo la capa más profunda realiza la predicción.

2.8.1. Bases de datos para la clasificación de objetos

Como se ha mencionado en reiteradas ocasiones, el uso de redes neuronales implica contar con una base de datos enorme para obtener resultados satisfactorios, es por ello que se han realizado varios proyectos de colaboración para tener datos confiables, estandarizados y de buena calidad con los cuales comparar e implementar modelos de aprendizaje profundo. Dos de los proyectos más importantes en la clasificación de objetos son:

ImageNet

ImageNet (Deng et al., 2009) es una base de datos enorme a la que se puede acceder fácilmente. Está organizada de acuerdo a la jerarquía de la base de datos WordNet. En WordNet, un concepto rico en significado, generalmente sustantivos, recibe el nombre de *synonym set* (synset). El objetivo de ImageNet es proveer de aproximadamente 1000 imágenes por cada synset, manualmente etiquetadas y revisadas. Al momento el proyecto cuenta con aproximadamente catorce millones de imágenes anotadas, con más de veinte mil categorías. El proyecto fue organizado como una cooperación entre las universidades de Princeton, Stanford y otras universidades estadounidenses.

El proyecto también organiza el desafió anual *The ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) (Russakovsky et al., 2015), para este desafío se utiliza una base de datos reducida a solo mil categorías independientes. El ILSVRC se ha vuelto uno de los referentes para comparar el estado del arte en arquitecturas de detección y clasificación de objetos.

Common Objects in Context (COCO)

De forma parecida a ImageNet, COCO (Lin et al., 2014) tiene como objetivo proveer una base de datos confiable, de fácil acceso y de alta calidad a la comunidad de visión por computadora. La principal diferencia es que en lugar de tomar como referencia los synsets de WordNet, COCO toma como referencia noventa y un etiquetas de objetos fáciles de reconocer en escenas complejas del día a día (figura 2.18), en la investigación mencionan

Figura 2.18: Comparación entre imágenes, las del inciso (c) son las que se tomaron con mayor importancia para la creación de la base de datos COCO, ya que tienen mayor complejidad e involucran más de un objeto. (Lin et al., 2014)

que de esta manera es más fácil para los algoritmos generalizar. El proyecto fue realizado principalmente por Microsoft.

La base de datos cuenta con más de trescientas mil imágenes, las cuales tienen aproximadamente un millón y medio de instancias etiquetadas.

2.9. Transferencia de aprendizaje

Una de las mayores limitantes del aprendizaje profundo es la excesiva cantidad de imágenes necesarias para poder entrenar un sistema y obtener un buen desempeño. Se ha demostrado que una manera de sortear este problema es hacer uso de la transferencia de aprendizaje (Yosinski, Clune, Bengio & Lipson, 2014).

Esta técnica consiste en hacer uso de los pesos de una red entrenada en una base de datos gigantesca como son COCO e ImageNet, y a partir de ellos entrenar un modelo especializado en alguna tarea. La justificación de esta técnica es que la red aprende ciertas características generales de las imágenes y conforme se profundiza en las capas convolucionales estas características se vuelven más específicas a la tarea. Con lo anterior se puede tomar una red de clasificación entrenada previamente y "desconectarla" de la red FC, "conectarla" a una nueva y solamente entrenar la red FC con las capas más profundas de convolución, dejando fijos los pesos de las primeras capas de convolución. En la figura 2.19, se tiene una red convolucional(bloques 3D verdes) con su respectiva red FC (bloques en 2D azules) como clasificador entrenada en la tarea A. Esta técnica permite entrenar una nueva red cuyas capas de convolución sean iguales y con los pesos calculados en la

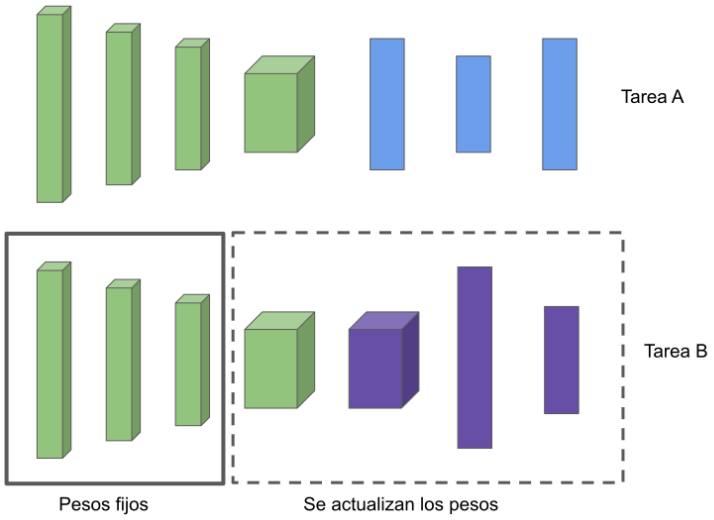


Figura 2.19: Ejemplificación de la transferencia de aprendizaje.

tarea A y un nuevo clasificador (en morado), que puede contener nuevas capas de convolución, especifico para la tarea B. La manera de entrenarlo es congelar los pesos de la red convolucional a excepción de las capas más profundas(recuadro con línea continua), y modificar los del resto de la red (recuadro con línea punteada).

2.10. Aumento de datos

Otra técnica que ayuda a mejorar el desempeño de la red sin necesidad de una enorme cantidad de datos de entrenamiento es el aumento de datos [35]. Esta consiste en procesar las imágenes de tal manera que parezcan ser imágenes nuevas. Este procesamiento puede ser tan sencillo como la rotación de las imágenes o la traslación de los objetos, crear oclusión, a técnicas más complejas como generar imágenes nuevas con redes neuronales (Shorten & Khoshgoftaar, 2019). En la figura 2.20 se observan algunos ejemplos de esta técnica.

[35] *Data augmentation* en inglés

[36] Data augmentation-assisted deep learning of hand-drawn partially colored sketches for visual search - Scientific Figure on ResearchGate. Disponible en: https://www.researchgate.net/figure/Data-augmentation-using-semantic-preserving-transformation-for-SBIR_fig2_319413978

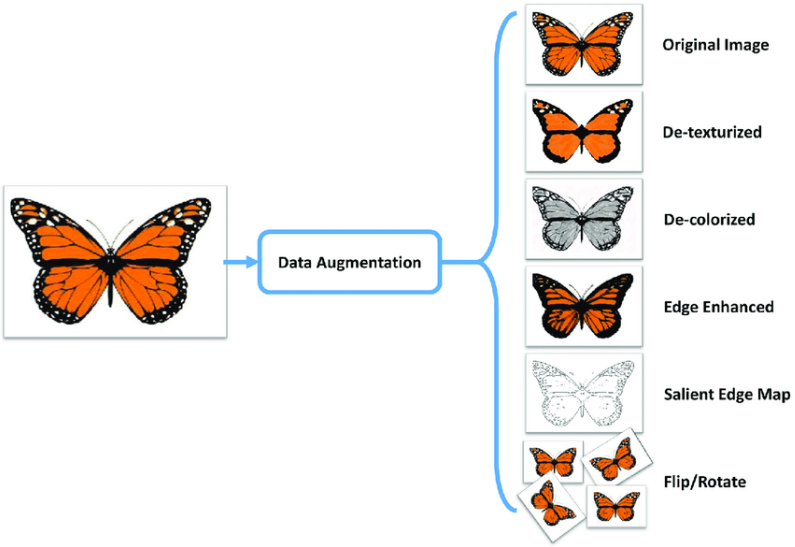


Figura 2.20: Ejemplos de procesamiento de imágenes para el aumento de datos[36].

2.11. Mask R-CNN

Con base en todos los conceptos que se han mencionado a lo largo de este capitulo, se puede presentar la arquitectura que se usa en este trabajo, esta se llama Mask R-CNN (He, Gkioxari, Dollár & Girshick, 2017b)y fue desarrollada principalmente por Facebook con su laboratorio de investigación en inteligencia artificial *Facebook Artificial Intelligence Research* (FAIR). Esta arquitectura es el resultado de cuatro iteraciones previas y es, hasta el momento de escritura de este trabajo, el estado del arte en la segmentación de instancias. Antes de profundizar en el funcionamiento de la arquitectura, se hablará un poco sobre las iteraciones que permitieron llegar a ella.

2.11.1. Antecedentes de la arquitectura Mask R-CNN

Para cumplir con el objetivo de la detección de objetos, no es posible utilizar directamente una arquitectura convolucional estándar, ya que no se sabe de antemano cuantos objetos se deben detectar en la imagen, por lo que el tamaño de la salida es variable. Una manera de solucionar este problema puede ser seleccionando regiones de interés dentro de la imagen, y

[Accesado el 10 de Enero de 2020]

posteriormente alimentar una red convolucional con ellas para determinar lo que hay dentro. El problema yace en que la cantidad de regiones de interés puede ser exorbitantemente grande, logrando que el costo computacional sea imposible. Las arquitecturas que se mencionan a continuación tratan este problema.

R-CNN

Una solución al problema de selección de las regiones de interés es utilizar un algoritmo llamado búsqueda selectiva [37] (Uijlings, van de Sande, Gevers & Smeulders, 2013), este consiste en encontrar 2000 regiones de interés que luego se alimentan a la red convolucional para su clasificación. A esta arquitectura se le bautizó como R-CNN *Region Convolutional Neural Network* (R. Girshick et al., 2014). La arquitectura se puede observar en la figura 2.21

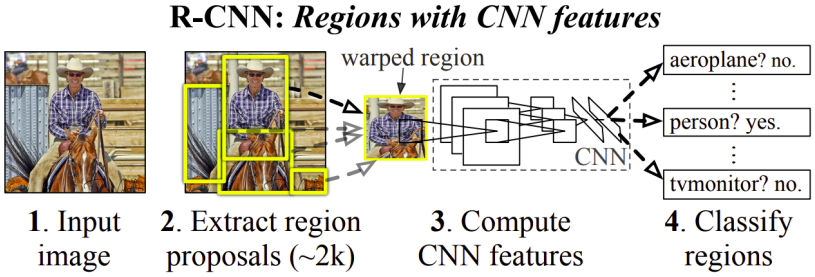


Figura 2.21: Esquema de la arquitectura R-CNN (R. Girshick, Donahue, Darrell & Malik, 2014).

El algoritmo de búsqueda selectiva es relativamente sencillo, primero se hace una segmentación con una enorme cantidad de propuestas de regiones de interés, posteriormente se agrupan las regiones de interés similares para obtener ROIs más grandes, estas medidas de similaridad son similaridad de color, similaridad de textura, similaridad en tamaño de los cuadros delimitadores y compatibilidad de forma, es decir, si es posible agruparlos o no. El proceso se detiene cuando se obtienen dos mil regiones propuestas con las que se alimentara la red convolucional.

Una de las características más interesantes de esta arquitectura es que cuenta con una etapa de corrección del cuadro delimitador. Es decir que si

[37] *Selective search*

en una predicción aparece una cara cortada a la mitad, el sistema es capaz de corregir la predicción para mejorar la clasificación.

A pesar del gran avance que esta arquitectura presentaba en el área, mantenía un desempeño lejos del ideal para la tarea. Las desventajas principales eran que la red debía aplicar la operación convolución a las 2000 regiones propuestas y que el algoritmo *selective search* no aprendía a mejorar en sus predicciones. Otro problema significativo es que la red no podía entrenarse en conjunto, se debía entrenar primero la parte convolucional, luego tomarla y entrenar únicamente los clasificadores.

Fast R-CNN

Para solucionar algunos de lo problemas anteriores, el autor de R-CNN propone la arquitectura Fast R-CNN (R. B. Girshick, 2015). La principal diferencia es que ahora en lugar de aplicar la convolución a cada una de las regiones propuestas, se aplica una sola vez a la imagen original obteniendo el mapa de características y las regiones propuestas se proyectan en este mapa de características. La arquitectura puede observarse en la figura 2.22.

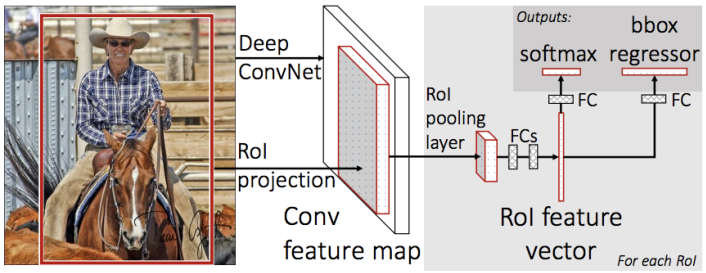


Figura 2.22: Esquema de la arquitectura Fast R-CNN (R. B. Girshick, 2015).

Una vez que se tienen todas las regiones de interés proyectadas en el mapa de características, pasan por una capa de *pooling* que los transforma a una dimension $H \times W$ fija. Posteriormente cada una de estas ROI transformadas alimentan a una red FC de clasificación y a una red FC de corrección del cuadro delimitador.

Esta arquitectura significó un gran avance con respecto a la anterior (R-CNN), ya que disminuyó de forma considerable el procesamiento, esto se debió a que solo se tenía que calcular una vez la convolución en la imagen original. También permitió que la red se entrenara de principio a fin desde

cero. A pesar de la mejoría en desempeño, mantenía un gran cuello de botella, el algoritmo de búsqueda selectiva no mejoraba con el entrenamiento.

Faster R-CNN

Para solucionar el problema de la propuesta de regiones, Shaoqing Ren propone la arquitectura Faster R-CNN (Ren et al., 2015) que unifica todas las tareas necesarias para la detección de objetos en una sola red neuronal, es decir, logra que la propuesta de regiones sea realizada por la red. El esquema de la arquitectura puede observarse en la figura 2.23.

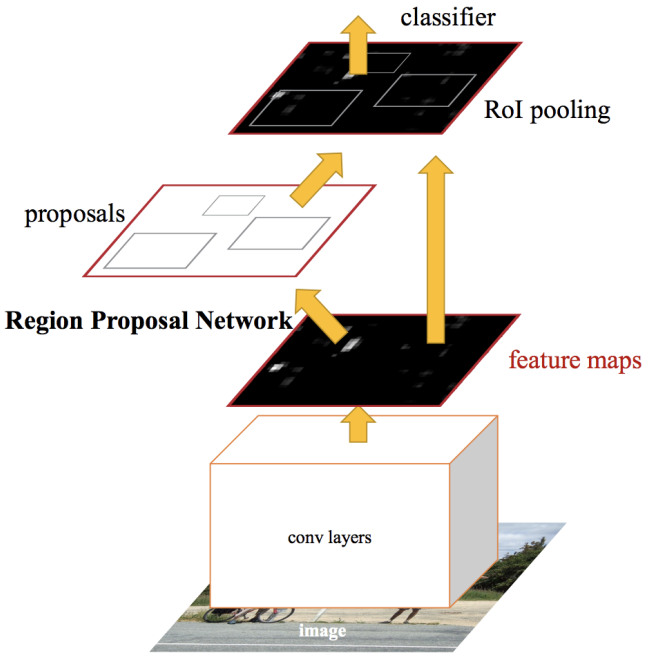


Figura 2.23: Esquema de la arquitectura Faster R-CNN (Ren, He, Girshick & Sun, 2015).

La red de propuesta de regiones [38] (RPN) se encarga de proponer los cuadros delimitadores en donde puede haber un objeto. Se profundizará en su funcionamiento en la siguiente sección (ver sección 2.11.2, en donde se habla de la modificación a la arquitectura Faster R-CNN para cumplir con el objetivo de la segmentación de instancias.

[38] *Region Proposal Network*

2.11.2. Arquitectura de Mask R-CNN

Esta arquitectura está compuesta por dos etapas, la primera observa la imagen y genera propuestas de regiones, la segunda etapa clasifica las propuestas y genera los cuadros delimitadores y las máscaras del objeto hallado, el esquema se puede observar en la figura 2.24. La arquitectura está

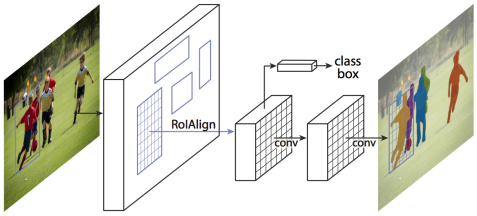


Figura 2.24: Esquema de la arquitectura Mask R-CNN (He, Gkioxari, Dollár & Girshick, 2017a).

compuesta por los siguientes módulos:

Red neuronal convolucional"*Backbone*"

Esta es una red neuronal convolucional clásica (*backbone*[39]), en este trabajo se usa una ResNet101 (He, Zhang, Ren & Sun, 2015b). Esta tiene la función de extraer el mapa de características.

Red de proposición de regiones (*Region Proposal Network* RPN)

Esta es una red neuronal convolucional que realiza un barrido en el mapa de características para encontrar objetos de interés. Cada vez que se realiza la convolución, esta tiene asociada k cuadros delimitadores que son llamados anclas [40]. Estas regiones están centradas en el filtro de la red convolucional RPN, pero sus dimensiones son con respecto a la imagen de entrada. Para determinar el número de anclas asociadas a cada punto, es necesario definir el área en pixeles y la relación de aspecto, los autores proponen 3 escalas diferentes (128^2,256^2 y 512^2) y 3 relaciones de aspecto diferentes (1:2, 1:1 y 2:1).

[39] Este término hace referencia a que es la red principal del modelo, por cuestiones de estilo y de significado se prefirió no traducirlo.

[40] *Anchors*

Como se observa en la figura 2.25, la salida de esta red son dos vectores, uno de dimensiones $2k$ que permite clasificar cada una de las anclas en objeto o no objeto. El otro vector de dimensiones $4k$ proporciona información de las coordenadas corregidas del centro del cuadro delimitador y de sus dimensiones. Dada una imagen de entrada con dimensiones $W \times H$ se proponen WHk anclas en total.

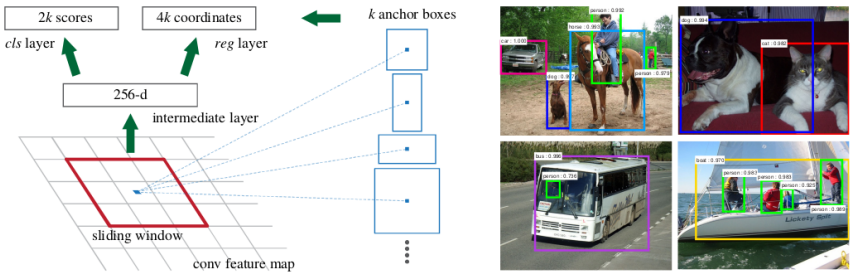


Figura 2.25: Esquema del funcionamiento de la RPN. (Ren, He, Girshick & Sun, 2015).

Usando las predicciones de la RPN, se toman las anclas que tengan mayor probabilidad de contener un objeto y se refina su posición y tamaño. En caso de que existan muchas anclas que se superpongan, se toma la que tenga mayor probabilidad y se desechan todas las demás. Las anclas que quedan son pasadas como regiones de interés (*Region Of Interest* ROI) a la siguiente etapa. Durante el entrenamiento de la red, la manera de determinar si un ancla pasa o no como región de interés es clasificándolas como positivas o negativas. Esta clasificación depende de una métrica llamada intersección sobre unión [41] (IoU), que es una manera de cuantificar el parecido entre dos áreas. Esta se calcula como el cociente entre el área de intersección y la unión de las áreas (ver figura 2.26). Este cálculo se realiza entre el área del ancla y el cuadro delimitador etiquetado manualmente. Para determinar si un ancla es positiva o negativa se toma un umbral. Las anclas que tengan un IoU mayor a 0.7 son positivas, es decir que contienen un objeto de interés, las anclas que tengan un IoU menor a 0.3 son negativas, es decir que no contienen ningún objeto de interés. Las anclas que no pertenecen a ninguna de la clasificación no se toman en cuenta para el entrenamiento.

La función de costo, respetando la notación de los autores (Ren et al.,

[41] *Intersection over union*

[42] Por Adrian Rosebrock - http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=5771856

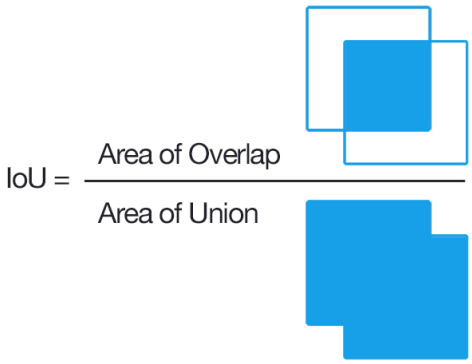


Figura 2.26: Ejemplo de la métrica intersección sobre unión. [42]

2015), se puede observar en la ecuación 2.36:

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{2.36}$$

En donde:

i: Es el índice del ancla.
p_i: Es la probabilidad predicha de que el ancla i sea un objeto.
p_i^*: Es la etiqueta real, es 1 si el ancla es positiva y cero si es negativa.
t_i: Es un vector que incluye las coordenadas parametrizadas del cuadro delimitador predicho.
t_i^*: Es un vector que incluye las coordenadas parametrizadas del cuadro delimitador verdadero.
L_{cls}: Es la función de costo de clasificación. Esta es la *log loss* (es la entropía cruzada cuando la predicción está entre cero y uno).
L_{reg}: Es la función de costo de regresión. Esta se calcula como $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$ en donde R es la función L1 suavizada [43].
N_{cls}: Es un factor de normalización, que es igual al tamaño del *mini-batch*.
N_{reg}: Es un factor de normalización, que es igual al número de posiciones de las anclas $W \times H$.
λ: Es un factor de balance entre la importancia del costo de regularización y el de clasificación.

[43] *Smooth L1*

La función L1 suavizada se observa en la ecuación 2.37 y su gráfica se observa en la figura 2.27

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{Si } |x| < 1 \\ |x| & \text{En cualquier otro caso} \end{cases} \tag{2.37}$$

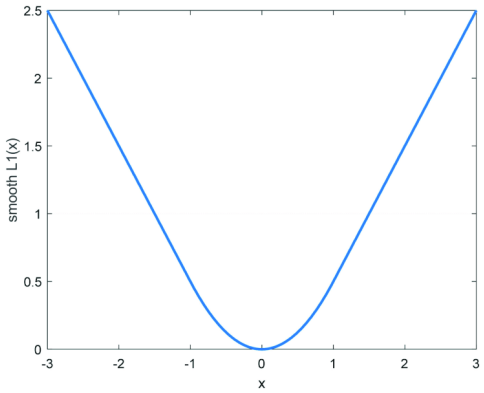


Figura 2.27: Gráfica de la función L1 suavizada. [44]

La parametrización de las coordenadas del cuadro delimitador se realiza conforme a las ecuaciones 2.38-2.41.

$$t_x = (x - x_a)/w_a, \qquad t_y = (y - y_a)/h_a, \tag{2.38}$$
$$t_w = \log(w/w_a), \qquad t_h = \log(h/h_a), \tag{2.39}$$
$$t_x^* = (x^* - x_a)/w_a \qquad t_y^* = (y^* - y_a)/h_a \tag{2.40}$$
$$t_w^* = \log(w^*/w_a), \qquad t_h^* = \log(h^*/h_a), \tag{2.41}$$

En donde x e y denotan las coordenadas del centro del cuadro delimitador y w y h el ancho y el alto de este. Las variables x, x_a y x^* son para el cuadro predicho, el ancla asociada y el cuadro delimitador verdadero (lo mismo aplica para y, w y h). Al hacer uso de redes neuronales convolucionales se puede explotar al máximo la paralelización con GPU's, los autores reportan una mejora de 2 segundos que tomaba con el método de búsqueda selectiva a 10 milisegundos con RPN.

[44] End-to-End Airplane Detection Using Transfer Learning in Remote Sensing Images - Scientific Figure on ResearchGate. Disponible en: https://www.researchgate.net/figure/The-curve-of-the-Smooth-L1-loss_fig5_322582664 [consultado el 17 enero de 2020]

Una vez que se han obtenido las regiones de interés con la RPN que se van a clasificar, los pasos siguientes son prácticamente los mismos que en la arquitectura Fast R-CNN.

Capa ROI *Align*

En la implementación original de Fast R-CNN se propone el método ROI *Pooling* y se retoma en la de Faster R-CNN, sin embargo para la arquitectura Mask R-CNN tiene varias desventajas, una de ellas es que pierde mucha información de la alineación de pixeles al realizar la operación *max pooling*. Es por ello que lo autores proponen un método nuevo llamado ROI *align* que preserva la información de la posición de los pixeles.

Para contrastar el funcionamiento de ambas capas, primero se explicará la ROI *pooling*. En esta se toman las regiones de interés propuestas por la RPN y se proyectan al mapa de características, la región asociada en el mapa de características pasa a la capa ROI *pooling* que transforma las dimensiones $h \times w$ de la región a dimensiones fijas $H \times W$, esto es debido a que el clasificador es una red FC y estas no son capaces, en principio, de trabajar con entradas de dimensiones variables. Esta capa realiza la operación de *max pooling* para convertir las dimensiones variables de entrada $h \times w$ a un arreglo de ventanas más pequeñas que tengan dimensiones $h/H \times w/W$, con esto se logra que la salida tenga dimensiones $H \times W$. La operación se realiza de manera independiente a cada canal. En la arquitectura Fast R-CNN los autores proponen que la dimensión sea $H = 7 \times W = 7$. El problema es que cuando se calcula el número de ventanas en las que hay que dividir, muchas veces es decimal, por lo que se trunca al valor entero. Esto introduce ruido en la posición de los objetos que luego puede complicar el cálculo de la máscara.

En cambio en la arquitectura Mask R-CNN, se utiliza la capa ROI *align*, que básicamente realiza lo mismo pero al dividir el mapa de características de la ROI en $H \times W$ ventanas, lo hace de forma flotante. Como esto causa que en algunos puntos se tenga que obtener el valor de un pixel que no existe (por ejemplo el valor del pixel en la posición (2.33, 1.67), se realiza interpolación bilineal para determinar su valor. Posteriormente se realiza el *max pooling* de manera análoga al método ROI *Pooling*. Los autores mostraron que este ligero cambio en la manera de transformar las ROI mejoró hasta en un 50 % la precisión de las máscaras.

Clasificador, regresor de cuadros delimitadores y red de propuesta de máscaras

Cuando se tiene el mapa de características a la salida de la red ROI *Align*, se pasan a través de dos redes FC que se encargan de proporcionar un vector de activaciones, este vector se conecta en paralelo a otras tres redes, una que se encarga de la clasificación, otra que se encarga de refinar la posición de cuadro delimitador y la tercera que propone la máscara asociada a la ROI. Una red FC se encarga de clasificar en alguna de las $K+1$ categorías (perro, gato, persona, etc. más la categoría de fondo). El vector de salida $p = (p_0, ..., p_K)$ de esta red tiene $K+1$ elementos, en donde cada uno de ellos es la probabilidad, calculada con una capa *softmax*.

La segunda red FC sirve para la corrección del cuadro delimitador. La corrección es prácticamente la misma que en la RPN, con la diferencia de que cada clase tiene un regresor asociado, es decir, que esta red tiene $4K$ unidades de salida, en donde K es el número de categorías. El vector de salida tiene la forma $t^k = (t_x^k, t_y^k, t_w^k, t_h^k)$, para las K categorías de objetos, k es el índice que indica la categoría. Este proceso se realiza para cada una de las propuestas hechas por la RPN.

La tercera y última red es una red convolucional completamente conectada, que se encarga de proponer las máscaras, esta tiene a su salida K imágenes de dimensiones $m \times m$. Es decir que para cada ROI propone una salida de dimensiones $K \times m \times m$. Las K imágenes corresponden a las K clases definidas.

Al igual que la RPN esta red tiene una función de costo que depende del costo de clasificación y del costo de regresión (ver ecuación 2.36).

$$L(p, u, t^u, v) = L_{cls}(p, u) + [u \geq 1] L_{loc}(t^u, v) + L_{mask} \tag{2.42}$$

En donde:

- p: Es el vector de salidas de la red de clasificación.
- u: Es la etiqueta real de la clase.
- v: Son las dimensiones reales del cuadro delimitador. $v = (v_x, v_y, v_w, v_h)$
- $L_{cls}(p, u)$: Es la función de costo de la clasificación.
- $[u \geq 1]$: Es una función que es igual a uno cuando $u \geq 1$ y 0 en cualquier otro caso. Por convención la categoría fondo, esta etiquetada como $u = 0$.
- $L_{loc}(t^u, v)$: Es la función de costo de la regresión.

- L_{mask}: Es la función de costo de las máscaras.

La función de costo (ver ecuación 2.42) es de multitarea, ya que incluye, clasificación (máscara y clasificación de clases) y regresión (posición de los cuadros delimitadores). La parte de clasificación se calcula con la verosimilitud logarítmica negativa. La parte de la regresión se calcula con la norma L1 suavizada (ver ecuación 2.37). La parte de las máscaras se calculan como la entropía cruzada entre la máscara real y la máscara predicha, las propuestas que no pertenecen a la clase real, no se toman en cuenta al calcular el costo.

Con todos los conceptos mencionados en este capitulo, se plantea el método de trabajo para la creación del sistema de detección. Los detalles de la implementación de la arquitectura, el entrenamiento y la evaluación se mencionan en el capitulo siguiente.

Capítulo 3

Metodología

Para la realización de este trabajo se cuenta con imágenes que fueron obtenidas de los videos tomados por Espinosa (Espinosa, 2019) durante su trabajo de campo en el petén yucateco. Para la obtención de la base de datos compuesta de por imágenes, el primer paso consistió en la extracción de los cuadros, imágenes consecutivas de los que se compone un video. El segundo paso fue la separación de las imágenes en diferentes conjuntos con base en sus características, el etiquetado manual y la separación en los conjuntos de entrenamiento, validación y prueba.

El siguiente paso fue la elección de una arquitectura de red neuronal con base en la necesidad de la segmentación de instancias, esto es para que en un futuro se puede realizar la clasificación por especies y el posible etiquetado automático de imágenes nuevas. La arquitectura seleccionada que cumple con el objetivo de segmentación de instancias y con el mejor desempeño fue la *Mask R-CNN* (He et al., 2017b). Esta arquitectura tiene ciertas características específicas, dimensión de entrada, salidas, usa una red neuronal ResNet101 como *backbone* para la extracción del mapa de características, está preentrenada en ImageNet o COCO. Con base en la arquitectura seleccionada se crearon tres modelos, cada uno con dos versiones, una preentrenada en COCO y la otra en ImageNet. Se entrenaron durante 90 épocas de 200 pasos cada uno y se mantuvo registro de los pesos durante cada época.

Finalmente se proponen dos métricas sencillas para medir el desempeño de los modelos propuestos, la precisión y la sensibilidad y se conjugan en un solo número con la métrica valor-F.

3.1. Preparación del sistema

3.1.1. Obtención de la base de datos

Para la creación de la base de datos se utilizaron videos, los cuales equivalían a 1.2 TB de información, que corresponden a 85 horas de grabación en 247 videos en formato MP4. Los videos fueron grabados en intervalos de aproximadamente 17 minutos en diferentes horarios y zonas del petén yucateco con una cámara GOPRO Hero 4, esta cámara permite grabar vídeo a una resolución de 1920×1080 pixeles a 30 cuadros por segundo. Los vídeos estaban organizados por mes de la captura, de Julio a Noviembre de 2018, y por momento del día, amanecer, medio día, tarde y noche (Espinosa, 2019).

Dado que realizar un sistema capaz de trabajar en todas las condiciones de luz (día y noche) resulta muy demandante y complejo, se optó por tomar únicamente los videos con suficiente iluminación natural. También se decidió ignorar los videos en los que se alcanza a observar el espejo de agua, esto es para evitar detectar el reflejo de los peces, como se observa en la figura 3.1.

Figura 3.1: Ejemplo de imagen con reflejo en el espejo de agua.

Con base en la tesis de Espinosa (Espinosa, 2019) se decidió trabajar con imágenes extraídas cada cinco segundos, esto permite reducir la cantidad de información a procesar así como introducir suficiente variación entre imágenes consecutivas. Con todas las consideraciones mencionadas al final se obtuvieron 11460 imágenes. En la figura 3.2 se observan algunos ejemplos de las imágenes que componen la base de datos. Nótese la variabilidad de las características del fondo.

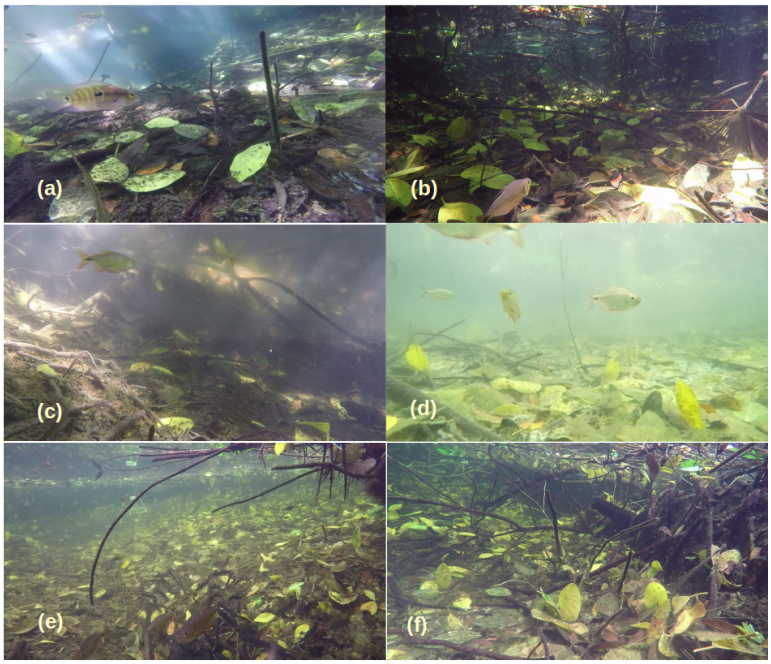


Figura 3.2: Ejemplo de las imágenes seleccionadas para la base de datos de entrenamiento.

3.1.2. Etiquetado de la base de datos

Una vez que se contó con las imágenes de los videos se segmentó manualmente el contorno de los peces (figura 3.3) de la base datos para el entrenamiento y la validación del sistema.

Con ayuda del software *VGG Image Annotator* (VIA) (Dutta, Gupta & Zissermann, 2016) desarrollado por el grupo de la Universidad de Oxford *Visual Geometry Group* (VGG), se etiquetaron las imágenes. El software VIA guarda la información de las máscaras y de las etiquetas en formato JSON. Se hizo la elección de este programa ya que el código que se utilizó (Abdulla, 2017) hace uso del formato de salida del software VIA para leer las máscaras de las imágenes.

Debido a que el objetivo es la segmentación de instancias, es necesario que las etiquetas sean del contorno de los objetos de interés, es decir un polígono que contenga al objeto. En este trabajo solo se realiza la detección de la clase "pez", se plantea en la sección 6 que en un trabajo futuro se agreguen las categorías que incluyan la especie de los peces.

Figura 3.3: Ejemplo de imagen segmentada y etiquetada con VIA.

Dada la cantidad de imágenes con las que se contaba y la necesidad de etiquetar la mayor cantidad posible para obtener un buen desempeño del sistema, el trabajo se distribuyó entre 5 personas. Gracias a ello, al final del proyecto se contó con 998 imágenes etiquetadas.

3.1.3. Distribución de la base de datos (entrenamiento, validación y prueba)

Dado que la base de datos es la materia prima de todo sistema de aprendizaje automático, es necesario administrarla de la mejor manera posible.

El primer paso consistió en separar del total de imágenes etiquetadas, imágenes que contuvieran peces que se observan claros y nítidos. A este conjunto se le llamo "*easy test*", ya que se consideró que sería más sencillo para el modelo predecir lo que se encuentra en ellas. A las imágenes etiquetadas restantes se les separó en tres conjuntos, el de entrenamiento, el de validación y el de prueba. La división se realizó conforme la regla de Pareto mencionada en la sección 2.4.5, aunque esta no se respeto del todo para la creación de la base de datos, la modificación fue que el conjunto de prueba fuese del 5 % del tamaño de la base de datos, y los elementos sobrantes se separaron con base en la regla de Pareto.

Los tamaños de las bases de datos se reportan en la tabla 3.2.

Creación del conjunto *easy test*

Como se ha mencionado en los capítulos anteriores, las condiciones en las que el modelo debe trabajar son muy ruidosas, ya sea por materia orgánica suspendida, iluminación, vegetación, etc. Si a lo anterior se le suma el hecho de que las imágenes que se extrajeron de los videos no tienen la mejor calidad, muchos de los peces se ven difuminados y sus bordes no son claros, se puede esperar que al sistema le sea complicado obtener desempeños altos. Es por lo anterior que se decidió crear una base de datos de control, en la que se puede medir el desempeño eliminando algunas de las variables mencionadas anteriormente.

Esta base de datos se llamó *easy test*, y se creó con imágenes en donde se observaran peces grandes, con poca oclusión de otros objetos u otros peces, con sus bordes nítidos y que no existieran demasiados peces en los planos más profundos de la imagen. Para obtener las imágenes se agregó una etiqueta a los archivos durante la creación de la base de datos. Esta etiqueta indicaba si la imagen contaba con las características mencionadas o no. Al final del etiquetado se contaban con 150 imágenes de este estilo. Debido a que estas imágenes se separan del conjunto de entrenamiento y validación, no era viable tomarlas todas. En lugar de tomar las 150, se separaron aleatoriamente 50. Estas imágenes estuvieron a parte en todo momento durante la creación de los modelos. En la figura 3.4 se puede observar un ejemplo de este conjunto.

3.1.4. Transferencia de aprendizaje

Como se mencionó en la sección 2.9 se puede aprovechar la transferencia de aprendizaje para reducir la cantidad de imágenes necesarias para obtener resultados interesantes. La arquitectura de red que se está utilizando permite que se hagan uso de pesos preentrenados en las bases de datos COCO y en ImageNet, estos se introducen en la parte de la red convolucional (*backbone*) que obtiene el mapa de características. Durante las pruebas se crearon dos versiones de cada modelo, una usando COCO y otra ImageNet.

3.1.5. Aumento de datos

Dado que no se contaba con una gran cantidad de imágenes etiquetadas, se aprovechó la técnica de aumento de datos (sección 2.10).

Figura 3.4: Ejemplo de las imágenes que pertenecen al conjunto *easy test*.

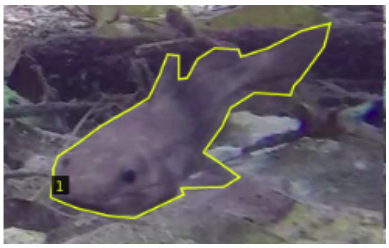
La técnica de aumento de datos que se utilizó fue relativamente sencilla, de todas las imágenes etiquetadas se extrajeron los objetos etiquetados y posteriormente se crearon nuevas imágenes repitiéndolos a lo ancho y a lo alto. Un ejemplo se puede observar en la figura 3.5b.

Se tomaron algunas consideraciones para la creación de las nuevas imágenes, la primera fue que se tomó un 3 por ciento más a lo ancho y a lo alto de los objetos para incluir información sobre el fondo. La segunda consideración fue que la imagen nueva contuviera un número entero de repeticiones cuyas dimensiones fueran menores o iguales a 1080×1080 pixeles, esto se debe a que la entrada de la red neuronal es de 1024×1024 pixeles.

3.1.6. Recursos de cómputo para el entrenamiento de la red

Para el entrenamiento de la red se está utilizando una cuenta en el clúster proporcionado por el Laboratorio Universitario de Cómputo de Alto Rendimiento (LUCAR) del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS) de la UNAM. Los recursos asignados a esta cuenta son:

- Un procesador Intel Xeon con 12 núcleos.



(a)

(b)

Figura 3.5: Ejemplo de la creación de nuevas imágenes usando aumento de datos. (a) Ejemplo de un objeto aislado. (b) Ejemplo de la imagen creada repitiendo el objeto a lo ancho y a lo alto.

- 64 GB de memoria RAM.
- Un GPU Nvidia Tesla K20.
- 2.6 TB de almacenamiento.

3.2. Implementación de Mask R-CNN

En este trabajo se hace uso de la implementación realizada por Waleed Abdulla (Abdulla, 2017). En los siguientes párrafos se habla de algunos detalles de la implementación de la arquitectura Mask R-CNN.

3.2.1. Red neuronal convolucional "*Backbone*"

Esta es una red neuronal convolucional (*backbone*[1]) para obtener el mapa de características, en este trabajo se usa una ResNet101 (He et al., 2015b). La implementación que se está usando agrega a la red convolucional una FPN (Lin et al., 2016), esta propaga la información aprendida en las últimas capas de baja resolución a las primeras capas con resoluciones más altas (véase la sección 2.8). Esto mejora en gran medida la detección de los objetos en diferentes escalas.

Es importante mencionar que todas las imágenes que se alimentan a la ResNet son transformadas a dimensiones de 1024×1024 pixeles, manteniendo su relación de aspecto y agregando ceros para cumplir con las dimensiones, como se puede observar en la figura 3.6. Esta red tiene una salida con dimensiones $32 \times 32 \times 2048$.

La FPN como se observa en la figura 2.17, da una predicción por cada nivel de la pirámide, en la implementación de Waleed Abdulla (Abdulla, 2017) se toma dinámicamente la que tenga la dimensión más cercana al objeto de interés. Los niveles de la FPN son cinco: 256×256, 128×128, 64×64, 32×32 y 16×16.

3.2.2. Red de proposición de regiones (RPN)

Como se mencionó en la sección 2.11.2, la red de proposición de regiones crea anclas que luego son clasificadas entre fondo y objeto de interés. En la figura 3.7 se pueden observar algunas de las propuestas de la RPN, es importante recordar que estas propuestas se realizan sobre el mapa de características, aquí solo se están proyectando sobre la imagen original con fines ilustrativos. Dependiendo de si se está realizando entrenamiento o predicción es el número de propuestas que se pasan a la siguiente etapa. Cuando es entrenamiento se pasan 2000, cuando es inferencia solo 1000.

Para proponer las anclas se definieron algunos hiperparámetros:

- Las escalas de las anclas en pixeles $(32, 64, 128, 256, 512)$. Esto es el ancho y alto de las anclas propuestas, estas dimensiones luego se proyectan al mapa de características. Cada una de las escalas esta asociada con un nivel de la FPN.

[1]Este término hace referencia a que es la red que obtiene el mapa de características. Por cuestiones de estilo y de significado se prefirió no traducirlo.

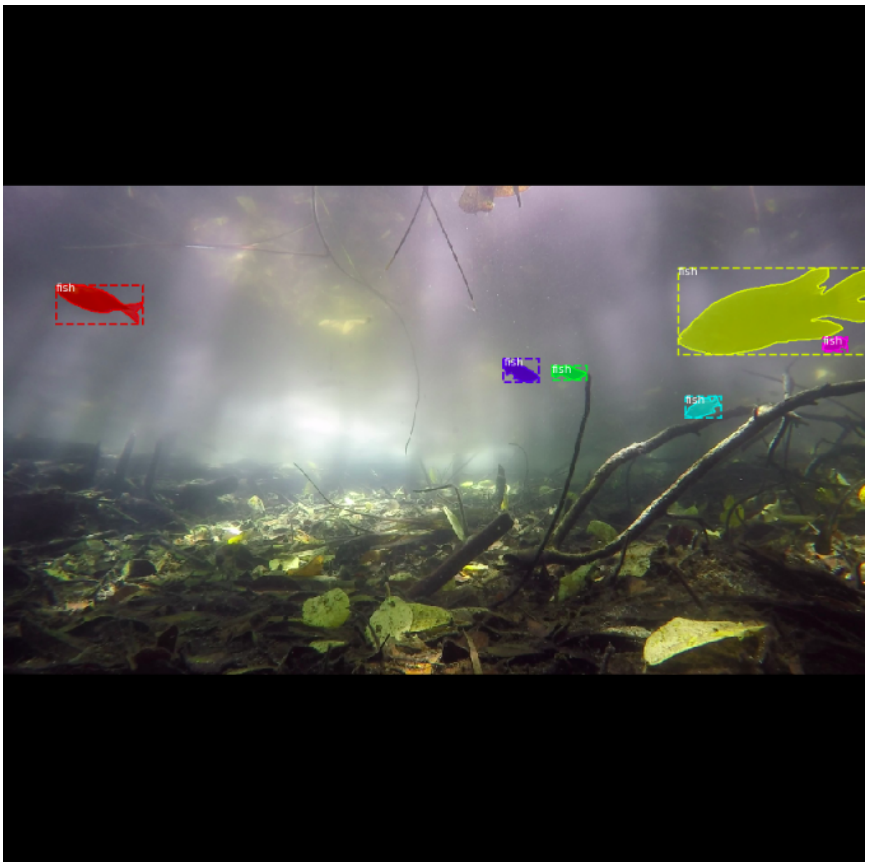


Figura 3.6: Ejemplo de la transformación de las imágenes de entrada a la red convolucional.

- Las razones de aspecto de las anclas $(0.5, 1, 2)$.
- El *stride* de la generación de anclas, es decir a que distancia estarán separados los centros de cada conjunto de anclas (1).

En la figura 3.8 se pueden observar las anclas en la imagen original.

3.2.3. Clasificador de las regiones de interés (ROI) y regresor de cuadros delimitadores

Una vez que se tienen las propuestas de la RPN, se deben de transformar a dimensiones fijas, que en esta implementación es de 7×7. Para

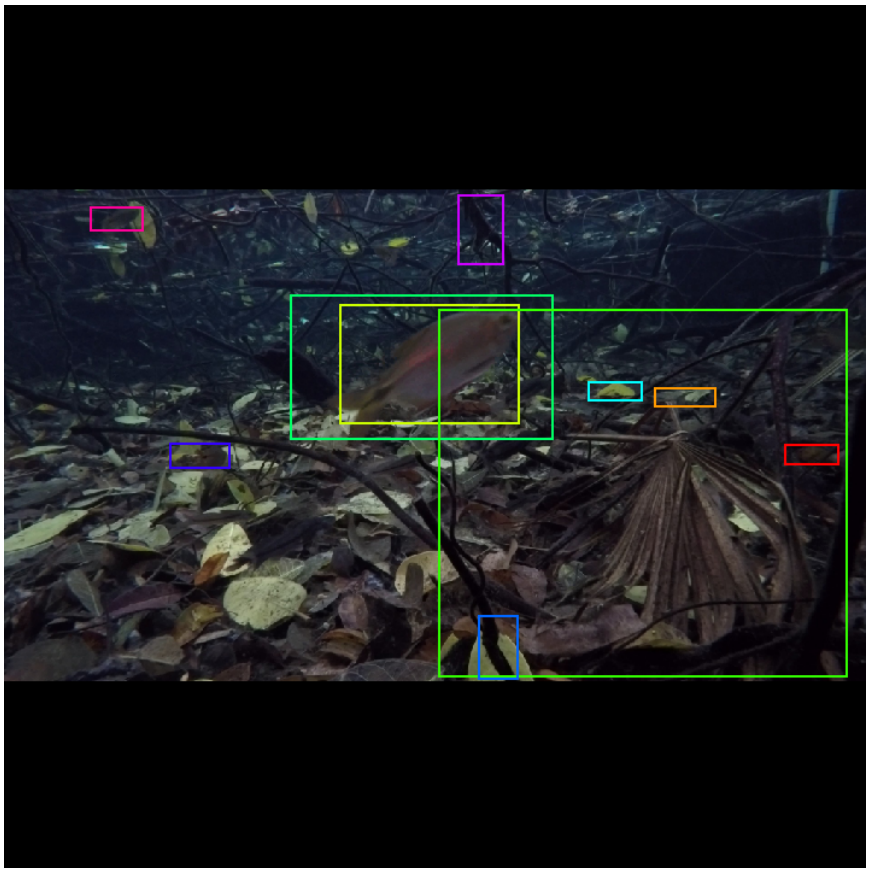


Figura 3.7: Ejemplo de las ROI propuestas por la RPN, en este caso el número de propuestas que se enseñan están limitadas a diez.

lograr esto en la implementación original los autores proponen un método llamado *ROIAlign* que usa la interpolación bilinear para hallar el valor de pixeles desconocidos, mientras que en la implementación de Abdulla (Abdulla, 2017) se usa el método *crop_and_resize* incluido en la biblioteca de aprendizaje profundo TensorFlow. Este método se encarga de hacer el escalamiento necesario, en la documentación se puede hallar que este método usa la interpolación bilinear, así que se puede considerar que los resultados son idénticos a los del articulo original.

Una vez que se han convertido las propuestas a un vector de dimensiones fijas, se puede alimentar al clasificador de ROI's y al regresor de cuadros delimitadores. En esta implementación el clasificador de ROI's solo clasifica

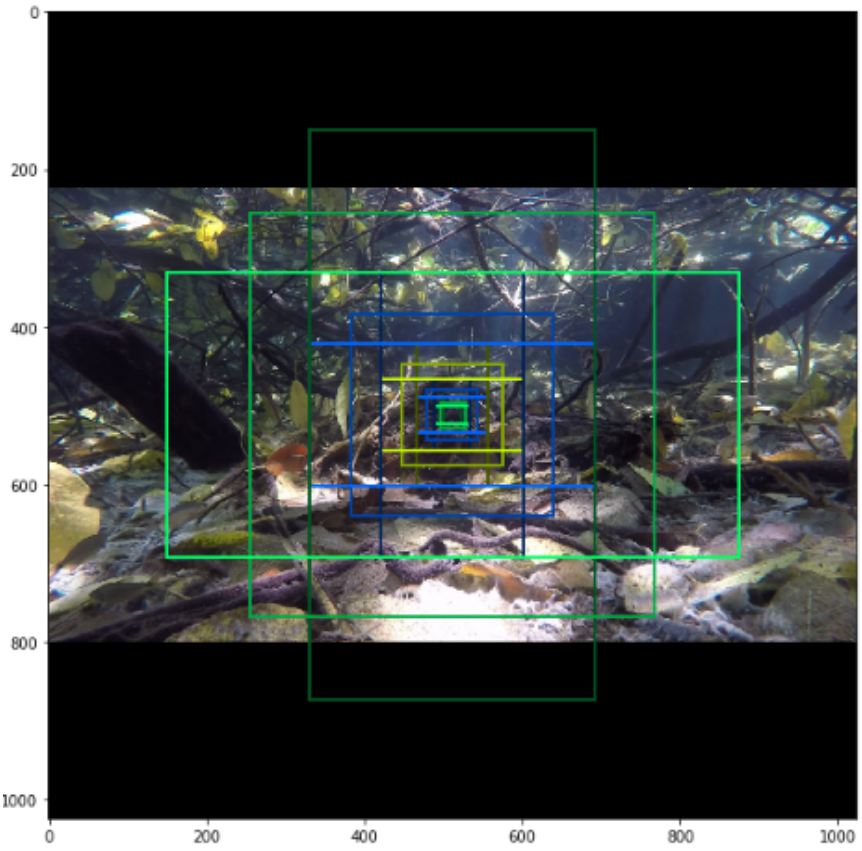


Figura 3.8: Ejemplo de las anclas propuestas en un solo punto de la imagen.

entre "pez" y "fondo". En la figura 3.9 se puede observar un ejemplo de la red clasificando las ROI.

En paralelo se alimenta la red de corrección de los cuadros delimitadores. Esto sirve para mejorar la calidad de los cuadros delimitadores y que contengan de mejor manera al objeto de interés. En la figura 3.10 se observa un ejemplo de la corrección de los cuadros delimitadores, en esta imagen el número de objetos está limitado a cinco. Para eliminar las ROI que no son relevantes, primero se descartan las que son clasificadas como fondo, posteriomente se descartan las que tengan puntajes menores al umbral de detección (en la sección 4.2 se entra en detalles del valor). Finalmente se usa el algoritmo de supresión no máxima[2] que consiste en tomar las propuestas

[2] *non-maximum supression*

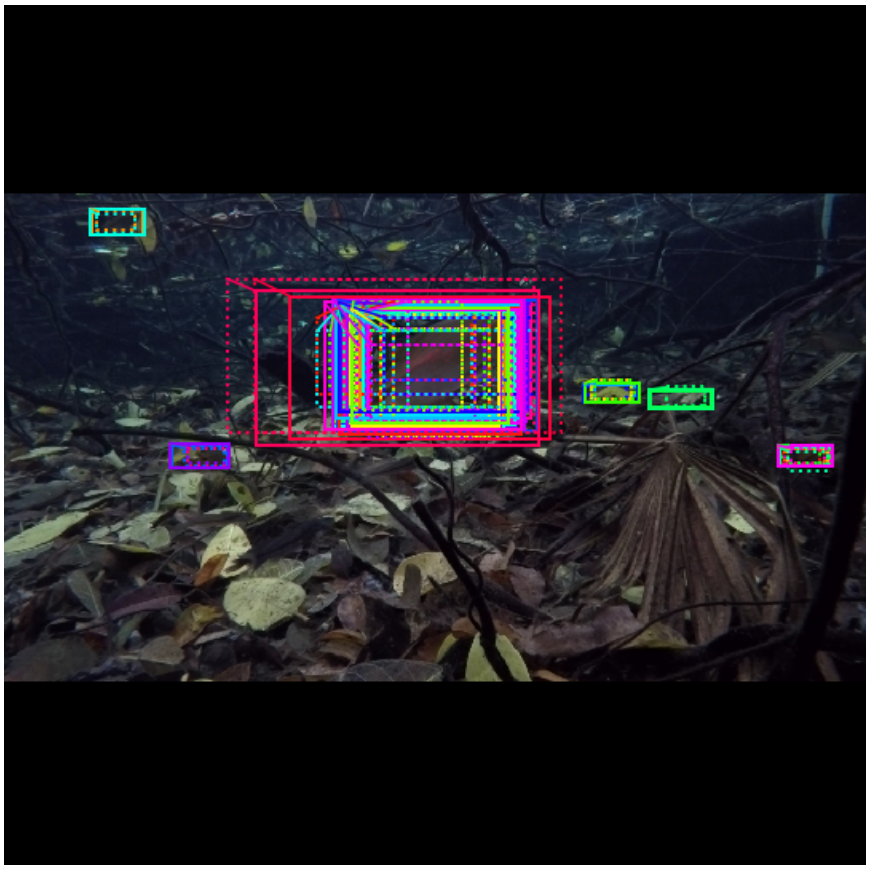


Figura 3.9: Ejemplo de la clasificación hecha por el clasificador de ROI's. Los cuadros delimitadores punteados son los que pertenecen a la clase fondo, los que están con linea continua son los que pertenecen a la clase pez.

con los valores de probabilidad mas altos y se descartan todas las propuestas que tengan un IoU menor a un umbral dado (únicamente durante el entrenamiento, pues es cuando se puede comparar con una etiqueta real), en esta implementación el umbral es de 0.3. En la figura **??** se puede observar la detección final, con los que solo se mantienen dos ROI's y se corrigen las coordenadas de sus cuadros delimitadores.

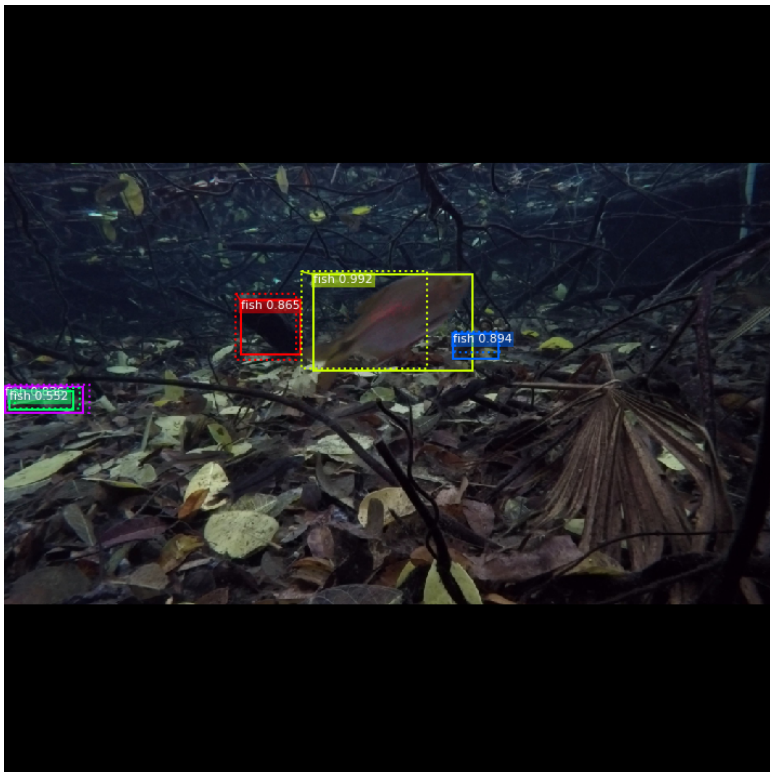


Figura 3.10: Ejemplo de la corrección hecha por el regresor de cuadros delimitadores. Los cuadros delimitadores punteados son los iniciales, los que están con linea continua son los corregidos.

3.2.4. Máscaras de segmentación

Esta parte de la arquitectura es la que se encarga de proponer las máscaras para las regiones de interés. En esta implementación las máscaras tienen dimensiones de 28×28 pixeles, son de baja resolución, pero están representadas con números flotantes, esto les permite guardar más información que las máscaras binarias. Durante el entrenamiento las máscaras reales son reescaladas a las dimensiones 28×28 para poder calcular el costo con las máscaras predichas. Cuando se está ejecutando el modelo en modo de inferencia las máscaras son reescaladas para coincidir con el tamaño de los cuadros delimitadores de las regiones de interés, esto proporciona las máscaras finales, una por objeto. En la figura 3.11 se pueden observar las máscaras propuestas por la red, y en la figura 3.12 se puede observar como se reescalan y se posicionan en el lugar adecuado al presentar la detección.

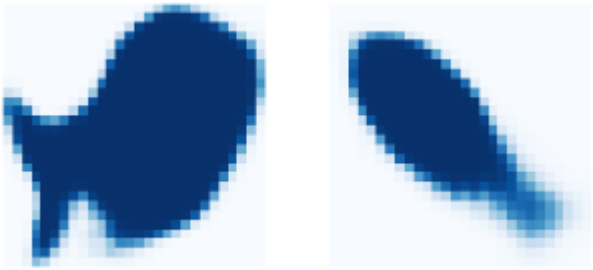


Figura 3.11: Ejemplo de las máscaras propuestas por la red. Estas son flotantes con dimensiones 28×28 pixeles.

Figura 3.12: Ejemplo de las máscaras en la detección. En esta imagen se observa como son reescaladas y posicionadas en la ubicación adecuada.

3.3. Modelos de predicción

Se corrió el entrenamiento por 90 épocas de 200 iteraciones cada uno y solo se pretende detectar peces, por lo que se declaró la clase de pez y la de *background* para el clasificador. De esta manera se observa si el sistema es capaz de al menos detectar peces. Se almacenan los pesos del entrenamiento para cada época, esto sirve para poder elegir el conjunto de pesos que se desempeñe de mejor manera, ya que este no siempre es el último. El optimizador del modelo durante el entrenamiento es el descenso por gradiente estocástico con momento.

Algunos hiperparámetros que vale la pena mencionar se encuentran en la tabla 3.1.

Hiperparámetro	Valor
Tamaño del *minibatch*	1
Tasa de aprendizaje	0.001
Tasa del momento de aprendizaje	0.9
Número máximo de instancias en detección	100
Épocas de entrenamiento	90
Pasos por Época	200
Pasos para la validación por época	50

Tabla 3.1: Tabla con los valores de algunos hiperparámetros utilizados en la creación de los modelos.

El tamaño del *minibatch* se tuvo que mantener en uno, ya que la capacidad de memoria de la tarjeta de vídeo no permitía mas imágenes a la vez. La tasa de aprendizaje, el momento de aprendizaje que son parámetros del optimizador, se mantuvieron en los valores predeterminados de TensorFlow. El numero máximo de instancias es el numero de objetos que se pueden detectar en una sola imagen, se mantuvo en el valor de cien instancias sugerido en el código original.

También se entrenaron algunos modelos que no se reportaron en el trabajo, estos sirvieron para determinar el numero de épocas de entrenamiento y los pasos por época. El numero de pasos para la validación por época se mantuvo en cincuenta, ya que con esto se logra una buena representación del desempeño en el conjunto de validación.

Los tamaños de los conjuntos de entrenamiento, validación y prueba para

cada modelo se reportan en la tabla 3.2.

Conjunto (No. Imágenes)	Modelo 1	Modelo 2	Modelo 3
Entrenamiento (Origs. + *D. Aug.*)	2128 (459 + 1669)	2395 (559 + 1836)	2906 (721+2185)
Validación	114	139	180
Prueba	30	36	47
Prueba "*easy test*"	50	50	50

Tabla 3.2: Tamaños de los conjuntos.

La construcción de cada modelo de predicción estará dada por el conjunto de pesos preentrenados en ImageNet o COCO, el número de imágenes, el número de épocas y pasos que se usaron para su entrenamiento. Los resultados se reportarán con la precisión, la sensibilidad y el valor-F sobre el conjunto de prueba *easy test* y el de prueba. En todos los modelos únicamente se entrenaron las redes FPN, RPN, el clasificador de ROI's, el regresor de cuadros delimitadores y la red de propuesta de máscaras, es decir, la ResNet101 no se modificó en absoluto. A las redes que se entrenan se les llama *heads*.

En la tabla 3.3 se observa la cantidad de parámetros de la arquitectura Mask R-CNN. Los de la red ResNet101 son preentrenados y no se modifican. Los de las redes *heads*, son los que se deben hallar durante el entrenamiento.

Red Neuronal	Cantidad de parámetros
ResNet101	44.5 M
Heads	19.1 M
Total	63.6 M

Tabla 3.3: Tabla con el número de parámetros entrenables de la arquitectura Mask R-CNN.

3.4. Métricas de comparación

Para poder comparar el desempeño de los modelos es necesario contar con parámetros de medición, que digan que tan bien, o que tan mal, se está desempeñando un modelo. En este trabajo se hace uso de dos principalmente, la precisión y la sensibilidad.

Antes de definir las métricas, es necesario hablar de los conceptos de verdadero positivo, falso positivo, verdadero negativo y falso negativo. Estos

conceptos son más fáciles de ilustrar con un gráfico. En la figura 3.13 se puede observar de manera gráfica como se realiza la clasificación de las predicciones, lo que se encuentran del lado izquierdo de los elementos relevantes,es decir, los que cumplen con la característica deseada. Los que se encuentran del lado derecho son los falsos, es decir, los que no cumplen con la característica deseada. Supóngase un conjunto de elementos en los que solo se tiene en cuenta una categoría o característica, se puede dividir en dos, es decir, los que cumplen o no con esta característica. De la misma manera supóngase que se somete el conjunto a la clasificación con un modelo entrenado para esta tarea. Los elementos que el sistema de clasificación etiquete correctamente son llamados verdaderos positivos, mientras que los que etiquete como correctos pero no lo sean, son llamados falsos positivos. Los elementos que no etiquetó pero debieron ser etiquetados son falsos negativos. Finalmente, los que no se etiquetaron y tampoco debían serlo son verdaderos negativos.

Una vez que se han introducido estos conceptos es más sencillo definir las métricas: precisión y sensibilidad.

3.4.1. Precisión y sensibilidad

La precisión se calcula como el cociente entre el número verdaderos positivos con el total de predicciones del clasificador. Se puede interpretar como el nivel de confianza en la predicción del modelo. La sensibilidad se calcula como el cociente entre el número de verdaderos positivos con el total de elementos que cumplen con la categoría de interés. Este se puede interpretar como la capacidad de identificar los elementos de interés en el conjunto.

En la figura 3.13 se puede observar de manera gráfica como se calculan la precisión y la sensibilidad.

Para aclarar todavía más el significado de la precisión y la sensibilidad, se puede observar la figura 3.14. En ella se encuentran seis peces, los cuales se etiquetaron manualmente, se observan con el contorno verde. El modelo predice cinco peces, se observan con el contorno en rojo. De los cinco peces predichos, los cinco son correctos, es decir son verdaderos positivos. Como no hubo ninguna predicción errónea el número de falsos positivos es cero. Al modelo le faltó clasificar uno de manera correcta por lo que existe un falso negativo. La precisión del modelo es el cociente entre el número de verdaderos positivos con el número de predicciones, es decir, la suma de verdaderos positivos y falsos positivos. Ya que todas las predicciones fueron correctas, la precisión es de uno o del 100 %. la sensibilidad se calcula como

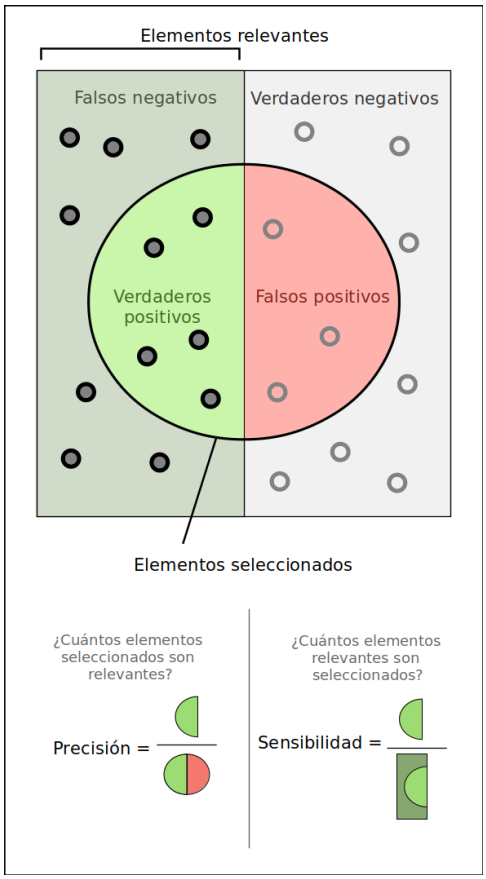


Figura 3.13: Representación gráfica de la precision y la sensibilidad.

el cociente del número de verdaderos positivos con el número de elementos que debieron ser predichos, es decir, la suma de los falsos negativos con los verdaderos positivos. Como solo se detectaron cinco de los seis peces en la imagen, la sensibilidad es de 5/6 o del 83.3 %. linewidth

3.4.2. Valor-F

Cuando se quieren comparar modelos, es difícil para las personas tomar en cuenta muchas variables a la vez, por lo que siempre se intenta reducirlas, pero mantener la mayor cantidad de información posible. El valor-F es una medida del desempeño de los modelos de clasificación que conjuga en un

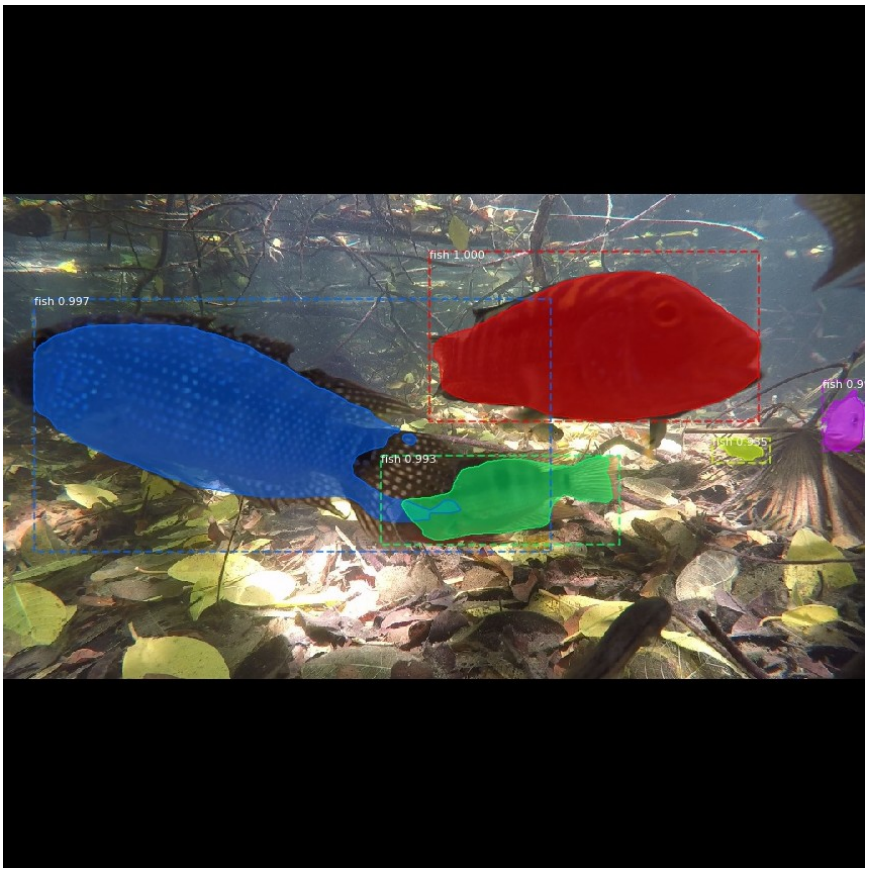


Figura 3.14: Imagen procesada por los modelos de predicción para ejemplificar el concepto de precisión y de sensibilidad.

solo valor la precisión y la sensibilidad. La fórmula general es la siguiente:

$$F_\beta = (1 + \beta^2)\frac{\text{Precisión} \cdot \text{Sensibilidad}}{(\beta^2 \cdot \text{Precisión}) + \text{Sensibilidad}} \tag{3.1}$$

Dónde β denota el peso que se le otorga a la sensibilidad sobre la precisión. Si el valor es igual a uno, ambos valores son igual de importantes. Si es mayor que uno, se le está otorgando mayor importancia a la sensibilidad, es decir, a capturar la mayor cantidad de peces en la imagen, menos falsos negativos. Si el valor es menor que uno, se le está otorgando mayor importancia a la precisión, es decir, que las predicciones sean correctas, menos falsos positivos. Por ello en la sección 4 se tomaron dos valores de β para el valor-F, $\beta = 1$ y

$\beta = 0.5$. El primero es para darle la misma importancia a ambas mediciones, y el segundo es para darle el doble de importancia a la precision que a la sensibilidad.

En el siguiente capitulo se presentaran los resultados obtenidos al evaluar los modelos de predicción en los diferentes conjuntos de prueba y modificando algunos de sus hiperparámetros.

Capítulo 4

Resultados

En este capitulo se presentan los resultados obtenidos para cada modelo de detección. Se entrenaron 3 modelos de predicción, cada uno con la distribución de la base de datos mencionados en la tabla 3.2. Cada modelo tiene dos versiones, una en la que la ResNet que obtiene el mapa de características utiliza pesos preentrenados en ImageNet y otra en COCO.

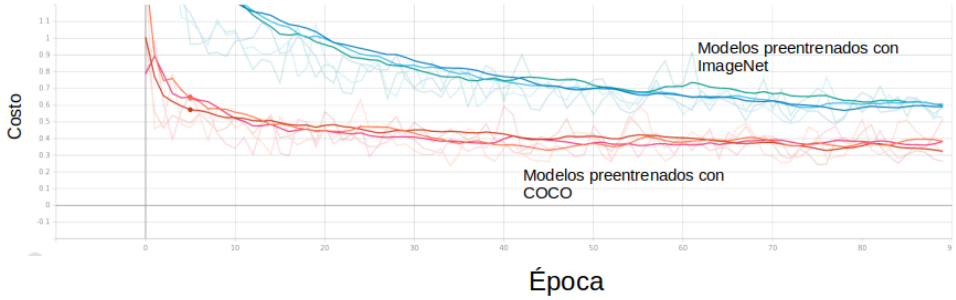
Durante el entrenamiento se mantuvo registro de cada una de las 90 épocas de entrenamiento, es decir, que cada 200 pasos se guardaron los pesos.

4.1. Función de costo durante el entrenamiento.

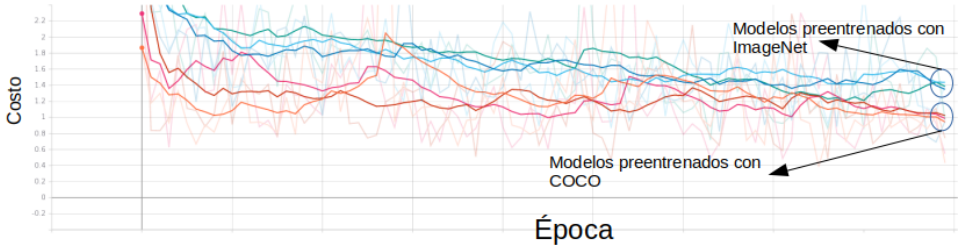
Como se menciona en la sección 2.5.4, la función de costo es el parámetro que cuantifica que tanto se esta acercando a realizar la tarea de manera óptima. Es por ello que es un buen parámetro para medir el desempeño del sistema y al mantener registro de su valor durante las épocas de entrenamiento se pueden elegir los pesos de las épocas con los menores valores en la función de costo.

Con ayuda del Software Tensorboard, en la figura 4.1a se puede observar el cambio en el valor de la función de costo conforme se entrenaba la red. En la figura 4.1b se puede observar lo mismo pero para el costo en el conjunto de validación. En estas imágenes las lineas en tonos de azul son los correspondientes a los tres modelos con pesos preentrenados en ImageNet, las lineas con tonos de rojo son los tres modelos con pesos preentrenados en

COCO.



(a) Valor de la función de costo durante el entrenamiento.



(b) Valor de la función de costo durante la validación.

Figura 4.1: Evolución del valor de la función de costo durante el entrenamiento y la validación. El eje de las abscisas corresponde a la época y el de las ordenadas al valor de la función de costo.

Existen dos parámetros que se pueden modificar una vez que los modelos de predicción se han entrenado y obtener diferentes desempeños en la detección. El primero es la probabilidad que asigna la red de que un cuadro delimitador pertenezca a la clase pez. Mientras mayor sea este, habrá menor cantidad de falsos positivos, este si repercute directamente en la inferencia, ya que puede causar que se propongan menos peces. El segundo parámetro es el IoU, que se definió en la sección 2.11.2, este no cambia el número de predicciones que la red hace, ya que toma como referencia una etiqueta real de contorno previa, lo que hace es cuantificar la calidad de las propuestas de región que contienen al pez, mientras mayor sea el límite inferior, mejores son las etiquetas. El IoU, se calcula como el cociente entre el área de intersección con la unión de las áreas.

4.2. Comparación de los modelos de predicción

Para comparar los modelos existían dos parámetros que se podían modificar independientemente del modelo y la época de entrenamiento, el umbral de probabilidad de que pertenezca a la clase pez y el IoU respecto a la etiqueta real. En las siguientes tablas y figuras de modifica el umbral de detección (probabilidad) pero se mantiene el umbral del IoU para todas. Es decir, si el valor del IoU es mayor o igual al 50 % se toma la detección como correcta. Este umbral podría hacerse más estricto, es decir que la máscara sea más precisa pero esto puede resultar en un decremento del desempeño de la red. En la mayoría de los artículos relacionados a la detección de objetos se toma el umbral mayor o igual al 50 %, es por lo anterior que en este trabajo se utiliza.

4.2.1. Evaluación en el conjunto *easy test*

Para las tablas 4.1 y 4.2, se toma como correcta la clasificación si la probabilidad de que el objeto sea un pez es mayor a 0.95 y el valor del IoU es mayor al 50 % respecto a la etiqueta real.

Modelo	Preentrenamiento	Precisión	Sensibilidad	Valor F_1	Valor $F_{0.5}$
1	COCO	0.8077	0.8923	0.8479	0.8233
	ImageNet	0.7700	0.6224	0.6884	0.7351
2	COCO	0.8243	0.8292	0.8267	0.8253
	ImageNet	0.8200	0.8472	0.8334	0.8253
3	COCO	0.7383	0.8766	0.8015	0.7624
	ImageNet	0.7800	0.6290	0.6964	0.7443

Tabla 4.1: Desempeño en easy test, calculado con IoU > 50 %, en la época 90.

Modelo	Preentrenamiento	Época	Precisión	Sensibilidad	Valor F_1	Valor $F_{0.5}$
1	COCO	75	0.8400	0.8280	0.8340	0.8376
	ImageNet	88	0.7200	0.6766	0.6976	0.7109
2	COCO	72	0.8413	0.8400	0.8406	0.8410
	ImageNet	52	0.8400	0.8357	0.8378	0.8391
3	COCO	72	0.7350	0.8970	0.8080	0.7625
	ImageNet	80	0.7753	0.8022	0.7885	0.7805

Tabla 4.2: Desempeño en *easy test*, calculado con IoU > 50 %, en la época con el menor costo.

Como se observa en las gráficas de la función de costo (veasen figuras 4.1a y 4.1b), el desempeño de los modelos entrenados con ImageNet es inferior a

los entrenados con COCO. Por lo anterior las tablas siguientes solo toman en cuenta los modelos entrenados con COCO.

Si se asume que el modelo 3 es el más robusto por haber sido entrenado con un conjunto de entrenamiento más grande, se puede analizar el comportamiento si se varia el umbral de probabilidad de predicción, haciéndolo cada vez más estricto. La gráfica 4.2 muestra el desempeño del modelo 3 en diferentes épocas de entrenamiento y con umbrales cada vez más estrictos en el conjunto *easy test*. En el eje de las abscisas se puede observar el umbral de predicción que varia desde 0.95 hasta .999, en el eje de las ordenadas se tiene el valor $F_{0.5}$.

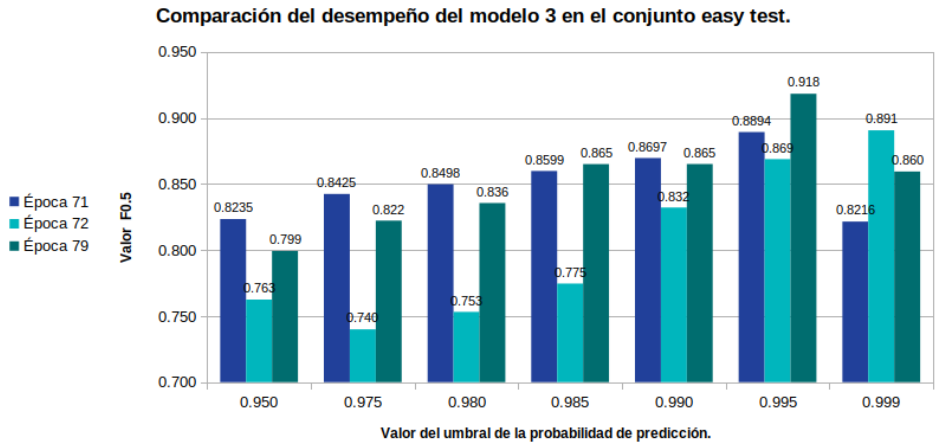


Figura 4.2: Comparativa del valor $F_{0.5}$ para las épocas 71, 72 y 79 del modelo 3 evaluadas en el conjunto *easy test*.

4.2.2. Evaluación en el conjunto de prueba

La tabla 4.3 muestra el desempeño de los modelos en las épocas con los menores costos.

Al igual que en la sección anterior los resultados se pueden visualizar en una gráfica de barras (véase figura 4.3). En ella se compara el desempeño del modelo tres, con base en el valor $F_{0.5}$, en tres épocas diferentes (71, 72 y 79) y con umbrales de predicción que van de 0.95 a 0.999.

Modelo	Época	Precisión	Sensibilidad	Valor F_1	Valor $F_{0.5}$
1	75	0.6983	0.6020	0.6466	0.6767
2	72	0.5414	0.6573	0.5937	0.5612
3	72	0.5398	0.7353	0.6226	0.5701

Tabla 4.3: Desempeño en el conjunto de prueba, calculado con IoU $> 50\,\%$, en la época con el menor costo.

4.2.3. Ejemplos de las predicciones de los modelos.

En las figuras 4.4, 4.5 y 4.6, se pueden observar ejemplos de las predicciones de los modelos en sus mejores épocas. El contorno verde es la etiqueta manual, el contorno rojo es el propuesto por el modelo. En los números que se encuentran en la parte superior izquierda de la región, separados por una diagonal (/), el primero indica la probabilidad de que pertenezca a la clase pez, el segundo es el valor de IoU de las regiones.

l

En el siguiente capitulo se analizan los resultados obtenidos y se explica por que se dieron de esta manera. También se habla de las ventajas y las desventajas de la metodología propuesta.

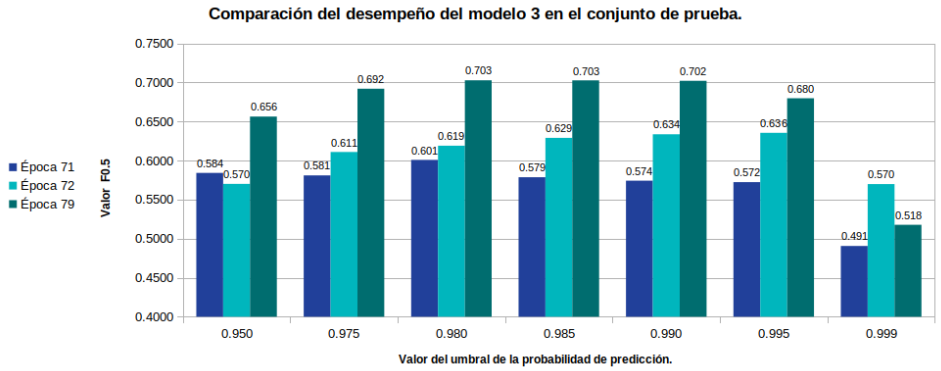


Figura 4.3: Comparativa del valor $F_{0.5}$ para las épocas 71, 72 y 79 del modelo 3 evaluadas en el conjunto de prueba.

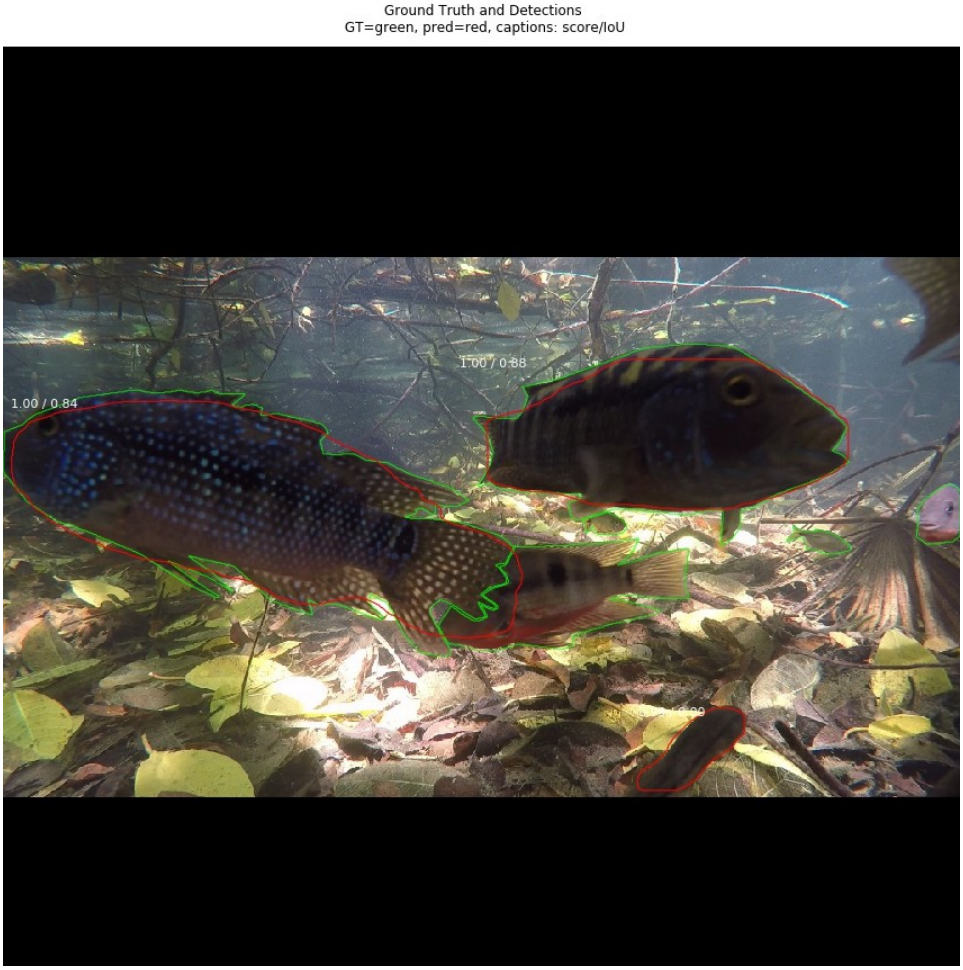


Figura 4.4: Ejemplo de predicción del modelo 1.

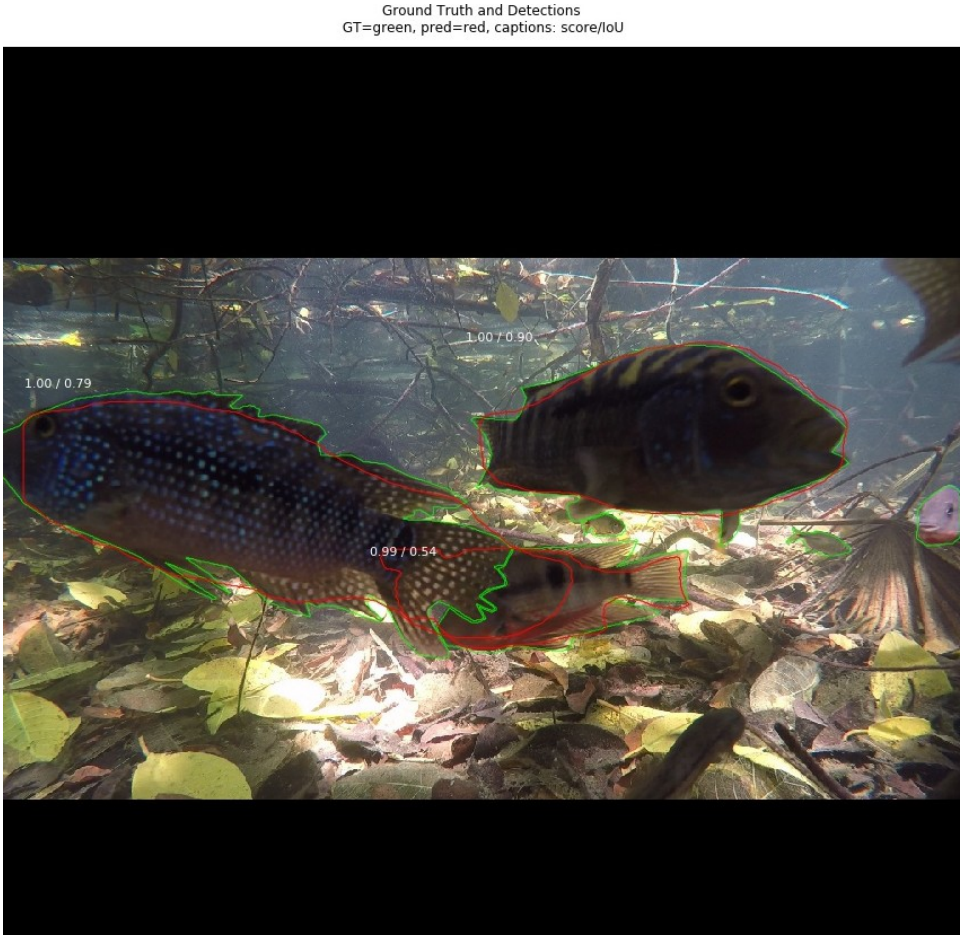


Figura 4.5: Ejemplo de predicción del modelo 2.

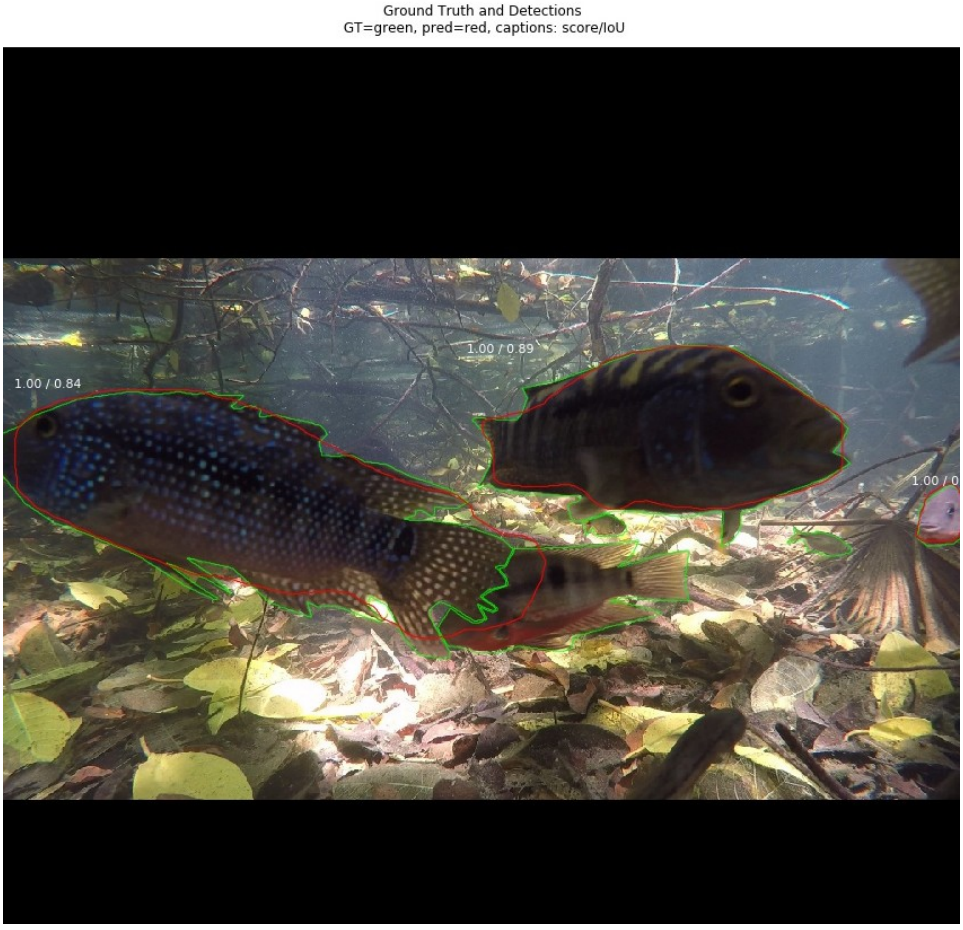


Figura 4.6: Ejemplo de predicción del modelo 3.

Capítulo 5

Discusión de resultados

En el capítulo anterior se presentaron los resultados del desempeño de los modelos usando diferentes umbrales de detección y evaluados en los diferentes conjuntos de prueba. A continuación se dan argumentos que explican el por que se obtuvieron. También se comparan con los métodos tradicionales mencionados en la introducción y se mencionan algunas ventajas y desventajas del uso del aprendizaje profundo para el monitoreo de peces.

5.1. Comportamiento de la función de costo

En las figuras 4.1a y 4.1b se observa que la tendencia de los modelos de predicción tiene pendiente negativa. Es normal que el valor de la función de costo sea mayor en el conjunto de validación, pero es importante que la pendiente sea negativa ya que si fuese positiva es altamente probable que se esté sobre ajustando el modelo al conjunto de entrenamiento. En ambas gráficas se puede ver que los modelos que usaron pesos preentrenados en COCO tuvieron mejores desempeños que los de ImageNet, esto también se observa en las tablas 4.1 y 4.2, es por ello que no se reportan los resultados de los modelos preentrenados con ImageNet en las tablas subsecuentes.

Consultando la literatura disponible con respecto a la transferencia de aprendizaje no se encontró una explicación determinante del mejor desempeño de COCO sobre ImageNet, dependiendo de la aplicación esto puede ocurrir de manera inversa. Una de las hipótesis del autor es la cantidad de clases declaradas para cada base de datos. Los pesos de ImageNet fueron entrenados para clasificar mil categorías, mientras que los de COCO sola-

mente ochenta categorías, esto puede causar que los filtros en la última capa sean más específicos con COCO que con ImageNet.

Algo que vale la pena mencionar es que ambas gráficas están suavizadas para poder observar la tendencia general, en algunos casos durante el entrenamiento el costo aumentaba en lugar de disminuir, esto es normal debido a que se esta utilizando un optimizador estocástico, es decir, toma un subconjunto del conjunto de datos de entrenamiento y actualiza los pesos con base en ellos en lugar de tomar la totalidad de los elementos del conjunto de entrenamiento. Debido a las limitaciones de recursos de cómputo, el tamaño del subconjunto para realizar el entrenamiento era de una imagen a la vez. Esto hace que el entrenamiento sea muy ruidoso y es el causante de los saltos durante cada época. A pesar de ellos se logra mantener una tendencia con pendiente negativa que se puede interpretar como que el sistema está aprendiendo.

5.2. Detención temprana

Una práctica común en el aprendizaje automático es la detención temprana, esta es un tipo de regularización. La idea principal es que al usar un método iterativo estocástico como el SGD, en algunas ocasiones se puede sobreajustar si se entrena demasiado, por lo que no siempre la última iteración es la mejor. Este concepto se observa al comparar los resultados de la tabla 4.1 con la tabla 4.2. El mejor desempeño se observa en la tabla 4.2 en lugar de la tabla 4.1 que es la última iteración. Es por ello que para comparar el desempeño de los modelos de predicción se toman las iteraciones con los valores más pequeños de la función de costo.

5.3. Desempeño en el conjunto *easy test*

Para poder comparar el desempeño de los modelos se utilizaron dos parámetros sencillos, el de precisión y el de *recall*, y uno que conjuga a ambos, el valor-F con un valor de $\beta = 0.5$ para darle el doble de importancia a la precisión que a la sensibilidad. Como se realizó una clasificación binaria, "pez" o "no pez", estas métricas logran representar el comportamiento de los modelos de predicción de manera correcta ya que solo se tiene una clase de objetos de interés. Cuando se tiene más de una categoría de clasificación generalmente se utilizan otros parámetros.

Como era de esperarse, el desempeño en el conjunto *easy test* es mejor que en el de prueba, esto es debido a que las imágenes contenidas en el *easy test* son peces grandes, que en su mayoría se pueden observar completos y nítidos. Los resultados obtenidos fueron de valor-$F_{0.5}$ igual a 84.10 % para el modelo 2 con un umbral de predicción del 95 %. Para el modelo 3 después de la modificación del umbral fue de valor-$F_{0.5}$ igual a 91.8 % con un umbral del 99.5 %.

5.4. Desempeño en el conjunto de prueba

El conjunto de prueba cuenta con imágenes que no fueron elegidas cuidadosamente, por lo que es él que refleja de mejor manera el desempeño en el mundo real de los modelos de predicción. Es normal que el desempeño en este conjunto sea menor dadas las características de las imágenes. Los resultados obtenidos fueron de valor-$F_{0.5}$ igual a 67.67 % para el modelo 1 con un umbral de predicción del 95 %. Para el modelo 3 después de la modificación del umbral fue de valor-$F_{0.5}$ igual a 70.3 % con un umbral del 98.5 %.

Estos valores pueden mejorarse si se continúa alimentando a la red con más información para su entrenamiento. Si se lograra mover la distribución de las imágenes de prueba, es decir, que en las pruebas reales se usará una cámara que tomase fotos similares a la calidad de *easy test* fácilmente se mejorarían los resultados.

5.5. Intercambio entre sensibilidad y precisión

Como se menciona en el capítulo anterior, se asume que el modelo de predicción 3 es el más robusto dado que el entrenamiento se realizó con una mayor cantidad de imágenes. En los resultados que se observan en las tablas 4.2 y 4.3 parece ser que el modelo 3 no es el mejor, de hecho el modelo 2 y el modelo 1 tienen el mayor valor $F_{0.5}$ respectivamente. Sin embargo, si se observa el valor de la sensibilidad, este es significativamente mayor en el modelo 3 en ambas gráficas. Existe un intercambio entre el valor de precisión y sensibilidad al variar el umbral de predicción. Es decir, dado un modelo de predicción si se aumenta la probabilidad mínima para que un elemento pertenezca a la clase de interés,la precisión incrementa y la sensibilidad disminuye, esto se debe a que el número de propuestas disminuye, disminuyendo los falsos positivos y los verdaderos positivos. Con

base en este intercambio entre sensibilidad y precisión, se puede decir que el modelo 3 otorga una mayor libertad para modificar el umbral de predicción y mejorar el desempeño en general.

En las gráficas 4.2 y 4.3 se comparan tres versiones del modelo 3 y se varía el umbral de predicción. En ellas se observa el intercambio entre sensibilidad y predicción conjugados en el valor $F_{0.5}$, conforme se aumentaba el valor del umbral de predicción en general los modelos aumentaban su valor $F_{0.5}$, hasta que en algún punto alcanzaban su valor máximo y empezaban a disminuir el valor $F_{0.5}$. Con estas modificaciones del umbral de predicción se logra que el desempeño del modelo 3 supere por 7.7 puntos porcentuales el del modelo 2 en el conjunto *easy test* y por 2.63 puntos porcentuales al del modelo 2 en el conjunto de prueba.

5.6. Ventajas y desventajas del aprendizaje profundo contra los métodos tradicionales.

Las ventajas del uso del aprendizaje profundo son claras sobre los métodos tradicionales, en general los sistemas son robustos y una vez que se cuenta con un modelo entrenado es posible modificar las condiciones del ambiente de trabajo (iluminación, orientación, fondo, etc.) sin que el desempeño se vea afectado de maneras importantes. También cuenta con la ventaja de que gran parte del trabajo pesado es realizado por la arquitectura de red neuronal en lugar del ingeniero o científico, por ejemplo, no se tiene que decir en qué secciones de la imagen aparecen los peces, el tamaño, ni los filtros o características que se deben usar. Otra ventaja es que una vez que se ha planteado el protocolo de trabajo, la creación de modelos de predicción para nuevas tareas también se vuelve metódico.

Las desventajas también son claras, la cantidad de imágenes necesarias para obtener desempeños adecuados suele ser muy alta. La preexistencia de una base de datos correctamente segmentada y clasificada es una desventaja, ya que el ahorro del esfuerzo en la creación del modelo pasa a la creación de la base de datos (generalmente para aprendizaje supervisado). Los recursos de cómputo también suelen ser un problema, no todas las instituciones cuentan con los recursos necesarios, afortunadamente esto va cambiando con la cada vez mayor oferta de computo en la nube a precios cada vez más baratos.

5.7. Comparación de los resultados con trabajos similares.

Si se compara este sistema de detección de peces con otros trabajos similares se puede decir que su desempeño es igual o mejor. En los métodos de visión por computadora tradicionales se tiene el trabajo de Spampinato (Spampinato et al., 2010) y de Shevchenko (Shevchenko et al., 2018), en los cuales utilizan descriptores de forma y algoritmos de sustracción de fondo para detectar peces respectivamente. Spampinato obtiene una sensibilidad del 92 %, no reporta la precisión ya que su base de datos solo cuenta con imágenes que contienen peces. Schevchenko obtiene un valor-$F_{0.5}$ de 58.8 %, este aparenta ser muy bajo, pero si se considera que las imágenes que se tomaron en cuenta son extremadamente ruidosas se entiende la dificultad a la que se enfrentó.

Si ahora se toman en cuenta los trabajos que hacen uso de redes neuronales convolucionales las cosas no son muy diferentes, Konovalov (Konovalov et al., 2019) obtiene un valor-$F_{0.5}$ de 96 % realizando clasificación binaria. Abdelouahid reporta resultados muy contrastantes,una precisión media [1] (AP) del 99.32 % en la base de datos Fish4Knowledge, pero del 58.81 % en la base de datos LifeClef2015. Si se analizan las razones por las cuales existen estos resultados tan contrastantes, se nota que la calidad de la base de datos es fundamental sin importar el método que se esté utilizando. Algo en lo que coinciden la mayoría de los trabajos consultados es en que al contar con imágenes de mejor calidad, la clasificación puede mejorar significativamente. Este trabajo presenta desempeños competitivos, valor-$F_{0.5}$ del 91.8 % en el conjunto *easy test* y del 70.3 % en el conjunto de prueba.

[1] *average precision*

Capítulo 6

Conclusiones

En este trabajo se logra plantear el camino necesario para la automatización de la tarea de detección y clasificación de peces. La arquitectura que se utilizó tiene la posibilidad de fácilmente escalarse a la clasificación por especies, que en este trabajo se optó por no realizar, debido al tiempo que conlleva crear la base de datos y tampoco era el objetivo principal. El protocolo que se realizó permitirá a los expertos en estudios biológicoambientales, que impliquen monitoreo de peces, incluir en sus tareas el etiquetado de peces para facilitar el entrenamiento de este y nuevos modelos. La base de datos que se creó y se utilizó en este trabajo queda como testimonio y recurso para nuevos proyectos, ya que generalmente en todo trabajo de aprendizaje automático la obtención de una base de datos confiable y de calidad es la parte más complicada y demandante de tiempo. Al final de este trabajo se lograron etiquetar 998 imágenes con 2187 peces en ellas. En el aprendizaje automático y específicamente en el aprendizaje profundo la materia prima son los datos y mientras mayor sea la cantidad, mejor será el desempeño del modelo de predicción.

Al evaluar los 3 modelos en los diferentes conjuntos de prueba se logró obtener valores de $F_{0.5}$ entre el 85 % y el 90 % en el *easy test* y alrededor del 70 % en los mejores casos en el conjunto de prueba. Esto es prometedor ya que las pruebas se realizaron con imágenes reales, que representan la variabilidad del entorno, ya sea por materia orgánica suspendida, iluminación, falta de nitidez en las imágenes y diferentes tamaños de peces. Como se mencionó anteriormente, se le otorga una mayor importancia a la precisión, es por ello que se usa la métrica $F_{0.5}$ para comparar los modelos. Esto se debe a que al comparar con la detección visual por expertos, es poco probable que se

incurra en falsos positivos, la probabilidad es prácticamente nula, pero es altamente probable que se omita algún pez, es decir falsos negativos.

Si se compara contra la detección manual se tienen ventajas abismales en el tiempo de detección, tardando aproximadamente 3 minutos para procesar 50 imágenes, lo que equivale a 3.6 segundos por imagen esto en una computadora de escritorio sin GPU, podría aumentarse hasta en un factor de 10 si se usara una computadora con mayores capacidades. En el mejor de los casos la detección manual es de 1 minuto por imagen, implica un aumento de al menos 17 veces la velocidad de detección.

Capítulo 7

Trabajo futuro

Se observa claramente que el desempeño mejora conforme se aumenta la cantidad de imágenes para el entrenamiento. El desempeño podría mejorar haciendo uso del *Fine Tuning*. Esta técnica consiste en que cuando se tiene un modelo con un buen desempeño en la tarea de interés que fue creado con ayuda de la transferencia de aprendizaje, se "descongelan" las primeras capas de la red neuronal, cuyos pesos fueron preentrenados, y se entrenan con la base de datos propia. Esto mejora el desempeño de la red en la tarea. Es importante tomar algunas consideraciones, ya que si la tasa de aprendizaje o la base de datos propia es muy pequeña, el desempeño puede empeorar o se puede sobreajustar al conjunto de entrenamiento.

Uno de los mayores problemas fue la calidad de la base de datos y el hecho de que los peces se difuminan y se observan borrosos al extraer los cuadros de los videos. Es por lo anterior que se propone la adquisición de imágenes de mayor calidad con cámaras especializadas, que en lugar de video tomen fotos cada determinado tiempo, pero que cuenten con sensores capaces de tomar fotos nítidas a objetos en movimiento, esto no solo mejorará el entrenamiento, sino que también ayudará a alcanzar el objetivo de etiquetar las imágenes que se procesan para usarlas en nuevos entrenamientos del sistema. Lo anterior se debe a que las máscaras de segmentación seguirán el contorno de una forma más exacta. Otra ventaja de cambiar el almacenamiento de videos a imágenes es el ahorro en espacio de almacenamiento, ya que gran parte de la información no se utilizó.

Uno de los objetivos fácilmente alcanzable a corto plazo es la clasificación por especies. La arquitectura Mask R-CNN cuenta con la capacidad

de realizar la clasificación de objetos, solo falta agregar la información de especies a las etiquetas de la base de datos. Para la creación de una base de datos con esta información es necesaria la ayuda de expertos en el área. Por ello se propone que los biólogos incorporen a su trabajo el mantenimiento y aumento de la base de datos.

Un sistema mucho más complejo pero que mejorará de manera muy significativa la detección y clasificación de los peces es uno híbrido, en donde la información de entrada incluya datos de ultrasonido o de visión estereoscópica para agregar la dimensión de profundidad a la imagen 2D y mejorar el desempeño.

Eventualmente se encapsulará todo el sistema en un programa que sea fácil de instalar en cualquier computadora y que cuente con una interfaz de usuario amigable que muestre la distribución de las especies observadas en las imágenes.

Referencias

A. Tidd, R. & Wilder, J. (2001). Fish detection and classification system. *J. Electronic Imaging*, *10*, 283-288. doi:10.1117/1.1329338

Abdulla, W. (2017). Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. https://github.com/matterport/Mask_RCNN. Github.

Arthur C. Guyton, J. E. H. (2005). *Textbook of Medical Physiology* (11.ª ed.). Guyton Physiology. Saunders.

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, *25*(1847), 536-538.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. En *CVPR09*.

Duchi, J., Hazan, E. & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(Jul), 2121-2159.

Dutta, A., Gupta, A. & Zissermann, A. (2016). VGG Image Annotator (VIA). Version: X.Y.Z, Accessed: 15 June 2019. Recuperado desde http://www.robots.ox.ac.uk/~vgg/software/via/

Espinosa, D. A. (2019). *Variaciones temporales de la comunidad de peces en un humedal costero de Yucatán, mediante imágenes subacuáticas y técnicas tradicionales* (Universidad Nacional Autónoma de México).

Fukushima, K. (1980). Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, *36*, 193-202.

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. En *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Vol. 00, pp. 580-587). doi:10.1109/CVPR.2014.81

Girshick, R. B. (2015). Fast R-CNN. *CoRR*, *abs/1504.08083*. arXiv: 1504.08083. Recuperado desde http://arxiv.org/abs/1504.08083

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. En Y. W. Teh & M. Titterington (Eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Vol. 9, pp. 249-256). Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR. Recuperado desde http://proceedings.mlr.press/v9/glorot10a.html

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Hahnloser, R., Sarpeshkar, R., Mahowald, M., Douglas, R. & Seung, H. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, *405*, 947-51. doi:10.1038/35016072

He, K., Gkioxari, G., Dollár, P. & Girshick, R. B. (2017a). Mask R-CNN. *CoRR*, *abs/1703.06870*. arXiv: 1703.06870. Recuperado desde http://arxiv.org/abs/1703.06870

He, K., Gkioxari, G., Dollár, P. & Girshick, R. B. (2017b). Mask R-CNN. *CoRR*, *abs/1703.06870*. arXiv: 1703.06870. Recuperado desde http://arxiv.org/abs/1703.06870

He, K., Zhang, X., Ren, S. & Sun, J. (2015a). Deep Residual Learning for Image Recognition. *CoRR*, *abs/1512.03385*. arXiv: 1512.03385. Recuperado desde http://arxiv.org/abs/1512.03385

He, K., Zhang, X., Ren, S. & Sun, J. (2015b). Deep Residual Learning for Image Recognition. *CoRR*, *abs/1512.03385*. arXiv: 1512.03385. Recuperado desde http://arxiv.org/abs/1512.03385

He, K., Zhang, X., Ren, S. & Sun, J. (2015c). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, *abs/1502.01852*. arXiv: 1502.01852. Recuperado desde https://dblp.org/rec/bib/journals/corr/HeZR015

Hubel, D. H. & Wiesel, T. N. (1959). Receptive Fields of Single Neurons in the Cat's Striate Cortex. *Journal of Physiology*, *148*, 574-591.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Konovalov, D. A., Saleh, A., Bradley, M., Sankupellay, M., Marini, S. & Sheaves, M. (2019). Underwater Fish Detection with Weak Multi-Domain Supervision. *CoRR*, *abs/1905.10708*. arXiv: 1905.10708. Recuperado desde http://arxiv.org/abs/1905.10708

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. En D. Touretzky (Ed.), *Advances in Neural Information Processing Systems (NIPS 1989)* (Vol. 2), Denver, CO: Morgan Kaufman.

Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B. & Belongie, S. J. (2016). Feature Pyramid Networks for Object Detection. *CoRR*, *abs/1612.03144*. arXiv: 1612.03144. Recuperado desde http://arxiv.org/abs/1612.03144

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., ... Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. *CoRR*, *abs/1405.0312*. arXiv: 1405.0312. Recuperado desde http://arxiv.org/abs/1405.0312

Lowe, D. G. (1999). Object Recognition from Local Scale-Invariant Features. En *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2* (pp. 1150-). ICCV '99. Washington, DC, USA: IEEE Computer Society. Recuperado desde http://dl.acm.org/citation.cfm?id=850924.851523

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. United States of America: The Maple-Vail Book Manufacturing Group.

McCulloch, W. S. & Pitts, W. (1988). Neurocomputing: Foundations of Research. En J. A. Anderson & E. Rosenfeld (Eds.), (Cap. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15-27). Cambridge, MA, USA: MIT Press. Recuperado desde http://dl.acm.org/citation.cfm?id=65669.104377

Minsky, M. & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press.

Mitchell, T. (1997). *Machine learning* (1st). Mcgraw-Hill International Edit. McGraw Hill Higher EducatFishDetectionLowVisibilityion.

Papert, S. A. (1966). The Summer Vision Project. MIT. Recuperado desde http://hdl.handle.net/1721.1/6125

Pareto, V. (1896). *Cours d'Economie Politique*. Genève: Droz.

Prince, S. J. D. (2012). *Computer Vision: Models, Learning, and Inference* (1st). New York, NY, USA: Cambridge University Press.

Ren, S., He, K., Girshick, R. B. & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, *abs/1506.01497*. arXiv: 1506.01497. Recuperado desde http://arxiv.org/abs/1506.01497

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

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. En D. E. Rumelhart, J. L. McClelland & C. PDP Research

Group (Eds.), (Cap. Learning Internal Representations by Error Propagation, pp. 318-362). Cambridge, MA, USA: MIT Press. Recuperado desde http://dl.acm.org/citation.cfm?id=104279.104293

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211-252. doi:10.1007/s11263-015-0816-y

Shevchenko, V., Eerola, T. & Kaarna, A. (2018). Fish Detection from Low Visibility Underwater Videos. En *2018 24th International Conference on Pattern Recognition (ICPR)* (pp. 1971-1976). doi:10.1109/ICPR.2018.8546183

Shorten, C. & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, *6*(1), 60. doi:10.1186/s40537-019-0197-0

Solem, J. E. (2012). *Programming Computer Vision with Python - Tools and algorithms for analyzing images*. O'Reilly. Recuperado desde http://www.oreilly.de/catalog/9781449316549/index.html

Spampinato, C., Giordano, D., Di Salvo, R., Chen-Burger, Y.-H. J., Fisher, R. B. & Nadarajan, G. (2010). Automatic Fish Classification for Underwater Species Behavior Understanding. En *Proceedings of the First ACM International Workshop on Analysis and Retrieval of Tracked Events and Motion in Imagery Streams* (pp. 45-50). ARTEMIS '10. doi:10.1145/1877868.1877881

Tamou, A. B., Benzinou, A., Nasreddine, K. & Ballihi, L. (2018). Transfer Learning with deep Convolutional Neural Network for Underwater Live Fish Recognition. *2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS)*, 204-209.

Tieleman, T. & Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Tinku Acharya, A. K. R. (2005). *Image Processing: Principles and Applications*. Wiley-Interscience.

Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T. & Smeulders, A. W. M. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision*, *104*(2), 154-171. doi:10.1007/s11263-013-0620-5

Viola, P. & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. En *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* (Vol. 1, I-511-I-518 vol.1). doi:10.1109/CVPR.2001.990517

Wolpert, D. H. & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67-82. doi:10.1109/4235.585893

Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. (2014). How transferable are features in deep neural networks? *CoRR*, *abs/1411.1792*. arXiv: 1411.1792. Recuperado desde http://arxiv.org/abs/1411.1792