

**UNIVERSIDAD DE INGENIERÍA Y
TECNOLOGÍA**

CARRERA DE CIENCIA DE LA COMPUTACIÓN



**Detección y clasificación de especies de
peces: Un *pipeline* eficiente en un
dataset no etiquetado**

AUTOR

Cesar Antonio Madera Garcés
cesar.madera@utec.edu.pe

ASESOR

Cristian López
clopezd@utec.edu.pe

Lima - Perú
2022

Resumen

A lo largo de los tiempos, el error humano y la falta de la automatización ha sido uno de los problemas más grandes de las instituciones y usuarios inexpertos en general. A consecuencia de esto, la creación y uso de aplicaciones con inteligencia artificial para la detección y clasificación de imágenes ha ido aumentando conforme al tiempo para disminuir el posible error humano. En el contexto peruano, esta tarea nunca ha sido automatizada, generando también una falta de *datasets* con imágenes etiquetadas. Actualmente, los modelos de *Deep Learning*(DL) son los más usados generando una necesidad de este tipo de *datasets*. Es por ello que en el siguiente trabajo se propuso, implementó y experimentó con un *pipeline* de DL para la detección y clasificación de peces dentro de la fauna marina peruana, finalmente escogiendo una configuración que redujo la pérdida en el *accuracy* y el tiempo de entrenamiento. Para ello, se evaluaron diferentes modelos del estado del arte en base al tamaño en memoria, el numero de parámetros y la precisión obtenida con el *dataset* de ImageNet como métricas teóricas para su comparación, para luego aplicar *transfer learning* sobre ellas para adaptarlas a nuestra aplicación y obtener un balance entre precisión-costo computacional para finalmente obtener la mas eficiente.

Abstract

Throughout history, human error and the lack of automation have been significant challenges for institutions and inexperienced users in general. Consequently, the development and utilization of artificial intelligence applications for image detection and classification have steadily increased over time, aiming to mitigate potential human errors. However, in the Peruvian context, this task has yet to be automated, leading to a scarcity of labeled image datasets as well. Presently, deep learning models are widely employed, creating a demand for such datasets. Therefore, this study proposes, implements, and conducts experiments with a deep learning pipeline for detecting and classifying fish within the Peruvian marine fauna. The goal is to identify a configuration that reduces accuracy loss and training time. To achieve this, various state-of-the-art models were evaluated based on factors such as memory size, number of parameters, and accuracy attained using the ImageNet dataset as theoretical benchmarks for comparison. Subsequently, transfer learning was applied to adapt these models to our specific application, striking a balance between precision and computational cost, ultimately determining the most efficient approach.

Índice general

1. Contexto y Motivación	5
1.1. Introducción	5
1.2. Descripción del problema	6
1.3. Justificación	6
1.4. Objetivos	6
1.5. Aportes	7
2. Marco Teórico	8
2.1. Modelos de <i>Machine Learning</i> para clasificación	8
2.1.1. <i>Multi Layer Perceptron</i> (MLP)	8
2.1.2. <i>Convolutional Neural Networks</i> (CNN)	9
2.2. Arquitecturas del estado del arte	11
2.2.1. VGG-19	11
2.2.2. InceptionV3	11
2.2.3. ResNet50	12
2.2.4. EfficientNet	13
2.2.5. YOLO	15
2.2.6. UniDet	16
2.3. <i>Transfer Learning</i>	16
2.4. Pre-procesamiento de entradas	18
2.4.1. Tamaño de entrada	18
2.4.2. <i>One Hot Encoder</i>	18
2.5. <i>Overfitting</i>	19
3. Revisión de la Literatura	20
4. Metodología	24
4.1. <i>Dataset</i>	25
4.2. Detector	26
4.3. Clasificador	26
4.4. <i>Labeled Boxes</i>	27
4.5. Evaluación de modelos	27

5. Experimentaciones y Resultados	30
5.1. Aspectos generales	30
5.1.1. <i>Datasets</i>	30
5.1.2. Software y Hardware utilizado	31
5.1.3. Preprocesamiento de <i>Bounding boxes</i>	31
5.2. Evaluación de detectores utilizados	32
5.2.1. Extracción de imágenes por el detector	32
5.3. Evaluación de desempeño de diferentes clasificadores	34
5.3.1. Arquitectura general	34
5.3.2. Experimentación #1	35
5.3.3. Experimentación #2	36
5.4. Experimentación del clasificador	38
5.4.1. Experimentación #1 (training/validation)	39
5.4.2. Experimentación #2 (extracción de datasets desde <i>training</i>)	40
5.4.3. Experimentación #3 (extracción de datasets desde <i>training+testing</i>)	42
5.5. Experimentación del <i>pipeline</i>	43
5.5.1. Prueba del clasificador	44
5.6. Evaluación	46
6. Conclusiones y Trabajos Futuros	49
6.1. Conclusiones	49
6.2. Trabajos futuros	49
Referencias	51
A. Experimentación con distribución de datos diferentes	54
A.1. Experimentación A (training/ validation/ testing)	54
A.2. Experimentación #3 (training/ validation/ testing con <i>unfreeze</i>)	56

Capítulo 1

Contexto y Motivación

1.1. Introducción

En los últimos años *Machine Learning* (ML) ha ganado mayor importancia debido a su capacidad para aprender patrones, realizar regresiones y, especialmente, al potencial de los diferentes modelos de clasificación. En este contexto, *Deep Learning*(DL), una subárea del ML, ha demostrado ser superior al humano en diversas aplicaciones, como la detección, localización y clasificación de imágenes de rostros, números, lugares e incluso fauna y flora.

En la actualidad existen muchas aplicaciones a nivel global para el contexto de la fauna marina. Trabajos previos señalan que DL puede utilizarse en la investigación de recursos pesqueros, la obtención de conocimiento sobre poblaciones, el procesamiento de acuicultura, la protección de especies en peligro de extinción y el mantenimiento de la pesquería sostenible, entre otros (Lan, Bai, Li, y Li, 2020; Manandhar y Burris, 2019; Mejía, 2020; Chen, Sun, y Shang, 2017).

Dentro de estos trabajos, Nibha Manandhar y J. W. Burris (Manandhar y Burris, 2019), además de Mejía y Rosales (Mejía, 2020) utilizaron *transfer learning*(TL) para crear sus modelos, una técnica inspirada en el trabajo de Chen Guang, *et al.*(Chen y cols., 2017). Además, se han investigado modelos para la clasificación en el fondo marino, como el estudio de Suxia Cui, *et al.* (Cui, Zhou, Wang, y Zhai, 2020), quienes emplearon (Deng y cols., 2009) como *dataset* de imágenes para su entrenamiento.

Siguiendo esta línea de investigación, sería conveniente explorar los posibles usos en el contexto peruano, similar a lo que se hace a nivel mundial, para optimizar o automatizar los procesos actuales de detección y clasificación, además de crear nuevos *datasets* para futuros trabajos. Sin embargo, en el área de la clasificación de fauna marina peruana, Mejía y Rosales (Mejía, 2020) han

sido los únicos en realizar investigaciones sobre *pipelines* de ML y visión computacional, analizando modelos con un gran costo computacional lo cual los hace poco factibles para su recreación en tiempo real.

Por lo tanto, el objetivo de este trabajo es implementar un *pipeline* que mejore la escalabilidad de estos modelos, evitando generar un costo computacional elevado y reduciendo la pérdida de precisión, para que pueda utilizarse en tiempo real en la detección, clasificación y etiquetado de la fauna marina peruana. La estructura del documento se compone de: contexto y motivación, marco teórico, revisión de la literatura, metodología, experimentos y resultados, y por último, conclusiones y trabajos futuros.

1.2. Descripción del problema

Muchos investigadores en el campo del aprendizaje profundo (DL) han centrado sus esfuerzos en mejorar la precisión de sus modelos. Para lograrlo, han creado redes más complejas y requisitos cada vez más exigentes para su entrenamiento. Dentro de ellos, los modelos de detección y clasificación actuales requieren un *dataset* etiquetado para su entrenamiento. Este proceso de etiquetado suele ser largo, costoso y requieren especialistas, los cuales también cuentan con la posibilidad de cometer errores humanos. Todos estos problemas dificultan la creación de *datasets* peruanos para entrenar los modelos de vanguardia que pueden requerir miles de imágenes por especie.

1.3. Justificación

La creación de un *pipeline* no solo facilitaría la consolidación de nuevos *datasets* relacionados con la fauna marina peruana, sino también la detección y clasificación de objetos en general, los cuales podrían ser disponibilizados a los investigadores. Además, este *pipeline* tendría un bajo costo computacional, tanto para su entrenamiento como para su uso en entornos reales, aprovechando el conocimiento previo de redes ya entrenadas.

1.4. Objetivos

- Objetivo principal:
 - Proponer, desarrollar y probar un *pipeline* de DL para el etiquetado de imágenes complejas con múltiples especies de peces en la fauna marina peruana.
- Objetivos secundarios:
 - Unificar un *dataset* para el procesamiento de la fauna marina peruana.

- Comparar de manera práctica la precisión y el costo computacional de realizar el mismo *pipeline* para algunos modelos del estado del arte y obtener el más eficiente para su posible uso en un sistema.
- Automatizar el proceso de etiquetado para el beneficio de los pescadores inexpertos y entidades nacionales que no tengan automatizado este trabajo.

1.5. Aportes

Este trabajo no solo comparará diferentes arquitecturas de redes neuronales convolucionales (CNN) actuales, lo que facilitará la elección de un modelo para futuros investigadores, sino también demostrará que la creación de un pipeline para la detección y clasificación de imágenes permite aprovechar el aprendizaje de modelos preentrenados. Esto evitará la necesidad de entrenar modelos más complejos y permitirá etiquetar un conjunto de datos para futuros trabajos.

Capítulo 2

Marco Teórico

En el siguiente capítulo se revisarán algunos conocimientos previos que serán útiles para poder tener un mejor entendimiento tanto del problema de investigación como de la solución que se planteará. Entre estos conocimientos previos, se explicarán a detalle algunos de los modelos del estado del arte en temas de clasificación de imágenes utilizando ML, el procedimiento de *transfer learning*(TL) y *continuous learning* CL, las restricciones de un sistema a tiempo real, el pre-procesamiento de las entradas y el *dataset* que se planea utilizar para este trabajo.

2.1. Modelos de *Machine Learning* para clasificación

ML ha evolucionado a lo largo del tiempo, de tal manera que ahora los diferentes modelos pueden realizar trabajos mas complejos hasta el punto de sobrepasar en eficiencia y precisión a los humanos en algunas tareas. Dentro de estas tareas, las de clasificación son una de las áreas mas investigadas y la cual también presenta varias aplicaciones en el mundo real. Dentro de esta tarea, el *Multi Layer Perceptron* (MLP) y las *Convolutional Neural Networks* (CNN) son las dos arquitecturas dentro del estado del arte para el análisis de imágenes.

2.1.1. *Multi Layer Perceptron* (MLP)

Tal como lo dice su nombre, esta arquitectura consiste de múltiples capas de neuronas artificiales, las cuales reciben la data a ser procesada y es pasada por una función de activación, para que se conviertan en la entrada de la siguiente capa. Este proceso imita el trabajo de las neuronas humanas como se puede ver en la figura 2.1.

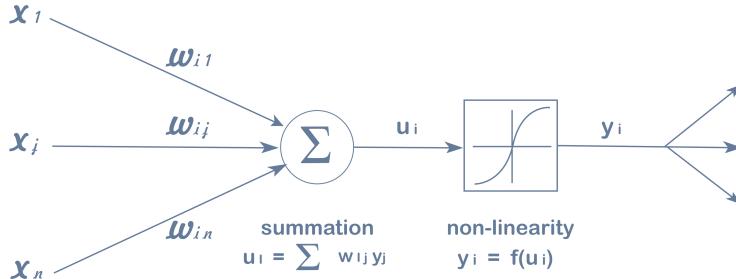


Figura 2.1: Comparacion entre una neurona real y una artificial(Magiquo, 2019).

Una MLP consta de varias capas “densas” de neuronas conectadas entre ellas. Una vez realizado el procesamiento de las entradas por toda la red neuronal, comienza el proceso de modificación de los pesos w de cada una de las capas. Este proceso se llama “retro propagación” o *back propagation* y como lo dice su nombre, se basa en propagar las derivadas de cada una de las funciones de las neuronas desde la última capa hasta la primera.

Este modelo espera como entrada un vector de características o de información, en ese sentido, no es especialmente bueno para poder analizar información como imágenes ya que realizar este proceso sería demasiado complejo computacionalmente hablando al tener una imagen de alta calidad, también por el hecho de que la extracción de características ya esta omitiendo información y, por último, que la información de una imagen suele cumplir el principio de localidad (la información relevante suele estar reunida en espacios cercanos) y vectorizarla estaría omitiendo características importantes para su análisis.

2.1.2. *Convolutional Neural Networks (CNN)*

Las CNN's son modelos basados en una arquitectura especialmente diseñadas para el análisis de imágenes y en especial, su clasificación. Ellas realizan la labor de extracción de características que no realiza el MLP a través de convoluciones. Esto evita la perdida de información y mejoran su eficiencia y precisión sobre todo.

Las convoluciones se basan en un procedimiento que permite extraer características al aplicar una matriz cuadrada de transformación (generalmente bastante pequeña) a lo largo de la imagen original, la cual luego devuelve otra imagen modificada. Estas matrices de transformación son llamadas kernels. En la figura 2.2 se ilustra una iteración de la convolución .

Por otra parte, existen otras capas importantes, los *poolings*, como se ilustra en la imagen 2.3. Estas capas consisten de la aplicación de una lógica a un cuadrante de la matriz resultante de las convoluciones. Esta lógica varía dependiendo de las necesidades del usuario. En la imagen mencionada

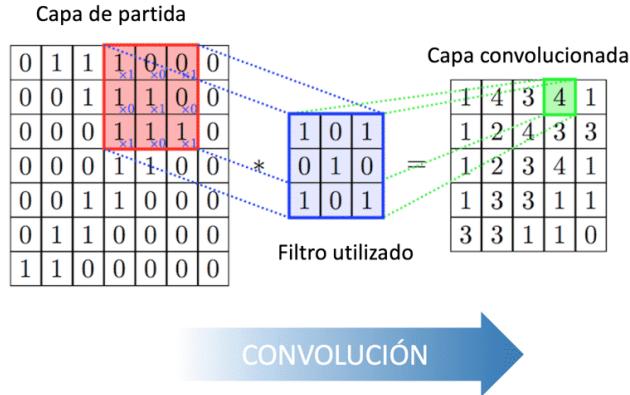


Figura 2.2: Proceso de convolución simplificado(Diego Calvo, 2019a).

anteriormente se puede ver la comparación entre el *Max Pooling* y el *Average Pooling*, las cuales tal como dicen sus nombres, obtienen el máximo y el promedio de los cuadrantes seleccionados para generar un nuevo resultado de menor dimensión.

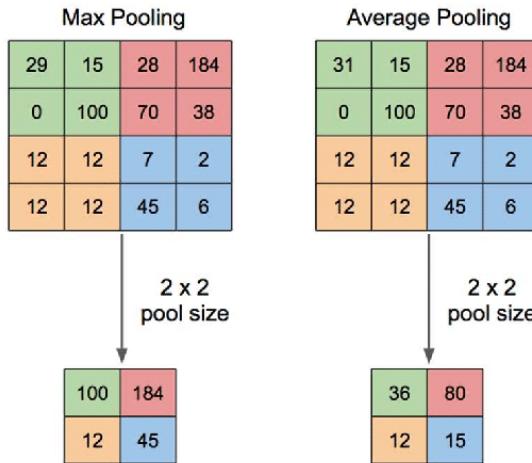


Figura 2.3: Proceso de convolución simplificado(Muhamad Yani, Budhi Irawan, Casi Setianingsih, 2019).

Las CNN's se conforman por la unión de capas convolucionales y *pooling* repetidas veces, terminando en una MLP, la cual realiza el trabajo final del procesamiento de las características extraídas. En la imagen 2.4 se puede ver la estructura de una CNN. Como se mencionó anteriormente, el proceso de la extracción de características se realiza dentro de la misma red neuronal, evitando perdida de información a comparación de la MLP, dejándole a esta únicamente

el trabajo de la clasificación.

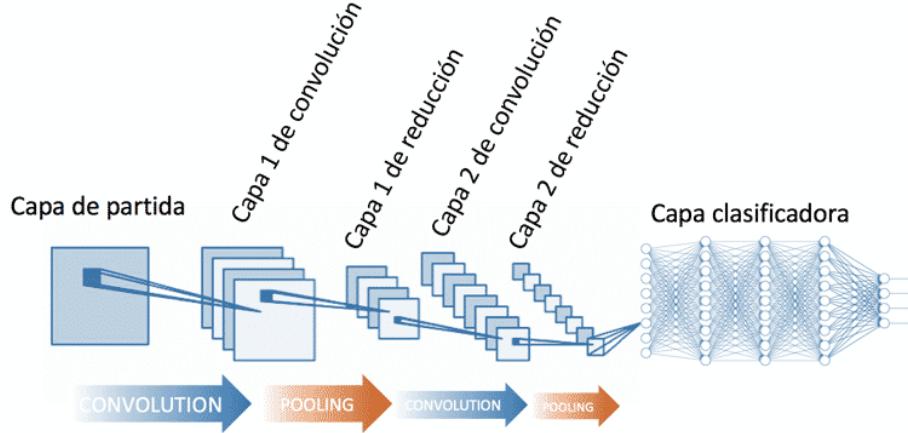


Figura 2.4: Arquitectura general de una CNN(Diego Calvo, 2019b).

2.2. Arquitecturas del estado del arte

2.2.1. VGG-19

Esta es una red neuronal convolucional que tiene 19 capas de profundidad. Fue construido y entrenado por Karen Simonyan y Andrew Zisserman(Simonyan y Zisserman, 2015) en la Universidad de Oxford en 2014, y posteriormente publicado en 2015. Su versión detallada se ilustra en la imagen 2.5. Este modelo fue utilizado para intentar clasificar las imágenes del *dataset* ImageNet, llegando a clasificar hasta 1000 objetos diferentes. Como mencionamos anteriormente, cada uno de los modelos necesita una entrada específica. En este modelo se espera una imagen de 224x224 píxeles para su procesamiento. Debido a su profundidad, este modelo es bastante pesado, llegando a consumir 550 MB de memoria con alrededor de 143 millones de características. Cabe resaltar que el 70 % de todas estas características son debido al paso entre la última capa convolucional y la primera de clasificación. Con todas estas características, logró obtener un resultado de 90 % de precisión con ImageNet.

2.2.2. InceptionV3

Si bien la anterior red lograba obtener una precisión muy elevada, esta consumía demasiados recursos. Es por ello que Google desarrolló InceptionV3, una red que prometió obtener los mismos resultados pero con menor costo.

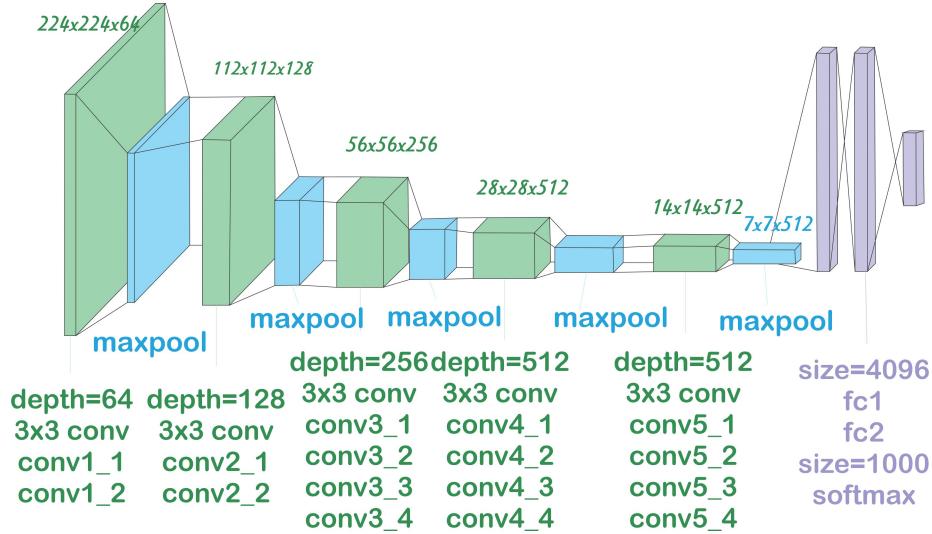


Figura 2.5: Versión simplificada de VGG19 (IchiPro, 2020).

Presentado en *"Going deeper with convolutions"* (Szegedy y cols., 2014), este modelo alcanza una precisión del 93.7% con el mismo *dataset*.

Si bien esta red contiene 50 capas de profundidad (las cuales superan a las de la VGG19), el numero de características a entrenar era menor (23.8 millones). Esto es debido a que esta red proponía que realizar convoluciones unidimensionales como se ve en la figura 2.6 en serie equivalían a realizar una bidimensional, pero con menores costos. En ese sentido disminuyeron la complejidad que se tenía en los modelos de CNN convencionales, haciéndolo mas ligero y aumentando la capacidad de aprendizaje del modelo. En total, este modelo llega a pesar 92 MB, lo cual es alrededor de 6 veces menor que el anterior. Su entrada requiere de imágenes de 299x299 píxeles y su arquitectura se ve reflejada en la figura 2.7

2.2.3. ResNet50

Creada por Microsoft en 2015, cuenta también con 50 capas profundas. Este modelo emplea lo que se le llama ‘aprendizaje residual’ (He, Zhang, Ren, y Sun, 2015), la cual consiste en guardar una copia de lo aprendido actualmente, y sumarlo al resultado después de aplicar una cantidad de convoluciones (en este caso cada tres). La imagen 2.8 ilustra esta modificación dentro del modelo además de su arquitectura general. Este modelo también fue probado con el mismo *dataset* obteniendo un 92.1% de precisión y gracias al aprendizaje residual se evitó aumentar la dimensionalidad del modelo. Esta red contiene 25.6 millones de características aproximadamente y llega a ocupar 98MB de memoria.

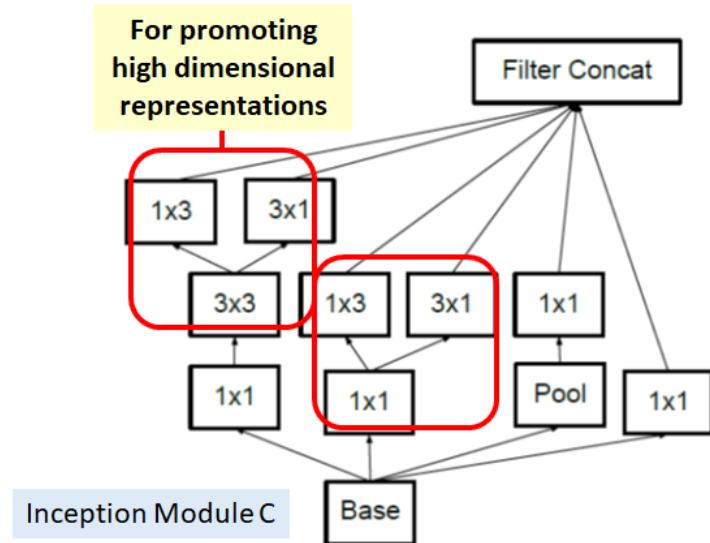


Figura 2.6: Reducción de la dimensionalidad del InceptionV3. (IchiPro, 2020)

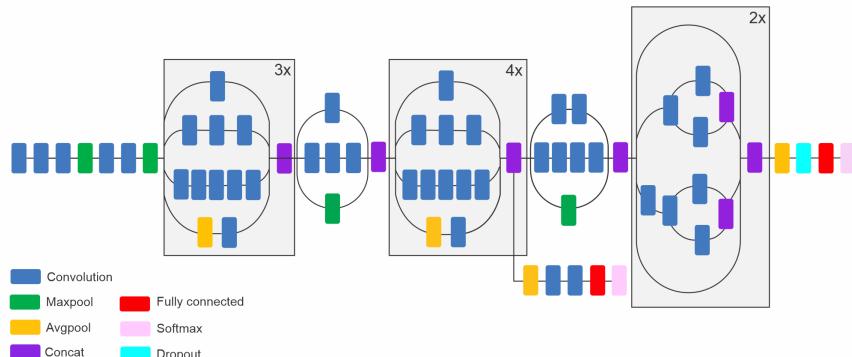


Figura 2.7: Versión simplificada de InceptionV3. (IchiPro, 2020)

2.2.4. EfficientNet

Creada por Google en 2019 y publicada en "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" (Tan y Le, 2020) en 2020 consta de ocho implementaciones diferentes (B0 a B7). La implementación mas liviana (B0) consiste de 5.5 millones de características aproximadamente siendo probada en el mismo *dataset* con una precisión de 93 %. Las demás implementaciones continúan aumentando el numero de características y a su vez la precisión que obtienen. Haciendo una comparación con los anteriores modelos, EfficientNetB4,

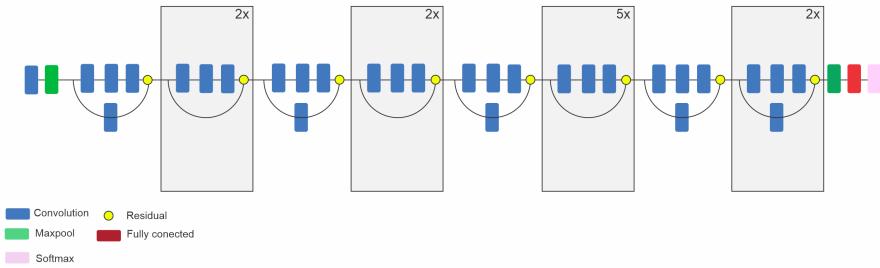


Figura 2.8: Versión simplificada de ResNet50. (IchiPro, 2020)

el cual consta de 19.5 millones de características, genera una precisión de 96.4 %, superándolos. La manera en la que este modelo optimiza su propio aprendizaje es a través de un algoritmo de aproximación que permite generar parámetros para la creación de cada uno de los ocho modelos. Este algoritmo toma en cuenta 3 factores:

- Profundidad de las capas
- Ancho de las capas (multicapa)
- Resolución de las imágenes

En la figura 2.9 se puede ver la estructura de la EfficientNetB0.

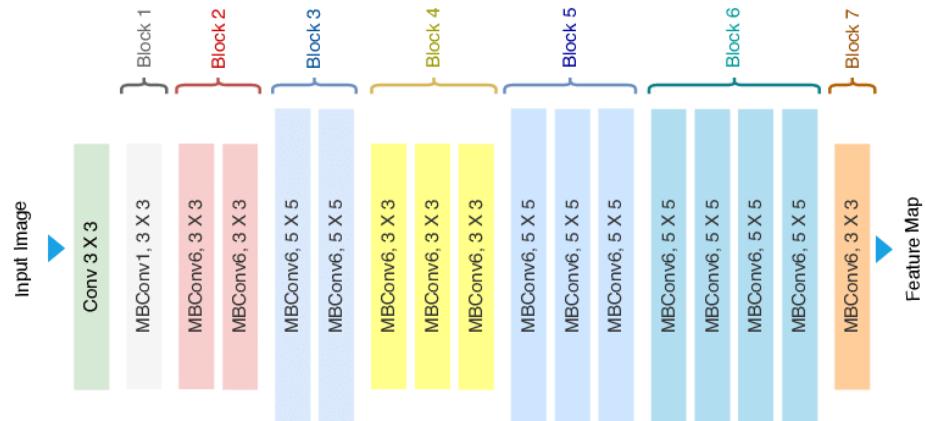


Figura 2.9: Versión simplificada de EfficientNetB0. (Tashin Ahmed, 2020)

2.2.5. YOLO

YOLO (*you only look once*) se presenta como una arquitectura que permite “predecir simultáneamente múltiples cuadros delimitadores (*bounding boxes*) y probabilidades de su clase para ellos” (Redmon, Divvala, Girshick, y Farhadi, 2015). Este modelo se basa en una CNN convencional, la cual a comparación de implementaciones previas(R-CNN y FR-CNN) realiza la predicción de los *bounding boxes* de manera interna, permitiendo una menor latencia, creando un modelo factible para su uso en tiempo real.

Este calculo se realiza en base a la generación de grillas (*grids*) o secciones de la imagen, dentro de las cuales se inicializan un número de *bounding boxes* predeterminados, ambos siendo hiperparámetros de la arquitectura. Esta es replicable utilizando cualquier arquitectura de CNN como base, como por ejemplo alguna de las anteriormente mencionadas, solamente ajustando la salida y hiperparámetros correspondientes. Versiones más modernas obtienen una mejor capacidad de detección, módulos de atención, entre otras variaciones que lo hacen cada vez más robustos. En la imagen 2.10 se puede ver ejemplificada la arquitectura del modelo yolov5.

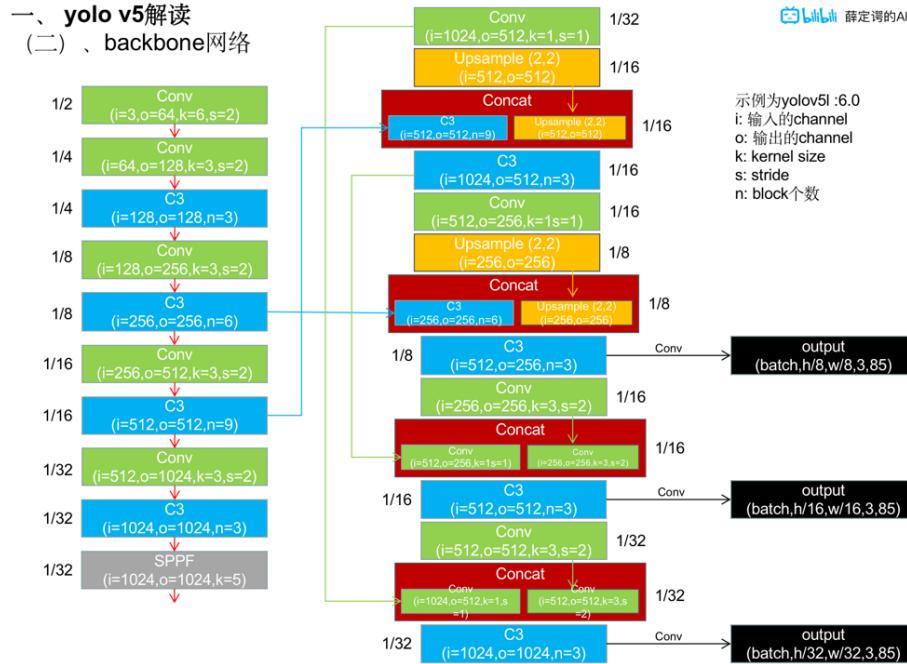


Figura 2.10: Representación de la arquitectura YOLOV5 (Jocher y cols., 2022)

2.2.6. UniDet

UniDet (*Unified Detector*) fue presentado en el año 2022 e intenta resolver el problema de la unificación de *datasets* para la creación de un modelo robusto que logre también predecir en *datasets* externos a los de entrenamiento, generando una precisión comparable al estado del arte. Esta arquitectura consta de un *backbone* (extractor de características), que pasa después a tres diferentes *heads* (quien realiza la detección y clasifica el *bounding box* por cada dataset para luego finalmente generar una respuesta unificada. Esta arquitectura se puede resumir en la imagen 2.11

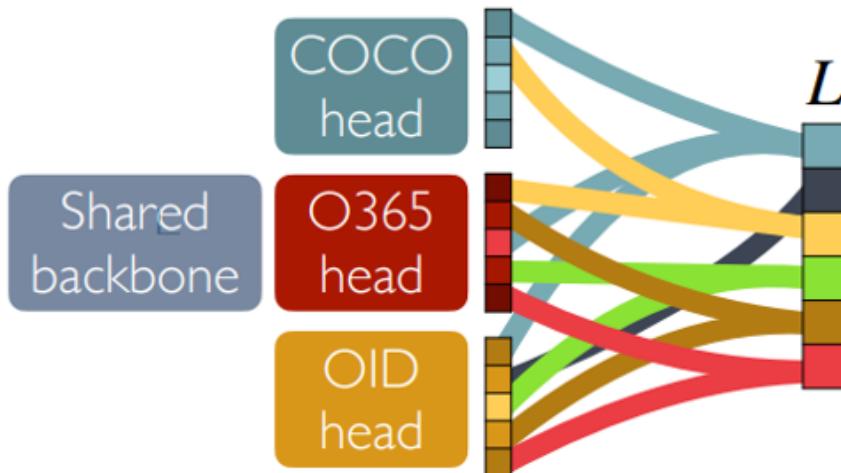


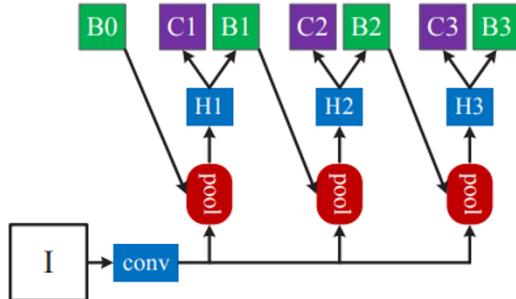
Figura 2.11: Representación de la arquitectura UniDet (Zhou y Krahenbuhl, 2022).

El *backbone* esta basado en una arquitectura Cascade RCNN, el cuál consiste de usos recursivos de capas convolucionales, *heads* y generación de predicciones para manejar imágenes borrasas utilizando una técnica de *resampling* a través de la recursividad mencionada. La arquitectura de este modelo está resumida en la imagen 2.12

Esta arquitectura representa el estado del arte en el tema de detección y clasificación en la mayoría de los *datasets* más comunes tales como COCO, Objects365, Open Images, VOC, VIPER, entre otros.

2.3. *Transfer Learning*

Según Muhamad Yani(Yani, Irawan, y Setiningsih, 2019), se le llama *Transfer Learning(TL)* al “proceso de la transferencia del conocimiento de un entrenamiento previo para ser usado en una nueva para la reducción de tiempo



(d) Cascade R-CNN

Figura 2.12: Representación de la arquitectura Cascade RCNN (Cai y Vasconcelos, 2017).

en el proceso de aprendizaje”. En la figura 2.13 se puede evidenciar este proceso.

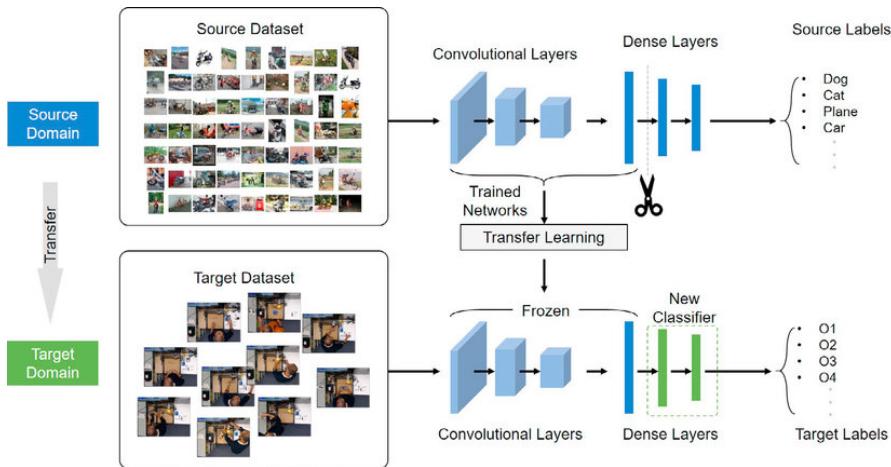


Figura 2.13: Comparación entre el aprendizaje común y transfer learning. (Wenjin Taoa, Md Al-Aminb, Haodong Chena, Ming C. Leua, Zhaozheng Yinc, Ruwen Qinb, 2020)

Transfer Learning difiere al proceso convencional de entrenamiento de una red ya que no es necesario entrenarlo con un *dataset* grande. En contraste con el proceso convencional, se congelan las capas iniciales del modelo (en nuestro caso las capas convolucionales), las cuales tienen todo el conocimiento pre-aprendido por la red sobre el *dataset* con el que fue diseñado. Una vez obtenidas esas capas, se les anexan nuevas capas densas (iguales a las de un

MLP) para servir como los clasificadores para nuestro uso. Gracias a esto, únicamente es necesario entrenar las capas finales, lo cual no necesita tantos datos y generando predicciones generalmente certeras. A este tipo de entrenamiento se le llama “*fine tuning*” o ajuste, que permite a la red modificar el aprendizaje anterior para poder predecir en base a un *dataset* diferente al que fue originalmente entrenado, ahorrando tiempo y mejorando su precisión.

2.4. Pre-procesamiento de entradas

Teniendo conocimientos generales acerca de como funcionan estos modelos, necesitamos ahora conocer acerca de las características necesarias de las entradas para que el modelo aprenda.

2.4.1. Tamaño de entrada

Tomando únicamente en cuenta a las diferentes CNN, cada uno de los modelos espera una matriz (la imagen) de diferente tamaño. En las implementaciones actuales, se suele utilizar tensores para referenciar un *batch* (conjunto) de imágenes. La representación es la siguiente:

$$(batch, canales, m, n)$$

en donde:

- batch: Numero de imagenes que se estan ingresando a la vez
- canales: Numero de canales de la imagen (en caso de ser a color, serian 3 canales representando RGB, mientras que a blanco y negro sería solo 1 canal)
- m y n: dimensiones de la imagen

Las dimensiones de la imagen depende de la arquitectura del modelo ya que cada capa realizará operaciones que irán disminuyendo la dimensionalidad de la misma. Esto varía con cada implementación debido a las diferentes configuraciones que se pueden hacer a cada matriz convolucional y a los *pooling*.

2.4.2. One Hot Encoder

Este es un tipo de representación el cual consiste en la creación de una matriz de identidad de tamaño $n \times n$, donde n es el numero de *labels*. La codificación de cada uno de los labels se puede tomar como una de las filas de la matriz. En ese sentido podemos representar la codificación de un objeto de la clase j, de n clases como:

$$OneHotEncoder(i, j, n) = [a_1, a_2, \dots, a_n]$$

$$a_{i,j} = \begin{cases} 1 & j = i \\ 0 & \text{Caso contrario} \end{cases}$$

Este tipo de codificación trae como ventaja el hecho de que se evita la necesidad de tener una relación entre los labels, además de permitir obtener una clasificación directa a la hora de comparar los resultados. Por contraparte, tiene como desventaja la alta dimensionalidad si se tienen muchos *labels*.

2.5. *Overfitting*

El proceso por el cual los modelos aprenden es gracias a la retroalimentación que se obtiene realizando la retroprogacación mencionada anteriormente. Una vez actualizados los pesos de cada neurona con la gradiente que se obtiene, se podría afirmar que el modelo ya logró aprender a clasificar esa imagen específica. Sin embargo, esta gradiente se puede desvanecer debido a la profundidad de la arquitectura, las funciones de activación, entre otros motivos. Este desvanecimiento de la gradiente evita que el modelo continue aprendiendo del *dataset*, haciendo al modelo inservible para usos reales. Algunas de las estrategias aplicadas son:

- *Data Augmentation*: Aumentar el *dataset* original en base a rotaciones, escalados, cortados, simetría, etc. Con ello el entrenamiento del modelo se hace más resistente a estos cambios en la posición o rotación del objeto en la imagen.
- *Batch Normalization*: Re-escalar los datos de entrada a en diferentes escalas respecto a una escala común para poder generar una distribución de los datos más manejable.
- *Dropout*: Desactivar de manera aleatoria un porcentaje de las neuronas artificiales de una capa. Esto obliga a cada neurona a no depender de las neuronas desactivadas, generando una mejora en general.

En el presente capítulo se presentaron conocimientos previos que servirán al lector a entender un poco más acerca de la metodología que en el futuro se va a presentar, en especial las arquitecturas de los modelos que se planea revisar, utilizar y comparar para poder obtener el mejor resultado posible para la detección y clasificación de imágenes de peces dentro de la fauna marina peruana.

Capítulo 3

Revisión de la Literatura

En el presente capítulo se analizarán los modelos y técnicas de DL para la clasificación de imágenes y su impacto en algunas aplicaciones actuales. Como ya había sido mencionado, los modelos de DL son los que en la actualidad han reemplazado a varios algoritmos y al humano mismo en las diferentes tareas de regresión, clasificación, detección y localización de objetos en imágenes. Esto gracias a su gran precisión en esta tarea y a la automatización que traen a las diferentes empresas y instituciones gubernamentales.

Alsmadi y Almarashdeh (Alsmadi y Almarashdeh, 2022) lo comprobaron al realizar un *survey* donde revisaban y comparaban trabajos con diferentes enfoques. En la investigación, se compararon algoritmos de visión computacional, estadística, ML, DL, entre otros aplicados al contexto de extracción de características, detección y clasificación. Corroboraron que la eficacia y velocidad de los modelos de DL para ello fue superior a los demás algoritmos clásicos.

Dentro del sub-área de DL, Xiaojuan Lan, *et al.*(Lan y cols., 2020) son los autores revisados más recientes. Ellos diseñaron un modelo utilizando TL en base a Inception-V3, una CNN, para resolver el problema de la clasificación de imágenes borrosas de peces en el fondo marino con una gran precisión (arriba del 85 %) y de manera más eficiente que otros modelos pre-existentes. Un punto que cabe resaltar de este trabajo es que detallan que utilizaron InceptionV3 como base para el TL, pero no mencionan que capas del modelo se congelaron y cuáles fueron cambiadas para su adaptación a su *dataset*, y saltan directamente a las conclusiones y dan como resultado la precisión promedio, lo cual podría esconder que ellos obtuviesen un modelo *overfitted*, considerando también la baja cantidad de imágenes de su *dataset*.

Detrás de ellos están Nibha Manandhar y John W. Burris (Manandhar y Burris, 2019), quienes utilizaron las mismas herramientas mencionadas anteriormente, pero enfocándose a una problemática diferente. Ellos no se

preguntaban si la entrada era un determinado tipo de pez, mas bien, se preguntaban si “esta imagen contenía esa determinada especie dentro”, ya que no eran imágenes del hábitat de los peces, sino imágenes recuperadas de Google, las cuales podían contener seres humanos u otro entorno y no únicamente el pez. Con este enfoque, lograron obtener una precisión del 85 % en la etapa de entrenamiento.

De la misma manera que en el trabajo de los anteriores autores, no especifican a detalle como es que se está realizando el TL en el modelo. Además, en la sección de resultados muestran que se generó una buena precisión solo en algunas especies, mientras que en otras rondaba alrededor de 50 a 70 %

Por último, Guang Chen, *et al.* (Chen y cols., 2017) realizaron un modelo en base a dos ramas, una para la clasificación a nivel de imagen y la otra para la clasificación a nivel de instancia de las imágenes. Mientras que la primera se basaba únicamente en un modelo sencillo de CNN, la segunda se basaba en un proceso de cuatro pasos:

- Detección: Utilizando los modelos YOLOv2 y SSD, realizando el cálculo de los *bounding boxes*, los cuales contienen coordenadas de las dos esquinas del rectángulo que contiene al pez.
- Estimación de la pose: Se maneja otro modelo de CNN para definir la dirección del pez como un problema de clasificación (se está clasificando la dirección hacia donde apunta la cabeza del pez).
- Alineamiento: Se rota la imagen para que el pez mantenga una dirección horizontal y mirando hacia la derecha.
- Predicción: Se aplica otro modelo de CNN para la clasificación de la imagen obtenida.

Hay varios aspectos positivos en este trabajo. El primero es que nos da detalle acerca de todo el procedimiento que se necesitó para realizar la implementación de su modelo. Segundo, combinan muchas técnicas y modelos para hacer más robusta su propuesta. Por último, obtuvieron un resultado que no solo clasificaba al pez como tal, sino que utilizaba características del entorno y de los objetos cercanos para poder hacer su labor de clasificación.

Por otra parte, considero que la estimación de la pose y el alineamiento son pasos que no eran necesarios en este *pipeline* ya que las CNN's del estado del arte suelen ser resistente a este tipo de problemas. Esto podría haber tenido influencia en el costo computacional del mismo, haciéndolo más lento y podría haber tenido una mejor precisión sin ello.

Dentro de la problemática de localización de fauna marina, Suxia Cui,*et*

al.(Cui y cols., 2020) desarrollaron un modelo para localizar y clasificar los peces dentro del agua a través de un *Autonomous Underwater Vehicle (AUV)*. El modelo consistió en 24 capas convolucionales, a las cuales les siguen dos capas *full connected*. Ellos se inspiraron en la arquitectura del modelo YOLO para poder realizar la localización de los peces.

Al igual que el trabajo anterior, uno de los puntos mas favorables de este es que documenta tanto el procedimiento como los métodos utilizados, además de evidenciar unos buenos resultados para la localización y clasificación de múltiples peces dentro de una imagen dentro del océano, el cual ya es de por sí un gran desafío. Un problema que suele ocurrir con este tipo de modelos es la falta de data etiquetada para el entrenamiento de los modelos, haciendo inviable la creación de este tipo de modelos en casos reales.

Dentro del área de Visión Computacional(VC) se encuentra el trabajo de Mejía y Rosales (Mejía, 2020), quienes son los únicos autores que han investigado sobre el uso de técnicas de DL para la clasificación de la fauna marina peruana por el momento. Ellos se basaron en el documento revisado anteriormente para su trabajo. Si bien desarrollaron como solución final un modelo de DL para la clasificación, este era retro-alimentado a través de varios algoritmos de visión computacional.

Entre estos algoritmos de visión se encuentra el algoritmo SURF y SIFT, el cual trabaja con una silueta de un pez para obtener características, de las cuales se eliminan el *background* y se identifican los bordes del pez, siendo este resultado analizado y después usado para el entrenamiento del modelo. Con este, lograron obtener un 80 % de precisión en la etapa de testeo.

El mayor aporte de este trabajo ha sido la experimentación de nuevas entradas para las CNN's a través de un pre-procesamiento, queriendo mejorar la precisión del modelo final. Aún así, pareciera que el valor de precisión obtenido podría tener un *overfit* ya que no muestra ejemplos de las imágenes utilizadas, el número de ellas y a la alta precisión final. Por otra parte, hace poco sentido que realicen este pre-procesamiento considerando nuevamente que las CNN's fueron diseñadas para que las imágenes sean pasadas enteramente como entradas.

Como se pudo ver a lo largo de este capítulo, hubo una evolución de los algoritmos de detección y clasificación de imágenes, lo cual llevó a diferentes investigadores a utilizar algoritmos de ML y DL para estas tareas. En el presente capítulo se presentaron algunas de las investigaciones del estado del arte enfocadas en modelos de CNN y YOLO para la detección y clasificación de imágenes en diferentes *datasets*. Dentro de ellas, se pudo ver diferentes pipelines que lograron obtener una precisión muy elevada para problemas complejos, por lo cual el presente trabajo planea seguir con esa línea de

investigación y proponer un nuevo pipeline para mejorar la detección y clasificación de peces dentro de la fauna marina peruana.

Capítulo 4

Metodología

En el presente capítulo se presentará el *pipeline* que resume la metodología que se planea utilizar en la experimentación detallada en los siguientes capítulos. La figura 4.1 ilustra el pipeline que se propone para la realización de los experimentos.

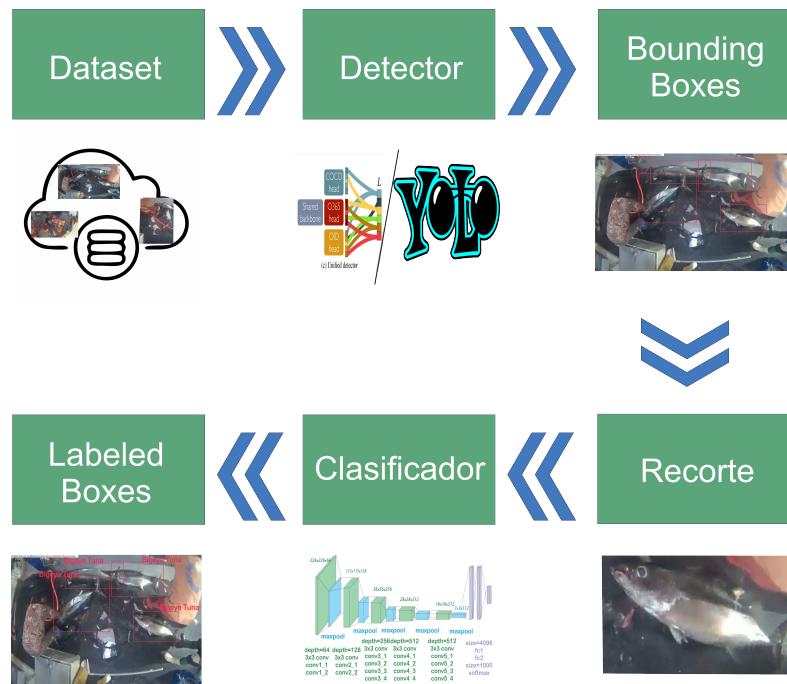


Figura 4.1: Pipeline propuesto.



(a) Imagen de un *dataset* “poco complejo” obtenido de *Fish Species*



(b) Imagen de un *dataset* “complejo” obtenido de *The Nature Conservancy*

Figura 4.2: Comparación de imágenes dentro de datasets diferentes

4.1. *Dataset*

Como se mencionó previamente, existen varios *datasets* disponibles para cada problema que se pretende resolver. Sin embargo, es importante destacar que existen distintos tipos y se puede observar que algunos son más o menos complejos para ciertas tareas.

En el caso de nuestra tarea de detección y clasificación de imágenes, podríamos considerar un *dataset* de baja complejidad como aquel que contiene imágenes con objetos centrados, alineados en una sola dirección, con una resolución óptima y enfocados en el centro. Se ha demostrado en el marco teórico y en la revisión del estado del arte que este tipo de conjuntos de datos son bastante fáciles de clasificar, especialmente para modelos como los mencionados anteriormente.

En contraste, un conjunto de datos complejo difiere considerablemente de esta descripción. Las imágenes pueden contener varios objetos que no son los que se deben clasificar, presentando desalineación, baja resolución o falta de enfoque. Estas características dificultan que las redes neuronales convolucionales (CNN) logren una alta precisión, ya que no pueden aprender todos los patrones y realizar predicciones precisas para cada objeto. La Figura 4.2 ilustra claramente la marcada diferencia entre estos dos tipos de conjuntos de datos.

Además, existen clasificaciones de *datasets*, cada uno con una función de distribución de imágenes y un objetivo específico. En la primera categoría, encontramos conjuntos de datos homogéneos y heterogéneos. Los *datasets* homogéneos se caracterizan por tener la misma cantidad (o un número similar) de muestras por clase, lo que permite que los modelos aprendan de manera equitativa de cada clase. Sin embargo, esto no siempre refleja la realidad, donde la probabilidad de que aparezca una clase de objeto determinada puede

ser mayor o menor que otra.

En ese sentido, los conjuntos de datos heterogéneos son aquellos que no tienen un número igual de muestras para cada clase. Esto puede ser intencional, ya que el creador del conjunto de datos diseñó la distribución de muestras de esa manera, o puede ser resultado de la falta de muestras disponibles.

Además, los *datasets* suelen venir etiquetados en relación a su objetivo. En nuestro caso, para entrenar una CNN, se dispone de un repositorio de imágenes junto con sus etiquetas correspondientes. En cambio, para entrenar una red como YOLO, se requieren los “bounding boxes” que delimitan la ubicación de los objetos.

Dado que encontrar un *dataset* de este tipo resulta inviable para el caso peruano, se han utilizado dos tipos de conjuntos de datos encontrados en la plataforma Kaggle para esta investigación. Ambos conjuntos contienen imágenes y sus respectivas etiquetas, pero uno presenta peces centrados y sin ruido, mientras que el otro incluye fotografías tomadas en un entorno real donde se encuentran peces, personas e instrumentos de pesca, lo que lo convierte en un conjunto de datos heterogéneo. El primero se utilizará para realizar comparaciones entre los modelos, mientras que el segundo se empleará para el entrenamiento y prueba del pipeline final.

Estas imágenes serán re-dimensionadas dependiendo del tamaño de entrada que se espera para cada red, para luego ser pasadas al clasificador.

4.2. Detector

Para la detección de cada imagen del *dataset*, utilizaremos un módulo de detección basado en YOLO v5, el cual ya ha sido preentrenado con el conjunto de datos “Objects365”, brindando la capacidad de detectar la categoría de peces. Además, se entrenó un YOLO v5 con el *dataset*, ignorando las clases detectadas. Por último, también utilizaremos un modelo llamado UniDet, el cual ha sido entrenado con los *datasets* “Objects365”, “OpenImages” y “COCO”. Una vez que las imágenes sean procesadas por el detector, esperamos obtener los *bounding boxes* donde se encuentren los peces individualmente. Se puede visualizar este proceso en la imagen 4.3.

4.3. Clasificador

Una vez ubicados los peces, se procede a recortar y redimensionar las imágenes para ocupar la entrada que espera el clasificador. Para los experimentos, se utilizará la resolución específica para cada una de las redes a



Figura 4.3: Flujo de entrada y salida del detector

probar. Cada una de estas redes también serán obtenidas con sus pesos pre-entrenados con “Imagenet”, pero a ellas se les aplicará el proceso de TL, congelando todas las capas convolucionales. Por último, a cada uno de estos modelos se les añadirá una capa llamada “*GlobalAveragePooling*” y finalmente ser pasada a las capas clasificadoras. Se espera que con este procedimiento se disminuya el número de parámetros finales que pasarán a la capa de clasificación y mejorar también su precisión y reducirá su tiempo de entrenamiento. El flujo se ve evidenciado en la imagen 4.4.



Figura 4.4: Flujo de entrada y salida del clasificador

4.4. *Labeled Boxes*

Después de haber clasificado todas las imágenes de las instancias de peces obtenidas por el detector, se procederán a obtener todas las etiquetas obtenidas, asignándolas a cada uno de los *bounding boxes* que se habían generado anteriormente. En ese sentido se obtendrá como salida la imagen con los *labeled boxes* para cada instancia de pez que se encuentre.

4.5. Evaluación de modelos

Orhan Yalsin, en 2020 realizó un resumen acerca de algunas métricas que ayudan a comparar algunos de los modelos del estado del arte, de lo cual nos guiarímos para la evaluación de los mismos. Los datos se resumen en la tabla 4.1.

Dentro de esta tabla se pueden resaltar varios modelos interesantes: VGG19 (si consideramos solo las capas convolucionales), ResNet50, InceptionV3, y los EfficientNetBX resultan tener una buena relación entre el

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	0.771	0.933	5,330,571	-
EfficientNetB1	31 MB	0.791	0.944	7,856,239	-
EfficientNetB2	36 MB	0.801	0.949	9,177,569	-
EfficientNetB3	48 MB	0.816	0.957	12,320,535	-
EfficientNetB4	75 MB	0.829	0.964	19,466,823	-
EfficientNetB5	118 MB	0.836	0.967	30,562,527	-
EfficientNetB6	166 MB	0.840	0.968	43,265,143	-
EfficientNetB7	256 MB	0.843	0.970	66,658,687	-

Cuadro 4.1: Evaluación de diferentes modelos. Tabla brindada por Orhan Yalcin (Yalcin, 2020).

peso, el número de parámetros y la precisión que estas obtienen, esta última siendo de las más altas dentro de todos estos modelos. Se tendrá en consideración que este pipeline planea ser ejecutado en videos en tiempo real y por ello se necesita que el modelo sea el más pequeño posible evitando perder la precisión. Por otra parte, están las MobileNet, ya han sido utilizadas para el procesamiento de imágenes en tiempo real por ser más ligeras, enfocadas a ser utilizadas en celulares. Creemos que todas estas redes son aptas para este problema y pasarán a ser evaluadas en el siguiente capítulo, en base al primer *dataset* mencionado anteriormente.

Una vez analizado esas redes en base al *dataset*, se compararán la precisión y costo entre ellas y se obtendrá la que finalmente pasará a formar parte del

pipeline final, en donde será encargada de clasificar cada imagen obtenida por la red Yolo.

En el siguiente capítulo finalmente se verá la experimentación con todos los modelos anteriormente señalados, tanto como la especificación de las imágenes y la experimentación del *pipeline* final.

Capítulo 5

Experimentaciones y Resultados

Considerando el *pipeline* mencionado en el anterior capítulo, ahora se procederá a explicar a detalle el proceso de experimentación que ha sido realizado para el presente trabajo.

5.1. Aspectos generales

5.1.1. *Datasets*

Como ya había sido mencionado anteriormente, se utilizará un banco de imágenes de diferentes especies peruanas, las cuales fueron recogidas de diferentes repositorios y competencias de la página *Kaggle*. Estos *datasets* contienen las imágenes de peces y sus *labels*, los cuales representan las especies del pez que se encuentra dentro de la imagen. Se obtuvieron 3 diferentes *datasets*, de entre los cuales se extrajeron únicamente las especies que también se encuentran en el mar peruano y de entre ellos, el primero se utilizará para la comprobación del *pipeline* final, mientras que los otros dos se utilizaron para probar el clasificador. La lista de cada uno de los *datasets* se encuentra a continuación:

- *The Nature Conservancy Fisheries Monitoring*: Concurso realizado en 2017 a través de la plataforma mencionado anteriormente. Contiene varias imágenes sobre las siguientes especies: atún blanco o “bonito”, atún de ojo grande, atún de aleta amarilla y pez delfín o comúnmente llamado “perico”.
- *Fish Species*: Contiene 2000 imágenes de cada una de las 20 especies del mediterráneo de entre las cuales se ha recolectado las imágenes de la lisa o “*Mugil cephalus*”, el pez guitarra o “*Rhinobatos cemiculus*”, la caballa o “*scomber japonicus*” y el pez espada o “*tetrapurus belone*” .

- *A Large Scale Fish Dataset*: Contiene 1000 imágenes de 9 especies diferentes encontradas en Turquía, de entre las cuales, una de ellas, la trucha marrón, también se encuentra en aguas peruanas.

5.1.2. Software y Hardware utilizado

A lo largo de toda la experimentación, se utilizará un computador de escritorio con las siguientes características:

Componente	Descripción
Procesador	AMD Ryzen 7 3700X 8-Core 3.6 GHz.
Memoria RAM	Crucial Ballistix 16GB DDR4 - 3600MHZ
Tarjeta Gráfica	Nvidia RTX 2060 6GB

Por otra parte, se utilizó Python, junto con Keras, sklearn y tensorflow para el desarrollo del *pipeline* en general y el pre y post procesamiento de los datos. Para el procesamiento de las imágenes (con Yolo) se utilizaron opencv compilado con cudnn para la habilitación del uso de tarjeta gráfica.

5.1.3. Preprocesamiento de *Bounding boxes*

Se llevó a cabo la creación manual de los *bounding boxes* para todas las imágenes del *dataset* “The Nature Conservancy”, utilizando el software de código abierto *YoloLabel*, el cual proporciona una interfaz sencilla. Se contó con ayuda de un experto para crear los *bounding boxes* y etiquetar cada imagen.

Es importante mencionar que, aunque las imágenes en general eran borrosas, las del conjunto de entrenamiento resultaron ser menos complejas que las del conjunto de prueba. De hecho, estas últimas fueron difíciles de clasificar incluso para el experto. Una vez creados los *bounding boxes*, se procedió a recortar las imágenes. Como resultado de este proceso, se obtuvieron diferentes cantidades de imágenes por clase, como se muestra en la tabla siguiente:5.1.

Una observación crucial que debe ser mencionada es la presencia de heterogeneidad en ambos *datasets*, lo cual se asemeja a la realidad, ya que algunos de los peces son más difíciles de capturar que otros. Por lo tanto, este caso de estudio también involucra otros desafíos, como clases desequilibradas y un número bajo de imágenes, lo que puede provocar *overfitting* en muchos casos, ya que debido a la disparidad de clases y la complejidad de los modelos, existe la posibilidad que se categorice las clases con más instancias (Calvo, 2022).

Especie	Número de imágenes de entrenamiento	Número de imágenes de prueba
<i>Albacore Tuna</i>	2341	383
<i>Bigeye Tuna</i>	288	391
<i>Dolphinfish</i>	127	20
<i>Moonfish (LAG)</i>	98	59
<i>Shark</i>	300	131
<i>Yellowfin Tuna</i>	195	21
<i>Other</i>	778	164
Total	4127	1169

Cuadro 5.1: Número de imágenes por *dataset*

5.2. Evaluación de detectores utilizados

En esta primera experimentación se realizarán pruebas con 3 diferentes modelos: Yolov5 preentrenado con el *dataset* “Objects 365”, UniDet y Yolov5 entrenado con nuestro *dataset*. Para ello, se obtuvo un conjunto aleatorio del 20 % de las imágenes recortadas entre el conjunto de entrenamiento y prueba, mientras que para el entrenamiento del Yolov5, se utilizaron todos los datos de entrenamiento y prueba.

5.2.1. Extracción de imágenes por el detector

Una vez obtenidos el 20 % de las imágenes del *dataset* “The Nature Conservancy” en donde se unificaron los conjuntos de entrenamiento y prueba (acción que se justificará en la sección 5.4.1 y que está evidenciada en el anexo A). se utilizaron tres diferentes detectores para el recorte de las imágenes. Para cada uno de ellos, se clasificaron los recortes por la etiqueta de la imagen de la cual fueron obtenidas y las que eran de prueba fueron nuevamente etiquetados por el experto. Por otra parte todas las imágenes recortadas que no contenían un pez dentro, no estaban enfocadas o no contenían perfectamente al pez, fueron etiquetadas como errores. Para esta experimentación se utilizaron dos yolov5 (uno preentrenado con Objects365 y uno entrenado con el *dataset*) y un UniDet, entrenado con COCO, Objects 365 y OpenImages. Los datos obtenidos fueron recopilados en la tabla 5.2. Cabe resaltar que el yolov5 entrenado fue utilizado como detector general (ignorando las clases que predecía).

Como se esperaba, utilizar un modelo preentrenado no especializado en detectar una clase específica genera una menor precisión a comparación de uno que si está entrenado y más aún en el *dataset* objetivo. Se utilizaron dos métricas para el análisis de estos datos: el porcentaje de peces detectados con

	Peces detectados	Otros objetos (errores)	Total
Real	1061	-	1061
Predicted Yolov5 pretrained(40 % threshold)	150	335	485
Predicted UniDet pretrained (no threshold)	426	124	550
Predicted Yolov5 trained (freezed backbone 95 % threshold)	843	399	1233

Cuadro 5.2: Número de peces vs errores detectados por cada modelo

respecto del *dataset* original y el porcentaje de error general obtenido por cada detección. Estos resultados se pueden ver en la tabla 5.3.

	Porcentaje de peces detectados	Porcentaje de errores detectados del total
Predicted Yolov5 pretrained(40 % threshold)	14.13 %	69.07 %
Predicted UniDet pretrained (no threshold)	40.15 %	22.54 %
Predicted Yolov5 trained (freezed backbone 95 % threshold)	79.45 %	32.36 %

Cuadro 5.3: Porcentaje de detecciones correctas vs errores generados

Se puede ver una baja precisión por parte del yolov5 preentrenado, lo cual es comprensible considerando que no ha sido preentrenado especialmente para detectar peces, sino con un *dataset* genérico y es un modelo bastante ligero. UniDet, en cambio, es un modelo mucho más complejo, y tiene 3 detectores internamente (uno para cada dataset), los cuales en conjunto predicen la clase final del objeto y es por lo cual generan un mejor resultado que el anterior pero a un costo mucho mayor. Finalmente el yolov5 entrenado específicamente con el *dataset* obtenido muestra una especialización en la detección de estos individuos que finalmente logra un 79.45 % de precisión. Aún así se necesita considerar el hecho que el entrenamiento de este modelo toma alrededor de 80 minutos, siendo este aproximadamente 13 veces más el tiempo que le toma entrenar a un modelo como lo es EfficientnetB0 con un bajo número de imágenes(4300 imágenes, a comparación de 5300 para el CNN).

Además, al considerar el porcentaje de errores generados, se observa que UniDet produce una menor cantidad de detecciones incorrectas en comparación con *pretrained* YOLOv5 y *trained* YOLOv5, lo que demuestra que es un modelo más robusto. Sin embargo, debido a la falta de especialización durante su entrenamiento, UniDet ofrece resultados más pobres en las detecciones. En este sentido, se cree que si tuviéramos los pesos de un modelo específicamente entrenado para la detección de peces, se mejoraría el resultado y se evitaría la necesidad de entrenar uno desde cero o utilizar uno que haya sido entrenado de manera genérica. Esto aumentaría la precisión general del sistema y tendría un impacto positivo en el flujo de trabajo.

5.3. Evaluación de desempeño de diferentes clasificadores

En la presente sección se realizarán dos experimentos para poder seleccionar una arquitectura que será utilizada para actuar como nuestro clasificador dentro del *pipeline* final. Para ello, se compararán algunas arquitecturas del estado del arte utilizando los 3 *datasets*, analizando tanto las pérdidas y precisión en *training*, *validation* y *testing* para cada uno, así como la velocidad por batch (8 imágenes) y el número de épocas necesarias para entrenar dichas arquitecturas.

5.3.1. Arquitectura general

Como fue mencionado, para la creación de cada una de las arquitecturas se utilizarán modelos ya preentrenados del estado del arte, a los cuales se les aplicará TL para su entrenamiento con nuestro *dataset*. Este proceso consistirá en congelar todas las capas convolucionales, para luego ser pasado por una capa de *GlobalAveragePooling2D*, luego por una *Flatten*, la cual pasa de un formato bidimensional a uno unidimensional y luego por las siguientes tres capas:

- *BatchNormalization*
- *Dropout* de 50 %
- Capa densa a X *outputs* con una función de activación RELU o softmax, dependiendo si es una capa oculta o la de salida.
- 8 imágenes por batch

Estas tres capas son ejecutadas para cada uno de los siguientes tamaños: [1024, 256, 64, *LABELS_SIZE*], donde *LABELS_SIZE* representará el número de clases de salida que generará el clasificador.

5.3.2. Experimentación #1

Para el primer análisis, se utilizaron las consideraciones anteriores para comparar las diferentes arquitecturas del estado del arte mencionadas en el anterior capítulo. Para esta comparación, se utilizaron las imágenes obtenidas de los *datasets Fish Species* y *A Large Scale Fish Dataset* combinados y se compararon los resultados de las métricas obtenidas. La primera de las métricas a comparar serán las gráficas de pérdida. En la figura 5.1 se ilustra una comparativa de la disminución de la pérdida con respecto al tiempo (épocas).

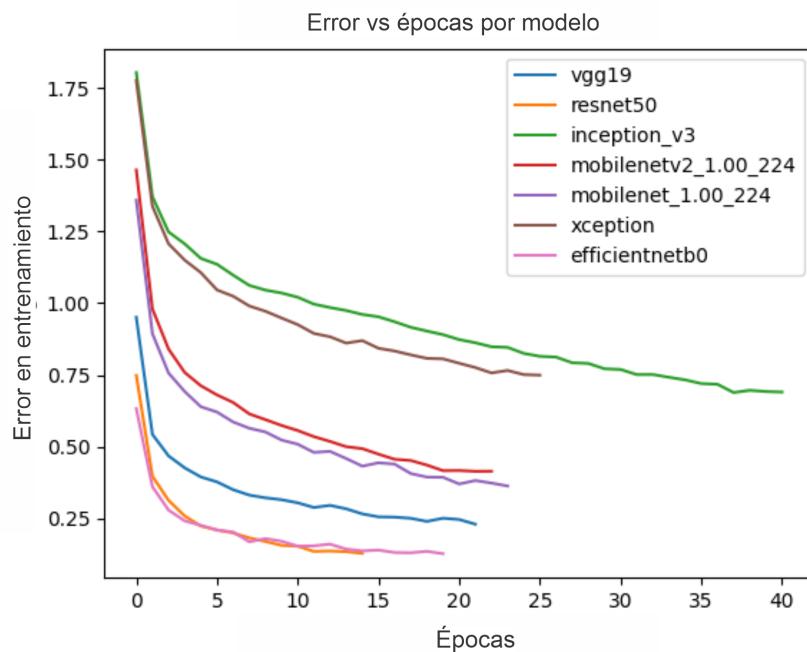


Figura 5.1: Gráfica comparativa de la pérdida entre todos los modelos a estudiar

En esta gráfica se puede ver que los modelos que consiguen mejores descensos en esta métrica son tanto Resnet50, Vgg19 y EfficientnetB0, lo cual era bastante esperado conforme a la teoría, salvo por el Vgg19. Otro detalle interesante que salió a relucir de esta experimentación fue que InceptionV3 y Xception tuvieron una pérdida bastante elevada, contrario a lo que se esperaba en un inicio, lo cual podría ser motivo de análisis en un futuro para saber cuáles fueron los motivos por los cuales no consiguieron aprender del *dataset*. Por último, se puede ver como las dos MobileNet están en el medio de todas las demás redes analizadas, lo cual, es un dato a resaltar considerando lo ligero que son .

También se obtuvieron algunos otros datos que fueron recopilados en la tabla 5.4.

	Precisión en training	Precisión en validation	Precisión en testing	Perdida final en training	Velocidad por batch (8 imágenes) en ms	Número de épocas
Vgg19	0.9193	0.9928	0.9926	0.2299	71	21
Resnet50	0.9596	0.9993	0.9950	0.1282	52	14
Inception V3	0.7362	0.6620	0.6681	0.6901	42	40
MobileNet V2	0.8521	0.9052	0.9104	0.4143	28	31
MobileNet V1	0.8694	0.9288	0.9519	0.3628	23	23
Xception	0.7244	0.7085	0.7178	0.7490	48	25
EfficientNet B0	0.9577	0.9993	1.0000	0.1270	41	19

Cuadro 5.4: Datos recopilados de la prueba de diferentes modelos del estado del arte

Esta tabla también proporciona datos interesantes sobre la experimentación. En primer lugar, Vgg19 tuvo la mayor pérdida entre todos los modelos, pero logró una precisión bastante aceptable en términos de entrenamiento, mientras que su precisión en validación y prueba fue casi perfecta. En segundo lugar, siguiendo con el análisis anterior, se observa que InceptionV3 y Xception obtuvieron resultados acordes a su pérdida en términos de precisión general.

Otro punto a resaltar es el hecho que ambos MobileNet obtienen un resultado bastante similar en términos de precisión en general, pero MobileNetV2 obtiene una pequeña diferencia, la cual la hace mejor que su versión anterior, pero a un costo en términos de velocidad, lo cual también se esperaba conforme a la teoría. Por último, cabe señalar que los tres modelos que obtuvieron la mejor precisión fueron EfficientnetB0, Resnet50 y Vgg19 respectivamente.

A manera de conclusión, se pudo ver que con un *dataset* relativamente sencillo, se puede obtener resultados bastante favorables, que logran casi a su totalidad automatizar este tipo de problemas con modelos ya pre-entrenados que son fácilmente modificables para su uso.

5.3.3. Experimentación #2

Para este segundo análisis, se utilizaron las imágenes obtenidas del *datasets* *The Nature Conservancy Fisheries Monitoring*. Dentro de estas imágenes se podrían encontrar muchos peces de la misma especie por imagen. En ese

sentido la labor del clasificador será identificar si esta imagen contiene ejemplares de un determinado pez en ella. Para este experimento, únicamente se tomaron los *datasets* de entrenamiento y validación, ya que el de testeo no contenía etiquetas, pero será utilizado en futuras experimentaciones. De la misma manera que en la anterior experimentación, primero se procederá a analizar las gráficas de error, tal como se ven en la imagen 5.2. Se puede ver

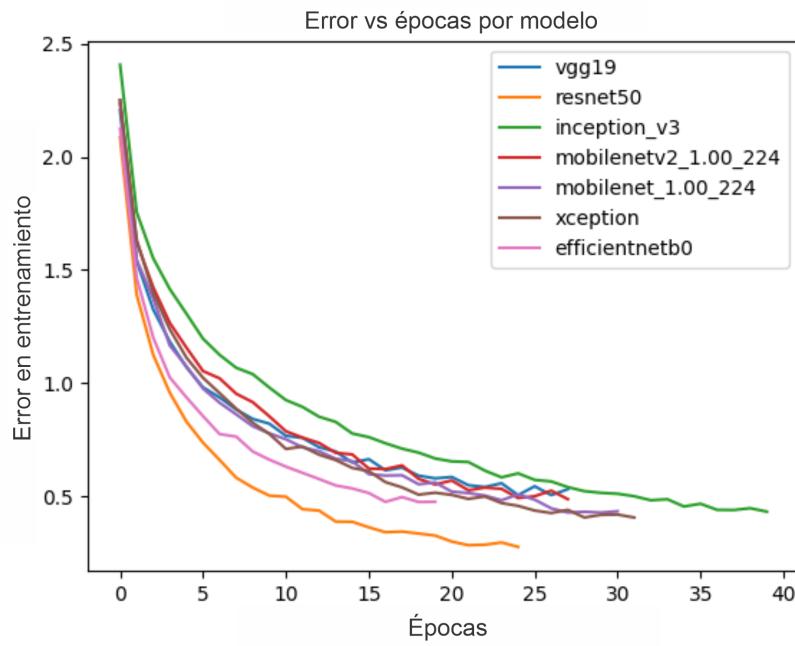


Figura 5.2: Gráfica comparativa de la perdida entre todos los modelos a estudiar

que en esta experimentación se tuvieron rangos de error más elevados que el anterior, esto debido a la complejidad de la tarea. También se puede ver un descenso del error bastante más parejo entre todos los modelos experimentados. Nuevamente aparecen Resnet50 y Efficientnet con los menores resultados y también un menor número de épocas necesitadas para este entrenamiento.

Ahora pasaremos a analizar las métricas obtenidas para estos modelos, los cuales están plasmados en la tabla 5.5.

	Precisión en training	Precisión en validation	Perdida final en training	Velocidad por batch (8 imágenes) en ms	Numero de épocas
Vgg19	0.8173	0.9061	0.5317	68	29
Resnet50	0.9139	0.9219	0.2769	50	24
Inception V3	0.8510	0.8637	0.4324	42	39
MobileNet V2	0.8348	0.8516	0.4875	28	27
MobileNet V1	0.8543	0.8743	0.4340	24	30
Xception	0.8676	0.8267	0.4065	48	31
EfficientNet B0	0.8467	0.9101	0.4759	41	19

Cuadro 5.5: Datos recopilados de la prueba de diferentes modelos del estado del arte

En comparación con la gráfica anterior, se observa un deterioro en todas las métricas obtenidas. Resnet y Efficientnet mostraron una alta precisión en entrenamiento y validación, superando a Vgg19, el cual no obtuvo una buena precisión en entrenamiento y presentó una alta pérdida final, lo cual sugiere que sufrió de *overfitting*. Si consideramos además la relación entre precisión y eficiencia, ambas MobileNet resultan bastante robustas, ya que generan una precisión relativamente alta, manteniendo una velocidad por *batch* razonable. Exceptuando ambas MobileNets, InceptionV3 y EfficiententB0 fueron las que obtuvieron una velocidad por *batch* menor que las demás. Por último cabe resaltar que Resnet50 y Efficientnet fueron las redes que necesitaron un menor número de épocas para detenerse.

Considerando los resultados obtenidos, proponemos el uso de EfficientnetB0 para la creación del *pipeline*, debido a su alta precisión tanto en las pruebas como en el entrenamiento, así como en la validación y prueba. Además, es significativamente más ligero, presenta una curva de aprendizaje más estable a lo largo de las épocas, un menor número de épocas necesitadas para entrenarse y una velocidad superior a todas las demás.

5.4. Experimentación del clasificador

En esta sección se llevaron a cabo 5 experimentos con el objetivo de validar y obtener un modelo entrenado basado en la arquitectura seleccionada en la sección anterior, con el fin de maximizar la precisión al entrenarlo con las imágenes recortadas. Dos de los experimentos se realizaron debido a una distribución inadecuada de los datos entre el conjunto de entrenamiento y el de pruebas. Estos dos experimentos se separaron y se mencionan en la sección de anexos para evitar interferencias con la continuidad de los demás

experimentos.

5.4.1. Experimentación #1 (training/validation)

En el primer experimento se emplearon únicamente las imágenes recortadas del conjunto de entrenamiento para comprobar la capacidad de aprendizaje del modelo. De estas imágenes, se seleccionó una parte para ser utilizada como conjunto de validación. En esta prueba, se llevaron a cabo 10 iteraciones utilizando el algoritmo de K-Fold para la muestra y se obtuvieron los resultados que se presentan en la tabla 5.6

<i>K</i>	<i>training loss</i>	<i>training accuracy</i>	<i>validation loss</i>	<i>validation accuracy</i>
1	0.0115	99.660 %	0.2697	94.43 %
2	0.0011	100.00 %	0.2243	96.06 %
3	0.0106	99.690 %	0.2106	95.13 %
4	0.0186	99.510 %	0.2286	93.97 %
5	0.0069	99.820 %	0.2265	95.36 %
6	0.0060	99.820 %	0.1204	97.22 %
7	0.0079	99.870 %	0.2417	91.88 %
8	0.0147	99.640 %	0.2182	93.50 %
9	0.0154	99.430 %	0.2767	93.27 %
10	0.0327	98.920 %	0.2814	93.26 %

Cuadro 5.6: Perdida y precisión de entrenamiento y validación

Con estos resultados, podemos observar que al ejecutar el mismo modelo varias veces, se obtienen resultados similares, lo que demuestra que no hay *overfitting* y que el modelo no ha sido influenciado por la aleatoriedad en la inicialización de los pesos. Además, al evaluar imágenes de manera individual, se logra aumentar la precisión del modelo en comparación con la clasificación de imágenes completas.

A continuación, se procedió a experimentar con el *dataset* de *testing* para poder comprobar si el modelo generó buenos resultados al predecir las clases de las imágenes recortadas. Por el contrario, se comprobó que pertenecían a distribuciones de datos muy diferentes. En el *dataset* de prueba se tenían imágenes en diferentes entornos, con menor visibilidad y mayor falta de características distintivas de las especies estudiadas, entre otros factores que dificultaban el aprendizaje. Para poder comprobarlo, se realizaron las experimentaciones #2 y #3 (detalladas en el anexo A). Se obtuvo que el modelo predijo a las imágenes complejas como “bonito” (*Albacore Tuna*), ya que en el *dataset* de entrenamiento había una mayor cantidad de estas imágenes, por lo que era estadísticamente más probable acertar de ese modo. Además, estas imágenes eran las que presentaban más defectos dentro del conjunto. En ese sentido se cambió el enfoque de las experimentaciones siguientes.

5.4.2. Experimentación #2 (extracción de datasets desde training)

Considerando lo anteriormente expuesto, se decidió dividir el conjunto de datos de entrenamiento en tres conjuntos diferentes: entrenamiento, validación y prueba, utilizando una proporción del 70 % para entrenamiento, 20 % para validación y 10 % para pruebas. El objetivo de esta nueva división fue comprobar si la suposición previa de que el problema era causado por el uso de conjuntos de datos con diferentes distribuciones era correcta. Los resultados obtenidos se muestran en la tabla 5.7.

	<i>Loss</i>	<i>Balanced accuracy</i>	<i>F1-weighted</i>	<i>Precision</i>	<i>Recall</i>
<i>Train</i>	0.10	96.57 %	96.57 %	96.76 %	96.39 %
<i>Validation</i>	0.22	95.64 %	95.70 %	95.75 %	95.65 %
<i>Testing</i>	0.23	90.28 %	95.29 %	95.39 %	95.14 %

Cuadro 5.7: Estadísticos obtenidos del experimento #2

Se puede observar un comportamiento similar al experimento #1, lo cual nos afirma entonces que realmente era un problema de malas distribuciones de datos en ambos *datasets*. De la misma manera, se revisaron las matrices de confusión de los datos, los cuales se recopilaron en la tabla 5.3

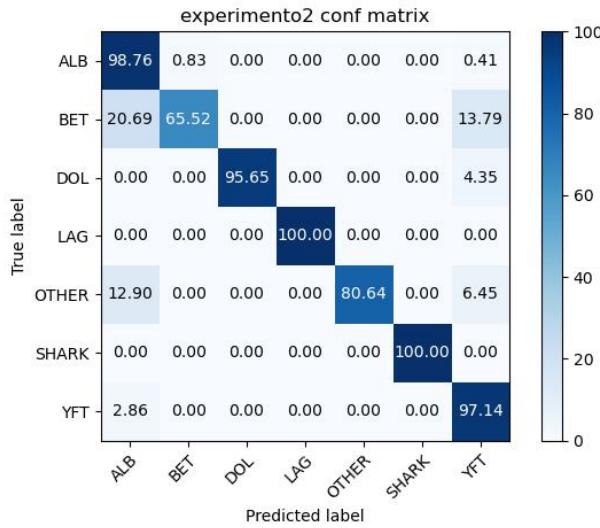
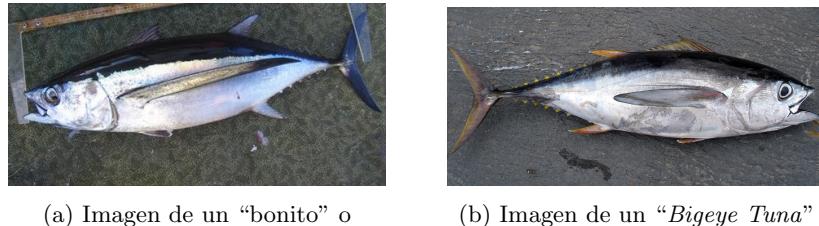


Figura 5.3: Matriz de confusión generada por el experimento #2 en porcentajes

Como se puede apreciar en la matriz de confusión, en general se obtuvo una

buenas precisiones en la mayoría de las clases, excepto en la clase “Bigeye Tuna”, lo cual es comprensible ya que se trata de una especie bastante similar al “Albacore Tuna” y las imágenes no permiten apreciar las características diferenciales con claridad, como se pueden apreciar en las imágenes 5.4.



(a) Imagen de un “bonito” o
“Albacore Tuna”
(b) Imagen de un “Bigeye Tuna”

Figura 5.4: Comparación entre el “Albacore Tuna” o bonito y el “Bigeye Tuna”

Además, se puede observar que no existe *overfitting*, ya que las clases con menor cantidad de imágenes también consiguieron una alta precisión. También se revisaron las gráficas de pérdida y precisión tanto para el conjunto de entrenamiento como para el de validación, las cuales se muestran en las figuras 5.5 y 5.6.

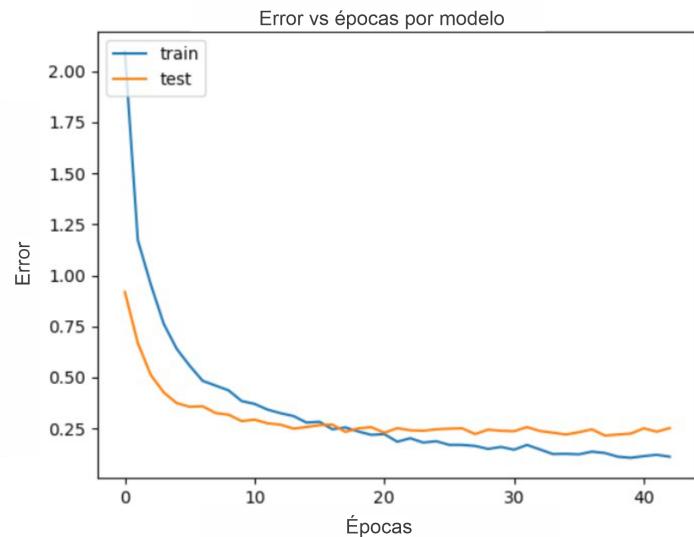


Figura 5.5: Gráfica comparativa de la perdida de entrenamiento y validacion

Las gráficas muestran un aprendizaje constante, lo que refuerza la conclusión de que la red está aprendiendo correctamente, evitando aprender en exceso

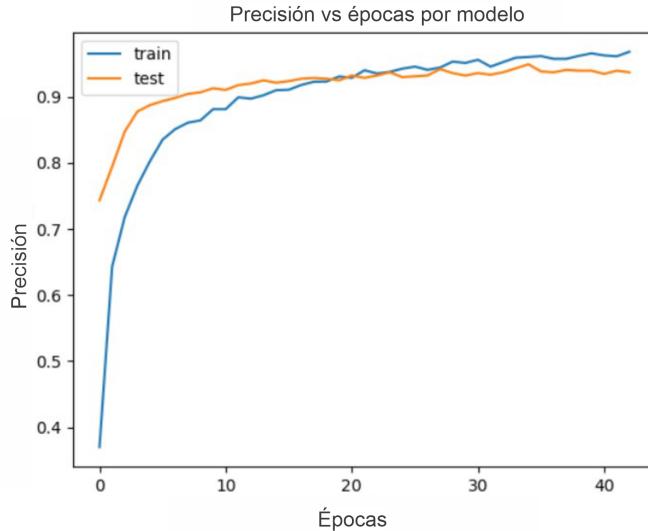


Figura 5.6: Gráfica comparativa de la precisión de entrenamiento y validacion

de ciertas clases e incluso obteniendo resultados positivos con las clases que tienen menor cantidad de imágenes. Además, es importante destacar que, en comparación con los experimentos previos del clasificador, estas nuevas redes están tardando más épocas en aprender de los datos debido a la complejidad de los mismos.

5.4.3. Experimentación #3 (extracción de datasets desde *training+testing*)

Con los resultados del anterior experimento se planteó unificar ambos *datasets* y en base a esa nueva distribución de datos realizar la separación de los *datasets* de *training*, *validation* y *testing*. Dicha separación se realizó con los mismos porcentajes del anterior experimento. Al ejecutar la prueba, se obtuvieron los resultados de la tabla 5.8

	<i>Loss</i>	<i>Balanced Accuracy</i>	<i>F1-weighted</i>	<i>Precision</i>	<i>Recall</i>
<i>Train</i>	0.2061	93.04 %	93.12 %	93.93 %	92.34 %
<i>Validation</i>	0.4659	87.83 %	88.11 %	88.36 %	87.87 %
<i>Testing</i>	0.4564	87.73 %	88.47 %	90.06 %	88.03 %

Cuadro 5.8: Estadísticos obtenidos del experimento #3

La unión de ambos *datasets* ha arrojado resultados positivos, lo que confirma que sí había una gran diferencia en la distribución de los datos, lo

que causaba que el modelo no genere una buena predicción al momento de realizar el *testing*. Además, las gráficas de error y precisión (imágenes 5.7 y 5.8) muestran el descenso en el error y el aumento en la precisión esperados.

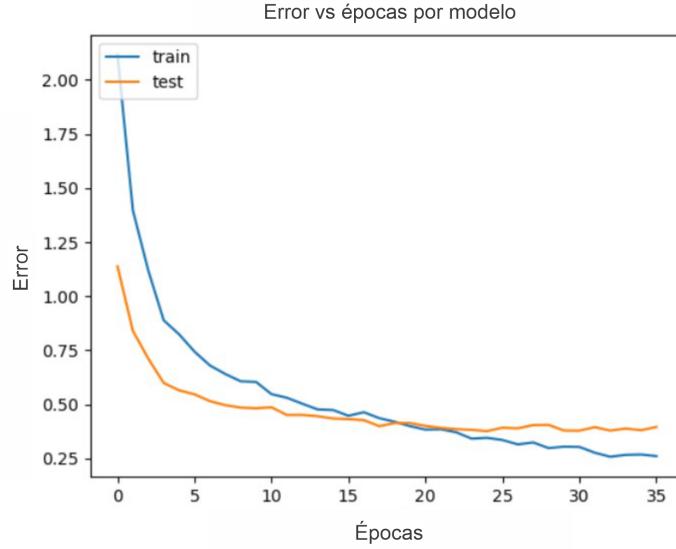


Figura 5.7: Gráfica comparativa de la perdida de entrenamiento y validacion del experimento #3

Por último, se realizó la matriz de confusión de las clases clasificadas que se pueden ver en la tabla 5.9

Con base en los resultados obtenidos, se puede afirmar que este modelo está preparado para ser utilizado en el *pipeline* correspondiente. Es importante destacar que el entrenamiento de este modelo tardó 36 épocas, con un promedio de 10 segundos por época, lo que significa que el tiempo total de entrenamiento fue de 6 minutos, un tiempo relativamente corto. Además, el tiempo de procesamiento por *batch* (32 imágenes) es de 85 ms, lo que indica que en el futuro se podría ampliar la base de datos con más imágenes de especies adicionales, para proporcionar una mayor especificidad en la detección de objetos o animales por medio del detector. En comparación, el proceso de entrenamiento para una red más compleja, que requiera una base de datos etiquetada, sería significativamente más largo y complicado.

5.5. Experimentación del *pipeline*

Debido a que el *pipeline* consta de dos pasos para la detección y clasificación de las imágenes, realizar una experimentación típica se hace inviable. En ese sentido, se simuló un *benchmark* utilizando el flujo descrito por la imagen 5.10

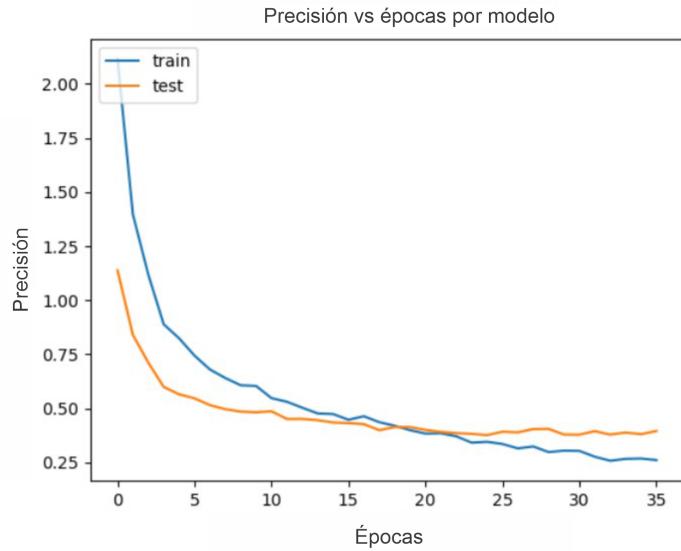


Figura 5.8: Gráfica comparativa de la precisión de entrenamiento y validacion del experimento #3

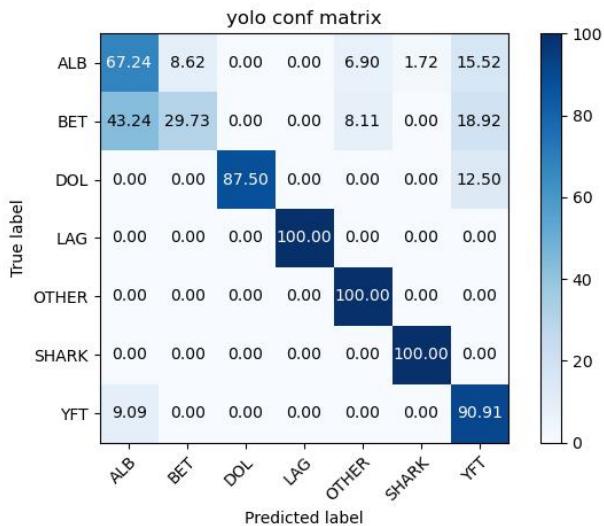


Figura 5.9: Matriz de confusión generada por el experimento #3 en porcentajes

5.5.1. Prueba del clasificador

Una vez obtenidos las detecciones de los peces, quitado los errores del *dataset* y etiquetar los peces que se tenían dentro del *dataset* de prueba, se generaron

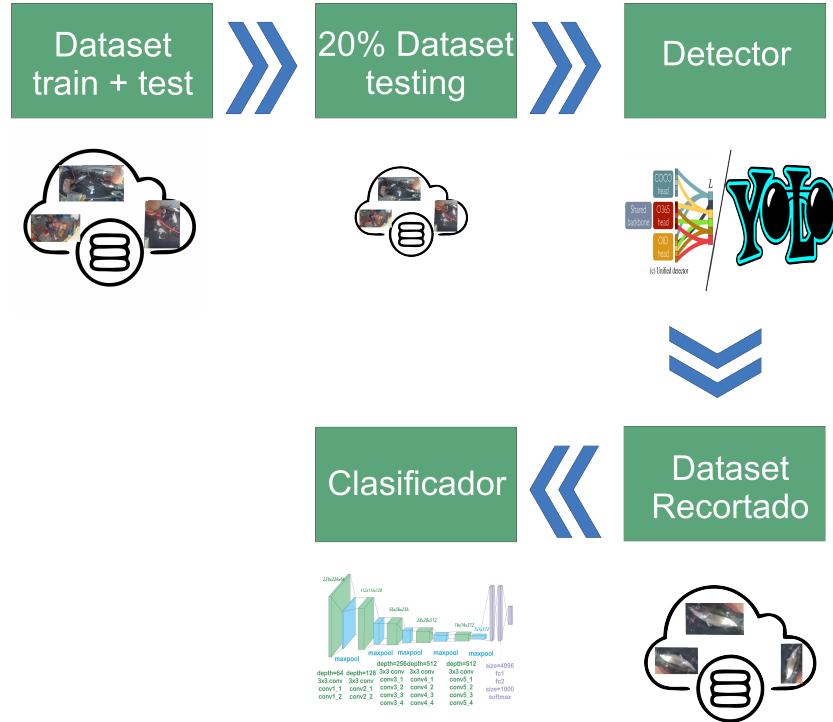


Figura 5.10: Flujo para la experimentación del *pipeline*

tres nuevos *datasets* de imágenes de peces detectados. Consiguentemente, se procedió a utilizar el modelo ya entrenado en la sección 5.4.3. A continuación se presentarán los resultados obtenidos para cada uno de los modelos, los cuales se recopilaron en la tabla 5.9.

	<i>Balanced Accuracy</i>	<i>F1-weighted</i>	<i>Precision weighted</i>	<i>Recall weighted</i>
<i>Predicted Yolov5 pretrained(40 % threshold)</i>	72.99 %	68.35 %	76.08 %	66.89 %
<i>Predicted UniDet pretrained (no threshold)</i>	84.47 %	84.67 %	86.50 %	84.03 %
<i>Predicted Yolov5 trained (freezed backbone 95 % threshold)</i>	91.47 %	91.90 %	91.99 %	91.93 %

Cuadro 5.9: Estadisticos generados por los 3 diferentes modelos

Los resultados obtenidos muestran que el *pipeline* funciona relativamente bien con todos los diferentes detectores, generando un *balanced accuracy*

bastante elevado. Por otra parte, se puede ver que los *pipelines* que utilizaban los detectores preentrenados obtuvieron un resultado similar, aunque peor que el que usaba el detector entrenado. Esto es debido a que los *bounding boxes* generados por el modelo entrenado coincidían con las imágenes recortadas con las que se entrenó el clasificador, mientras que UniDet generaba resultados aproximados. Por otra parte el yolov5 preentrenado los generaba de manera muy justa, haciendo necesario que sean preprocessadas antes de ser ingresados al clasificador, por lo que también eran aproximados. Además se obtuvieron las matrices de confusión(tablas 5.11, 5.12 y 5.13) de cada uno de los experimentos, los cuales se muestran a continuación:

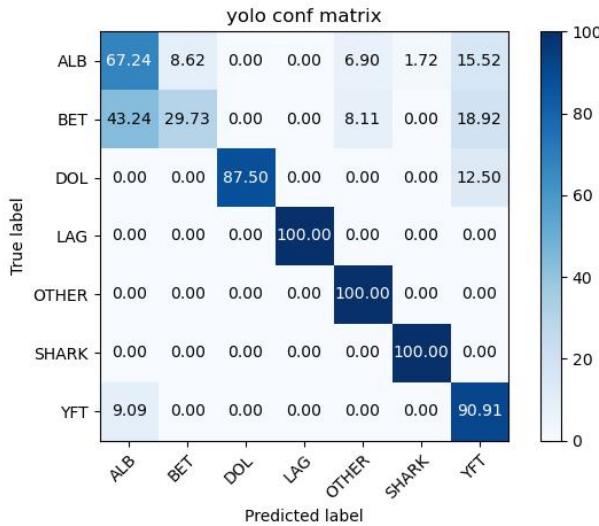


Figura 5.11: Matriz de confusión para el testeo del *pipeline con yolov5 preentrenado*

En general, se comprobó que el clasificador cumplió su función correctamente de clasificar las imágenes que le fueron proporcionadas por el detector. Aún así, la precisión de este radica en lo bien que fueron detectados los peces dentro de las imágenes. En ese sentido podemos ver que el aplicar este *pipeline* generaría una pérdida en la precisión a comparación de entrenar un modelo de un 10 % a 28 % (que podría variar dependiendo del detector usado y del tratamiento de los *bounding boxes*) en imágenes complejas como las que se tienen(sería conveniente realizar pruebas en imágenes menos complejas).

5.6. Evaluación

Finalmente, se comparó los resultados en términos de precisión final y el error generado de los modelos, tomando en cuenta la siguiente fórmula:

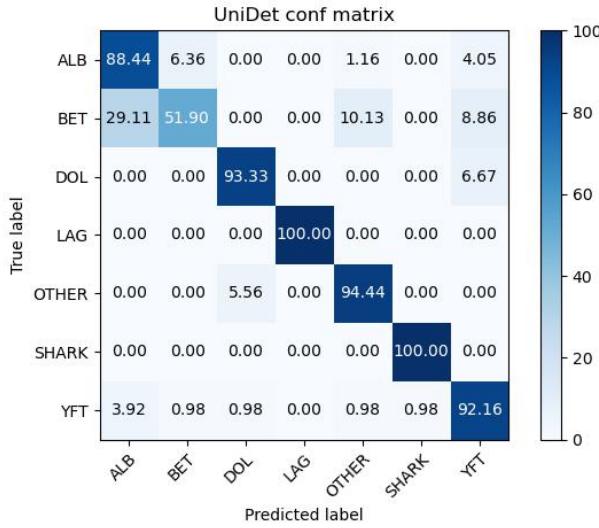


Figura 5.12: Matriz de confusión para el testeo del *pipeline con UniDet preentrenado*

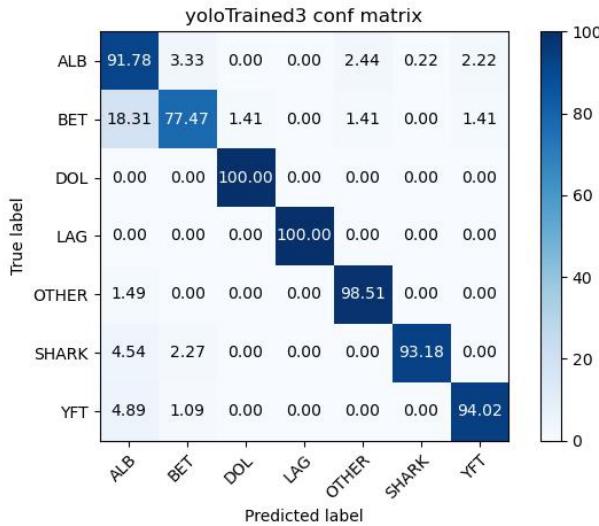


Figura 5.13: Matriz de confusión para el testeo del *pipeline con yolov5 entrenado*

$$\text{precision_final} = \frac{\text{peces_detectados} * \text{accuracy_claseficator}}{\text{imagenes_reales}}$$

Siguiendo esta métrica, y el porcentaje de error en la detección, se obtuvo

la tabla de resultados final 5.10:

	<i>Accuracy parcial</i>	<i>Accuracy del clasificador</i>	<i>Accuracy final</i>	<i>Porcentaje de error del total</i>	<i>Tiempo de entrenamiento</i>
<i>Predicted Yolov5 pretrained (40 % threshold)</i>	14.13 %	72.99 %	10.31 %	69.07 %	9 minutos (CNN)
<i>Predicted UniDet pretrained (no threshold)</i>	40.15 %	84.47 %	33.91 %	22.54 %	9 minutos (CNN)
<i>Predicted Yolov5 trained (backbone freezed 95 % threshold)</i>	79.45 %	91.47 %	72.67 %	32.36 %	9 minutos(CNN) + 80 minutos(yolov5)

Cuadro 5.10: Accuracy final y porcentaje de errores detectados de los 3 modelos

Considerando todos los experimentos realizados, podemos concluir que el *pipeline* propuesto puede ser aplicable inclusive para casos complejos de manera que este no disminuye mucho el rendimiento de la precisión final ni aumenta mucho el tiempo de procesamiento. Además evita necesitar la obtención de un *dataset* etiquetado con *bounding boxes* para el entrenamiento, reducir el tiempo de entrenamiento y aumentar la escalabilidad de la especificación de especies detectadas. Cabe resaltar como punto importante que el desempeño de este *pipeline* recae principalmente en poder encontrar un detector especializado o debidamente entrenado y robusto para poder detectar las diferentes clases y en el tratamiento de los *bounding boxes* obtenidos para el paso por el clasificador.

Capítulo 6

Conclusiones y Trabajos Futuros

6.1. Conclusiones

En el presente trabajo se desarrolló un *pipeline* para la detección y clasificación de imágenes de peces peruanos en un entorno real, evitando el uso de un *dataset* etiquetado para la detección en imágenes. Para ello, se recolectó un *dataset* a través de la plataforma Kaggle, conteniendo varios peces dentro de cada imagen, el cual fue recortado y clasificado por un experto. Finalmente se comprobó que utilizar un modelo ligero pero al mismo tiempo robusto como lo es EfficientNetB0, resulta ser la mejor opción para poder implementarlo al evitar una perdida significativa de la precisión final, un bajo tiempo de entrenamiento, una mayor escalabilidad para poder realizar la tarea de detección y clasificación de imágenes.

6.2. Trabajos futuros

Algunas de las limitaciones que se tiene con este modelo consiste en que el *pipeline* depende casi completamente del detector escogido, de tal manera que se necesita poder obtener uno bastante robusto y entrenado de manera general o incluso mejor si está especialmente entrenado para detectar las clases que se están estudiando. De esta manera, se generaría buenas entradas para el clasificador, reduciendo el efecto que este tendría en el *pipeline* final. En ese sentido, sería bueno poder realizar más experimentos con diferentes arquitecturas y modelos.

De la misma manera, actualmente existen ciertos *datasets* que son ampliamente utilizados para la investigación y creación de aplicaciones de DL, mientras que otros presentan pocas implementaciones, lo que hace más

complicado la obtención de modelos preentrenados con las clases genéricas requeridas (por ejemplo, peces). En ese sentido, otro trabajo futuro consistiría en preentrenar estos modelos con solo las clases genéricas a estudiar o incrementar tal vez una capa de atención a los modelos para poder incrementar la *accuracy* del detector y por ende la del *pipeline* completo.

Por último, este trabajo se realizó con un *dataset* de peces peruanos (en su mayoría) pero consistía de un bajo número de imágenes y consistían de una distribución de datos muy complejas. En ese sentido, el error y la precisión obtenido también contemplaban *bias* debido a estos factores. Es por ello que se recomendaría replicar las experimentaciones realizadas en el presente documento pero con un *dataset* que incluya un número de especies e imágenes mayor para poder calcular como es que esto influye en la precisión, el costo de entrenamiento y si se necesita mejorar o incrementar algunas capas del modelo.

Referencias

- Alsmadi, M. K., y Almarashdeh, I. (2022, may). A survey on fish classification techniques. *Journal of King Saud University - Computer and Information Sciences*, 34(5), 1625–1638. doi: 10.1016/J.JKSUCI.2020.07.005
- Cai, Z., y Vasconcelos, N. (2017, dec). Cascade R-CNN: Delving into High Quality Object Detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 6154–6162. Descargado de <https://arxiv.org/abs/1712.00726v1> doi: 10.1109/CVPR.2018.00644
- Calvo, I. (2022, may). A survey on fish classification techniques. *Journal of King Saud University - Computer and Information Sciences*, 34(5), 1625–1638. doi: 67976/1
- Chen, G., Sun, P., y Shang, Y. (2017). Automatic fish classification system using deep learning. En *2017 ieee 29th international conference on tools with artificial intelligence (ictai)* (p. 24-29). doi: 10.1109/ICTAI.2017.00016
- Cui, S., Zhou, Y., Wang, Y., y Zhai, L. (2020). Fish detection using deep learning. *Applied Computational Intelligence and Soft Computing*, 2020. Descargado de <https://dl.acm.org/doi/pdf/10.1145/3419635.3419643> doi: 10.1155/2020/3738108
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. En *2009 ieee conference on computer vision and pattern recognition* (p. 248-255). doi: 10.1109/CVPR.2009.5206848
- Diego Calvo. (2019a). *red-neuronal-convolucional*. Descargado de <https://www.diegocalvo.es/red-neuronal-convolucional/> ([Online; accessed December 05, 2021])
- Diego Calvo. (2019b). *red-neuronal-convolucionar-arquitectura*. Descargado de <https://www.diegocalvo.es/red-neuronal-convolucionar/red-neuronal-convolucionar-arquitectura/> ([Online; accessed December 05, 2021])
- He, K., Zhang, X., Ren, S., y Sun, J. (2015). *Deep residual learning for image recognition*.
- IchiPro. (2020). *4 modelos de cnn previamente entrenados para usar en visión artificial con aprendizaje por transferencia*. Descargado de <https://ichi.pro/es/4-modelos-de-cnn-previamente-entrenados-para>

- usar-en-vision-artificial-con-aprendizaje-por-transferencia-9370731228668 ([Online; accessed December 05, 2021])
- J, Z. (2015). *Writing for computer science*. Springer.
- Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., ... Jain, M. (2022, noviembre). *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*. Zenodo. Descargado de <https://doi.org/10.5281/zenodo.7347926> doi: 10.5281/zenodo.7347926
- Lan, X., Bai, J., Li, M., y Li, J. (2020). Fish image classification using deep convolutional neural network. En *Proceedings of the 2020 international conference on computers, information processing and advanced education* (p. 18–22). New York, NY, USA: Association for Computing Machinery. Descargado de <https://doi.org/10.1145/3419635.3419643> doi: 10.1145/3419635.3419643
- M., S. J., y B., F. C. (2012). *Academic writing for graduate students*. The university of Michigan Press.
- Magiquo. (2019). *Atomicredes neuronales o imitar al cerebro humano?* Descargado de <https://magiquo.com/wp-content/uploads/2019/11/neurona.png> ([Online; accessed December 05, 2021])
- Manandhar, N., y Burris, J. W. (2019). An application of image classification to saltwater fish identification in louisiana fisheries. En *Proceedings of the 2019 3rd international conference on information system and data mining* (p. 129–132). New York, NY, USA: Association for Computing Machinery. Descargado de <https://doi.org/10.1145/3325917.3325934> doi: 10.1145/3325917.3325934
- Mejía, G., Rodrigo Rosales. (2020). Sistema de detección y clasificación de peces utilizando visión computacional. En *Innovando la educación en tecnología* (p. 127-141). Lima, LIM, Peru: Actas del II Congreso Internacional de Ingeniería de Sistemas. Descargado de <https://hdl.handle.net/20.500.12724/11174> doi: 20.500.12724/11174
- Muhamad Yani, Budhi Irawan, Casi Setianingsih. (2019). *Application of transfer learning using convolutional neural network method for early detection of terry's nail*. Descargado de https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451 ([Online; accessed December 05, 2021])
- Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2015, jun). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-December*, 779–788. Descargado de <https://arxiv.org/abs/1506.02640v5> doi: 10.48550/arxiv.1506.02640
- Simonyan, K., y Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition*. Descargado de <http://www.robots.ox.ac.uk/>
- Szegedy, C., Liu, W., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., ... Rabinovich, A. (2014). *Going deeper with convolutions*.
- Tan, M., y Le, Q. V. (2020). *Efficientnet: Rethinking model scaling for*

- convolutional neural networks.* Descargado de <https://arxiv.org/pdf/1905.11946.pdf>
- Tashin Ahmed, N. H. S. (2020). *Classification and understanding of cloud structures via satellite images with efficientunet.* Descargado de https://www.researchgate.net/figure/Architecture-of-EfficientNet-B0-with-MBConv-as-Basic-building-blocks_fig4_344410350 ([Online; accessed December 05, 2021])
- Wenjin Taoa, Md Al-Aminb, Haodong Chena, Ming C. Leua, Zhaozheng Yinc, Ruwen Qinb. (2020). *Real-time assembly operation recognition with fog computing and transfer learning for human-centered intelligent manufacturing.* Descargado de https://www.researchgate.net/figure/The-architecture-of-our-transfer-learning-model_fig4_342400905 ([Online; accessed December 05, 2021])
- Yalcin, O. (2020). *Pre-trained model performances.csv.* Descargado de <https://gist.github.com/ogyalcin/052f2df49b3288e62086aa0e5fd25fc>
- Yani, M., Irawan, B., y Setimingsih, C. (2019, 5). Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, 1201. Descargado de https://www.researchgate.net/publication/333593451_Application_of_Transfer_Learning_Using_Convolutional_Neural_Network_Method_for_Early_Detection_of_Terry's_Nail doi: 10.1088/1742-6596/1201/1/012052
- Zhou, V., Xingyi; Koltun, y Krahenbuhl, P. (2022, Apr). *Simple Multi-dataset Detection.* Descargado de <https://arxiv.org/pdf/2102.13086.pdf> doi: 2102.13086

Apéndice A

Experimentación con distribución de datos diferentes

Los siguientes experimentos fueron realizados para poder comprobar que la distribución de datos tanto del *dataset* de entrenamiento como el de *testing* eran diferentes, generando un *overfitting* accidental en la experimentación general.

A.1. Experimentación A (training/ validation/ testing)

Una vez de terminado la creación de los *bounding boxes* de parte del experto, se procedió a realizar la comprobación del modelo en base a los datos de *testing*. Los estadísticos resultantes del *testing* fueron los mostrados en la tabla A.1.

<i>Loss</i>	<i>Balanced Accuracy</i>	<i>F1-weighted</i>	<i>Precision</i>	<i>Recall</i>
2.7014	46.50 %	41.46 %	50.28 %	49.54 %

Cuadro A.1: Estadísticos obtenidos del experimento #A

Los estadísticos obtenidos son bastante malos en comparación con los datos de entrenamiento. Para confirmar las sospechas de un posible *overfitting*, se revisaron las gráficas de pérdida a través de las épocas, que se pueden ver en la imagen A.1.

Una vez más, la gráfica muestra un posible caso de *overfitting* de los datos y, aún peor, la gráfica de error indica un aumento en lugar de una disminución a lo largo de las épocas. Para evaluar de manera más detallada los resultados, se revisó la matriz de confusión generada, la cual se muestra en la tabla A.2

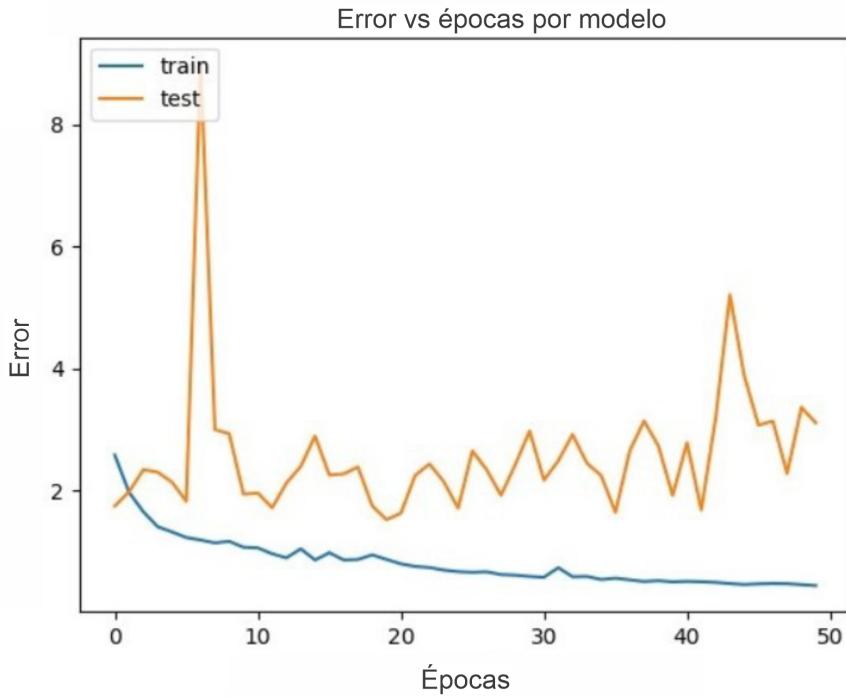


Figura A.1: Gráfica comparativa de la perdida de entrenamiento y *testing*

	Albacore Tuna	Bigeye Tuna	Dolphin Fish	Moonfish (LAG)	Shark	Yellowfin Tuna	Other
<i>Albacore Tuna</i>	94.26 %	00.78 %	00.00 %	00.00 %	00.78 %	01.57 %	02.61 %
<i>Bigeye Tuna</i>	88.23 %	03.07 %	00.00 %	00.00 %	00.51 %	00.77 %	07.42 %
<i>Dolphin Fish</i>	35.00 %	00.00 %	50.00 %	00.00 %	00.00 %	05.00 %	10.00 %
<i>Moonfish (LAG)</i>	25.42 %	00.00 %	03.39 %	57.63 %	06.78 %	00.00 %	06.78 %
<i>Shark</i>	50.38 %	00.76 %	00.00 %	00.00 %	38.93 %	04.58 %	05.34 %
<i>Yellowfin Tuna</i>	33.33 %	09.52 %	00.00 %	00.00 %	00.00 %	38.09 %	19.05 %
<i>Other</i>	23.78 %	01.22 %	01.22 %	00.00 %	00.00 %	00.00 %	73.78 %

Cuadro A.2: Matriz de confusión generada por las imágenes de *testing* en porcentajes

En la tabla se puede observar que la red tiende a predecir que los ejemplares pertenecen a la primera especie, lo que sugiere que está sobre aprendiendo de esta clase, posiblemente debido a la mayor cantidad de datos disponibles. A

pesar de que esta hipótesis podría parecer plausible, no es la correcta, como se verá en las siguientes secciones.

A.2. Experimentación #3 (training/validation/ testing con *unfreeze*)

Para intentar reducir el posible *overfitting*, se decidió quitar el *freeze* de los dos últimos bloques de capas convolucionales del modelo pre-entrenado, permitiendo así que el modelo aprenda más patrones dentro de las imágenes. Después de esta modificación, se obtuvieron los resultados que se muestran en la tabla A.3.

	<i>Loss</i>	<i>Balanced Accuracy</i>	<i>F1-weighted</i>	<i>Precision</i>	<i>Recall</i>
<i>Bloque 7 unfreezed</i>	4.1616	51.71 %	42.23 %	51.17 %	50.92 %
<i>Bloques 6 y 7 unfreezed</i>	3.1206	19.56 %	28.97 %	40.58 %	33.90 %

Cuadro A.3: Estadísticos obtenidos del experimento #3

De la misma manera, las gráficas de error y accuracy, junto con las matrices de confusión resultaron similares al anterior experimento, por lo que surgió la duda acerca de si tal vez existía una incongruencia con la base de datos.