Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Segmentación de imágenes de foto-trampeo mediante técnicas de aprendizaje profundo

Alumno: Sergio Perea de la Casa

Tutor: Francisco Charte Ojeda

María J. del Jesus Díaz

Dpto: Informática

UNIVERSIDAD DE JAÉN

D./Dª Francisco Charte Ojeda y D./Dª María J. del Jesus Díaz, tutor(es) del Trabajo Fin de Grado titulado: **Segmentación de imágenes de foto-trampeo mediante técnicas de aprendizaje profundo**, que presenta Sergio Perea de la Casa, autoriza(n) su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, septiembre de 2023

El estudiante Los tutores

Sergio Perea de la Casa

Francisco Charte Ojeda

María J. del Jesus Díaz

Agradecimientos

La importancia que supone este documento va más allá de lo que imaginaba en mi primer año como universitario. Durante mi formación han ocurrido sucesos que quedarán marcados dentro de mi corazón, momentos que jamás olvidaré.

Ese primer día cuando conocí a mis compañeros y acabé formando una familia. Amigos con los que he podido compartir risas, tensión previa a una nota final e incluso llantos en nuestros momentos más afectivos.

Esa supuesta semana que teníamos que pasar encerrados para evitar contagiarnos, donde debo dar gracias por conseguir reunirme con toda mi familia y disfrutar las tardes juntos.

Esa estancia temporal a 10 000 kilómetros de casa en un país como Japón, donde aprender nunca fue solo un concepto académico. Doy gracias por la oportunidad de comenzar de cero, vivir sus costumbres, observar su forma de ser y, lo más importante, crear una nueva familia.

Además, no podía olvidarme de cada uno de los profesores que ha confiado en mí, siempre recordaré el buen trato que me dieron. Espero que, si algún día leen esto, consigan darse por aludidos.

Por último, me gustaría dedicar el final a la Luna. Como me dijo mi abuela, si miramos hacia la Luna a la misma hora, da igual si no estamos juntos, ya que siempre estaremos conectados.

Tabla de contenidos

Lista de figuras

Lista de tablas

Capítulo 1

INTRODUCCIÓN

1.1. Planteamiento del problema

En la actualidad, gracias a la evolución tecnológica, tenemos una infinidad de posibilidades de monitorizar todo aquello que el humano no puede supervisar por diversos motivos. El número limitado de personal o el tiempo que supone una monitorización continua nos abre la posibilidad de desarrollar estas tareas humanas mediante el análisis de imágenes.

Una cámara de foto-trampeo automática [1] nos proporciona una diversidad de usos como la obtención de información para contribuir a la conservación de especies (rango de posicionamiento y tamaño de la población), estudio del comportamiento de especies o la identificación de especies diurnas, nocturnas y tímidas no observables directamente. A continuación se indican algunos aspectos que caracterizan a este tipo de dispositivos:

1. La **velocidad de disparo**, que debe ser suficiente para captar la especie en movimiento.
2. La **zona de detección**, indicando el área cubierta por el haz infrarrojo. Es muy importante para indicar la tasa de detección y el número de fotografías a tomar.
3. El **campo de visión**, que cubre el área de la lente de la cámara y será lo que aparezca en las fotografías.
4. **Tiempo de recuperación**, indicando el tiempo transcurrido entre la foto tomada y la siguiente.

5. La **capacidad para obtener imágenes nocturnas**, las cuales se centran en los métodos de flash incandescente y luz infrarroja.

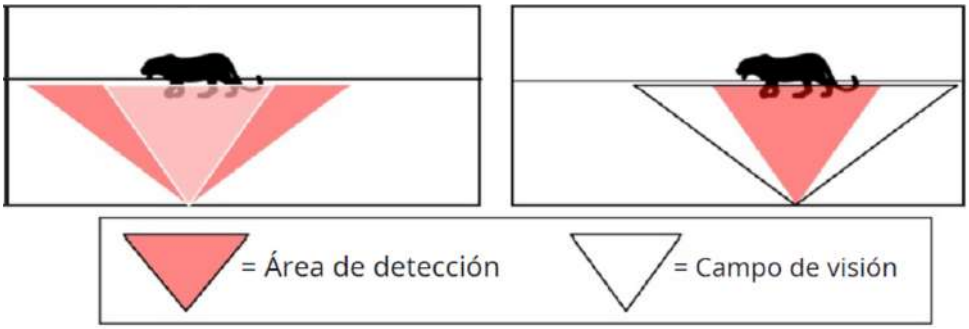


Figura 1.1: Zona de detección y campo de visión [1].

Es muy importante centrarse en el alcance de la **zona de detección** y del **campo de visión**. Por ejemplo, como aparece en la figura 1.1, supongamos que la zona de detección (infrarrojos que detectan el movimiento) es de mayor amplitud que el campo de visión (encargado de fotografiar). En este caso, se obtiene una mayor precisión a la hora de captar animales de naturaleza veloz. El problema radica cuando dicho animal no aparece en el campo de visión; es decir, en la zona a fotografiar. Es por ello que se obtiene una cantidad notable de **imágenes vacías** donde no aparece ningún animal.

El tratamiento de imágenes vacías ya ha sido estudiado y aplicado [23]. Por ello, en este TFG nos centramos en aplicar las técnicas de segmentación de imágenes de foto-trampeo mediante técnicas de aprendizaje profundo sobre imágenes en las que aparecen animales.

La segmentación de animales en una imagen foto-trampeo tiene unos beneficios y aplicaciones prácticas como la mejora de precisión del análisis de imágenes, la facilitación de identificación de especies y la mejora en el seguimiento de especies.

1.2. Objetivo

El propósito del documento será realizar un estudio con métodos de segmentación de imágenes mediante técnicas de aprendizaje profundo y analizar sus resultados con otros procedimientos basados en la segmentación. Por ello, en el capítulo 3 se explica en mayor medida cada uno de los objetivos específicos para alcanzar nuestra finalidad.

1.3. Estructura del proyecto

En esta sección se explica brevemente el esquema del TFG, el cual nos da una panorámica general sobre el contenido del documento.

Introducción

Introducimos el problema a abordar, dando una explicación a preguntas como por qué y cómo será resuelto. Además, se incluye el glosario de términos para facilitar al lector la comprensión de los acrónimos utilizados en el documento.

Antecedentes

En este capítulo se estudia los fundamentos de las técnicas a valorar, empezando por conceptos como la segmentación de imágenes y los primeros modelos planteados para su resolución hasta las técnicas de aprendizaje profundo.

Objetivos

Marcado por el objetivo general del que parte el TFG, este capítulo explica los objetivos específicos a cumplimentar para el desarrollo satisfactorio del proyecto.

Materiales y Métodos

En este capítulo se proporciona información sobre el conjunto de datos que se utilizará, cómo debe ser tratado previamente, herramientas y librerías de uso para el entrenamiento del modelo escogido, entre otros.

Resultados

Tras la explicación detallada del modelo a usar, en este capítulo se muestran los resultados obtenidos una vez aplicadas la métricas para la segmentación de imágenes del conjunto de imágenes a evaluar.

Conclusiones

En este capítulo se analizan los resultados obtenidos dando una explicación completa sobre el estudio realizado a lo largo del TFG, dificultades al respecto y trabajos futuros, entre otros.

1.4. Glosario de términos

En esta sección se muestran los acrónimos usados a lo largo del proyecto. En él se incluye el acrónimo, el significado y su traducción en español en caso de ser necesario. Debido a la gran variedad de términos, se ha decidido dividir el glosario en dos tablas: la tabla 1.1 (Desde la letra `a` hasta la `l` inclusive) y la tabla 1.2 (Desde la letra `m` hasta la `z` inclusive).

Acrónimo	Descripción (Inglés)	Descripción (Español)
AE	Autoencoder	Autocodificador
AAE	Adversarial Autoencoder	Autocodificador adversarial
BBDD	Base de Datos	-
CAE	Convolutional Autoencoder	Autocodificador convolucional
CNN	Convolutional Neural Network	Red neuronal convolucional
COCO	Common Objects in Context	Conjunto de datos estandar en tareas de VC
CRF	Conditional Random Field	Campo aleatorio condicional
CSV	Comma-Separated Values	Valores separados por comas
DCNN	Deep Convolutional Neuronal Network	Redes neuronales convolucionales profundas
DL	Deep Learning	Aprendizaje profundo
FCNN	Fully Convolutional Neural Netork	-
FN	False Negative	Falso negativo
FP	False Positive	Falso positivo
GELU	Gaussian Error Linear Units	-
IoU	Intersection over Union	Intersección sobre unión
JSON	JavaScript Object Notation	Notación de objetos de JavaScript
KDD	Knowledge Discovery in Databases	Descubrimiento de conocimiento en BBDD
L-ReLU	Leaky Rectified Linear Unit	-

Tabla. 1.1: Primera tabla de acrónimos usados en el proyecto.

Acrónimo	Descripción (Inglés)	Descripción (Español)
MAE	Mean Absolute Error	Error absoluto medio
MLP	Multi Layer Perceptron	Perceptrón multicapa
ML	Machine Learning	Aprendizaje automático
MSE	Mean Squared Error	Error cuadrático medio
NMS	Non-Maximum Suppression	Supresión de no máximo
P-ReLU	Parametric Rectified Linear Unit	-
PSPNet	Pyramid Scene Parsing Network	Red de segmentación de escenas en pirámide
RCNN	Recurrent Convolutional Neural Network	Red neuronal convolucional recurrente
RNN	Recurrent Neural Network	Red neuronal recurrente
ReLU	Rectified Linear Unit	-
RoI	Region of Interest	Región de interés
SELU	Scaled Exponential Linear Unit	-
SETR	Segmentation Transformer	-
SDG	Stochastic Gradient Descent	Descenso de gradiente estocástico
SSIM	Structural Similarity Index	Índice de similitud estructural
RPN	Region Proposal Network	Red de Proposición de Regiones
SVM	Support Vector Machine	Máquina de soporte de vectores
TFG	Trabajo Fin de Grado	-
VC	Visión por Computador	-
ViT	Vision Transformer	-
TN	True Negative	Verdadero negativo
TanH	Tangente Hiperbólica	-
TP	True Positive	Verdadero positivo

Tabla. 1.2: Segunda tabla de acrónimos usados en el proyecto.

Capítulo 2

ANTECEDENTES

En este capítulo se muestra la investigación llevada a cabo para que, a posteriori, se desarrolle la segmentación de imágenes de foto-trampeo con el conocimiento necesario para ello.

En el desarrollo del aprendizaje habrá un análisis de los antecedentes, el cual muestra cómo se ha resuelto este problema con diferentes técnicas. Estas no tienen por qué ser de aprendizaje profundo, por ello vamos a centrar nuestro análisis al estudio en este conjunto de técnicas teniendo en cuenta qué se ha estudiado con anterioridad.

Se explican a continuación conceptos básicos de forma general y conceptos de las técnicas a desarrollar, indagando en ellas con mayor profundidad.

2.1. Segmentación de imágenes

En esta sección se exploran y analizan los diferentes enfoques y métodos utilizados en la segmentación de imágenes, abordando técnicas clásicas basadas en propiedades visuales y características de los píxeles, así como enfoques más avanzados basados en el aprendizaje automático.

2.1.1. Introducción

Es un reto generalizar el concepto de segmentación. Este concepto puede ser tratado de muchas formas, dependiendo del ámbito en el que queramos desarrollar su utilidad. Para nuestro caso, nos centramos en la segmentación en imágenes.

La segmentación, en términos orientados a nuestra vida cotidiana, se define como el proceso de dividir la población en grupos definidos por sus características, necesidades y preferencias de consumo. Se puede también usar como técnica para obtener ventaja competitiva en el mercado, siendo la segmentación desarrollada en el producto, marca, tipo de cliente, etc.

Por otro lado, si nos orientamos hacia nuestra área, la Informática, la segmentación se considera como el proceso de dividir un programa informático o un fragmento de datos en segmentos o secciones más pequeños y manejables. Este proceso nos ayuda a mejorar el rendimiento o la escalabilidad de un programa, facilitar su comprensión o depuración, o permitir que diferentes partes de un programa sean trabajadas por diferentes equipos o individuos. Para nuestro interés, buscamos el enfoque de la segmentación de imágenes.

La **segmentación de imágenes** [24] es el proceso de dividir una imagen digital en múltiples segmentos de imagen; es decir, si una imagen se compone de una serie de píxeles, la segmentación consiste en crear subconjuntos de píxeles sobre dicha imagen.

El objetivo de la segmentación es simplificar la representación de una imagen para que facilite el análisis de la misma. Por lo tanto, la segmentación de imágenes suele ser usada para localizar objetos; es decir, una clasificación de los objetos que aparecen en la imagen. Más concretamente, la segmentación de imágenes es el proceso de asignar una etiqueta a cada píxel de una imagen de forma que los píxeles con la misma etiqueta compartan características similares.

Este proceso de segmentación se realiza mediante un conjunto de algoritmos. Podemos agruparlos en los basados en visión por computador y en el aprendizaje automático. Ambos serán explicados en profundidad en las siguientes subsecciones.

La segmentación de imágenes es un paso fundamental en muchos algoritmos de procesamiento de imágenes, como el reconocimiento de objetos, el análisis de imágenes y la compresión de imágenes. También puede utilizarse para mejorar la precisión y velocidad de otras tareas de visión por ordenador, como el seguimiento de objetos, la estimación del movimiento y la navegación.

2.1.2. Segmentación basada en Visión por Computador

La **Visión por Computador** (VC) [25] es una rama interdisciplinar que trata de entender las imágenes digitales o vídeos, de forma que pueda procesarlos automáticamente tal y como el sistema visual humano lo hace. En otras palabras, se encarga de facilitar la capacidad de ver y entender el mundo al igual que los humanos lo percibimos.

Su capacidad de percepción le proporciona la posibilidad de incluir el procesamiento de imágenes, el seguimiento y reconocimiento de objetos, **la segmentación de imágenes** y la reconstrucción 3D, entre otras.

Entre los algoritmos basados en VC encontramos como tarea objetivo la segmentación. Para que podamos entenderlos con claridad, se explican a continuación algunos de los métodos de segmentación basados en visión por computador.

Métodos basados en contornos activos

Los contornos activos, introducidos en [26], utilizan las fronteras para dividir una imagen en diferentes regiones o etiquetas; es decir, cada una de las segmentaciones será un objeto diferente de la imagen.

Esta técnica se basa en la idea de que los contornos de un objeto son una característica importante para su identificación. El algoritmo, el cual se puede observar en la figura 2.1, inicia su procedimiento en un punto específico dentro de la imagen y rastrea tanto hacia adelante como hacia atrás el contorno del objeto, empleando ciertos criterios para determinar la finalización del contorno. Durante el seguimiento del contorno, se añaden progresivamente puntos a una máscara que representa la región correspondiente al objeto en la imagen. Una vez que el algoritmo ha completado el contorno, se puede utilizar la máscara para segmentar la imagen en diferentes regiones.

Los contornos activos se utilizan para seguir los bordes de un objeto en una imagen y permiten que la segmentación se adapte dinámicamente a los cambios en el contorno del objeto. Estos algoritmos son útiles para segmentar objetos en imágenes que pueden tener formas complejas y cambiantes.

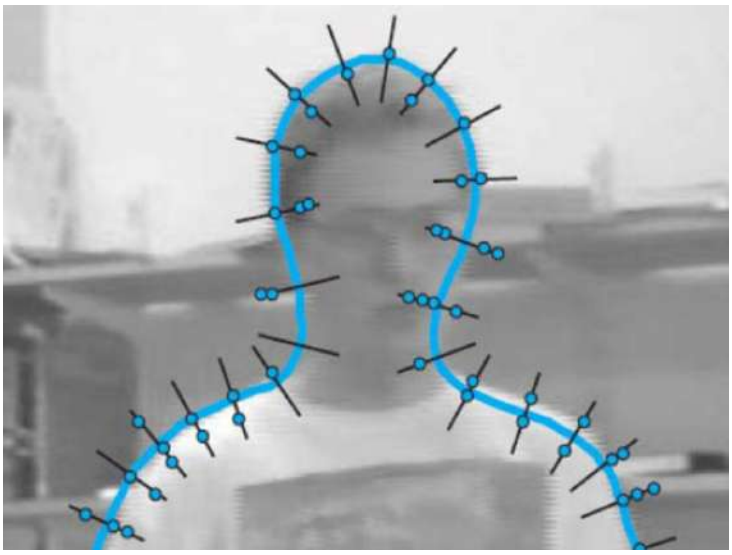


Figura 2.1: Técnica de segmentación basada en contornos activos [2].

Métodos basados en dividir y fusionar

Esta técnica de segmentación de imágenes [25] divide la imagen en múltiples subregiones pequeñas, las cuales posteriormente se fusionan para conformar regiones de mayor tamaño. Este proceso se repite de forma iterativa hasta obtener las regiones deseadas. Este procedimiento tiene una filosofía parecida a la técnica clásica, divide y vencerás; es decir, es más fácil segmentar una imagen en pequeñas subregiones y luego fusionarlas que intentar segmentar la imagen completa de una sola vez.

Este enfoque de segmentación es útil en situaciones en las que la imagen contiene objetos de diferentes tamaños y formas, y puede ser difícil segmentar la imagen completa de una sola vez.

Figura 2.2: Segmentación por fusión basada en gráficos [3].

A modo de ejemplo, se muestra en la figura 2.2 una representación gráfica de un enfoque de segmentación por fusión basado en gráficos que emplea la técnica de dividir y fusionar. A partir de una imagen de entrada en escala de grises, se aprecia una segmentación resultante con un estudio del vecindario de 8 píxeles. Sin embargo,

es importante tener en cuenta que este enfoque supone un coste computacional alto y, por lo tanto, requiere una cantidad significativa de recursos computacionales.

Métodos basados en el desplazamiento de la media y búsqueda de la moda

Esta técnica [27] se fundamenta en el cálculo de la media o moda de los valores de intensidad de los píxeles presentes en una imagen. Una vez calculado, se desplaza la media o moda hacia una dirección en la que se encuentra una mayor concentración de píxeles; es decir, se emplean los cálculos para determinar si un punto pertenece al fondo o al objeto de interés en la imagen, utilizándolo para segmentar la imagen en diversas regiones.

La ventaja de esta técnica radica en su rapidez y eficiencia en términos de recursos computacionales. Sin embargo, puede llegar a ser menos precisa que otras técnicas de segmentación, ya que solo utiliza información sobre la media y la moda de los píxeles en una región, en lugar de utilizar información más detallada sobre los píxeles individuales.

Métodos basados en los cortes normalizados

Técnica [28] utilizada para llevar a cabo la segmentación de imágenes mediante el empleo de un algoritmo de corte de grafos. En este enfoque, se construye un grafo en el cual los nodos representan los píxeles de la imagen, mientras que las aristas se ponderan en función de la similitud entre los píxeles que conectan. Luego, se aplica un algoritmo de corte de grafos para dividir el grafo en diferentes componentes conexas, cada una de las cuales representa una región de la imagen.

2.1.3. Segmentación basada en Aprendizaje Automático

Por último, se aborda la segmentación de imágenes mediante el uso de técnicas de *Machine Learning* (ML). Este enfoque se basa en utilizar algoritmos de aprendizaje automático para aprender a segmentar imágenes a partir de un conjunto de datos de entrenamiento. El objetivo principal es que el modelo sea capaz de generalizar y segmentar de manera precisa imágenes que no se encuentren presentes en el conjunto de datos de entrenamiento, es decir, que pueda aplicar el conocimiento adquirido a nuevas imágenes.

2.2. Aprendizaje Automático

Para explicar los algoritmos de segmentación de imágenes basados en el aprendizaje automático, se introduce previamente una breve definición sobre qué son los algoritmos basados en el aprendizaje, por qué se les considera algoritmos de aprendizaje automático y qué categorías existen.

El aprendizaje automático [29] es un área dentro de la Inteligencia Artificial, Machine Learning (ML), en la que se diseñan algoritmos que automatiza el proceso de aprendizaje a partir de datos. Una vez se les ha proporcionado una cantidad suficiente de datos relevantes, estos pueden identificar patrones, relaciones entre los datos y ser utilizados para realizar predicciones precisas o tomar decisiones a partir del conocimiento adquirido.

Podemos seccionar las técnicas de ML según las formas de abordar la extracción de conocimiento; es decir, basándose en la información disponible del conjunto de datos que vayamos a tratar. En las siguientes subsecciones se explica en detalle las posibles variantes de estudio sobre el ML.

2.2.1. Diferentes enfoques de Aprendizaje Automático

El aprendizaje automático es un campo amplio que engloba diversos enfoques para abordar la resolución de problemas y la toma de decisiones a partir de los datos. En esta subsección, se describen tres enfoques principales del ML: el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo.

Aprendizaje supervisado

En el aprendizaje supervisado [29] se dispone de un conjunto de datos compuestos por elementos que corresponden a los datos de entrada y los datos de salida. Los algoritmos que utilizan este enfoque se caracterizan por generar un modelo el cual ha sido entrenado con los resultados esperados (salida) a partir de un conjunto de variables (entrada). Este modelo genera las operaciones oportunas sobre un nuevo conjunto de datos de entrada para obtener resultados; es decir, obtiene datos de salida nuevos a partir de unos datos de entrada nuevos. Los resultados obtenidos habituales son los valores numéricos en el caso de problemas de regresión, o etiquetas de clase en el caso de problemas de clasificación, entre otros.

Por ejemplo, consideremos un algoritmo que utiliza el enfoque de aprendizaje supervisado para el reconocimiento de animales en imágenes. Este algoritmo sería entrenado utilizando un conjunto de imágenes etiquetadas, donde cada imagen tiene asociada una etiqueta de clase que indica a qué animal corresponde la imagen (por ejemplo, perro, gato, conejo, etc.). Durante el proceso de entrenamiento, el modelo aprende a reconocer patrones y características visuales que distinguen a cada animal en las imágenes. Una vez que el modelo ha sido entrenado utilizando imágenes con sus correspondientes etiquetas, se puede utilizar para realizar predicciones sobre nuevas imágenes que no tienen definido a qué animal corresponde lo que aparece en su imagen.

Aprendizaje no supervisado

El aprendizaje no supervisado [29] no necesita trabajar con datos etiquetados; es decir, en lugar de ser entrenado con datos asociados a sus resultados esperados, el algoritmo de aprendizaje no supervisado puede aprender con datos sin etiquetar.

Por ejemplo, un algoritmo de aprendizaje no supervisado podría ser utilizado para analizar el comportamiento de los clientes de una tienda en línea y descubrir grupos de clientes similares en términos de sus patrones de compra. Esto puede ayudar, como Amazon suele sugerir, a indicar al siguiente cliente si estaría interesado en comprar un artículo relacionado con el producto que ya deseaba comprar.

Por otro lado, si buscamos un ejemplo orientado a la segmentación de imágenes, este podría ser el uso de un algoritmo para analizar un conjunto de imágenes sin etiquetas y descubrir patrones y relaciones en ellas. El algoritmo podría utilizar técnicas de *clustering* para agrupar las imágenes en diferentes categorías en función de sus características visuales similares.

Existen una serie de modelos de aprendizaje automático que solventan el enfoque del aprendizaje no supervisado, como puede ser el *clustering* o las reglas de asociación, las cuales son explicadas en la subsección 2.2.3.

Aprendizaje por refuerzo

El aprendizaje por refuerzo se utiliza para enseñar a un agente a tomar decisiones que maximicen una recompensa específica; es decir, el agente es liberado en un entorno y recibe recompensas o castigos en función de sus acciones. A medida que el

agente experimenta el entorno y recibe recompensas, va mejorando en su capacidad para tomar decisiones que maximicen la recompensa.

Por ejemplo, un algoritmo de aprendizaje por refuerzo podría ser utilizado para enseñar a un robot a moverse de forma eficiente en un laberinto. Si lo orientamos a la segmentación por imágenes, un ejemplo podría ser el uso de un algoritmo para enseñar a un agente a identificar objetos específicos en una imagen. El agente podría ser liberado en un entorno que le permita interactuar con una imagen y recibir recompensas por la identificación correcta de objetos en la imagen y, a medida que el agente experimenta con el entorno, aprender a identificar objetos con mayor precisión. Por último, el conocimiento adquirido es utilizado para realizar una segmentación precisa de la imagen.

2.2.2. Tareas de Aprendizaje Automático

Las tareas son problemas específicos que se pueden resolver mediante técnicas y algoritmos de ML, los cuales se categorizan de forma diferente según el tipo de problema que se está resolviendo y el tipo de datos que se están utilizando. A continuación, se desarrolla brevemente una serie de tareas de interés.

Regresión

La regresión [30] es una de las tareas clásicas del aprendizaje automático utilizada para predecir valores numéricos continuos a partir de un conjunto de características o variables predictivas. Por ejemplo, podríamos utilizar un algoritmo de regresión para predecir el precio de una casa en función de su tamaño, su ubicación y su antigüedad. Podemos diferenciar entre varios tipos de modelos de regresión, como por ejemplo la regresión lineal y la regresión no lineal, entre otros.

Clasificación

La clasificación [31], como tarea enfocada al aprendizaje supervisado, está definida por la identificación de etiquetas en cada uno de los datos del conjunto; es decir, los datos de entrenamiento están previamente etiquetados y contienen información sobre las características o variables independientes de cada observación, así como la variable objetiva o etiqueta que indica a qué clase pertenece cada observación. La

clasificación se basa en la predicción de dicha etiqueta a partir de un conjunto de datos de entrenamiento etiquetados, los cuales proporcionan al modelo la posibilidad de crear el camino que predice a qué etiqueta corresponde un dato nuevo.

Para abordar problemas basados en la tarea de la clasificación de los datos en una clase u otra, es necesario encontrar un modelo que permita asignar una clase a una nueva observación a partir de su conjunto de características. Para ello, se usan una serie de modelos basados en la clasificación (véase la subsección 2.2.3) como por ejemplo: el vecino más cercano, los árboles de decisión, la máquina de soporte de vectores (SVM) y las redes neuronales, entre otros.

Agrupamiento

El agrupamiento [32] o *clustering*, como tarea del ML, es utilizada para dividir un conjunto de datos en grupos o *clusters* de forma que los datos dentro de cada *cluster* sean similares entre sí, pero diferentes de los datos de otros grupos. Esta tarea es empleada para la exploración de datos y la segmentación de ellos, ya que permite descubrir patrones y estructuras ocultas en los datos. Por ello, debemos tener presente que esta técnica utiliza un enfoque de aprendizaje no supervisado.

Supongamos un ejemplo para la segmentación de imágenes en el que se procesan imágenes médicas para identificar órganos o estructuras en una imagen. Con este ejemplo, podemos tener un conjunto de imágenes de resonancia magnética que muestre diferentes órganos del cuerpo del paciente (torso, cabeza, corazón, riñones, etc.) donde se usa el *clustering* para dividir estas imágenes en diferentes grupos, de acuerdo con las características similares de los órganos que se muestran en cada imagen. De esta forma, podemos identificar las imágenes que muestran el cerebro como un grupo, otro grupo para las imágenes que muestren el corazón, y así sucesivamente.

Existen una serie de algoritmos mencionados en la siguiente subsección, como los basados en el centroides, los basados en densidad y los basados en jerarquías, entre otros.

2.2.3. Modelos de Aprendizaje Automático

En este apartado, se procede a desarrollar una serie de modelos de aprendizaje automático junto con sus respectivos enfoques, tal como se ha mencionado previamente.

Basados en árboles

Los modelos de ML basados en árboles [29] son uno de los modelos efectivos utilizados para la clasificación y la regresión. Estos representan una toma de decisiones que se genera en forma de árbol; es decir, el árbol muestra en cada nodo una decisión y en cada hoja una clasificación o predicción.

Supongamos que queremos diseñar un sistema para diagnosticar enfermedades; es decir, clasificar según una serie de patrones si el paciente tiene una enfermedad o no, basándose en síntomas y pruebas de laboratorio. Este modelo podría ayudar de forma que cada nodo del árbol podría representar una pregunta, ¿el paciente tiene fiebre?, o ¿el paciente tiene dolor de cabeza?, entre otras. Además, mediante las ramas del árbol, se representan las posibles respuestas a cada pregunta. Esto nos lleva a indagar en el dato con sus respectivas variables (pruebas, datos del paciente, etc.) para la representación del diagnóstico final.

El entrenamiento del algoritmo se realiza a partir de un conjunto de datos o *dataset* que contenga las observaciones de las variables que se quieren utilizar para tomar decisiones y la decisión final que se quiere predecir. Es importante tener en cuenta, en cualquier técnica de ML, qué datos han de ser utilizados para entrenar un modelo, siendo considerada la parte de preprocesamiento necesaria para una buena construcción del modelo.

La construcción del modelo comienza por la selección de la variable que permite hacer una mejor división o diferenciación de los datos en base a la clase y divide el conjunto de datos en varios subconjuntos basándose en esa variable. Luego se repite el proceso para cada subconjunto de datos, el cual ya no contiene la variable por la cual ha sido dividido previamente. Una vez los subconjuntos son lo suficientemente pequeños u homogéneos, se toma la decisión final.

Al hablar sobre homogeneidad, se refiere a que todo el subconjunto que ha sido dividido en la anterior ejecución contiene el mismo valor para la variable predictora; es decir, para el ejemplo de la clasificación de enfermedades nos indicaría que todo ese subconjunto de datos indica que tiene la misma etiqueta indicando el tipo de enfermedad.

En la figura 2.3 se muestra un ejemplo del resultado del C4.5, algoritmo basado en la construcción de un árbol para el conjunto de datos de Iris. En cada nodo del árbol se representa una de las cuatro características del conjunto de datos; es decir, longitud y anchura del sépalo y pétalo. Cada ramificación representa un valor límite

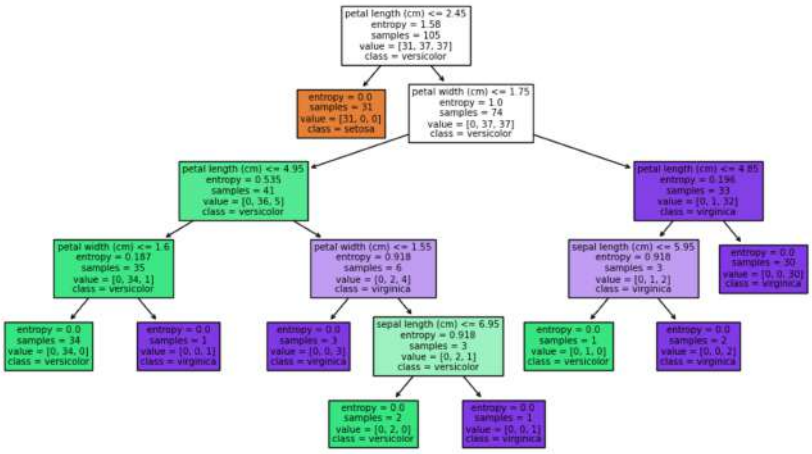


Figura 2.3: Ejemplo de Árbol de decisión C4.5 para el *dataset* de Iris de sklearn. Elaboración propia.

para la característica elegida y cada hoja del árbol, o nodo final, representa una clase predicha para las observaciones que llegan a ella. Como vemos, si un nodo hoja indica que su clase es versicolor, nos proporciona la información de que todo aquel dato u observación que acabe llegando a dicha hoja del árbol será clasificada como versicolor. Este *dataset* es frecuentemente usado para la simulación y observación de cómo funciona un algoritmo basado en árboles de decisión.

En general, los árboles de decisión son muy populares debido a su simplicidad y facilidad de uso, dando una serie de ventajas, como por ejemplo que son modelos explicables, rápidos de entrenar y no requieren de una gran cantidad de datos para su posible entrenamiento, entre otros. Pero, por otro lado, son propensos al sobreajuste y pueden ser menos precisos que otros modelos con un entrenamiento más complejo que requieran de más tiempo y esfuerzo para ser entrenados.

Basados en máquina de vectores

Los modelos basados en máquinas de vectores o SVM [29] se centran en encontrar la mejor separación posible entre las diferentes clases del conjunto de datos, lo que les permite clasificar de manera efectiva nuevos datos.

Las SVM marcan como objetivo la búsqueda del hiperplano óptimo de separación, el cual debe maximizar la distancia entre las clases; es decir, el margen a la distancia entre el hiperplano de separación y los puntos más cercanos a este hiperplano en cada

clase. Por ello, la parametrización resulta un punto clave para encontrar el equilibrio entre la maximización del margen y la minimización del error de clasificación.

Consideremos un ejemplo de segmentación de imágenes mediante el uso de SVM, como la detección de células cancerígenas respecto de las células naturales. La tarea de segmentación es identificar y separar las células cancerígenas de las células normales en una imagen, de forma que el modelo entrene con un conjunto de datos de imágenes histológicas etiquetadas como células cancerígenas o no. Una vez se entrena, clasifica cada uno de los píxeles de la imagen como célula cancerígena o normal.

Clustering basados en centroides

Este modelo crea grupos o *clusters* con centroides [29]; es decir, puntos que representan el centro de cada grupo. El objetivo es minimizar la distancia entre cada punto y el centroide de su grupo correspondiente para poder agrupar todos los datos que se asemejen por características. Dentro de este modelo de agrupamiento, existen algoritmos como el K-Means.

El algoritmo **K-Means** selecciona k puntos de los datos como centroides iniciales y asigna cada punto del conjunto de datos al grupo cuyo centroide esté más cerca. Después, se recalculan los centroides como la media de los puntos del grupo y se repite el proceso hasta que los centroides ya no cambien. Este tipo de algoritmo de *clustering* es de los más conocidos y se basa en la siguiente estructura:

1. Se especifica el número de grupos que se quieren encontrar, k.
2. Se eligen k puntos al azar del conjunto de datos como centroides iniciales de cada grupo.
3. Se asigna cada punto del conjunto de datos al grupo cuyo centroide esté más cerca.
4. Se calcula la media de cada grupo y se reemplaza el centroide por esa media.
5. Se vuelven a asignar los puntos al grupo cuyo centroide esté más cerca.
6. Se repiten los pasos 4 y 5 hasta que los centroides de los grupos no cambien más o hasta que se alcance un número máximo de iteraciones que se le indiquen al algoritmo.

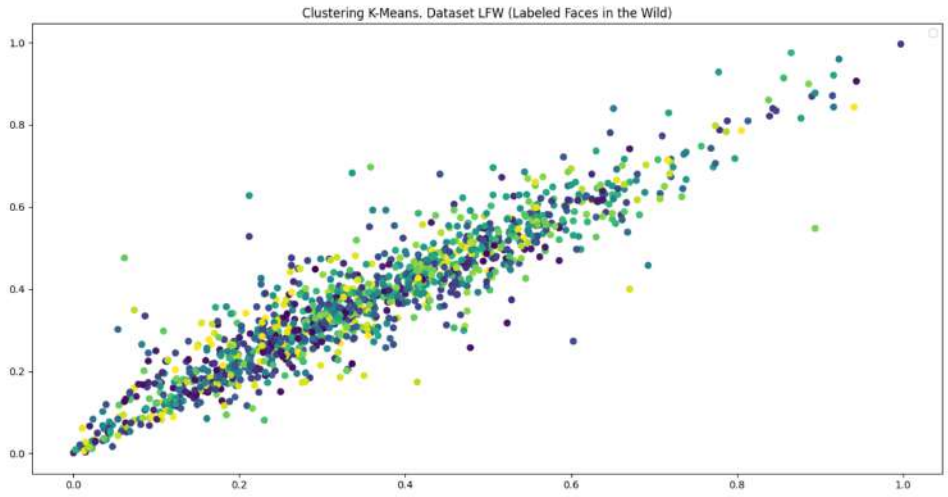


Figura 2.4: Ejemplo de *Clustering* K-Means para el dataset LFW (*Labeled Faces in the Wild*). Elaboración propia.

En la figura 2.4 se muestra la representación del conjunto de datos de imágenes LFW (*Labeled Faces in the Wild*), utilizando el algoritmo K-Means para dividir las imágenes en 50 grupos basándose en sus características. Después, el algoritmo K-Means se ajusta a los datos y se utiliza para predecir a qué grupo pertenece cada imagen. Como podemos observar, cada punto representa una imagen del conjunto LFW y las imágenes que tienen el mismo color indican que son del mismo grupo.

Basados en redes neuronales

Las redes neuronales [29] representan modelos inspirados en la estructura y funcionamiento del cerebro humano. Una red neuronal se compone de capas de neuronas interconectadas, donde cada una de ellas lleva a cabo una operación matemática en la entrada y genera una salida que se transfiere a la capa siguiente.

La capacidad de las redes neuronales para identificar patrones complejos en los datos las posiciona como uno de los modelos más interesantes del momento. Durante el proceso de entrenamiento, se realizan ajustes en los pesos de las conexiones entre las neuronas con el fin de minimizar una función de pérdida que cuantifica la diferencia entre las salidas esperadas y las salidas predichas.

Consideremos un conjunto de imágenes en blanco y negro que representan dígitos, y nuestro objetivo es clasificar cada imagen de manera que se corresponda con su número equivalente; es decir, si una imagen muestra el número uno en su repre-

sentación de píxeles, esta debe ser clasificada en la etiqueta 1, y así sucesivamente con cada uno de los dígitos.

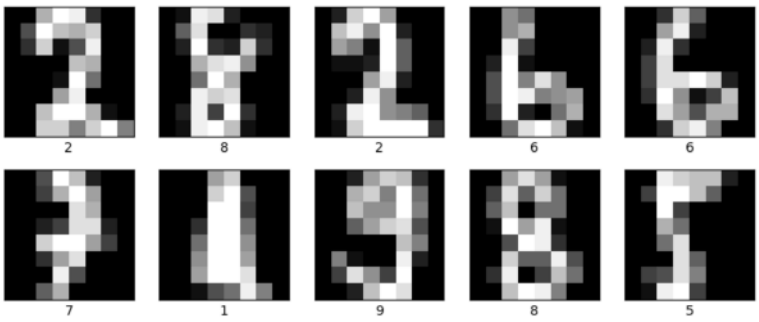


Figura 2.5: Resultados obtenidos en un proceso de identificación de números con una red neuronal con 3 capas y 30 neuronas. Elaboración propia.

En la figura 2.5 se presenta un análisis del ejemplo de clasificación de dígitos utilizando el algoritmo del Perceptrón Multicapa (MLP) con una arquitectura de 3 capas ocultas, cada una compuesta por 30 neuronas. Se muestra una serie de datos del conjunto de imágenes de pruebas y su correspondiente clasificación a partir de la etiqueta.

En la sección 2.3 se explica en detalle el concepto de aprendizaje profundo, el cual está ligado a las redes neuronales.

2.3. Aprendizaje Profundo

El Aprendizaje Profundo o *Deep Learning* (DL) [33] se considera una rama del aprendizaje automático que se fundamenta en la aplicación de redes neuronales artificiales. La definición de este concepto se caracteriza por las diversas capas de procesamiento que integran las redes neuronales, donde cada capa se enfoca en la detección de características específicas. Estas capas reciben como entrada los resultados de las capas anteriores y aprenden progresivamente, desde patrones simples en las capas iniciales hasta patrones complejos en las capas superiores, basándose en la información extraída por las capas inferiores.

En la figura 2.6 se ve representado, a través de un diagrama de conjuntos, la relación existente entre los conceptos planteados hasta el momento, como la Inteligencia Artificial, el Aprendizaje Automático (*Machine Learning*) y el Aprendizaje Profundo (*Deep Learning*).

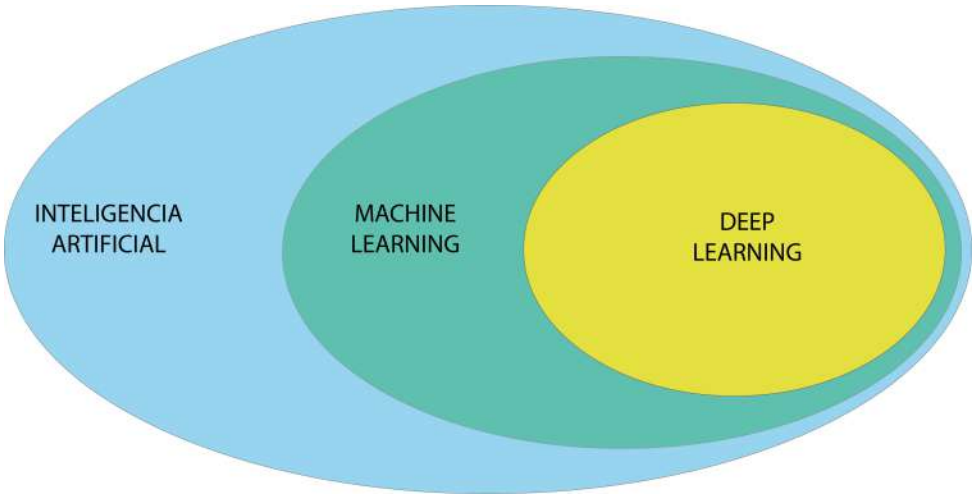


Figura 2.6: Diagrama de conjuntos, representando la IA, el ML y el DL. Elaboración propia.

Como hemos mencionado anteriormente, el ML puede abordarse a través de enfoques supervisados y no supervisados, entre otros. Por lo tanto, como nos encontramos en el ámbito del Aprendizaje Profundo que utiliza modelos basados en redes neuronales con una estructura profunda, esto implica que se puede emplear tanto para resolver problemas con enfoque supervisado como no supervisado.

El DL en la actualidad se basa en gran medida en modelos neuronales, es por eso que en la siguiente sección se empieza con la descripción del modelo más sencillo.

2.3.1. El Perceptrón

El Perceptrón [4], también conocido como la unidad mínima de procesamiento en los algoritmos basados en redes neuronales, fue uno de los primeros modelos de aprendizaje automático desarrollados en la década de 1950 por Frank Rosenblatt (véase la figura 2.7).

Este modelo matemático está inspirado en la forma en que el cerebro procesa la información; es decir, consiste en una estructura compuesta por una capa de entrada con n nodos y una capa de salida con un solo nodo. Cada nodo de la capa de entrada está conectado a una entrada correspondiente de un patrón de entrada y se le asigna un peso específico. El nodo de salida es una función que toma en cuenta los pesos asignados y las entradas provenientes de la capa de entrada.

Durante el proceso de entrenamiento, se presentan diferentes patrones de entrada junto con sus correspondientes salidas deseadas. A partir de estos ejemplos, el Per-

ceptrón ajusta los pesos para minimizar la diferencia entre las salidas generadas y las salidas deseadas. Como resultado, genera una salida que representa la clasificación del patrón de entrada.

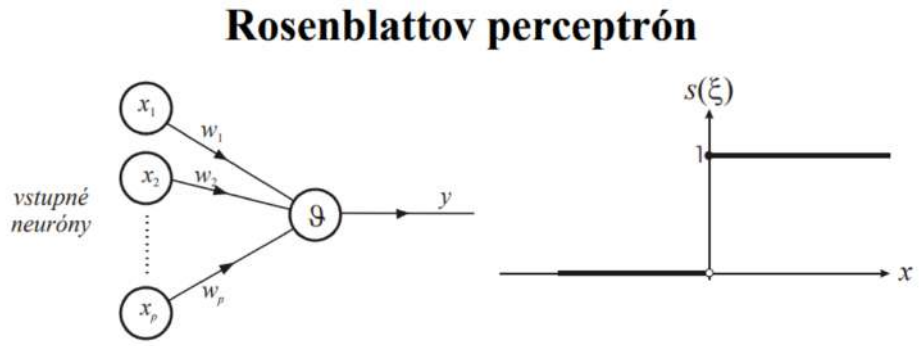


Figura 2.7: Representación gráfica del Perceptrón. [4].

2.3.2. El Perceptrón Multicapa

A partir del Perceptrón, se desarrolló una arquitectura más compleja y poderosa conocida como el Perceptrón Multicapa, también llamado MLP (*Multi-Layer Perceptron*) [5], red neuronal artificial compuesta por múltiples capas de neuronas. El MLP ha sido ampliamente utilizado en diversas tareas de ML (véase la subsección 2.2.2) debido a su capacidad para aprender funciones más complejas que el Perceptrón, gracias a la introducción de capas ocultas y funciones de activación no lineales suaves.

Arquitectura del Perceptrón Multicapa

La arquitectura del MLP consta de una capa de entrada, una o más capas ocultas y una capa de salida. Cada capa contiene una serie de neuronas completamente conectadas a las neuronas de la siguiente capa, donde las conexiones se ponderan por coeficientes llamados pesos, los cuales se ajustan durante el proceso de entrenamiento para minimizar el error en las predicciones realizadas por la red.

La figura 2.8 proporciona una representación gráfica de la arquitectura del MLP, que muestra claramente la diferencia entre la capa de entrada en verde, las capas ocultas en rosa y la capa de salida en azul.

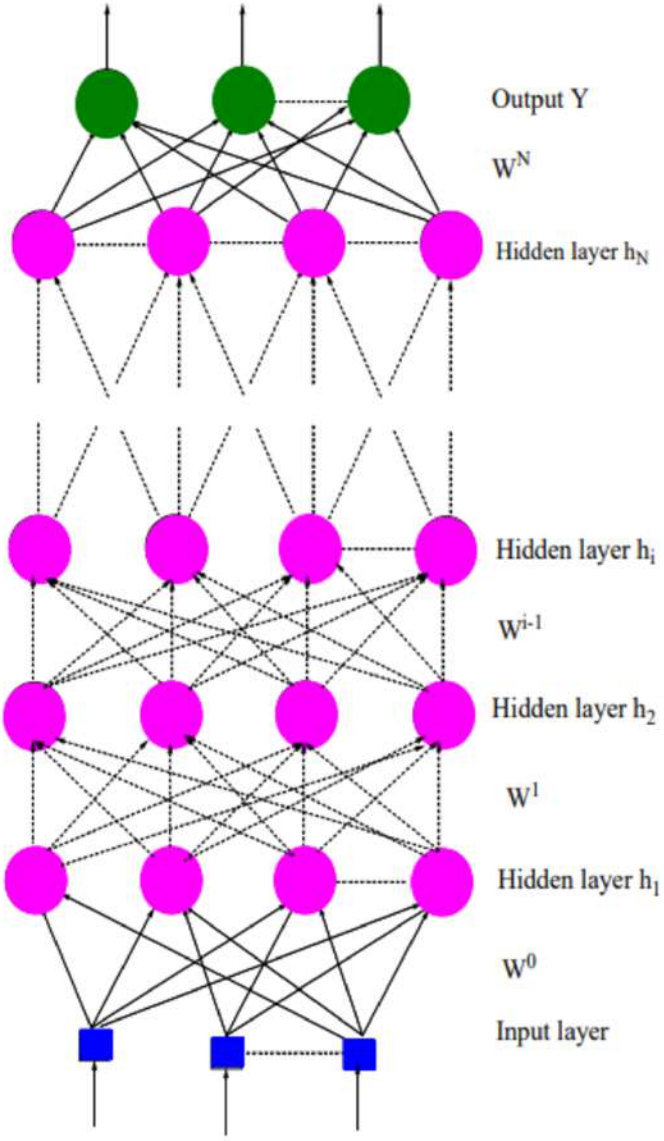


Figura 2.8: Representación gráfica del Perceptrón Multicapa [5].

- **Capa de entrada**

  Esta capa está formada por neuronas que representan las características de entrada del problema. Cada neurona en esta capa se asocia con un atributo de los datos de entrada y no realiza ningún cálculo. Su función principal es distribuir los valores de entrada a las neuronas en la siguiente capa.

- **Capas ocultas**

  Capas compuestas por neuronas que realizan cálculos basados en las entradas recibidas de la capa anterior y los pesos asociados a las conexiones. El número de capas ocultas y el número de neuronas en cada capa son determinadas por la arquitectura del MLP. Además, a medida que se aumenta el número de capas y neuronas, la capacidad de la red para modelar funciones complejas y aprender

relaciones no lineales en los datos mejora, pero también aumenta el riesgo de sobreajuste y costo computacional.

- **Capa de salida**

  La capa de salida está formada por neuronas que representan las predicciones del modelo. El número de neuronas en la capa de salida depende del tipo de problema que se esté resolviendo, como clasificación binaria o multiclase, regresión, etc. Las neuronas en esta capa utilizan una función de activación específica para el problema, como la función softmax para clasificación multiclase o la función lineal para regresión, entre otras.

Entrenamiento del MLP

El MLP se entrena utilizando un algoritmo de optimización conocido como *backpropagation* [34]. Este algoritmo utiliza el gradiente de la función de coste para determinar la dirección en la que deben actualizarse los pesos para minimizar el error de la red; es decir, el objetivo del entrenamiento es ajustar los pesos de la red para minimizar el error en las predicciones realizadas por el modelo.

A continuación, se explica el proceso por el cual pasa el MLP para entrenar su red, fortaleciendo la métrica a optimizar.

- **Inicialización**

  Los valores de los pesos de la red se inicializan de forma aleatoria, generalmente con valores pequeños. Esto permite que a red tenga la capacidad de mejorar y buscar el óptimo global de los hiperparámetros con los siguientes pasos.

- **Propagación hacia delante**

  Para cada instancia de entrenamiento, los valores de entrada se propagan a través de la red, capa por capa, hasta llegar a la capa de salida. En cada neurona, se calcula una suma ponderada de las entradas y se genera una salida según una función de activación. La salida final de la red se compara con el valor objetivo o etiqueta de la instancia de entrenamiento para calcular el error.

- **Propagación del error**

  Una vez calculado el error en las neuronas de salida, este se propaga hacia atrás, desde la capa de salida hasta la capa de entrada, en un proceso conocido como retropropagación del error o *backpropagation*.

En cada neurona, se calcula una derivada parcial del error con respecto a cada peso de entrada y se acumula en un término de corrección de peso.

- **Actualización de pesos**

  Después de procesar todas las instancias de entrenamiento, los pesos de la red se actualizan utilizando los términos de corrección de peso acumulados y una tasa de aprendizaje.

  La tasa de aprendizaje es un hiperparámetro que controla la magnitud de los cambios en los pesos durante el entrenamiento y puede ajustarse para mejorar el rendimiento del modelo.

- **Iteración**

  Se repiten los pasos anteriores (excepto la inicialización) para procesar todas las instancias de entrenamiento y actualizar los pesos de la red en cada iteración. Esto se repite hasta que se alcance un criterio de detención, como un número máximo de iteraciones o una mejora mínima en la métrica de evaluación.

Durante el entrenamiento del MLP, es importante tener en cuenta el riesgo de sobreajuste, ya que el modelo puede aprender de manera demasiado específica los ejemplos de entrenamiento y no generalizar bien a nuevos datos.

2.3.3. Funciones de activación

Continuando con la comprensión de las redes neuronales, es importante entender cómo cada neurona se conecta con la siguiente a través de una función matemática, conocida como **función de activación** [35]. Cada neurona está asociada con una función de activación que determina su comportamiento en respuesta a la entrada. La motivación de estas funciones es fortalecer la no linealidad en las redes neuronales, lo que les permite aprender patrones complejos en los datos de entrada.

Existen diferentes tipos de funciones de activación, entre las cuales se encuentran las mencionadas a continuación.

Función escalón

La función escalón es una función de activación binaria que produce una salida de 1 si el argumento es mayor o igual a 0 y una salida de 0 si es menor que cero.

$$f(x) = \{1\ si\ x\ \geq\ 0,\ 0\ si\ x\ <\ 0\} \tag{2.1}$$

Esta función se utilizó en el Perceptrón original (véase la figura 2.7) y en algunas redes neuronales simples. En contextos de problemas de baja complejidad suele ser útil para clasificaciones binarias, pero puede ser limitada para problemas más complejos en comparación con otras funciones de activación.

Función sigmoide

Es una función continua y diferenciable que produce una salida entre 0 y 1 en respuesta a la entrada. Se utiliza en las capas ocultas de las redes neuronales debido a su capacidad para modelar relaciones no lineales y suavizar las transiciones entre la entrada y la salida.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

La función sigmoide, como aparece en la figura 2.9, no es simétrica con respecto a 0; es decir, el valor en el eje de ordenadas oscila siempre entre [0,1] y tienen el mismo signo para todos los valores de entrada. Esta función puede escalar el resultado ajustando sus parámetros.

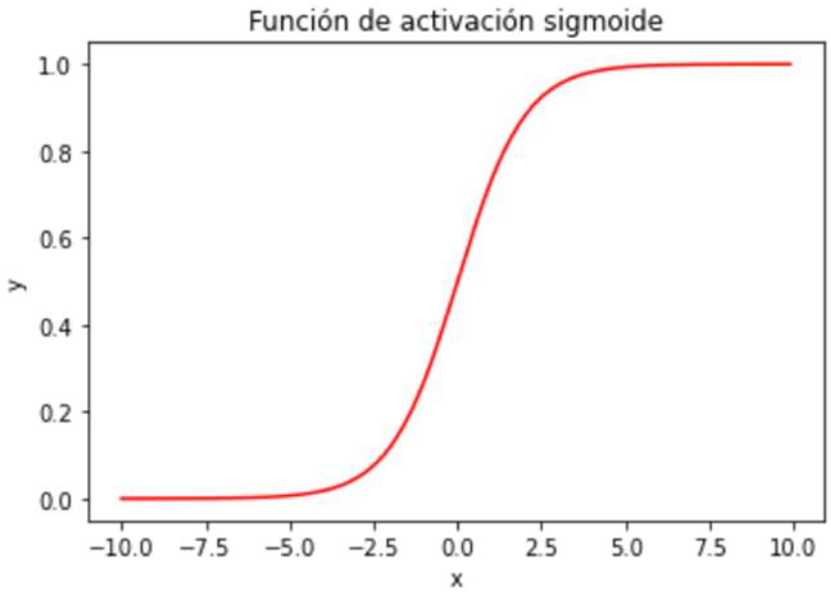


Figura 2.9: Función de activación sigmoide. Elaboración propia.

Función tangente hiperbólica

La función tangente hiperbólica [35], o TanH, es una función de activación continua y diferenciable que produce una salida entre -1 y 1 en respuesta a la entrada.

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

En comparación con la función sigmoide, el gradiente de TanH es más pronunciado, como se ilustra en la figura 2.10. Además, a diferencia de la función sigmoide, TanH está centrada en 0 y puede generar tanto valores positivos como negativos en su salida.

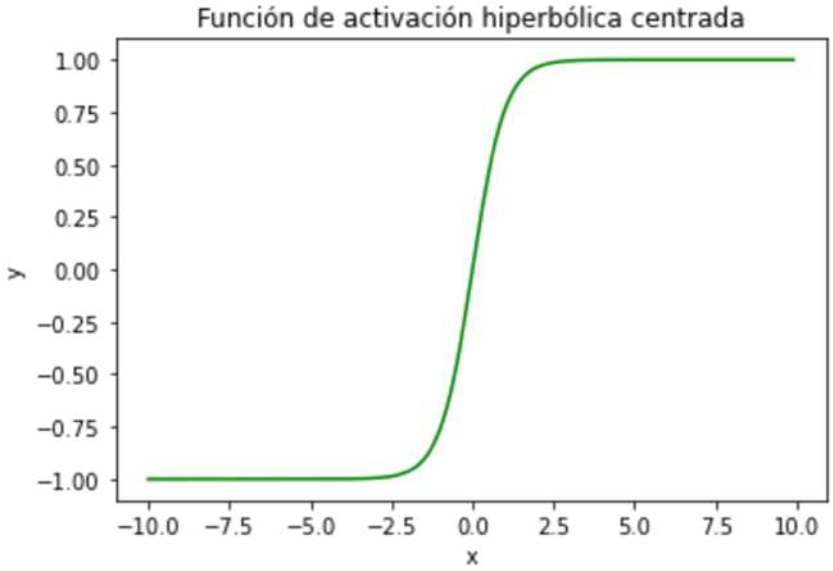


Figura 2.10: Función de activación tangente hiperbólica o TanH. Elaboración propia.

Hasta el momento, las funciones de activación mencionadas pueden sufrir del problema del **gradiente evanescente** (véase la subsección 2.3.4). Para abordar este problema, han surgido funciones de activación diseñadas específicamente para resolverlo, como la función ReLU, SELU y GELU, entre otras.

La función ReLU (*Rectified Linear Unit*) es una función de activación no lineal que se utiliza comúnmente en las redes neuronales convolucionales debido a su simplicidad y eficiencia computacional, lo que la hace popular en aplicaciones de DL.

$$f(x) = \text{máx}(0, x) \tag{2.4}$$

En la figura 2.11 se observa que ReLU produce una salida de 0 cuando el argumento es negativo y una salida igual al argumento cuando es positivo. Sin embargo, puede sufrir de un problema conocido como *Dying ReLU* [36], que ocurre cuando una neurona con función de activación ReLU siempre produce una salida de 0, lo que significa que no aprende y no contribuye al proceso de aprendizaje. Esto puede suceder cuando la suma ponderada de las entradas a la neurona es siempre negativa durante el entrenamiento, lo que hace que el gradiente sea 0 y no haya actualización de los pesos.

Existen variantes de ReLU [37] que abordan este problema, como las mencionadas a continuación.

- **L-ReLU**, o *Leaky Rectified Linear Unit*

  Modificación de la función ReLU que permite un pequeño gradiente negativo cuando la entrada es menor que 0.

$$f(x) = \max(\alpha x, x) \tag{2.5}$$

  Si nos fijamos en la fórmula, α corresponde a un hiperparámetro que controla la pendiente negativa, permitiendo que la función tenga un gradiente distinto de cero incluso cuando la entrada es negativa, lo que ayuda a mitigar el problema.

- **P-ReLU**, o *Parametric Rectified Linear Unit*

  Modificación de ReLU que fuerza al parámetro del coeficiente de la pendiente negativa a ser aprendido en lugar de ser un hiperparámetro fijo.

$$f(x) = \max(\alpha_i x, x) \tag{2.6}$$

  En este caso, α_i es un parámetro aprendido específico para la neurona i que permite a la función de activación adaptarse mejor a los datos por su aprendizaje del valor óptimo para la pendiente negativa.

Función SELU

SELU (*Scaled Exponential Linear Unit*) es una función de activación que induce propiedades de auto-normalización en las redes neuronales profundas, de esta forma favorece a las redes neuronales a reducir el problema del gradiente evanescente.

$$f(x) = \lambda \{x \; si \; x > 0, \; \alpha(e^x - 1) \; si \; x \; \leq \; 0\} \tag{2.7}$$

Si nos fijamos en la fórmula, tenemos constantes como λ y α, las cuales son constantes positivas predefinidas.

En comparación con la función de activación ReLU, esta normaliza internamente los datos, evitando tener que hacerlo cada n capas o por lotes (véase la comparativa en la figura 2.11).

Función GELU

La función de activación GELU (*Gaussian Error Linear Units*) es utilizada para obtener salidas no lineales en las neuronas. Existe similitud con ReLU, pero su diferencia radica en la suavidad de su curvatura, lo que la hace diferenciable.

$$f(x) = GELU(x) = x\Phi(x), \tag{2.8}$$
$$donde \; \Phi(x) = \frac{1}{2}(1 + erf \; (x/\sqrt{2})).$$

En general, las funciones de activación mostradas en la figura 2.11 se comportan de manera diferente según el contexto en el que se utilizan. Por ejemplo, la función GELU es eficiente para la retroalimentación positiva, lo que implica aplicar una serie de transformaciones lineales y no lineales a la entrada para producir una salida, también conocido como *feedforward*. Por esta razón, la función GELU se utiliza comúnmente en redes neuronales profundas, como los *Transformers*, para mejorar su capacidad para aprender tareas de procesamiento del lenguaje natural. Es por ello que, en general, se utiliza comúnmente para mejorar la capacidad de la red neuronal profunda para aprender tareas de procesamiento del lenguaje natural.

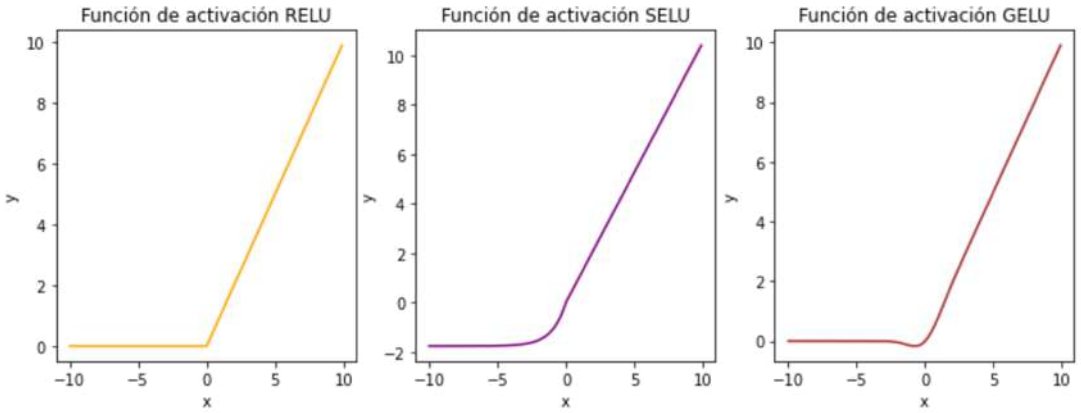


Figura 2.11: Función de activación ReLU, SELU y GELU. Elaboración propia.

2.3.4. El problema del gradiente evanescente

El gradiente evanescente [38], o también conocido como el problema del gradiente que desaparece, es un problema clave en la evolución del DL. Cuando entrenamos una red neuronal, utilizamos un algoritmo llamado retropropagación o *backpropagation* [34] para actualizar los pesos de las conexiones entre las neuronas.

En una red neuronal profunda, donde hay muchas capas entre la entrada y la salida, el gradiente de la función de coste se propaga hacia atrás a través de múltiples capas para actualizar los pesos. Sin embargo, a medida que el gradiente se propaga, cada vez se hace más pequeño. Esto se debe a que el gradiente de cada capa es el producto de los gradientes de las capas anteriores, y estos gradientes se multiplican juntos a medida que se propagan hacia atrás.

Cuando los gradientes se vuelven muy pequeños, la actualización de los pesos también se vuelve muy pequeña, dificultando el aprendizaje de las capas profundas. En algunos casos, los gradientes pueden llegar a ser tan pequeños que los pesos no se actualizan en absoluto, lo que se conoce como el problema del gradiente evanescente. La forma en la que este problema perjudica en el entrenamiento de las redes neuronales profundas, debido a la importancia de la información contenida en las capas profundas, nos conduce a un mal entrenamiento y, por consiguiente, la red no podrá aprender patrones complejos en los datos de entrada viéndose afectado su rendimiento.

2.3.5. La evolución hacia el Aprendizaje Profundo

Aunque el MLP fue uno de los primeros modelos en utilizar capas ocultas, el DL como tal surgió a partir de avances significativos en el entrenamiento de redes neuronales más profundas y en la comprensión teórica de su funcionamiento. Estos avances nacen de la necesidad de aprender representaciones más complejas y abstractas en problemas de ML.

El entrenamiento de las redes neuronales profundas fue posible gracias a una serie de factores clave que, al combinarlos, posibilitaron su origen:

- **Mejoras en el algoritmo de retropropagación**

  Como se comentó anteriormente, este algoritmo era fundamental para el algoritmo MLP, pero su capacidad de manejar redes con muchas capas estaba restringido por el problema del gradiente evanescente (véase la subsección **??**). Por ello, gracias a las investigaciones de nuevas funciones de activación que mitigan dicho problema, se permite entrenar redes neuronales más profundas de manera eficiente.

- **La capacidad de procesamiento hardware**

  El crecimiento exponencial en el poder de cómputo, en particular con la llegada de las unidades de procesamiento gráfico (GPU) y la computación distribuida, permitió realizar cálculos intensivos en paralelo y acelerar significativamente el proceso de entrenamiento de las redes neuronales profundas.

Características clave del Aprendizaje Profundo

Los modelos basados en DL están caracterizados por darnos la posibilidad de representar datos más complejos y abstractos, esto es debido a una serie de factores o características diferenciadoras como son las siguientes.

- **La gran capacidad de jerarquizar características**

  Las capas inferiores capturan características de bajo nivel y las capas superiores capturan características de alto nivel que son combinaciones de las características de bajo nivel, dando la posibilidad al algoritmo de aprender patrones complejos y generalizar mejor a datos a tratar a futuro.

- **La capacidad de modelado**

  Comparado con las redes neuronales no profundas, su capacidad de modelado permite adaptarse a una amplia variedad de tareas y dominios de interés para nuestra investigación; es decir, involucran datos de alta dimensionalidad como por ejemplo las imágenes.

- **El uso de funciones de activación no lineales**

  Estas funciones, como por ejemplo las mostradas en la figura 2.11, permiten a las redes neuronales profundas aprender y representar relaciones más complejas debido a su no linealidad entre las variables de entrada y salida.

- **La capacidad de transferencia de aprendizaje**

  Permite la transferencia de conocimientos [39] aprendidos en una tarea a otra; es decir, una red neuronal profunda entrenada en un conjunto de datos grande y diverso puede ser ajustada para adaptarse a una tarea relacionada con menos datos disponibles, lo que reduce el tiempo de entrenamiento y suele mejorar el rendimiento en comparación con entrenar una red desde cero.

Hitos en el desarrollo del Aprendizaje Profundo

La evolución de las redes neuronales profundas ha surgido a partir de una serie de hitos clave en el desarrollo de la arquitectura de las mismas y sus aplicaciones; es decir, dependiendo de la aplicación o tarea a desarrollar, han surgido una serie de redes neuronales profundas relevantes.

- **Redes Neuronales Convolucionales (CNN)** [8]

  La CNN es un tipo de red neuronal profunda especialmente diseñada para el procesamiento de imágenes y la detección de características visuales. Debemos tener en cuenta que las CNN aprovechan la relación entre los píxeles cercanos en una imagen (estructura local) y la capacidad de una red neuronal para reconocer objetos independientemente de su posición en la imagen (invarianza traslacional) para reducir el número de parámetros y mejorar el rendimiento en tareas de clasificación y reconocimiento de objetos. Este hito será desarrollado en profundidad en la sección 2.3.6. Además, existen diferentes aplicaciones de las CNN para la segmentación de imágenes, las cuales se verán en la sección 2.4.

- **Redes Neuronales Recurrentes (RNN)** [40]

  La RNN es un tipo de red neuronal profunda que puede manejar secuencias de datos variables en longitud, como series temporales o texto. Por lo tanto, este hito de red neuronal profunda tiene conexiones recursivas entre sus neuronas, lo que le permite procesar secuencias de longitud arbitraria, es decir, estas conexiones le permiten recordar información previa y usarla para influir en su salida actual.

  En otras palabras, la red neuronal recurrente mantiene en memoria los datos que ha procesado previamente y los utiliza para tomar decisiones futuras. Existen diferentes formas de representación entre las conexiones de las neuronas, puede ser una conexión totalmente interconectada o parcialmente interconectada, como aparece en la figura 2.12.

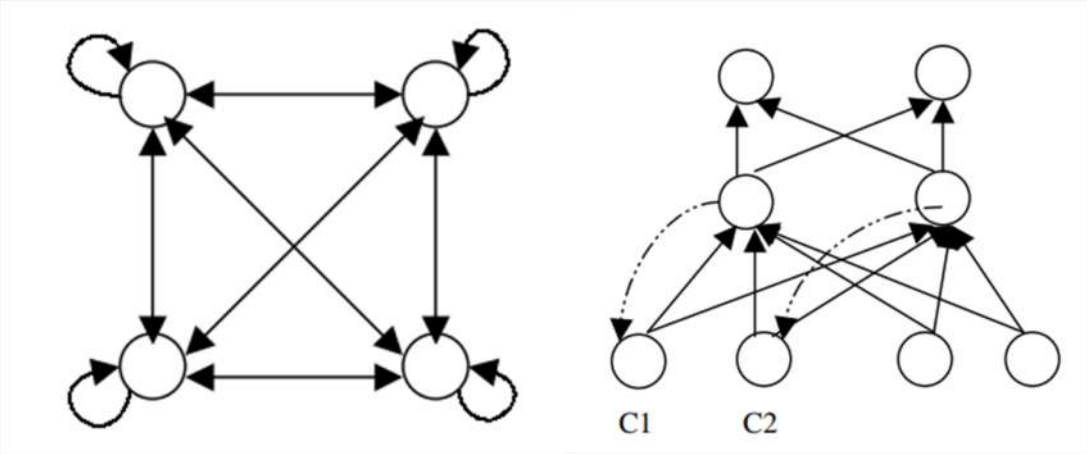


Figura 2.12: RNN totalmente y parcialmente interconectada [6].

- **Autocodificadores profundos** [41]

  Red neuronal profunda que aprende a comprimir y reconstruir datos, lo que les permite aprender representaciones de características útiles y de baja dimensión de los datos de entrada. Estos modelos han demostrado ser efectivos en tareas como la reducción de dimensionalidad, la extracción de características y el aprendizaje de representaciones no supervisadas.

  Los *autoencoders* (AE) son redes neuronales que codifican los datos de entrada y son entrenados para que, al decodificar dichos datos, sean lo más parecido posible a las entradas. Es importante tener en cuenta que, aunque se ha desarrollado para el contexto del DL, no todos los *autoencoders* son redes neuronales con múltiples capas ocultas o redes neuronales profundas. Con esto sabemos que el AE puede ser profundo o superficial con una única capa oculta.

- **Redes Generativas Adversarias (GAN)** [42]

  La GAN es un marco de DL que utiliza dos redes neuronales en competencia, una generadora y una discriminadora, para aprender a generar datos que se asemejen a los datos de entrenamiento. La GAN ha demostrado ser efectiva en una amplia gama de aplicaciones, como la generación de imágenes, la transferencia de estilo y la síntesis de datos.

  En una GAN, hay dos redes neuronales que trabajan juntas: el generador y el discriminador. El generador se encarga de generar datos sintéticos a partir de un ruido aleatorio de entrada, mientras que el discriminador se encarga de distinguir entre los datos sintéticos y los datos reales. Por lo tanto, durante el entrenamiento, el generador crea datos sintéticos y los presenta al discriminador como si fueran reales. Por último, el discriminador evalúa la calidad de los datos y devuelve una puntuación que indica cuán cerca están los datos de ser reales para que el generador pueda mejorar su rendimiento y crear datos sintéticos de mayor calidad.

- ***Transformers*** [7]

  Los *Transformers* nacen de la necesidad generada por las limitaciones de las RNN o LSTM (*Long Short-Term Memory*) debido a la dificultad del procesamiento de secuencias largas de datos, causado por el problema del desvanecimiento del gradiente (véase la subsección 2.3.4). La arquitectura de los *Transformers* usa una estructura de atención en lugar de la recurrencia, la cual calcula pesos de atención entre todas las entradas de la secuencia, permitiendo que la red pueda procesar de manera paralela y eficiente información relevante en la secuencia completa. La estructura de atención también permite que la red se centre en las partes más importantes de la secuencia, lo que puede mejorar su capacidad para capturar información relevante y hacer predicciones precisas.

  Como podemos observar en la figura 2.13, se expone la arquitectura del *Transformer*. Este modelo consta de dos componentes principales: el codificador y el decodificador. El codificador toma una secuencia de entrada y la procesa para crear una representación de la entrada, mientras que el decodificador toma dicha representación y la utiliza para generar una secuencia de salida. Ambas están compuestas por múltiples capas de atención y de transformación de características (*feedforward*).

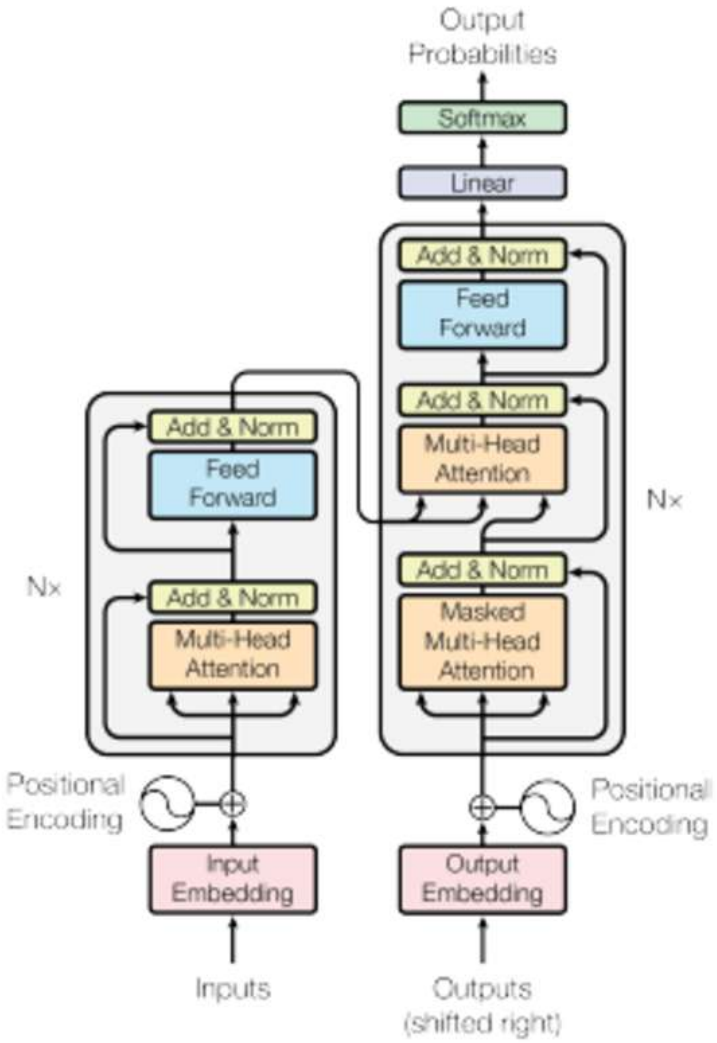


Figura 2.13: *Transformer*. Arquitectura modelo [7].

2.3.6. Redes Neuronales Convolucionales

Una vez introducida una serie de hitos, las redes neuronales que más se adaptan a nuestro problema a resolver son las Redes Neuronales Convolucionales, debido a su buena adaptación y aplicación en el procesamiento de imágenes.

La **CNN** [8] es un tipo especializado de red neuronal profunda basada principalmente en el uso de la red para el análisis de imágenes, la clasificación y la segmentación de imágenes, entre otras muchas aplicaciones. Surgen de la variación de las Redes Neuronales Artificiales clásicas diseñadas para trabajar con datos estructurados en forma de matrices; es decir, redes que trabajan con imágenes debido a su representación en matrices de sus valores.

En la figura 2.14 se muestra un ejemplo de red convolucional para reconocer caracteres, donde el plano de entrada recibe imágenes de caracteres aproximadamente normalizados en tamaño y posición, donde cada unidad de una capa recibe entradas de un conjunto de unidades situadas en una pequeña vecindad en la capa anterior.

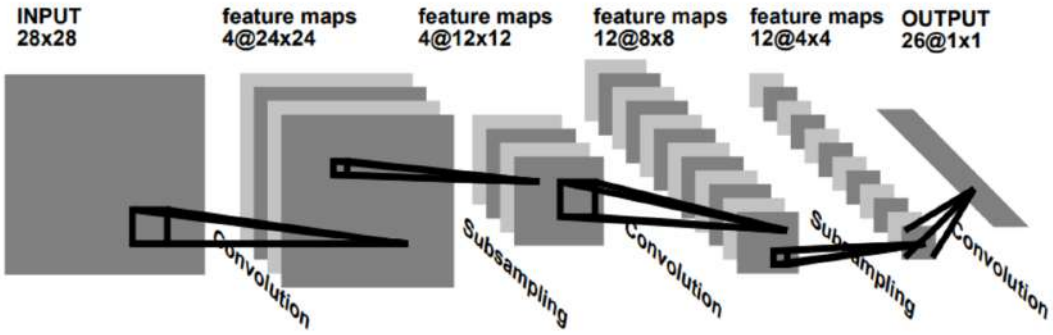


Figura 2.14: CNN para procesamiento de imágenes, reconocimiento de escritura. [8].

Las conexiones locales han sido utilizadas en su mayoría por modelos de redes neuronales basados en el procesamiento de imágenes. Por ello, las neuronas pueden extraer características visuales elementales con los campos receptivos locales, como bordes orientados, puntos finales, esquinas, etc.

A continuación, se detalla la arquitectura de las CNN y en la sección 2.4 se muestran una serie de variantes específicas para el problema a resolver.

Arquitectura de las CNN

Las CNN generalmente incluyen tres tipos de capas distribuidas a lo largo de su estructura (véase la figura 2.15). Entre las variantes nos encontramos las capas convolucionales, las capas no lineales y las capas de agrupación.

- **Capas convolucionales**

  Caracterizadas por ser la parte central de una CNN, utilizan una serie de filtros o núcleos que se aplican a la imagen de entrada, permitiendo detectar características de bajo nivel como bordes, texturas y colores. Esto es aplicado en la red durante el proceso de aprendizaje, donde cada filtro se aplica sobre una región local de la imagen, lo que permite a la CNN aprender características a la posición.

- **Capas no lineales**

  Estas capas aplican, generalmente por elementos, una función de activación a los mapas de características, lo que permite que la red modele funciones no lineales.

- **Capas de agrupación o *pooling***

  Estas capas se utilizan para reducir la dimensionalidad de los datos y mejorar la eficiencia computacional. Es decir, reducen la resolución espacial al reemplazar los vecindarios pequeños en un mapa de características con información estadística sobre esos vecindarios (por ejemplo, la media, el máximo u otros.).

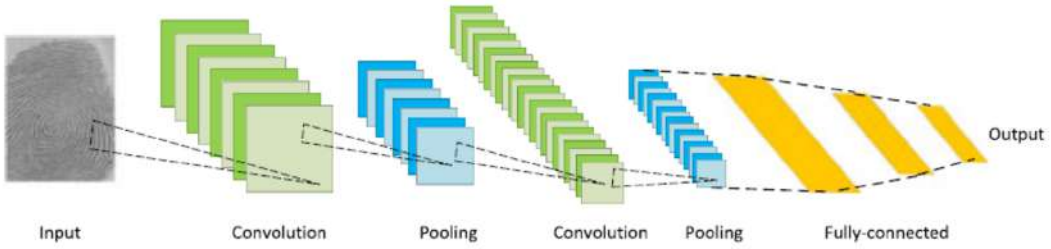


Figura 2.15: Arquitectura de las CNN. [9].

2.4. Segmentación de imágenes con Aprendizaje Profundo

Siguiendo con el conocimiento adquirido en la sección 2.1, la segmentación de imágenes se plantea como un proceso de clasificación de píxeles, y su aplicación varía según el problema a abordar, presentando las siguientes variantes.

- **Segmentación semántica**

  Esta segmentación aplica el etiquetado píxel a píxel, a partir de un conjunto de etiquetas. Por ejemplo, si tenemos un *dataset* de fotos de vecindarios, las etiquetas pueden ser humano, árbol, cielo, casa, entre otros. En nuestro caso, utilizaremos este tipo de segmentación para la resolución de la segmentación de los animales en nuestro conjunto de imágenes de foto-trampeo.

- **Segmentación de instancias**

  Por otro lado, tenemos una ampliación en la detección y recorte de cada objeto de interés, pudiendo individualizar objetos que a priori con la segmentación semántica serían de la misma etiqueta. Por ejemplo, si aparecen varias personas en la imagen, este se encargaría de poder diferenciar y etiquetar por separado.

- **Segmentación panóptica**

  Esta segmentación aplica una combinación de la segmentación semántica y la segmentación de instancias.

A continuación, se muestra una serie de algoritmos de segmentación de imágenes basada en el DL que nos ayudan a entender cómo resolver nuestro problema.

2.4.1. Mediante Redes Neuronales Convolucionales

En este apartado, se exploran diferentes modelos de segmentación de imágenes basados en redes neuronales convolucionales y se analiza su aplicación en la tarea de la segmentación semántica.

Redes Neuronales Totalmente Convolucionales, o FCN

La **FCN** [10] es una variante de la CNN que se utiliza principalmente para la segmentación semántica, las cuales no tienen capas totalmente conectadas al final de la red, sino que utilizan capas convolucionales en toda la red, lo que les permite manejar imágenes de cualquier tamaño y proporcionar mapas de características más detallados. Estas redes son capaces de aprender a asignar píxeles de entrada a píxeles de salida, lo que resulta en una segmentación con mayor precisión.

Esta red neuronal convolucional, como la mostrada en la figura 2.16, es caracterizada por la **eliminación de las capas totalmente conectadas** al final de la red, debido a la restricción de tener que usar un tamaño fijo de entrada para el conjunto de imágenes a tratar. Con esta red neuronal, esta restricción desaparece, ya que las FCN reemplazan estas capas por capas convolucionales, lo que permite a la red manejar imágenes de diferentes tamaños. Es decir, en la figura 2.15 aparece en la última fase de la red neuronal a lo que nos referimos como las capas totalmente conectadas, siendo en este caso sustituido por capas convolucionales.

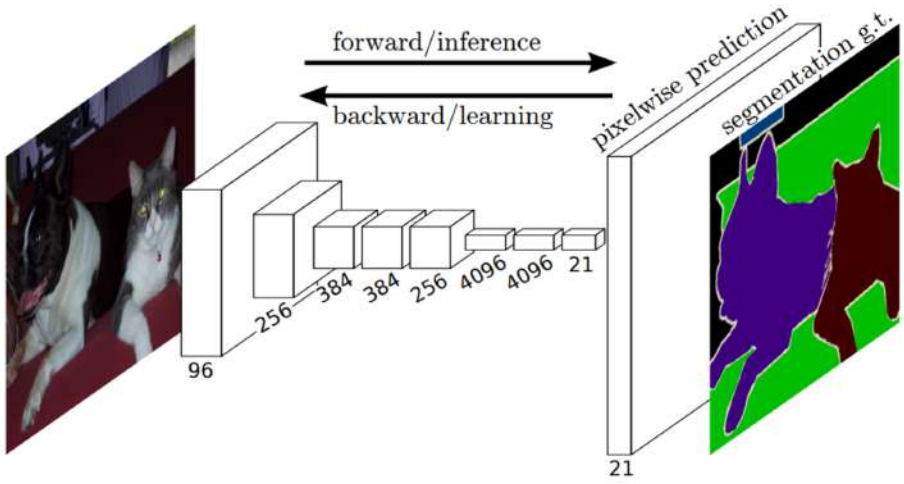


Figura 2.16: Red Neuronal Totalmente Convolucional. [10].

Otra característica de esta red neuronal es el uso de la técnica ***upsampling***, proceso en el que se incrementa la resolución espacial de una imagen o un conjunto de datos. Es decir, se crea una imagen de mayor resolución a partir de una de menor resolución, permitiendo que la red genere una salida de segmentación con la misma resolución que la imagen de entrada. Se aplica esta técnica debido a que, conforme se avanza en la red convolucional, la resolución de los mapas de características disminuye a medida que se avanza a través de las capas convolucionales y las capas (*pooling*). Existen una serie de técnicas de *upsampling*, algunas de ellas son la interpolación *Nearest-neighbor*, la interpolación bilineal y la deconvolución, entre otras.

La interpolación *Nearest-neighbor* utiliza la definición del algoritmo clásico del vecino más cercano, replicando los valores de los píxeles para llenar los espacios vacíos al aumentar la resolución.

La interpolación bilineal utiliza un número de vecinos estipulado, calculando la ponderación lineal de los píxeles vecinos para calcular el valor de los nuevos píxeles en la imagen de mayor resolución.

Y la deconvolución, técnica que aplica capas convolucionales invertidas para la obtención de una mayor resolución de los mapas de características.

También tenemos las **conexiones de salto**, mostrada en la figura 2.17, las cuales se basan en saltar una o varias capas de la red y conectan una capa anterior con una posterior con el propósito de mejorar el flujo de información y conocimiento en la red neuronal. Además, incitan a eliminar en cierta medida el problema del gradiente evanescente, ya que evitan que se pierda información importante durante el entrenamiento.

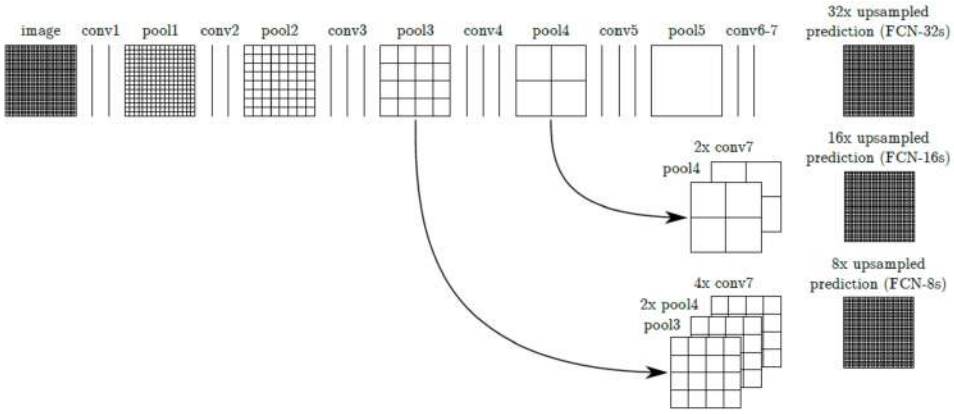


Figura 2.17: Conexiones de salto en FCN. [10].

Existe una limitación al respecto con este tipo de red neuronal convolucional, la **limitación en la captura del contexto semántico**. Esta limitación afecta (entre otras) al rendimiento y precisión de la predicción o clasificación a la hora de detectar los objetos y la segmentación semántica; es decir, en este tipo de tareas como la detección de objetos es posible que existan objetos similares con diferentes contextos semánticos, provocando una clasificación incorrecta por parte de la red neuronal o quizás su segmentación no sea suficientemente exacta para el problema a abordar.

Es por ello que, se hablara posteriormente de otros modelos que mitigan el problema de la limitación del contexto semántico, como pueden ser las CNN con modelos basados en grafos 2.4.1, entre otros.

CNN con modelos basados en grafos

Las Redes Neuronales Convolucionales basadas en grafos, o GCNs [11], son una extensión de las CNNs, ya que aparte de estar diseñadas para procesar imágenes y datos en formas de matrices, estas manejan datos en forma de **grafos** permitiendo capturar relaciones más complejas y estructuras jerárquicas en los datos (véase la figura 2.18).

Las **CNN con Campos Aleatorios Condicionales Totalmente Conectados** [12]. Este modelo fue creado a partir de dos técnicas principales, las Redes Convolucionales Profundas (DCNN) y Campos Aleatorios Condicionales Totalmente Conectados (CRF), con el objetivo de segmentar semánticamente.

Por un lado, tenemos la DCNN (figura 2.19), especializada para el procesamiento y reconocimiento de imágenes, siendo uno de los algoritmos base para este modelo, el VGG-16, siendo una red neuronal profunda útil para la extracción de mapas

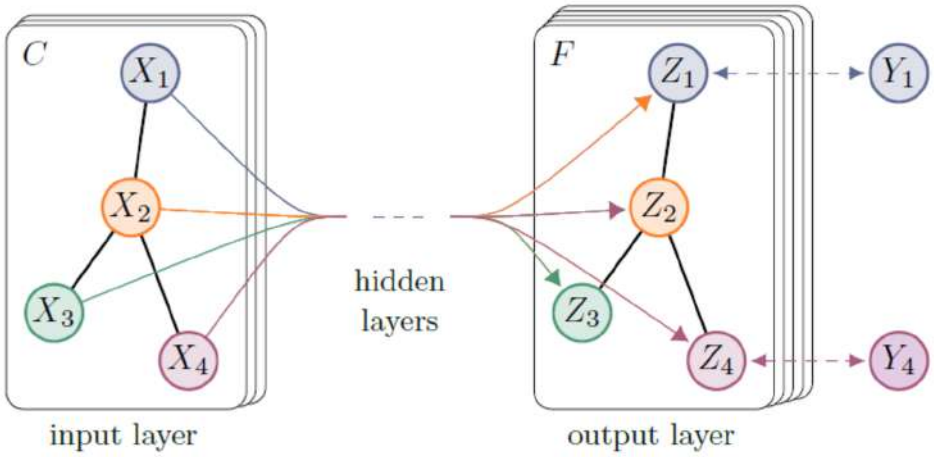


Figura 2.18: Red Convolucional basada en grafos. [11].

de características. Sabemos que las redes neuronales profundas se caracterizan por utilizar las capas totalmente conectadas para la clasificación, pero en este caso se reemplazan por capas convolucionales 1x1, dando mapas de puntuación de clase de resolución espacial más alta (beneficioso para la segmentación semántica). Es decir, al reemplazar las capas totalmente conectadas por estas capas convolucionales 1x1, se conserva la información espacial que de otro modo se perdería en las capas totalmente conectadas y se consigue de esta forma obtener información más detallada entre píxel para una mejor segmentación semántica.

Por otro lado, tenemos los CRF, los cuales aparecen en el último paso de la figura 2.19 antes de obtener la salida final. Estos modelos gráficos probabilísticos permiten modelar la relación existente entre los atributos del conjunto de datos y su variable predictora. En este caso, al ser totalmente conectados, mantienen la información contextual para la segmentación.

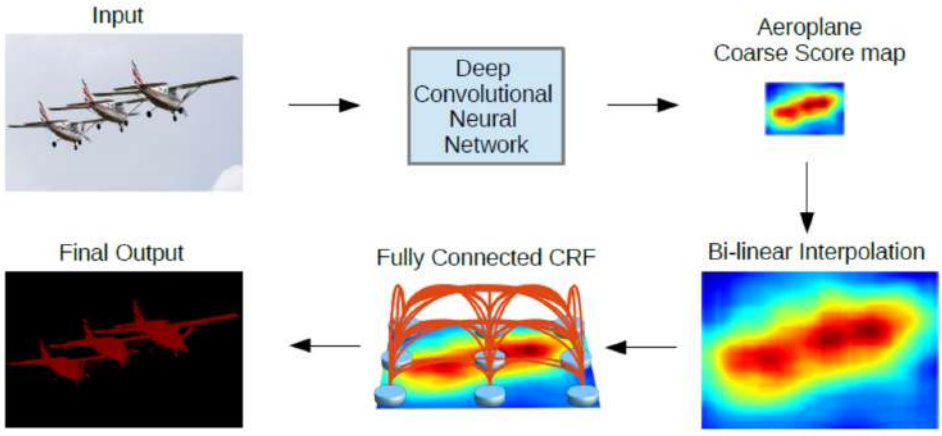


Figura 2.19: CNN unido con CRF. [12].

Modelos multiescala y basados en redes piramidales

Estos modelos, caracterizados por solventar problemas de VC, ya sea mediante la segmentación de imágenes o el etiquetado semántico, capturan información contextual y detalles en las imágenes que proporcionan en sus resultados una interpretación mejorada de la semántica y la segmentación.

Como modelo puntero tenemos el *Pyramid Scene Parsing* o PSPNet. Este modelo nos proporciona una combinación entre la arquitectura de una CNN con una estructura piramidal capacitada para captar información contextual a diferentes escalas y, además, obtener así una mejora de la segmentación semántica.

Este modelo, aparte de usar la extracción de características mediante redes convolucionales y *pooling*, tiene un módulo de Pirámide de Escena (PSP) que se considera el componente clave de este modelo. Esta característica diferenciadora divide las características extraídas con las redes convolucionales en varias regiones de diferentes tamaños y aplica *pooling* a cada región para capturar la información contextual relevante en cada escala. Por último, estas características agrupadas por regiones se redimensionan al tamaño original y se concatenan con las extraídas en el paso de extracción de características, permitiendo lo que se conoce como la fusión de características.

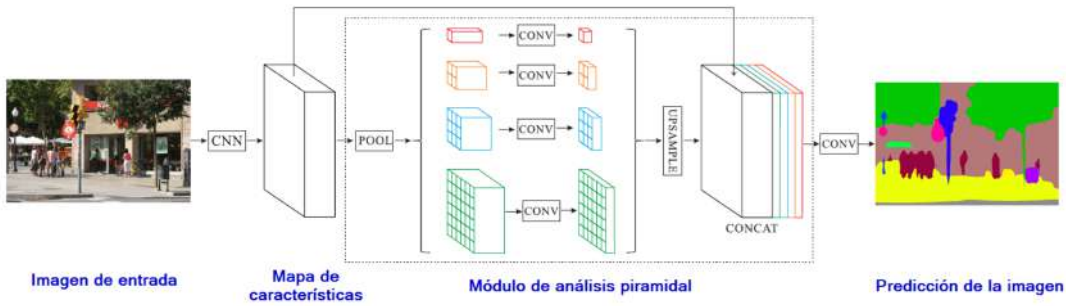


Figura 2.20: Arquitectura de la PSPNet. [13].

En la figura 2.20 podemos observar el paso desde que se introduce una imagen hasta la segmentación de dicha imagen. En esta figura nos fijamos en la sección del módulo de análisis piramidal que, a partir de la CNN usada para obtener el mapa de características, este módulo realiza diferentes subregiones con capas de re-muestreo y concatenación para que se pueda formar la representación final mediante contexto local y global. Por último, se aplica la representación sobre una capa de convolución para obtener la predicción final píxel a píxel.

Modelos basados en R-CNN

Mask R-CNN [14] es un modelo de segmentación de imágenes y detección de objetos desarrollado a partir de una extensión del modelo *Faster R-CNN*, el cual contiene una arquitectura aplicada para la detección de objetos. Pero, **Mask R-CNN** agrega una rama de segmentación en paralelo a la rama de detección de objetos en la Faster R-CNN dando la posibilidad de ser eficiente en problemas de segmentación de imágenes.

Mask R-CNN, aparte de usar la extracción de características de las CNN mediante capas convolucionales, aplica una **red de Propuestas de Región** o RPN (*Region Proposal Network*). Esta es una red neuronal convolucional que aplica regiones de interés (RoI) candidatas a partir de las características extraídas en las capas convolucionales.

Posteriormente, aparece el **RoIAlign** que no es lo mismo que el *RoI Pooling* de la *Faster R-CNN* (explicado más adelante). RoIAlign tiene como objetivo convertir las características extraídas de cada RoI, las cuales pueden tener diferentes tamaños y proporciones, en un único tamaño fijo (por ejemplo, 6x6). Los pasos del RoIAling se basan en lo siguiente:

1. Evasión de la cuantización. Sabemos que, en *Faster R-CNN*, se aplica la cuantización del RoI (redondear valores, para asignar las coordenadas del RoI a ubicaciones específicas en el mapa de características). En este caso, para **Mask R-CNN**, RoIAlign evita la cuantización de las coordenadas de las RoI, lo que reduce los errores de alineación y mejora la precisión en la segmentación de imágenes.

2. Interpolación bilineal. Mientras que *Faster R-CNN* divide las RoI en intervalos espaciales fijos y aplica *pooling* (en general, *max pooling*), RoIAlign utiliza interpolación bilineal para calcular los valores de las características en las ubicaciones exactas de las RoI en el mapa de características, obteniendo en ambos casos un mapa de características de tamaño fijo.

Otra característica de su arquitectura es la ramificación en dos del siguiente paso, la cual diferencia también a la red neuronal *Faster R-CNN* de *Mask R-CNN*. Esta ramificación consiste en dos ramas de aplicación: La rama de detección de objetos y la rama de segmentación.

- **La rama de detección de objetos**

  Similar a la rama de Faster R-CNN, la cual utiliza un clasificador *softmax* para predecir la clase de objeto en cada RoI y un regresor para ajustar las coordenadas del cuadro delimitador. Hasta aquí, aplicando RoI Pooling en vez de RoIAlign, sería las características de la red Faster R-CNN.

- **La rama de segmentación**

  Es una adición clave en la Mask R-CNN, la cual la hace especialmente adecuada para tareas de segmentación de imágenes. Es decir, la rama de segmentación es paralela a la rama de detección de objetos y predice una máscara binaria para cada clase de objeto en cada RoI. Además, como se ha explicado antes, en lugar de utilizar una capa de RoI Pooling la Mask R-CNN utiliza una capa de RoIAlign que evita la cuantización de las coordenadas del cuadro delimitador y las máscaras, lo que resulta en una segmentación más precisa.

Por último, tenemos la **decodificación y asignación de etiquetas**. Tras la predicción de las clases de objetos y las máscaras en cada RoI, Mask R-CNN combina esta información para generar segmentaciones de objetos en la imagen. Es decir, se selecciona la máscara binaria de la clase de objeto con la puntuación más alta para cada RoI, y las máscaras se combinan para formar la segmentación final de la imagen.

Para explicar esta última característica supongamos el ejemplo de un conjunto de imágenes donde aparecen personas y/o coches. Después de procesar las imágenes a través de la red, la Mask R-CNN generará máscaras binarias para cada RoI, que representan las diferentes clases de objetos (personas y coches). Si tenemos un RoI que contiene un coche, la Mask R-CNN aplica una máscara binaria para la clase coche y otra para la clase persona en esta RoI y da puntuaciones para cada una de las máscaras. Si la máscara de la clase coche tiene una puntuación de 0.8 y la de la persona tiene una puntuación de 0.3, entonces por definición se escogerá la máscara con mayor puntuación; es decir, la máscara coche.

Este proceso se repite en todas las RoI de cada una de las imágenes y una vez que, en cada imagen, todas las RoI tienen una máscara seleccionada, estas se combinan para formar la segmentación final de la imagen.

En la figura 2.21 vemos cada una de las características de la arquitectura descritas anteriormente y su aplicación en la segmentación de la imagen. Por ejemplo, podemos observar que se añade una rama para predecir máscaras de segmentación en cada región de interés (RoI), en paralelo con la rama existente para la clasificación y la regresión.

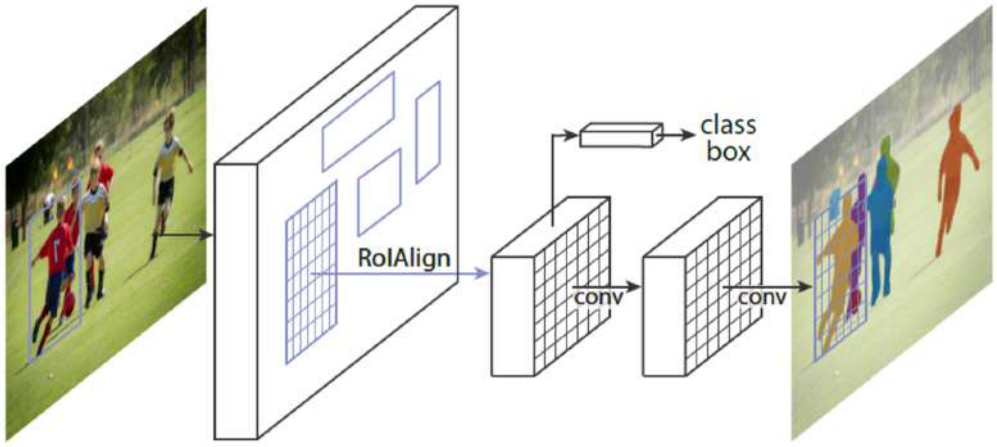


Figura 2.21: Mask R-CNN para segmentación de imágenes. [14].

2.4.2. Otros mecanismos de aprendizaje profundo

En esta sección se dará una breve explicación sobre otros mecanismos de DL para la segmentación de imágenes con modelos basados en *Autoencoders* o *Transformers*.

Mediante Autocodificadores profundos

En su mayoría, los modelos de segmentación de imágenes populares usan algún tipo de arquitectura de codificador-decodificador, arquitectura caracterizada por los autocodificadores explicada en la sección 2.3.5.

Nos centramos en las arquitecturas codificador-decodificador basadas en tareas de segmentación de imágenes, como por ejemplo modelos de segmentación de imágenes médicas y biomédicas, las cuales fueron creadas para una tarea concreta, pero que se extendieron su uso fuera del ámbito médico.

Por un lado, tenemos la **U-Net** [15] para segmentar imágenes de microscopia biológica, pero que se aplicó con éxito en otras tareas de segmentación de imágenes. U-Net es una variante del autocodificador convolucional con conexiones de salto o *Skip-CAE*. Es decir, los CAE (*Convolutional Autoencoders*) utilizan capas convolucionales tanto en el codificador como en el decodificador en vez de capas completamente conectadas con el objetivo de minimizar la diferencia entre la imagen de entrada y la imagen reconstruida, pero con los objetos de interés segmentados. Sin embargo, la *Skip-CAE* añade conexiones de salto entre las capas correspondientes del codificador y el decodificador para que las características de alta resolución del codificador se combinen

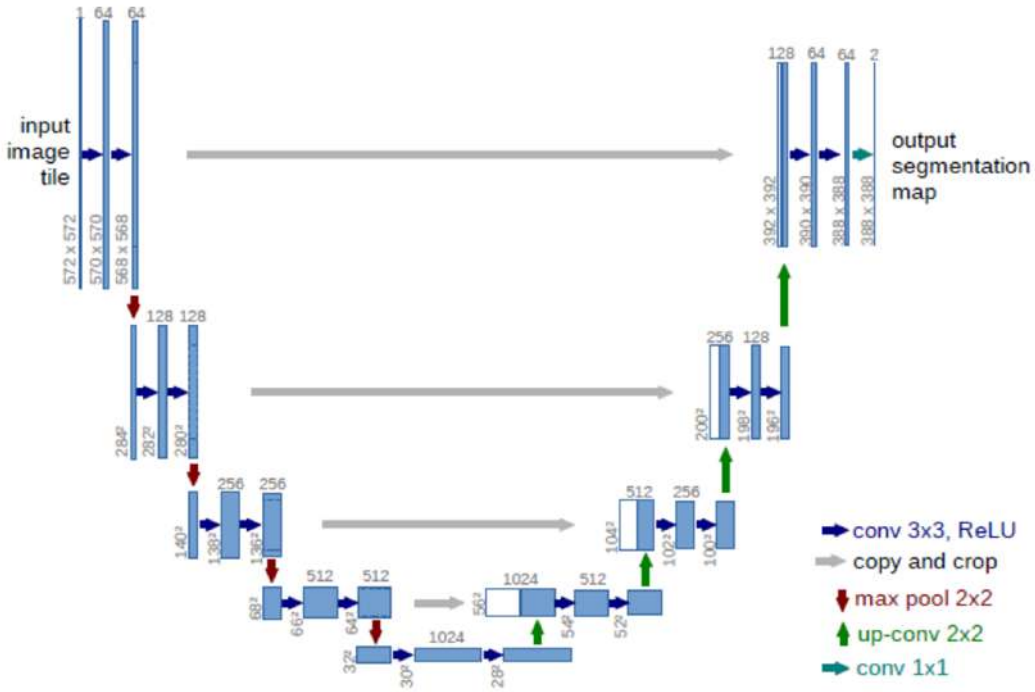


Figura 2.22: Arquitectura U-Net. [15].

con las características del decodificador, mejorando la calidad de la segmentación y la reconstrucción.

Pero, la U-Net es diferenciada de las anteriores por aplicar la conexión de salto a través de operaciones de concatenación, la cual implica unir la información de dos capas en un solo conjunto de datos. Esta combinación permite que las características de bajo nivel y las características de alto nivel se utilicen juntas en la reconstrucción de la imagen segmentada, mejorando así la precisión en la localización espacial.

En la figura 2.22 se observa la arquitectura de la U-Net para un ejemplo de 32x32 píxeles en la resolución más baja. Existen una gran cantidad de variaciones de este modelo como pueden ser el *nested U-Net*, algoritmo de segmentación de carreteras basado en U-Net o incluso para imágenes 3D.

Por otro lado, orientado más a modelos entrenados a partir de datos no etiquetados, tenemos el ***Adversarial Autoencoder*** o AAE [43]. Estos autocodificadores combinan AE con una red neuronal adicional llamada discriminador; es decir, aparte de contener un codificador y un decodificador, existe un discriminador. En el contexto de la segmentación de imágenes, el autocodificador se adapta para generar máscaras de segmentación, mientras que el discriminador se entrena para distinguir entre las máscaras de segmentación reales y las generadas por el autocodificador.

El entrenamiento de un AAE se divide en dos partes, el entrenamiento aplicado al autocodificador y el aplicado al discriminador.

1. Entrenamiento del autocodificador. Mientras el discriminador se mantiene constante sin ningún tipo de actualización, el *autoencoder* actualiza los pesos de su red neuronal, entrenándose para generar máscaras de segmentación que engañen al discriminador.

2. Entrenamiento del discriminador. Por otro lado, ahora es el autocodificador quien se mantiene constante y es el discriminador quien actualiza sus pesos, de forma que este es entrenado para mejorar su capacidad para distinguir entre máscaras de segmentación reales y generadas.

Mediante *Transformers*

Los *Transformers* [44] han demostrado ser muy eficientes en tareas de lenguaje natural y es por eso que, se están aplicando en todos los ámbitos. Así, el ámbito de la visión por computador no ha sido menos y se han realizado estudios de algoritmos basados en *Transformers* para resolver tareas como la segmentación de imágenes.

A continuación se listan una serie de algoritmos interesantes basados en *Transformers* con el objetivo de segmentar imágenes.

- ***Vision Transformer*, o ViT [16]**

  Aplica la segmentación semántica gracias al añadido de la autoatención multi-cabeza, el cual conecta en cascada una serie de capas de *Transformers*. Al hablar de *transformers* es necesario adaptar la entrada a ello, es por eso que a la imagen se le aplica una división en diferentes porciones del mismo tamaño (por ejemplo, 16x16 píxeles) los cuales se aplanan y se transforman en vectores 1D y cada uno de ellos se pasa a través de una capa de incorporación lineal para obtener una representación de dimensión fija y, así, poder ser usados como entrada para la arquitectura de un *Transformer*.

  Las incorporaciones de estas porciones o divisiones, las cuales deben de ir con información de posición o localización para no perder las relaciones espaciales entre las divisiones, se pasan a través de una serie de capas de *Transformer*, que constan de bloques de atención multi-cabeza y bloques de *feedforward* permitiendo al modelo capturar relaciones espaciales y contextuales a través de la imagen de entrada.

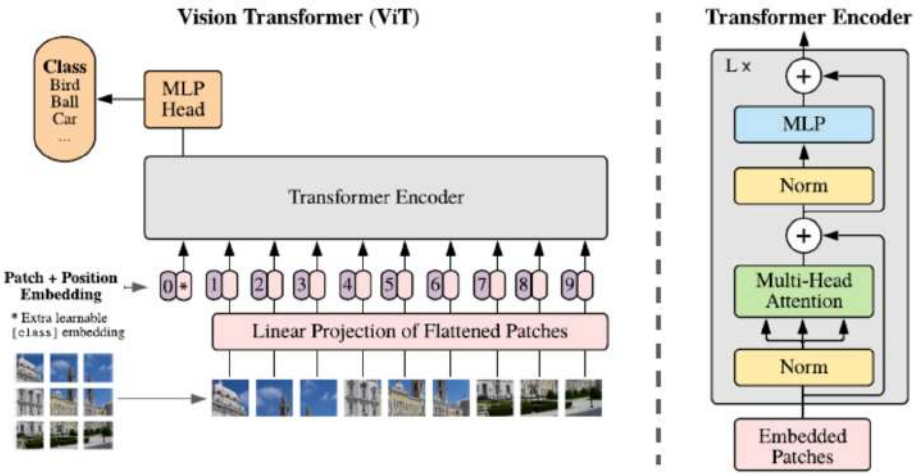


Figura 2.23: Modelo ViT. [16].

La figura 2.23 nos proporciona una visión general de cómo funciona el algoritmo ViT. Si nos fijamos en el inicio, podemos observar como la imagen es dividida por igual en porciones, se añade la posición a cada uno de ellos y son impulsados hacia el codificador *Transformer*.

- ***Segmentation Transformer*, o SETR [17]**

  Como ya se ha explicado en ViT, como un *transformer* necesita una secuencia 1D, se debe de aplicar un proceso que aplane el valor de los píxeles en un vector de una dimensión para secuencializar. Una vez tenemos la secuencia de incrustación en 1D, se emplea un codificador basado en un *transformer* y que aprenda de esta forma representaciones de características; es decir, cada capa del transformador tiene un campo receptivo global.

  En resumen, si nos fijamos en ViT podemos observar que es prácticamente similar a él, pero con un objetivo diferente; es decir, SETR está especializado para la segmentación de imágenes, mientras que ViT aborda problemas de clasificación de imágenes.

  Además, este algoritmo utiliza tanto una codificación como una decodificación en su arquitectura. Esto provoca que tras procesar los parches generados tras dividir la imagen en el codificador, se reconstruye la máscara de segmentación de la imagen en el decodificador. Es decir, el decodificador es el encargado de generar una salida detallada de la segmentación de píxeles para cada clase.

  En la figura 2.24 podemos observar el esquema general del proceso de segmentación de una imagen usando SETR. Primero, *(a)* se observa la división de la imagen en parches, como ya se ha comentado anteriormente para posterior-

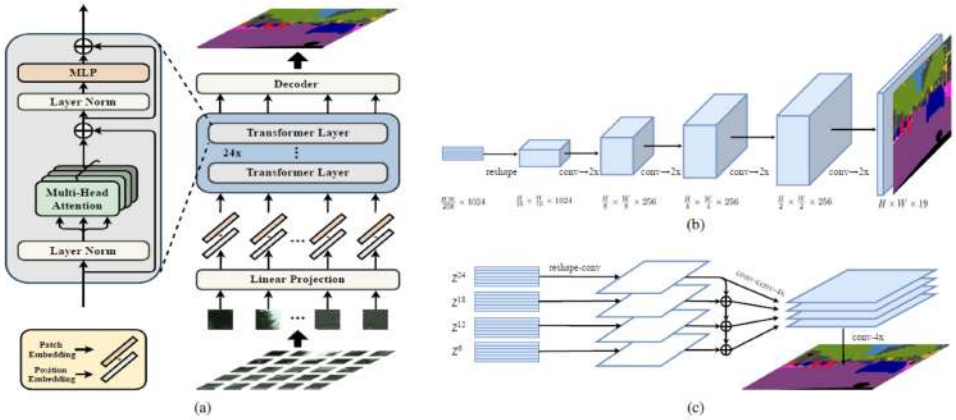


Figura 2.24: Arquitectura SETR. [17].

mente aplicar diferentes diseños de decodificación:*(b)* muestreo ascendente y *(c)* agregación de características multi-nivel.

Por un lado, tenemos el muestreo ascendente, el cual nos proporciona una variante de SETR llamada SETR-PUP. Es decir, en lugar de usar un único paso de muestreo ascendente que puede perjudicar a la segmentación debido a las predicciones ruidosas, se considera una estrategia de muestreo ascendente progresivo, evitando dicho ruido, limitando la ascendencia en x2.

Por otro lado, tenemos la variante SETR-MLA proporcionada por la agregación de características multi-nivel. Esta técnica se centra en combinar la información de las capas intermedias del codificador en lugar de depender únicamente de la salida de la última capa del codificador.

- ***Self-supervised Transformer with Energy-based Graph Optimization*, o STEGO [18]**

  Acabamos esta sección con un enfoque diferente para la solución de la tarea de la segmentación; es decir, mediante un enfoque **no supervisado**. La particularidad de STEGO radica en su enfoque no supervisado, lo que significa que no requiere un conjunto de datos etiquetados previamente para funcionar. En lugar de eso, se basa en la información visual inherente de la imagen para realizar la segmentación.

  STEGO utiliza diferentes formas de comparar imágenes y entrenar el modelo de segmentación. Se basa en una técnica llamada *correspondence losses*, que es una forma de medir lo bien que se relacionan las características similares entre imágenes. Por ejemplo, como en la figura 2.25, si queremos encontrar las características importantes de una imagen a segmentar, STEGO compara la imagen consigo misma, con sus vecinos más cercanos (KNN) y con otras imágenes se-

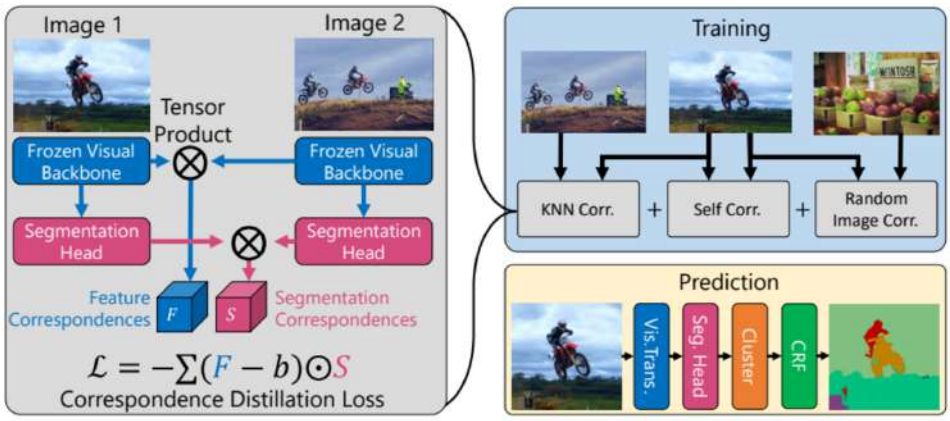


Figura 2.25: Arquitectura STEGO. [18].

leccionadas de forma aleatoria. Una vez han sido comparadas consigo misma y KNN, se obtienen principalmente resultados positivos, encontrando características similares. Sin embargo, cuando se comparan pares de imágenes aleatorias, se obtienen señales negativas, lo que indica que las características son diferentes.

Capítulo 3

OBJETIVOS

En este capítulo se explica en detalle cuál es el objetivo por el cual se realiza este proyecto, es decir, el objetivo general, y en qué objetivos más pequeños ha de dividirse para ir realizando su desarrollo de forma incremental mediante los objetivos específicos.

3.1. Objetivo general

El propósito principal del TFG consiste en estudiar el problema de la segmentación de imágenes y proponer un método basado en aprendizaje profundo para el mismo, el cual se aplicará a imágenes de dispositivos de foto-trampeo.

3.2. Objetivos específicos

Para desarrollar el conjunto de *scripts* y cumplir con el objetivo general, es necesario dividirlo en los siguientes objetivos desarrollados de forma incremental a lo largo del documento.

Investigación previa

Antes de comenzar a experimentar, es necesario profundizar en el desarrollo teórico sobre los conceptos a trabajar. Por ello, se realiza una revisión exhaustiva de la literatura científica relacionada con la segmentación de imágenes y las técnicas de aprendizaje profundo aplicadas a la identificación y separación de animales en imáge-

nes de foto-trampeo. Gracias a dicha investigación, se adquiere el conocimiento teórico necesario para que pueda comprender cómo desarrollar las técnicas necesarias para la segmentación y, por lo tanto, el desarrollo del objetivo general.

Preparación del conjunto de datos

Este objetivo se centra en el análisis del conjunto de imágenes con el que trabajar y preparar un conjunto de datos adecuado para el entrenamiento y evaluación del modelo a usar. Esto implica recopilar imágenes de foto-trampeo que contengan animales y realizar tareas de preprocesamiento, como etiquetado manual y limpieza de datos, para garantizar la calidad y consistencia del conjunto de datos.

Implementación de un modelo DL

Continuamos con el objetivo de diseñar e implementar un modelo de aprendizaje profundo para la segmentación de imágenes de foto-trampeo, incluyendo la selección de una arquitectura de red neuronal adecuada, el entrenamiento del modelo utilizando el conjunto de datos preparado y la optimización de los hiperparámetros del modelo para obtener un rendimiento óptimo.

Evaluación y análisis detallado de los resultados

Por último, tenemos como objetivo final la exposición de los resultados dados por el modelo y el análisis profundo sobre ellos, discutiendo las ventajas, limitaciones y posibles mejoras del modelo propuesto. Para ello, se deben interpretar los resultados de las métricas de evaluación y discutir los desafíos encontrados durante el desarrollo del proyecto, dando una visión sobre la posibilidad de trabajos futuros para la mejora de la segmentación de imágenes con técnicas de DL.

Capítulo 4

MATERIALES Y MÉTODOS

En este capítulo se abordan los aspectos relacionados con los materiales utilizados en el desarrollo del TFG. Específicamente, se centra en el preprocesamiento necesario para el conjunto de datos empleado, así como en los métodos correspondientes para la transformación de los datos y la elección del modelo final.

4.1. Conjunto de datos a usar

Como se ha mencionado en el capítulo 1, el conjunto de datos con el que entrenaremos y evaluaremos nuestro modelo se basa en un conjunto de imágenes de fototrampeo cedidas por WWF/Adena, una organización sin fines de lucro que se dedica a la conservación del medio ambiente y la protección de la biodiversidad en España.

WWF/Adena trabaja en diferentes áreas de conservación, incluyendo la protección de especies en peligro de extinción, por ello la BBDD que nos proporciona para el TFG contiene una serie de características que serán explicadas en detalle en la sección 4.2.

Aunque el conjunto completo de imágenes proporcionadas por WWF/Adena sea un total de, 96 372 imágenes, estas contienen imágenes vacías [23] por lo que no se tratará el conjunto de datos por completo.

4.2. Análisis exploratorio sobre los datos

El conjunto de datos con el que se trabajará está centrado en cuatro especies en concreto: Lynx pardinus (linces), Meles meles (tejones), Oryctolagus cuniculus (conejos) y Vulpex vulpex (zorros).

En la figura 4.1 tenemos una representación de diferentes escenarios que pueden darse en el conjunto de imágenes a segmentar, donde podemos observar que la iluminación y el movimiento de la especie serán detalles clave en el rendimiento del procesamiento del modelo.

Figura 4.1: Comparación de algunos escenarios sobre los que se trabaja. Elaboración propia.

Para entender la estructura del conjunto de datos base a tratar, debemos saber que partimos de una BBDD completa dividida en dos partes:

1. Primera parte: la BBDD se divide en cuatro carpetas que corresponden a cada una de las especies mencionadas anteriormente. A su vez, estas se dividen en más carpetas que suponen cada uno de los nombres asignados a cada uno de los animales.

2. Segunda parte: la BBDD a su vez, por otro lado, está dividida en un grupo de carpetas nombradas como L1, N1, L2, N2, . . . , L9, N9. Estas carpetas nos muestran

la división de las imágenes a partir de un sistema de información espacial, es decir, a partir de diferentes localizaciones geográficas.

La información estructurada en carpetas nos proporciona información sobre el nombre de los animales, el tipo de especie o su localización. En nuestro caso, esta información nos es indiferente, por lo se realizará la limpieza y selección correspondiente para abordar todo el conjunto de datos en una misma carpeta.

Además, existen dos archivos (uno por cada parte de la BBDD) que recogen los metadatos de todas las imágenes de la BBDD. En estos archivos, en formato CSV (*Comma-Separated Values*), podemos encontrar datos interesantes como el atributo `especieMetadata` del archivo CSV2 correspondiente a la segunda parte. En dicho atributo podemos encontrar las diferentes etiquetas de clasificación sobre lo que aparece en imagen: `conejo`, `ave`, `conejo_y_ave`, `humano`, `motor`, `vacío`, etc. Este atributo, por lo tanto, será de gran importancia para la limpieza de imágenes de nuestro conjunto de datos a tratar, ya que no nos interesará trabajar con imágenes vacías donde no aparezcan animales o imágenes donde aparezcan personas, coches u otros objetos que no interesan segmentar en el proyecto.

Es importante destacar que el conjunto de imágenes con animales viene sin etiquetar; es decir, no existe una carpeta añadida que nos proporcione las máscaras de segmentación de los animales en la BBDD. Este factor será de gran importancia para el desarrollo de los siguientes puntos, donde se explicará al detalle la evolución de la preparación de las imágenes para su uso en el modelo (véase la sección 4.6).

4.3. Recogida de datos

En este apartado, nos centramos en la ejecución de los programas desarrollados para la limpieza de imágenes sin animales del conjunto de imágenes en ambas BBDD.

Cada parte de la BBDD tiene un CSV correspondiente y, por lo tanto, el tratamiento de cada parte ha sido diferenciado en dos programas en Python para cada una de las partes de la BBDD.

Los resultados obtenidos de la limpieza de ambas BBDD, quedándonos únicamente con las imágenes que contienen animales, ha supuesto la reducción del conjunto a un total de 46 222 imágenes gracias al análisis exploratorio y la recogida de datos específica sobre ambas BBDD.

4.4. Herramientas y métodos de desarrollo

Las herramientas y metodologías optadas para el desarrollo del modelo deben ser estudiadas para que nos faciliten al máximo nuestro proyecto. Es por ello que, en esta sección, se discuten las posibles opciones para la implementación y la elección de las herramientas más convenientes a partir de unos criterios.

4.4.1. Lenguaje de programación

Para la elección del lenguaje de programación [45] debemos de tener en cuenta una serie de factores clave de análisis previo para llegar a la conclusión final. A día de hoy, el uso de modelos DL está en auge y, por ello, existen una variedad notable de lenguajes de programación para la implementación de proyectos informáticos que, además, tengan la capacidad para el desarrollo de tareas como esta.

En esta sección se profundizará en lenguajes de programación como C++, Java, R y Python. La elección de estos lenguajes para su estudio y, posteriormente, la elección de uno de ellos, estará condicionada por puntos como sus características, las librerías centradas en DL y su popularidad, entre otros.

Es cierto que, independientemente del lenguaje que se use, la diferencia está en la librería y su optimización, ya que será dicha librería quien nos proporcionará los métodos necesarios para la ejecución de nuestro modelo.

Características del lenguaje

C++ es un lenguaje de programación compilado de alto rendimiento, que permite una gran optimización de recursos y una gestión manual de la memoria, como se ha podido estudiar durante la formación en el grado. Como inconveniente, su complejidad y curva de aprendizaje comparado con otros lenguajes es superior al resto. Aun así, las librerías centradas en DL como TensorFlow o PyTorch, entre otras, son escritas en C++ gracias a su alta eficiencia en el manejo de datos y cálculos numéricos, ya que los modelos de aprendizaje profundo requieren un alto grado de procesamiento de datos siendo un factor clave en el éxito de estas librerías.

Java también es un lenguaje compilado como C++ y es popular para su uso en el aprendizaje, pero no ha sido explotado en el ámbito DL al nivel de lenguajes como Python, como veremos más adelante.

R, por otro lado, es un lenguaje de programación interpretado; es decir, aquel en el que el código fuente se ejecuta directamente por un programa interpretador que lo traduce a lenguaje de máquina en tiempo de ejecución. Esto supone que la ejecución del programa sea más lenta y menos eficiente que en el caso de un lenguaje compilado, pero al ser interpretado nos proporciona mayor flexibilidad en la escritura del código. Aun así, R está centrado en el análisis estadístico y cuenta con una capacidad para el análisis exploratorio y visualización de los datos bastante potente.

Python, por último, es también un lenguaje de programación interpretado a alto nivel, de propósito general y orientado a objetos. Este lenguaje se ha convertido en el lenguaje de referencia en el ámbito de la inteligencia artificial y el aprendizaje profundo, debido a su sintaxis clara y legible, su amplia comunidad de desarrolladores y su extenso ecosistema de librerías especializadas.

Librerías centradas en Aprendizaje Profundo

En cada uno de los lenguajes de programación mencionados, existen librerías asociadas de gran importancia.

En C++ nos encontramos una serie de librerías como las que se mencionan a continuación:

1. Caffe [46], biblioteca interesante para el uso del DL y focalizada en la visión por computador. El problema de esta librería es que, para ciertas redes neuronales profundas avanzadas, como podrían ser las estudiadas en la sección 2.4, se encuentra limitada en comparación con bibliotecas más recientes y versátiles como TensorFlow y PyTorch.

2. TensorFlow C++ API y LibTorch (Pytorch para C++) [47]. Ambas bibliotecas resuelven la limitación de la librería Caffe; es decir, podemos encontrar modelos DL avanzados, los cuales serán de interés (FCN o R-CNN). Como contraposición, estas librerías usadas en C++ pueden ser menos intuitivas y completas que su contraparte en Python, ya que la mayoría de la comunidad que usa TensorFlow dan soporte informativo en Python, lo que podría dificultar la implementación de arquitecturas avanzadas en C++.

3. Microsoft Cognitive Toolkit (CNTK) [48], una biblioteca de aprendizaje profundo desarrollada por Microsoft. Aunque esta librería (como TensorFlow y PyTorch) pueda ser usada en Python, nos puede proporcionar también la capacidad de desarrollo de modelos DL avanzados para C++.

En Java podemos destacar librerías como Deeplearning4j (DL4J) [49] para modelos de DL, ya que soporta una gran variedad de arquitecturas de redes neuronales, como por ejemplo las CNN para visión por computador. Además, esta librería tiene como característica peculiar su integración con sistemas de computación distribuida como Apache Spark y Hadoop, facilitando el escalado y el procesamiento distribuido. Además, en Java también es posible utilizar la librería TensorFlow Java.

Por otro lado, en R destacamos la interfaz KerasR [50] de la librería popular de Keras de Python, siendo Keras una librería bastante reconocida a alto nivel para el diseño y entrenamiento de modelos de DL. Es cierto que proporciona una gran variedad de ejemplos y modelos preentrenados, pero, como se explica más adelante, no es un lenguaje de programación con el que esté familiarizado comparado con Python.

Por último, tenemos Python. En este lenguaje de programación podemos encontrar la mayoría de librerías mencionadas anteriormente como son TensorFlow, PyTorch, Keras, entre otras.

Popularidad

Para estudiar la popularidad de los lenguajes de programación mencionados, debemos de tener en cuenta que nos encontramos en el ámbito del uso de modelos DL para la ejecución del TFG. Por ello, no podemos optar por una simple investigación en *Google Trends* sobre cuál es el lenguaje de programación más popular entre los mencionados. Para ello, en la figura 4.2 tenemos una referencia de los puestos de trabajo que surgieron en relación con la ciencia de datos. En este aspecto, Python es un buen candidato para ser elegido si queremos aprender a trabajar con modelos DL.

Es cierto que estas estadísticas quedan un poco atrás respecto al año en el que nos encontramos, sobre todo sabiendo cómo evoluciona nuestra área año tras año. Por ello, también muestro a continuación en la figura 4.3 un índice de los lenguajes de programación más populares en el momento. Como podemos observar, por muy poco tenemos que Python se ha hecho con la popularidad entre los lenguajes de programación en general, aunque está claro que lenguajes consistentes como Java y C se mantienen en el ranking.

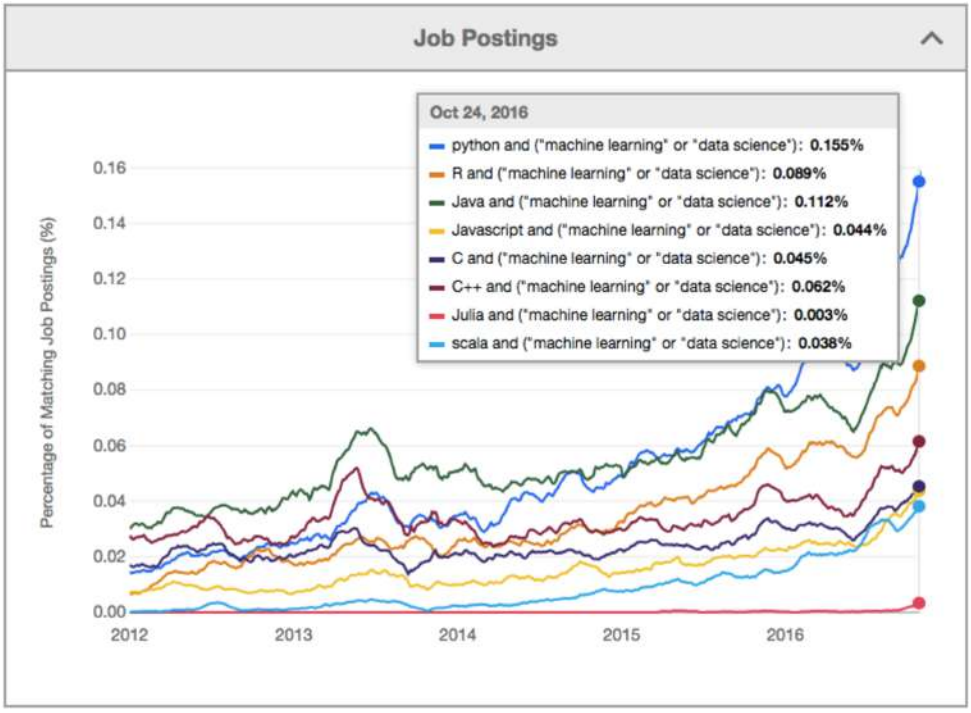


Figura 4.2: Gráfica que representa los puestos de trabajo en relación con los lenguajes de programación y la ciencia de datos. [19].

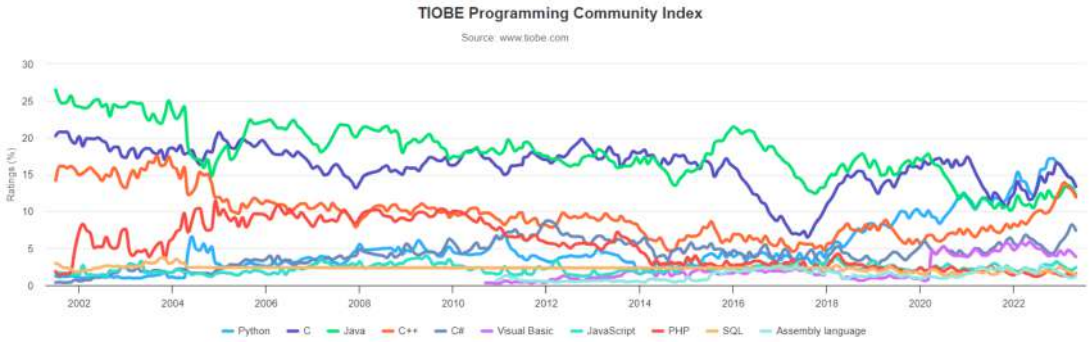


Figura 4.3: Gráfica comparativa de popularidad entre varios lenguajes de programación hasta el momento. Python (azul claro), C (azul oscuro), Java (verde claro). [20].

Experiencia personal con cada lenguaje

Esta sección está conectada a mi desarrollo y formación en los lenguajes de programación desde que inicié el grado hasta el momento. Por ello no consideraré como una opción óptima usar lenguajes de programación que no haya utilizado, como puede ser R, Julia o Scala, entre otros. Es cierto que mi experiencia en lenguajes de programación como C++ o Java es de un grado avanzado, pero el uso intuitivo y claro conseguido en Python y mi curva de aprendizaje respecto a este lenguaje me dan un grado de confianza considerable para el desarrollo del TFG.

Decisión del lenguaje a utilizar

Tras el estudio realizado en los apartados anteriores, donde se ha tenido en cuenta las características propias de cada lenguaje, su popularidad y el ecosistema de librerías centradas en DL, la decisión tomada es el uso del lenguaje de programación Python para el desarrollo de modelos DL para la segmentación de imágenes.

4.4.2. Librerías para modelos de Aprendizaje Profundo

Una vez seleccionado el lenguaje de programación Python, se va a estudiar qué librería se utilizará para el desarrollo del modelo. Para ello se han realizado estudios sobre la capacidad de estas librerías para abordar diferentes características del DL, como el *supported method* que referencia a las funciones o clases que facilitan la implementación de diferentes tipos de capas en una red neuronal o, por otro lado, las funciones de activación que pueden usarse directamente sin tener que crear líneas de código extra.

Si analizamos las estadísticas relacionadas con la popularidad actual (véase la figura 4.4) respecto al número de búsquedas en *Google* de cada una de las librerías mencionadas, podemos observar la igualdad notable entre TensorFlow y PyTorch, despuntando respecto a las demás, Keras y Caffe.

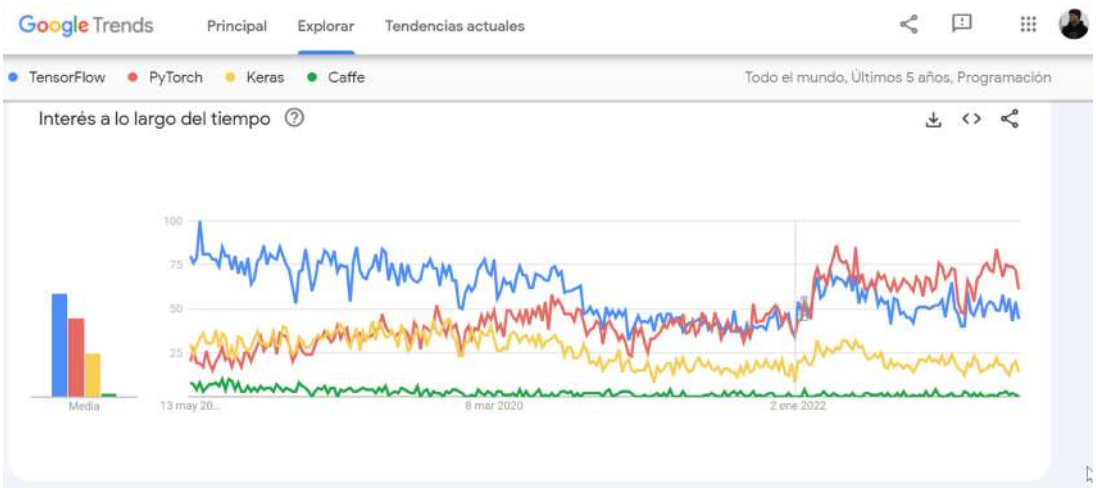


Figura 4.4: Gráfica que representa la popularidad en búsquedas con *Google Trends* para las librerías de TensorFlow (azul), PyTorch (rojo) y Keras (amarillo). Elaboración propia.

Observando la figura 4.5 tenemos marcado en negro las partes que las librerías no pueden dar soporte directo, pero, es cierto que, no significa que no puedan desarro-

llarse dichos puntos mediante líneas de código adicionales, añadiendo complejidad a su uso.

Category	Supported method	TensorFlow	Kearas	PyTorch	Caffe
Layers	Conv1D, Conv2D, Conv3D	+	+	+	+
	ConvTranspose1D, ConvTranspose2D, ConvTranspose3D,	+	+	+	+
	SeparableConv1D, SeparableConv2D	+	+		
	MaxPool1D, MaxPool2D, MaxPool3D	+	+	+	+
	AvgPool1D, AvgPool2D, AvgPool3D	+	+	+	+
	AdaptivePool (all combinations)			+	
	GlobalPool		+		
	Dense	+	+	+	+
	Dropout	+	+	+	+
	Flatten	+	+		+
	Padding	+	+	+	+
	RNN	+	+	+	+
	LSTM	+	+	+	+
	GRU	+	+	+	
	Normalization	+	+	+	+
	Noise		+		
	others	+	+	+	+
Activation functions	ReLu	+	+	+	+
	ReLu6	+		+	
	PReLu		+	+	+
	LeakyReLu		+	+	
	CReLu	+			
	ThresholdedReLu		+	+	
	Elu	+	+	+	+
	Selu	+	+	+	
	Softplus	+	+	+	
	Softsign	+	+	+	
	Bias_add	+			
	Sigmoid	+	+	+	+
	Hard_sigmoid		+		
	Exponential		+		+
	Linear		+		
	Softmax	+	+	+	+
	Tanh	+	+	+	+
	others	+	+	+	+

Figura 4.5: Comparativa entre librerías de Python para DL. [21].

Es importante tener en cuenta ambas figuras mostradas para esta sección porque, aunque sea PyTorch la librería más popular del momento, tenemos que tener en cuenta que con TensorFlow desde la versión 2.0, Keras se ha integrado como la interfaz oficial de alto nivel en TensorFlow, y su desarrollo y mantenimiento continúa como parte del proyecto TensorFlow. Aun así, es cierto que la popularidad de PyTorch nos proporciona mayor facilidad a la hora de encontrar los algoritmos de DL necesarios para el proyecto, dando una ventaja sobre el resto.

4.5. Modelos a valorar

En esta sección nos centramos en el desarrollo y descripción del trabajo realizado con diferentes algoritmos para nuestro objetivo, la segmentación de imágenes con animales. Para ello, comenzamos hablando sobre la ineficacia de los resultados obtenidos mediante algoritmos de VC (mencionados en la subsección 2.1.2), continuamos con la búsqueda de algoritmos DL y, por último, el desarrollo del modelo elegido.

4.5.1. ¿Por qué aplicar una técnica de Aprendizaje Profundo?

Tras el análisis de algunas técnicas de VC y ML para la segmentación de imágenes, se muestra a continuación una serie de resultados obtenidos para demostrar la necesidad de usar técnicas DL en este tipo de tareas si queremos buscar una mayor eficacia.

Antes de mostrar algunos resultados conseguidos con estas técnicas, se eligen cuatro imágenes del conjunto de datos con animales (véase la figura 4.6). Estas imágenes tienen características diferentes, como por ejemplo que sea diurna o no, la similitud del animal respecto al escenario, entre otras.

Figura 4.6: Cuatro imágenes aleatorias del conjunto de datos a trabajar. Elaboración propia.

Como podemos observar, la dificultad de segmentar los elementos del entorno en cada imagen, obteniendo un resultado favorable sobre la segmentación del animal, será una tarea complicada para las técnicas de VC debido a la homogeneidad de colores entre los elementos presentes.

Entre los algoritmos probados mediante sus correspondientes *scripts*, se encuentran los mencionados a continuación.

- **Basados en Contornos Activos**

  Tras la aplicación de esta técnica de VC mediante el uso del algoritmo proporcionado en la librería cv2, podemos observar en la figura 4.7 que consigue acercarse hacia la segmentación del animal respecto al entorno, pero su eficacia respecto a lo que buscamos sigue estando lejos de lo deseado.

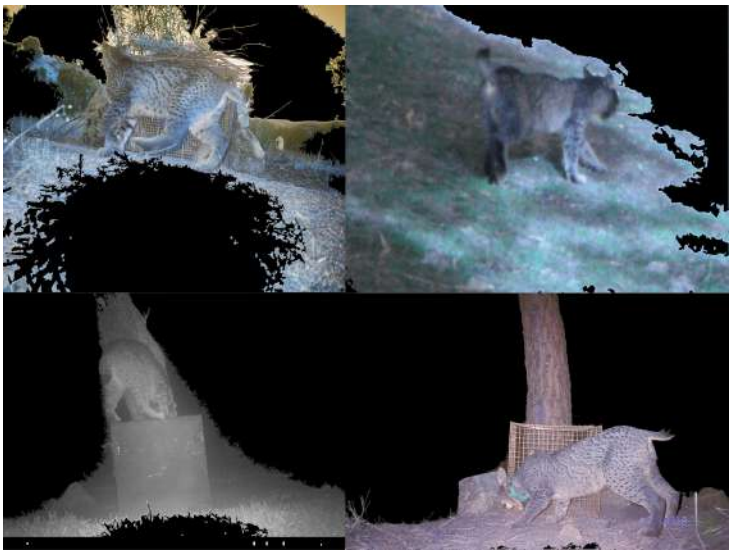


Figura 4.7: Segmentación de cuatro imágenes aleatorias mediante Contornos Activos. Elaboración propia.

- **Basados en Dividir y Fusionar**

  Por otro lado, tenemos el algoritmo de Dividir y Fusionar aplicado mediante la técnica clásica del *quadtree*. La agrupación por colores a partir de píxeles parecidos puede ser eficaz en algunas ocasiones, aunque este algoritmo sea más costoso que el resto de los mencionados en esta sección.

  El problema de usar este algoritmo en escenarios de imágenes de foto-trampeo es que la similitud del animal con el escenario provoca que a veces se mezclen píxeles del animal con el entorno, como ocurre en la figura 4.8, en concreto en la primera imagen de la segunda fila. Por lo tanto, este algoritmo para el conjunto de datos a manejar es ineficaz en la mayoría de las situaciones.

Figura 4.8: Segmentación de cuatro imágenes aleatorias mediante Dividir y Fusionar. Elaboración propia.

- **Basados en Agrupamientos o *Clustering***

  Por último, se ha aplicado el algoritmo K-Means de ML para el agrupamiento basado en centroides. Este algoritmo proporcionado por la librería cv2 tiene una complejidad añadida para el volumen de nuestro conjunto de imágenes; es decir, parametrizar el valor del número de *clusters* (véase la figura 4.9).

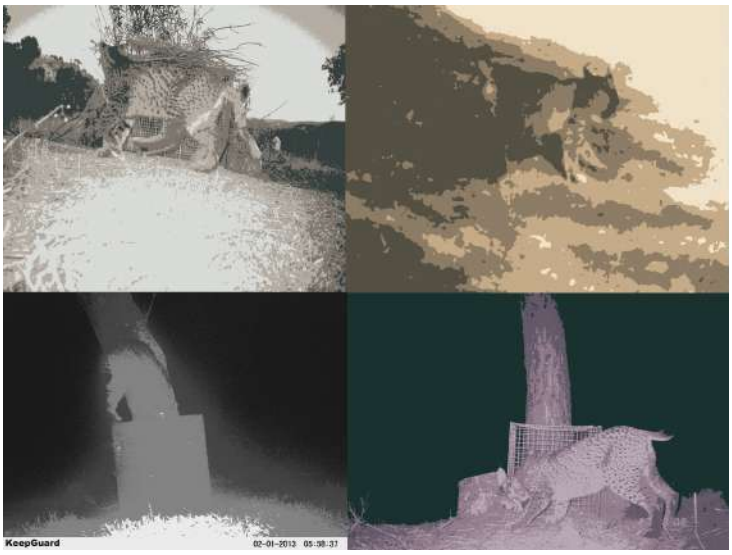


Figura 4.9: Segmentación de cuatro imágenes aleatorias mediante K-Means (k = 4). Elaboración propia.

En consecuencia, tras examinar los ejemplos obtenidos mediante estas técnicas, se puede inferir que el resultado para la composición de imágenes con animales puede resultar ineficaz. Aquí es donde entra la utilidad del uso de algoritmos de DL para

obtener un mayor rendimiento a la hora de segmentar los animales para diferentes escenarios.

4.5.2. Técnicas de Aprendizaje Profundo a valorar

Nos dirigimos nuevamente a nuestro conjunto de datos pre-procesado, el cual se distingue por la presencia de uno o más animales en cada una de las imágenes, donde se observa que existe uniformidad en los escenarios en relación con el animal presente. Esto implica que la detección y, por consiguiente, la segmentación del animal sea complicada para tareas como ML. Además, debemos destacar una vez más que **el conjunto de datos a trabajar no proporciona un etiquetado previo** de la segmentación de los animales en las imágenes, siendo el etiquetado manual de un experto el paso más costoso de todo el proceso de recopilación y preparación de datos.

Para solucionarlo, el siguiente paso es entender qué técnicas son las apropiadas a valorar a partir de los enfoques de aprendizaje existentes, explicados en la subsección 2.2.1.

Enfoque No Supervisado

Este enfoque nos ayuda a segmentar nuestro conjunto de datos sin necesidad de etiquetar manualmente una parte del conjunto de datos, pero aplicará una segmentación global del entorno de las imágenes; es decir, intentará segmentar cajas, fauna, cielo, animales, entre otros.

Como ejemplo, tenemos el algoritmo STEGO (véase la subsección 2.4.2). Sabemos que este algoritmo basado en *Transformers* nos proporciona la posibilidad de obtener resultados sin un entrenamiento previo, obteniendo resultados de segmentación como el que aparece en la figura 4.10.

Si nos fijamos en la diferencia de la segmentación con respecto a los algoritmos ML, esta es considerable a la hora de obtener una mejoría en la localización y segmentación del animal. Para entender la figura 4.10, la imagen de la izquierda muestra de nuevo la representación base para recordar de qué figura partimos, luego aparece la imagen centrada donde se muestra el resultado del algoritmo STEGO, por último, se utiliza el algoritmo de refinamiento *Dense Conditional Random Field* (CRF) para ajustar las predicciones.

STEGO nos proporciona una solución a la tarea de segmentar sin necesidad de aplicar una preparación de los datos para entrenar al modelo y posteriormente obte-

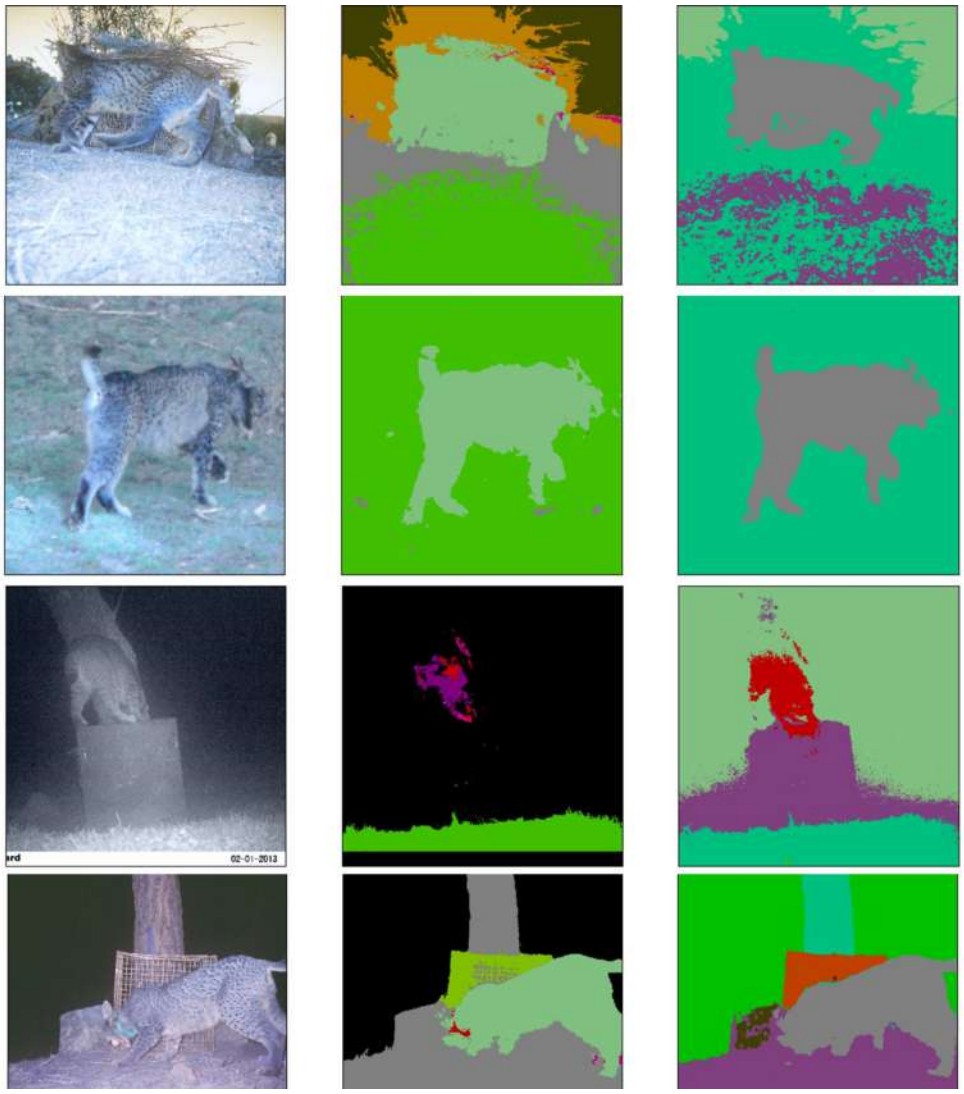


Figura 4.10: Segmentación de cuatro imágenes aleatorias mediante STEGO. Elaboración propia.

ner unos resultados de segmentación coherentes comparada con los algoritmos de VC vistos en la sección previa. Por otro lado, como es lógico debido a la dificultad del conjunto de datos a trabajar, los resultados de un modelo no supervisado no nos acerca a la segmentación esperada para el conjunto a trabajar. Por ejemplo, la imagen mostrada en la figura 4.11, donde aparece el resultado de una imagen del conjunto de datos, nos deja mucho que desear para obtener una segmentación coherente del animal respecto de lo demás. Si nos fijamos, no consigue reconocer el objeto que nos interesa; es decir, no reconoce en la imagen al lince debido a su gran similitud con respecto a la vegetación de la imagen.

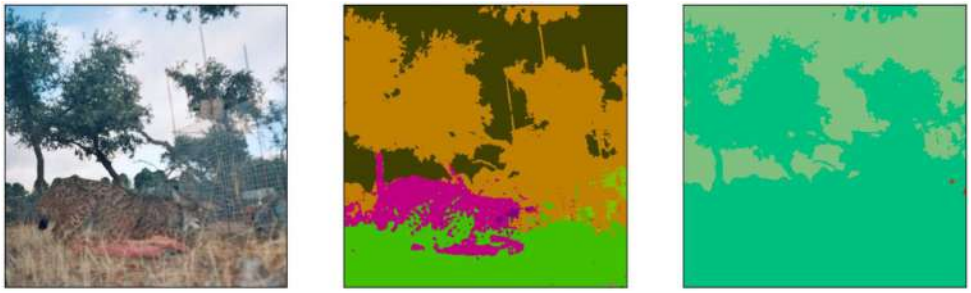


Figura 4.11: Segmentación de una imagen del conjunto *test* con STEGO. Elaboración propia.

Es cierto que nos supone una ventaja a nivel de coste de recolección y preparación de datos, pero debido a la eficiencia que suelen dar los algoritmos no supervisados en tareas complejas, se ha valorado buscar un enfoque supervisado. Aunque nos suponga un trabajo añadido de etiquetado manual, nos proporcionará un resultado que se adecue más al contenido que nosotros buscamos, la segmentación del animal.

Enfoque Supervisado

Este enfoque supone un trabajo adicional debido al etiquetado manual previo para la posibilidad del entrenamiento del modelo, el cual se explica cómo se ha llevado a cabo en la subsección 4.6.2. Por lo tanto, la elección del modelo es consecuente del esfuerzo de adaptación del conjunto de datos a los requisitos necesarios para que este pueda ser entrenado por el conjunto de datos de animales con el que venimos trabajando desde el inicio.

Hemos estudiado una gran variedad de algoritmos a utilizar para la segmentación de imágenes mediante algoritmos supervisados. En particular, hemos investigado la posibilidad de emplear el algoritmo FCN y el algoritmo de Mask R-CNN.

Por un lado, las FCN han sido utilizadas ampliamente en tareas de reconocimiento de objetos y localización de regiones de interés en imágenes gracias a su capacidad de procesar imágenes completas y producir mapas de segmentación a nivel de píxeles. Por otro lado, las Mask R-CNN han sido explotadas en esta área debido a su gran eficiencia al combinar la precisión de detección de objetos con la capacidad de generar máscaras de segmentación precisas para cada objeto detectado.

Como conclusión, la propuesta de facilitar el aprendizaje y uso del algoritmo Mask R-CNN [51], junto con sus variantes a evaluar, nos ha llevado a elegir el modelo Mask R-CNN.

4.6. Modelo final

En esta sección se explica detalladamente cómo ha sido configurado el modelo final y qué programas han sido necesarios para poder utilizar el modelo de la forma más óptima posible.

4.6.1. Planteamiento KDD

En esta sección se explica un esquema general de cómo estará construida cada una de las siguientes secciones, basadas en la filosofía de la extracción de conocimiento KDD (*Knowledge Discovery in Databases*) [22]. Para enumerar cómo se lleva a cabo cada una de las fases, se muestra en la figura 4.12 el proceso completo desde que tenemos en nuestras manos la BBDD a tratar hasta obtener el conocimiento esperado.

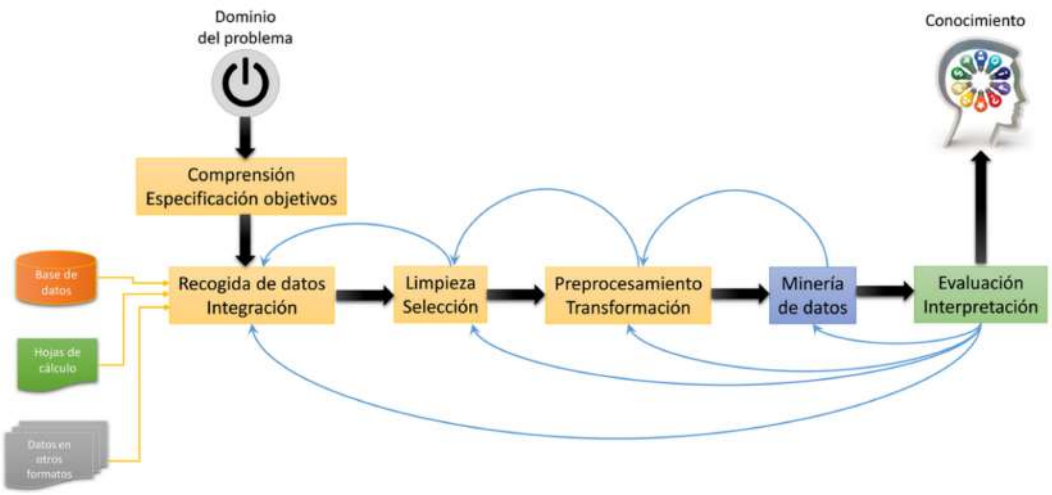


Figura 4.12: Proceso de extracción de conocimiento, KDD. [22].

1. **Recogida de datos e integración**. Esta parte ha sido desarrollada en la sección 4.3 donde se aplican los correspondientes programas para obtener las imágenes que contenían únicamente animales, reduciendo la BBDD al total de 46 222 imágenes.

2. **Limpieza y selección**. En la sección 4.6.2 se explica cuál es el uso real del conjunto de imágenes con animales, ya que el modelo a entrenar es un modelo supervisado y, por lo tanto, requiere de imágenes previamente etiquetadas para poder ser entrenado.

3. **Preprocesamiento y transformación**. En esta sección se explican las variantes escogidas para el preprocesamiento de las imágenes y la necesidad de la transformación de nuestros datos para poner en marcha el uso del modelo Mask R-CNN. Entre las variantes existe el uso exclusivo del contorno de la segmentación de los animales o el uso de toda la segmentación del animal, añadiendo una carga de puntos de segmentación alta.
4. **Minería de datos**. Aquí desarrollamos la configuración de nuestro modelo, donde explicamos las posibles vertientes, las cuales nos proporcionarán diferentes resultados a ser interpretados en la siguiente sección.
5. **Evaluación e interpretación**. Por último tenemos la evaluación y su posterior análisis, los cuales serán desarrollados en profundidad en el capítulo 5.

4.6.2. Limpieza y selección

Como ha sido mencionado anteriormente, nuestra BBDD no nos proporciona las etiquetas de segmentación necesarias para poder entrenar un modelo supervisado. Por esta razón, debido al tiempo y costo que implica etiquetar manualmente la segmentación (un proceso más costoso que el etiquetado para otras tareas), hemos tenido que reducir el conjunto de imágenes a un total de 1 998 instancias.

Etiquetado manual

Para el etiquetado, ha sido necesario el uso del programa Photoshop. Este programa nos proporciona una herramienta de selección y segmentación útil para agilizar el proceso de etiquetado de las imágenes. Como podemos observar en la figura 4.13, se muestra consecutivamente el proceso desde que se introduce una imagen hasta que conseguimos obtener la máscara que nos proporciona la misma imagen con su etiquetado de segmentación.

Aunque en la figura 4.13 parezca un proceso sencillo, dependiendo de la imagen a ser etiquetada, esta herramienta se vuelve algo tediosa, ya que hay que ir seleccionando aquellas partes del animal que no han sido reconocidas hasta obtener una segmentación coherente del animal. También es cierto que, el modelo aprenderá a partir de mis segmentaciones, las cuales no mantienen una precisión perfecta.

Eliminar los metadatos de la imagen

En algunos casos, las imágenes contienen metadatos en forma de una barra inferior, lo cual puede afectar negativamente el rendimiento y la precisión del modelo DL.

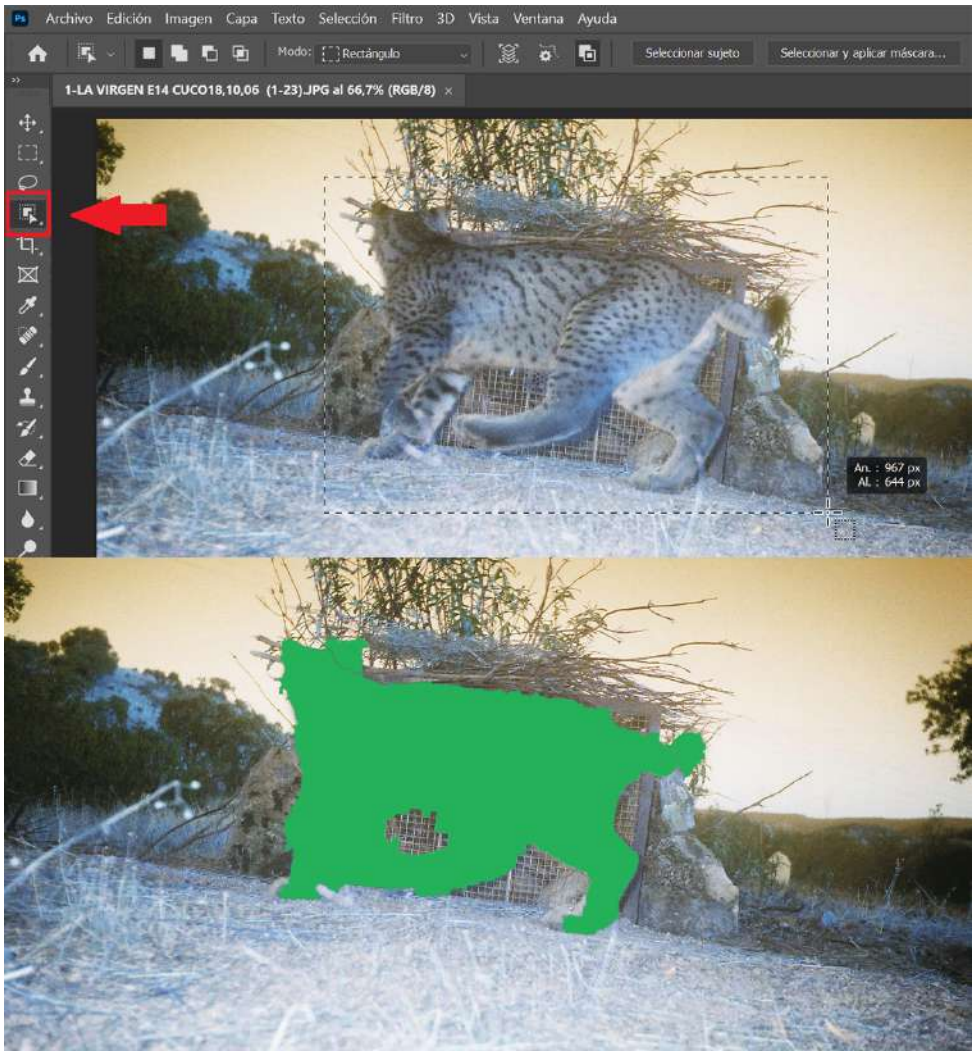


Figura 4.13: Etiquetado manual de la BBDD. Elaboración propia.

Es por ello que se propone un método de recorte de imágenes para eliminar dichos metadatos y mejorar la eficacia de los modelos de segmentación.

En la figura 4.14 se muestra qué parte ha de ser recortada y cuál es el resultado final de ello.

Figura 4.14: Primer recorte de las imágenes para mejorar el rendimiento del modelo. Elaboración propia.

La heterogeneidad de nuestra BBDD nos lleva a tratar imágenes donde existen diferentes dimensiones para los metadatos o incluso no aparecen directamente en las imágenes. Además, en ocasiones es considerado más oportuno un recorte aplicado a la parte inferior derecha, como el de la figura 4.15.

Figura 4.15: Segundo recorte de las imágenes para mejorar el rendimiento del modelo. Elaboración propia.

Para solucionarlo, se ha partido de un primer recorte general donde se recorta los primeros 225 píxeles de alto de la parte inferior a todas las imágenes y se ha analizado a qué imágenes ha sido necesario aplicar un segundo recorte, inferior o de la parte derecha de la imagen. Con estos recortes conseguimos eliminar por completo sus metadatos y obtener un resultado limpio en todas las imágenes.

No obstante, existen otras imágenes que contienen metadatos que no deben ser eliminados (véase la figura 4.16), dado que ello podría ocasionar el recorte del animal y, por consiguiente, influir negativamente en el rendimiento del modelo una vez más. Es por ello que se ha optado por aceptar dichos metadatos para el entrenamiento y evaluación de nuestro modelo.

Como hemos podido observar, el recorte en algunas ocasiones de los metadatos provoca la eliminación parcial o completa del animal en la imagen. En estos casos, se ha decidido eliminar estas imágenes del conjunto de datos a trabajar teniendo que tratar con **1 841 imágenes**.

Figura 4.16: Muestra de metadatos en una imagen. Elaboración propia.

Partición del conjunto de datos

Por último, aplicamos la separación de nuestro conjunto de imágenes en conjuntos de entrenamiento, validación y evaluación. La división de un conjunto de datos de esta forma es una práctica común en el ML conocida como *Hold Out* [29], la cual se realiza con el objetivo de evaluar y medir el rendimiento de un modelo de manera más objetiva y realista a partir del conjunto total de imágenes etiquetadas que hemos conseguido hasta el momento.

- **Entrenamiento, 70 %**

  Esta carpeta contiene los datos que se utilizarán para entrenar el modelo; es decir, para obtener la capacidad de reconocer patrones y realizar predicciones basadas en estos datos. El conjunto de entrenamiento es la parte más grande del conjunto de datos y se utiliza para ajustar los parámetros del modelo, por lo que se ha decidido dar un 70 % del conjunto de imágenes totales a trabajar.

- **Validación, 15 %**

  Esta carpeta se utiliza para evaluar el rendimiento del modelo y ajustar sus hiperparámetros; es decir, ajustar las configuraciones del modelo que no se aprenden durante el entrenamiento como puede llegar a ser el ajuste de la tasa de aprendizaje.

- **Evaluación, 15 %**

  Esta carpeta se reserva para evaluar el rendimiento final del modelo una vez que se ha ajustado y validado, el cual es utilizado para simular cómo el modelo se comportaría con datos nuevos y no vistos previamente.

Por lo tanto, tras las dificultades presentadas anteriormente y la capacidad de etiquetado manual limitada, el número de imágenes de cada conjunto que será utilizado se especifica en la tabla 4.1.

Conjunto	Total de imágenes
Entrenamiento	1 290
Validación	277
Test	274

Tabla. 4.1: Número de imágenes para entrenamiento, validación y test.

4.6.3. Preprocesamiento y transformación

Antes de configurar el modelo, para optimizar sus parámetros, debe entenderse qué necesita para poder ser entrenado a partir del conjunto de datos.

El modelo Mask R-CNN [51] necesita ser entrenado a partir de un conjunto de datos que mantenga las mismas características que el conjunto de datos COCO (*Common Objects in Contex*). Por lo tanto, necesitamos adaptar nuestro conjunto de datos de imágenes a un conjunto de imágenes en formato COCO, el cual será transformado gracias al etiquetado manual hecho en la sección anterior y a la creación del JSON (*JavaScript Object Notation*) correspondiente para ello.

Primero, debemos recopilar los puntos de interés, o también considerados como los puntos de segmentación (x,y), de nuestro conjunto de datos etiquetados. Para ello, estudiamos en profundidad cómo se obtienen los puntos de interés en el archivo JSON, conocido como *balloon dataset*. Este conjunto de datos contiene una serie de objetos, cada uno representando una imagen del conjunto de datos.

En la figura 4.17 se observa el resultado de la máscara obtenida a partir de una imagen del conjunto, una vez recogidos los puntos de interés a segmentar. Cada imagen puede tener varias regiones de interés, las cuales proporcionan información sobre la forma de la región y las coordenadas de los puntos que definen la forma. Las

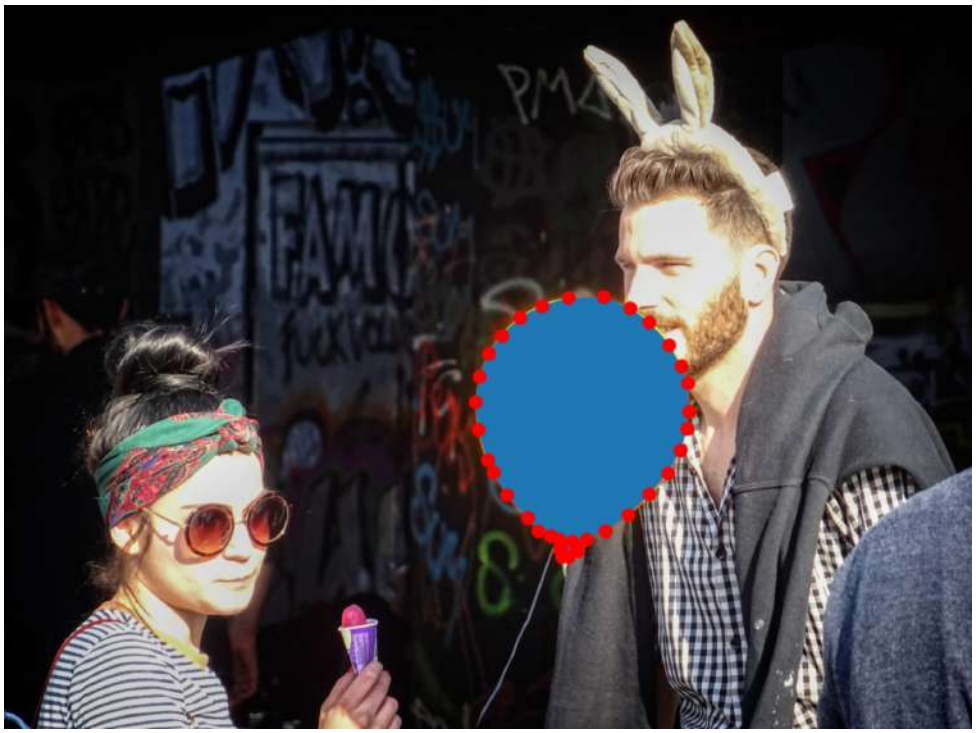


Figura 4.17: Máscara de imagen aleatoria del *balloon dataset* obtenida a partir de los puntos de interés. Elaboración propia.

regiones de interés están representadas como polígonos, donde los puntos (coordenadas) definen los bordes del polígono mediante dos listas separadas, `all_points_x` y `all_points_y`, que contienen las coordenadas x e y respectivamente.

A partir de esta información, se planteó una primera solución básica, la cual recopilaba cada uno de los píxeles que estuviesen contenidos dentro de la segmentación a partir de un rango de color específico (el color del etiquetado). Basándonos en estos requisitos, se almacenaba tanto la coordenada x como la coordenada y, siendo un problema de ineficiencia a nivel de almacenamiento (4 GB en un único archivo JSON). Como podemos comprobar, esta primera solución era inviable debido al exceso de memoria almacenado para un mismo archivo.

Por ello, se plantea a continuación una serie de opciones de almacenamiento que nos acompañarán hasta la evaluación del modelo.

- **Almacenar únicamente el contorno de la segmentación**

  A partir de algoritmos ya implementados, como el algoritmo `findContours` de la librería de OpenCV (cv2). Este algoritmo nos devuelve únicamente el contorno de la segmentación, dándonos una solución de almacenamiento al problema anterior y similar a la solución de almacenamiento del *balloon dataset*.

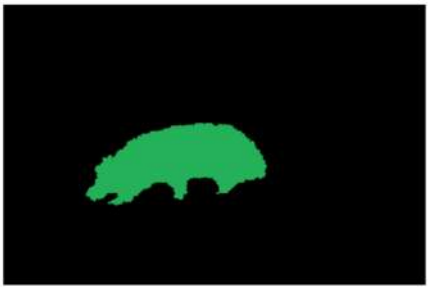
Por otro lado, esta reducción de hasta un 99 % de los puntos de interés puede ser eficaz respecto al almacenamiento del archivo JSON anterior. La problemática que supone esta reducción es que no le estamos dando al modelo la posibilidad de entrenarse y evaluarse con puntos de interés ciertamente válidos, ya que buscará el aprendizaje del contorno únicamente. Aun así, se ha valorado su posibilidad de estudio gracias a los resultados de segmentación obtenidos por ello, los cuales serán mostrados posteriormente.

- **Almacenar puntos de interés a partir de una tolerancia, tolerancia = 10**

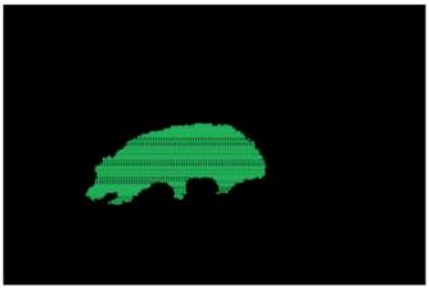
  Se aplica la misma metodología que al principio, pero en este caso se aplica una tolerancia, la cual permite reducir la cantidad de puntos encontrados al eliminar aquellos que están muy cerca entre sí. Esto puede ser útil en aplicaciones donde se desea tener una representación más concisa de los puntos de interés, como es en nuestro caso.

  Debemos de buscar qué valor de tolerancia es el más apropiado a usar, por ello se ha hecho las pruebas necesarias para comprobar qué máscara resultante aparece usando diferentes umbrales de tolerancia.

  Si elegimos un umbral de tolerancia bajo, como puede ser tolerancia de 2 o 3 píxeles, las máscaras resultantes aparecen completas tal y como las originales (véase la figura 4.18a). Por otro lado, si elegimos un umbral alto, como puede ser de 15 píxeles, los resultados se verán modificados considerablemente (véase la figura 4.18b).



(a) Tolerancia igual a 2.



(b) Tolerancia igual a 15.

Figura 4.18: Imagen del conjunto de datos donde se ha aplicado la tolerancia igual a 2 y tolerancia igual a 15. Elaboración propia.

El problema que tiene utilizar una tolerancia baja como el ejemplo anterior es que la reducción del archivo no llega a ser aceptable. Por ello, se ha buscado el umbral de tolerancia que no afecte a la apariencia de la máscara y nos ayude a reducir al máximo nuestro archivo.

De esta forma, y con un ajuste de tolerancia óptimo igual a 10, conseguimos reducir el archivo JSON hasta un 10 % de su capacidad actual; es decir, de 4 GB previos hasta un tamaño final de 360 MB, optimizando el tratamiento del archivo para nuestro entrenamiento.

Una vez aplicado el filtro de almacenamiento de los puntos de interés correspondiente, el cual nos devuelve las coordenadas x e y de segmentación apropiadas, el siguiente paso es crear el JSON mediante los diccionarios oportunos. Para la creación del JSON, se realizan los siguientes pasos:

1. Renombrar las imágenes con caracteres especiales. Antes de comenzar con la creación de un diccionario que almacene los metadatos de cada imagen, debemos comprobar que la imagen no tenga caracteres latinos, como por ejemplos las tildes o la ñ. En caso de que así sea, se aplica la modificación correspondiente para ser renombrado en la misma ruta.
2. Diccionario general a partir del diccionario de regiones. Tras obtener las coordenadas x e y, se crea un diccionario de regiones que contiene información sobre los puntos de interés de la imagen. Además, cada imagen debe almacenar en el diccionario general una serie de metadatos, entre los cuales tenemos al diccionario de regiones creado previamente. Cada uno de estos diccionarios se irán almacenando en una lista para posteriormente obtener el JSON.
3. Creación del JSON con reajuste. Por último, tras obtener una lista de diccionarios correspondientes a cada una de las imágenes, se aplica un filtrado de limpieza del primer y último carácter del archivo para obtener el JSON necesario para el modelo Mask R-CNN [51].

4.6.4. Arquitectura y parámetros del modelo

En esta sección se muestran las arquitecturas e hiperparámetros esenciales del modelo final utilizando Mask R-CNN, y su variante, el modelo Mask Scoring R-CNN, siendo esta última explicada en la sección 4.6.4.

Mask R-CNN

Para explicar la arquitectura y los parámetros del modelo Mask R-CNN se ha desglosado en la figura 4.19 la arquitectura completa, desde que tenemos las imágenes

pre-procesadas (eliminación de ruido, etiquetados, etc.) hasta que obtenemos la salida de dichas imágenes segmentadas por el modelo.

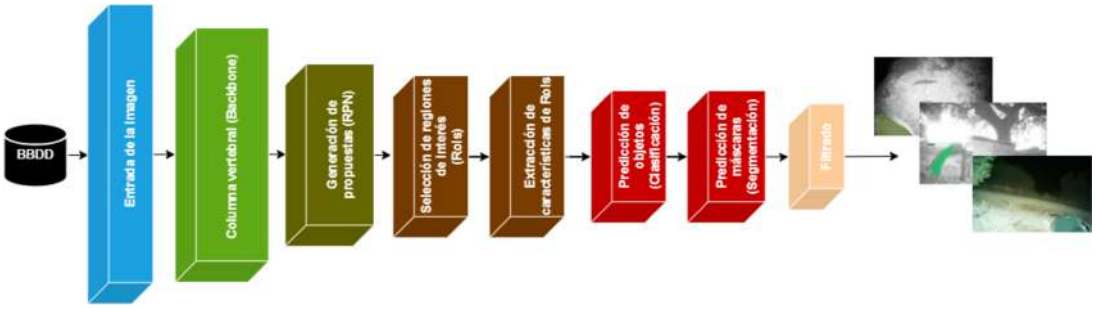


Figura 4.19: Arquitectura desglosada del Mask R-CNN. Elaboración propia.

Como podemos observar en la figura 4.19, tenemos una serie de cajas diferenciadas por colores, explicadas al detalle a continuación.

- **Entrada de la imagen**

  Representado con color azul, donde la imagen de entrada se pasa a través de un preprocesamiento que incluye la normalización de los valores de píxeles para que tengan una media de cero y una desviación estándar de uno.

- **Columna vertebral o *Backbone***

  Representado con verde claro, la imagen pre-procesada se pasa a través de la columna vertebral interpretada por una red ResNet de 50 capas preentrenada. ResNet extrae características de interés de la imagen como son los bordes, texturas y formas en la imagen.

- **Generación de propuestas o RPN**

  Representado con un tono verde oscuro, se utiliza el RPN para que las características del *backbone* generen propuestas iniciales de regiones que podrían contener objetos.

- **Selección y extracción de características RoIs**

  Representan dos cajas marrones secuenciales, donde las propuestas generadas por el RPN son filtradas utilizando un umbral de confianza; es decir, las RoIs (*Region of Interest*) seleccionadas son las que tienen una alta probabilidad de contener un objeto. Por último, para cada RoI seleccionada, se extraen características específicas utilizando una capa de RoI Align (véase la subsección 2.4.1 para más información).

- **Predicción de objetos y máscaras**

  Representan dos cajas rojas secuenciales, donde las características RoIs son clasificadas a partir de la clase `animal`, dando como resultado una caja delimitadora que marca el objeto. Además, con estas características se obtiene la máscara a partir del componente `FCNMaskHead`, especializado para la extracción de la máscara en el modelo.

- **Filtrado y refinamiento**

  Por último, representado en un tono naranja claro, tenemos un filtrado y refinamiento de las predicciones de cajas delimitadoras y máscaras mediante el uso de umbrales y técnicas de NMS (*Non-Maximum Suppression*). Este NMS se utiliza para reducir las predicciones redundantes y seleccionar las más relevantes y precisas de la imagen.

Una vez explicada la arquitectura de nuestro modelo, introducimos la explicación de los parámetros optimizados para Mask R-CNN.

- **Máximo número de épocas:** `max_epochs = 300`

  Este hiperparámetro depende de varios factores, como la complejidad del modelo, el tamaño del conjunto de datos, la velocidad de convergencia, la capacidad de cómputo disponible y el tiempo de entrenamiento que se puede permitir. En nuestro caso, se aplica un número máximo de 300 épocas; es decir, se recorre 300 veces el dataset hasta finalizar el entrenamiento de nuestro modelo, siendo elegido este número con el fin de encontrar el umbral apropiado de épocas mostradas en las figuras de la sección 4.6.5.

- **Optimizador:** `type = SDG` (*Stochastic Gradient Descent*)

  El optimizador trata de modificar los parámetros del modelo para buscar la mayor eficiencia posible. En el caso del SDG, busca la optimización a partir de los recursos y la función de pérdida.

- **Ratio de aprendizaje:** `lr = 0,0025`

  El ratio de aprendizaje es un hiperparámetro del optimizador y marca la velocidad de convergencia, la calidad del resultado y la estabilidad del proceso de entrenamiento. En nuestro caso, sabiendo que el 0,001 puede ser el valor por defecto de un ratio de aprendizaje, se ha decidido aumentar un poco el valor para obtener una mayor velocidad de convergencia que nos ayudará a dirigir en las etapas iniciales hacia una región de mejores soluciones.

- **Número de pasos de validación:** `val_interval = 3`

  Por último, este parámetro controla con qué frecuencia se realiza la validación durante el proceso de entrenamiento. De esta forma, se establece un valor de 3 épocas por validación simplemente para poder monitorizar con mayor frecuencia cómo actúa nuestro modelo según evoluciona.

Mask Scoring R-CNN

Por otro lado, tenemos una versión modificada de la segmentación denominada Mask Scoring R-CNN. Aunque partimos de la misma arquitectura que Mask R-CNN (véase la figura 4.19), la diferencia de este modelo radica en la forma en la que se evalúan las máscaras de segmentación predichas. Es decir, en este modelo se aplica un añadido en la evaluación de la segmentación, el IoU (*Intersection over Union*), el cual mide la superposición entre una máscara predicha y su máscara verdadera correspondiente (véase la sección 5.1 de métricas de evaluación).

Por último, existe una asignación de puntuación o *Scoring*. Basándose en la evaluación de IoU, las máscaras que tienen una alta superposición con las máscaras reales obtienen puntuaciones más altas, mientras que las máscaras menos precisas obtienen puntuaciones más bajas. De esta forma, durante la técnica de NMS o en la etapa de filtrado, las puntuaciones de calidad de las máscaras influyen en qué máscaras se mantienen y cuáles se descartan.

Este modelo mantiene la misma configuración de parámetros que Mask R-CNN, para comprobar su eficacia en la mejora de la segmentación.

4.6.5. Versiones finales del modelo

En esta sección se presentan las versiones apropiadas de nuestro modelo, junto con las consideraciones expuestas en secciones anteriores, con el fin de alcanzar el objetivo principal mediante un conjunto de ocho variantes distintas (véase la tabla 4.2). Cada una de estas variantes se describe en detalle a continuación:

- **Mask R-CNN y Mask Scoring R-CNN**

  La selección inicial en la estructura arquitectónica de nuestro modelo es de suma importancia, por lo que se valoran estas dos versiones clave para nuestra arquitectura.

- **Delimitación de contornos o tolerancia**

  En la última instancia, se abordará la obtención de puntos de interés tanto en los contornos como en el cuerpo completo del animal (aplicando la tolerancia previamente explicada). De esta manera, será posible evaluar la viabilidad de realizar un etiquetado manual del animal en su totalidad, con los costos asociados, o si es suficiente con los contornos únicamente.

  Este planteamiento permitirá minimizar los gastos relacionados con el etiquetado manual por parte de expertos, ya que en el proceso de Extracción de Conocimiento (KDD), la mayor parte del tiempo transcurrido desde la incorporación de los datos en la BBDD hasta la obtención de conocimiento radica en la fase de etiquetado por parte del experto.

- **Consideración de aplicación de la transferencia de aprendizaje**

  Como se explicó en la sección 2.3.5, una de las características clave del DL es la transferencia del aprendizaje; es decir, utilizar los pesos preentrenados de una red neuronal profunda, previamente entrenada en el conjunto de datos COCO. Este enfoque se emplea para inicializar los pesos de la red neuronal profunda, un enfoque que puede tener implicaciones beneficiosas en la mejora de las métricas de segmentación seleccionadas.

#	Arquitectura de modelo	Contornos o tolerancia	Transferencia de aprendizaje
1	Mask R-CNN	Contornos	Transferencia
2	Mask R-CNN	Contornos	Sin transferencia
3	Mask R-CNN	Tolerancia	Transferencia
4	Mask R-CNN	Tolerancia	Sin transferencia
5	Mask Scoring R-CNN	Contornos	Transferencia
6	Mask Scoring R-CNN	Contornos	Sin transferencia
7	Mask Scoring R-CNN	Tolerancia	Transferencia
8	Mask Scoring R-CNN	Tolerancia	Sin transferencia

Tabla. 4.2: Versiones finales del modelo DL elegido.

Para cada una de estas versiones se ha realizado el estudio de la función de pérdida que nos interesa; es decir, la función de pérdida de la máscara. Tanto para Mask R-CNN como Mask Scoring se utiliza `CrossEntropyLoss`, la cual utiliza una función de entropía cruzada para medir la diferencia entre las máscaras predichas y las máscaras

de referencia de los datos de entrenamiento. Con ello, podemos ajustar el hiperparámetro del número de épocas de entrenamiento, reduciendo el costo computacional para futuras pruebas.

Versión 1

Comenzamos con la primera versión de nuestro modelo final, la cual se caracteriza por el uso de Mask R-CNN con transferencia de aprendizaje y los puntos de interés del etiquetado del experto marcados únicamente por los contornos de los animales.

La representación gráfica de la evolución de la función de pérdida durante el proceso de entrenamiento de la red neuronal se encuentra ilustrada en la figura 4.20. En dicha figura, se puede observar la velocidad de convergencia hasta la época 100, así como la posterior estabilización de la función de pérdida a partir de la época 200.

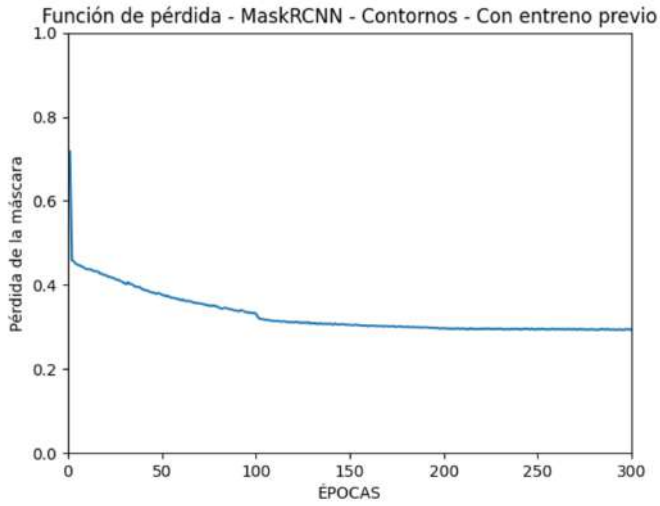


Figura 4.20: Representación de la función de pérdida de la máscara. Versión 1 del modelo final. Elaboración propia.

Versión 2

Continuamos con Mask R-CNN y la recogida de puntos de interés del contorno, cambiando a un estudio sin transferencia de aprendizaje previo; es decir, en esta versión se entrena a partir de los pesos de la red iniciales.

Si nos fijamos en la figura 4.21, comparada con la versión 1, obtenemos una representación de la función similar. Por ello, no existe una diferencia notable por ahora

entre el uso de la transferencia de aprendizaje o no para este estudio. Aunque si observamos las primeras épocas, estas parten de un valor de pérdida de la máscara mucho mayor que en la versión 1; es decir, los modelos parecen converger hacia un óptimo local similar entre el uso de la transferencia de aprendizaje y el entrenamiento desde cero.

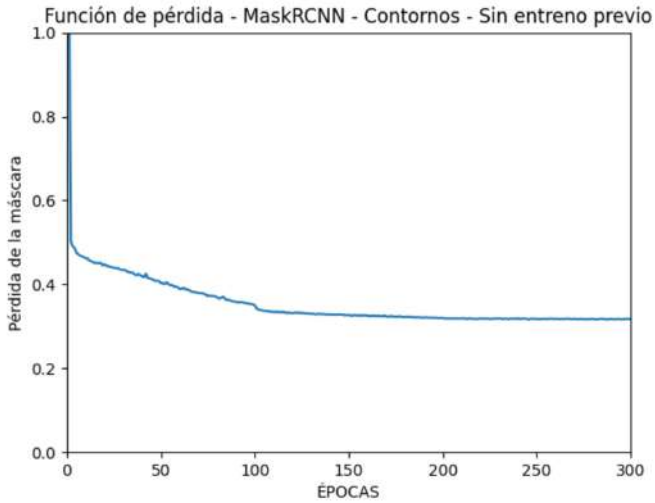


Figura 4.21: Representación de la función de pérdida de la máscara. Versión 2 del modelo final. Elaboración propia.

Versión 3

Partiendo del Mask R-CNN, nos centramos ahora en el etiquetado completo del animal a segmentar y el uso de la transferencia de aprendizaje.

La representación de cómo actúa la función de pérdida en este caso aparece en la figura 4.22. Podemos apreciar que la velocidad de convergencia hasta la época 100 es algo más pronunciada que en las versiones anteriores, donde se había entrenado únicamente a partir del contorno. Por otro lado, como hemos utilizado la transferencia de aprendizaje, el inicio de la gráfica comienza en un valor inferior a $0{,}6$; es decir, manteniendo el comportamiento de la versión 1. Por último, podemos analizar que se obtiene la estabilidad de la función a partir de la época 150, quedando completamente estable en la época 200.

En esta versión podemos destacar la importancia marcada por el etiquetado previo de nuestras imágenes. Supone un mayor costo realizar el etiquetado completo del animal, pero la diferencia respecto a la función de pérdida es notable.

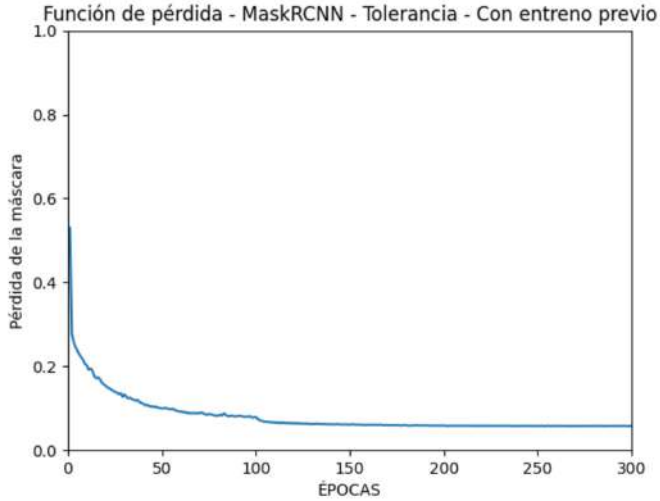


Figura 4.22: Representación de la función de pérdida de la máscara. Versión 3 del modelo final. Elaboración propia.

Versión 4

Como última versión utilizando Mask R-CNN y aunque sigamos con un etiquetado completo del animal, nos diferenciamos con respecto a la versión anterior en la ausencia de la transferencia de aprendizaje.

En la figura 4.23 podemos destacar la rapidez de convergencia similar a la versión anterior, caracterizada por el uso del etiquetado completo del animal. Además, como ya se ha explicado previamente, vuelve a converger hacia el mismo óptimo local que con el uso de la transferencia de aprendizaje.

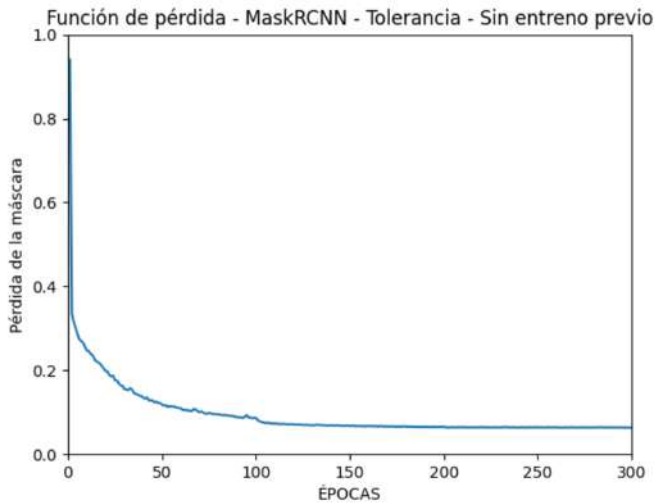
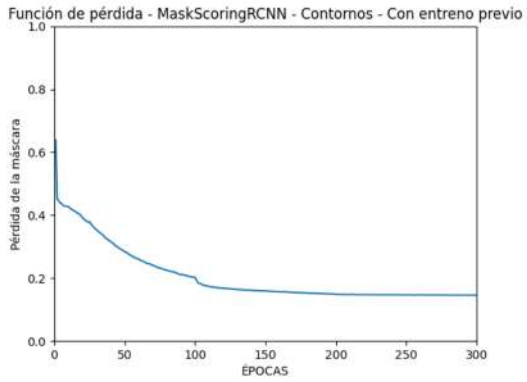


Figura 4.23: Representación de la función de pérdida de la máscara. Versión 4 del modelo final. Elaboración propia.

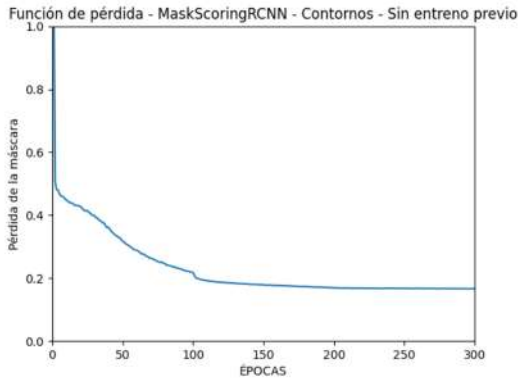
Versiones 5 y 6

A partir de la versión 5 hasta la última, la evolución de la función de pérdida estará marcada por Mask Scoring R-CNN, siguiendo el mismo orden de características elegidas en las cuatro versiones anteriores. Por lo tanto, en estas versiones se utilizan Mask Scoring R-CNN, con etiquetado del contorno y con la transferencia de aprendizaje (versión 5) o sin ella (versión 6).

Al fijarnos en la función de pérdida representada en la figura 4.24a, podemos considerar que el comportamiento de la función respecto a la forma en la que esta converge es similar a la versión 1, diferenciándose en la mejora del óptimo local alcanzado; es decir, la función comienza a estabilizarse a partir de las 100 épocas en un valor de pérdida de la máscara de 0,1 aproximadamente. Por otro lado, con las mismas características, pero utilizando el modelo Mask R-CNN, este valor no llega a alcanzar un valor inferior a 0,3 (figura 4.20 de la versión 1).



(a) Versión 5 del modelo final.



(b) Versión 6 del modelo final.

Figura 4.24: Representación de la función de pérdida de la máscara de la versión 5 y versión 6, respectivamente. Elaboración propia.

Seguimos con el uso de Mask Scoring R-CNN con etiquetado del contorno y sin transferencia de aprendizaje en la versión 6. Una vez más, como ocurría entre las versiones 1 y 2, la figura 4.24b se diferencia únicamente en los valores iniciales, pero su óptimo local converge de forma similar a usar la transferencia de aprendizaje.

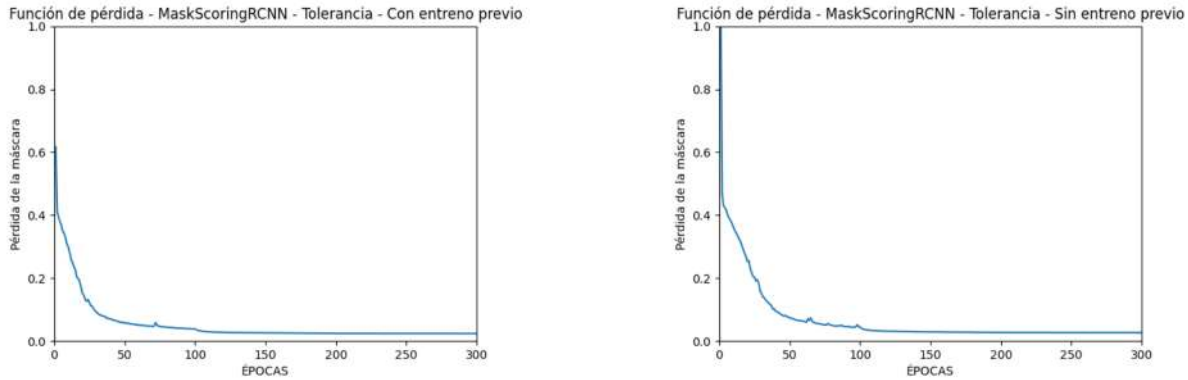
Versiones 7 y 8

Tras la mejora en la función de pérdida utilizando un etiquetado completo del animal en las versiones 3 y 4, volvemos a dicho uso del etiquetado con tolerancia. Es decir,

las versiones 7 y 8 representan el uso de Mask Scoring R-CNN con tolerancia y la transferencia de aprendizaje (versión 7) o sin dicha transferencia (versión 8).

La evolución de las funciones de pérdida de estas versiones, representadas en la figura 4.25, son consideradas como las versiones que mejor han conseguido adaptar el modelo al conjunto de entrenamiento.

La velocidad de convergencia y la reducción del valor de pérdida de la máscara a valores casi nulos nos proporcionan una sensación positiva.



(a) Versión 7 del modelo final.

(b) Versión 8 del modelo final.

Figura 4.25: Representación de la función de pérdida de la máscara de la versión 7 y versión 8, respectivamente. Elaboración propia.

Aunque consigamos visualizar los valores en la función de pérdida, debemos comprobar cómo estas versiones actúan frente a un conjunto de datos de *test*; es decir, qué capacidad de generalización tienen.

Por lo tanto, tras el análisis de la evolución de la función de pérdida en cada una de las versiones, podemos deducir que conseguimos la estabilidad en la función en la época 200, el cual será marcado como el parámetro final para todas las versiones.

Capítulo 5

RESULTADOS

En este capítulo se muestran los resultados obtenidos de las ocho versiones mostradas en la sección anterior. Para ello, se explicarán las métricas de evaluación elegidas para valorar la fiabilidad del modelo respecto a las variantes, dando paso a las tablas con los resultados.

5.1. Métricas de evaluación

Las métricas de evaluación [52] nos ayudan a entender cómo de bien funciona o no un modelo y, dependiendo de la tarea a abordar, será más eficiente usar una u otra. En nuestro caso, las métricas principales escogidas para la segmentación han sido MAE, MSE, SSIM, F1-Score y Boundary F1-Score.

5.1.1. Métricas de evaluación de error y similitud estructural

Comenzamos con métricas de evaluación del error en la segmentación de nuestro modelo respecto a la máscara de referencia.

Por un lado, MAE (*Mean Absolute Error*) [52] calcula la diferencia absoluta entre las predicciones y los valores reales para cada muestra; es decir, en el caso de la segmentación, se evalúa la diferencia entre las máscaras de segmentación predichas y las máscaras de referencia para cada píxel.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{5.1}$$

Si nos fijamos en la fórmula 5.1.1 tenemos n como el número total de imágenes a evaluar, y_i representado como los valores de los píxeles en la máscara de referencia e $\hat{y}_i$ como la predicción del modelo para la muestra.

Por otro lado, MSE (*Mean Squared Error*) [52] mide la magnitud promedio de los errores al cuadrado entre las predicciones de un modelo y los valores reales; es decir, MSE comparado con MAE penaliza en mayor medida los errores cometidos por el modelo.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.2}$$

Por último, SSIM (*Structural Similarity Index*) [53] es utilizada para medir la similitud estructural entre dos imagenes; es decir, el SSIM trata la percepción visual y la similitud en la estructura, el contraste y la luminancia de las imágenes.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{5.3}$$

Como podemos observar en la fórmula 5.1.1, tenemos la comparación entre la imagen x y la imagen y. En dicha comparación tenemos las medias (μ), las varianzas (σ) y las covarianzas (σ^2) de ambas imágenes, las cuales son utilizadas para comparar la luminancia, el contraste y la estructura.

Tras explicar cada una de las métricas anteriores, se muestra la tabla 5.1 con los resultados aparentes para cada una de las ocho versiones finales del modelo.

#	MAE	MSE	SSIM
1	6,28	1,10	0,96
2	7,18	1,14	0,96
3	13,02	1,23	0,94
4	13,09	1,24	0,94
5	6,74	1,13	0,96
6	6,96	1,12	0,96
7	8,60	1,17	0,95
8	8,68	1,16	0,95

Tabla. 5.1: Evaluación final del error y similitud estructural para las ocho versiones finales.

5.1.2. Métricas de evaluación con *accuracy* y *recall*

Para entender cómo vamos a calcular las métricas que involucran al *accuracy* y el *recall*, es necesario introducir una serie de conceptos previos.

- **Verdaderos Positivos (TP)**

  Es el número de píxeles correctamente clasificados como pertenecientes a la clase objetivo.

- **Verdaderos Negativos (TN)**

  Es el número de píxeles correctamente clasificados como no pertenecientes a la clase objetivo.

- **Falsos Positivos (FP)**

  Es el número de píxeles incorrectamente clasificados como pertenecientes a la clase objetivo cuando en realidad no lo son. Por ejemplo, segmentar fuera de la superficie del animal.

- **Falsos negativos (FN)**

  Es el número de píxeles que deberían haber sido clasificados como pertenecientes a la clase objetivo, pero fueron incorrectamente etiquetados o no detectados por el modelo.

Los conceptos introducidos nos ayudan a comprender cómo calcular la *accuracy* y el *recall* de una segmentación en una imagen a partir de las siguientes fórmulas:

$$accuracy = \frac{TP}{TP + FP} \tag{5.4}$$

$$recall = \frac{TP}{TP + FN} \tag{5.5}$$

La *accuracy* 5.1.2 es la proporción de verdaderos positivos (TP) sobre el total de predicciones positivas (TP + FP), dando así entender qué tan bien evita segmentar píxeles que no forman parte del animal a segmentar. Por otro lado, el *recall* 5.1.2 nos ayuda a averiguar qué tan bueno es el modelo para capturar todos los píxeles donde aparece el animal; es decir, tiene como objetivo evaluar qué píxeles de la máscara del animal han sido segmentados y cuáles no.

Por último, para obtener una visión conjunta de ambas métricas, se utilizan métricas que combinan ambas según el interés. Por ejemplo, existen métricas que dan más peso a la *accuracy* que al *recall* (F0.5), o viceversa (F2). En nuestro caso, nos interesa una evaluación equitativa sobre ambas, por lo que se le aplica el mismo peso a la *accuracy* y el *recall*, obteniendo la métrica F1.

$$F1 = \frac{2 \cdot (accuracy \cdot recall)}{accuracy + recall} \tag{5.6}$$

Además, para estudiar la calidad de segmentación sobre el contorno de nuestro modelo, se aplica la métrica de *Boundary F1-Score*; es decir, esta métrica está diseñada para evaluar qué tan bien un algoritmo de segmentación puede detectar y coincidir con los bordes reales en la imagen.

La diferencia respecto a la fórmula de F1 5.1.2 radica en que únicamente evalúa TP, FP y FN del contorno de la imagen predicha y la imagen de referencia.

Por lo tanto, gracias a la métrica de F1 de la librería sklearn, se ilustran a continuación los resultados en la tabla 5.2 de cada una de las versiones del modelo.

#	*F1 Score*	*Boundary F1 Score*
1	72,88	8,12
2	69,47	7,62
3	27,19	2,44
4	26,84	2,20
5	70,56	7,35
6	70,49	6,94
7	53,44	3,12
8	53,40	3,08

Tabla. 5.2: Evaluación final del *accuracy* y *recall* para las ocho versiones finales del modelo. Representación porcentual de los resultados.

5.1.3. Análisis de resultados finales

Tras los resultados obtenidos en las métricas de las secciones anteriores, vamos a analizar algunos aspectos de interés.

Como podemos observar, la mejor versión obtenida ha sido la **versión 1** (véase la sección 4.6.5), la cual se caracteriza por el uso de Mask R-CNN, entrenado a partir de los contornos de los animales y con transferencia de aprendizaje. Para confirmar la relación de los resultados de las métricas con la función de pérdida se muestra en la tabla 5.3 un resumen de estos resultados para un entrenamiento de 100 épocas, 200 épocas (actual) y 300 épocas. En la tabla podemos observar que los resultados se mantienen desde las 200 épocas hasta las 300 épocas, dando la razón al uso de 200 épocas para todas nuestras evaluaciones.

Épocas	MAE	MSE	SSIM	*F1 Score*	*Boundary F1 Score*
100	7,12	1,12	0,96	68,67	6,49
200	**6,28**	**1,10**	**0,96**	**72,88**	**8,12**
300	6,29	1,10	0,96	72,82	8,06

Tabla. 5.3: Resultados por épocas de la versión 1 del modelo final.

Es cierto que, en los resultados obtenidos para la versión 1, existen disparidades entre las segmentaciones generadas por el modelo y las máscaras originales de los animales. Este caso se ilustra desde la figura 5.1 donde alcanzamos un total de $90{,}21\,\%$, hasta situaciones más adversas, como las observadas en la figura 5.2, en la que la tasa de predicción desciende al $19{,}74\,\%$.

Figura 5.1: Representación de ejemplo de segmentación buena por el modelo de la versión 1. Elaboración propia.

Figura 5.2: Representación de ejemplo de segmentación mala por el modelo de la versión 1. Elaboración propia.

Como hemos podido observar en las comparaciones anteriores, no siempre obtenemos unos resultados buenos de segmentación para todo el conjunto de datos *test*. Por ello, se ilustra en la siguiente gráfica de puntos (figura 5.3) cómo se distribuyen los resultados de las segmentaciones según el conjunto de evaluación.

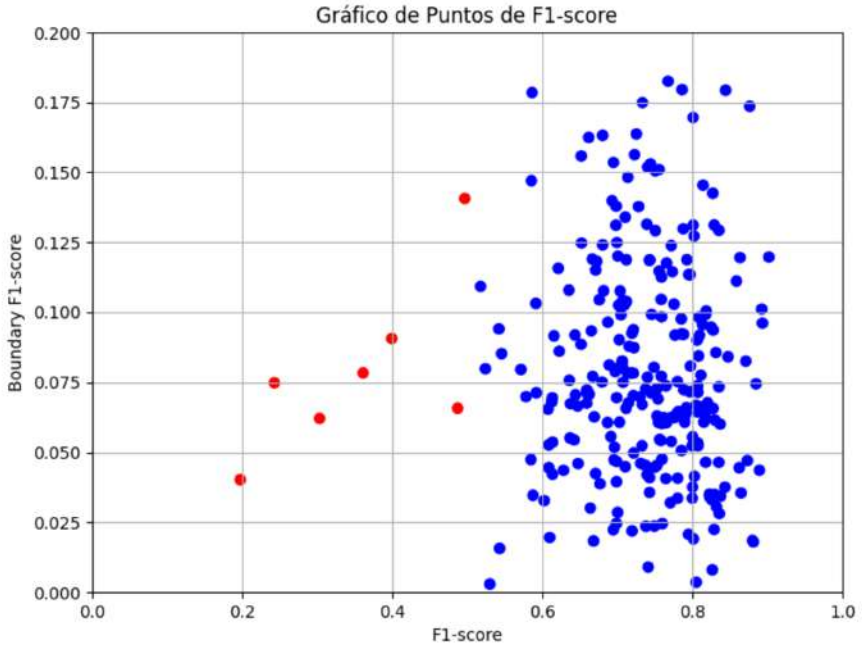


Figura 5.3: Resultados de la versión 1 representados en dos dimensiones (*F1 Score*, *Boundary F1 Score*). Elaboración propia.

En la figura 5.3 podemos diferenciar las imágenes segmentadas con un *F1 Score* inferior (rojo) y superior (azul) al $50\,\%$. Aunque existan imágenes con mala segmentación por parte del modelo, como en la figura 5.2, la representación gráfica recalca que la cantidad de tales instancias se encuentra en un margen aceptable.

Capítulo 6

CONCLUSIONES

Llegamos al final de nuestro proyecto, donde se concluye lo aprendido después de explorar y analizar nuestros resultados. Es decir, se hará un resumen sobre el trabajo realizado, los conocimientos adquiridos y las posibles mejoras a realizar en un futuro.

6.1. Estudio realizado

En esta sección hacemos un breve recorrido sobre nuestro trabajo. El reto de segmentar imágenes de foto-trampeo comenzó con la investigación en profundidad de los modelos DL para la segmentación de imágenes, estudiando diferentes posibilidades como las mostradas en la sección 2.2.3. Una vez elegido el modelo más apropiado a ser usado, continuamos con el preprocesamiento de los datos, desde el etiquetado manual de las imágenes hasta la transformación apropiada para el modelo final. Por último, se valoraron los resultados de cada una de las versiones de nuestro modelo DL a partir de las métricas del capítulo 5.

Gracias al capítulo de resultados hemos extraído el conocimiento necesario para concluir en los siguientes puntos.

- **Etiquetado manual únicamente con contornos**

  La tabla 5.2 ilustra que en las versiones donde el modelo se entrena y evalúa solamente utilizando la segmentación del contorno del animal, los resultados han sido notablemente superiores en comparación con aquellos obtenidos al entrenar con todos los puntos de interés del animal.

Esta observación tiene una ventaja importante, ya que nos permitirá etiquetar las imágenes futuras de manera mucho más rápida, lo que a su vez reducirá uno de los tiempos más significativos en el proceso KDD. En otras palabras, se verá beneficiado el proceso de etiquetado manual llevado a cabo por un experto.

- **La transferencia de aprendizaje**

  Si comparamos las versiones similares, pero que difieren en usar o no la transferencia de aprendizaje, podemos demostrar la importancia de su uso, mejorando los resultados obtenidos en las métricas.

  Al fijarnos en la versión 2 (sin transferencia de aprendizaje), obtenemos un MAE y *F1 Score* de $7{,}17$ y $69{,}86\,\%$ respectivamente. Por otro lado, la versión 1 (con transferencia de aprendizaje) obtiene una mejora significativa con valores de MAE y *F1 Score* de $6{,}28$ y $72{,}90\,\%$ respectivamente.

- **Complejidad del conjunto de datos utilizado**

  Es cierto que el conjunto de datos proporcionado por WWF ha representado un desafío considerable en lo que respecta a la segmentación de animales. Es decir, la tarea de segmentar animales con un plumaje de camuflaje en medio de la vegetación ha limitado las capacidades de nuestro modelo para llevar a cabo dicha segmentación. Aun así, los resultados de la versión 1 de nuestro modelo se han considerado aceptables.

  La representación gráfica de puntos en la figura 5.3 ilustra en rojo las imágenes que han sido segmentadas con una precisión inferior al cincuenta por ciento. Esta representación nos da la oportunidad de investigar las características compartidas entre estas imágenes y explorar posibles patrones subyacentes.

  Además, debemos de hablar del *Hold-out*, el cual fue optado como técnica para la evaluación de los modelos debido a restricciones temporales y recursos disponibles. El *Hold-out* puede proporcionar una evaluación rápida y sencilla, pero el enfoque de validación cruzada (*Cross-validation*) se considera generalmente más realista y robusto en la evaluación de modelos. Esto se debe a que los resultados obtenidos a partir del conjunto *test* pueden estar sesgados a un enfoque optimista o pesimista, dependiendo de la distribución aleatoria realizada para cada uno de los conjuntos de entrenamiento, validación y *test*.

6.2. Conclusiones

Si volvemos al capítulo 3, donde se mencionan los objetivos, podemos concluir el documento afirmando que se han cumplido los propósitos esperados. Es cierto que no se ha obtenido unos resultados perfectos, pero la dificultad del conjunto a trabajar con la tarea de segmentarlos nos da pie a expresar nuestra satisfacción con el trabajo realizado.

La utilidad de solventar el problema de la segmentación de imágenes de dispositivos de foto-trampeo y proponer un método basado en aprendizaje profundo para el mismo, se ha considerado como un avance significativo en el campo de la investigación de la conservación de la fauna y la ecología.

En nuestro caso, la segmentación del animal puede suponer la conservación de especies en extinción, comprobando su estado y comportamiento de forma automática entre miles y miles de imágenes.

Además, este documento demuestra la viabilidad del uso del algoritmo Mask R-CNN para solventar tareas de segmentación ciertamente complejas.

6.3. Conocimiento adquirido

Partiendo de los conocimientos que he adquirido durante mi grado, el TFG ha presentado un desafío significativo. La exploración de algoritmos de aprendizaje profundo para tareas de segmentación me ha brindado la oportunidad de mejorar y aplicar mis habilidades en el desarrollo de soluciones tecnológicas avanzadas. Además, me ha proporcionado una valiosa oportunidad para expandir y aplicar mis conocimientos en el campo de la ciencia de datos, visión por computadora y aprendizaje automático avanzado. A lo largo de este proceso, he adquirido una serie de habilidades y experiencias significativas que considero fundamentales para mi desarrollo profesional.

La comprensión y aplicación de modelos de aprendizaje profundo, junto con la adaptación del conjunto de datos para su puesta en marcha, han sido un proceso beneficioso en mi formación. Además, no solo se limita a la solución de un problema técnico, sino al aprendizaje de cómo debe ser estructurado un proyecto de tal importancia.

A continuación, listo algunos de los conocimientos adquiridos que considero relevantes a comentar.

- **Resolución de un problema de Ciencia de Datos real**

  Este proyecto representó una oportunidad única para abordar un problema de ciencia de datos real y relevante en el contexto de la conservación de la fauna. He aprendido a identificar y definir problemas específicos, recopilar y explorar datos, y aplicar metodologías de análisis rigurosas para obtener soluciones prácticas.

- **Visión por Computador**

  Me introduje en el campo de la CV, donde me enfrenté al desafío de desarrollar algoritmos capaces de identificar y segmentar animales en imágenes de foto-trampeo. Esta experiencia me permitió comprender en detalle los conceptos clave de detección y segmentación de objetos en imágenes.

- **Herramientas de Aprendizaje Automático**

  Utilicé una variedad de herramientas de ML, incluyendo bibliotecas como PyTorch, para implementar y entrenar modelos de aprendizaje profundo. Esto me proporcionó una comprensión exhaustiva de cómo trabajar con estas bibliotecas para desarrollar soluciones efectivas.

- **Modelos avanzados de Aprendizaje Profundo**

  Exploré modelos avanzados de aprendizaje profundo, como redes neuronales convolucionales profundas. Esta experiencia me permitió apreciar la importancia de la elección de modelos adecuados para tareas específicas y cómo ajustar sus hiperparámetros para lograr un rendimiento óptimo.

Por último, aunque exista una sección para ello, me gustaría terminar mencionando el gran apoyo de mis tutores. Juntos, con su conocimiento y dedicación, hemos conseguido el objetivo.

6.4. Trabajo futuro

Siempre es bueno valorar cómo mejorar nuestro proyecto, es por eso que se lista a continuación una serie de posibles mejoras a realizar en un trabajo futuro:

- **Aumentar el etiquetado del conjunto de datos**

  Esta primera mejora es considerable siempre y cuando se dispongan del tiempo y los recursos necesarios para ello. Es cierto que tratar con 1 845 imágenes etiquetadas ha sido suficiente para la realización de este trabajo, pero el etiquetado de más imágenes proporcionadas por WWF nos proporcionaría una mayor precisión y cobertura en nuestros resultados finales.

- **Validación cruzada**

  Aunque esta técnica suponga un aumento del coste temporal, como ya se ha explicado en la sección 6.1, el uso de la validación cruzada puede darnos un resultado que se ajuste más a la realidad.

- **Separación del conjunto según características**

  Nuestro conjunto de datos se destaca por su diversidad de escenarios, abarcando variaciones en términos de luminosidad, presencia de vegetación y otros aspectos.

  Por lo tanto, surge una oportunidad de mejorar aún más nuestro modelo mediante la exploración de la segmentación por medio de la separación de imágenes diurnas y nocturnas. Esta estrategia busca optimizar nuestros resultados considerando las diferencias a cada conjunto, ayudado por algoritmos de ML, como el K-Means.

- **Crear aplicación para los estudiantes**

  Como última mejora, me gustaría mencionar aquello que me hubiera gustado tener en mi desarrollo del grado. Sería de gran interés proporcionar una aplicación a los estudiantes donde puedan utilizar modelos DL, en nuestro caso para solventar tareas de segmentación de imágenes.

  Esta última mejora puede ser de gran ayuda para familiarizar al estudiantado con modelos más complejos como los tratados en este proyecto.

Bibliografía

[1] Franck Trolliet, Cédric Vermeulen, Marie-Claude Huynen, and Alain Hambuckers. Use of camera traps for wildlife studies: a review. *Biotechnologie, Agronomie, Société et Environnement*, 18(3), 2014.

[2] Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5–28, 1998.

[3] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[4] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[5] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. Multilayer perceptron: Architecture optimization and training. 2016.

[6] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[8] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series.

[9] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3523–3542, 2021.

[10] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[12] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[13] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[17] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6881–6890, 2021.

[18] Mark Hamilton, Zhoutong Zhang, Bharath Hariharan, Noah Snavely, and William T. Freeman. Unsupervised semantic segmentation by distilling feature correspondences. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=SaKO6z6Hl0c.

[19] Adarsh Verma. Most popular programming languages for machine learning and data science, Jan 2017. URL https://fossbytes.com/popular-top-programming-languages-machine-learning-data-science/.

[20] Popularity of programming languages, Jun 2022. URL https://www.tiobe.com/tiobe-index/.

[21] Igor Stančin and Alan Jović. An overview and comparison of free python libraries for data mining and big data analysis. In *2019 42nd International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 977–982. IEEE, 2019.

[22] F Charte. Cómo es el proceso de extraer conocimiento a partir de bases de datos. campusmvp. es. *campusMVP. es. https://www. campusmvp. es/recursos/post/el-proceso-de-extraccion-de-conocimiento-a-partir-debases-de-datos. aspx*, 2020.

[23] María José del Jesus Díaz David de la Rosa, Francisco Charte Ojeda. Procesamiento de imágenes de foto-trampeo con técnicas de aprendizaje profundo para el descarte automático de imágenes vacías. 2021.

[24] Nikhil R Pal and Sankar K Pal. A review on image segmentation techniques. *Pattern recognition*, 26(9):1277–1294, 1993.

[25] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[26] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[27] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[28] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[29] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[30] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[31] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[32] Michael R Berthold, Christian Borgelt, Frank Höppner, and Frank Klawonn. *Guide to intelligent data analysis: how to intelligently make sense of real data*. Springer, 2010.

[33] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[34] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[35] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[36] Bing Xu and Tianqi Chen. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019. URL https://arxiv.org/abs/1903.06733.

[37] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[38] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[39] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27*, pages 270–279. Springer, 2018.

[40] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1017–1024, 2011.

[41] David Charte, Francisco Charte, Salvador García, María J del Jesus, and Francisco Herrera. A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines. *Information Fusion*, 44:78–96, 2018.

[42] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[43] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[44] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[45] Cliff B Jones. *Understanding Programming Languages*. Springer, 2020.

[46] Hamed Habibi Aghdam, Elnaz Jahani Heravi, Hamed Habibi Aghdam, and Elnaz Jahani Heravi. Caffe library, 2017.

[47] Pramod Singh, Avinash Manure, Pramod Singh, and Avinash Manure. Introduction to tensorflow 2.0. *Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python*, pages 1–24, 2020.

[48] Leila Etaati and Leila Etaati. Deep learning tools with cognitive toolkit (cntk). *Machine Learning with Microsoft Technologies: Selecting the Right Architecture and Tools for Your Project*, pages 287–302, 2019.

[49] Md Rezaul Karim. *Java Deep Learning Projects: Implement 10 real-world deep learning applications using Deeplearning4j and open source APIs*. Packt Publishing Ltd, 2018.

[50] Taylor B Arnold. kerasr: R interface to the keras deep learning library. *J. Open Source Softw.*, 2(14):296, 2017.

[51] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[52] Alberto León Batallas, Javier Bermeo Paucar, Juan Paredes Quevedo, and Henry Torres Ordoñez. Una revisión de las métricas aplicadas en el procesamiento de imágenes. *RECIMUNDO: Revista Científica de la Investigación y el Conocimiento*, 4(3):267–273, 2020.

[53] Jim Nilsson and Tomas Akenine-Möller. Understanding ssim. *arXiv preprint arXiv:2006.13846*, 2020.