



UNIVERSIDAD DE GRANADA

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA Y
ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS

Identificación de micro y macro organismos
usando Redes Neuronales Convolucionales

Subtítulo

Autor

Jacobo Casado de Gracia

Director

Siham Tabik



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Junio de 2022

Índice general

1. Introducción	5
1.1. Descripción del problema	5
1.2. Objetivos	7
1.3. <i>Workflow</i> del proyecto y planificación temporal	10
2. Fundamentos teóricos	12
2.1. Machine Learning	12
2.1.1. Aprendizaje supervisado	12
2.1.2. Aprendizaje no supervisado	12
2.2. Deep Learning	13
2.3. Redes Neuronales Convolucionales para la clasificación	16
2.3.1. Extracción de características	16
2.3.2. Clasificación	21
2.3.3. Visualizando las <i>convnets</i>	23
2.4. Entrenamiento de la red	24
2.5. Transfer Learning y Fine Tuning	25
2.5.1. Transfer learning (extracción de características fija)	26
2.5.2. Fine-Tuning	26
2.5.3. Reentrenamiento completo	26
2.6. Limitaciones del <i>deep learning</i>	27
3. Estado del arte	29
3.1. Clasificación manual de Diatomeas	29
3.2. Clasificación automática de Diatomeas	30
3.2.1. Obtención del <i>outline</i>	30
3.2.2. Extracción de la ornamentación	31
3.2.3. Detección de los <i>ultimate points</i> de la ornamentación	32
3.2.4. Comparación con diatomeas de la base de datos	32
3.3. ADIAC como el estudio piloto de clasificación usando <i>Machine Learning</i>	33
3.4. <i>Deep Learning</i> para la clasificación de Diatomeas	34
3.4.1. EfficientNet	34
3.4.2. EfficientNetV2	40
4. Creación del <i>dataset</i>	44
4.1. Obtención de los datos (<i>dataset</i> inicial)	44
4.1.1. Datos obtenidos	44
4.2. Análisis de las imágenes	46
4.3. Preprocesado	48
4.3.1. Limpieza y recorte	48
4.3.2. <i>Data Augmentation</i>	50
4.4. Creación de los <i>dataset</i> de entrenamiento y <i>test</i>	50

5. Diseño e implementación	51
5.1. <i>Framework</i> utilizado	51
5.2. Arquitecturas utilizadas	51
6. Análisis de resultados	52
6.1. Técnica utilizada para el análisis	52
6.2. Comparativa de arquitecturas	52
6.3. Selección del mejor modelo	52
6.4. Visualización del aprendizaje	52

Índice de figuras

1.1.	Diatomeas en una muestra de microscopio	5
1.2.	Dos diatomeas de diferentes especies	6
1.3.	Imágenes de la diatomea Achnantes	7
1.4.	Diagrama de Gantt realizado para la planificación del trabajo	11
2.1.	Ejemplos de aprendizaje supervisado y no supervisado	13
2.2.	Comparativa del proceso de <i>machine learning</i> versus <i>deep learning</i>	14
2.3.	Idea básica de una red neuronal para clasificar Diatomas	14
2.4.	Ejemplo de una ConvNet básica	16
2.5.	Generación de mapa de características a partir de una convolución	17
2.6.	Representación visual del proceso de convolución	17
2.7.	Generación de varios mapas de características usando convolución con varios kernels	18
2.8.	Ejemplo del uso de la función <i>maxpooling</i> en una matriz de datos	19
2.9.	Ejemplo de la función de activación ReLU	20
2.10.	Ejemplos gráficos de varias funciones de activación	20
2.11.	Ejemplo de sobreajuste de manera gráfica con dos polinomios	21
2.12.	Función <i>softmax</i> aplicada a un vector	22
2.13.	Ejemplo de una ConvNet	23
2.14.	Esquema de <i>transfer learning</i> y <i>fine tuning</i>	27
2.15.	Uso del algoritmo LIME para detectar los píxeles informativos de la imagen	28
3.1.	Características principales utilizadas para la clasificación manual de Diatomeas	30
3.2.	Extracción del <i>outline</i> de la Diatomea en ImageJ	31
3.3.	Extracción de la ornamentación de la Diatomea en ImageJ	31
3.4.	Extracción de los <i>ultimate points</i> de la Diatomea en ImageJ	32
3.5.	Comparativa de la Diatomea con los diatomeas candidatos en ImageJ	33
3.6.	Fórmula para modificar la profundidad, anchura y resolución de la red usando ϕ en el <i>compound scaling</i>	35
3.7.	Método del <i>compound scaling</i> en comparación a escalados unidimensionales de la arquitectura	36
3.8.	Resultados del <i>compound scaling</i> comparando tres modelos base y escalados en una dimensión con los mismos modelos tras aplicar este escalado compuesto	37
3.9.	Mapas de activación de clases (CAM) de dos imágenes con un modelo base y tras aplicar diferentes tipos de escalado	37
3.10.	Baseline obtenida en el proceso de búsqueda de la red que se comporta mejor con el <i>compound scaling</i> (Finalmente llamada EfficientNetB0)	38
3.11.	Comparativa de los modelos de la familia EfficientNet con modelos del estado del arte.	39
3.12.	Dos diatomeas de diferentes especies	40
3.13.	Sustitución de los bloques convolucionales MBConv por Fused-MBConv . .	41

3.14. Resultados del experimento de aumento de la regularización del modelo conforme el tamaño de entrada aumenta	42
3.15. Arquitectura de la red EfficientNetV2S	42
3.16. Comparativa de la arquitectura EfficientNetV2 respecto a otras redes del estado del arte	43
4.1. Cantidad de imágenes obtenidas según clase y coloreadas según procedencia	46
4.2. Muestra de 25 diatomeas aleatorios de la base de datos construida	47
4.3. Ejemplo del preprocesado de una imagen	49
4.4. Muestra de 25 diatomeas aleatorios tras el preprocesado	50

1 | Introducción

1.1. Descripción del problema

Los diatomas son un grupo de algas unicelulares, siendo el grupo más común de microorganismos en los hábitats acuáticos y presentes en gran cantidad de ellos. Son uno de los principales contribuidores en la cadena alimenticia de los ecosistemas acuáticos, formando parte de la base de ésta.

Debido a que los diatomas se adaptan al ecosistema y modifican su forma y textura dependiendo de éste, actualmente se usan como uno de los indicadores más precisos para medir la calidad del agua en estudios ambientales y, por tanto, para estudios más complejos como los relacionados con el cambio climático y su evolución [1, 2].

Los diatomas tienen varias ventajas sobre otros bioindicadores que los hacen los bioindicadores ideales: es relativamente fácil obtener muestras grandes de ellos sin impactar en el ecosistema en el proceso, tienen una respuesta rápida a la variación de las condiciones del ecosistema y son muy sensibles a cambios del ambiente que otros bioindicadores ni siquiera plasman (sobre todo los cambios en las variables químicas del agua) [3].

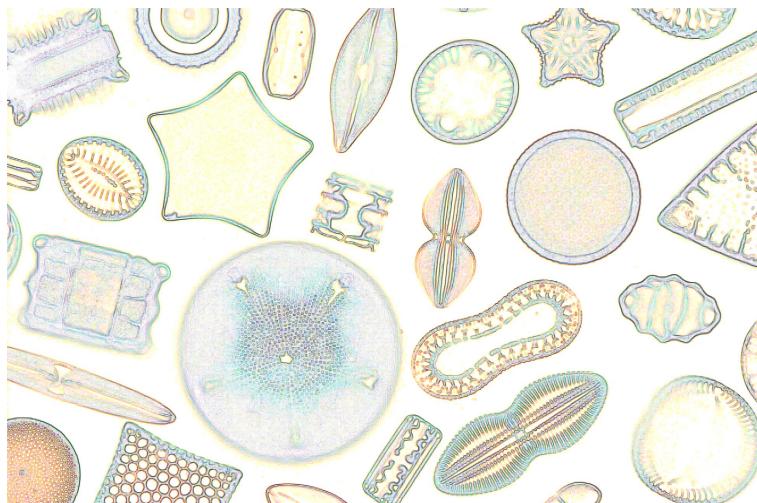


Figura 1.1: Varias clases de Diatomeas en una muestra de microscopio. Fuente:

Tradicionalmente y al igual que en mucho otros campos científicos, la tarea de identificación y clasificación de diatomas de muestras tomadas en microscopios es tarea de los biólogos. Estos profesionales generalmente buscan características **morfométricas** (longitud, anchura y forma del diatoma, sobre todo) así como detalles internos del organismo, como la densidad de estrías (**información sobre la textura**). Estas son unas pocas de las cientos de características necesarias para obtener información relevante sobre qué clase de diatoma se está analizando. Una vez se han medido estas características principales (*key features*), la manera de clasificarlos es compararlos respecto a diatomas previamente clasificados, pero hay varios retos en el camino:

1. Se requieren expertos en la taxonomía de los diatomas para resolver este problema, ya que algunas de las *key-features* que se toman para identificar al diatoma requieren conocimiento amplio en el tema, incluyendo medidas no cuantitativas.
2. Los expertos deben analizar continuamente estas *key-features* cuando quieren clasificar una nueva diatomea, por lo que tienen que realizar un análisis de características distintivas cada vez que se encuentran con una nueva clase.
3. El análisis manual de las imágenes es un proceso muy largo debido a la cantidad de características por analizar, y para llegar a una conclusión válida sobre la calidad del agua además se necesitan alrededor de 400 imágenes por estudio [4].
4. Los diatomas tienen similitudes inter-especies y disimilitudes intra-especies que hacen esta tarea más árdua aún.

Un ejemplo de similitudes inter-especies es el siguiente, donde se presentan dos diatomas de clases distintas.

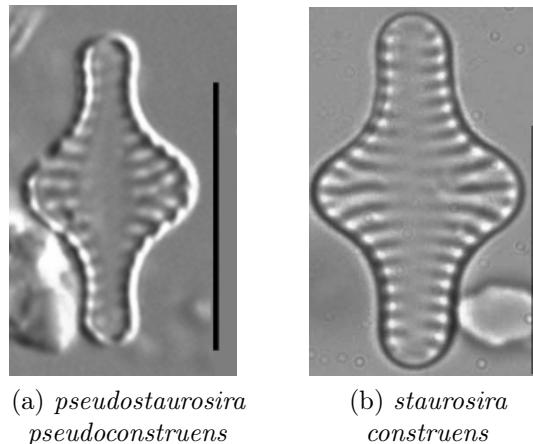


Figura 1.2: Dos diatomeas de especies diferentes. Fuente: añadir.

Estos dos diatomeas son de clases diferentes pese a que a simple vista tengan muchas características en común, como su forma o estrías. El caso contrario es encontrar diatomeas que pertenecen a la misma especie teniendo formas muy diferentes (disimilitudes intra-especie)

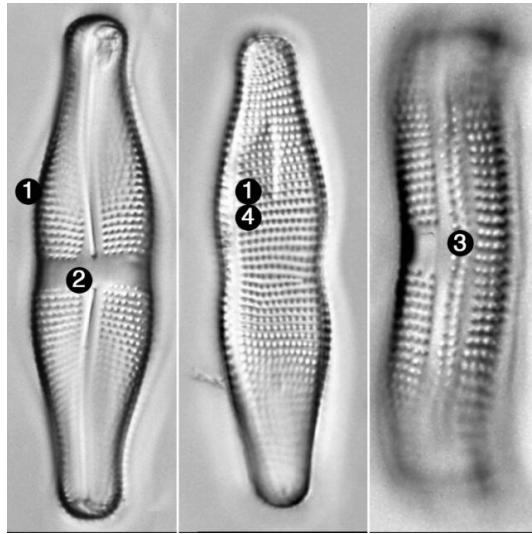


Figura 1.3: Varias imágenes de la misma diatomea *Achnantes*. Fuente: añadir

Podemos ver que la tarea de clasificación no es una tarea sencilla a simple vista.

5. Como se comenta en [5], actualmente hay tasadas alrededor de 10.000 especies, pero se estima que el número total de especies es mayor a 200.000, por lo que el tamaño del problema hace que la tarea de clasificación sea aún más complicada.
6. Para más complejidad, un diatoma cambia de forma durante su ciclo de vida (la figura 1.3 es un ejemplo), y ese *polimorfismo* hace que sea mucho más difícil clasificarlo con métodos manuales [6].

En este proyecto se trabaja en un marco de investigación con la Universidad de Granada, en la creación de un sistema de clasificación de diatomas con un enfoque basado en el aprendizaje automático (*machine learning*), proponiéndose una serie de arquitecturas conocidas en el mundo de la *visión por computador* como Redes Neuronales Convolucionales (CNN) para resolver este problema de clasificación, con todos los problemas asociados que incluye.

Para ello, se probarán distintas arquitecturas y se realizará una comparativa para ver de cuál de ellas es la que mejor se adapta a este problema en cuestión y obtiene mejores resultados.

1.2. Objetivos

El objetivo de este trabajo no es sólo dar un enfoque diferente para resolver este problema de clasificación usando las técnicas de *aprendizaje automático*, sino paliar los problemas surgidos del enfoque clásico de clasificación de diatomas, automatizando la tarea sin necesidad de recurrir a expertos en el ámbito, reducir el error humano de estos problemas de clasificación y ahorrar un tiempo en la tarea de búsqueda de características y de clasificación.

El **objetivo global** de este Trabajo de Fin de Grado es diseñar un sistema automático para la clasificación de Diatomeas en imágenes microscópicas.

Para lograr ese objetivo, es necesario desglosar y trabajar los siguientes objetivos parciales:

- **Subobjetivo 1. Construir una base de datos de Diatomeas para la clasificación de Diatomeas usando Redes Neuronales Convolucionales.**

Antes de diseñar una red neuronal convolucional que sea capaz de clasificar Diatomeas, debemos de fijar qué clases de Diatomeas queremos clasificar para que la base de datos que construyamos contenga ejemplos o muestras de cada clase, y además de ello, que las imágenes de cada clase sean representativas a lo que nos podemos encontrar en la realidad.

Para hacer que la red sea más robusta (por ejemplo, independiente a imágenes rotadas, imágenes provenientes de distintos microscopios y de diferentes laboratorios), deberemos de diseñar una base de datos de calidad: se usarán técnicas de preprocesado junto a las imágenes obtenidas para lograr este subobjetivo, que tendrá un apartado en la práctica independiente debido a la importancia de cara al problema.

No podemos olvidar que estamos en un problema de *machine learning*, y, por tanto, la solución al problema se plantea mediante el aprendizaje, donde los **datos** (en nuestro caso, **imágenes**) forman una parte muy importante de éste. En el apartado de fundamentos teóricos hablaremos del enfoque del *machine learning* para solucionar este problema y la función de los datos en el aprendizaje.

- **Subobjetivo 2. Analizar el uso de Redes Neuronales Convolucionales para la clasificación de Diatomas.**

En este apartado se usarán las Redes Neuronales Convolucionales como extractores de características y clasificadores de las imágenes de Diatomas.

Para analizar el uso de estas redes, se comenzará utilizando una Red Neuronal Convolucional *ad-hoc* (es decir, diseñada desde cero), y posteriormente se usarán arquitecturas ya prediseñadas para esta tarea que han logrado resultados satisfactorios en bases de datos de millones de imágenes, como DenseNet o EfficientNet, presentes actualmente en el estado del arte del *deep learning* y que tienen arquitecturas más avanzadas.

Estas últimas arquitecturas se podrán implementar de dos maneras; o bien como extractores de características ya preentrenados y sin entrenarlas de nuevo para cubrir nuestro problema (*transfer learning*), o se podrán reentrenar con nuestros datos para extraer características del problema en cuestión (*fine tuning*); se comprobará cual de estas dos alternativas es mejor de cara a nuestro problema de clasificación.

Para cada una de estas arquitecturas (y sus diferentes hiperparámetros) se evaluará su rendimiento a la hora de solucionar este problema de clasificación, usando, para ello, diferentes métricas que nos servirán para evaluarlas individualmente y compararlas entre ellas. De esta manera habremos cumplido satisfactoriamente el objetivo.

- **Subobjetivo 4. Integrar el mejor modelo en una interfaz gráfica.** Para cumplir este objetivo, usaremos las métricas vistas en el proceso de entrenamiento y validación de los modelos y elegiremos un modelo ganador teniendo en cuenta estos valores y aplicando cierto criterio. Posteriormente, para la segunda parte del subobjetivo se creará una plataforma web donde se automatice el proceso de clasificación de Diatomas bajo una interfaz gráfica, por ejemplo, una página *web*. El usuario, tras adjuntar una imagen de una Diatomea, podrá ver de qué clase es ese

Diatomea en cuestión, consiguiendo haber automatizado el proceso de clasificación usando nuestro modelo.

Cada uno de estos subobjetivos es igual de relevante para cumplir el objetivo final, por lo que se dispondrá de un apartado en este trabajo con el proceso realizado para cubrir cada uno de ellos (obtención de base de datos, experimentación, evaluación del mejor modelo y desarrollo en interfaz gráfica).

En el siguiente apartado se establece un marco temporal donde se siguen todos estos objetivos, indicando también las dependencias entre cada uno de ellos.

1.3. *Workflow* del proyecto y planificación temporal

El workflow de este proyecto sigue una planificación lineal mayoritariamente, debido que para realizar una tarea como entrenar y comparar los modelos es necesario obtener los datos previamente. No obstante, a pesar de ser lineal se puede volver hacia etapas anteriores en caso de ser necesario.

El diagrama de la imagen anterior muestra las tareas fundamentales para el desarrollo del proyecto, que se desglosan a continuación.

Tarea 1: Definir la tarea que queremos conseguir con el modelo y los requisitos que debe de cumplir nuestro modelo de cara a realizar esa tarea. En nuestro caso, la tarea ya está definida (automatizar la clasificación de diatomas) pero quedan algunos aspectos por ver, como la cantidad de clases que el modelo va a aprender a clasificar, y cuáles en concreto. Los requisitos de nuestro modelo también deben de especificarse.

Tarea 2: Obtener datos con los que entrenar a nuestro modelo. Para ello es necesario validar la calidad de esos datos y asegurarnos de que son representativos del problema que estamos queriendo solucionar (de ahí la importancia de definir bien el problema en el paso anterior). Separar los datos en un conjunto de **entrenamiento** y **test**, para las tareas 3 y 4.

Tarea 3: Entrenar modelos de machine learning con los datos del paso anterior y realizar comparativas entre los modelos. Para ello es necesario establecer líneas de base con las que comparar. En este paso se comprueba si los datos son suficientes para ajustar nuestro modelo (si no, se vuelve a la tarea 2 para obtener más datos) o si el modelo no cumple la tarea (en ese caso, se vuelve a la tarea 1, porque hay que reformular el objetivo).

Tarea 4: Seleccionar un modelo usando estas líneas de base y se evalúa en la distribución de datos de *test*, datos que jamás ha visto y que nos permiten ver cómo generaliza el modelo ante nuevos datos.

En caso de que los resultados con nuevos datos sean lo esperado, se pasa al siguiente paso; en caso contrario es necesario ver si faltan datos o hay que buscar otros modelos, para volver a la tarea 3 o 4 respectivamente.

Tarea 5: Lanzar el modelo en la plataforma, comprobando que el uso es correcto y monitorizando sus resultados.

Si se detecta alguna desviación en algún paso, se podría volver hacia atrás. No sólo eso; podemos implementar un modelo y posteriormente encontrar más datos que se ajustan a nuestro problema haciendo el modelo más robusto o consiguiendo que generelize mejor, o puede surgir la necesidad de hacer que el modelo detecte una clase nueva que no detecta actualmente (cosa que es común que pase, sobre todo cuando se recopilan más datos). En ese caso el ciclo volvería a empezar en la fase correspondiente, pero se seguiría siguiendo este *workflow*.

Cada etapa de las comentadas anteriormente se realizarán en un apartado separado de la memoria; por último, en el siguiente gráfico se representa el espacio temporal asignado a las tareas necesarias para desarrollar el proyecto

mediante un diagrama de Gantt, dando especial importancia a las tareas 2, 3 y 4 debido al tiempo que consume la obtención de datos y el entrenamiento de los modelos. Las tareas 1 a la 5 se corresponden con las mencionadas anteriormente, mientras que la tarea 6 se trata de la redacción continua de la memoria del proyecto y la tarea 7 son las reuniones fijadas a lo largo del proyecto con el tutor.

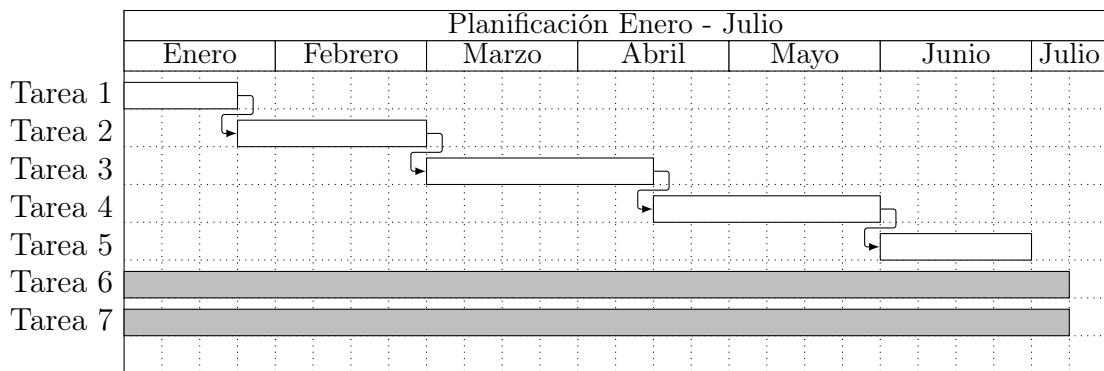


Figura 1.4: Diagrama de Gantt estructurado en semanas para la planificación del trabajo.

Fuente: Elaboración propia

2 | Fundamentos teóricos

2.1. Machine Learning

El *Machine Learning* es una rama de las Ciencias de la Computación y a su vez de la Inteligencia Artificial. Pese a tener muchas definiciones y ser un concepto relativamente abstracto, una definición que considero bastante apropiada es llamar Machine Learning al “desarrollo de sistemas capaces de **cambiar su comportamiento de forma autónoma en base una experiencia**, consiguiendo la capacidad de aprender a partir de unos datos dados”. [7]

A mí me gusta dar otra definición que la complementa: el uso de un conjunto de observaciones para descubrir un patrón oculto detrás de ellas. Es una premisa difícil y a veces no se cumplirá, pero es la intención de trabajar con modelos de aprendizaje automático: hallar un patrón que permita clasificar un dato o detectar un objeto en una imagen. Dentro del *machine learning*, hay dos metodologías o paradigmas de aprendizaje fundamentales, de los cuales trabajaremos con uno. No obstante, considero interesante saber acerca de la otra ya que es comúnmente usada.

2.1.1. Aprendizaje supervisado

Cuando los datos con los que el modelo se entrena contienen **ejemplos explícitos** de cuál es la **correcta salida o etiqueta** para un dato (por ejemplo, al cliente A no se le concede un crédito pero al cliente B sí) y por tanto, tenemos una colección de (**datos, etiquetas**), cada dato con su etiqueta correspondiente, estamos ante este tipo de aprendizaje.

El aprendizaje se llama supervisado debido a que alguna persona ha tenido que supervisar los datos de cierta manera para darle un valor correcto a cada uno de ellos.

Nuestro problema es un problema de aprendizaje supervisado debido a que coleccionaremos imágenes de diatomeas (**datos**) de diferentes clases (la clase en este caso es la etiqueta). Cada imagen tendrá una etiqueta asociada correspondiente a la clase a la que pertenece.

2.1.2. Aprendizaje no supervisado

En este tipo de aprendizaje los datos de entrenamiento no poseen una etiqueta asociada, es decir, sólo tenemos las entradas. El modelo en este caso aprende a categorizar los datos y a diferenciarlos entre ellos (no puede decir que un dato es de cierto tipo porque no tiene ejemplos para ello); en nuestro caso, si no tuviésemos etiquetas de nuestras imágenes de Diatomeas, podríamos diseñar un modelo que diferenciase las imágenes de las diferentes clases a pesar de no tener información sobre de qué clase es cada imagen. A menudo este tipo de aprendizaje se le llama *clustering* ya que separa los datos en *clústers o agrupamientos*.

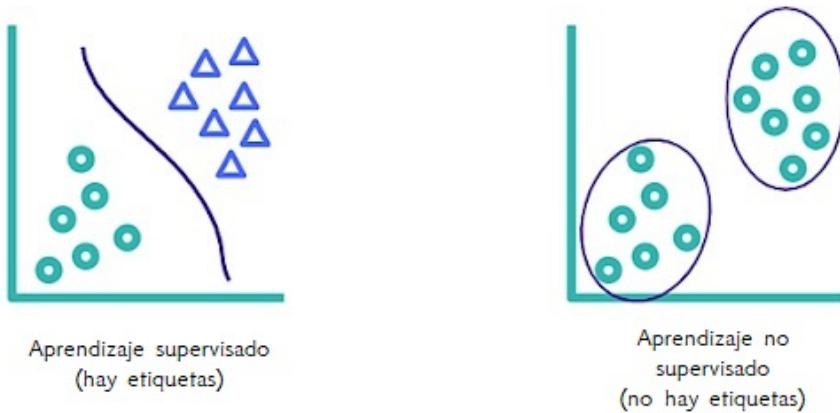


Figura 2.1: Ejemplo de un modelo a partir del aprendizaje supervisado y no supervisado.

Fuente: Elaboración propia

En la figura anterior se muestra un ejemplo sencillo de la diferencia entre los dos tipos de aprendizaje. Un mismo problema permite abordarse de ambas maneras, dependiendo de la información que se disponga (si hay etiquetas o no).

En nuestro caso, queremos diseñar una frontera de decisión o un clasificador: este problema se abordará mediante el aprendizaje supervisado.

2.2. Deep Learning

Una vez visto qué es el *machine learning*, el siguiente paso es hablar del *deep learning*, o aprendizaje profundo.

Las técnicas convencionales de *machine learning* necesitan un paso previo antes de su aplicación, y es la **extracción de características**, puesto que lo único que se entrena es el **clasificador** que usa las características durante el entrenamiento para delimitar una frontera de decisión entre una clase u otra.

Es decir, para saber diferenciar entre un objeto de la clase A y otro de la clase B es necesario **delimitar qué características los diferencian**, y usarlas como datos de entrada para nuestra red. De esta manera el modelo, una vez entrenado, usará las variables que hemos determinado y con ellas establecerá una diferencia entre las clases.

El problema surge cuando tenemos problemas como el que estamos estudiando donde se requiere a un experto en el dominio para delimitar las características principales de cada clase y, por tanto, la selección de características se convierte en un problema más.

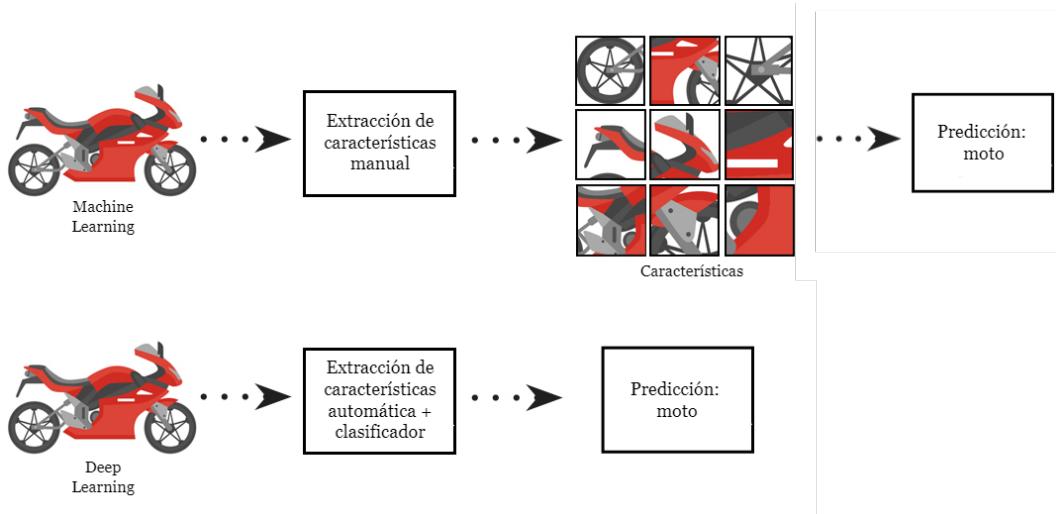


Figura 2.2: Comparativa del proceso de *machine learning* versus *deep learning*.

Fuente: Elaboración propia a partir de [8].

En el *deep learning* esto no es así. Las características que definen las clases se aprenden de manera automática generalmente mediante modelos llamados *redes neuronales*, estructuras de capas apiladas una encima de otra donde cada capa surge a partir de una transformación de la capa anterior. Estas transformaciones de datos se aprenden por la exposición a ejemplos, ahorrando una gran cantidad de tiempo a la hora de decidir qué características son las que definen a cada tipo de dato.

Debemos de tener en cuenta que al trabajar con datos es necesario disponer de una gran cantidad de ellos para poder aprender no sólo el cómo clasificarlos, sino que el modelo extraerá las características fundamentales de cada tipo de dato y los datos con los que entrenemos serán los que el modelo utilizará para recrear las características que diferencian a cada clase, por lo que los datos con los que entrenemos tienen que ser representativos de la realidad para que el modelo obtenga características reales de nuestro problema que permitan clasificar datos nuevos. Vemos que si los datos juegan un rol importante en el *machine learning*, en el *deep learning* son aún más importantes.

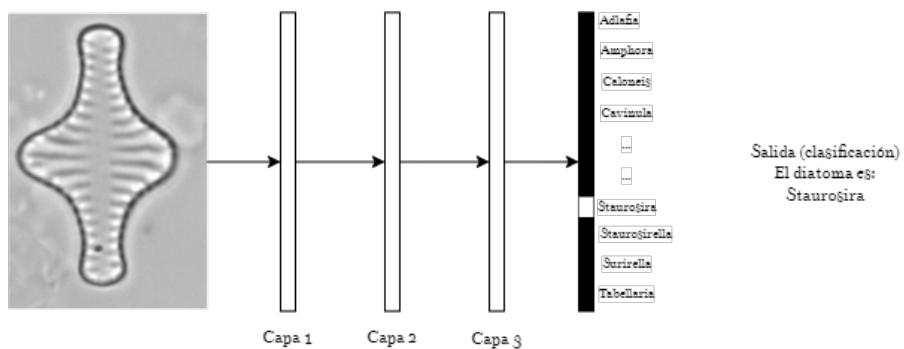


Figura 2.3: Ejemplo gráfico de una red neuronal usada para la clasificación de Diatomas.
La salida es la clase estimada por la red.

La idea plasmada en la figura 2.2 es que, para dar la predicción final, la red transforma el dato de entrada (imagen) en representaciones que son cada vez más diferentes de la

imagen original y más informativas respecto al resultado final. Más tarde veremos cómo, de manera concreta.

2.3. Redes Neuronales Convolucionales para la clasificación

Las Redes Neuronales Convolucionales (llamadas a partir de ahora *ConvNets*), son un tipo de redes neuronales muy usadas para las tareas de reconocimiento y clasificación de imágenes, debido a que son capaces de detectar una gran cantidad de características de una imagen mediante decenas o cientos de capas ocultas. Cada capa oculta aumenta la complejidad y cantidad de características de la imagen aprendida.

Por ejemplo, la primera capa oculta podría aprender el cómo detectar bordes o esquinas, mientras que la segunda aprende a detectar formas más complejas propias del objeto que se intenta reconocer (orejas en caso de un animal, hojas en caso de un árbol...).

La estructura fundamental de estas redes se compone de combinaciones de bloques que, conectados entre sí, realizan el proceso de extracción de características y clasificación. Cada bloque se genera de una manera distinta y tiene una función específica en el proceso de aprendizaje, aunque, a alto nivel, el primer conjunto de capas estarán dedicadas a la **extracción de características** y las capas finales serán las que se ocupen de la **clasificación** de la imagen.

En la parte de extracción de características, como su nombre indica, se extraerán las características de los datos de entrada formando los mapas de características (*feature maps*), los cuales generalmente aumentan en número y cantidad, pero disminuyen en tamaño, obteniendo al final características de alto nivel y mapas de características muy pequeños que representan pequeños detalles de la imagen.

Cuando estos mapas de características han sido extraídos, una red neuronal *fully connected*, es decir, donde todos sus elementos tienen cierto enlace entre sí, se encarga de separar y clasificar cada característica.

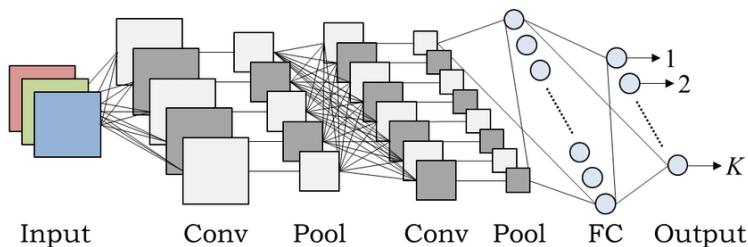


Figura 2.4: Ejemplo gráfico de una ConvNet, donde se diferencia entre extracción de características y clasificación.

Fuente: [9]

Una vez viendo el esquema de la red, tomando como ejemplo la figura , es importante entender además que hay que tener en cuenta el cómo **las capas se entrena**n y modifican sus valores para adaptarse al problema.

Por tanto, desglosaremos los conceptos teóricos de las ConvNets en tres partes: **extracción de características, clasificación y entrenamiento**.

2.3.1. Extracción de características

En esta parte se comentan los fundamentos teóricos principales que conforman la parte de extracción de características realizadas por las ConvNets.

Convolución

Las ConvNets son capaces de aplicar una gran cantidad de transformaciones a los datos gracias al operador de **convolución**. La convolución es el operador usado para transformar un bloque de entrada (ya sea el bloque de datos inicial u otro creado a lo largo del proceso) en otro bloque de salida con otra estructura.

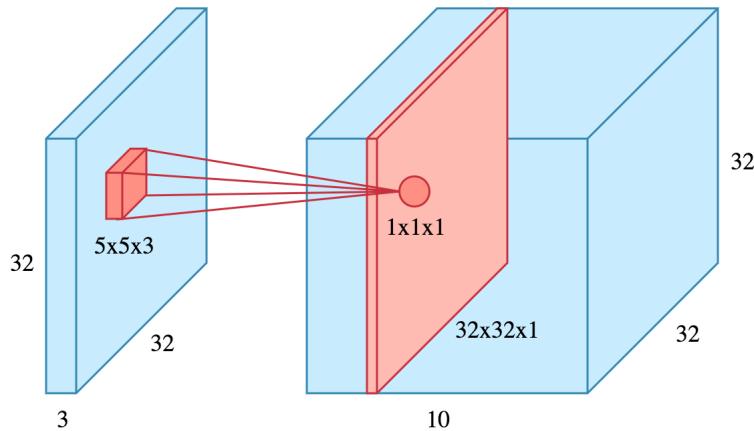


Figura 2.5: Generación de un nuevo mapa de características (**rojo**) a partir de una convolución con un kernel 2D.

Como se aprecia en la figura 2.5, la operación de convolución necesita dos cosas: un bloque de entrada que será transformado (o convolucionado), bloque que llamaremos a partir de ahora **mapa de características** y un **kernel**, que será lo que aplicaremos sobre el bloque de entrada para transformarlo al bloque de salida, que será un mapa de características resultante.

Queda claro que en la convolución se utiliza un kernel que transforma un bloque de entrada en otro de salida, pero, ¿qué se hace exactamente? En la siguiente imagen se describe de manera visual el proceso matemático de convolución:

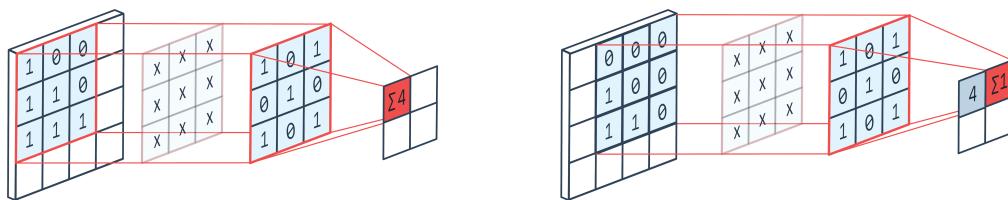


Figura 2.6: Representación visual del proceso de convolución, utilizando dos convoluciones de ejemplo.

En la figura anterior se puede apreciar que se han realizado dos convoluciones, una por cada píxel resultante. Cada píxel es resultado de una operación matemática muy sencilla: se multiplica, píxel a píxel, el kernel con la zona sobre la que se aplica del bloque y la sumatoria de ese producto es el valor del píxel resultante.

En ambas convoluciones el **kernel** es el mismo y lo que varía es el lugar donde el kernel se 'posa' para finalmente generar el píxel de salida. Podemos ver que cada píxel tiene un campo receptivo (en definitiva, los píxeles que han sido utilizados para generar ese nuevo

píxel). Convoluciones de mayor tamaño harán que el píxel de salida tenga un mayor campo receptivo y se vean involucrados más píxeles en la operación.

Lo principal que tenemos que entender de la operación de convolución es que es una operación local (pues utiliza varios píxeles de una zona en concreto para generar uno nuevo) y punto a punto (se recorren todos los píxeles) en la que el **kernel** se **desliza o desplaza** sobre el bloque de características de entrada **completo**, elemento a elemento, y de manera horizontal y vertical y a lo largo de toda su profundidad para generar un mapa de características y que el **kernel** no opera con un solo elemento del bloque de entrada por cada pasada, sino que toma varios de los elementos de este bloque para generar **uno** del siguiente bloque. Si nos fijamos en la figura 2.5, el **kernel** de tamaño $5 \times 5 \times 3$ usaría 25 elementos del mapa de características A para generar un elemento del mapa de características B. Como se ha especificado que la profundidad del kernel es 3 ($5 \times 5 \times 3$), se utilizan 75 elementos, 25 por cada capa del bloque A, para generar un solo elemento del siguiente bloque B.

Otra cosa a destacar es que no todos los datos que se utilizan y combinan para crear un dato del siguiente mapa de características **tienen la misma importancia**. La importancia que cada dato tiene varía en función del **kernel**, que **pondera** cada dato con un cierto valor (generalmente, entre 0 y 1) de manera que cada dato tiene su importancia relativa en la operación de convolución.

Esto hace que los mapas de características varíen dependiendo del **kernel** aplicado, tanto en dimensión como en el valor. Como ya hemos visto matemáticamente, el bloque de características de entrada se transforma en nuevos mapas de características y por tanto, otra representación de los datos diferente; en el proceso de aprendizaje se buscará el conjunto de valores **para los pesos de todas las capas de la red** de manera que la red asigne correctamente las etiquetas a los ejemplos que le hemos proporcionado (es la única información de la que dispone, los datos). Lo que el ingeniero deberá de especificar son otros parámetros como el tamaño del kernel (generalmente 3×3 o 5×5), la cantidad de kernels utilizados para generar un bloque convolucional (definirá el tamaño del bloque) o la profundidad de la red (número de convoluciones realizadas de manera secuencial).

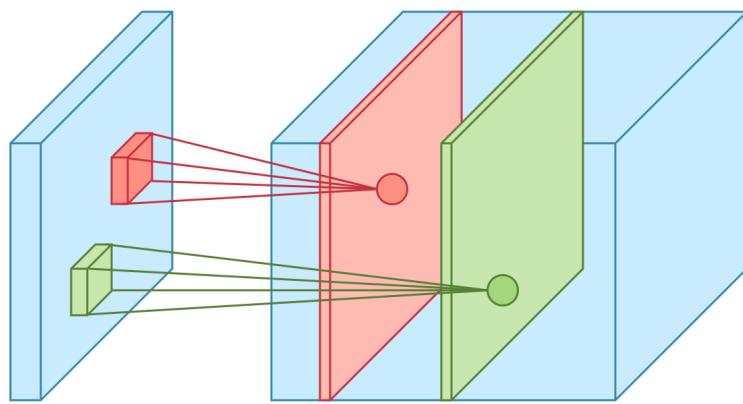


Figura 2.7: Ejemplo anterior pero mostrando el uso de otro **kernel** para generar un mapa de características más.

En la figura 2.7 se puede ver que usa más de un **kernel** para generar el nuevo bloque de características debido a que se ha especificado más de un mapa de características como

salida, y por tanto, diferentes representaciones del bloque anterior. Aquí se ve la clara distinción entre **mapa de características** (parte roja y verde del bloque de características) y **el bloque de características al completo** (azul).

Pooling

El *pooling* es una operación que se suele aplicar a bloques de características cuando consideramos que en ese bloque hay información redundante entre los píxeles, es decir, que de cierta manera, los píxeles del entorno tienen bastante información que comparten en común y se podrían resumir.^{en} uno solo, **reduciendo el número de parámetros entre bloques** (ya que el bloque se reduce en dimensiones) y aportando cierta **invarianza a las traslaciones**, de manera que las características de un objeto no tengan que estar en un lugar determinado, sino en cualquier lugar de la imagen. [10]

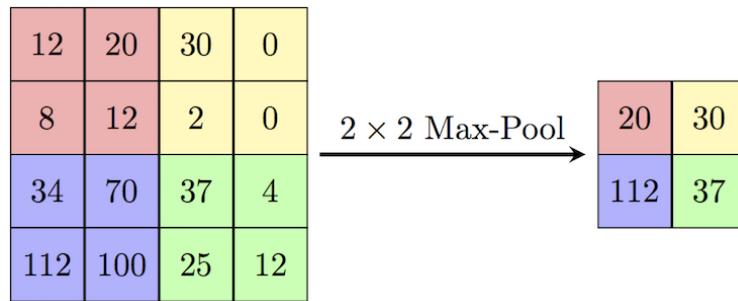


Figura 2.8: Ejemplo de la función *maxpooling* de tamaño 2 (2x2) en una matriz de datos. Como podemos ver, el tamaño del bloque se reduce considerablemente.

En la figura 2.8 podemos ver un ejemplo de *max pooling*, una función de *pooling* que toma el valor máximo de un cuadrado de lado l ; otras funciones conocidas son el L_p *pooling*, *mixed pooling*, *stochastic Pooling*... (más en [11]).

Activación

La función de activación se aplica a todos los elementos de mapas de características, y es aquella que permite principalmente el aprendizaje, es decir, el ajuste de los pesos.

La función de activación decide si una neurona debería ser activada o no (si vale 0 o más de cero), y por tanto, **delimita** los valores de los elementos del bloque convolucional.

Para entender mejor lo que hace, tomemos un ejemplo usando la función de activación más conocida y sencilla, la **ReLU**. Esta función transforma la entrada tal que así:

$$x = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{en otro caso} \end{cases}$$

Es decir, transforma todas las entradas en valores no negativos. Si algún valor es negativo, lo transforma a cero, y el resto de valores positivos mantienen su valor. Esta simple activación hace que el proceso de aprendizaje tenga efecto y se puedan aprender patrones en las imágenes.

Un ejemplo gráfico aplicado a un bloque convolucional es el siguiente, mostrado en la figura 2.2:

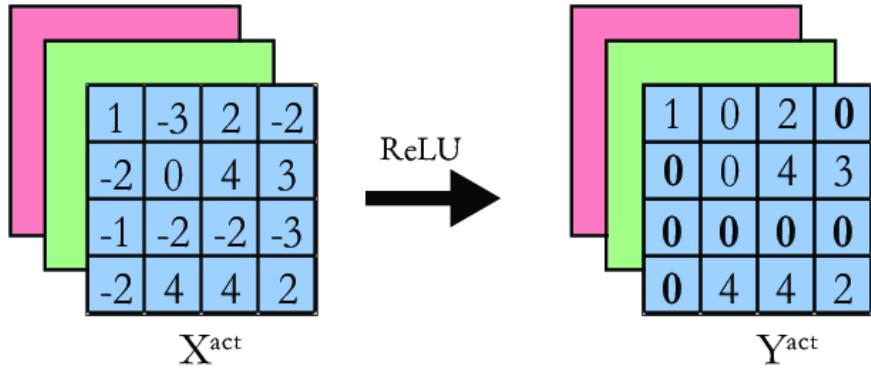


Figura 2.9: Ejemplo gráfico de una función de activación (en este caso, ReLU) aplicada a un mapa de características.

La salida es otro mapa con la función ya aplicada. Fuente: [12]

Si no hubiese función de activación, en el proceso de aprendizaje estaríamos **únicamente transformando la imagen de entrada una y otra vez**, sin **cribar o filtrar** información de un bloque a otro. Aplicando una función de activación, es decir, funciones **no lineales**, evitamos esto, haciendo que cada bloque tenga información distinta al anterior.

Se ha presentado la función de activación **ReLU**, pero hay muchas otras funciones no lineales de activación como la **sigmoide, hiperbólica o variantes de la ReLU, como la Leaky ReLU** (más en [11]); lo importante aquí es entender hasta ahora que la convolución transforma los datos y la función de activación nos criba parte de ellos, y que **en la gran mayoría de los casos**, cada bloque convolucional que se genera va acompañado de una función de activación, antes de volver a convolucionar.

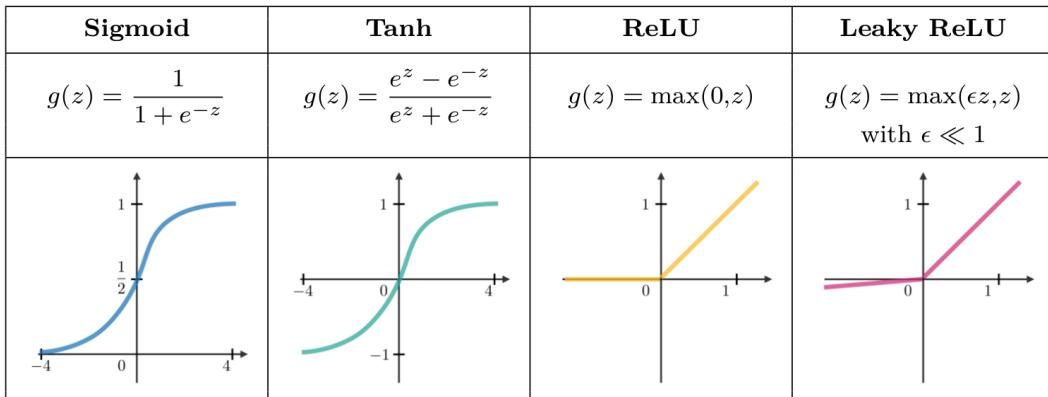


Figura 2.10: Ejemplos gráficos del comportamiento de algunas funciones de activación.
Fuente: [13]

Más tarde veremos la función de activación *softmax*, utilizada en nuestro problema para poder transformar la salida de la red neuronal en una predicción sencilla y fácil de entender.

Overfitting

Para poder definir bien este concepto, necesitamos definir bien previamente el concepto de *overfitting*. El *overfitting* es un problema muy común en las ConvNets, debido a la cantidad

de parámetros que estos modelos tienen, y ocurre cuando estas arquitecturas son capaces de ajustar bien los datos de entrenamiento, pero son muy pobres a la hora de predecir sobre datos nuevos, debido a que los **pesos** de la red se han sobreajustado para adaptarse a los datos de entrenamiento de una manera demasiado precisa. **En este momento, ya no estamos aprendiendo, sino memorizando literalmente los datos que tenemos** y la capacidad de predicción es mucho peor, debido a que los datos nuevos difieren de aquellos con los que hemos entrenado.

Podemos pensar que cuanta mayor es la complejidad de nuestro modelo (por ejemplo, un polinomio de grado 15 en comparación de un polinomio de grado 4), ajustaremos mejor nuestros datos ya que tenemos más grados de libertad, **pero puede ocurrir que la función objetivo real sea más simple que la del polinomio de grado 15 y los grados de libertad sobrantes de este polinomio se usan para ajustar información que no es de la función objetivo (datos incorrectos o con ruido... en general, información que no es la función objetivo)** como ocurre en la siguiente imagen, donde el polinomio de grado 15 hace lo que acabamos de comentar y ajusta el ruido, generalizando peor ante nuevos datos ya que no se ha conseguido acercar a la función objetivo y demostrando que **más grados de libertad no suponen un mejor modelo en todos los casos**.

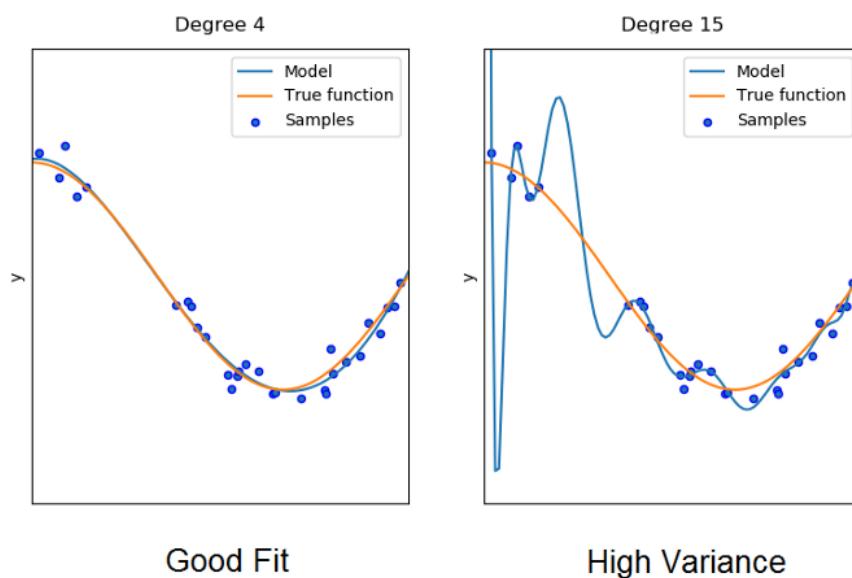


Figura 2.11: Ejemplo de sobreajuste o *overfitting* con un polinomio de grado 4 en comparación a uno de grado 15. Fuente:

En el gráfico anterior, pese a que se dibuja la función objetivo, en la realidad no será así, y por tanto la elección de la complejidad de nuestro modelo es un factor muy importante de cara al entrenamiento y buscar un modelo que no solo ajuste bien los datos sino que generalice ante datos nuevos, aunque sea a costa de no ajustar el total de los datos de entrenamiento. Esto se verá en el apartado práctico cuando experimentemos con diferentes arquitecturas, cada una con un número de parámetros diferente.

2.3.2. Clasificación

Una vez generados los bloques de características, el modelo debe de cumplir su función principal: clasificar el dato. La fase de extracción de características es una herramienta

de obtener información acerca de qué características son necesarias para diferenciar la imagen, pero la tarea de decidir acerca de qué etiqueta es la imagen que se está trabajando se realiza al final de la red, usando un clasificador para ello.

Capa totalmente conectada o *fully connected*

Como se ha visto en todas las operaciones anteriores, las neuronas de cada bloque convolucional no están totalmente conectadas, sino que únicamente la salida de un bloque depende del bloque anterior, pero no de toda la red.

En esta capa, al igual que en la convolucional, existe una ecuación para la salida de neuronas de manera que **cada neurona depende matemáticamente de las demás, estando, como el nombre de esta capa indica, totalmente conectadas** (más en [10]).

Si bien esta capa se puede aplicar en cualquier lugar de la red, su lugar más apropiado es al final de ella, debido a que es muy útil en las tareas de clasificación gracias a la función de activación *softmax*, que aplicada sobre esta capa permite obtener una clasificación final acerca del bloque de entrada.

Clasificación multiclas: Función de activación *softmax*

La función de activación *softmax* es comúnmente usada en problemas de clasificación multiclas, es decir, problemas donde hay que decidir la clase de un dato en un conjunto de más de dos clases.

Se utiliza porque transforma un vector de entrada en un vector de probabilidades (vector donde todos los elementos tienen un valor entre 0 y 1 y, además, la suma de todos los elementos del vector es igual a 1). Esto se consigue aplicando a cada elemento i del vector de entrada la siguiente fórmula:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, 2, \dots, K$$

Viéndolo de otra forma, se está midiendo la **contribución del valor de cada elemento del vector al total del valor del vector** en tanto por uno, o coloquialmente hablando, su importancia o peso en el vector. Un resultado de un vector tras aplicar *softmax* puede ser el siguiente:

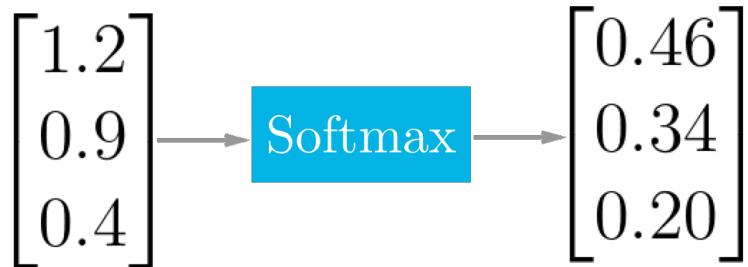


Figura 2.12: Función *softmax* aplicada a un vector de tres elementos.

Fuente: <https://towardsdatascience.com/deep-learning-concepts-part-1-ea0b14b234c8>

Podemos ver en la figura anterior que el vector final es un vector de probabilidades y que el elemento con mayor valor del vector antes de aplicar la función es aquel con mayor probabilidad tras aplicarla.

Por tanto, si añadimos un vector de dimensión i al final de nuestra red, donde i es exactamente el número de clases que nuestro problema tiene y queremos diferenciar en la clasificación, tras aplicar esta función tendremos un excelente clasificador ya que nos **devolverá la probabilidad de la imagen de pertenecer a esa clase i** y la red etiquetará a la imagen como la clase i , siendo i la posición del vector con mayor probabilidad.

2.3.3. Visualizando las convnets

Este esquema de 2 fases (extracción de características y clasificación) se sigue aplicando incluso a las ConvNets más actuales, pese a que muchas de las arquitecturas modernas incluyan nuevas técnicas de generación de bloques de características o usen funciones de activación más complejas; en este apartado se han cubierto los fundamentos básicos o pilares de esta metodología de aprendizaje, que se pueden observar todos juntos en la siguiente imagen:

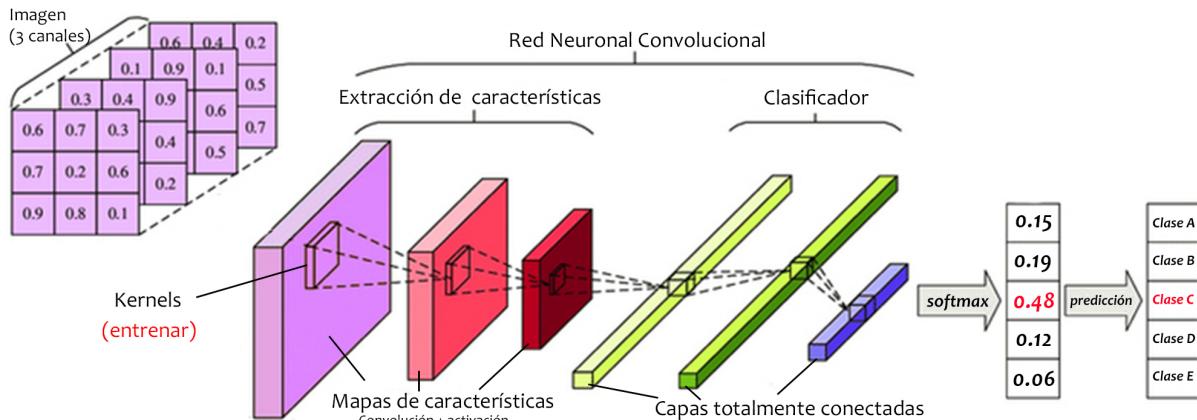


Figura 2.13: Ejemplo de una ConvNet utilizando los fundamentos teóricos vistos.

Fuente: Elaboración propia

En esta arquitectura de ejemplo podemos ver varias cosas. En primer lugar, la imagen (el dato de entrada) tiene una representación matemática muy sencilla: una matriz RGB de 3 canales (donde cada canal corresponde a los colores rojo, verde y azul, respectivamente). En cada valor de la matriz y por cada canal se almacena **la intensidad del píxel** en un valor que generalmente está en el rango [0, 255], siendo 255 la máxima intensidad posible.

Posteriormente, se generan nuevos mapas de características a través de la convolución con diversos **kernels** y aplicando la función de activación de manera conjunta y el bloque de características se transforma en un conjunto de capas totalmente conectadas, siendo la última aquella sobre la que se aplica la función *softmax* y obteniendo una predicción del dato de salida (marcada en rojo). Aquella clase con mayor probabilidad es la seleccionada por el modelo como la clase final.

Lo importante de ver este diagrama una vez entendidos estos conceptos es que la **incógnita** para obtener una buena clasificación sobre nuestros datos de entrada es fijar el valor de los **kernels** ya que el resto de operaciones son funciones matemáticas: la función

de activación es una función ya dada, el pooling es agrupar los píxeles y, por supuesto, disponemos del dato de entrada y su etiqueta asociada.

Los kernels son la herramienta que nos permite generar distintos mapas de características dado un dato: dependiendo del kernel usado en la operación de convolución, el resultado (el bloque de salida) **será totalmente distinto** y recordemos que ese resultado se propaga con otro kernel a la siguiente capa. Lo que se busca en el proceso de entrenamiento es en primer lugar encontrar los **kernel que minimicen el error de los datos que tenemos**, de manera que las predicciones que la red haga de nuestras imágenes sean precisas y coincidan con la etiqueta real de la imagen. Para ello es necesario kernels que encuentren mapas de características con información relevante de la imagen y que permitan llegar al clasificador con un criterio adecuado para que, al aplicar la función **softmax** sobre ese mapa de características, el resultado sea una predicción correcta. El segundo paso será ver si este proceso de **ajuste** realmente es un proceso de **aprendizaje**, y eso se comprobará en el comportamiento de la red ya entrenada sobre nuevos datos.

2.4. Entrenamiento de la red

En el apartado anterior hemos visto que lo que debemos de modificar son los kernel, los elementos encargados de extraer características de la imagen inicial y de los siguientes mapas de características hasta el clasificador, con el que obtendremos una predicción de la imagen. Pero, ¿cómo entrenamos los kernels?

El proceso de entrenamiento de una ConvNet se conoce como *backpropagation*. Se llama así ya que los **kernels** se actualizan desde la última capa, la más cercana al clasificador hacia atrás, **propagando el valor del gradiente de la función de error obtenida** a los kernel de capas anteriores. Este proceso tiene varios conceptos matemáticos detrás, pero aquí se expone la idea básica de cómo se actualizan y la simple idea de que es un proceso **iterativo y calculable paso a paso**, con el que acabaremos minimizando el error de la red. La idea se plasma en el siguiente pseudocódigo:

Algorithm 1 Algoritmo de *backpropagation* para problemas de clasificación múltiple

1. Inicializamos los pesos $w_1, w_2 \dots w_n$ de cada kernel de manera aleatoria o siguiendo alguna heurística.
2. Hasta cierto número de iteraciones (épocas):
 - a) Se selecciona un conjunto de N datos (imágenes) y se pasan por la red hasta obtener un valor de la etiqueta predicha por la red, x_i para cada imagen en concreto.
 - b) Se calcula el error de entropía cruzada categórica (para problemas de clasificación múltiple, como es el caso) de la siguiente manera:

$$-\sum_{i \in numclases}^{M} y_{true} \log(y_pred_i)$$

Donde y_{true} es el valor de la etiqueta verdadera de cada imagen en codificación *one-hot* y y_pred el valor predicho por la capa softmax **el vector de probabilidades para esa etiqueta en cuestión**. El error es la sumatoria de las N imágenes que se procesan en cada iteración (en otras palabras, cada conjunto de N imágenes es un *batch*, siendo el tamaño del *batch* un parámetro que se puede cambiar).

- c) Se actualizan los pesos w_i utilizando el valor anterior usando reglas derivativas para alterar los pesos de los kernels de cada capa. Este apartado contiene términos matemáticos que no son relevantes de cara al proceso de investigación, por lo que si se desea saber más de lo que se está realizando de forma 'oculta' se puede consultar el siguiente enlace:
 - d) Se comprueba si se ha alcanzado el número de iteraciones máximas. En caso contrario, se vuelven a recorrer todos los datos de nuevo.
-

Este proceso acaba ajustando los pesos de los **kernels** para que la red evite fallar con los datos de entrenamiento. Más tarde veremos que no siempre una red bien entrenada con datos de entrenamiento es capaz de generalizar bien ante datos nuevos, que realmente es lo que nos interesa, por lo que será necesario realizar un proceso de **validación** junto al entrenamiento para asegurarnos de ello.

En nuestro caso, la función de error con la que el algoritmo de *backpropagation* ha sido la de entropía cruzada categórica debido a que está diseñada para problemas de clasificación múltiple, pero se puede cambiar dependiendo del problema; el proceso de optimización es el mismo siempre.

2.5. Transfer Learning y Fine Tuning

Entrenar una red neuronal convolucional puede ser un proceso bastante costoso en términos de tiempo y de cómputo; muchas de las arquitecturas usadas actualmente tienen **millones de parámetros** que hay que entrenar a lo largo de toda la arquitectura de la

red. La pregunta que nos podemos hacer es: ¿Hay alguna manera de no tener que entrenar una arquitectura tan compleja como una red neuronal convolucional y 'aprovechar' el proceso de aprendizaje de otra red ya entrenada para adaptarlo a nuestro problema, transmitiendo parte del conocimiento de la red al problema nuevo? La respuesta es sí.

2.5.1. Transfer learning (extracción de características fija)

El *transfer learning* consiste en, como el nombre dice, 'transferir' el conocimiento de una red ya preentrenada con un conjunto de datos grandes, por lo general en una tarea de clasificación de imágenes a gran escala. Si el conjunto de datos con el que la red ha sido entrenada es lo suficientemente grande y general, entonces las características que aprenderá esa red preentrenada puede actuar como una especie de 'extractor de características genérico' y sus características pueden ser útiles para muchos problemas de Visión por Computador, aunque estos problemas cuenten con clases totalmente diferentes a la tarea original.

Lo que generalmente se suele hacer (y haremos en la práctica) es utilizar redes preentrenadas con la base de datos de *ImageNet* [poner link](#), donde la mayoría de clases son animales y objetos cotidianos, y después cambiar la parte final de la red (**el clasificador**) para adaptarlo al tamaño de nuestro problema y, en general, adaptar la red a nuestra tarea. *ImageNet* se utiliza como *Benchmark* para las redes del estado del arte ya que consta de 1000 clases (es un problema de clasificación múltiple bastante complejo) y de 1.4 millones de imágenes etiquetadas.

Veremos si entrenar únicamente el clasificador de las redes del estado del arte (EfficientNet y DenseNet) es suficiente para solucionar nuestro problema, aunque podemos esperar que no sea así debido a que las clases de imágenes de *ImageNet* son muy genéricas y nuestro problema es un problema muy específico.

2.5.2. Fine-Tuning

El *fine tuning* consiste en dar un paso más al transfer learning. En el *fine tuning* también se usa una red preentrenada para otro problema, pero ahora no se descongela únicamente el clasificador sino que, en primer lugar, se descongelan las capas correspondientes al clasificador (como en el paso anterior) pero una vez esta capa ha sido reentrenada para adaptarse a nuestro problema, se descongelan progresivamente partes de la red (**los kernels**) de manera que poco a poco estos kernels se van alterando para extraer características explícitas de nuestro problema. La diferencia principal es que dejamos que la red se adapte un poco más a nuestro problema en cuestión ya que los **kernels** se modifican levemente para extraer características más 'útiles' para nuestro problema.

Probaremos realizando *fine tuning* progresivo con varias de las redes del estado del arte para ver hasta qué punto es necesario adaptar una arquitectura preentrenada a nuestro problema.

2.5.3. Reentrenamiento completo

En este caso se utilizan la arquitecturas del estado del arte pero se **reentrenan desde cero**, aprovechando la topología de la red en cuestión pero no utilizando los pesos preentrenados, sino entrenando la red al completo desde un inicio. Este caso se aplica cuando

los pasos anteriores no han sido suficientes para obtener un buen ajuste con los nuevos datos y es necesario adaptar toda la arquitectura al completo a nuestro problema.

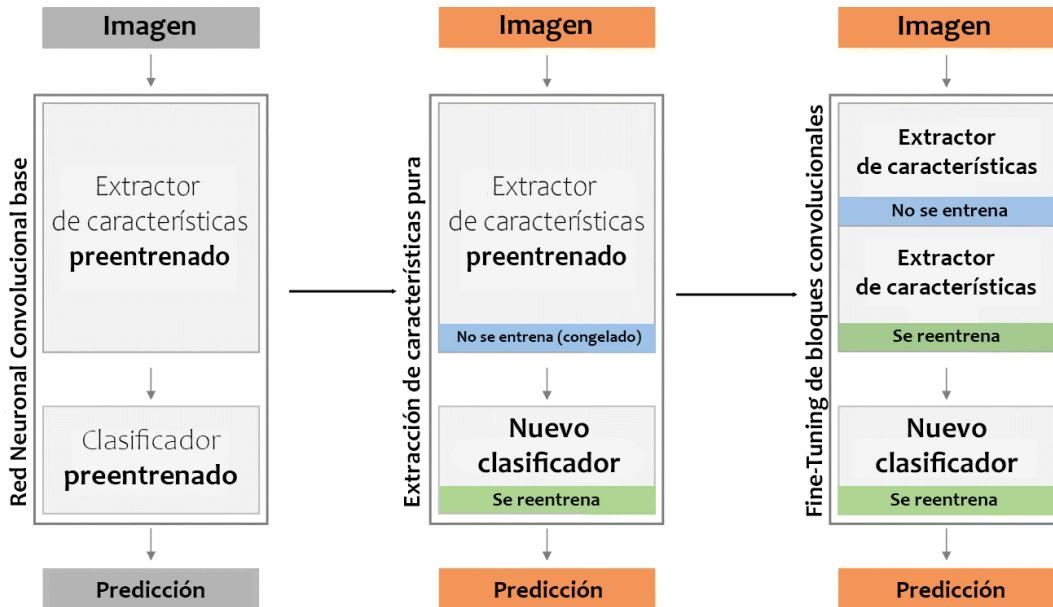


Figura 2.14: Esquema de *transfer learning* y *fine tuning*.

En la última fase, se puede descongelar la arquitectura de la red al completo, si es necesario.

Fuente: Elaboración propia.

2.6. Limitaciones del *deep learning*

Las Redes Neuronales Convolucionales son arquitecturas que han provocado el auge del *deep learning*, y utilizadas actualmente como estado del arte para tareas de Visión por Computador. Sin embargo, existen algunas limitaciones que hay que tener en cuenta cuando trabajamos con ellas:

- Requieren una gran cantidad de datos para obtener buenas precisiones, lo que implica que la construcción de la base de datos sea un proceso caro y complejo, ya que estas construcciones suelen ser muy manuales; aun así, existen técnicas para ayudarnos en el proceso de obtención de datos, como veremos en el apartado correspondiente.
- Los modelos no son interpretables, es decir, sabemos que matemáticamente los pesos se hallan de manera que la función de error se minimice con nuestros datos, pero sería muy difícil explicar a una persona **qué criterio está tomando la red para decidir que una imagen en concreto pertenece a una clase en concreto**, debido a que las redes cuentan con cientos de miles de neuronas interconectadas entre sí, que se unen para tomar una decisión final.

La búsqueda de la interpretabilidad de las redes es un campo de estudio actual en el que muchos investigadores se encuentran. Actualmente se suelen aplicar técnicas visuales que intentan graficar qué es lo que hay dentro de la red neuronal o qué es lo que tiene en cuenta cada neurona para tomar la decisión final. Una de estas se

denomina **LIME** y consiste en primer lugar en dividir la imagen en trozos. Esta división se hace teniendo en cuenta que todos los píxeles de un trozo tienen que tener el mismo valor. Después, va eligiendo de forma aleatoria una serie de trozos y lo pasa por el modelo para ver si predice correctamente. Este proceso se hace varias veces hasta que se pueda ver qué segmentos de la imagen son más importantes a la hora de hacer la predicción.

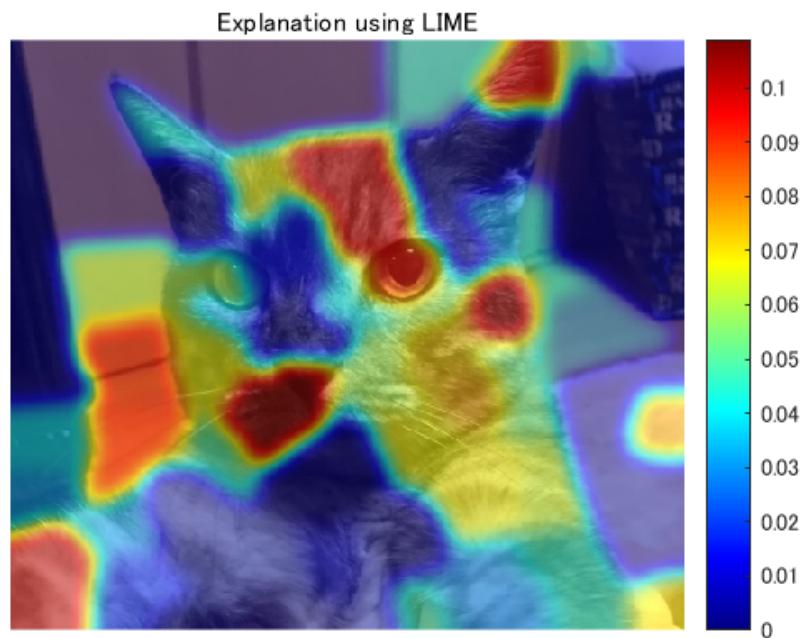


Figura 2.15: Uso del algoritmo LIME para detectar qué píxeles han dado información a la red para clasificar la imagen.

Fuente: añadir

3 | Estado del arte

En este apartado se comentarán las técnicas actuales de clasificación de diatomas que se realizan utilizando el *deep learning* tras ver todos los avances en este campo y posteriormente se presentarán las últimas arquitecturas de Redes Neuronales Convolucionales con las que se probará a automatizar este problema.

3.1. Clasificación manual de Diatomeas

La clasificación manual de Diatomeas es una tendencia que actualmente está a la baja, pero que hace menos de diez años era el único método posible para clasificar las Diatomeas debido a la inexistencia de modelos de clasificación automática y menos de modelos de machine learning.

El método convencional de clasificación es analizar cada diatoma de manera individual, extrayendo sus *key features* (generalmente suelen ser características morfométricas) y viendo el parecido con las *key features* de cada clase. Se ha comprobado en experimentos como el realizado en [14] que la habilidad de cada experto para identificar correctamente un Diatomea de una lista de Diatomeas dada no es del todo exacta, sino que se sitúa alrededor del 80 % de elementos bien clasificados, sufriendo sobre todo las Diatomeas de las clases menos conocidas. Esta técnica además es sensible a la luz del microscopio y muchos de ellos son transparentes, por lo que algunos de los detalles son muy poco visibles, necesitando la aplicación de técnicas de postprocesado como el DIC o el *phase contrast* para poder ver todas las partes de la Diatomea de manera previa a la extracción de características.

Este tipo de clasificación tiene varios inconvenientes añadidos a los anteriores, como el tener que realizar el proceso de nuevo cuando se quiere añadir una clase más, se necesitan expertos para delimitar las características que delimitan a cada Diatomea (las *key features*) y, sobre todo, la cantidad de tiempo necesario para realizar este trabajo, como ya comentamos anteriormente.

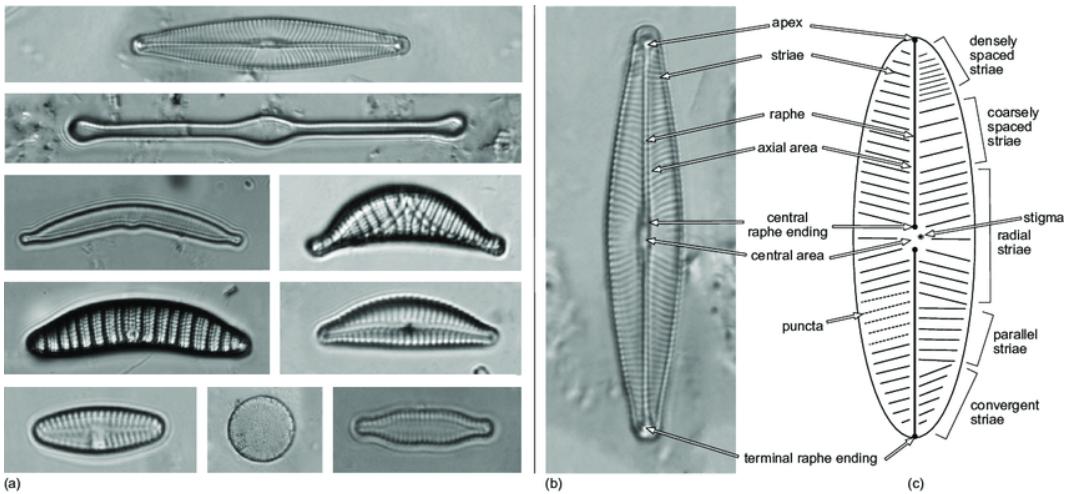


Figura 3.1: Características principales utilizadas para la clasificación manual de Diatomeas.

Fuente: [15].

3.2. Clasificación automática de Diatomeas

Dada la complejidad de la tarea manual, se han diseñado métodos de clasificación de Diatomeas de forma automática y casi todos ellos siguen la misma línea de trabajo. En este apartado se verá uno los métodos pero si se desea saber más, se pueden consultar artículos como [16] o [15] donde se habla con más tranquilidad de los procesos en general; destacar también otras técnicas muy populares como los filtros holográficos [17] o la decomposición armónica [18], técnicas que automatizan el proceso pero sin utilizar *machine learning*.

El método automático que se comentará ahora basa el fundamento en utilizar un software de procesamiento de imagen llamado 'ImageJ' para hacer una búsqueda de similitud del diatoma con los diatomas de la base de datos. Para realizar esta búsqueda de similitud, el sistema obtiene las características del Diatomea, que son el *outline* y la ornamentación. Después de obtener estas características busca la clase con más parentesco en estas dos características, fijando una Diatomea previamente por clase como el mejor representante de la clase que se compararán con nuestro Diatomea que queremos clasificar. Aquel representante que más se parezca a la Diatomea indicará que esa Diatomea pertenece a la clase del representante.

3.2.1. Obtención del *outline*

Este primer paso reduce bastante el número de candidatos que analizar, y se obtiene el 'esqueleto' de la Diatomea tras un proceso de binarización, relleno y el final filtro de Sobel para obtener el *outline*. Este *outline* se transforma en un vector contando el número de dígitos pares e impares y obteniendo un número. Ese número es el que definirá el *outline* de la Diatomea.

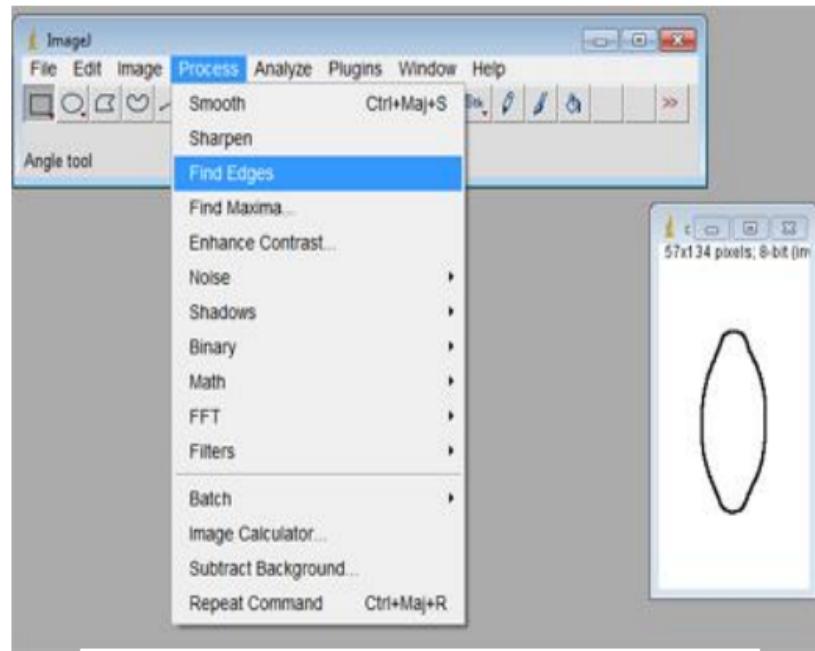


Figura 3.2: Extracción del *outline* de la Diatomea en ImageJ.

Fuente: [19].

3.2.2. Extracción de la ornamentación

En este paso se obtiene información acerca del interior de la Diatomea y de los patrones de dentro de ella. Tras pasar la imagen por un umbral de intensidad y realizar una operación de sustracción del *outline*, obtenemos una imagen con información únicamente del interior de la Diatomea, y por tanto, de su ornamentación.

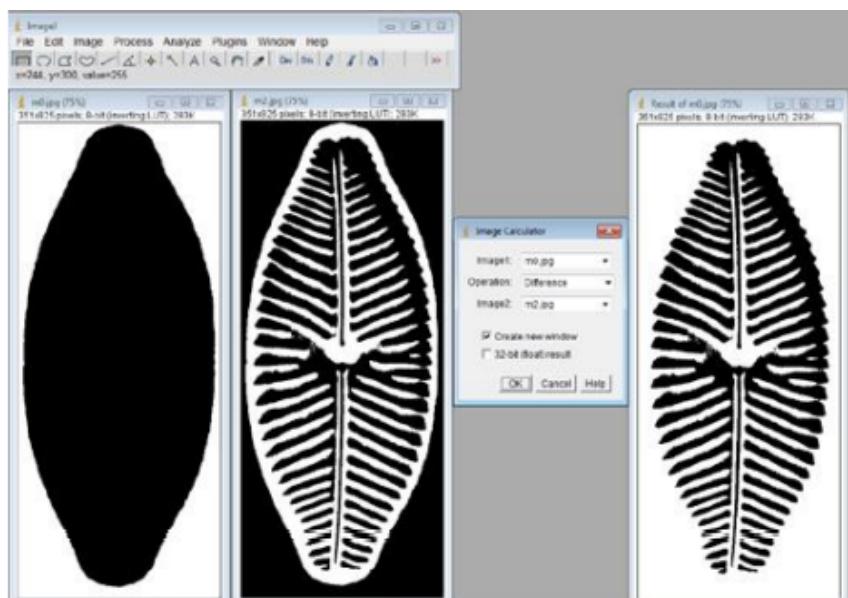


Figura 3.3: Extracción de la ornamentación de la Diatomea en ImageJ

Fuente: [19].

3.2.3. Detección de los *ultimate points* de la ornamentación

En este paso extraemos los ultimate points de la ornamentación de la Diatomea, que son aquellas zonas que sintetizan la información de las características más importantes de la ornamentación y permiten, a la hora de la comparación, no contar en consideración todos los píxeles del interior de la diatomea sino las zonas representativas de la Diatomea, evitando realizar cálculos con 100.000 píxeles en una imagen de 100x100 y pasar a muchos menos píxeles (por ejemplo, 100) manteniendo la información principal para el siguiente paso.

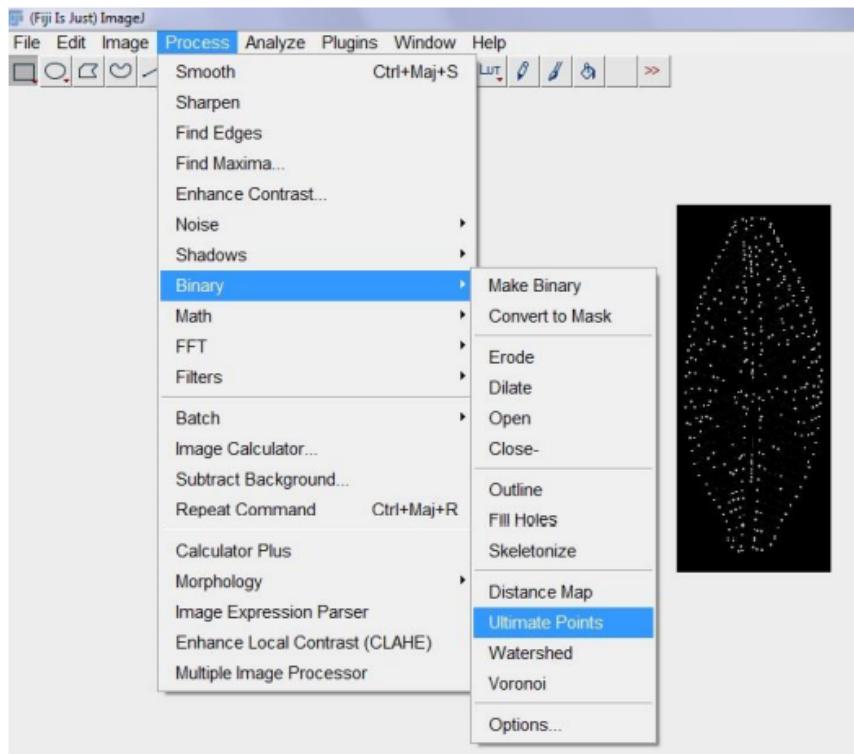


Figura 3.4: Extracción de los *ultimate points* de la Diatomea en ImageJ

Fuente: [19].

3.2.4. Comparación con diatomeas de la base de datos

En este paso, una vez se ha obtenido el *outline* (exterior) y la ornamentación (interior) del diatoma y se ha sintetizado la información en los ultimate points, se comparan estos valores con los valores de una diatomea representativa de la clase que tengamos almacenado, uno por cada clase. De esta manera, la diatomea representativa de la clase que más se parezca numéricamente en valores a la diatomea analizada indicará la clase de nuestro diatomea.

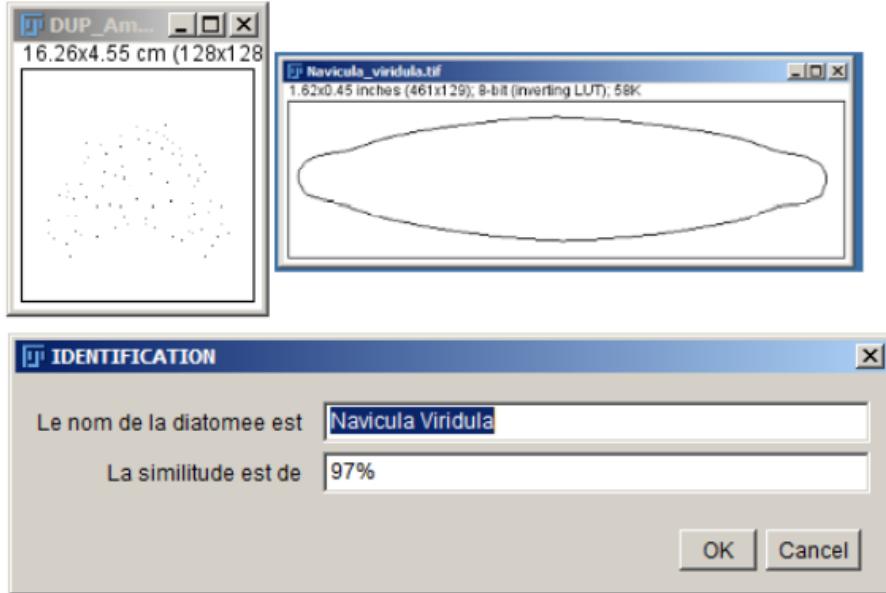


Figura 3.5: Comparativa de la Diatomea con los diatomeas candidatos en ImageJ

Fuente: [19].

Este proceso de clasificación al completo se puede ver de manera más detallada en el artículo original [19].

Como podemos apreciar, este tipo de procesos automáticos no tienen en cuenta las características a bajo nivel de las Diatomeas, por lo que suelen fallar a la hora de clasificar Diatomeas poco conocidas o de las que no se posee mucha información. A raíz de estos procesos surgió el primer proceso de clasificación automática de Diatomeas utilizando Machine Learning, conocido como ADIAC.

3.3. ADIAC como el estudio piloto de clasificación usando *Machine Learning*

ADIAC (*Automatic Diatom Identification and Classification*) [20] fue el primer proyecto dedicado puramente a aplicar algoritmos de *machine learning* para identificar y clasificar Diatomeas. Este proyecto se centró en construir una base de datos de Diatomeas grande y bien etiquetada, y se tomaron descriptores morfométricos de los Diatomas como extractores de características para construir finalmente modelos de *machine learning* usando esas características. Consiguieron reunir 4700 imágenes de 500 subespecies de Diatomeas y obtener alrededor de un 96.7 % de porcentaje de aciertos con un algoritmo llamado *random forest*; necesitaron utilizar 321 descriptores matemáticos de los Diatomeas para diferenciar las clases. Con ADIAC se demostró que no sólo era posible automatizar un modelo de clasificación, sino obtener mejores resultados que una clasificación realizada por expertos e incluso ahorrar un tiempo enorme de trabajo; los trabajos que continuaron a ADIAC siguieron utilizando estos descriptores morfométricos, así como otro tipo de extractores de características diseñados para el problema, pero se seguían utilizando características prediseñadas que generaban varios problemas: en primer lugar, si se quiere añadir una Diatomea nueva a la clasificación suele ser necesario añadir más extractores de características y definir la clase matemáticamente para que esa Diatomea pueda ser detectada y

clasificada correctamente. Por tanto, el proceso no es del todo automático. **El deep learning entra en escena: no hacen falta definir las características que diferencian a cada clase puesto que estas se obtienen en el proceso de aprendizaje.**

El avance de estos últimos años en el poder computacional ha abierto nuevas posibilidades a la clasificación y detección de cualquier clase de objetos mediante las *ConvNets*. Esto antes apenas era viable, debido a que estas arquitecturas suelen ser muy precisas y cómodas pero requieren de millones de parámetros para entrenar y el proceso suele ser costoso. El auge de las tarjetas gráficas (GPU) y las mejoras en memorias digitales han hecho que seamos capaces de almacenar y entrenar estos modelos con un tiempo relativamente bajo para los resultados obtenidos. Este enfoque es el que seguimos en el proyecto, presentando a continuación unas de las *ConvNets* más avanzadas de todos los tiempos.

3.4. *Deep Learning* para la clasificación de Diatomeas

El *dataset* de ImageNet [citar](#) contiene 14.197.122 imágenes etiquetadas de 1000 clases diferentes, generalmente de animales y objetos cotidianos. Desde 2010, este *dataset* es usado en el ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*), un *benchmark* en la clasificación y detección de objetos en imágenes. Por tanto, cuando una arquitectura nueva surge, se entrena y se evalúa su rendimiento contra las mejores en esta tarea de clasificación de 1000 clases; en caso de ser mejor, es declarada como la red del *estado del arte*. A continuación se explican las características fundamentales de la familia de Redes Neuronales Convolucionales ganadoras actualmente en esta tarea de clasificación de ImageNet que, además de incorporar nuevas técnicas que hacen que se aprovechen todos los parámetros de la red con mucha eficiencia, utiliza nuevas técnicas que le dan, en el momento de redacción de este trabajo, el puesto número 1 en clasificación de imágenes. Hablamos de la familia de redes de EfficientNet y su versión actualizada EfficientNetV2, redes que se utilizarán en esta tarea de clasificación de Diatomeas y se evaluará su desempeño. Veamos qué tienen.

3.4.1. EfficientNet

Las *ConvNets* de la familia de EfficientNet surgen alrededor de mayo de 2019 de mano de dos ingenieros del equipo de Google Brain Team, llamados MingXing Tan y Quoc V. Ambos publicaron un paper llamado “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks” [citar](#). La idea principal de esta publicación es sencilla: **escalar las *ConvNets* usando una estrategia en concreto** para obtener buenos resultados. Las redes de la familia EfficientNet son muy eficientes computacionalmente y también consiguieron el resultado de *estado del arte* en el *dataset* de ImageNet con una *accuracy* (porcentaje de aciertos) del 84.4 %.

En primer lugar, hablaremos de la estrategia principal y diferenciadora de estas redes: el *escalado compuesto*.

Escalado compuesto (compound scaling)

El escalado de las *ConvNets* es algo que se lleva haciendo mucho tiempo y consiste en, como la palabra dice, escalar el modelo en términos de anchura (mapas de características más anchos), profundidad (añadir más mapas de características) o resolución de imagen

de entrada (se aumenta la resolución de la imagen de entrada para captar más detalles). El escalado en profundidad es el más popular; redes como *ResNet* pueden ser escaladas usando este criterio desde *ResNet18* hasta *ResNet200*, pasando de 18 bloques convolucionales hasta 200.

Si medimos el rendimiento de *ResNet18* y *ResNet200*, está claro que la última es la ganadora demostrando que el escalado en profundidad funciona para mejorar la calidad de nuestros resultados, pero hay un problema fundamental: **El criterio manual de escalado no mejora el rendimiento llegado hasta cierto punto**. Esto es, si seguimos aumentando la profundidad de la red, puede que obtengamos incluso peores resultados.

El planteamiento de estos ingenieros fue el estudiar y rediseñar un método de escalado que permita evitar el degradamiento de la red conforme se escala y a la vez obtener buenos resultados en términos de eficiencia y accuracy. Para conseguir esto, se hizo un estudio empírico donde se demostró que, para conseguir esto es **crítico balancear todas las dimensiones (anchura, profundidad y resolución de imagen de entrada) de manera conjunta en vez de escalar una dimensión de las tres**. Basándose en esta observación, se propuso un método de escalado compuesto o compound scaling que escala uniformemente las dimensiones de la red para obtener mejores resultados. A priori tiene sentido, porque a más tamaño de la imagen de entrada, la red necesita más capas para aumentar el campo receptivo y más canales para detectar patrones en la imagen.

El *compound scaling* utiliza un coeficiente (le llamaremos ϕ) que es especificado por el usuario y se usa para escalar la anchura, profundidad y resolución de imagen de entrada de manera conjunta. En la imagen inferior está la fórmula con la que se escalan las dimensiones usando ϕ :

$$\begin{aligned} \text{depth: } d &= \alpha^\phi \\ \text{width: } w &= \beta^\phi \\ \text{resolution: } r &= \gamma^\phi \\ \text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\ \alpha \geq 1, \beta \geq 1, \gamma \geq 1 & \end{aligned}$$

Figura 3.6: Fórmula para modificar la profundidad, anchura y resolución de la red usando ϕ en el *compound scaling*.

Fuente: [21]

Nos podemos preguntar viendo la fórmula anterior por qué en la restricción no se fija α^2 al igual que las otras dos variables y por qué se restringe el producto de estas tres variables a 2; la respuesta es debido a que los FLOPS (una medida de la cantidad de cálculos que se han de realizar) de una convolución es proporcional a profundidad de manera lineal pero es proporcional de manera cuadrática a la profundidad y la resolución, por lo que duplicando la profundidad se duplicarán los FLOPS mientras que duplicando la resolución o la profundidad hará que los FLOPS aumenten casi cuatro veces más. De esa manera,

al crear la restricción nos aseguramos que los FLOPS no excedan un valor superior a 2 y además penalizamos más el aumento en resolución y anchura, debido al coste que tiene en los FLOPS. Esta restricción ha permitido crear modelos eficientes.

También se puede apreciar que alpha, beta, y gamma son los multiplicadores de escalado para la profundidad, anchura y resolución respectivamente y éstos se obtienen usando una búsqueda empírica para obtener los mejores resultados de estos valores. Visualmente, el escalado compuesto en comparación a los escalados arbitrarios y unidimensionales es la siguiente:

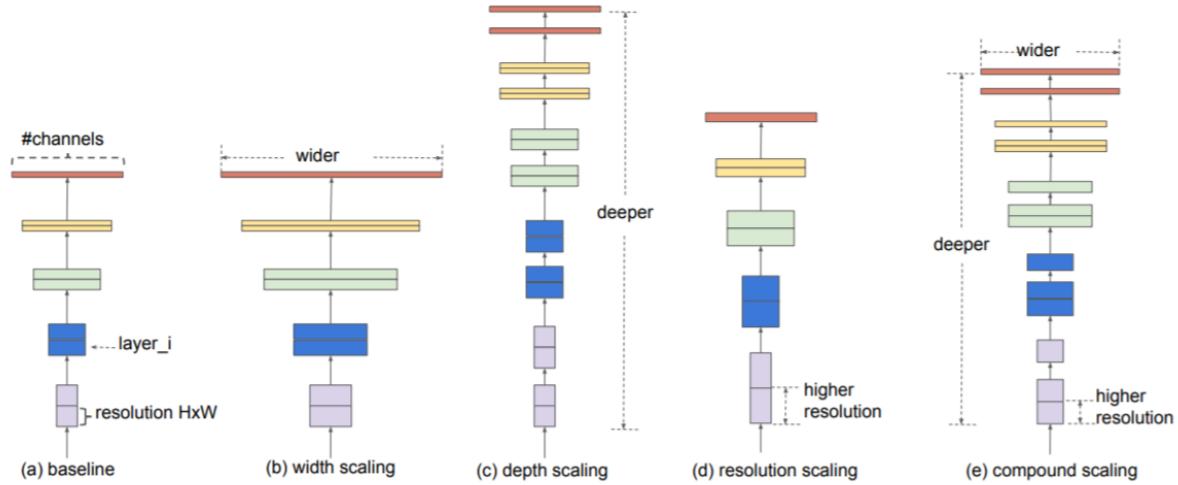


Figura 3.7: Método de *compound scaling* en comparación a escalados unidimensionales de la arquitectura.

Fuente: [21]

Usando este método de escalado compuesto en redes como MobileNet o ResNet, se obtuvieron mejoras bastante sustanciales en ambas arquitecturas y se comprobó que este escalado puede ser utilizado en cualquier ConvNet obteniendo buenos resultados:

Model	FLOPS	Top-1 Acc.
Baseline MobileNetV1 (Howard et al., 2017)	0.6B	70.6%
Scale MobileNetV1 by width ($w=2$)	2.2B	74.2%
Scale MobileNetV1 by resolution ($r=2$)	2.2B	72.7%
compound scale ($d=1.4$, $w=1.2$, $r=1.3$)	2.3B	75.6%
Baseline MobileNetV2 (Sandler et al., 2018)	0.3B	72.0%
Scale MobileNetV2 by depth ($d=4$)	1.2B	76.8%
Scale MobileNetV2 by width ($w=2$)	1.1B	76.4%
Scale MobileNetV2 by resolution ($r=2$)	1.2B	74.8%
MobileNetV2 compound scale	1.3B	77.4%
Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth ($d=4$)	16.2B	78.1%
Scale ResNet-50 by width ($w=2$)	14.7B	77.7%
Scale ResNet-50 by resolution ($r=2$)	16.4B	77.5%
ResNet-50 compound scale	16.7B	78.8%

Figura 3.8: Resultados del *compound scaling* comparando tres modelos base y escalados en una dimensión con los mismos modelos tras aplicar este escalado compuesto.

Fuente: [21]

Podemos ver que en las tres arquitecturas con las que se probó este escalado (MobileNetV1, MobileNetV2 y ResNet50) se consiguieron mejores resultados haciendo *compound scaling* que en los modelos base y escalados con una dimensión.

Por curiosidad, se puede ver cómo afectan los escalados en la búsqueda de las regiones donde la red se fija para tomar una decisión acerca de la imagen:

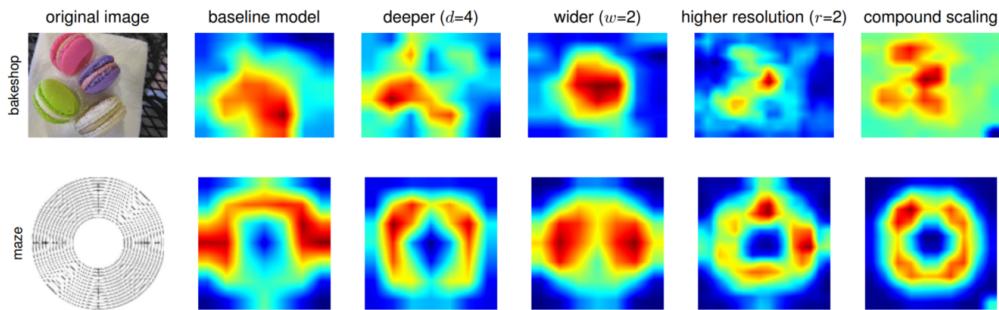


Figura 3.9: Mapas de activación de clases (CAM) de dos imágenes con un modelo base y tras aplicar diferentes tipos de escalado. Cuando se usa *compound scaling* la red es capaz de apreciar mejor la forma original del objeto.

Fuente: [21]

Pero en este estudio también se vió que el escalado afecta de manera diferente dependiendo del modelo base usado, por lo que la segunda parte de este proceso de investigación fue buscar aquella ConvNet base que al escalararse obtuviese los mejores resultados (con ImageNet). De esa manera surgió **EfficientNetB0**.

EfficientNetB0 como baseline para la familia EfficientNet

Como el escalado no afecta las operaciones que realiza la red, se comprobó que era mejor tener un buen modelo base y escalarlo usando esta nueva técnica para tener un modelo

que no solo fuese eficiente y preciso, sino que se pudiese escalar multiplicándolo por un factor ϕ y generar arquitecturas escaladas para solucionar problemas más complejos.

EfficientNetB0 fue una red obtenida mediante un método llamado NAS (Neural Architecture Search) que optimiza tanto la *accuracy* del modelo y los FLOPS (en definitiva, el tiempo que tarda el modelo), haciendo un compendio entre rendimiento y eficiencia. La arquitectura es la siguiente:

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels \hat{C}_i	#Layers \hat{L}_i
1	Conv3x3	224×224	32	1
2	MBCConv1, k3x3	112×112	16	1
3	MBCConv6, k3x3	112×112	24	2
4	MBCConv6, k5x5	56×56	40	2
5	MBCConv6, k3x3	28×28	80	3
6	MBCConv6, k5x5	28×28	112	3
7	MBCConv6, k5x5	14×14	192	4
8	MBCConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figura 3.10: Baseline obtenida en el proceso de búsqueda de la red que se comporta mejor con el *compound scaling* (Finalmente llamada EfficientNetB0)

Fuente: [21]

Una vez obtenida la baseline, podemos buscar valores óptimos para los parámetros del escalado. Si volvemos a ver la figura 3.6, podemos ver que fijando un ϕ , los valores de alpha, beta y gamma se pueden obtener mediante una búsqueda exhaustiva, y así se hizo. Para EfficientNetB0 se fijó un $\phi = 1$ y se obtuvieron unos valores de $\alpha = 1.2$, $\beta = 1.1$ y $\gamma = 1.15$; para crear las redes superiores (desde EfficientNetB1 a EfficientnetB7) se fijaron estos valores de alpha, beta y gamma y se experimentaron con distintos valores de ϕ , generando modelos de anchura, profundidad y resolución escalada progresiva. En la siguiente figura se puede apreciar la ganancia en rendimiento y en eficiencia de esta familia de redes en comparación con las redes más famosas que había anteriormente.

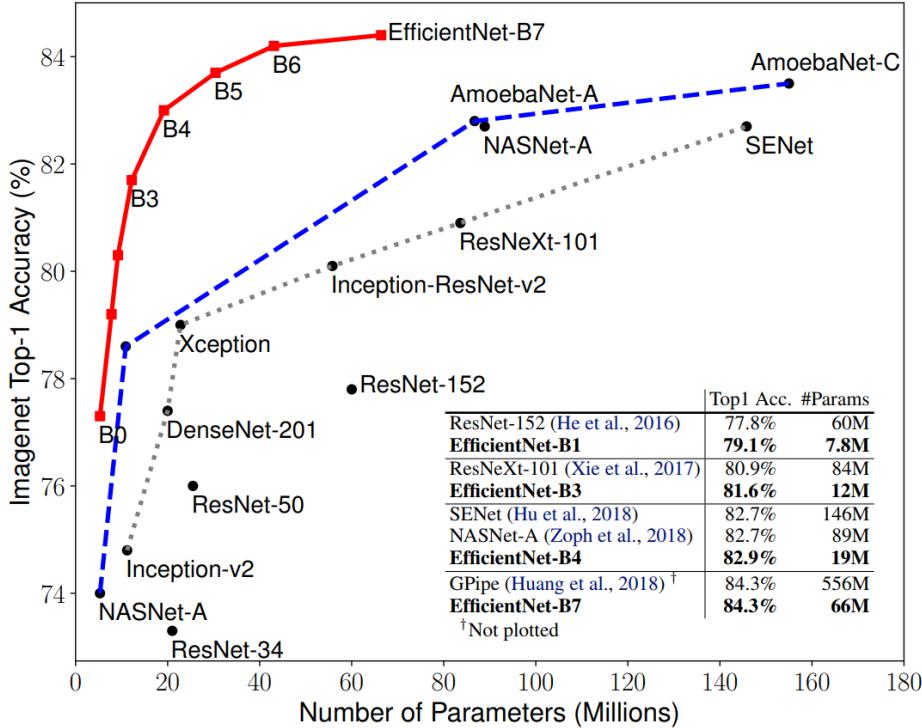


Figura 3.11: Comparativa de los modelos de la familia EfficientNet con modelos del estado del arte. Se puede apreciar, tanto gráficamente como en la tabla, una mejora en el rendimiento y en la eficiencia.

Fuente: [21]

Se puede ver en la figura 3.11 con esta familia de redes se ha conseguido mejorar la tarea de clasificación de imágenes tanto en rendimiento (más accuracy) como en eficiencia (para la misma accuracy se necesitan menos número de parámetros que otras redes como DenseNet, casi 10 veces menos en general).

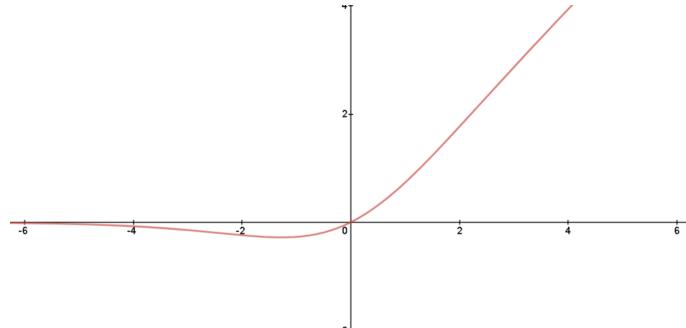
Función de activación *Swish*

En el apartado teórico vimos que la función de activación generalmente usada para estos modelos es la función ReLU, que funciona generalmente bien. Esta función transforma los valores negativos a cero, por lo que la derivada de todos los valores negativos también es cero. Frente a este inconveniente surgieron varias funciones de activación que hacían que los valores negativos tuvieran un valor, por pequeño que sea, para que el gradiente de todos los valores negativos no valiese lo mismo; la función Leaky ReLU corrige este problema.

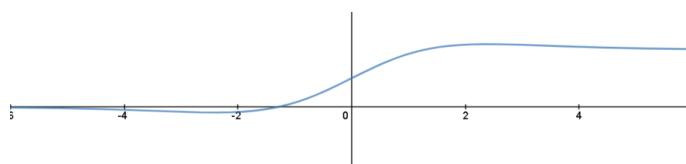
El equipo de Google Brain añadió a esta familia de redes una nueva función de activación que demostró funcionar mejor para las redes del estado del arte, que generalmente suelen ser bastante profundas: la función *Swish*. Con un simple cambio en una línea de código (sustituyendo la llamada de ReLU por *Swish*) consiguieron un 0,6 % más de *accuracy* en el *dataset* de ImageNet. Esta función *Swish* es igual de sencilla que el resto; consiste en la multiplicación de la función *sigmoide* por un coeficiente *x* de la siguiente manera:

$$\text{swish}(x) = x * \text{sigmoid}(x)$$

En la siguiente imagen se muestra el comportamiento gráfico de la función *Swish* y su gradiente, comprobando que, efectivamente, el gradiente es mucho menos brusco para valores negativos que en funciones como la ReLU.



(a) Comportamiento de la función *Swish*



(b) Comportamiento del gradiente de la función *Swish*

Figura 3.12: Comportamiento de la función de activación *Swish* y su gradiente. Fuente: [21]

Si se desea consultar más información acerca de estas arquitecturas en detalle, se puede consultar el *paper* oficial [21].

3.4.2. EfficientNetV2

Tras unos meses de la creación de las redes EfficientNet, surgió una versión actualizada de éstas llamada EfficientNetV2, que incorporó varias mejoras afectando tanto el rendimiento de estas redes como la velocidad de cálculo, por lo que el estado del arte de las ConvNets fue superado por su propia familia de redes. En este apartado hablaremos de la familia de redes EfficientNetV2 y comentaremos las diferencias principales respecto a la versión inicial de éstas.

Tamaños de imagen pequeños en entrenamiento

Es fácil observar que tamaños grandes de imagen resultan en un gran consumo de memoria; generalmente los requisitos de GPU son escasos o son limitantes en el proceso de aprendizaje y los modelos que reciben tamaños de imagen de entrada grandes necesitan ser entrenados con *batches* más pequeños. Recordemos que un *batch* es el conjunto de imágenes que se pasan al algoritmo en cada iteración para entrenar; a mayor tamaño de *batch*, más rápido es el entrenamiento, pero más carga de imágenes en memoria.

Una simple mejora que se propone en [22] es entrenar en tamaños de imagen pequeños y evaluar el modelo en tamaños de imagen más grandes, sin perder apenas *accuracy* y obteniendo modelos que entran más rápido y que generalizan prácticamente igual/

Tamaños de imagen pequeños en entrenamiento

Por otro lado, las redes de la familia EfficientNetV1 utilizaban convoluciones bastante profundas, que generalmente tienen menos parámetros y FLOPS que las convoluciones regulares, pero no pueden utilizar al completo las técnicas de aceleración del aprendizaje. **Aquí se demostró que un número inferior de FLOPS no tiene por qué significar una mayor velocidad en el aprendizaje.**

Cambio en los bloques convolucionales

Para EfficientNetV2 se propuso una convolución llamada Fused-MBConv que sustituye la convolución 3x3 y la expansión 1x1 realizada en EfficientNetV1 por una convolución sencilla 3x3; en la siguiente imagen se comparan los bloques de convolución antiguos (MBConv sin fusionar) y la aplicación de estos bloques sobre la red gradualmente:

	Params (M)	FLOPs (B)	Top-1 Acc.	TPU imgs/sec/core	V100 imgs/sec/gpu
No fused	19.3	4.5	82.8%	262	155
Fused stage1-3	20.0	7.5	83.1%	362	216
Fused stage1-5	43.4	21.3	83.1%	327	223
Fused stage1-7	132.0	34.4	81.7%	254	206

Figura 3.13: Sustitución del bloque convolucional MBConv (EfficientNetV1) por Fused-MBConv (EfficientNetV2) progresivamente en los bloques de la red.

Fuente: [23]

Como podemos ver en la figura 3.14, cuando se sustituyen los bloques en los primeros bloques de la red, se obtiene una mejora en la velocidad de la red a cambio de más parámetros y FLOPS. Sin embargo, si se aplica en todos los bloques de la red, los parámetros aumentan mucho más y la *accuracy* disminuye en vez de aumentar. Podemos ver que no hay un criterio exacto de cuándo usar estos bloques convolucionales, así que se hizo una búsqueda exhaustiva de la mejor combinación para la baseline de EfficientNetB0.

Compound scaling más complejo

La arquitectura de EfficientNetV1 escalaba todas las dimensiones de la red de forma uniforme usando una regla de escalado compuesta muy sencilla que hace que no afecten de forma equitativa a la eficiencia y rendimiento del modelo. Para EfficientNetV2, los autores una regla de escalado no uniforme para gradualmente añadir más capas conforme se avanza en profundidad (más en [23]).

Resizing progresivo de la imagen

El *resizing* progresivo consiste en gradualmente incrementar el tamaño de la imagen de entrada conforme el entrenamiento avanza y el modelo mejora. Por ejemplo, si queremos entrenar un modelo fijando 10 épocas de duración, comenzamos con un tamaño de entrada de 224x224 y posteriormente ajustamos los pesos en las 4 últimas épocas reajustando el tamaño de la imagen a un tamaño superior, por ejemplo 256x256.

No obstante se plantearon que, al igual que en modelos más grandes de la familia EfficientNet hay más regularización para combatir el *overfitting* (EfficientNetB7 tiene mucho más *dropout* y aumento de datos que EfficientNetB0), **también debe de combatirse este problema dentro de la misma red**. Fijando una red en concreto, menor tamaño de imagen hace modelos más pequeños y se necesita menos regularización; viceversa ocurre lo mismo, tamaños de imagen más grandes hacen modelos más grandes y cuya regularización debe de ser mayor. Esto que comentamos se demostró experimentalmente, obteniendo la siguiente tabla:

	Size=128	Size=192	Size=300
RandAug magnitude=5	78.3 ±0.16	81.2 ±0.06	82.5 ±0.05
RandAug magnitude=10	78.0 ±0.08	81.6 ±0.08	82.7 ±0.08
RandAug magnitude=15	77.7 ±0.15	81.5 ±0.05	83.2 ±0.09

Figura 3.14: Resultados del experimento de regularizar más el modelo conforme el tamaño de entrada aumenta.

Fuente: [23]

Es sencillo apreciar que los mejores resultados para cada tamaño de imagen son proporcionales al nivel de regularización usada; con tamaños de imagen de 128, pocos aumentos de datos han sido necesarios para obtener buenos resultados; conforme aumentamos el tamaño de la imagen ocurre lo contrario y se necesita más regularización para que el modelo tenga buenos resultados.

EfficientNetV2S como *baseline*

La siguiente imagen muestra la arquitectura de la *baseline* para la familia de EfficientNetV2: EfficientNetV2S, que fue encontrada usando la misma técnica que EfficientNetB0 en su momento (usando NAS, Neural Architecture Search) y contiene todos los cambios que hemos mencionado en este apartado anterior:

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBCConv4, k3x3, SE0.25	2	128	6
5	MBCConv6, k3x3, SE0.25	1	160	9
6	MBCConv6, k3x3, SE0.25	2	272	15
7	Conv1x1 & Pooling & FC	-	1792	1

Figura 3.15: Arquitectura de la *baseline* EfficientNetV2S

Fuente: [23]

Se puede apreciar en la figura 3.15 que esta nueva arquitectura utiliza una mezcla de los bloques MBConv y los experimentalmente probados Fused-MBConv en las primeras capas (que era donde mejores resultados se apreciaban); no sólo eso, sino que esta arquitectura opta por realizar más convoluciones 3x3 en comparación a las convoluciones 5x5 de la versión V1 y por último, la última fase de la red queda totalmente eliminada. Estas son las principales características de esta nueva familia de *ConvNets*, y finalmente se muestra una comparativa del rendimiento de esta familia respecto a su versión anterior y otras redes del estado del arte.

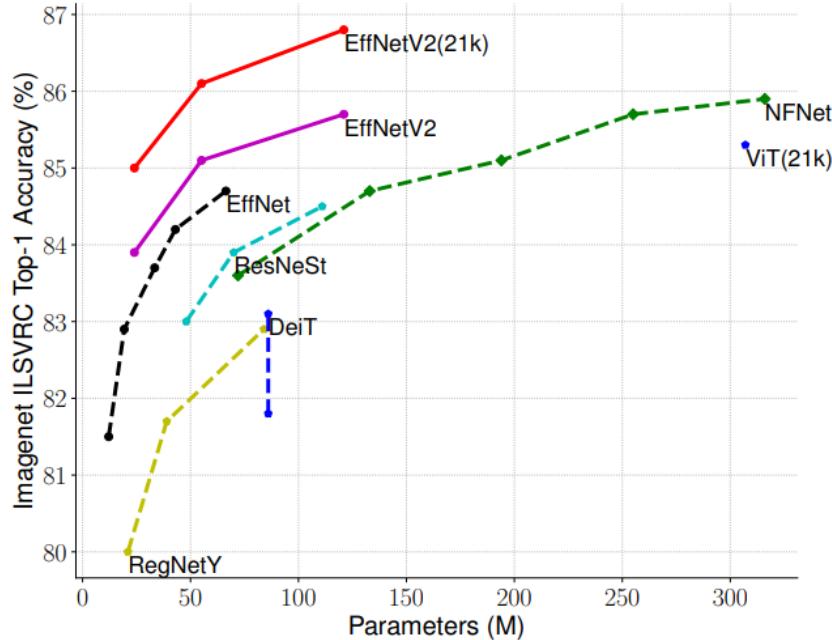


Figura 3.16: Comparativa de la arquitectura EfficientNetV2 respecto a otras redes del estado del arte; en el eje X se mide el número de parámetros y en el eje Y la *accuracy* sobre ImageNet.

Fuente: [23]

Podemos ver que los resultados son los esperados y las mejoras realmente hacen que los modelos necesiten muchos menos parámetros para conseguir los mismos resultados que el resto de arquitecturas. Al igual que antes, si se desea saber más de esta arquitectura, consultar la fuente original [23].

4 | Creación del *dataset*

4.1. Obtención de los datos (*dataset* inicial)

Todo problema de *machine learning* se soluciona tomando como el pilar fundamental los datos: los datos definirán el problema que resolveremos y la calidad y cantidad de estos es uno de los factores más decisivos para obtener un modelo de calidad.

Para este proyecto importante construir un *dataset* que sea lo más general posible para poder usar nuestro modelo en cualquier proyecto de clasificación de Diatomeas que lo necesite; el principal problema que tenemos que combatir es el sesgo de las imágenes al ser tomadas por los distintos laboratorios: cada laboratorio toma las imágenes con un modelo de microscopio diferente y utilizando diferentes vistas y *zooms*, por lo que para combatir este sesgo debemos de obtener Diatomeas de varias bases de datos y usar técnicas de aumento de datos para generar datos artificiales que reduzcan este sesgo, como veremos posteriormente en el apartado de preprocesado.

En principio se probará a reunir imágenes de 44 clases de Diatomeas (veremos cuáles son a continuación) para resolver un problema de clasificación de 44 clases, pero es importante recalcar que el modelo puede reajustarse con nuevas clases o eliminar clases ya existentes en cualquier momento y que si se obtienen nuevas imágenes para una clase ya existente es posible reentrenar el modelo para que ajuste esos nuevos datos. **La intención de este Trabajo de Fin de grado es demostrar que el diseño de un modelo es factible, pero tras este trabajo el modelo mejorará y se desarrollará si se consigue demostrar su funcionamiento, aceptando más clases y obteniendo datos provenientes de más fuentes.**

4.1.1. Datos obtenidos

Las Diatomeas que se han seleccionado para obtener imágenes han sido las mostradas en la siguiente tabla, donde se muestra además la cantidad de datos obtenidos para cada uno de ellos:

Achnanthidium	28	Hantzschia	3
Adlafia	7	Luticola	12
Amphora	23	Mayamaea	3
Aulacoseira	19	Meridion	5
Brachysira	13	Navicula	102
Caloneis	14	Neidiopsis	4
Cavinula	14	Neidium	14
Chamaepinnularia	8	Nitzschia	73
Coccconeis	17	Pinnularia	49
Craticula	15	Planothidium	23
Cyclotella	12	Psammothidium	22
Cymbopleura	38	Pseudostaurosira	12
Diadesmis	6	Rossithidium	5
Diatoma	7	Sellaphora	28
Encyonema	36	Stauroforma	2
Encyonopsis	29	Stauroneis	56
Eolimna	5	Staurosira	5
Eunotia	61	Staurosirella	9
Fragilaria	33	Stenopterobia	2
Frustulia	20	Stephanodiscus	11
Geissleria	9	Surirella	26
Gomphonema	84	Tabellaria	3

Cuadro 4.1: Lista de Diatomeas que se han seleccionado para obtener imágenes junto a la cantidad obtenida. Fuente: Elaboración propia.

De estas clases de Diatomeas se ha intentado obtener la mayor cantidad de imágenes buscando además que haya una gran variabilidad entre las subespecies de cada clase. Se han obtenido imágenes de 3 fuentes, con su página web correspondiente: Diatoms.org, ANSP, AQUALITAS y las usadas en el proyecto ADIAC comentado anteriormente, que se encuentran en la página web de RGBE.

Tras un mes de obtención de datos de manera manual debido a que no se tienen conocimientos de *web scrapping* y aprender esta técnica requiere tiempo adicional, se han obtenido 10650 imágenes de las 44 clases, cuya procedencia y cantidad son mostradas en el gráfico siguiente:

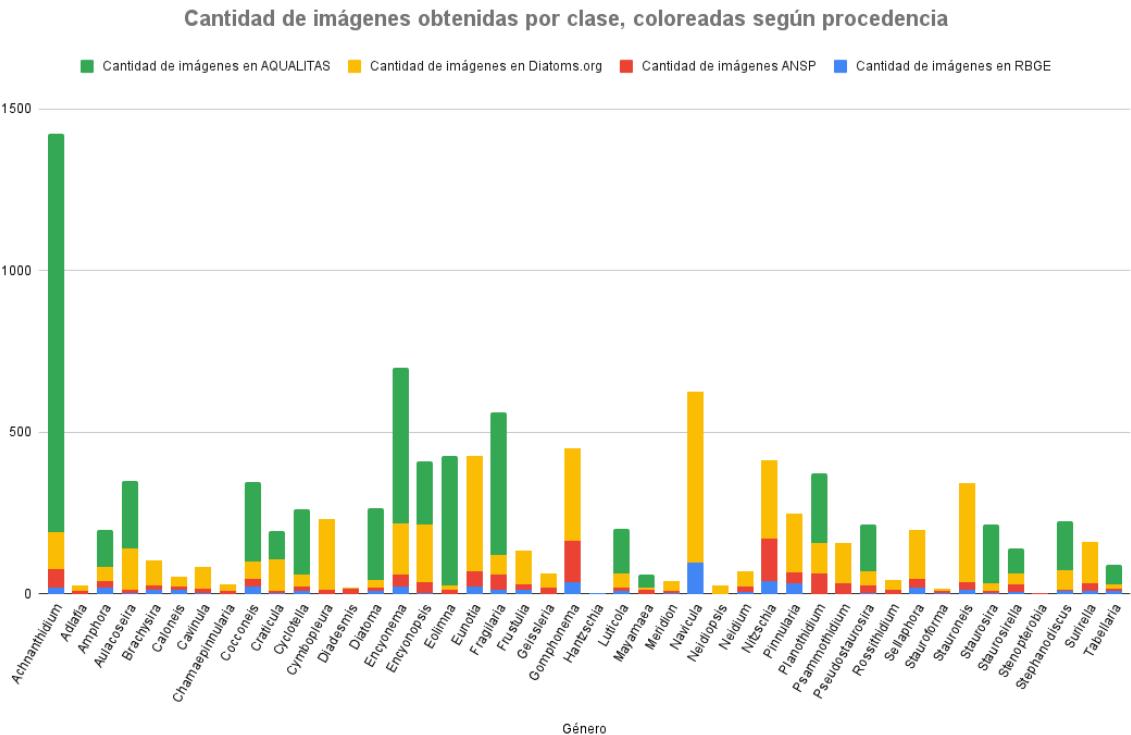


Figura 4.1: Cantidad de imágenes obtenidas según clase y coloreadas según procedencia

Fuente: Elaboración propia.

Este gráfico se puede acompañar con la información numérica de la tabla 4.1. En primer lugar, ha sido posible obtener imágenes de estas 44 clases pero de dos de ellas ha sido posible obtener muy pocas imágenes (3 de la clase *Hantzschia* y 5 de *Stenopterobia*) y no podemos hablar del comportamiento ni del aprendizaje de un modelo con tan pocos datos, por lo que estas clases quedan eliminadas para este estudio con posibilidad de obtener datos en un futuro, quedándonos finalmente con 10642 imágenes. Por otro lado, vemos que los *datasets* de los que hemos podido obtener una mayor cantidad de imágenes han sido AQUALITAS y Diatoms.org, *datasets* que además incluyen una gran cantidad de imágenes de otras clases de Diatomeas que pueden ser interesantes de cara a mejorar el modelo en futuras actualizaciones. Por último, destacar que la clase de diatomeas que sobresale es *Achnanthidium*, con 1424 imágenes, mientras que el resto de ellas no llegan ni a 600 en las clases con más imágenes.

Lo más importante de este análisis es que estamos trabajando con un *dataset* que numéricamente, sin tener analizar las imágenes previamente, está **desbalanceado**. Tenemos clases de Diatomeas con 20 imágenes y otras clases con 600, por lo que el modelo tenderá a clasificar las diatomeas de las clases mayoritarias y no aprender las características principales de las clases minoritarias porque tiene menos ejemplos. **Este problema es un reto** que se abordará de una manera en particular en el apartado de preprocesado.

4.2. Análisis de las imágenes

Es interesante ver las imágenes de este compendio de fuentes para ver las diferencias que comentamos anteriormente a nivel de toma de imágenes: calidad del microscopio, nivel

de zoom y tamaño del Diatomea en la imagen, intensidad de la luz a la hora de tomar la imagen...

Para ello, mostramos un mosaico de 25 imágenes de diatomeas seleccionados de manera aleatoria en la base de datos, con su clase correspondiente como título, en la siguiente figura:

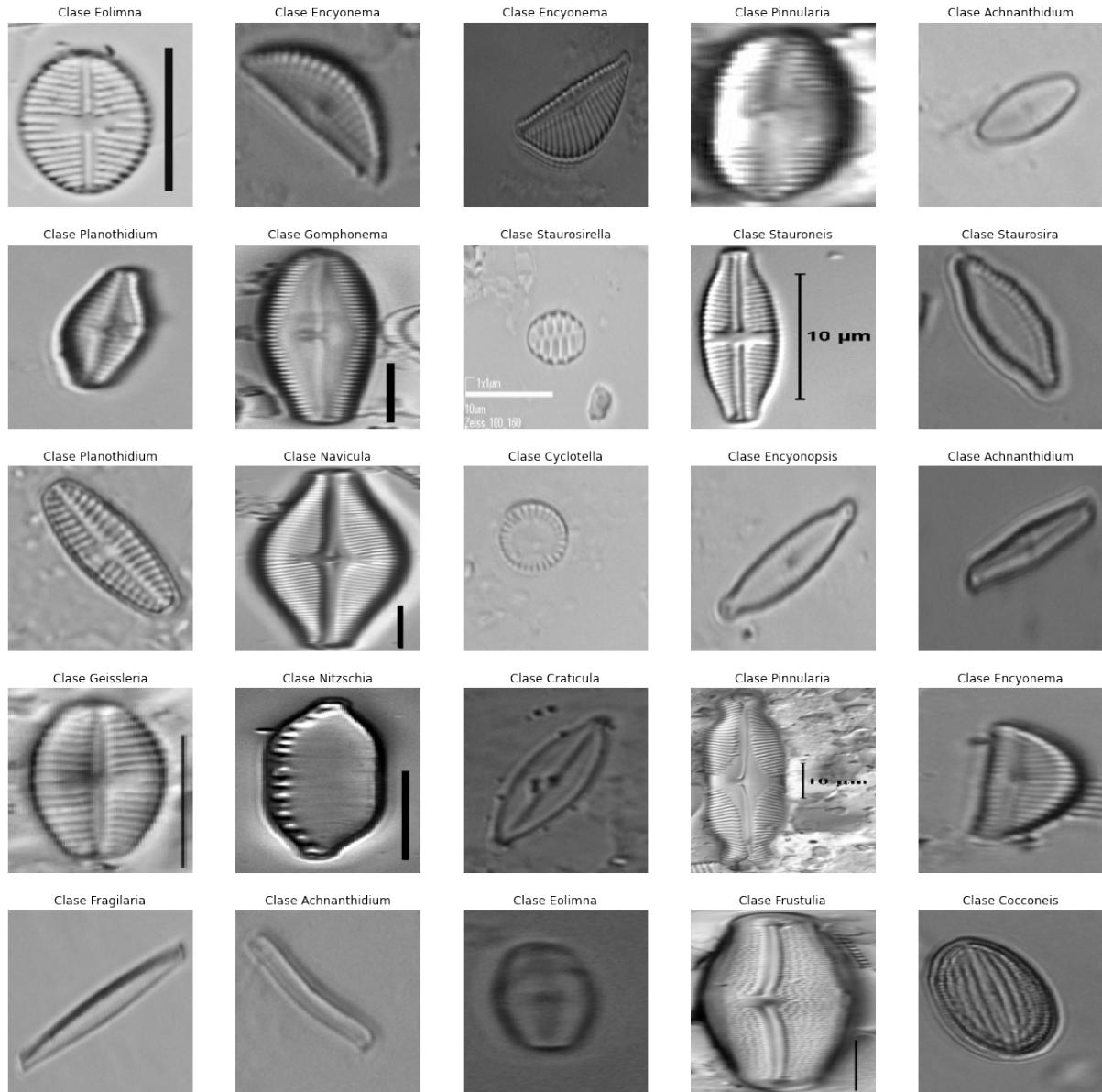


Figura 4.2: Muestra de 25 diatomeas aleatorios de la base de datos construida.

Fuente: Elaboración propia.

En primer lugar, podemos ver que las imágenes necesitan de algo de preprocesado para eliminar las marcas generadas por los microscopios electrónicos, que pueden hacer que muchos parámetros de la red traten de asociar a esa marca como una característica relevante y perder capacidad de generalización ante imágenes que no posean esta marca, al contrario que por ejemplo, los trozos de diatomeas muertos o trozos de polvo que sí son interesantes mantener debido a que son parte del contexto general de una imagen de este tipo y ayudan a que la red sepa diferenciar bien lo que es una diatomea y lo que no. Si entrenásemos nuestras imágenes con fondo totalmente limpio, cuando la red tenga que

clasificar una imagen nueva que tenga suciedad podría equivocarse tomando esa suciedad como parte de la diatomea. Dejar información que **no es de la diatomea** también ayuda a la red a clasificar bien.

Por otro lado, vemos las diferencias en tamaño de los diatomeas, donde hay imágenes que el diatomea la cubre completamente y otras donde el diatomea es menos del 30 % de la imagen. Por lo general, queremos que nuestras diatomeas ocupen como mínimo el 60 % de la imagen y el restante sea contexto, por lo que se reescalarán las imágenes que sean pequeñas para que cumplan lo anterior.

Otra cosa que podemos apreciar es que las imágenes no tienen la misma intensidad y, pese a que algunas de ellas están rotadas, la mayoría están mostradas de forma vertical y horizontal. Analizando los *datasets*, vemos que **todas las diatomeas de las imágenes de RBGE, Diatoms.org y ANSP** están mostradas de manera vertical u horizontal. Tendremos esto en cuenta para el apartado de entrenamiento, pero también para el de validación, puesto que en la realidad no todos los diatomas estarán distribuidos de esa manera en una imagen y podrán aparecer como en la base de datos de ADIAC, es decir, rotados.

Teniendo todo esto en cuenta, es interesante preprocesar las imágenes para que sean entradas correctas a la red y lograr nuestro objetivo, eliminando los problemas que acabamos de comentar y que surgirán si entrenamos nuestras *ConvNets* con las imágenes así.

4.3. Preprocesado

4.3.1. Limpieza y recorte

El proceso de limpieza y recorte ha sido un proceso manual, imagen por imagen, en el que se detecta si la imagen necesita ser recortada y si tiene alguna marca generada por el microscopio alrededor de la diatomea. Este proceso no se ha podido automatizar debido a que cada imagen tiene un tamaño diferente y la marca se genera dependiendo de la ubicación del diatomea en la imagen. Para eliminar las marcas, se ha usado el software *Adobe Photoshop* [24] y usando una herramienta de relleno inteligente que tiene en cuenta las zonas alrededor de la marca para sustituirla por píxeles parecidos, simulando ser parte de la imagen original. Un ejemplo de este preprocesado es el realizado con la siguiente imagen:

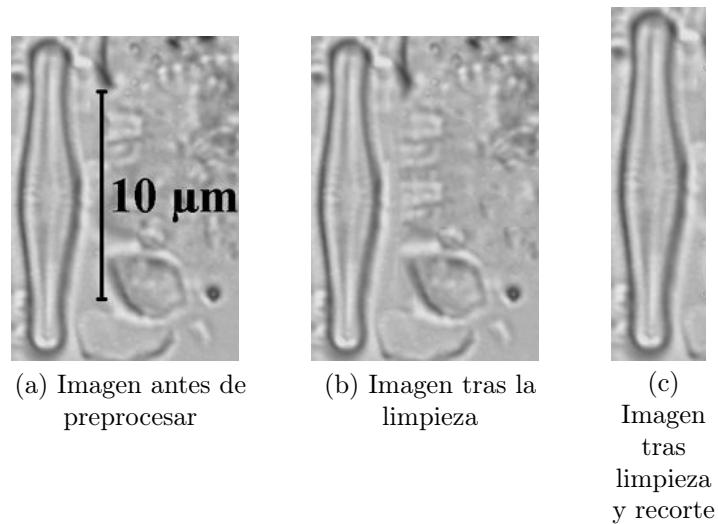


Figura 4.3: Ejemplo del preprocessado de una imagen.
Fuente: Elaboración propia.

En la imagen anterior, tras eliminar las marcas de microscopio de la imagen se ha recortado parte de ella para adaptarlo al Diatomea, de manera que ocupe la mayor parte de la imagen, a modo de ejemplo. En la base de datos hay imágenes que requieren uno de los dos pasos y otras que requieren ambos o ninguno. Tras realizar este preprocessado, los diatomeas obtenidos se ven como en la siguiente imagen:

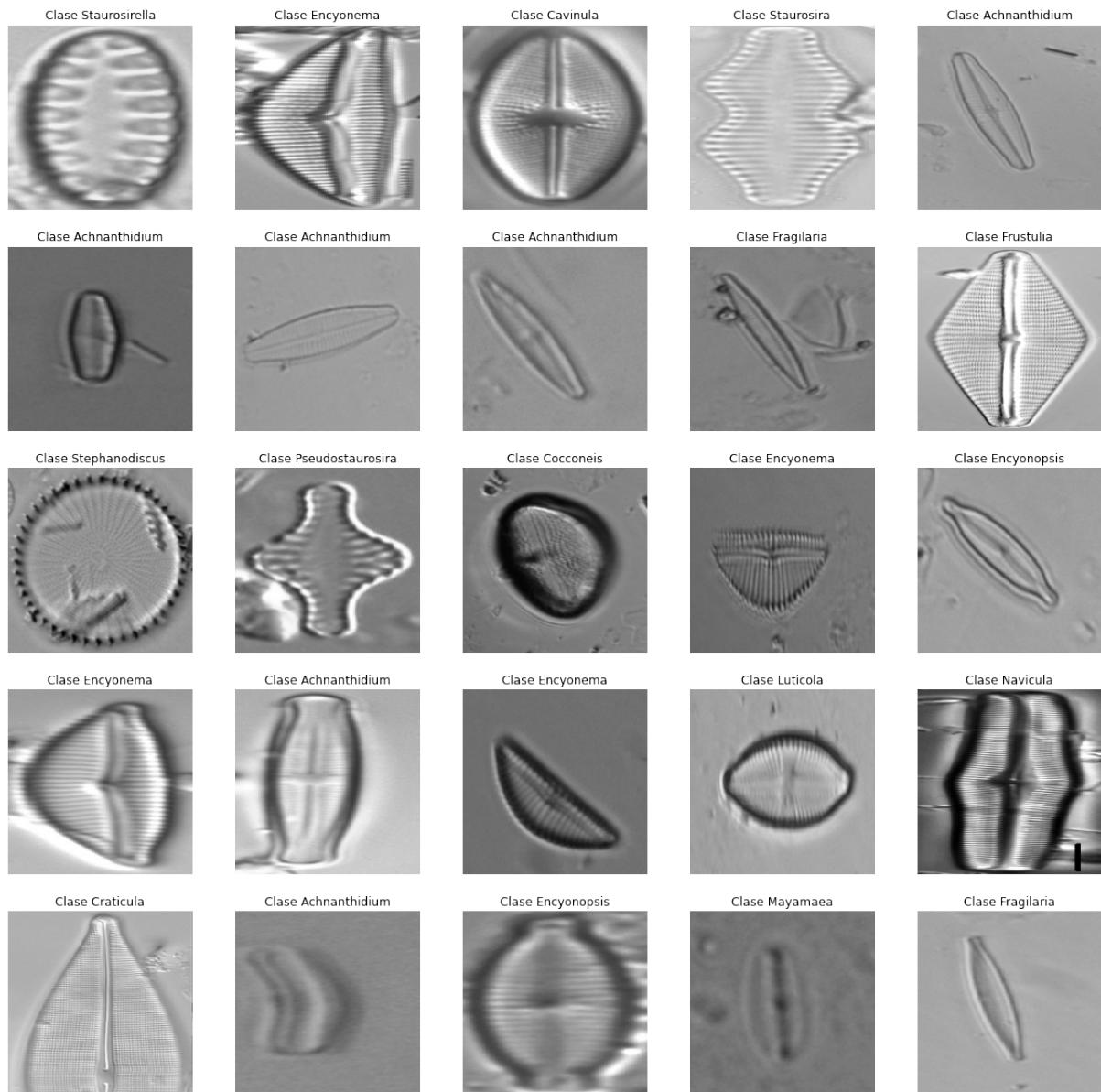


Figura 4.4: Muestra de 25 diatomeas aleatorios de la base de datos construida tras el preprocesado
Fuente: Elaboración propia.

Si comparamos con la muestra anterior, las imágenes ahora ocupan un porcentaje de la imagen bastante mayor en general y no hay marcas de microscopio, por lo que serán fáciles de identificar. Se puede ver también que se han mantenido las marcas de suciedad, restos y trozos de diatomeas en el fondo de las imágenes consiguiendo finalmente datos valiosos para la red.

4.3.2. *Data Augmentation*

4.4. Creación de los *dataset* de entrenamiento y *test*

5 | Diseño e implementación

5.1. *Framework* utilizado

5.2. Arquitecturas utilizadas

6 | Análisis de resultados

- 6.1. Técnica utilizada para el análisis
- 6.2. Comparativa de arquitecturas
- 6.3. Selección del mejor modelo
- 6.4. Visualización del aprendizaje

Bibliografía

- [1] Eduardo Lobo y col. «Diatoms as Bioindicators in Rivers». En: jun. de 2016, págs. 245-271. ISBN: 978-3-319-31983-4. DOI: [10.1007/978-3-319-31984-1_11](https://doi.org/10.1007/978-3-319-31984-1_11).
- [2] Luc Ector y Frédéric Rimet. «Using bioindicators to assess rivers in Europe: An overview». En: ene. de 2005, págs. 7-19. DOI: [10.1007/3-540-26894-4_2](https://doi.org/10.1007/3-540-26894-4_2).
- [3] Saúl Blanco y col. «Comparison of Biotic Indices for Water Quality Diagnosis in the Duero Basin (Spain)». En: *Archiv für Hydrobiologie. Supplementband. Large rivers* 17 (ene. de 2007), págs. 267-286.
- [4] British Standards Institute Staff. *Water Quality. Guidance Standard for the Identification, Enumeration and Interpretation of Benthic Diatom Samples from Running Waters*. B S I Standards, 2004. ISBN: 9780580442476. URL: <https://books.google.es/books?id=un5SPQAACAAJ>.
- [5] David G. Mann. «The species concept in diatoms». En: *Phycologia* 38 (1999), págs. 437-495.
- [6] Carlos Sánchez-Bueno, Gabriel Cristóbal y Gloria Bueno. «Diatom identification including life cycle stages through morphological and texture descriptors». En: (2019). DOI: [10.7717/peerj.6770](https://doi.org/10.7717/peerj.6770).
- [7] Nils J. Nilsson. *Introduction to Machine Learning: An Early Draft of a Proposed Textbook*. 1998. URL: <http://robotics.stanford.edu/people/nilsson/mlbook.html>.
- [8] Edpresso. *What is feature extraction?* 2019. URL: <https://www.educative.io/edpresso/what-is-feature-extraction> (visitado 30-09-2010).
- [9] Akinori Hidaka y Takio Kurita. «Consecutive Dimensionality Reduction by Canonical Correlation Analysis for Visualization of Convolutional Neural Networks». En: vol. 2017. Dic. de 2017, págs. 160-167. DOI: [10.5687/ss.2017.160](https://doi.org/10.5687/ss.2017.160).
- [10] Mete Ahishali y col. «Classification of polarimetric SAR images using compact convolutional neural networks». En: *GIScience & Remote Sensing* 58.1 (2021), págs. 28-47. DOI: [10.1080/15481603.2020.1853948](https://doi.org/10.1080/15481603.2020.1853948). eprint: <https://doi.org/10.1080/15481603.2020.1853948>. URL: <https://doi.org/10.1080/15481603.2020.1853948>.
- [11] Jiuxiang Gu y col. *Recent Advances in Convolutional Neural Networks*. 2015. DOI: [10.48550/ARXIV.1512.07108](https://arxiv.org/abs/1512.07108). URL: <https://arxiv.org/abs/1512.07108>.
- [12] Kamel Abdelouahab. «Reconfigurable hardware acceleration of CNNs on FPGA-based smart cameras». Dic. de 2018.
- [13] Amidi Shervine. *Teaching - CS 229*. <https://stanford.edu/~shervine/teaching/cs-229/>. 2018.
- [14] Hans du Buf y Micha M Bayer. *Automatic Diatom Identification*. WORLD SCIENTIFIC, 2002. DOI: [10.1142/4907](https://doi.org/10.1142/4907). eprint: <https://www.worldscientific.com/doi/pdf/10.1142/4907>. URL: <https://www.worldscientific.com/doi/abs/10.1142/4907>.

- [15] Jesús Salido y col. «A Low-Cost Automated Digital Microscopy Platform for Automatic Identification of Diatoms». En: *Applied Sciences* 10.17 (2020). ISSN: 2076-3417. DOI: [10.3390/app10176033](https://doi.org/10.3390/app10176033). URL: <https://www.mdpi.com/2076-3417/10/17/6033>.
- [16] Gloria Bueno y col. «Automated Diatom Classification (Part A): Handcrafted Feature Approaches». En: *Applied Sciences* 7.8 (2017). ISSN: 2076-3417. DOI: [10.3390/app7080753](https://doi.org/10.3390/app7080753). URL: <https://www.mdpi.com/2076-3417/7/8/753>.
- [17] John Cairns, Silverio P. Almeida e Hitoshi Fujii. «Automated Identification of Diatoms». En: *BioScience* 32.2 (1982), págs. 98-102. ISSN: 00063568, 15253244. URL: <http://www.jstor.org/stable/1308561> (visitado 14-06-2022).
- [18] F. James Rohlf y Dennis Slice. «Extensions of the Procrustes Method for the Optimal Superimposition of Landmarks». En: *Systematic Zoology* 39.1 (1990), págs. 40-59. ISSN: 00397989. URL: <http://www.jstor.org/stable/2992207> (visitado 14-06-2022).
- [19] Nouzha Chahboune, Mohamed Mehdi y Allal Douira. «Automatic classification of diatoms of Merja fouarate». En: *bioRxiv* (2017). DOI: [10.1101/099135](https://doi.org/10.1101/099135). eprint: <https://www.biorxiv.org/content/early/2017/01/09/099135.full.pdf>. URL: <https://www.biorxiv.org/content/early/2017/01/09/099135>.
- [20] J. du Buf y col. «Diatom identification: A double challenge called ADIAC». En: ene. de 1999, págs. 734-739. DOI: [10.1109/ICIP.1999.797682](https://doi.org/10.1109/ICIP.1999.797682).
- [21] Mingxing Tan y Quoc V. Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». En: *CoRR* abs/1905.11946 (2019). arXiv: [1905.11946](https://arxiv.org/abs/1905.11946). URL: [http://arxiv.org/abs/1905.11946](https://arxiv.org/abs/1905.11946).
- [22] Hugo Touvron y col. *Fixing the train-test resolution discrepancy*. 2019. DOI: [10.48550/ARXIV.1906.06423](https://doi.org/10.48550/ARXIV.1906.06423). URL: <https://arxiv.org/abs/1906.06423>.
- [23] Mingxing Tan y Quoc V. Le. «EfficientNetV2: Smaller Models and Faster Training». En: *CoRR* abs/2104.00298 (2021). arXiv: [2104.00298](https://arxiv.org/abs/2104.00298). URL: <https://arxiv.org/abs/2104.00298>.
- [24] Adobe Inc. *Adobe Photoshop*. Ver. CC 2019. 6 de mar. de 2019. URL: <https://www.adobe.com/products/photoshop.html>.