Universitat Oberta de Catalunya

uoc.edu

# EXPLORACIÓN DE TÉCNICAS DE SE-SEMI-SUPERVISED LEARNING PARA LA CLASIFICACIÓN DE IMÁGENES DE CÉLULAS DE SANGRE PERIFÉRICA

Isaac León Ortiz
Máster en Bioinformática y Bioestadística
Machine Learning

Nombre Director/a
Edwin Santiago Alférez Baquero
Junio de 2021

FICHA DEL TRABAJO FINAL

| | |
|---|---|
| Título del trabajo: | Exploración de técnicas de semi-supervised learning para la clasificación de células de sangre periférica. |
| Nombre del autor: | Isaac León Ortiz |
| Nombre del consultor/a: | Edwin Santiago Alférez Baquero |
| Nombre del PRA: | Laura Calvet Liñan |
| Fecha de entrega (mm/aaaa): | 08/06/2021 |
| Titulación: | Máster en Bioinformática y Bioestadística |
| Área del Trabajo Final: | Machine Learning |
| Idioma del trabajo: | Castellano |
| Número de créditos: | 15 |
| Palabras clave | Inteligencia Artificial, Machine Learning, Red Neuronal Convolucional. |


Resumen del Trabajo:

Mediante el procesamiento de imágenes digitales, el aprendizaje automático y las herramientas de aprendizaje profundo, se ha logrado crear una base de datos de imágenes de células de muestras de frotis de sangre periférica de un periodo de 9 años. Con este y otros conjuntos de datos tan grandes y con tanta información, puede ser difícil lograr encontrar el algoritmo de machine learning capaz de analizar y clasificar la información con el menor consumo de recursos y obteniendo los mejores resultados.

Este trabajo final de máster utiliza técnicas de semi-supervised learning para minimizar la problemática y mejorar la precisión de clasificación respecto a otras técnicas como las Redes Neuronales Convolucionales.

Como resultado del estudio se han generado dos modelos de redes neuronales, una Red Neuronal Convolucional que permite la clasificación de las imágenes con una precisión del 95%, y una red neuronal profunda que utiliza codificadores automáticos y una capa densa para identificar cada célula con una precisión del 96%.

Estos resultados evidencian la utilidad de técnicas de semi-supervised learning en la clasificación y análisis de datos sin etiquetar, y pueden dar pie a un mejor entendimiento del funcionamiento y aplicabilidad de las redes neuronal en cualquier campo de la bioinformática.

Abstract:

Through digital image processing, machine learning and deep learning tools, a database of cell images of peripheral blood smear samples from a period of 9 years. With this and other such large data sets and with so much information, it can be difficult to find the machine learning algorithm capable of analysing and classifying the information with the least consumption of resources and obtaining the best results.

This final master's thesis uses semi-supervised learning techniques to minimize problems and improve the classification of these samples with respect to other techniques such as Convolutional Neural Networks.

As a result of the study, two models of neural networks have been generated, a Convolutional Neural Network that allows the classification of images with a precision of 95%, and a deep neural network that uses autoencoders and a dense layer to identify each cell with a 96% accuracy.

These results show the usefulness of semi-supervised learning techniques in the classification and analysis of unlabelled data, and may lead to a better understanding of the functioning and applicability of neural networks in any field of Bioinformatics.

# Índice de contenidos

# Lista de tablas

# Lista de figuras

# Lista de códigos y comandos

# 1. Plan de trabajo

## 1.1 Contexto y justificación del trabajo

### 1.1.1 Descripción general

La inteligencia artificial (IA) es un concepto que apareció en los años 50 y se define como como aquellos sistemas informáticos que manifiestan un comportamiento inteligente, parecido a la inteligencia humana. Entre los algoritmos más comunes encontramos: los algoritmos de árbol de decisión, los algoritmos de Redes Neuronales Convolucionales (en inglés, *Convolutional Neural Networks* o CNN) o los algoritmos de Aprendizaje Profundo, entre otros[1–4]. En el caso del aprendizaje profundo, este suele enfocarse en producir un modelo de clasificación de alta eficiencia mediante la realización de entrenamiento, aprendizaje y retroalimentaciones repetidas utilizando conjuntos de datos previamente clasificados para lograr la extracción y detección automática de características que nos permitan clasificar nuevos datos[5,6].

A lo largo de un periodo de 9 años, el laboratorio CORE del *Hospital Clínic de Barcelona* ha creado una base de datos de imágenes de células de muestras de frotis de sangre periférica (sangre que circula por todo el cuerpo) de distintos pacientes. Mediante el procesamiento manual de imágenes digitales, el personal médico puede aprender a clasificar cada muestra e identificar el tipo de célula que se presente en esta misma. Sin embargo, a medida que la cantidad de datos aumenta gradualmente, la identificación manual de todas las características que nos interesan ya no puede satisfacer la demanda a tiempo. Aquí es donde entra el aprendizaje profundo que, basándose en clasificadores tradicionales para reducir la dimensionalidad de los datos, mediante el entrenamiento en una red neuronal multicapa permiten reducir costes y tiempo a la hora de clasificar los datos, además de generar mejores resultados[5,6]. La precisión del arquitecto de clasificación puede ser mejorada mediante el pre-procesamiento de imágenes (como la segmentación) y el uso de técnicas de *semi-supervised learning*[6,7].

Este trabajo tiene como objetivo la comparación de un arquitecto de clasificación mediante Redes Neuronales Convolucionales y un modelo de CNN que utilice técnicas de *semi-supervised learning*.

### 1.1.2 Justificación del trabajo

El término aprendizaje semisupervisado (en inglés, semi-supervised learning o SSL) puede hacer referencia a diferentes aplicaciones dependiendo del contexto o el campo en el que se esté trabajando. Entre los más "importantes" se comprenden: la clasificación de datos a través de su misma representación, la creación de las ya mencionadas anteriormente redes neuronales o el procesamiento de distintos lenguajes, por ejemplo. En todos los casos descritos, el objetivo principal es la generación automática de algún tipo de señal de supervisión que nos permita realizar una tarea. Si nos regimos por el nombre aprendizaje semisupervisado, normalmente nos referimos a aprender la representación de nuestros datos, la estructura o el orden que siguen, o a etiquetar automáticamente un conjunto de datos a partir de una muestra previamente etiquetada de los mismos.

Los SSL son una herramienta muy útil y con una gran versatilidad que nos permite ampliar nuestro análisis de los datos manteniendo una buena relación entre los recursos invertidos (sean estos el tiempo necesario, la maquinaria utilizada o cualquier otro tipo

de recurso empleado en el cumplimiento de nuestros objetivos) y los resultados obtenidos. Es por esta razón, y por mi gusto por la *machine learning* y el diseño y análisis de datos, que he considerado llevar a cabo este trabajo.

## 1.2 Objetivos del trabajo

### 1.2.1 Objetivos generales

Explorar diversas técnicas de *semi-supervised learning* para la creación de un modelo de clasificación de un conjunto de datos de muestras de frotis de sangre periférica.

### 1.2.2 Objetivos específicos

Con la realización de este trabajo se espera cumplir con los objetivos siguientes:

- Aprender y entender los métodos del aprendizaje profundo para la clasificación de imágenes digitales.
- Realizar una clasificación con redes neuronales convolucionales para clasificar imágenes de células de sangre periférica.
- Desarrollar modelos de redes neuronales que apliquen técnicas de *semi-supervised learning* para la clasificación de imágenes de células de sangre periférica.
- Comparar los modelos generados, con y sin la implementación de técnicas se *semi-supervised learning* para la clasificación de imágenes.

## 1.3 Enfoque y métodos a seguir

Con los objetivos planteados anteriormente, se pretende desarrollar una clasificación de las muestras de frotis de imágenes de células de sangre periférica obtenidas por el laboratorio CORE del *Hospital Clínic de Barcelona*, disponibles en el *Journal* de *Data in Brief* de Andrea Acevedo, a partir de técnicas *semi-supervised learning*[17]. La hipótesis de este trabajo es que los resultados mostraran una mejora respecto al uso tradicional de las redes neuronales convolucionales.

Para la realización de este trabajo, se seguirá el proceso siguiente:

En primer lugar, realizaremos una búsqueda intensiva de bibliografía relacionada con el uso de técnicas de aprendizaje profundo para la clasificación de imágenes digitales y repasaremos los modelos de CNN con y sin la aplicación de técnicas *semi-supervised learning* que deseamos utilizar para el cumplimiento de nuestros objetivos.

En segundo lugar, realizaremos los algoritmos de clasificación deseados utilizando el lenguaje de programación Python con herramientas que permitan trabajar en la nube.

Finalmente, se procesarán y compararán los resultados generados con ambas técnicas, se analizarán sus implicaciones y se obtendrán las conclusiones.

## 1.4 Planificación con hitos y temporización

### 1.4.1 **Tareas**

- Definición de los contenidos del trabajo (10 h.)

  Definir claramente cuál es la temática del trabajo, justificar su interés y/o relevancia y qué se quiere conseguir al finalizar el TFM.

- Definición del Plan de trabajo (30 h.)

  Concretar, delimitar y describir el trabajo que se va a llevar a cabo, sus objetivos, la metodología a seguir, así como detallar los hitos y la temporización prevista.

- Desarrollo del trabajo - Fase 1
  - Preparación de los recursos necesarios (5 h.)

  Recopilación del software necesario. Descarga del conjunto de datos a utilizar y preparación para la utilización de herramientas como Colab.

  - Realización del curso Fast.ai sobre aprendizaje profundo (20 h.)

  Reforzar los conocimientos de Python y prepararse para el desarrollo de los modelos de clasificación con aprendizaje profundo.

  - Recopilación de bibliografía (10 h.)

  Búsqueda de guías, referencias y estudios que permitan entender mejor el trabajo a realizar.

  - Selección del método de clasificación (10 h.)

  Definir diferentes métodos de clasificación que podrían cumplir con nuestros requisitos y mirar de evaluar su desempeño, eficacia y coste, además de comparar los resultados finales de cada uno.

- Desarrollo del trabajo - Fase 2
  - Implementación de semi-supervised learning con deep learning (20 h.)
  - Análisis y comparación de los resultados (10 h.)
  - Redacción de memoria (30 h.)

### 1.4.2 Calendario

**Tabla 1. Calendario con la planificación de las tareas a realizar**

| Tarea | Inicio | Finalización |
|---|---|---|
| **Definición de los contenidos del trabajo (PEC0)** | 17/02/21 | 01/03/21 |
| **Recopilación de bibliografía** | 01/03/21 | 19/04/21 |
| **Preparación de los recursos necesarios** | 02/03/21 | 07/03/21 |
| **Definición del Plan de trabajo (PEC1)** | 02/03/21 | 16/03/21 |
| **Realización del curso Fast.ai** | 17/03/21 | 29/03/21 |
| **Selección del método de clasificación** | 30/03/21 | 19/04/21 |
| **Implementación de semi-supervised learning con deep learning** | 20/04/21 | 01/05/21 |
| **Análisis y comparación de los resultados** | 02/05/21 | 17/05/21 |
| **Redacción de memoria** | 18/05/21 | 08/06/21 |

### 1.4.3 Hitos

Los hitos marcan los estados intermedios del proyecto y permiten avanzar en sucesivas etapas de resultados prácticos. Por lo tanto, es imprescindible ser el máximo de estricto en el cumplimiento de las fechas.

**Tabla 2. Calendario con los entregables de este trabajo**

| Nombre | Inicio | Entrega |
|---|---|---|
| **PEC0 - Definición de los contenidos del trabajo** | 17/02/2021 | 01/03/2021 |
| **PEC1 - Plan de trabajo** | 02/03/2021 | 16/03/2021 |
| **PEC2 - Desarrollo del trabajo - Fase 1** | 17/03/2021 | 19/04/2021 |
| **PEC3 - Desarrollo del trabajo - Fase 2** | 20/04/2021 | 17/05/2021 |
| **PEC4 - Cierre de la memoria** | 18/05/2021 | 08/06/2021 |
| **PEC5a - Elaboración de la presentación** | 09/06/2021 | 13/06/2021 |
| **PEC5b - Defensa pública** | 16/06/2021 | 23/06/2021 |

### 1.4.4 Análisis de riesgos

Existen factores que pueden repercutir negativamente en el seguimiento del plan de trabajo y en la consecución del proyecto. A continuación, se plantean algunos de ellos y las medidas que se han tomado para sobrellevarlos y minimizar su repercusión:

- Problemas técnicos: estos pueden ser cualquier avería del puesto de trabajo como de los programas utilizados para la realización de este.
  - Para mitigar su repercusión se dispone de otro equipo y se trabaja con software en línea.
- Pérdida de los datos generados: entre otros, la corrupción del conjunto de datos utilizado y de los resultados obtenidos.
  - Se dispone de copias de seguridad y se trabaja con programas en la nube.
- Accidentes personales: por ejemplo, una enfermedad.
  - En caso de no poder cumplir con los objetivos por motivo de una enfermedad o un accidente imprevisto, se intentará pactar otro plan de trabajo que permita la reorganización de las horas planteadas para poder, al menos, cumplir con las entregas finales.
- Imposibilidad de cumplir con la planificación: a veces, cuando se empieza a realizar un proyecto, nos damos cuenta de que el tiempo previsto para finalizarlo es mayor al esperado.
  - En estos casos, se intentará reorganizar las horas programadas para cada tarea con tal de cumplir con los hitos marcados anteriormente.

### 1.4.5 Costes asociados a la realización del trabajo

El trabajo aquí presente se ha realizado en el domicilio del autor con el equipo ya disponible, por lo que no se ha llevado a cabo ninguna inversión en hardware. Dicho esto, el equipo utilizado comporta un coste derivado de la luz y la energía necesarios para su funcionamiento de aproximadamente 30 €.

Por lo que al software se refiere, todos los programas utilizados han sido de licencia libre para su uso público. No se ha visto pertinente la ampliación de ningún recurso o el pago de ninguna tarifa.

Finalmente, en este trabajo se han empleado horas de esfuerzo y dedicación por parte tanto del autor como del consultor del TFM, pero al tratarse de aspectos metafísicos y no materiales, no se le aplica ningún coste monetario.

## 1.5 Resultados esperados

Con la realización de este trabajo se obtendrán los siguientes ítems tangibles:

### 1.5.1 Plan de trabajo

### 1.5.2 Memoria

### 1.5.3 Notebooks

### 1.5.4 Presentación virtual

### 1.5.5 Autoevaluación del proyecto

## 1.6 Estructuración del proyecto

Para la realización de esta memoria se ha seguido la siguiente estructura:

1. Resumen: contiene los antecedentes y una explicación de los métodos, resultados y conclusiones obtenidas con el trabajo.
2. Planteamiento del trabajo: con el contexto, justificación y plan de trabajo.
3. Introducción: con una descripción general del trabajo y su contexto.
4. Metodología: explicación de la metodología utilizada y justificación de esta.
5. Resultados y discusión: donde se mostrarán los resultados generados con el trabajo y se pondrán en relación con la metodología seguida. Finalmente, se plantean las implicaciones que pueden tener los resultados generados y se responden nuestras hipótesis.
6. Conclusiones: descripción de lo aprendido con la realización del trabajo y de la relevancia y justificación de este.
7. Glosario.
8. Bibliografía.
9. Anexos: con los apartados de los resultados ordenados por su metodología mostrados extensivamente.

# 2. Introducción

## 2.1 Aprendizaje Profundo

El aprendizaje profundo es una técnica informática que permite extraer y transformar datos para su posterior análisis y uso. Esta presenta un poder, flexibilidad y simplicidad que la hace no solo útil, sino que también recomendada para su aplicación en todo tipo de disciplinas como son las ciencias sociales y físicas, las artes, la medicina o la investigación científica[8,9].

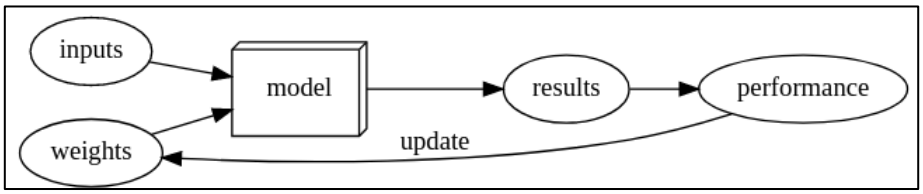


**Figura 1. Representación del proceso de entrenamiento y creación de un modelo con aprendizaje profundo. Fuente: Curso fast.ai**

Las aplicaciones de esta técnica van desde su uso para el reconocimiento de voz hasta la clasificación de imágenes digitales mediante el uso de múltiples capas de redes neuronales. Cada una de estas capas toma un input o entrada de capas anteriores y las refina progresivamente. Las capas, a su vez, están entrenadas por algoritmos que minimizan sus errores y mejoran su precisión. De esta manera, la red aprende a realizar una tarea específica[8].

## 2.2 Redes Neuronales

Aunque ya se conocía de ellas en el pasado, no fue hasta 1943 cuando Warren McCulloch, neurofisiólogo, y Walter Pitts, lógico, desarrollaron un modelo matemático de una neurona artificial, que se normalizó y extendió su uso en el común de las personas[10].

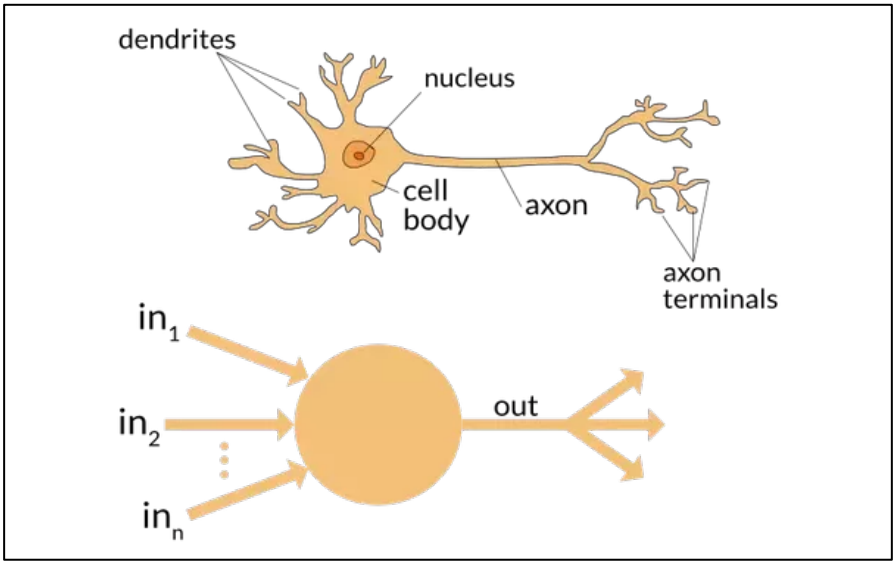


**Figura 2. Representación de las similitudes entre el funcionamiento de una neurona biológica y el de una neurona artificial. Fuente: Curso fast.ai**

Las redes neuronales son un conjunto de algoritmos, modelados libremente a partir del cerebro humano (en la figura 2 se puede observar un esquema), que están diseñados para reconocer patrones en nuestros datos. Los patrones que reconocen son numéricos, contenidos en vectores, a los que deben traducirse todos los datos del mundo real, ya sean imágenes, sonido, texto o series de tiempo.

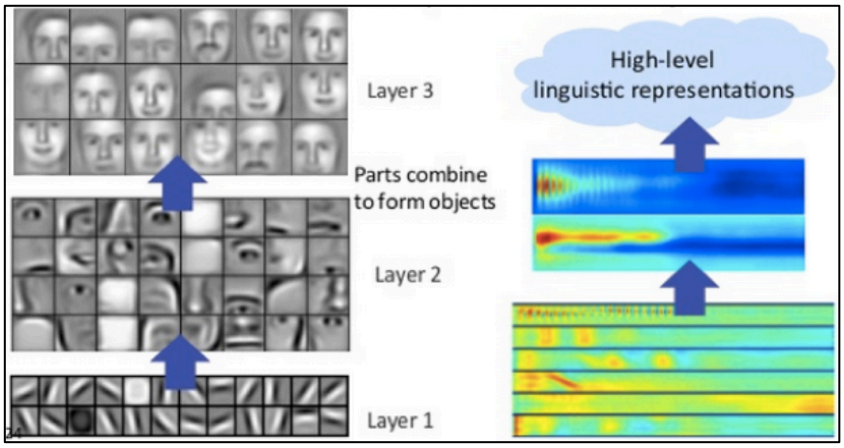


**Figura 3. Representación de un modelo de capas sucesivas que aprende a generar representaciones a partir de segmentos. Fuente:** ***Neural Networks and Deep Learning***

Las redes neuronales nos ayudan a agrupar y clasificar información. Además, también permiten la agrupación de datos sin etiquetar de acuerdo con las similitudes entre las entradas de estos y la arquitectura o modelo generado con el conjunto de datos etiquetados utilizado para entrenarlo. En la figura anterior (Fig. 3), podemos observar el resultado de una red neuronal profunda que permite agregar y recombinar características de las capas anteriores.

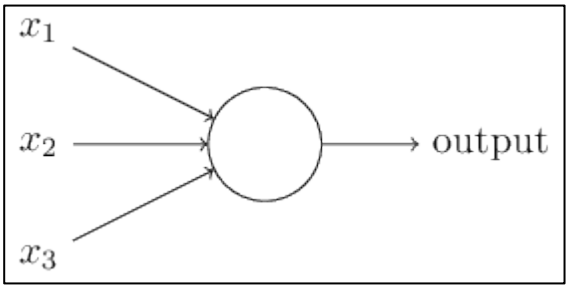


**Figura 4. Esquema del funcionamiento de un perceptrón. Fuente:** ***Neural Networks and Deep Learning***

Actualmente, existen diferentes tipos de redes neuronales. Una de ellas es el perceptrón creado por Frank Rosenblatt[11]. Estos se encargan de aceptar diferentes entradas binarias y producir una sola salida, también binaria. Un esquema de esto lo vemos en la figura 4, donde se introducen tres entradas. Esta idea ya la propusieron Warren McCulloch y Walter Pitts como se comentaba anteriormente[10]. Sin embargo, Rosenblatt también incluyo una nueva característica, los pesos. Estos son números reales que expresan la importancia de las respectivas entradas a la salida y, a partir de su suma, se generará la salida[11]. En definitiva, el funcionamiento del perceptrón viene definido por la ecuación siguiente:

$$\text{output} = \begin{cases} 0 \text{ if } \sum_j w_j x_j \leq \text{threshold} \\ 1 \text{ if } \sum_j w_j x_j > \text{threshold} \end{cases} \quad (1)$$

Donde el valor umbral o *threshold* es un número real como los pesos que representa el parámetro de la neurona. Será con la variación de los pesos y el umbral, que podremos generar diferentes arquitectos para la toma de decisiones[11].

Ahora, supongamos que tenemos una red de perceptrones que nos gustaría usar para aprender a resolver un problema como la generación de un dígito a partir de la entrada de una imagen con un número. Para que la red logre este resultado necesitará realizar diferentes cambios en los pesos hasta que se logre obtener una salida que presente una precisión aceptable. Esto es algo que un perceptrón no puede conseguir, ya que un cambio en cualquiera de los pesos comportaría un resultado completamente distinto al esperado[11,12].

Cuando queremos resolver problemas como el anterior, recurriremos a un tipo de neurona artificial llamada neurona sigmoidea. Estas varían de los perceptrones en que permiten pequeños cambios en los pesos sin que se modifique el comportamiento de la neurona por completo[12].

Esquemáticamente, una neurona sigmoidea funciona parecido a los perceptrones solo que, en vez de aceptar entradas binarias, ahora se permite cualquier valor entre el cero y el uno. Además, en este caso, la salida tampoco será binaria, sino que será el producto de la función siguiente[12]: $\sigma(w \cdot x + b)$, donde $\sigma(z) \equiv \frac{1}{1+e^{-z}}$.

Entendiendo que existes diferentes tipos de redes neuronales, y antes de pasar a describir las utilizadas en este trabajo, es necesario mostrar la estructura de una red neuronal simple que nos permitirá clasificar dígitos y resolver problemas como el anteriormente mencionado.

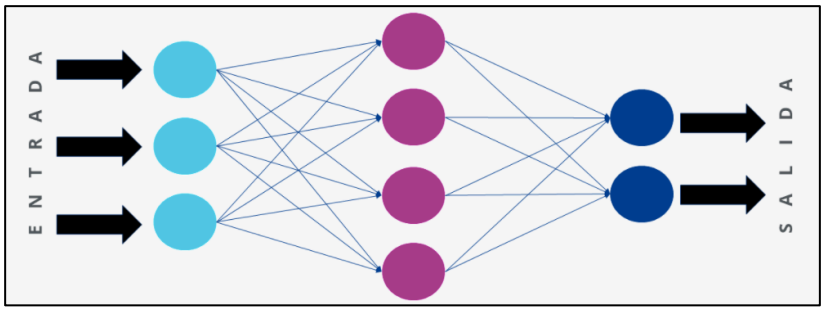


**Figura 5. Esquema de una red neuronal simple. En azul claro se identifican las neuronas de la capa de entrada, mientras que en azul oscuro se representan las de salida. Las neuronas violetas pertenecen a las capas ocultas. Fuente: IONOS**

La Figura 5 muestra que la primera capa es la capa de entrada y, los neuronas que la conforman son las neuronas de entrada. Además, también se puede reconocer la capa de salida con sus neuronas permitentes, en este caso dos. En el medio se representa la capa llama oculta. En este caso solo hay una, pero se pueden ir añadiendo más a la hora que vamos haciendo al modelo más complejo.

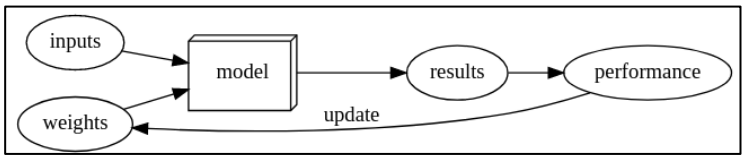


**Figura 6. Esquema de un modelo de aprendizaje entrenándose. Fuente: Curso fast.ai**

Al final, lo que queremos conseguir es una red neuronal recurrentes que vaya "aprendiendo" y modificando los pesos hasta obtener una arquitectura con gran precisión y utilidad.

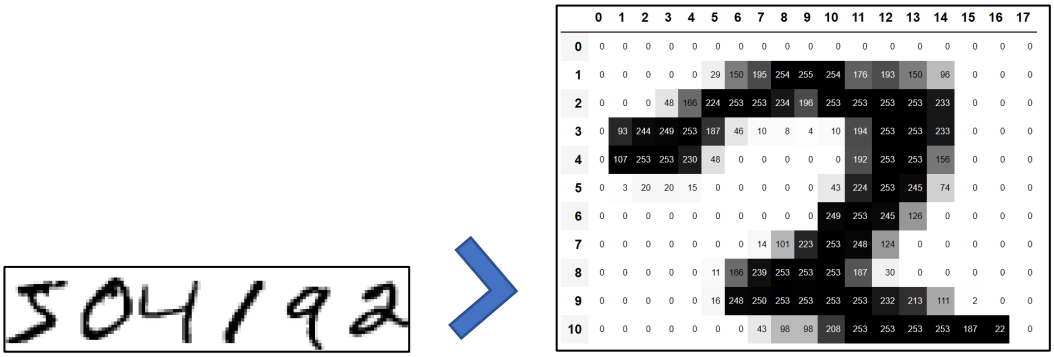

**Figura 7. Proceso de identificación píxel a píxel. Fuente: Curso fast.ai**

Si tomamos el ejemplo anterior. La red neuronal que nos interesa cogerá un dígito manuscrito y le aplicará un peso a cada píxel indicando su intensidad (Fig. 7). Finalmente, se entrenará el modelo para que identifique el número al que representa cada imagen.

### 2.2.1 Redes Neuronales Convolucionales

En el caso de trabajar con imágenes. Las redes neuronales convolucionales son el algoritmo de aprendizaje profundo más utilizado. Uno de los principales inconvenientes de la red neuronal sigmoidea creada anteriormente, es que aplica los pesos sin distinguir la distancia entre cada píxel. Las CNN, en cambio, sí que trabajan con la estructura espacial de las imágenes durante el entrenamiento del modelo con el conjunto de datos[12].

Principalmente, las redes neuronales convolucionales utilizan tres ideas básicas:

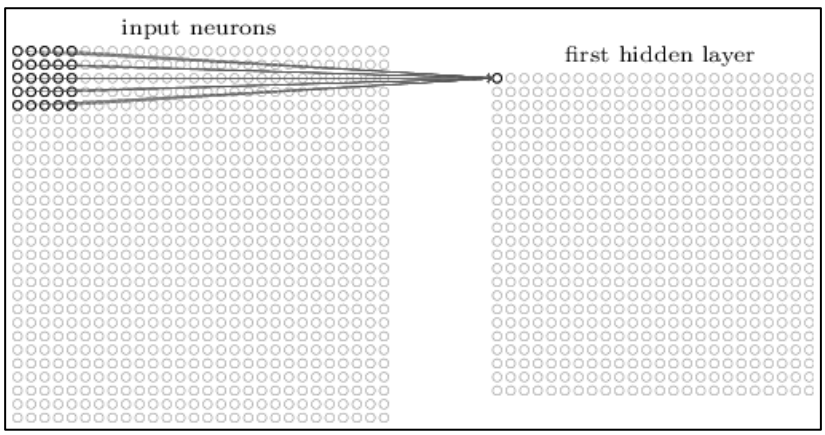


**Figura 8. Proceso de creación de la primera capa oculta a partir de la capa de entrada de una CNN. Fuente:** ***Neural Networks and Deep Learning***

En primer lugar, las CNN trabajan con campos receptivos locales. Esto significa que, en vez de entender la capa de entrada como una línea vertical de neuronas, utiliza cuadrados o "matrices" de neuronas que representan cada píxel (Fig. 8). Para poder trabajar con diferentes imágenes, es común redimensionalizarlas. Cuando se va a generar la primera capa oculta, se deslizará el campo receptivo para la primera neurona oculta y así consecutivamente para las siguientes[12].

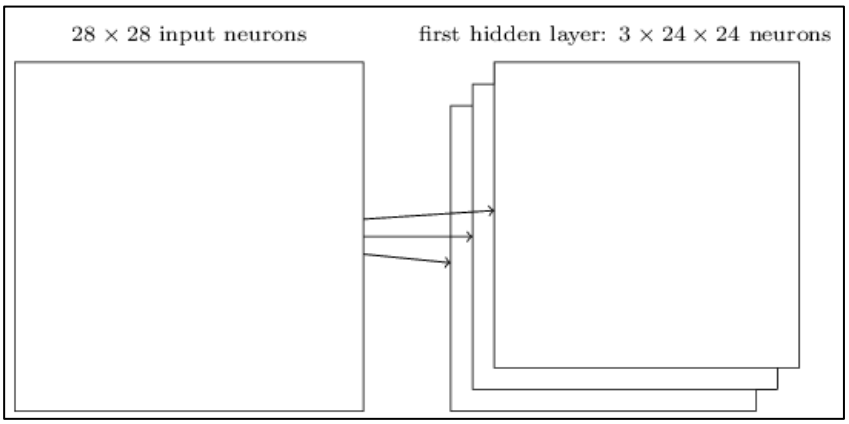


**Figura 9. Esquema de las capas ocultas para trabajar con tres características. Fuente:** ***Neural Networks and Deep Learning***

En segundo lugar, las CNN presentan pesos compartidos. Es decir, todas las neuronas de una capa detectan exactamente la misma característica (por ejemplo, el borde de la imagen o algún tipo de forma). Es evidente que si una capa describe a solo una característica (pongamos por ejemplo el color), la red neuronal convolucional necesitará de más de un mapa de características para trabajar con imágenes a color (Fig. 9).

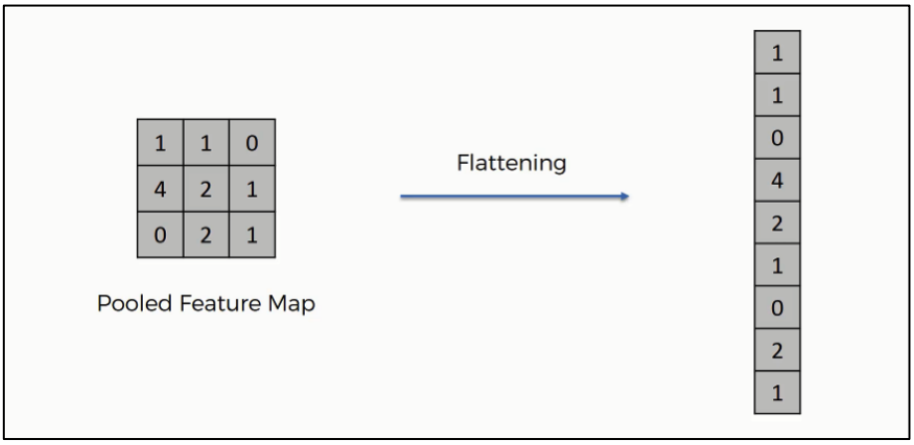


**Figura 10. Proceso de agrupamiento. Fuente:** ***Neural Networks and Deep Learning***

Finalmente, la CNN deberá encargarse de agrupar los resultados. De juntar todas las ideas para generar una red neuronal completa. En la figura 10 podemos ver como se realiza este proceso, mientras que en la figura 11 se ve el esquema de toda la red neuronal convolucional donde la última capa de en la red es una capa completamente conectada. Es decir, esta capa conecta todas las neuronas de la capa combinada máxima a cada una de las neuronas de salida (aunque en la imagen solo se represente una única flecha)[12].

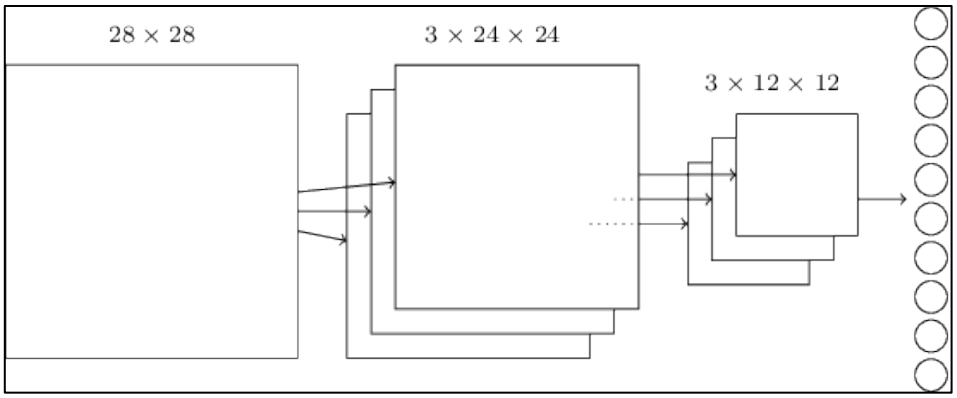


**Figura 11. Esquema de una red neuronal convolucional completa. Fuente: *Neural Networks and Deep Learning***

#### 2.2.1.1 Aprendizaje semisupervisado o *Semi-Supervised Learnind*

Los algoritmos de clasificación tradicionales necesitan de un conjunto de datos de entrenamiento previamente identificados y etiquetados. Sin embargo, con el aumento de los datos generados gracias a la mejora de las técnicas y metodologías empleadas, es cada vez más difícil y costoso (no solo económicamente hablando, sino también por el excesivo tiempo requerido) etiquetar manualmente los datos obtenidos. Contrariamente, la generación de datos sin etiquetar se vuelve cada vez más efectiva, incluso si su uso es más restrictivo[18].

El aprendizaje automatizado presenta tres diferentes procesos que permiten la identificación de patrones dentro de nuestros datos para generar modelos o arquitecturas capaces de clasificar nueva información[19].

En primer lugar, el aprendizaje supervisado (en inglés, *Supervised Learning*). Éste proceso genera un modelo de aprendizaje automático a partir de un conjunto de datos etiquetados, es decir, un conjunto de datos en el que se conoce la variable objetiva. Ejemplos de esta técnica son la clasificación y la regresión[19].

En segundo lugar, el aprendizaje no supervisado (en inglés, *Unsupervised Learning*). Éste entrena un modelo a partir de datos donde se desconoce la variable objetivo, cuando los datos no están etiquetados. Aquí, la arquitectura permite encontrar los patrones más relevantes dentro de los datos. Dentro del aprendizaje sin supervisión se encuentra el clustering, la segmentación, la reducción de dimensionalidad. Parecido a este, el aprendizaje autosupervisado (en inglés, *Self-Supervised Learning*), modifica los datos sin etiquetas para hacer tareas supervisadas e inferir sobre la estructura de los datos[19].

Finalmente, los procesos de aprendizaje semisupervisado (en inglés, *Semi-Supervised Learning*) combinan aspectos de los procesos sin y con supervisión anteriores[19].

Los algoritmos de aprendizaje semisupervisado permiten sobrellevar este inconveniente utilizando conjuntos con un número elevado de datos sin etiquetar junto con un conjunto menor de datos ya etiquetados para construir clasificadores eficientes y precisos. Al emplear ambos tipos de datos, el aprendizaje semisupervisado requiere de menor implicación y esfuerzo por parte del personal técnico para proporcionar una gran precisión. Las técnicas de aprendizaje semiautomático utilizan conjuntos de datos sin clasificar para reforzar, y en caso de que sea necesario modificar acordemente, el modelo generado a partir de los datos ya etiquetados[18].

El aprendizaje semisupervisado, al utilizar datos sin etiquetar moderadamente fáciles de encontrar, comporta un gran interés en el aprendizaje automático para mejorar las tareas de aprendizaje supervisado cuando los datos etiquetados son escasos o costosos[19].

En aquellos casos donde no se disponga de grandes cantidades de datos, el uso de redes neuronales como los codificadores automáticos o *autoencoders* permiten aumentar el tamaño del conjunto de datos y así permitir el entrenamiento de redes neuronales supervisadas[9,13].

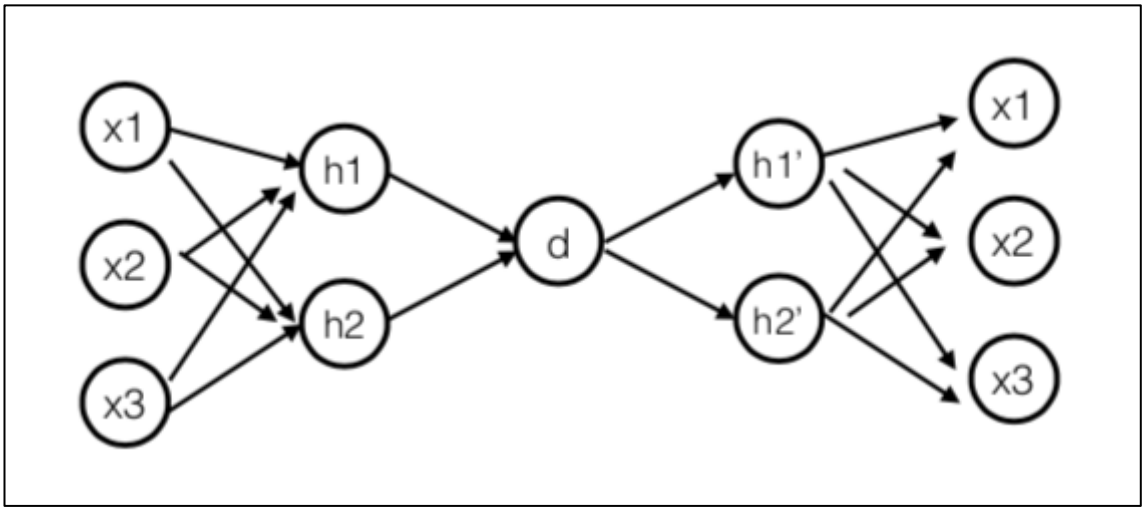


**Figura 12. Esquema de una Red Neuronal con Codificadores Automáticos. Fuente: Kaggle.com**

Los codificadores automáticos son un tipo especial de arquitecturas de redes neuronales en las que la salida es la misma que la entrada (Fig. 12). Estas redes tienen un estrecho cuello de botella, lo que las obliga a crear representaciones efectivas que comprimen la entrada en un código de baja dimensión que el decodificador puede utilizar para reproducir la entrada original. Al ser entrenados con imágenes sin etiquetas, se considera una herramienta de aprendizaje sin supervisión[9,13].

Para la realización de este trabajo, durante la etapa de entrenamiento de la red neuronal profunda, partes de la imagen original, dado que se tiene un conjunto de datos de imágenes no corruptas, se han corrompido estas con algo de ruido, por lo que se posibilita calcular algún tipo de distancia entre la imagen original y la ruidosa, donde la imagen original es la señal de supervisión. En este sentido, los codificadores automáticos son herramientas utilizadas junto con una red neuronal tradicional. Se combinan aprendizajes sin y con supervisión.

# 3. Materiales y Métodos

## 3.1 Configuración del entorno de trabajo

Como se ha mencionado anteriormente, para la correcta realización de este trabajo se debe preparar el entorno con el que se pretende construir las redes neuronales. En este caso se deberá constar de una cuenta de correo de Google y acceso a internet.

El trabajo en sí se ha llevado a cabo en Google Colab. Colab es un servicio en la nube, basado en los Notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de Google. Con él, se puede crear cuadernos (o importar los ya creados o importados del exterior) para trabajar en cualquier momento.

## 3.2 Conjunto de datos

El conjunto de datos utilizado ha sido obtenido a través del *Journal* de *Data in Brief* de Andrea Acevedo[17]. Estos presentan imágenes de células de muestras de frotis de sangre periférica (sangre que circula por todo el cuerpo) de distintos pacientes obtenidas por el laboratorio *CORE* del *Hospital Clínic de Barcelona*.

El conjunto de datos en cuestión contiene imágenes de:

- Basófilos: tipo de leucocitos circulantes derivados de la médula ósea. Son células mononucleares muy granulares.
- Eritroblastos: células nucleadas que se encuentran en la médula roja como una etapa o etapas en el desarrollo del glóbulo rojo o eritrocito.
- Linfocitos: tipo de glóbulos blanco (leucocitos) de fundamental importancia en el sistema inmunológico, ya que son las células que determinan la especificidad de la respuesta inmune a microorganismos infecciosos y otras sustancias extrañas.
- Neutrófilos: tipo de glóbulos blanco (leucocitos) que se caracterizan por su papel como mediadores de las respuestas inmunitarias frente a microorganismos infecciosos.
- Eosinófilos: tipo de glóbulos blanco (leucocitos) que se caracterizan por su papel en la mediación de ciertos tipos de reacciones alérgicas.
- Granulocitos Inmaduros (en inglés, *Immature granulocytes* o Ig): tipo de glóbulos blanco (leucocitos) que no se han desarrollado completamente antes de ser liberados de la médula ósea a la sangre. Pueden incluir metamielocitos, mielocitos y promielocitos.
- Monocitos: tipo de glóbulos blanco (leucocitos) que se caracterizan por ser el tipo más grande de leucocitos y poder diferenciarse en macrófagos y células dendríticas de linaje mieloide.
- Plaquetas, también llamada trombocito: componentes sanguíneos incoloros y no nucleados que son importantes en la formación de coágulos sanguíneos (coagulación). Las plaquetas se encuentran solo en la sangre de los mamíferos.

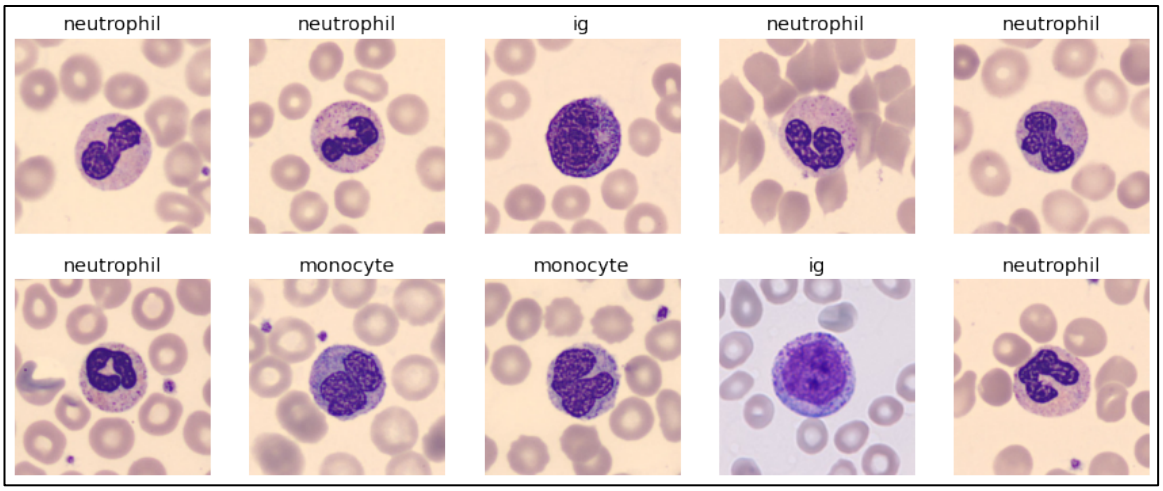


**Figura 13. Muestra del conjunto de datos**

La figura 13 nos muestra un ejemplo de las imágenes que podemos encontrar en nuestro conjunto de datos.

## 3.3 Red Neuronal Convolucional

Para la creación, el entrenamiento y uso de la CNN se ha seguido el siguiente procedimiento.

En primer lugar, se ha configurado el servidor y se han cargado los datos. Al trabajar con un servidor en la nube, se ha tenido que conectar Colab con la cuenta de Google que contenía el conjunto de datos. Para ello, se ha utilizado el siguiente código:

```
from google.colab import drive
drive.mount('/content/drive')
```

**Código 1. Comandos para conectarse a Google Drive**

En este caso, el conjunto de datos estaba disponible dentro de una carpeta comprimida. Para poder trabajar con él, primero se ha tenido que descomprimir la carpeta y acceder a los datos.

```
# accedemos al directorio con el conjunto de datos
cd /content/drive/MyDrive/Universitat/TFM
# descomprimimos el conjunto de datos
!unzip  PBC_dataset_norma_DIB.zip
# accedemos al conjunto de datos
cd PBC_dataset_normal_DIB/
```

**Código 2. Comandos para descomprimir el archivo y acceder al mismo**

Con los datos ya accesibles, pasaremos a realizar nuestra CNN. Para poder realizar este test, necesitaremos importar ciertos complementos a Colab.

Para la creación de nuestra red neuronal, utilizaremos la librería fast.ai. Fastai es una biblioteca de aprendizaje profundo que proporciona componentes aplicables a modelos de alto nivel que proporcionan de manera rápida y fácil resultados de vanguardia. Además, también consta de componentes de bajo nivel aptos para el aprendizaje[25].

```
!pip install -Uqq fastbook
```

```
import fastbook
fastbook.setup_book()
from fastbook import *
import fastai
```

**Código 3. Comandos para importar las librerías necesarias**

Con las librerías cargadas, ya podemos comenzar a crear nuestra red neuronal convolucional que nos permitirá clasificar e identificar el tipo de célula a partir de una imagen digital.

El conjunto de datos con el que se ha trabajado identifica el tipo de célula a partir de la carpeta en la que está presente. Por esta razón, a la hora de importar todos los datos para el modelo (*data loaders* o dls), utilizaremos una función especial de fastai.

```
from fastai.vision.all import *
path =
"/content/gdrive/MyDrive/Universitat/TFM/PBC_dataset_normal_DIB"

dls =
ImageDataLoaders.from_folder(path, valid_pct=0.2,
item_tfms=Resize(224))
```

**Código 4. Comandos para importar los datos**

Con los datos ya importados, solo falta entrenar el arquitecto. Este proceso en fastai es muy sencillo y con una simple línea ya conseguirás crear y entrenar tu red neuronal convolucional.

```
#id first_training
#caption Results from the first training

learn = cnn_learner(dls, resnet18, metrics=accuracy)
```

**Código 5. Comandos para entrenar la CNN**

## 3.4 Red Neuronal con Semi-Supervised Learning

Con tal de aplicar técnicas de *semi-supervised learning* a nuestro modelo anterior. Se ha decidido generar un código nuevo sin el uso de la librería fastai. A partir de ahora, la librería utilizada será keras. Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python capaz de ejecutarse sobre TensorFlow.

```
import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="0" #model will be trained on
GPU 0

import keras
from matplotlib import pyplot as plt
import numpy as np
import gzip
%matplotlib inline
from keras.models import Model
from keras.optimizers import RMSprop
```

```
from keras.layers import
Input,Dense,Flatten,Dropout,merge,Reshape,Conv2D,MaxPooling2D,UpS
ampling2D,Conv2DTranspose
from keras.layers.normalization import BatchNormalization
from keras.models import Model,Sequential
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adadelta, RMSprop,SGD,Adam
from keras import regularizers
from keras import backend as K
from tensorflow.keras.utils import to_categorical
```

**Código 6. Comandos para importar las librerías necesarias**

Para empezar con nuestra nueva red neuronal, esta vez necesitaremos dividir nuestro conjunto de datos en dos partes. Una parte servirá para entrenar el arquitecto (con los *encoders* y *decoders*), y otra parte nos ayudará a evaluar el rendimiento de todo el modelo. Para ello, utilizando los dls generados anteriormente (Código 6), extraeremos los datos de entrenamiento y de prueba.

```
train = dls.train
test = dls.valid
```

**Código 7. Comandos para extraer los conjuntos de entrenamiento y prueba**

En la figura 14, se puede observar una muestra de las imágenes del conjunto de entrenamiento.

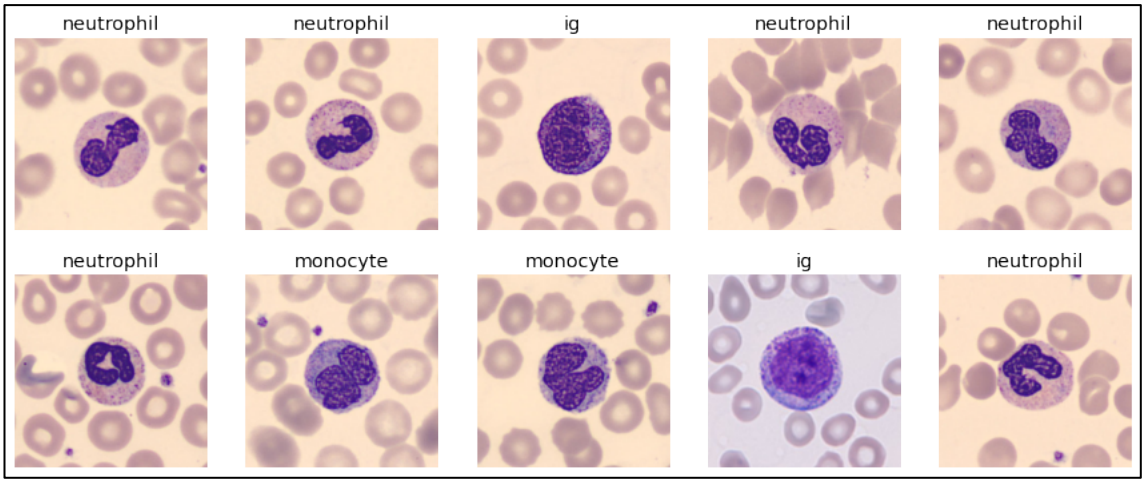


**Figura 14. Muestra de las imágenes disponibles en el conjunto de entrenamiento.**

Con los dos conjuntos de datos, ya podemos a empezar con la creación de nuestra red neuronal con *semi-supervised learning*.

En primer lugar, queremos entrenar los *autoencoder* que nos permitirán corromper las imágenes para luego poder volverlas a corregir. Este es un proceso de aprendizaje profundo que le dará a nuestro modelo la capacidad de identificar y clasificar correctamente datos incompletos o defectuosos y reducirá el porcentaje de sobreajuste (en inglés, *overfitting*).

```
# Data splitting
from sklearn.model_selection import train_test_split
train_X,valid_X,train_ground,valid_ground =
train_test_split(train_data, train_data, test_size=0.2,
                 random_state=13)
```

**Código 8. Comandos para dividir los conjuntos de entrenamiento y prueba**

Para el entrenamiento de esta fase, volveremos a dividir el conjunto de entrenamiento (80% de los datos originales), en dos conjuntos: 80% entrenamiento y 20% prueba. Estos serán los utilizados para entrenar el codificador automático.

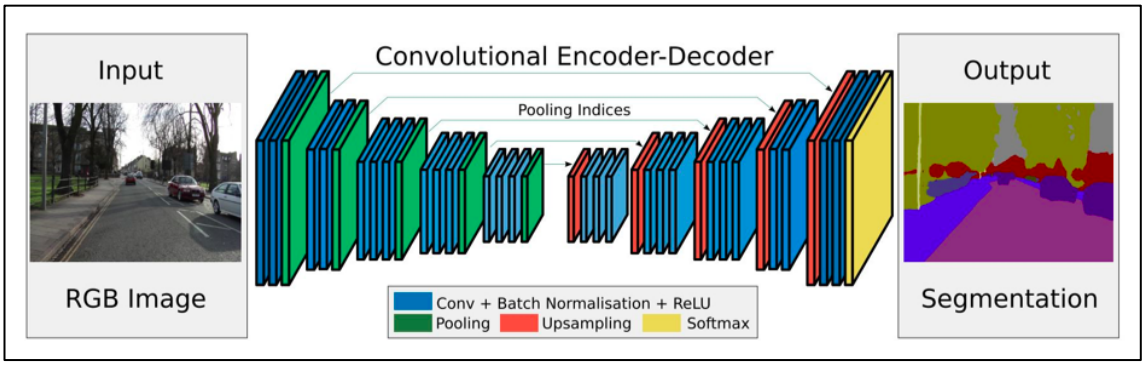


**Figura 15. Esquema del funcionamiento de un codificador (encoder en la izquierda) y decodificador (decoder en la derecha). Fuente: laptrinhx.com**

Una vez definidas las funciones de los *encoder* y *decoder* (Fig. 15), pasaremos a construir y entrenar el modelo (Código 9).

```
# Build and compile the model
autoencoder =
      Model(input_img, decoder(encoder(input_img)))
      autoencoder.compile(loss='mean_squared_error', optimizer =
      RMSprop())
# Train autoencoder
autoencoder_train =
      autoencoder.fit(train_X, train_ground,
      batch_size=batch_size,epochs=epochs,verbose=1,validation_dat
      a=(valid_X, valid_ground))
# Save the model
autoencoder.save_weights('autoencoder.h5')
```

**Código 9. Comando para construir, entrenar y guardar el modelo con los codificadores automáticos**

El modelo de red neuronal que se utiliza en este trabajo necesita que los datos se presenten segmentados. Una vez se hayan pasado las etiquetas de las imágenes (tipos celulares) de categóricas a la codificación *one-hot*, se volverá a dividir el conjunto de entrenamiento original tal y como se muestra en el Código 10.

```
# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_labels)
test_Y_one_hot = to_categorical(test_labels)
# Split the data
train_X,valid_X,train_label,valid_label =
```

```
train_test_split(train_data,train_Y_one_hot,test_size=0.2,ra
ndom_state=13)
```

**Código 10. Comando para re-codificar los datos y dividir en subconjuntos de entrenamiento y de prueba**

Con el proceso realizado hasta ahora tenemos un modelo que permite corromper y corregir las imágenes que se le entregan. Ahora, deberemos introducirlo dentro de una red neuronal como técnica de *semi-supervised learning*. Para eso, primero se deberá crear una red neuronal con *dense layers* (Código 11).

```
# Build the model
encode = encoder(input_img)
full_model = Model(input_img,fc(encode))
# loading the weights from the encoder part of the autoencoder
trained before
for l1,l2 in
zip(full_model.layers[:19],autoencoder.layers[0:19]):
    l1.set_weights(l2.get_weights())
# Freezing the encoder layers to not train them in this second
neural net
for layer in full_model.layers[0:19]:
    layer.trainable = False
# Compile the model
full_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
```

**Código 11. Comando para crear la red neuronal earlier encoder + Dense layers**

Finalmente, una vez tenemos el modelo final de nuestra red neuronal, solo faltará entrenarlo y obtener los resultados.

Con el código del Código 12 se podrá entrenar la red neuronal y evaluar su precisión.

```
# Training the net 2
classify_train = full_model.fit(train_X, train_label,
batch_size=64,epochs=1,verbose=1,validation_data=(valid_X,
valid_label))
# Evaluation
test_eval = full_model.evaluate(test_data, test_Y_one_hot,
verbose=0)
```

**Código 12. Comando para entrenar y evaluar la red neuronal con semi-supervised learning**

# 4. Resultados y discusión

A partir de la metodología empleada para la creación de las dos redes neuronales mostradas en este trabajo. Se presentan los resultados obtenidos y se comparan entre sí. En esta sección se presentan las gráficas y tablas más representativas obtenidas a partir de los códigos disponibles los anexos A.1 y A.2.

## 4.1 Red Neuronal Convolucional

A continuación, se exponen los resultados obtenidos por el modelo de CNN incluido en la librería fastai. Con la red neuronal convolucional se han obtenido los resultados que se muestran en la Tabla 3. Tal como se puede observar, la precisión alcanzada sobre el grupo de validación es superior al 95%.

**Tabla 3. Resultados de la CNN**

| Epoch | Train_Loss | Valid_Loss | Accuracy | Time |
|---|---|---|---|---|
| 1 | 0.236733 | 0.135956 | 0.956095 | 02:53 |

Para poder determinar la tasa de aprendizaje óptima de la red neuronal, en la Figura 16 se muestra la pérdida en función de la tasa de aprendizaje.

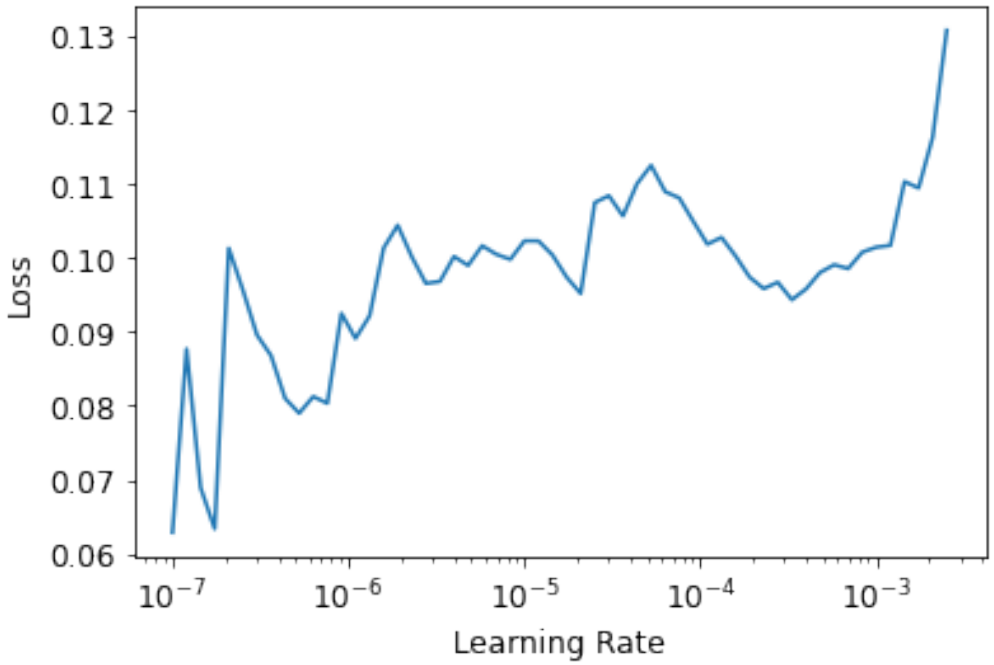


**Figura 16. *Learning rate finder* de la CNN**

Una vez se ha entrenado la red neuronal, se ha generado una matriz de confusión que permite visualizar la sensibilidad y especificidad que tiene el modelo para cada clase (Fig. 17). Se observa que se han predicho correctamente 1889 imágenes, mientras que 81 han sido mal identificadas. Cabe destacar que en 48 ocasiones se ha clasificado erróneamente como anticuerpo una célula que no lo era, siendo esta la categoría con más predicciones incorrectas.

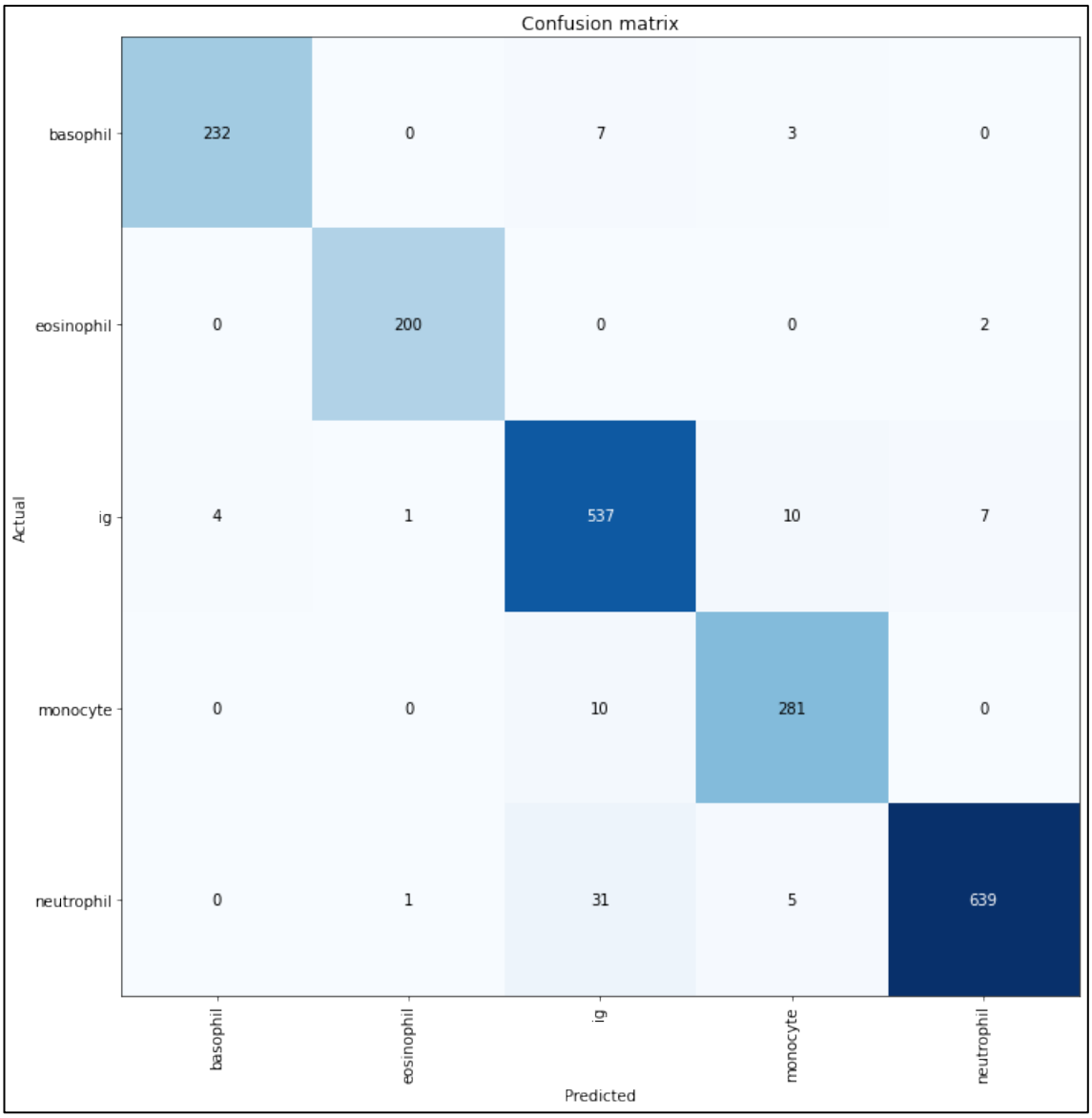


**Figura 17. Matriz de confusión de la CNN**

### 4.1.1 Discusión de la CNN

A partir del apartado anterior, se considera que los resultados obtenidos por la CNN han sido satisfactorios, puesto que la precisión con la que la red neuronal consigue clasificar correctamente una imagen digital de una célula del sistema inmunitario es del 95% (Tabla 3). Sin embargo, esta arquitectura es muy específica. Durante su entrenamiento, solo se ha trabajado con imágenes de linfocitos y, por lo tanto, a la hora de trabajar con nuevos conjuntos de datos y poner en práctica el modelo, su aplicación es limitada. Adicionalmente, actualmente la red neuronal se considera estática, lo cual puede comportar un problema para su uso en un futuro con el diseño de este trabajo.

## 4.2 Red Neuronal con *Semi-Supervised Learning*

A continuación, se exponen los resultados obtenidos por el modelo de red neuronal que emplea técnicas de *semi-supervised learning*. Como se menciona anteriormente, la librería empleada para esta tarea ha sido keras. En el caso de esta red neuronal, se ha visto necesaria la ampliación de ciclos de ejecución para mejorar la precisión del modelo. Tal como se puede observar en la Tabla 4, la precisión alcanzada sobre el grupo de validación

durante el primer ciclo no alcanza el 93.5%, mientras que después del duodécimo se supera el 96%.

**Tabla 4. Resultados de la Red Neuronal con *Semi-Supervised Learning***

| Epoch | Train_Loss | Valid_Loss | Accuracy | Time |
|---|---|---|---|---|
| 0 | 0.1767 | 0.5462 | 0.9338 | 00:05 |
| 1 | 0.1718 | 0.5433 | 0.9380 | 00:10 |
| 2 | 0.1591 | 0.5732 | 0.9411 | 00:15 |
| 3 | 0.1472 | 0.5710 | 0.9439 | 00:20 |
| 4 | 0.1424 | 0.6026 | 0.9461 | 00:25 |
| 5 | 0.1381 | 0.6817 | 0.9490 | 00:30 |
| 6 | 0.1317 | 0.6327 | 0.9505 | 00:35 |
| 7 | 0.1270 | 0.6491 | 0.9518 | 00:40 |
| 8 | 0.1236 | 0.6705 | 0.9543 | 00:45 |
| 9 | 0.1194 | 0.7221 | 0.9562 | 00:50 |
| 10 | 0.1332 | 0.7739 | 0.9546 | 00:55 |
| 11 | 0.1064 | 0.7492 | 0.9604 | 01:00 |

Para poder determinar la tasa de aprendizaje óptima de la red neuronal, en la Figura 18 se muestran, tanto la pérdida en función de la tasa de aprendizaje como la evolución de la precisión del modelo.

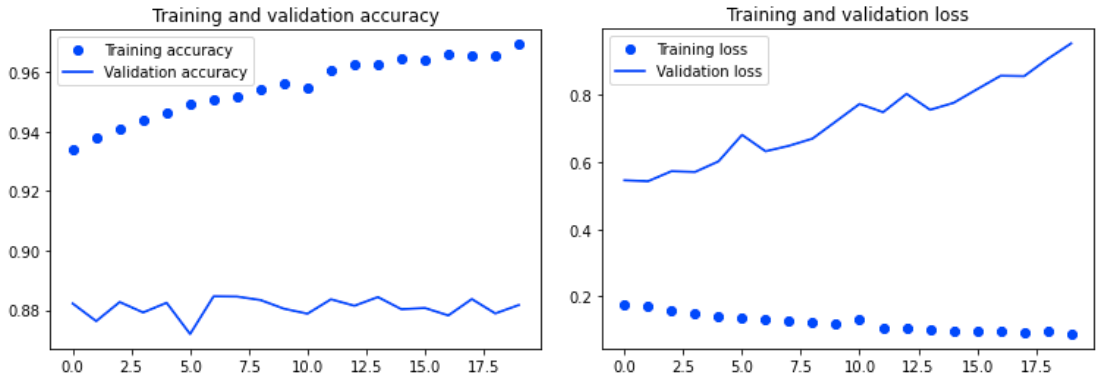


**Figura 18. Evolución de la precisión (gráfica de la izquierda) y pérdida (gráfica de la derecha) de la Red Neuronal Profunda**

Si se quiere analizar de manera manual las predicciones que genera el modelo no supervisado, en la Figura 19 se muestra un fragmento de los datos clasificados. Como se puede observar, en este caso todas las imágenes han sido correctamente identificadas por la red neuronal.

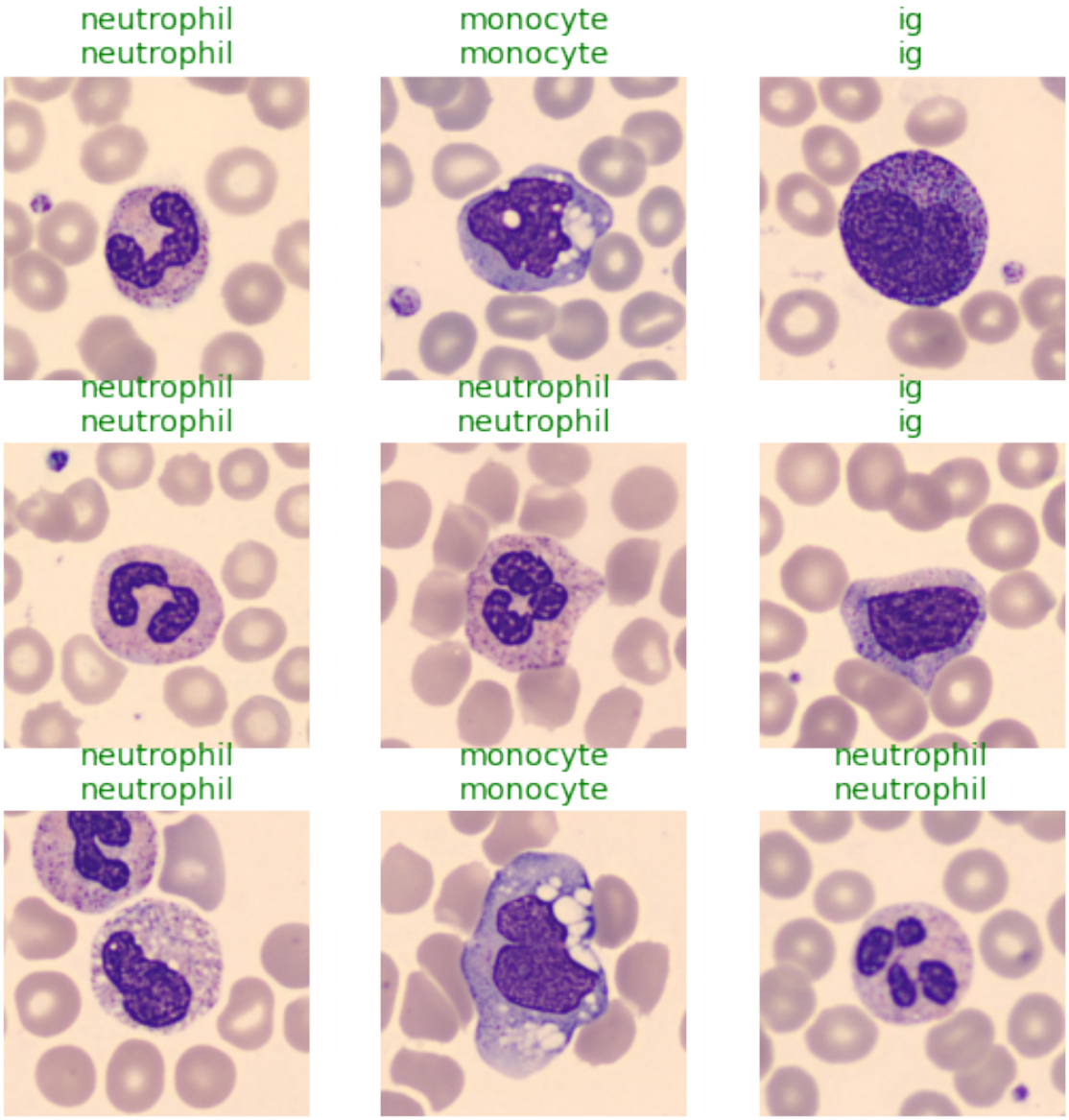


**Figura 19. Muestra de las predicciones generadas por la Red Neuronal Profunda**

### 4.2.1 Discusión de la SSL

A partir del apartado anterior, se considera que los resultados obtenidos por la Red Neuronal con técnicas de *semi-supervised learning* han sido satisfactorios, puesto que la precisión con la que la red neuronal consigue clasificar correctamente una imagen digital de una célula del sistema inmunitario es del 96% (Tabla 4). Sin embargo, esta precisión es obtenida después de 12 iteraciones o ciclos (en inglés, *epoch*). Si nos fijamos en los resultados de la Tabla 4, veremos que, durante el primer ciclo la precisión era del 93%.

Para la creación de esta red neuronal, se ha utilizado una arquitectura diseñada con la librería *keras*, que permitiera la realización de dos procesos distintos. En primer lugar, el entrenamiento de un codificador automático (en inglés, *autoencoder*) que funcionase con el conjunto de datos de este trabajo y, en segundo lugar, el entrenamiento de una red neuronal que aplicara entre sus pasos el *autoencoder* anterior. Todos estos pasos han influido en la eficacia final del modelo generado.

Como se menciona con la red neuronal convolucional, durante el entrenamiento de esta red, se ha trabajado con imágenes de linfocitos (aunque fastai preentrene el modelo con

el conjunto de datos de ImageNet) y, por lo tanto, antes de trabajar con un nuevo conjunto de imágenes, será necesario asegurar que todos los datos muestran glóbulos blancos. A diferencia de la red neuronal anterior, en este caso no estamos delante de una red estática y su aplicabilidad en el futuro es más amplia.

### 4.3 Comparativa entre Redes Neuronales

Finalmente, observando los resultados obtenidos por las dos redes neuronales generadas durante este trabajo, se puede considerar que, la red neuronal convolucional presenta una mejor relación entre el esfuerzo y el resultado obtenido.

A partir de los resultados de las Tablas 3 y 4, se puede ver como con un *epoch*, se obtiene una precisión de más del 95% en la red neuronal convolucional, mientras que para la red que aplica técnicas de *semi-supervised learning*, solamente se supera el 95% después de 7 ciclos. Adicionalmente, al utilizar la librería *fastai* para el diseño de la CNN, se dispone de un comando capaz de realizar el entrenamiento y la validación de la red neuronal con una única línea de código, acortando así el tiempo y esfuerzo necesarios para obtener resultados idóneos.

Sin embargo, existen varios aspectos de la red neuronal profunda que la harían una mejor opción para su uso en según qué situaciones. Esta red neuronal utiliza codificadores automáticos, que presentan una capa de entrada, una capa oculta y una capa de salida lo más idéntica posible a la de entrada.

La capa oculta es la que permite a la red neuronal aprender una representación (codificación) comprimida y distribuida de un conjunto de datos. O lo que es lo mismo, los codificadores automáticos reducen la dimensionalidad de los datos y, como interesa en este trabajo, permiten la detección de anomalías y ruido en las imágenes. Esta faceta es la que hace posible para la red neuronal trabajar con conjuntos de datos de menor tamaño o con imágenes de una calidad inferior sin que la precisión del modelo se vea afectada. Además, al hacer uso de la librería keras, más tradicional que fastai, la red neuronal semisupervisada no utiliza *transfer learning* y, sin embargo, la precisión sigue siendo muy alta respecto a la red generada con fastai. Éste es un método de aprendizaje automático en el que un modelo desarrollado para una tarea se reutiliza como punto de partida para un modelo en una segunda tarea.

Finalmente, para clasificar imágenes similares a la del conjunto de datos utilizado en este trabajo, se considera mucho más factible el uso de algoritmos que utilicen comandos de la librería *fastai*, como el utilizado para el entrenamiento y validación de la red neuronal convolucional mostrado en el Anexo 1. Con una complejidad muy inferior al del Anexo 2, se consiguen resultados aceptables y, lo que es más importante, útiles para el uso clínico y la investigación científica.

# 5. Conclusiones

Con la metodología mencionada anteriormente, y tal y como se corrobora en los resultados, se han logrado desarrollar y cumplir los objetivos de este trabajo.

Se ha realizado una clasificación con redes neuronales convolucionales para clasificar imágenes digitales de células de sangre periférica y desarrollado un modelo de redes neuronales convolucionales que aplica técnicas de *semi-supervised learning* para la clasificación de estas mismas imágenes. Finalmente, una vez generados los modelos, se han comparado y mejorado el entendimiento de los métodos del aprendizaje profundo para la clasificación de imágenes digitales. En todos los casos, los sistemas de clasificación del tipo de célula son de alto desempeño, con una buena precisión y rapidez.

Ambas arquitecturas realizadas presentan ventajas y desventajas respecto a la clasificación de imágenes digitales y, por lo tanto, posibles líneas de investigación futura que ayuden a relajar la carga de trabajo del personal sanitario con el análisis de grandes bases de datos. Si bien, la aplicación de técnicas de aprendizaje profundo puede aumentar la dificultad del problema, se ha demostrado que son una alternativa real a las arquitecturas más simples, y presentan perspectivas de futuro importantes.

Con este trabajo, se ha corroborado la posibilidad de automatizar pasos esenciales para la detección precoz de enfermedades como la leucemia a partir del uso de herramientas de aprendizaje automatizado, y que la única restricción real a su potencial es la capacidad computacional que el investigador tiene a su alcance.

Uno de los puntos que se podrían desarrollar más sobre este trabajo es la creación de la red neuronal profunda. En este momento, su entrenamiento y validación se basan en comandos y código basados en la librería *keras* y, un aspecto que se ha visto claro con la creación de la CNN es que la librería *fastai* permite realizar procesos complejos mediante comandos y líneas relativamente simples y lógicas. Sería conveniente comprobar la efectividad de crear una red neuronal convolucional aplicando técnicas de *semi-supervised learning* dentro de *fastai*. Otro aspecto a estudiar en un futuro sería la implicación de que la red neuronal semisupervisada no utiliza *tranfer learning* y, aun así, la precisión no parece estar afectada.

Debido a la limitación que ha sido el tiempo, no se ha podido profundizar en la creación y uso de redes neuronales profundas. Un punto claro después de la finalización de este trabajo ha sido que la planificación inicial era relativamente optimista. El tiempo requerido para adaptarse al uso de una nueva librería y para aprender a trabajar con un conjunto de datos como el empleado, al final ha sobrepasado el esperado y ha comportado que los resultados, si bien positivos, no se ajusten perfectamente a los deseados.

## 6. Bibliografía


1. Milletari, F., Navab, N. & Ahmadi, S.-A. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. *Proc. - 2016 4th Int. Conf. 3D Vision, 3DV 2016* 565–571 (2016).

2. Zhang, X. *et al.* Real-time gastric polyp detection using convolutional neural networks. *PLoS One* **14**, e0214133 (2019).

3. Xu, J. *et al.* Stacked sparse autoencoder (SSAE) for nuclei detection on breast cancer histopathology images. *IEEE Trans. Med. Imaging* **35**, 119–130 (2016).

4. Christodoulidis, S., Anthimopoulos, M., Ebner, L., Christe, A. & Mougiakakou, S. Multisource Transfer Learning with Convolutional Neural Networks for Lung Pattern Analysis. *IEEE J. Biomed. Heal. Informatics* **21**, 76–84 (2017).

5. Ma, J., Wu, F., Jiang, T., Zhu, J. & Kong, D. Cascade convolutional neural networks for automatic detection of thyroid nodules in ultrasound images. *Med. Phys.* **44**, 1678–1691 (2017).

6. Liang, X., Yu, J., Liao, J. & Chen, Z. Convolutional Neural Network for Breast and Thyroid Nodules Diagnosis in Ultrasound Imaging. *Biomed Res. Int.* **2020**, (2020).

7. Lang, I., Sklair-Levy, M. & Spitzer, H. Multi-scale texture-based level-set segmentation of breast B-mode images. *Comput. Biol. Med.* **72**, 30–42 (2016).

8. Grewal, P. D. S. A Critical Conceptual Analysis of Definitions of Artificial Intelligence as Applicable to Computer Engineering. *IOSR J. Comput. Eng.* **16**, 09–13 (2014).

9. Chen, L. *et al.* Self-supervised learning for medical image analysis using image context restoration. *Med. Image Anal.* **58**, 101539 (2019).

10. McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **5**, 115–133 (1943).

11. Rosenblatt, F. Perceptron Simulation Experiments. *Proc. IRE* **48**, 301–309 (1960).

12. Nielsen, M. A. Neural Networks and Deep Learning. (2015).

13. Gogna, A. & Majumdar, A. Semi supervised autoencoder. in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) vol. 9948 LNCS 82–89 (Springer Verlag, 2016).

14. Rose, N. Personalized Medicine: Promises, Problems and Perils of a New Paradigm for Healthcare. *Procedia - Soc. Behav. Sci.* **77**, 341–352 (2013).

15. Turing, A. M. COMPUTING MACHINERY AND INTELLIGENCE. *Mind* **49**, 433–460 (1950).

16. Amisha, Malik, P., Pathania, M. & Rathaur, V. Overview of artificial intelligence in medicine. *J. Fam. Med. Prim. Care* **8**, 2328 (2019).

17. Acevedo, S Alférez, A Merino, L Puigví, J Rodellar - Computer methods and programs in biomedicine, 2019.

18. ZHU, Xiaojin Jerry. Semi-supervised learning literature survey. 2005.

19. Zhu, X., & Goldberg, A. B. (2009). Introduction to semi-supervised learning. Synthesis lectures on artificial intelligence and machine learning, 3(1), 1-130.

20. Ertam, F. Data classification with deep learning using tensorflow. *2nd Int. Conf. Comput. Sci. Eng. UBMK 2017* 755–758 (2017) doi:10.1109/UBMK.2017.8093521.

21. Sagiroglu, S. & Sinanc, D. Big data: A review. in Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013 42–47 (2013). doi:10.1109/CTS.2013.6567202.

22. Rong, G., Mendez, A., Bou Assi, E., Zhao, B. & Sawan, M. Artificial Intelligence in Healthcare: Review and Prediction Case Studies. *Engineering* vol. 6 291–301 (2020).

23. Jiang, F. *et al.* Artificial intelligence in healthcare: Past, present and future. *Stroke Vasc. Neurol.* **2**, 230–243 (2017).

24. Garcia-Chimeno, Y. & Garcia-Zapirain, B. HClass: Automatic classification tool for health pathologies using artificial intelligence techniques. *Biomed. Mater. Eng.* **26**, S1821–S1828 (2015).

25. Koelzer, V. H., Sirinukunwattana, K., Rittscher, J. & Mertz, K. D. Precision immunoprofiling by image analysis and artificial intelligence. *Virchows Arch.* **474**, 511–522 (2019).

26. Ohu, I., Benny, P. K., Rodrigues, S. & Carlson, J. N. Applications of machine learning in acute care research. *J. Am. Coll. Emerg. Physicians Open* **1**, 766–772 (2020).

27. Howard, J. Self-supervised learning and computer vision. https://www.fast.ai/2020/01/13/self_supervised/

28. Howard, J.; Gugger, S. Fastai: A Layered API for Deep Learning. *Information* **2020**, *11*, 108. https://doi.org/10.3390/info11020108

# 7. Anexos

## 7.1 Red Neuronal Convolucional

**Tabla 5. Código de la Red Neuronal Convolucional**

```
# empezamos conectando con Google Drive
from google.colab import drive
drive.mount('/content/drive')
# a veces, puede ser útil situarnos
!ls
# a continuación, accederemos al directorio de nuestro conjunto
de datos
cd /content/drive/MyDrive/Universitat/TFM
# es importante asegurarnos de que el conjunto de datos está
presente
!ls
# descomprimimos el conjunto de datos
!unzip  PBC_dataset_norma_DIB.zip
# comprobamos que se ha creado la carpeta
!ls
# accedemos al conjunto de datos
cd PBC_dataset_normal_DIB/
# comprobamos que la carpeta contenga efectivamente los datos
!ls
# importamos los complementos y librerías requeridos
!pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
from fastbook import *
import fastai
from fastai.vision.all import *
# indicamos el directorio con el conjunto de datos
path =
      "/content/gdrive/MyDrive/Universitat/TFM/PBC_dataset_normal_
      DIB"
# importamos el conjunto de datos
dls = ImageDataLoaders.from_folder(path, valid_pct=0.2,
item_tfms=Resize(224))
# observamos los tres primeros ítems
dls.valid_ds.items[:3]
# observamos un ejemplo de nuestros datos
dls.show_batch()
# entrenamos la red neuronal convoucional
learn = cnn_learner(dls, resnet18, metrics=accuracy)
learn.fine_tune(1)
learn.lr_find()
# generamos una matriz de confusión (width 600)
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix(figsize=(12,12), dpi=60)
```

## 7.2 Red Neuronal con Semi-Supervised Learning

**Tabla 6. Código de la Red Neurona con Semi-Supervised Learning**

```
# importamos los complementos y librerías principales
import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="0"
import keras
from matplotlib import pyplot as plt
import numpy as np
import gzip
%matplotlib inline
from keras.models import Model
from keras.optimizers import RMSprop
from keras.layers import
Input,Dense,Flatten,Dropout,merge,Reshape,Conv2D,MaxPooling2D,UpS
ampling2D,Conv2DTranspose
from keras.layers.normalization import BatchNormalization
from keras.models import Model,Sequential
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adadelta, RMSprop,SGD,Adam
from keras import regularizers
from keras import backend as K
from tensorflow.keras.utils import to_categorical
# conectamos Colab con Google Drive
from google.colab import drive
drive.mount('/content/drive')
# a veces, puede ser útil situarnos
!ls
# a continuación, accederemos al directorio de nuestro conjunto
de datos
cd /content/drive/MyDrive/Universitat/TFM
# es importante asegurarnos de que el conjunto de datos está
presente
!ls
# descomprimimos el conjunto de datos
!unzip  PBC_dataset_norma_DIB.zip
# comprobamos que se ha creado la carpeta
!ls
# accedemos al conjunto de datos
cd PBC_dataset_normal_DIB/
# comprobamos que la carpeta contenga efectivamente los datos
!ls
# importamos los complementos y librerías requeridos de fastai
!pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
from fastbook import *
import fastai
```

```
from fastai.vision.all import *
# indicamos el directorio con el conjunto de datos
path =
      "/content/gdrive/MyDrive/Universitat/TFM/PBC_dataset_normal_
      DIB"
# importamos el conjunto de datos
dls = ImageDataLoaders.from_folder(path, valid_pct=0.2,
item_tfms=Resize(224))
# observamos un ejemplo de nuestros datos
dls.show_batch()
# reservamos los conjuntos de entrenamiento y de prueba
train = dls.train
test = dls.valid
# mostramos un ejemplo
train.show_batch(max_n=10, nrows=2)
train.dataset
train.vocab
train.dataset[0][0]
train.dataset[0][1], train.vocab[train.dataset[0][1]]
train_data = np.empty((len(train.dataset), 355, 355, 3))
train_labels = np.empty((len(train.dataset)))
test_data = np.empty((len(test.dataset), 355, 355, 3))
test_labels = np.empty((len(test.dataset)))
for i, pair in enumerate(train.dataset):
  train_data[i, :, :, :] = np.array(pair[0])[:355, :355, :]
  train_labels[i] = pair[1].numpy()
for i, pair in enumerate(test.dataset):
  test_data[i, :, :, :] = np.array(pair[0])[:355, :355, :]
  test_labels[i] = pair[1].numpy()
test_labels[1], test.vocab[2]
# Data splitting
from sklearn.model_selection import train_test_split
train_X,valid_X,train_ground,valid_ground =
      train_test_split(test_data, test_data, test_size=0.2,
      random_state=13)
# initial arguments
batch_size = 64
epochs = 1
inChannel = 3
x, y = 355, 355
input_img = Input(shape = (x, y, inChannel))
num_classes = 7
# Encoder
def encoder(input_img):
    #input = 28 x 28 x 1 (wide and thin)
    conv1 = Conv2D(32, (3, 3), activation='relu',
padding='same')(input_img) #28 x 28 x 32
    conv1 = BatchNormalization()(conv1)
```

```
    conv1 = Conv2D(32, (3, 3), activation='relu',
padding='same')(conv1)
    conv1 = BatchNormalization()(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1) #14 x 14 x 32
    conv2 = Conv2D(64, (3, 3), activation='relu',
padding='same')(pool1) #14 x 14 x 64
    conv2 = BatchNormalization()(conv2)
    conv2 = Conv2D(64, (3, 3), activation='relu',
padding='same')(conv2)
    conv2 = BatchNormalization()(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2) #7 x 7 x 64
    conv3 = Conv2D(128, (3, 3), activation='relu',
padding='same')(pool2) #7 x 7 x 128 (small and thick)
    conv3 = BatchNormalization()(conv3)
    conv3 = Conv2D(128, (3, 3), activation='relu',
padding='same')(conv3)
    conv3 = BatchNormalization()(conv3)
    conv4 = Conv2D(256, (3, 3), activation='relu',
padding='same')(conv3) #7 x 7 x 256 (small and thick)
    conv4 = BatchNormalization()(conv4)
    conv4 = Conv2D(256, (3, 3), activation='relu',
padding='same')(conv4)
    conv4 = BatchNormalization()(conv4)
    return conv4
# Decoder
def decoder(conv4):
    conv5 = Conv2D(128, (3, 3), activation='relu',
padding='same')(conv4) #7 x 7 x 128
    conv5 = BatchNormalization()(conv5)
    conv5 = Conv2D(128, (3, 3), activation='relu',
padding='same')(conv5)
    conv5 = BatchNormalization()(conv5)
    conv6 = Conv2D(64, (3, 3), activation='relu',
padding='same')(conv5) #7 x 7 x 64
    conv6 = BatchNormalization()(conv6)
    conv6 = Conv2D(64, (3, 3), activation='relu',
padding='same')(conv6)
    conv6 = BatchNormalization()(conv6)
    up1 = UpSampling2D((2,2))(conv6) #14 x 14 x 64
    conv7 = Conv2D(32, (3, 3), activation='relu',
padding='same')(up1) # 14 x 14 x 32
    conv7 = BatchNormalization()(conv7)
    conv7 = Conv2D(32, (3, 3), activation='relu',
padding='same')(conv7)
    conv7 = BatchNormalization()(conv7)
    up2 = UpSampling2D((2,2))(conv7) # 28 x 28 x 32
    decoded = Conv2D(1, (3, 3), activation='sigmoid',
padding='same')(up2) # 28 x 28 x 1
    return decoded
```

```
# Build and compile the model
autoencoder = Model(input_img, decoder(encoder(input_img)))
autoencoder.compile(loss='mean_squared_error', optimizer =
RMSprop())
# Visualize autoencoder
autoencoder.summary()
# Train autoencoder
autoencoder_train =
      autoencoder.fit(train_X, train_ground,
      batch_size=batch_size,epochs=epochs,verbose=1,validation_dat
      a=(valid_X, valid_ground))
# Save the model
autoencoder.save_weights('autoencoder.h5')
# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_labels)
test_Y_one_hot = to_categorical(test_labels)
# Split the data
train_X,valid_X,train_label,valid_label =
      train_test_split(train_data,train_Y_one_hot,test_size=0.2,ra
      ndom_state=13)
# Shape check of the datasets
train_X.shape,valid_X.shape,train_label.shape,valid_label.shape
def fc(enco):
    flat = Flatten()(enco)
    den = Dense(128, activation='relu')(flat)
    out = Dense(num_classes, activation='softmax')(den)
    return out
# Build the model
encode = encoder(input_img)
full_model = Model(input_img,fc(encode))
# loading the weights from the encoder part of the autoencoder
trained before
for l1,l2 in
zip(full_model.layers[:19],autoencoder.layers[0:19]):
    l1.set_weights(l2.get_weights())
# Check the weights of the firsts layers of nets 1 and 2 to check
that are the same
autoencoder.get_weights()[0][1]
full_model.get_weights()[0][1]
# Freezing the encoder layers to not train them in this second
neural net
for layer in full_model.layers[0:19]:
    layer.trainable = False
# Compile the model
full_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
# Visualize the Classifier
full_model.summary()
# Training the net 2
```

```
classify_train =
      full_model.fit(train_X, train_label,
      batch_size=64,epochs=20,verbose=1,validation_data=(valid_X,
      valid_label))
# Plot loss and accuracy
accuracy = classify_train.history['accuracy']
val_accuracy = classify_train.history['val_accuracy']
loss = classify_train.history['loss']
val_loss = classify_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
# Save the model
full_model.save_weights('autoencoder_classification.h5')
# Evaluation
test_eval = full_model.evaluate(test_data, test_Y_one_hot,
verbose=0)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
# Making predictions
predicted_classes = full_model.predict(test_data)
# From [0, 0, 0, 1, 0, 0, 0, 0, 0] to 3
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
# Check the number of predicted and test labels
predicted_classes.shape, test_labels.shape
# Checking the correct labels
correct = np.where(predicted_classes==test_labels)[0]
print ("Found %d correct labels" % len(correct))
for i, correct in enumerate(correct[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_data[correct].reshape(28,28), cmap='gray',
interpolation='none')
    plt.title("Predicted {}, Class
{}".format(predicted_classes[correct], test_labels[correct]))
    plt.tight_layout()
# Checking the incorrect labels
incorrect = np.where(predicted_classes!=test_labels)[0]
print "Found %d incorrect labels" % len(incorrect)
for i, incorrect in enumerate(incorrect[:9]):
    plt.subplot(3,3,i+1)
```

```
    plt.imshow(test_data[incorrect].reshape(28,28), cmap='gray',
interpolation='none')
    plt.title("Predicted {}, Class
{}".format(predicted_classes[incorrect], test_labels[incorrect]))
    plt.tight_layout()
# Classification report
from sklearn.metrics import classification_report
target_names = ["Class {}".format(i) for i in range(num_classes)]
print(classification_report(test_labels, predicted_classes,
target_names=target_names))
```