Detección y clasificación de células anormales de sangre periférica usando técnicas de N-Shot Learning.

José Lucas Guerrero
Máster Universitario de Bioinformática y Bioestadística
Área de Machine Learning

Director:
Edwin Santiago Alférez Baquero

02/01/2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Detección y clasificación de células anormales de sangre periférica usando técnicas de N-Shot Learning.*
Nombre del autor:	*José Lucas Guerrero*
Nombre del consultor/a:	*Edwin Santiago Alférez Baquero*
Nombre del PRA:	Fernan Prados
Fecha de entrega (mm/aaaa):	01/2021
Titulación::	*Máster universitario en Bioinformática y Bioestadística*
Área del Trabajo Final:	*Machine Learning*
Idioma del trabajo:	*Español*
Palabras clave	*Deep Learning, N-shot Learning, Células linfoides, Linfomas.*

Resumen del Trabajo :

Mediante el uso de tecnologías Machine Learning para el procesado digital de imágenes es posible desarrollar aplicaciones que clasifiquen imágenes en diferentes categorías.

En este trabajo fin de máster se han implementado modelos de Deep Learning del tipo n-shot learning para la clasificación de imágenes de sangre periférica. Estas técnicas son adecuadas cuando no se dispone de suficiente muestra para el entrenamiento. En concreto, se ha desarrollado una red siamesa que comprueba si dos imágenes diferentes corresponden al mismo tipo de célula y una red five-shot que clasifica leucocitos en sanos y afectados por linfoma de Burkkit.

Se desarrollaron dos modelos en PyTorch y Fast.ai obteniendo tasas de acierto superiores al 75% y 85% respectivamente.

La conclusión del trabajo es que las técnicas few-shot son un buen enfoque cuando no se dispone de suficiente muestra para aplicar técnicas convencionales.

Abstract (in English, 250 words or less):

Using Machine Learning technologies for digital image processing, it is possible to develop applications able to classify images between different categories.

In this master's thesis, Deep Learning models of the n-shot learning type have been implemented for the classification of peripheral blood images. These techniques are suitable when not enough sample is available for the training process. A Siamese Network has been developed for checking if two different images correspond to the same cell type. A five-shot network has been developed to classify leukocytes between healthy and affected by Burkkit's Lymphoma.

Both models were developed in PyTorch and Fast.ai obtaining success rates higher than 75% and 85% respectively.

The conclusion of the work is that few shot techniques are a good approach when not enough sample is available to apply conventional techniques.

Agradecimientos

-Al hospital Clínic de Barcelona, en particular a la doctora Anna Merino que nos ha proporcionado las imágenes de pacientes reales gracias a las cuales ha sido posible realizar este estudio.

-A la Universidad, en especial a mi tutor Edwin que además de confiar en mí para la realización de un TFM tan ambicioso, me ha proporcionado toda la ayuda y atención que he necesitado durante estos meses, que por culpa de la COVID, han sido especialmente duros para todos.

Índice

Listado de figuras

Listado de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

La temática escogida para el proyecto es la aplicación de técnicas avanzadas de Deep Learning para la clasificación de células utilizando imágenes de sangre periférica obtenidas mediante un microscopio óptico. En los últimos años la inteligencia artificial y en particular el Aprendizaje Profundo(Deep Learning) ha sufrido un crecimiento exponencial, cada vez hay metodologías más precisas para resolver una gran variedad de problemas al mismo tiempo que la potencia de cálculo es cada día más asequible.
Llegados a este punto, en que aparecen nuevas técnicas y las existentes mejoran cada pocos días, el cuello de botella pasa a ser los datos disponibles. De forma general para entrenar un modelo necesitamos un conjunto de datos etiquetados. Y aunque existen excepciones, cuando se dispone de conjuntos de entrenamiento lo suficientemente grandes y con la calidad adecuada, es posible construir un modelo que funcione correctamente.

El principal problema que motiva este TFM es la escasez de imágenes diagnósticas de enfermedades concretas. Para solventar dicho problema existen dos enfoques:

- Obtener la máxima información a partir de la información que disponemos.
- Generar casos adicionales mediante una red generacional.

En este TFM únicamente se va a trabajar sobre el primer punto. En particular nos centraremos en imágenes de células de sangre periférica para el diagnóstico de leucemias y linfomas raros. Se utilizarán redes experimentales de la familia n-shot learning para obtener un modelo que reconozca diferentes tipos de células y, además, sea capaz de clasificar entre sanas / enfermas con pocos casos de entrenamiento. Aunque el modelo se entrenará con imágenes de sangre periférica, en medida de lo posible la implementación será genérica y se creará un repositorio público para que cualquier persona interesada pueda reutilizar, continuar o modificar el trabajo realizado.

1.2 Objetivos del Trabajo

1.2.1. Objetivo General

Desarrollar modelos de clasificación de diferentes tipos de células linfoides anormales de sangre periférica mediante técnicas n-shot learning.

1.2.2. Objetivo Específicos

- Reforzar y ampliar los conocimientos de Deep Learning. En particular se trabajará con modelos de la familia n-shot learning como son las redes siamesas y self-supervised 5-Shot Learning .
- Desarrollar un pipeline automatizado, de modo que mediante un conjunto de imágenes en bruto, se realicen los tratamientos necesarios para que puedan ser utilizadas por cada una de las arquitecturas escogidas.
- Crear un repositorio donde subir los códigos creados y su correspondiente documentación de modo que estén disponibles.
- Evaluar el desempeño de las soluciones construidas en diferentes problemas, para la clasificación de células anormales de sangre periférica.

1.3 Enfoque y método seguido

El objetivo final de este proyecto es desarrollar una herramienta para clasificar células linfoides anormales a partir de imágenes de frotis de sangre periférica proporcionadas por el Hospital Clínic de Barcelona. El elemento diferenciador es el conjunto de técnicas utilizadas, en lugar de adaptar una red convolucional pre-entrenada como Imagenet, se utilizan técnicas de la familia few-shot learning, adecuadas cuando el conjunto de datos disponible para entrenamiento es muy reducido.

En el proceso se pueden distinguir tres pasos.

1. Se preparará un pipeline genérico que prepare los datos para cada uno de los modelos, es decir un proceso de que de forma automatizada adapte las imágenes para ser utilizadas en cada uno de los modelos implementados.

2. Se implementarán los clasificadores, las técnicas seleccionadas son las redes siamesas, y redes self supervised 5 shot. Si bien las redes desarrolladas en este TFM tienen una finalidad específica que es clasificar células, en medida de lo posible será genérico para que sea posible adaptarlo a varios problemas.

3. Una vez implementadas las redes, se procederá a su entrenamiento y ajuste, siendo el último paso la evaluación de las redes, tanto desde un punto puramente analítico como aplicado.

A continuación, se muestra un esquema del proceso de obtención del clasificador y su posterior uso para la clasificación de nuevos casos:

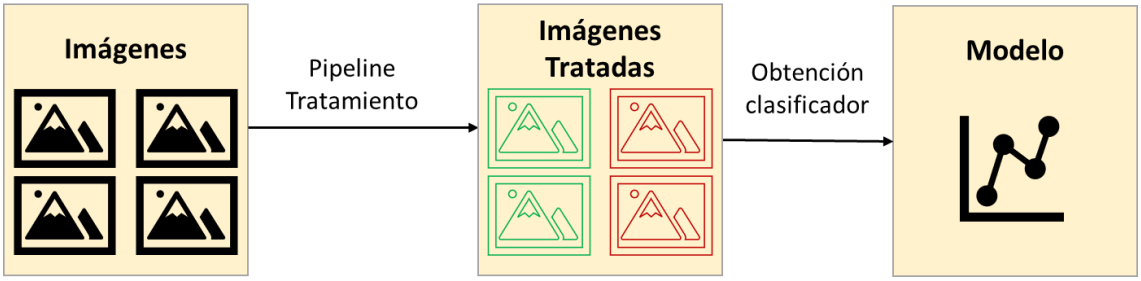

Figura 1- Esquema entrenamiento red neuronal.

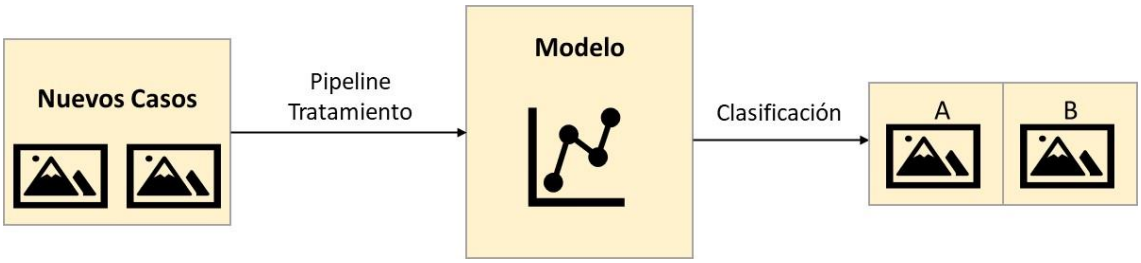

Figura 2- Esquema predicción red neuronal.

Respecto a los costes, el proyecto fin de máster será realizado en el domicilio del autor, se dispone de un equipo personal de gama media-alta que será suficiente para realización del proyecto. (Procesador I7, 64gb de RAM y gráfica Nvidia gtx1080TI). Aunque el equipo es de uso personal dispone de suficiente potencia para realizar los análisis requeridos. Todo el software utilizado será libre, por lo que no se contempla costes de licencias.

Una aproximación al consumo serían 250w/h cuando el equipo esté realizando cálculos al 100%, teniendo en cuenta el precio de la energía en España y estimando 100 horas de cálculo intensivo, el coste será inferior a 20€.

Adicionalmente y aunque no era estrictamente necesario, para obtener la posibilidad de trabajar con distintas redes en paralelo, se ha decidido contratar la versión PRO del servicio Colab de Google, dicho servicio permite trabajar con cuadernos Python en la nube, en entornos aislados y haciendo uso de GPUs de gama profesional de NVIDIA. El coste del servicio son 10$ al mes durante los 5 meses de duración del proyecto.

1.4 Planificación del Trabajo

1.4.1. Tareas

Tareas para la fase 1:

I. Preparar Equipo para Deep Learning, se configurará un equipo Linux con todas las librerías requeridas para la ejecución de PyTorch, Fast.ai sobre GPU. Además, se utilizarán la herramienta Lightly. (10 horas).

II. Formación Pytorch, se realizarán cursos online sobre PyTorch para adquirir las competencias de programación necesarias. (30 horas) y sobre Fast.ai (20 horas).

III. Formación Few-Shot Learning, se realiza la revisión y estudio de la bibliografía relacionada con los modelos few-shot. (15 horas).

IV. Implementación de la red siamesa en Pytorch (20 horas).

V. Implementación de las redes few-shot en Fast.ai (20 horas).

VI. Análisis del estado del arte de redes neuronales , Deep Learning y modelos few-shot learning. (10 horas).

Tareas para la fase 2:

I. Creación de un Pipeline de tratamiento automatizado de imágenes (15 horas).

II. Entrenamiento y validación de los modelos few-shot (55 horas).

III. Creación de un repositorio público (10 horas).

IV. Análisis de resultados y conclusiones (10 horas).

V. Redacción de memoria (15 horas).

1.4.2. Calendario

El calendario con las diferentes etapas del proyecto se ha establecido de forma semanal, dedicando entre 10 y 20 horas semanales a la elaboración del proyecto.

Planificación global:

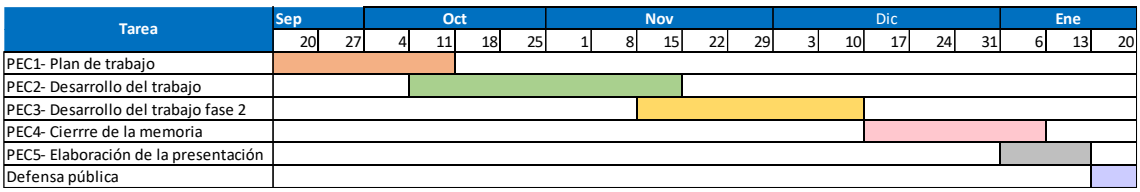

Figura 3- Planificación global.

Programación del tiempo para la primera fase del desarrollo (PEC2):

PEC2- Desarrollo del trabajo	Octubre				Noviembre		
	8	15	22	29	1	8	15
Preparar Equipo							
Formación Pythorch							
Formación Pythorch							
Implementar red siamesa							
Implementar red 5-shot							
Escribir parte teórica redes neuronales							
Escribir parte teórica few shot learning							

Figura 4 – Planificación PEC2.

Programación del tiempo para la segunda fase del desarrollo (PEC3):

PEC3- Desarrollo del trabajo fase1	Nov				Dic	
	8	15	22	29	3	10
Creación Pipeline Tratamiento						
Entrenamiento y validación modelos						
Repositorio público						
Conclusiones						
Memoria						

Figura 4- Planificación PEC3.

1.4.3. Hitos

- Configuración del equipo de Trabajo para Deep Learning.
- Creación del Pipeline de tratamiento de datos.
- Implementación y entrenamiento clasificador red siamesa.
- Implementación y entrenamiento de la red 5 Shot-Learning.
- Utilizar los clasificadores con nuevos ejemplos y medir la precisión.
- Redactar la memoria del TFM.
- Elaboración de la defensa del TFM.

1.4.4. Análisis de Riesgos

El proyecto como cualquier otra actividad profesional, es sensible a una serie de riesgos que puede afectar negativamente al desarrollo y/o consecución del mismo.

1) Problemas técnicos: problemas con el equipo informático, al utilizar un único equipo el riesgo operacional es elevado.
    i) -Se dispone de sistemas de protección contra cortes y picos en el suministro eléctrico, además se realizarán Backups periódicos tanto de forma local como online para evitar la pérdida de información.

ii) -Existe un plan de contingencia en caso de no disponer del equipo principal, se podrá sustituir con recursos de computación Cloud.

2) Desajuste de objetivos: si existiera desajuste, se replanificará teniendo siempre en cuenta la consecución de todos los hitos en la fecha final del proyecto.
3) Enfermedad o accidente: si existe una enfermedad leve-moderada se intentarán compensar las horas.
4) Crisis globales: situaciones como guerras, crisis económicas o pandemias que puedan causar cambios no previsibles en la rutina de las partes relacionadas con el proyecto.

1.5 Breve sumario de productos obtenidos

Los productos obtenidos a la finalización del trabajó serán:

- Plan de trabajo
- Informe de resultados
- Repositorio con el Software Python desarrollado para la clasificación de células anormales de sangre periférica mediante técnicas few-shot y su documentación.
- Videopresentación del proyecto.

1.6 Breve descripción de los otros capítulos de la memoria

El trabajo constará de 6 capítulos:

- 2-Contexto y antecedentes: Breve introducción al aprendizaje al aprendizaje profundo y Linfoma de Burkitt.
- 3-Red One shot Learning: Diseño e implementación de la red siamesa.
- 4- Red Few Shot Learning y Finetuning: Diseño e implementación de la red 5-Shot Learning.
- 5- Reproducibilidad y creación de repositorio: Se proporcionarán las instrucciones para adaptar y ejecutar el código con cualquier otro conjunto de imágenes. Además, se creará un pequeño repositorio público.
- 6- Resultados y conclusiones: Se realizará una valoración global del trabajo y los resultados obtenidos con los datos de ejemplo. Además, se propondrán posibles líneas de trabajo que continúen este proyecto.
- 7- Glosario: Definición de términos y acrónimos utilizados en el TFM.
- 8-Bibliografía: Bibliografía utilizada en el TFM.

2. Contexto y antecedentes

2.1 Deep Learning

Se utiliza el término “Deep Learning” para referirse a un conjunto de algoritmos de aprendizaje automático que intenta es emular los enfoques de aprendizaje de los seres humanos, utilizando para ello una serie de características específicas: transformaciones no lineales, diferentes niveles de abstracción.

Antes de continuar hablando de Deep Learning es importante definir varios conceptos:

- Inteligencia Artificial: se utiliza el término para referirse a una máquina imita las funciones cognitivas de los humanos. Aunque existen múltiples definiciones y enfoques (Singh, 2014), nos centraremos en los más actuales, que definen inteligencia artificial cuando la máquina dispone de la capacidad de aprender, resolver problemas y/o tomar decisiones (Auffarth, 2020).

- Aprendizaje Automático: Rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permiten que las máquinas aprendan y mejoren con la experiencia. Ya no se trata de un programa estático que se ejecuta para dar respuesta a unos inputs con unas reglas, si no que los programas se modifican dinámicamente. (Stevens et al., 2020) Un ejemplo sencillo es un filtro anti-spam de email, se entrena con un conjunto de mensajes iniciales, pero se adapta a la bandeja de correo de cada usuario y sus interacciones. Dentro del aprendizaje automático podemos distinguir dos variantes, supervisado y no supervisado. En el aprendizaje supervisado se entrena el modelo proporcionando las características y etiquetas. Sin embargo en el no supervisado únicamente se le facilitan las características y el algoritmo cataloga por similitud (Zhao Y Liu, 2007).

- Aprendizaje de características: dentro del aprendizaje automático, se define aprendizaje de características como un conjunto de técnicas que permite que un sistema descubra automáticamente las representaciones necesarias para la detección o clasificación de características a partir de datos sin procesar. (Stevens et al., 2020)

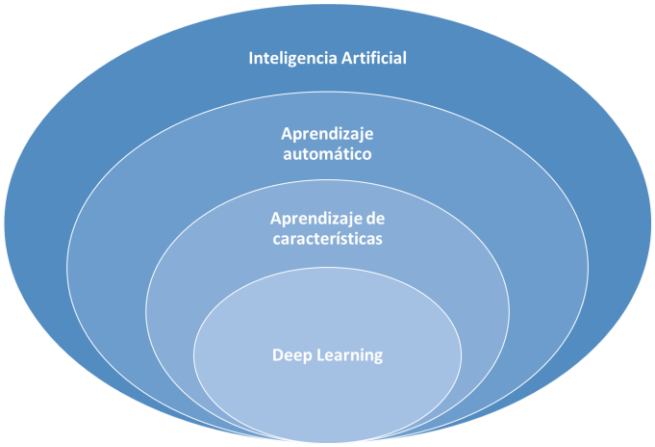

Figura 5- Esquema IA, Machine y Deep Learning

Por tanto, la principal peculiaridad del Deep Learning dentro de las metodologías de Machine Learning es que las características no son definidas por el usuario, sino que se obtienen de forma automática. El Deep Learning tiene una gran potencia debido a que es capaz encontrar características que para el usuario sería imposible gracias a transformaciones complejas.

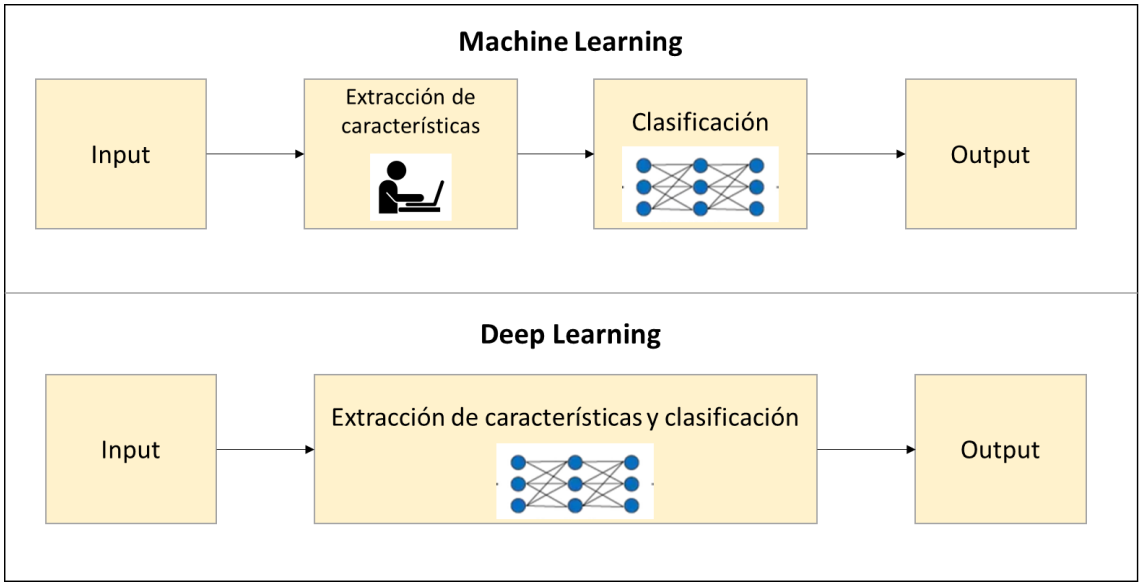

Figura 6 -Diferencias Machine y Deep Learning.

En un gran número de ocasiones las características obtenidas por los modelos de Deep Learning son difícilmente interpretables para el ser humano y son tratados como modelos de caja negra. Aunque en los últimos años la comunidad científica está poniendo un alto esfuerzo en avanzar en técnicas y metodologías para la interpretabilidad de estos modelos (Montavon et al., 2018).

En las definiciones anteriores hemos hablado de modelos, redes, máquinas que buscan imitar las funciones cognitivas del ser humano, pero no hemos detallado como se hace. En Machine Learning existen muchos enfoques diferentes tanto supervisados y no supervisados como pueden ser las regresiones, árboles de

decisión y máquinas de vectores de soporte entre muchos otros. En cambio, cuando hablamos de Deep Learning nos referimos a redes neuronales.

- Redes Neuronales: Las redes neuronales artificiales son un sistema computacional inspirado en el funcionamiento del cerebro humano. Existe un conjunto de neuronas artificiales conectadas entre sí para transmitir señales. Una señal de entrada atraviesa la red, y produce una salida.

Neurona artificial:

El perceptrón creado por Frank Rosenblatt es la forma más básica de una red neuronal, formada por una única neurona. (Rosenblatt, 1957). Está formado por 5 partes (Gerón, 2017):

1. Entradas que reciben los datos. En una neurona biológica corresponde a las dendritas.
2. Pesos sinápticos. Como en las neuronas biológicas se establecen sinapsis entre dendritas de una neurona y el axón de otra. En las neuronas artificiales se les asigna un peso, que se modifica durante el entrenamiento de la red neuronal.
3. Una regla de propagación o Bias. A partir de las entradas y los pesos sinápticos se realiza una agregación para obtener el potencial postsináptico (La más habitual es la suma ponderada, pero podrían utilizarse otras)
4. Función de activación: El valor obtenido en la regla de propagación se filtra a través de una función. Las funciones de activación más utilizadas son ReLU, Sigmoide y tangente hiperbólica. Pero pueden emplearse otras.
5. Salida: Valor de salida tras aplicar la función de activación.

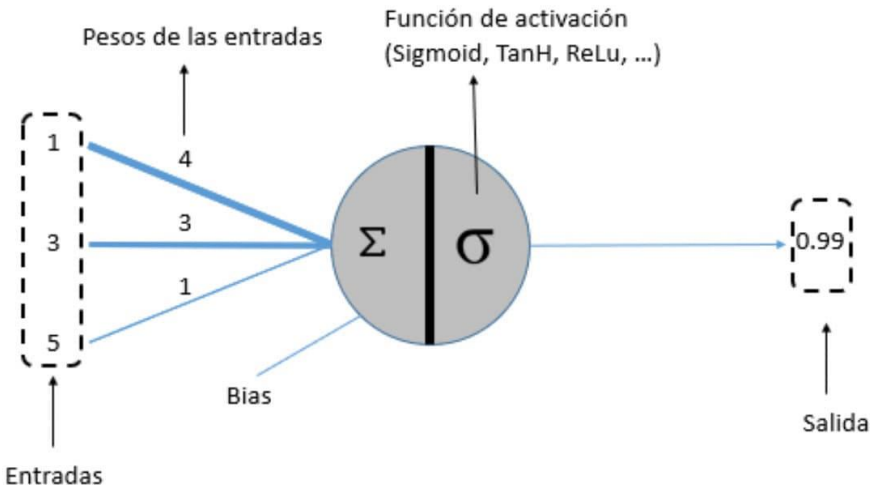

Figura 7- Esquema perceptrón.
Fuente: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i

Al igual que en el mundo biológico una red de una única neurona no tiene gran utilidad, en cambio cuando se combinan en redes formadas por varias neuronas agrupadas en diferentes niveles se puede entrenar para realizar multitud de

tareas. Los niveles de agrupación se organizan en capas, de modo que la primera capa utiliza los datos de entrada, la segunda capa utiliza como entrada los datos de la primera capa y así sucesivamente hasta la última capa (Noriega, 2005). Cuando trabajamos con un perceptrón multicapa aparecen dudas relacionadas con los algunos conceptos clave (Gerón, 2017):

- ¿Cómo se combinan las entradas? → Mediante los pesos
- ¿Cómo se determina la salida? Mediante una función de activación
- ¿Cómo se establecen estos parámetros? Mediante el aprendizaje
- ¿Qué otros parámetros intervienen? El sesgo

El perceptrón multicapa se entiende como la evolución del perceptrón simple donde aparecen diferentes capas intermedias (llamadas ocultas) entre la entrada y la salida (Noriega, 2005). Debido a que es una materia complicada y especialmente extensa, únicamente se proporcionan datos clave para comprender el proceso de construcción y entrenamiento de las redes multicapa (Gerón, 2017):

- El número de capas ocultas aumenta la complejidad de la red
- El proceso de aprendizaje puede verse como la búsqueda de reducir un valor llamado Coste. Este valor representa lo lejos que está nuestra red de producir un resultado perfecto sobre un conjunto de entrenamiento.
- En lugar de computar el error absoluto, se buscan funciones de coste que varíen ligeramente ante pequeños cambios en los parámetros (pesos o sesgo).
- El aprendizaje es un proceso iterativo donde se introduce un parámetro: factor o ratio de aprendizaje. Consta de etapas forward y backward:
    - Forward: a partir de unos valores de entrada, se realizan los cálculos intermedios, se almacenan para cálculos posteriores, y se obtiene un resultado.
- Backward: se aplica la regla de la cadena de forma que en cada neurona, se actualicen sus parámetros asociados(pesos) en base a:
    - 1.- El ratio de aprendizaje
    - 2.- El gradiente computado para esta neurona
- El ratio de aprendizaje suele ser una constante
- El gradiente es la derivada parcial del error total (descomponiendo la fórmula) con respecto al peso de la neurona a actualizar

El concepto de Deep Learning surge de utilizar un gran número de capas ocultas intermedias. A continuación, se muestra el esquema de una red neuronal de 4 capas.

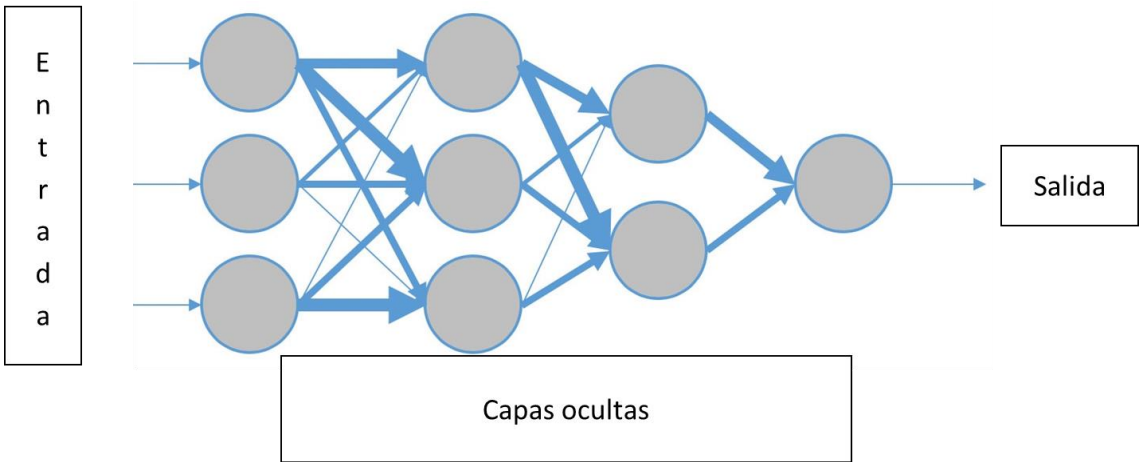

Figura 8- Esquema red neuronal de 4 capas.

Redes Neuronales Convolucionales

Una Red Neuronal Convolucional (CNN) es un tipo de Red Neuronal Artificial de aprendizaje supervisado que imita la visión del ser humano, su objetivo es identificar distintas características en imágenes para conseguir una generalización (Albawi et al, 2017). Como hemos mencionado en los apartados anteriores una red neuronal generalmente se entrena con casos de ejemplo, influyendo tanto la cantidad de ejemplos disponibles como la calidad de ellos en el proceso de aprendizaje de la red. Si bien las redes neuronales convolucionales son un tema muy extenso y bastante complejo de explicar, a continuación, se darán unas breves nociones para que el lector sea capaz de comprender el resto de capítulos (Albawi et al, 2017).

Preprocesado de las imágenes: De forma general cuando se trabaja con imágenes, es necesario tratarlas y convertidas al formato de entrada que espera nuestra red. Lo más habitual es suministrar una matriz de píxeles normalizados. Los valores de los píxeles en B/N se representan con un número entre 0 y 255 a los que se aplica normaliza para que su valor esté entre 0 y 1. Cuando la imagen es una representación a color, no basta con una única matriz sino que la imagen se compone de una combinación de 3 matrices, una para cada canal de color RGB (Rojo, Verde y Azul) .

De forma general la red toma como entrada los pixeles de una imagen, por lo que, si tenemos una imagen de 32x32 píxeles de una imagen a color (3 canales) dispondríamos de 32x32x3 conexiones de pesos. Conectando una neurona en la segunda capa tendríamos 32x32x3x2 parámetros en nuestra red(6.144). Una red de dos neuronas no es útil para prácticamente ningún fin, y si continuamos añadiendo capas el número de parámetros es demasiado grande para ser procesado. En este contexto aparecen el concepto de convolución .

Convolución: Proceso que consiste en seleccionar pequeños grupos de píxeles cercanos y multiplicarlos por una pequeña matriz que se denomina kernel. El resultado de dicho producto escalar será el valor de cada una de las posiciones

de la matriz de salida. Esta matriz corresponderá a una nueva capa oculta de la red.

En la figura 9 se muestra el proceso de convolución:

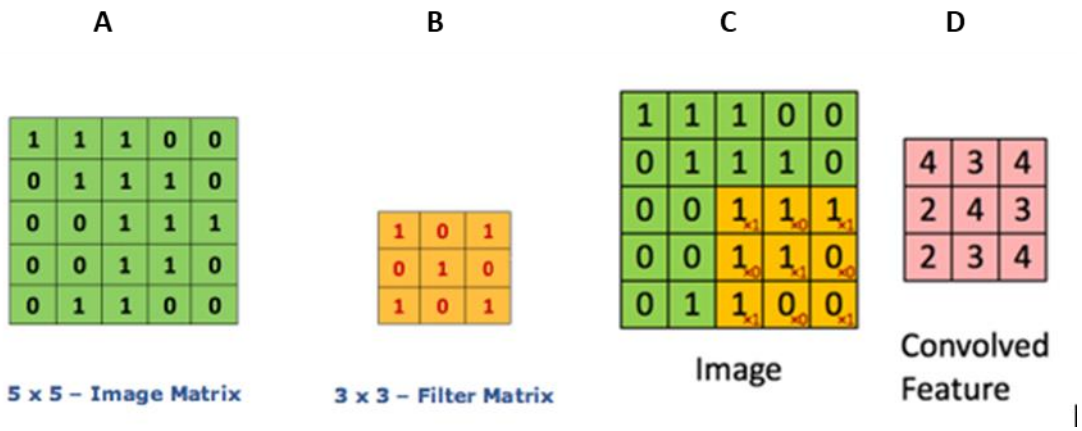

Figura 9- Ejemplo de convolución.
Fuente: https://medium.com/@parikhkadam/article-1-understanding-the-convolution-function-and-cnn-21dca53e2c27

- La figura 9A es la representación de una imagen de 5x5 bits.
- La figura 9B es el filtro kernel, de 3x3.
- Para aplicar la convolución se realiza el producto escalar de los subconjuntos de 3x3 de la imagen original por el kernel y se suman los resultados.
- En la figura 9C se observa cómo se realiza el proceso para el último subconjunto, cuyo resultado es 4.
- La figura 9D es la matriz de salida que contiene el total de los 9 pasos de convolución realizados.

Una vez realizada la convolución es frecuente aplicar una función de activación, la más utilizada es RELU, que devuelve el máximo entre 0 y el valor de entrada eliminando valores negativos (Stevens et al., 2020).

Otro paso necesario es reducir el tamaño de la salida para evitar que el número de neuronas necesarias crezca exponencialmente, se realiza un proceso de subsampling de modo que prevalezcan las características más importantes detectadas por los filtros. Hay diversos tipos, pero uno de los más utilizados es el max-pooling, que permite reducir la dimensionalidad a la vez que hace que prevalezcan las características más importantes que detecto cada filtro (Stevens et al., 2020). A continuación, se muestra un ejemplo de max-pooling, 2x2, en el que se reduce una matriz de 4x4 a una de 2x2 conservando el valor máximo de cada una de las 4 porciones coloreadas.

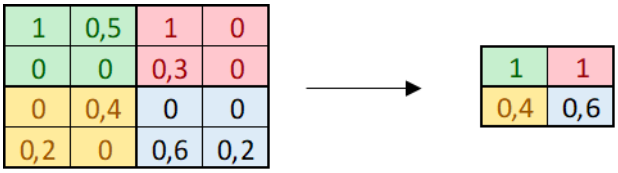

Figura 10- Ejemplo Max-Pooling.

Con los pasos anteriores Convolución-Activación- Max Pooling tendríamos una red convolucional mínima. Sin embargo, la utilidad de la una red de estas características sería prácticamente nula puesto que hemos creado una red con un único ciclo de convolución y aplicando un único Kernel cuando lo habitual es utilizar varios ciclos de convolución y grandes conjuntos de Kernels en cada convolución. A continuación, se muestra el esquema de red de una de las redes convolucionales más utilizadas VGG16 (Simonyan y Zisserman 2014):

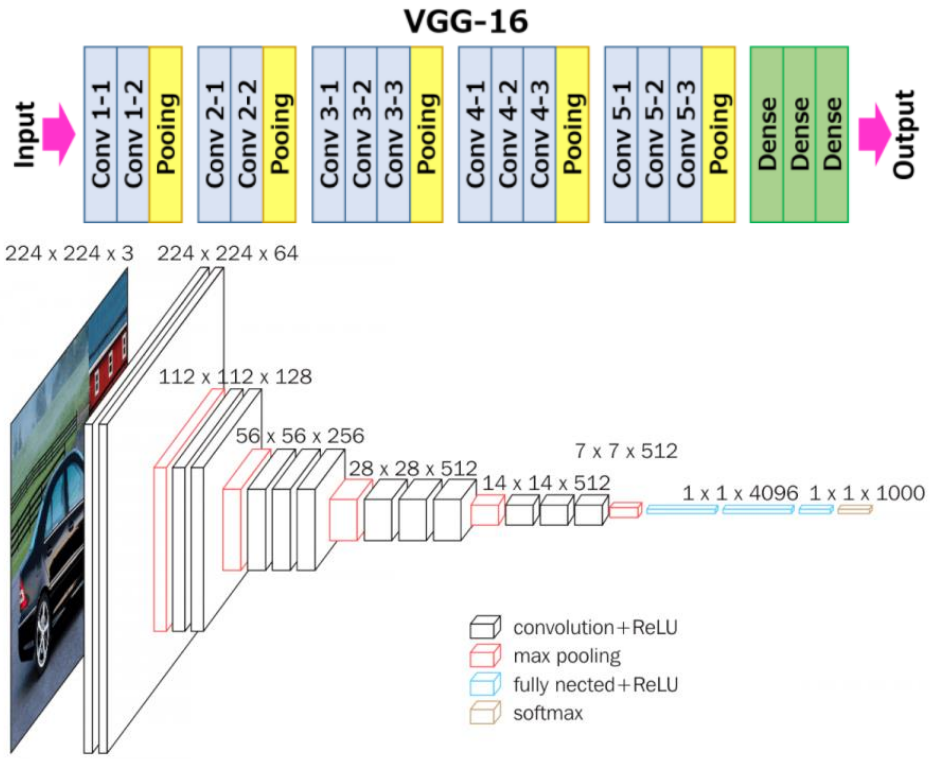

Figura 11- Estructura VGG16.
Fuente: https://neurohive.io/en/popular-networks/vgg16/

Podemos observar que existen 13 ciclos de convolución y 4 max-pooling.

Ejemplo de red neuronal convolucional

Un ejemplo clásico que se estudia cuando se comienza con Deep Learning y nos puede ayudar a entender los conceptos es una red neuronal convolucional que diferencia entre de perros y gatos. La red se entrena con un gran número de imágenes de perros y gatos con su etiqueta correspondiente, perro o gato. Se construye una red y se entrena. La salida del modelo es la probabilidad de perro y la probabilidad de gato, se selecciona la mayor y esa es la predicción de nuestro modelo. Una vez que ha concluido el proceso de entrenamiento y se tiene una red lo suficientemente precisa, se realizan pruebas con imágenes nuevas y probar que el clasificador hace una generalización correcta y no funciona únicamente con los datos de entrenamiento. En la competición de

kaggle puede encontrarse mucha información relacionada con este problema y su implementación (Kaggle, 2014).

Few Shot Learning

En los puntos anteriores hemos realizado una breve introducción al aprendizaje automático y Deep Learning. Estas tecnologías han sufrido una revolución en los últimos años, pasando de ser una tecnología experimental a tener muchísimas aplicaciones reales como reconocimiento facial y de objetos, conducción autónoma , diagnóstico por imagen, bots conversacionales etc.

Uno de los enfoques más utilizados de Deep Learning en la actualidad está basado en redes neuronales convolucionales para clasificar o identificar elementos en una imagen, requiere de grandes conjuntos de entrenamiento necesarios muchos ejemplos de cada clase para identificar las características de cada clase y obtener un modelo que sea preciso. Sin embargo, existen muchos casos en el mundo real en los que no es posible disponer de un gran conjunto de datos para el entrenamiento del modelo. Un ejemplo son imágenes de células linfoides anormales afectadas por enfermedades raras como linfoma de Burkkit.

El ser humano tiene una gran capacidad para distinguir y clasificar objetos con pocos o un único ejemplo, aun cuando son objetos que nunca ha visto. Un niño es capaz de aprender a identificar un objeto con un solo ejemplo, sin embargo, una red neuronal "clásica" necesita muchos ejemplos para extraer las características para hacer una generalización, y si no dispone de un número no suelen funcionar correctamente (Jadon y Garg, 2020).

En este contexto aparecen las técnicas few-shot learning, estas técnicas intentan obtener un modelo con una cantidad muy pequeña de datos de entrenamiento, al contrario de las prácticas habituales que usan grandes cantidades de datos.

Existen dos enfoques few shot learning (Jadon y Garg, 2020):

1. Enfoque a nivel de datos: en este enfoque el número de datos es insuficiente para obtener un modelo preciso y por tanto deben añadirse más datos. De modo que se añaden datos, bien sea realizando transformaciones en los datos existentes, incluyendo datos de fuentes externas, o utilizando Redes Generativas Antagónicas (GAN) para crear nuevos ejemplos.

2. Enfoque a nivel de modelo: este enfoque se utilizan algoritmos y redes especialmente diseñadas para extraer las características y ser capaces de obtener generalizaciones a partir de un pequeño número de ejemplos.

En este TFM se va a realizar la implementación de dos redes con enfoque a nivel de modelo, en primer lugar se utilizará una red siamesa para clasificar entre diferentes tipos de células. Y en segundo lugar una red 5 shot learning para diferenciar células sanas y afectadas por Linfoma de Burkitt.

Deep Learning se asocia a complejos esquemas con numerosas conexiones difíciles de implementar. Pero existen recursos que permiten su aplicación sin

necesidad de abordar todas las tareas, son los denomidados Frameworks, actualmente tres gozan de gran popularidad:

- Caffe2 by Facebook
- Pytorch by Facebook
- Tensorflow by Google

Aunque los tres son multiplataforma y tienen sus librerías en Python (además de otros lenguajes), realizaremos nuestros desarrollos sobre PyTorch que es el entorno más utilizado en investigación. Caffe2 está más orientado a escalabilidad y uso en producción y Tensorflow aunque es uno de los más extendidos, tiene una curva de aprendizaje es bastante más elevada que PyTorch.

En la primera red trabajaremos sobre PyTorch directamente, en la segunda trabajaremos sobre FastAI. FastAI es una librería de código abierto implementado sobre PyTorch que propoporciona una API de alto nivel que simplifica la creación y entrenamiento de redes neuronales.

2.2 Linfoma de Burkitt

El linfoma de Burkitt es una enfermedad rara, en concreto un cáncer muy agresivo, que puede pasar de un estadio a otro muy rápido, pero, que también tiene una rápida reacción al tratamiento con quimioterapia si se detecta a tiempo.

Según la American Cancer Society (2020) el linfoma de Burkitt es un tipo de linfoma no Hodgkin de células B dinámico (de crecimiento rápido). Normalmente afecta a niños y adultos jóvenes suponiendo entre un 1% y un 2% del total de los linfomas presentes en adultos. Siendo más frecuente en varones que mujeres. Existen tres variedades de linfoma de Burkitt diferentes:

- Africana o endémica: Afecta a la mandíbula y otros huesos faciales. Relacionado con la infección del virus Epstein-Bar.
- No endémico o esporádico: Más frecuente en EEUU, comienza afectando al abdomen , ovarios testículos pudiendo propagarse al cerebro y líquido cefalorraquídeo.
- Asociado a inmunodeficiencia: Presente en algunas afectas de VIH, SIDA o que han tenido sido sometidas a una operación de trasplante de órganos.

Los linfomas son cánceres de un tipo específico de glóbulos blancos (leucocitos) denominados linfocitos, que ayudan a combatir las infecciones. En concreto esta enfermedad afecta a los linfocitos B que son los encargados de producir anticuerpos, que son esenciales para combatir algunas infecciones (American Cancer Society, 2020).

Al ser una enfermedad tan poco frecuente apenas se dispone de casos documentados. De modo que este trabajo no sería posible sin la colaboración

del Hospital Clínic de Barcelona, que además de imágenes de calidad de célculas sanas nos ha facilitado imágenes de leucocitos afectados por el Linfoma de Burkitt. La técnica utilizada para la obtención de las imágenes ha sido el frotis de sangre periférica.

El frotis de sangre periférica es un procedimiento llevado a cabo en laboratorio mediante un microscopio óptico en el que se observa una muestra de sangre, permitiendo examinar células sanguíneas como glóbulos blancos, glóbulos rojos o linfocitos entre otros de forma individual (Romero, 2013).

3. Red One shot Learning- Red Siamesa

Una red neuronal siamesa, es una red que consta de dos subredes idénticas que comparten estructura y además los mismos pesos. Las subredes trabajan en paralelo con dos inputs diferentes y sus salidas se unen con una función de pérdida (Koch, 2015). Es importante matizar que la red siamesa no devuelve una predicción categórica, sino que devuelve la disimilaridad entre los inputs. A continuación, se muestra un esquema de una red siamesa y la salida esperada para cada caso.

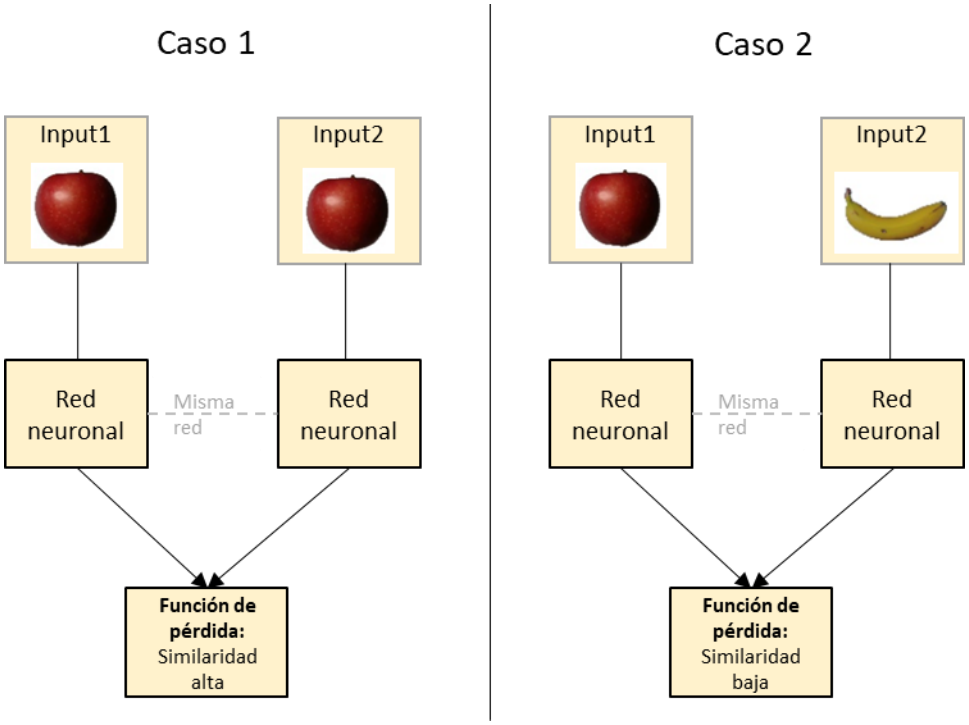

Figura 12- Esquema redes siamesas.

Aunque se pueden utilizar múltiples funciones de pérdida, en este caso se ha decidido utilizar la función de pérdida contrastiva explicada en el apartado 3.4.

3.1 Requisitos y objetivos de la red siamesa implementada

Para la creación de la red siamesa disponemos de un conjunto de imágenes de células obtenidas mediante frotis de sangre periférica. Al no disponer de imágenes de células enfermas en el momento de construcción de esta red, se ha diseñado para diferenciar entre tipos de célula. Se ha implementado el pipeline de forma genérica, de modo que cuando se disponga de nuevas imágenes únicamente será necesario cambiar el fichero comprimido que se suministra al programa. Para más detalles consultar el apartado 5 Reproducibilidad y creación de repositorio.

El principal objetivo de nuestra red siamesa es diferenciar entre tipos de células, nuestro desarrollo cumplirá los siguientes objetivos:

- Preprocesado automático de imágenes, de modo que el tratamiento sea invisible para el usuario. Las imágenes serán suministradas en un fichero comprimido. Se realizarán de forma automáticas las transformaciones como redimensionado y filtros de color.

- Creación de una estructura de red siamesa en PyTorch desde cero, y que sea adecuada para obtener la disimilaridad entre dos imágenes de células de células sangre periférica proporcionadas como input.

- Entrenamiento de la red siamesa con los datos de entrenamiento. Se seleccionará un número determinado de clases para proceder al entrenamiento.

- Evaluación de la red siamesa con los datos del conjunto de prueba (test). La diferencia con los desarrollos habituales es que para el conjunto de prueba se utilizarán clases no utilizadas en el entrenamiento. De modo que podremos evaluar si nuestra red es capaz de discernir entre tipos de célula diferentes no utilizados en el entrenamiento.

3.2 Librerías y paquetes utilizados

El programa ha sido desarrollado en Python 3.6.9 con las siguientes librerías:

- Torch: Librería de código abierto para aprendizaje automático. Dispone de una gran variedad de algoritmos de Deep Learning.
- TorchVision: Librería de PyTorch especializada en visión artificial.
- Pandas: Proporciona funcionalidad de Dataframes a Python.
- PIL:Librería que proporciona funcionalidades para trabajar con imágenes.
- Numpy: Numeric Python, proporciona funciones matemáticas de alto nivel.
- Random: Añade funcionalidad para generar números aleatorios.
- OS: Librería que proporciona Python funcionalidades para interactuar con el sistema operativo.

3.3 Preprocesado de datos

Nuestro Pipeline recibe una URL con un fichero comprimido con la siguiente estructura de directorios:

* Conjunto de entrenamiento (training set)

- Plaquetas.
- Neutrófilos
- Granulocitos inmaduros
- Eritroblastos
- Eosinófilos

* Conjunto de prueba (test set)

- Basófilos
- Linfocitos
- Monocitos.

Las clases del conjunto de prueba no tienen que estar incluidas en el conjunto training, de modo que nuestra red será capaz de decir como de diferentes son dos tipos de célula sin haber sido entrenada con dichos tipos.

Las imágenes suministradas son fotografías de gran calidad de un suministradas por el Hospital Clínic de Barcelona.

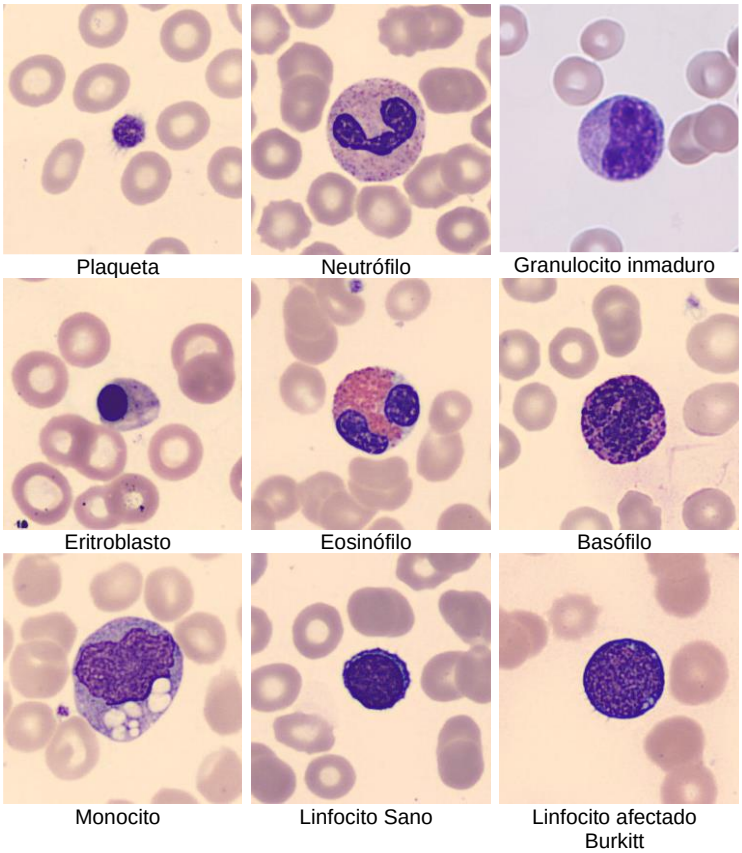

Figura 13- Muestra de imágenes sin preprocesar.
Fuente: Dataset Hospital Clínic de Barcelona.

Para poder trabajar con dichas imágenes es necesario realizar dos transformaciones, en primer lugar, redimensionarlas al tamaño de entrada de nuestra red que es 100 x 100 píxeles. En segundo lugar, se realiza la conversión al sistema de representación de color más adecuado para nuestro modelo. Se realizaron pruebas con distintas configuraciones de color :

-RGB (3 canales de color)
-CYMK (3 canales de color y 1 blanco y negro)
-ByN (1 canal de escala de grises).

Se comprobó que utilizando los canales de color la precisión apenas aumentaba e incluso disminuía, en cambio el tiempo de entrenamiento aumentaba notablemente. Por tanto, se decidió utilizar la representación en escala de grises.

En resumen, todas las imágenes del fichero comprimido de entrada se convierten a escala de grises y se redimensionan a 100 x 100 que es el tamaño de entrada definido en nuestra red siamesa debido a las limitaciones de memoria RAM del equipo doméstico utilizado.

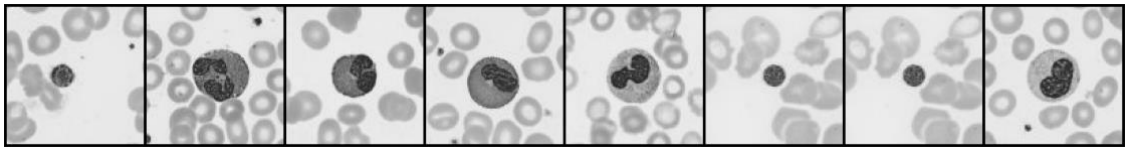

Figura 14- Muestra de imágenes sin preprocesadas.
Fuente: Dataset Hospital Clínic de Barcelona y tratamiento del autor.

3.4 Definición y estructura de la red siamesa

En los apartados anteriores hemos explicado que la red siamesa se compone de dos redes idénticas que realiza exactamente el mismo tratamiento para dos entradas diferentes.

Para este apartado hemos decidido implementar una red siamesa desde cero, utilizando como subredes una pequeña red convolucional con tres ciclos de convolución. Una vez se tienen los valores de cada una de las subredes se comparan mediante una función de pérdida que proporcionará una distancia final. La función de pérdida más utilizada en redes siamesas es la pérdida contrastiva (Contrastive Loss) definida inicialmente en el paper "*Dimensionality Reduction by Learning an Invariant Mapping"*. (Hadsell et al.,2006):

La fórmula para calcular es la siguiente:

$$L\left(W, Y, \overrightarrow{X_1}, \overrightarrow{X_2}\right) = (1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{\max(0, m - D_W)\}^2$$

Figura 15- Fórmula de la distancia contrastiva. Fuente: Hadsell et al. 2006.

Donde “Dw” es la distancia euclídea entre los dos puntos, “Y” toma el valor máximo cuando los dos inputs pertenecen a la misma clase. “m” es el valor margen definido para imponer un límite inferior a la distancia de imágenes con etiquetas diferente.

Esta forma de calcular distancias es adecuada cuando nuestros datos no están etiquetados correctamente o desconocemos las clases de los objetos a las que pertenecen. Los pares de imágenes con representaciones similares obtendrán distancias inferiores que los que sean más diferentes.

A continuación, se muestra la implementación de la función de pérdida utilizada en nuestra red:

```
Class ContrastiveLoss(torch.nn.Module):

    def __init__(self, margin=2.0):
        super(ContrastiveLoss, self).__init__()
        self.margin = margin

    def forward(self, resultA, 20os20id, label):
        20os20idean_distance = F.pairwise_distance(resultA,
20os20id, keepdim=True) # L2 Norm
        contrastive_loss = torch.mean((1-label) *
torch.pow(20os20idean_distance, 2) +
                                      (label) *
torch.pow(torch.clamp(self.margin - 20os20idean_distance,
min=0.0), 2))
        return contrastive_loss
```

Figura 16- Función ContrastiveLoss.
Fuente: https://github.com/harveyslash/Facial-Similarity-with-Siamese-Networks-in- PyTorch

En la siguiente tabla se muestra de forma esquemática los pasos que realiza cada una de las imágenes en la red. Como se puede ver son idénticas y el proceso finaliza en el paso Linear14 y Linear28 respectivamente. Donde cada una de las subredes devuelven el valor que se compara mediante Constrastive Loss.

Layer (type)	Output Shape	Param #
ZeroPad2d-1	[-1, 1, 102, 102]	0
Conv2d-2	[-1, 4, 100, 100]	40
ReLU-3	[-1, 4, 100, 100]	0
ZeroPad2d-4	[-1, 4, 102, 102]	0
Conv2d-5	[-1, 8, 100, 100]	296
ReLU-6	[-1, 8, 100, 100]	0
ZeroPad2d-7	[-1, 8, 102, 102]	0
Conv2d-8	[-1, 8, 100, 100]	584
ReLU-9	[-1, 8, 100, 100]	0
Linear-10	[-1, 500]	40,000,500
ReLU-11	[-1, 500]	0

Linear-12	[-1, 500]	250,500
ReLU-13	[-1, 500]	0
Linear-14	[-1, 5]	2,505
ZeroPad2d-15	[-1, 1, 102, 102]	0
Conv2d-16	[-1, 4, 100, 100]	40
ReLU-17	[-1, 4, 100, 100]	0
ZeroPad2d-18	[-1, 4, 102, 102]	0
Conv2d-19	[-1, 8, 100, 100]	296
ReLU-20	[-1, 8, 100, 100]	0
ZeroPad2d-21	[-1, 8, 102, 102]	0
Conv2d-22	[-1, 8, 100, 100]	584
ReLU-23	[-1, 8, 100, 100]	0
Linear-24	[-1, 500]	40,000,500
ReLU-25	[-1, 500]	0
Linear-26	[-1, 500]	250,500
ReLU-27	[-1, 500]	0
Linear-28	[-1, 5]	2,505

Tabla 1- Estructura red siamesa.

Una vez definida la estructura de nuestra red siamesa el siguiente paso es entrenarla con casos de ejemplo. Como el objetivo principal de la este TFM es diseñar y evaluar soluciones para trabajar con problemas en los que no se dispone de muestra suficiente, se ha diseñado una metodología para testear la red con células desconocidas, es decir, no utilizadas en el proceso de entrenamiento.

3.5 Entrenamiento de la red siamesa

El conjunto de datos proporcionado por el Hospital Clínic de Barcelona contiene imágenes de 8 tipos de células (Acevedo et al.,2020). Si bien las imágenes proporcionadas serían suficientes para construir una red convolucional tradicional, se han seleccionado aleatoriamente 20 imágenes de cada tipo de célula para construir nuestra red one shot.

Las cinco clases utilizadas para el entrenamiento de la red son : plaquetas, neutrófilos, granulocitos inmaduros, eritroblastos y eosinófilos. Las tres clases que han sido reservadas para evaluar la red son: basófilos, linfocitos y monocitos.

De este modo, entrenaremos la red siamesa con únicamente 100 imágenes de cinco tipos de células diferentes y después intentaremos que sea capaz de diferenciar entre 3 tipos de célula que nunca ha visto.

Tras más de 120 horas de ejecución de simulaciones y análisis de los resultados, los parámetros finales escogidos fueron:

- Learning Rate 0.0005
- Batch Size :32
- Epochs 250

Learning rate es un hiperparámetro que define cuánto varía el modelo dependiendo de los pesos obtenidos en cada iteración.
Un Epoch se corresponde con el paso del dataset de entrenamiento completo a través de la red neuronal. Cuando los conjuntos de entrenamiento son muy grandes se dividen en Batches de menor tamaño. Un Batch Size de 32 significa que en cada iteración pasarán conjuntos de 32 elementos (Howard y Gugger, 2020).

Puesto que equilibraban el tiempo de computación y la precisión del modelo.

Dicha configuración optimiza la relación entre la pérdida final y el tiempo de computación. Esta configuración consigue una pérdida del orden de 0.0001 en aproximadamente una hora de entrenamiento en nuestra GPU doméstica.

Si bien el número de Epochs puede parecer elevado, al tratarse de una red pequeña construida desde cero, no comienza a funcionar correctamente hasta Epochs alrededor de 100 Epochs. A continuación, se muestra cómo va reduciéndose la pérdida tras cada Epoch.

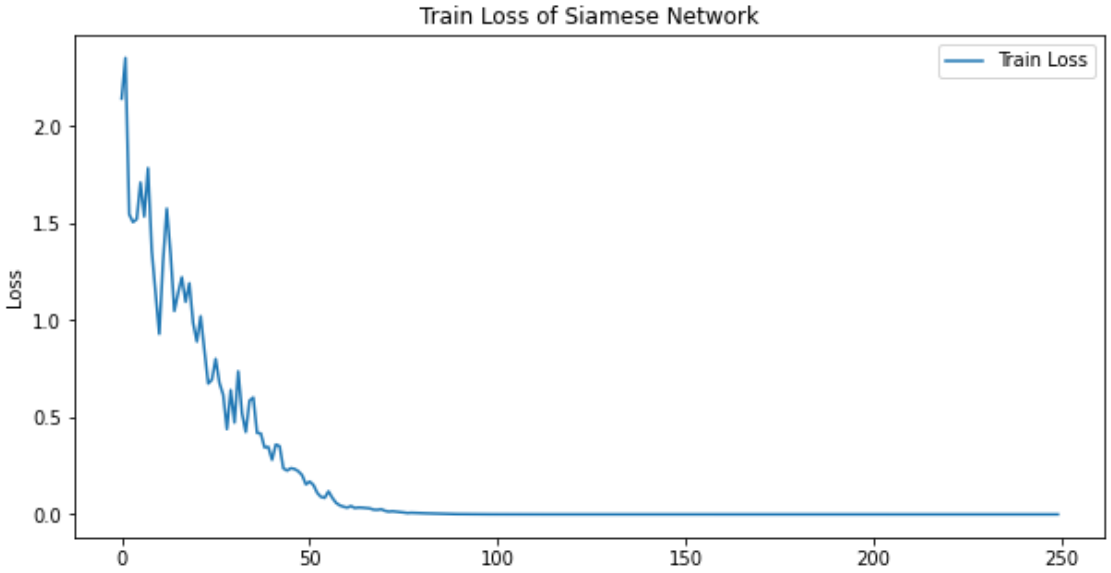

Figura 17- Proceso de entrenamiento red siamesa.

Una vez que disponemos de nuestra red entrenada, el siguiente paso es comprobar si es capaz de diferenciar entre nuevos tipos de células.

3.6 Evaluación de la red y del umbral de clasificación

Una vez entrenada la red, el siguiente paso es introducir los casos de entrenamiento para evaluar su comportamiento. Como podemos ver en los siguientes ejemplos, nuestra red no nos devuelve una predicción, sino que nos devuelve como de diferentes son las 2 imágenes que hemos proporcionado en el input.

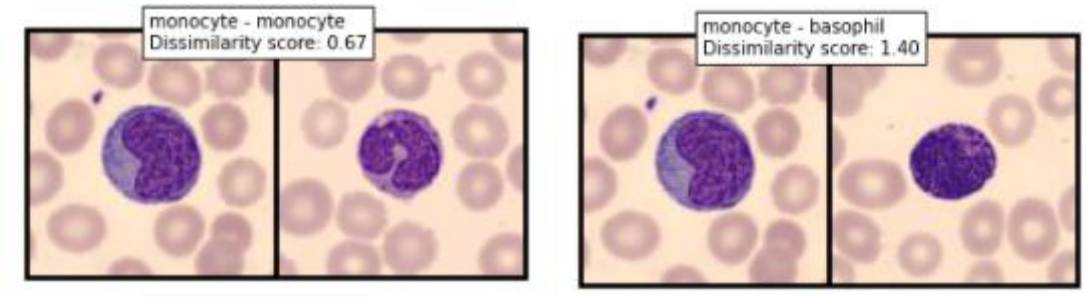

Figura 18- Output red siamesa.

En cambio, el objetivo de nuestra red es verificar si dos imágenes son del mismo tipo de célula, al disponer de un único valor numérico es necesario definir un umbral para traducir el valor número a son iguales/diferentes. Nuestro umbral funcionará del siguiente modo:

Supera Umbral	Misma célula	Predicción
NO	NO	ERROR
NO	SI	OK
SI	NO	OK
SI	SI	ERROR

Tabla 2- Umbral red siamesa.

Definir dicho umbral no es una tarea sencilla, y se decidió ejecutar el programa con diferentes valores y sobre diferentes conjuntos de imágenes. Se comprobó que el punto en el que mejor discriminaba estaba situado entre 0.75 y 1.25. De modo que se decide utilizar 1 como umbral, obteniendo una Accuracy del 80% en nuestro conjunto de entrenamiento.

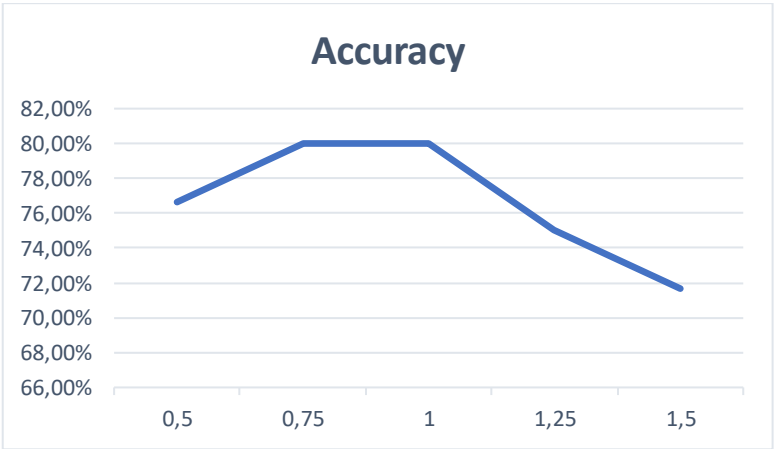

Figura 19- Accuracy red siamesa en clasificación de tipos de células sanas.

Si el modelo estuviese en producción sería crítico revisar y calibrar periódicamente este valor, ya que nuestra red puede hacer un gran trabajo evaluando como de diferentes son los inputs, pero si nuestro umbral es muy alto o bajo no nos servirá para clasificar.

Además de la tarea de clasificación de células sanas se ha probado la red siamesa con el umbral definido para diferenciar linfocitos sanos y linfocitos afectados por linfoma de Burkitt. La tasa de acierto obtenida es un 75%. A

continuación, se muestra un ejemplo de la aplicación, en la que se puede observar que diferencia correctamente las células, puesto que corresponden a clases distintas y tienen una distancia superior al umbral definido.

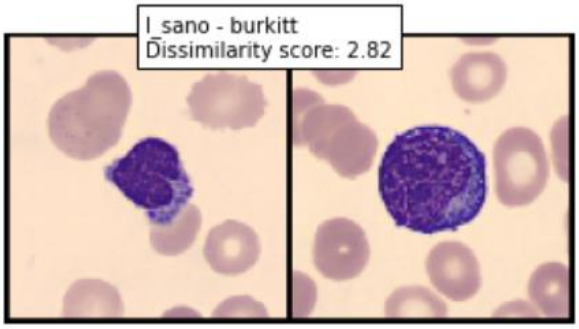

Figura 20- Ejemplo red siamesa Linfocito sano y Linfocito Burkitt.

3.7 Conclusiones de la red siamesa

En el desarrollo de este trabajo hemos conseguido todos los objetivos propuestos:

- Creación de un Pipeline de preprocesado de imágenes automatizado
- Diseño e implementación de una red siamesa completamente funcional en PyTorch.
- Entrenamiento de la red con casos de ejemplo.
- Evaluación de la red con casos de prueba con tipos de células nunca vistos por el modelo.
- Diseño de una metodología para convertir la salida de la red siamesa en una predicción Binaria que indica si dos imágenes pertenecen al mismo tipo de célula o no.

Los resultados son satisfactorios, puesto que se ha conseguido una red siamesa que identifica si dos imágenes corresponden al mismo tipo de célula con una precisión del 80%, empleando tipos no utilizados en el entrenamiento. Adicionalmente hemos utilizado la red para diferenciar entre linfocitos sanos y afectados por linfoma de Burkitt con una tasa de acierto del 75%. Si bien la tasa de acierto no es muy alta, hay que tener en cuenta que no se han utilizado imágenes de linfocitos de ningún tipo en el entrenamiento, por lo que se podría interpretar que el modelo está realizando una generalización para diferenciar células.

Sin embargo, existen varios aspectos sobre los que hubiera sido interesante continuar trabajando:

- Durante la fase de construcción y entrenamiento de la red únicamente se ha trabajado con imágenes de células sanas. Las imágenes de células enfermas no estuvieron disponibles hasta la segunda mitad del semestre, cuando ya se estaba implementando con la segunda red del TFM.

- La subred utilizada es una pequeña red convolucional implementada desde cero. Debido a las limitaciones de tiempo, no ha sido posible implementar versiones siamesas de arquitecturas estándar como VGG19, RESNET, Xception…Y comparar el rendimiento entre ellas

4. Red Few Shot Learning y Finetuning

4.1 Requisitos y objetivos de nuestra red few shot

En el apartado capítulo 3 se desarrolló una red siamesa que era capaz de distinguir si dos imágenes pertenecían al mismo tipo de célula o no. En este apartado el objetivo es construir una red capaz de diferenciar entre células sanas y enfermas. Continuando con el paradigma de técnicas few shot learning, se construirá una red convolucional que será entrenada únicamente con 5 imágenes de linfocitos sanos y 5 afectadas por el Linforma de Burkitt.

En la red siamesa hemos comprobado lo complejo que puede resultar entrenar una red neuronal convolucional desde cero. En esta red se va a utilizar otro enfoque. Se va a realizar fine-tuning de un modelo preentrenado. El fine-tuning es una técnica que consiste en actualizar los parámetros de un modelo preentrenado añadiendo epochs adicionales para realizar una tarea diferente a la tarea para la que fueron diseñados (Howard y Gugger, 2020). El uso del fine-tuning de modelos preentrenados ha sido utilizado con éxito en diferentes casos de uso (Tajbakhsh et al., 2017).

En nuestro caso realizaremos el fine tuning de un modelo preetrenado. El modelo escogido es Resnet18 que, aunque no es de los más avanzados ha demostrado que puede ser utilizado con éxito en contextos médicos. (Ayyachamy et al, 2019). Nuestro fine-tuning consiste en modificar la última capa para conectar a las dos clases de nuestro problema : células linfoides normales o afectadas por linfoma de Burkkit.

4.2 Librerías y paquetes utilizados

El programa ha sido desarrollado en Python 3.6.9 con las siguientes librerías:

- Fastai*: FastAi es una nueva biblioteca de código abierto para Aprendizaje Profundo,está construida sobre PyTorch y proporciona una única API consistente para las aplicaciones de aprendizaje profundo más importantes y los tipos de datos.
- Lightly: es un conjunto de herramientas que permiten obtener más información de nuestros datos y de este modo conseguir que los modelos obtengan mejores predicciones. Se ha utilizado la versión gratuita, en el apartado 4.2 *Preprocesado de datos*, se explica el funcionamiento de la librería.

- OS: Librería que proporciona Python funcionalidades para interactuar con el sistema operativo. La utilizaremos para poder trabajar con ficheros en el sistema de almacenamiento de nuestra máquina.

*Fast.ai importa automáticamente todas las dependencias necesarias de PyYorch que necesita para funcionar.

4.3 Preprocesado de datos

Del mismo modo que en la red siamesa se descomprime un zip que contiene las imágenes con la siguiente estructura de carpetas.

Train	Sano	5 imágenes
	Burkitt	5 imágenes
Test	Sano	Resto de imágenes
	Burkitt	Resto de imágenes

Tabla 3- Estructura de directorios red 5shot.

En esta red además de realizar el fine-tuning hemos realizado pruebas con Lightly.ai. Esta herramienta creada en 2017 y cuya popularidad ha crecido significativamente en los últimos años, utiliza aprendizaje auto supervisado(self-supervised learning) para extraer características de imágenes sin etiquetar. En esta modalidad de aprendizaje los datos de entrenamiento se etiquetan de forma autónoma, identificando relaciones y correlaciones entre ellos. (Zhai et al., 2019).

Existen dos versiones de la aplicación, la gratuita y la versión de pago con funcionalidades más avanzadas. Como el dataset utilizado para nuestro proyecto no es muy grande, la versión gratuita ha sido suficiente.

Las herramientas pueden ser utilizadas a través de la página web o mediante su paquete para Python, en nuestro proyecto red hemos combinado ambas. El Pipeline de Lightly utiliza un modelo para descubrir características y relaciones de forma automática en nuestro conjunto de entrenamiento y las sube a su herramienta web. Actualmente Lightly tiene soporte para dos modelos de self-supervised learning: SlimCLR (Chen et al, 2020) y MoCo (He et al. 2020). En nuestra implementación hemos utilizado SlimCLR puesto que es el recomendado por defecto en la herramienta. Una vez extraídas las características se puede utilizar la interfaz web para analizar las relaciones identificadas y crear subconjuntos con las imágenes que más difieren entre sí.

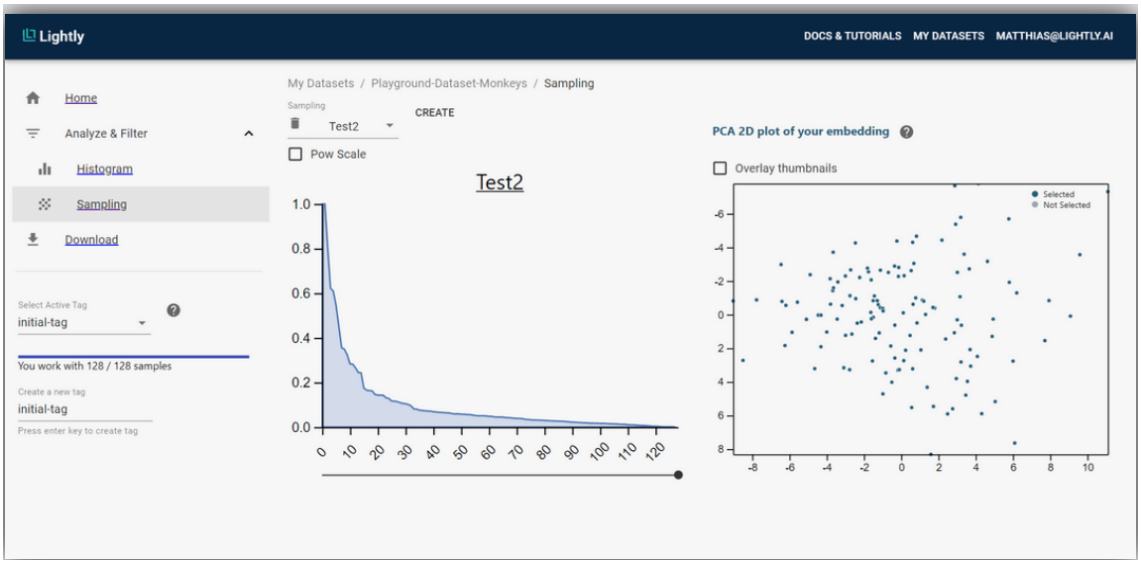

Figura 21- Interfaz web Lightly. Fuente Lightly.ai.

Nuestra red se entrenará y evaluará con dos conjuntos de training/test diferentes, además en cada uno de ellos se evaluarán resultados utilizando y sin utilizar el preprocesado de Lightly. De modo que se implementarán las siguientes versiones de la red:

- 1A: Selección aleatoria de cinco imágenes.
- 1B:Selección aleatoria de cinco imágenes y utilización de las características del preprocesado de Lightly.
- 2A:Preprocesado de Lightly para seleccionar las cinco imágenes más diferentes, entrenamiento de la red sin el preprocesado de Lightly.
- 2B-Preprocesado de Lightly para seleccionar las cinco imágenes más diferentes, entrenamiento de la red con el preprocesado de Lightly.

Una vez se dispone de las imágenes en los directorios, el tratamiento en Fast.ai resulta mucho más sencillo que en PyTorch. A continuación, se muestran las funciones claves utilizadas en el preprocesado y su función:

imagelist.from_folder y *data.split_by_folder* → Se crea el dataloader y se divide entre train y test.

Data.transform (get_transforms(do_flip=True, flip_vert=True), size=224) → Se redimensionan las imágenes a 224 x 224 (tamaño de entrada de nuestra red y se crean imágenes adicionales con data augmentation, en este caso se giran tanto horizontalmente como verticalmente)

data.normalize(imagenet_stats) → Por último y debido a que vamos a usar un modelo preeentrado, se normaliza el Dataset con las estadísticas de los canales RGB del conjunto Imagenet (Deng et al, 2009).

Tras los 3 pasos anteriores dispondremos de un conjunto de datos para entrenar en nuestra red.

4.4 Definición y estructura de la red few-shot

Para la construcción de esta red en lugar de construir una red convolucional desde cero, hemos utilizado un modelo preentrenado. En concreto resnet18, uno de los modelos preentrenados que ha demostrado funcionar en contextos médicos(Ayyachamy et al, 2019). En la figura 22, se muestra un esquema de la estructura de, resnet18 y en la tabla 4 podemos observar las dimensiones de las capas intermedias.

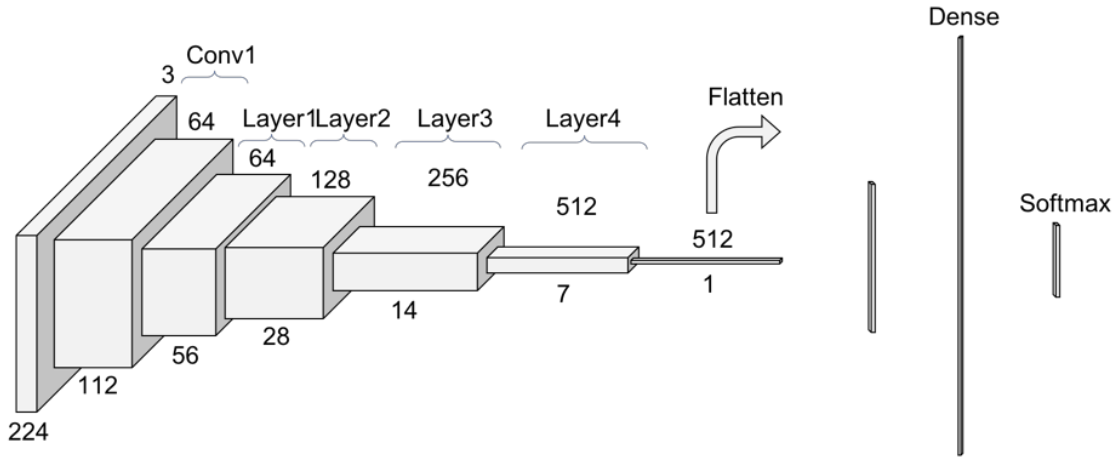

Figura 22- Estructura RESNET18.
Fuente: ***https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8***

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2 $\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Tabla 4- Detalle capas RESNET18.
Fuente ***https://www.researchgate.net/figure/ResNet-18-Architecture_tbl1_322476121***

El modelo Resnet18 es un modelo que ha demostrado su potencial en los últimos años y aunque existen algunos aún más precisos para clasificación

de imágenes continúa siendo un estándar y uno de los más utilizados. Es por ello que los diferentes frameworks de Deep Learning como PyTorch, Fast.ai o keras lo incluyen entre los disponibles.

En nuestra red lo que hacemos es conectar la salida del modelo con nuestras clases, que son Sano o Burkitt, de modo que utilizaremos el modelo Resnet con una ligera modificación para que sea válido para nuestro problema. Esta metodología nos proporciona una serie de ventajas con respecto a implementar la red desde cero:

1. Estamos utilizando un modelo de calidad demostrada, ha sido ampliamente utilizado en competiciones de Deep Learning.
2. Ahorramos una gran cantidad de tiempo y recursos, puesto que lo único que va a realizar nuestro entrenamiento es relacionar los parámetros de salida de Resnet18 con nuestras dos clases. Entrenar desde 0 un modelo como ResNet con más de 11 millones de parámetros es una tarea inabarcable con nuestros recursos.
3. Realizar este entrenamiento con Fast.ai resulta mucho más sencillo que construir una red desde 0, el diseño de redes neuronales es una tarea compleja. Y aunque tras muchos intentos hemos conseguido que nuestra red siamesa tenga un funcionamiento aceptable, no es comparable al rendimiento que podemos conseguir con un finetuning (Howard y Gugger,2020).

Con fast.ai inicializamos nuestra red basada en resnet18 con el siguiente comando:
cnn_learner(data, models.resnet18, metrics=[accuracy])

Siendo data el dataset que hemos preparado en el preprocesado.

4.5 Entrenamiento de la red few shot

En esta red utilizamos el fine-tuning de un modelo preentrenado, son suficientes 16 epochs para conseguir que el modelo alcance una precisión superior al 80% con únicamente 5 imágenes de células sanas y 5 afectadas por linforma de Burkitt. A continuación, se muestra el entrenamiento del conjunto de imágenes 1B (Selección aleatoria de 5 imágenes y utilización de las características del pipeline de preprocesado de Lightly.ai).

learner.fit_one_cycle(16, max_lr=slice(1e-4, 1e-3)) → Comando para definir el entrenamiento y validación.

Epoch	train_loss	valid_loss	accuracy	time
0	1518310	0.754483	0.676471	0
1	1817779	0.929201	0.617647	0

2	1789338	1033556	0.647059	0
3	1621019	0.938753	0.617647	0
4	1502011	0.867014	0.588235	0
5	1385107	0.907207	0.617647	0
6	1302024	0.857800	0.676471	0
7	1134469	0.775299	0.705882	0
8	1166249	0.709685	0.676471	0
9	1075676	0.645650	0.676471	0
10	1045709	0.573515	0.735294	0
11	0.943265	0.564151	0.794118	0
12	0.861522	0.508968	0.764706	0
13	0.793102	0.385025	0.852941	0
14	0.737155	0.332623	0.882353	0
15	0.772725	0.316192	0.852941	0

Tabla 5- Proceso entrenamiento red few-shot.

Como podemos observar con únicamente 16 Epochs y en un tiempo de ejecución de apenas unos minutos conseguimos entrenar y evaluar la red con un único comando de fast.ai.

4.6 Evaluación de la red few shot

En el paso anterior hemos visto como fast.ai dispone de un comando capaz de realizar el entrenamiento y validación de la red en un único paso. Pero como hemos definido 4 conjuntos de entrenamiento y test diferentes. Disponemos de 4 resultados con los que comparar:

Conjunto	Descripción	Accuracy
1A	Aleatorio-5 *clase	79,41%
1B	**Aleatorio- 5* clase + Procesado Ligthly**	**85,29%**
2A	5 *clase más diferentes seleccionadas Ligthly	82,35%
2B	5 * clase más Diferentes Seleccionadas Ligthly + Procesado Ligthly	91,18%

Tabla 6- Accuray de la red few shot.

En la tabla anterior observamos que los resultados son más precisos si se utiliza el pipeline de preprocesado de Lightly, en especial si se seleccionan las 5 imágenes más diferentes gracias a esta herramienta. En nuestro caso consideramos que esa metodología puede estar sesgada por lo que el conjunto que utilizaremos como final del experimento será el 1B- Selección aleatoria de 5 imágenes con preprocesado de Lightly únicamente para extraer características.

A continuación, se muestra la matriz de nuestra red final (conjunto 1-B).

		Predicción	
		Burkitt	Sano
Real	Burkitt	7	4
	Sano	1	22

Tabla 7- Matriz de confusión de red few shot 1B.

Aunque la tasa de acierto del modelo es superior al 85%, la sensibilidad del modelo es reducida (0.64) por lo que células linfoides anormales se están clasificando como normales. La especificidad es adecuada (0.95).

De igual modo que en la red siamesa, podemos obtener mostrar los casos que deseemos, a continuación, mostramos los 3 casos en los que la red ha funcionado peor (en el conjunto 1B).

interp.plot_top_losses(3, figsize=(12,12)) → Comando para mostrar las 3 imágenes con mayor pérdida. Las etiquetas son : Predicción /Clase Real /Pérdida /Probabilidad.

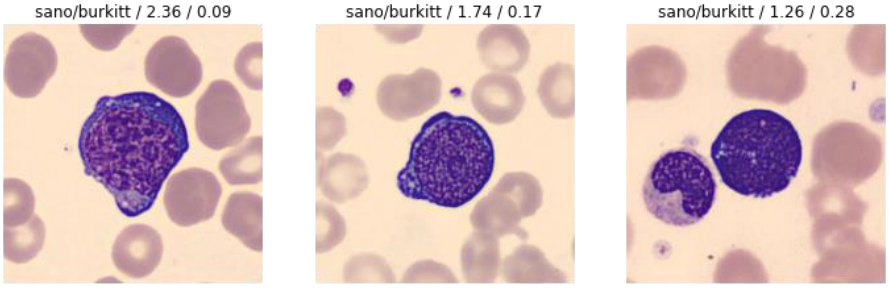

Figura 23- Ejemplos red few shot.

4.7 Conclusiones red few shot

En el desarrollo de este trabajo hemos conseguido todos los objetivos propuestos:

- Creación de una red 5 shot learning basada en un modelo pre-entrenado en Fast.ai.
- Entrenamiento de la red con únicamente 5 casos de cada clase.
- Evaluación de la red con imágenes de pacientes reales.
- Adicionalmente se han realizado pruebas con Lightly.AI una de las herramientas de mayor impacto en la industria en los últimos años. Mediante Self-Supervised Learning es posible preparar y extraer más información de nuestros datos.

Los resultados son satisfactorios, puesto que se ha conseguido una red capaz de diferenciar entre linfocitos sanos y afectados con Burkitt con una precisión superior al 85% utilizando únicamente 10 imágenes en el proceso de entrenamiento con sensibilidad inferior al 70% siendo algo baja para detectar enfermedades.

Sin embargo, existen varios aspectos sobre los que hubiera sido interesante continuar trabajando:

- Únicamente se ha trabajado con imágenes de linfocitos, una mejora importante sería que primero detectase si la imagen proporcionada es

un linfocito u otro tipo de célula y a continuación identificase si está sano o afectado por linfoma de Burkitt.

- La red se ha entrenado con 5 casos y es estática, para su uso en producción sería conveniente diseñar un proceso por el que se entrenara
- Debido al tiempo disponible las pruebas con lightly han sido bastante básicas, utilizando las funcionalidades avanzadas del paquete hubiera sido posible obtener aún más información de nuestros datos.

5. Reproducibilidad y creación de repositorio

Uno de los aspectos más destacables de los últimos años en todo tipo de investigaciones, es que cada vez es más frecuente compartir públicamente los desarrollos y permitir que todo el mundo tenga acceso para consulta, y en caso de que lo deseen reutilizar los desarrollos. Es por ello por lo que desde el principio del proyecto se consideró que además de tener una licencia libre, se crearía un repositorio público en GitHub.

GitHub es un sistema de gestión de proyectos y control de versiones de código, además de una plataforma social diseñada para desarrolladores. En los últimos años se ha convertido en el repositorio online más grande de trabajo colaborativo.

En este proyecto hemos tenido la gran suerte de poder trabajar con un conjunto de imágenes de sangre periférica de gran calidad proporcionadas por el Hospital Clínic de Barcelona. Disponer de dicho conjunto de datos nos ha permitido realizar un trabajo muchísimo mejor que si hubiésemos trabajado con algún dataset público. El único inconveniente es que no pueden compartirse.

Dado que no se puede compartir los desarrollos con los conjuntos de datos reales, se me ocurrió una idea para realizar este apartado sin incumplir el acuerdo de colaboración. En lugar de utilizar imágenes de células para el entrenamiento y evaluación de las redes, se utilizarán imágenes de frutas de un dataset público.

El repositorio está disponible en la siguiente url:

https://github.com/pepelucas/TFM_UOC_Fewshot_learning

Es evidente que no tiene el mismo impacto una red para detectar enfermedades raras que una que clasifique entre distintos tipos de frutas. Pero es una forma de compartir las redes desarrolladas para este TFM y a nivel personal ha sido una forma de aprender cómo se crea y gestiona un repositorio público .

6. Conclusiones

6.1 Conclusiones

Las técnicas Few Shot Learning son de gran utilidad para procesos donde no se dispone de una muestra lo suficientemente amplia para utilizar técnicas tradicionales. A continuación, se resumen los resultados obtenidos:

- Red Siamesa (One Shot). Se ha creado una red especializada en diferenciar tipos de célula, aunque no los haya visto con anterioridad. Los resultados obtenidos son satisfactorios, puesto que entre tres tipos de célula con los que no ha sido entrena obtiene una tasa de acierto del 80%. Adicionalmente se utilizó la red para diferenciar linfocitos sanos y afectados por linfoma de Burkitt, obteniendo una tasa de acierto del 75%. Durante la fase de construcción y entrenamiento de la red, únicamente se ha trabajado con imágenes de células sanas puesto que las imágenes de células enfermas no estuvieron disponibles desde el inicio. La subred utilizada es una pequeña red convolucional implementada desde cero. Hubiera sido idóneo implementar versiones siamesas de arquitecturas estándar como VGG19, RESNET, Xception y comparar el rendimiento entre ellas, pero no ha sido posible debido a las limitaciones de tiempo, por lo que se tendrá en cuenta para futuros desarrollos de esta línea de trabajo.

- Red Five Shot Learning. Esta red ha sido diseñada para identificar enfermedades raras con una muestra de entrenamiento muy pequeña. En nuestra implementación se ha entrenado con cinco imágenes de Leucocitos Sanos y cinco afectados por Linfoma de Burkitt. La tasa de acierto obtenida es superior al 85%.

La tasa de acierto de ambas metodologías es adecuada, las redes son capaces de diferenciar en un 75 y 85% de casos respectivamente, pero insuficiente para usarla como prueba diagnóstica. Sin embargo, ambas técnicas sí que podrían utilizarse para localizar células anómalas en un pipeline de trabajo normal. Un ejemplo de implementación que podría utilizar las dos redes en un entorno real es:

Obtención de las imágenes de los pacientes en laboratorio, a continuación, se utilizaría la red siamesa para localizar células muy diferentes a los tipos conocidos. Una vez localizadas dichas células se podría utilizar la red five-shot para identificar si tienen similitud con alguna enfermedad conocida. Por último, se remitiría un informe con los indicios para ser analizadas por el personal experto.

Es importante matizar que tanto la red siamesa como la red few shot implementadas son especialmente sensibles a las características de los ejemplos con las que se entrenan. De modo que, si se planea utilizarlas en un entorno de producción, es importante definir un protocolo para re-entrenar las redes cuando se disponga de nuevos casos. Del mismo modo, aunque la implementación es genérica y debería funcionar para cualquier problema

similar, dependiendo de las características de las imágenes las redes pueden ser adecuadas o necesitar adaptación.

6.2 Reflexión y análisis.

Aunque se han cumplido los objetivos planificados para este TFM, existen varios aspectos sobre los que es conveniente tratar en este apartado.

- Planificación: La planificación inicial era demasiado optimista, en especial en el apartado de formación en PyTorch y Few Shot Learning donde se planificaron 45 horas y el tiempo real empleado ha sido superior a las 100 horas. Adicionalmente ha sido necesario realizar formación en Fast.ai para implementar la red Few-Shot.

- Tratamiento de las imágenes y Data Augmentation: Debido al tiempo disponible la prioridad ha sido a tener redes funcionales para la fecha fin del proyecto, sin poder profundizar tanto como hubiera sido deseable en estos aspectos. Se han utilizado las imágenes con sus colores RGB o Blanco y negro en las redes, quedando pendiente la realización de pruebas con filtros adicionales. Al disponer de pocos ejemplos es buena idea utilizar técnicas de data augmentation para maximizar la información que se puede extraer de nuestros casos, debido a la falta de tiempo únicamente se incorporaron simetrías tanto horizontales como verticales, quedando pendiente realizar pruebas con otras transformaciones como rotaciones o recortes.

- Enfoques híbridos: En este proyecto se han utilizado las ténicas Few-Shot Learning de forma aislada, en un entorno real sería adecuado desarrollar técnicas híbridas combinando enfoque one-shot y redes convolucionales tradicionales. Es decir, desarrollar un pipeline que utilice ténicas Few-Shot hasta que se disponga de una muestra robusta para la utilización de técnicas tradicionales. Las técnicas Few-Shot han demostrado ser bastante capaces, pero al utilizar pocos casos para el entrenamiento (entre 5 y 10 en nuestras redes) es más probable que no recojamos con suficiente precisión las características intrínsecas y la variabilidad intraclase.

A continuación, resumo las líneas de trabajo futuras:

- Mejora de las redes y puesta en producción.
  Aunque las redes obtenidas en el proyecto han demostrado un funcionamiento adecuado, todavía se pueden afinar con los aspectos mencionados en el apartado anterior, además de incluir nuevas enfermedades raras además de Linfoma de Burkitt. Una vez ajustadas, un proyecto interesante sería realizar una colaboración con un hospital o laboratorio que suministrara periódicamente nuevas imágenes y se pasaran por ambas redes. Por la red siamesa para ver si existe algún tipo de célula que difiera mucho de los tipos habituales y por la segunda para ver si existiese alerta de algún tipo de enfermedad rara.

- Traducción del código a R
  Al comienzo del proyecto se eligió el lenguaje Python puesto que los principales frameworks de Deep Learning estaban implementados en Python. En los últimos meses ha sido lanzada la versión de Torch nativa para R. De modo que una línea de trabajo propuesta sería migrar todo el proyecto a R, lenguaje de programación con el que la comunidad Bioinformática está más familiarizada.

- Creación de una interfaz web para el usuario.
  Otra línea de trabajo interesante es la creación de un Front-End de usuario, de modo que un usuario sin conocimientos de programación se capaz de utilizar nuestras redes Few-Shot. Dependiendo de si se realizare en Python o R, se podría realizar un desarrollo medianamente rápido con Flask o Shiny respectivamente.

- Creación de un paquete especializado para Python/R.
  Si el proyecto alcanzase el punto de madurez necesario, el siguiente paso es compartirlo con la comunidad en forma de paquete. La creación del paquete va más allá de la distribución del código fuente, es necesario proporcionar al usuario una experiencia completa, un código robusto que funcione sin errores, una documentación detallada, correctamente documentado y con su correspondiente gestión de errores.

7. Glosario

Acrónimos utilizados en este TFM:

API: Application Programming Interface
CNN: Convolutional Neural Network
CPU: Central Processing Unit- Procesador PC
DL: Deep Learning
IA: Inteligencia artificial
GAN: Generative Adversarial Network- Tipo de red neuronal frecuentemente utilizado para la generación de nuevos ejemplos.
GPU: Graphics Proccesing Unit- Trarjeta Gráfica
ML: Machine Learning
PC: Personal Computer
RGB: Representación de color mediante 3 canales Red Green and Blue.
RAM: Memoria de acceso aleatorio
TFM: Trabajo fin de máster.

8. Bibliografía

Acevedo, A., Merino, A., Alférez, S., Molina, Á., Boldú, L., & Rodellar, J. (2020). A dataset of microscopic peripheral blood cell images for development of automatic recognition systems. *Data in Brief*, 105474.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017, August). *Understanding of a convolutional neural network*. In *2017 International Conference on Engineering and Technology (ICET)* (pp. 1-6). IEEE.

American Cancer Society. (2020). *Acerca del linfoma no Hodgkin. Visión general y tipos. Accedido diciembre de 2020* https://www.cancer.org/content/dam/CRC/PDF/Public/9079.00.pdf

Auffarth, B. (2020). *Artificial Intelligence with Python Cookbook*. Packt Publishing.

Ayyachamy, S., Alex, V., Khened, M., & Krishnamurthi, G. (2019, March). *Medical image retrieval using Resnet-18*. In *Medical Imaging 2019: Imaging Informatics for Healthcare, Research, and Applications* (Vol. 10954, p. 1095410). International Society for Optics and Photonics.

Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: *A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255)*. Ieee.

Dey, S., Dutta, A., Toledo, J. I., Ghosh, S. K., Lladós, J., & Pal, U. (2017). Signet: Convolutional siamese network for writer independent offline signature verification. *arXiv preprint arXiv:1707.02131*.

Fast.ai. Accedido octubre de 2020. Recuperado de https://www.fast.ai/

FloydHub Blog*. (2019). N-Shot Learning: Learning More with Less Data.* Accedido octubre de 2020. Recuperado de *https://blog.floydhub.com/n-shot-learning/*

Gerón, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media.

Gupta, H*. (2017). harveyslash/Facial-Similarity-with-Siamese-Networks-in-Pytorch.* Accedido septiembre de 2020. Recuperado de *https://github.com/harveyslash/Facial-Similarity-with-Siamese-Networks-in-Pytorch*

Hadsell, R., Chopra, S., & LeCun, Y. (2006, June). *Dimensionality reduction by learning an invariant mapping.* In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (Vol. 2, pp. 1735-1742). IEEE.

He, K., Fan, H., Wu, Y., Xie, S., & Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 9729-9738).

Howard, J., & Gugger, S. (2020). *Deep Learning for Coders with fastai and PyTorch*. O'Reilly Media.

Innovation Incubator Group of Companies *(2019). Siamese Neural Network (With Pytorch Code Example).* Accedido octubre de 2020. Recuperado de *https://innovationincubator.com/siamese-neural-network-with-pytorch-code-example/*

Jadon, S., & Garg, A. (2020). *Hands-On One-shot Learning with Python*. Packt Publishing.

Kaggle*. (2014). Dogs vs. cats.* Accedido octubre de 2020. Recuperado de *https://www.kaggle.com/c/dogs-vs-cats/overview/description*

Koch, G., Zemel, R., & Salakhutdinov, R. (2015, July). *Siamese neural networks for one-shot image recognition*. In *ICML deep learning workshop* (Vol. 2).

Lightly Data Preparation for Machine Learning. Accedido noviembre de 2020. Recuperado de https://www.lightly.ai/

Montavon, G., Samek, W., & Müller, K. R. (2018). *Methods for interpreting and understanding deep neural networks.* Digital Signal Processing, *73*, 1-15.

Neurohive *(*2019*). VGG16 - Convolutional Network for Classification and Detection.* Accedido octubre de 2020. Recuperado de *https://neurohive.io/en/popular-networks/vgg16/*

Noriega, L. (2005). Multilayer perceptron tutorial. *School of Computing. Staffordshire University*.

PyTorch. Accedido septiembre de 2020. Recuperado de https://www.pytorch.org/

Romero, M. D. (2013). *Manual del hemograma y el frotis de sangre periférica*. Universidad de los Andes.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition*. *arXiv preprint arXiv:1409.1556*.

Singh Grewal, D. (2014). *A Critical Conceptual Analysis of Definitions of Artificial Intelligence as Applicable to Computer Engineering*. *IOSR Journal of Computer Engineering*, *16*, 09-13.

Stevens, E., Antiga, L., & Viehmann, T. (2020). Deep Learning with Pytorch. Manning.

Subramanian, V. (2018). Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch. Packt Publishing Ltd.

Susmelj, I. *(2020). Few-Shot Learning with Fast.Ai.* Accedido noviembre de 2020. Recuperado de https://towardsdatascience.com/few-shot-learning-with-fast-ai-81c66064e372

Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., & Liang, J. (2016). *Convolutional neural networks for medical image analysis: Full training or fine tuning?*. *IEEE transactions on medical imaging*, *35*(5), 1299-1312.

Tiwari, S*. (2019). Humpback-Whale-Identification-Competition--Kaggle.* Accedido septiembre de 2020. Recuperado de *https://github.com/SwatiTiwarii/Humpback-Whale-Identification-Competition--Kaggle/*

Xeridia *(2019). Redes Neuronales artificiales: Qué son y cómo se entrenan.* Accedido octubre de 2020. Recuperado de https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i

Zhai, X., Oliver, A., Kolesnikov, A., & Beyer, L. (2019). S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE international conference on computer vision* (pp. 1476-1485).

Zhao, Z., & Liu, H. (2007, June). Spectral feature selection for supervised and unsupervised learning. In *Proceedings of the 24th international conference on Machine learning* (pp. 1151-1157).