

E.T.S. de Ingeniería Industrial, Informática
y de Telecomunicación

Análisis automático de imágenes de frotis de sangre periférica para diagnóstico de Leucemia



Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Hodei Zia López

Director: José Antonio Sanz Delgado

Pamplona, 19/01/2021

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen

Este proyecto trata sobre cómo conseguir detectar la leucemia de una forma rápida y poco costosa para que pueda ser accesible por todo el mundo. Para ello, solamente necesitaríamos una muestra de sangre del paciente con la que aplicar un frotis de sangre periférico para poder obtener muestras válidas para analizar. A continuación, tras haber obtenido dichas muestras, aplicaríamos técnicas de visión por computación y aprendizaje automático para poder clasificarlas. En caso de que alguna de las muestras del paciente fuera clasificada dentro de la clase “Leucemia”, avisaríamos al paciente para que pudiera empezar a tratarse lo antes posible. Además, con el fin de ir mejorando nuestro dataset, siempre y cuando el paciente nos lo permita, y tras el diagnóstico por parte de los servicios de salud, iríamos añadiendo las muestras dentro de nuestro dataset. Para facilitar el trabajo en futuras mejoras e ir analizando mejor los resultados, crearemos una base de datos manejada por una interfaz gráfica, en donde almacenaremos todos los datos obtenidos tras cada prueba del código.

Palabras clave

- Leucemia
- Leucocitos
- Aprendizaje automático
- Aprendizaje profundo
- Visión por computación
- Clasificación
- Clustering
- Base de datos
- Interfaz gráfica
- Frotis de sangre periférico

Índice

Resumen	3
Palabras clave	3
Índice	4
Capítulo 1	6
Introducción.....	6
1.1. Objetivo	7
Capítulo 2	8
Preliminares.....	8
2.1. La leucemia.....	8
2.1.1. Tipos de leucemias	9
2.2. Visión por computador	10
2.2.1. Conversiones de color	11
2.2.2. Segmentación basada en regiones	11
2.2.3. Tratamiento de la intensidad de los píxeles.....	11
2.2.4. Detección de formas geométricas.....	11
2.2.5. Reducción de imágenes	12
2.2.6. Recoloración de los píxeles	12
2.3. Aprendizaje Automático	13
2.3.1. Clasificación	13
2.3.2. KNN	14
2.3.3. Árboles de decisión	15
2.3.4. Regresión logística	15
2.3.5. Redes Neuronales	16
2.3.6. SVM	16
2.3.7. Random Forest.....	17
2.4. Entorno de programación.....	18
2.4.1. Lenguajes de programación.....	18
2.4.2. Herramientas de desarrollo	18
2.4.3. Requisitos mínimos	19
2.4.4. Librerías.....	20
Capítulo 3	21
Desarrollo del proyecto	21

3.1. Datasets candidatos.....	21
3.1.1. Dataset 1: LISC	22
3.1.2. Dataset 2: SN-AM	23
3.1.3. Dataset 3: AML-Cytomorphology_LMU.....	24
3.1.4. Dataset 4: ALL_IDB_Dataset	25
Capítulo 4	26
Implementación	26
4.1. Fase 1: Preprocesamiento de imágenes.....	26
4.1.1. Algoritmo de K-Means	26
4.2. Fase 2: Generación de variables	34
4.2.1. Variables del Dataset original.....	34
4.2.2. Variables del Dataset en rojo, blanco y morado	36
4.2.3. Variables del Dataset de K-Means	36
4.3. Fase 3: Clasificación	37
4.3.1. Clasificación manual	38
4.3.2. Clasificación automática.....	40
4.3.3. Selección de variables	43
4.4. Fase 4: BBDD y UI.....	49
4.5. Fase 5: Deep Learning	53
Capítulo 5	58
Pruebas y validación.....	58
Capítulo 6	60
Conclusiones y líneas futuras	60
Bibliografía.....	62

Capítulo 1

Introducción

El cáncer es una de las enfermedades genéticas más temidas por la humanidad, ya que existe la falsa creencia de que es sinónimo de muerte. Es una palabra casi tabú que muchas veces evitamos utilizar.

Sin embargo, y pese a que es verdad que existe una mayor incidencia de este tipo de enfermedades por los cambios económicos, demográficos y ambientales que se han dado desde el inicio de la revolución industrial, también es cierto que cada vez se consigue la curación de una mayor proporción de casos de cáncer. Además, gracias a los avances tecnológicos, cada vez son más los síntomas que se pueden tratar y controlar de forma adecuada [1]. Vivimos en una época en la que, afortunadamente, curarse de un cáncer es más probable que nunca.

Uno de los cánceres más comunes según el Instituto Nacional del Cáncer de Estados Unidos, es la leucemia [2]. Esta será la enfermedad sobre la que centraremos proyecto, basándome y haciendo uso de muestras obtenidas mediante frotis de sangre periférica, técnica que consiste en examinar por medio de un microscopio una muestra de sangre la cual se ha sometido a un tratamiento especial para poder visualizar los diferentes tipos de células que contiene. Esta técnica es de gran utilidad a la hora de diagnosticar problemas en la sangre [3].

Para poder extraer la mayor cantidad posible de información de cada muestra, haremos uso de la visión por computador y del aprendizaje automático.

La idea que tenemos con este proyecto es que del mismo modo que cuando nos hacen una analítica nos llegan a casa los resultados del nivel de todos nuestros componentes sanguíneos, también nos llegue un aviso en caso de que nuestra muestra haya sido clasificada como leucémica. De este modo, queremos que, ante la mínima sospecha de tener la enfermedad, el paciente sea consciente para así poder empezar a tratarlo antes de que sea demasiado tarde.

Tal y como hemos mencionado, para la realización de este proyecto haremos uso de distintas técnicas o disciplinas. Con ello, lo que se busca es aplicar todo lo aprendido durante la carrera, intentando englobar el máximo posible de asignaturas, o al menos, aquellas con las que más he aprendido.

1.1. Objetivo

El objetivo de este proyecto es detectar la presencia anómala de leucocitos (glóbulos blancos) en sangre para determinar si una persona sufre de leucemia, y así poder empezar a tratarla antes de tener síntomas y cuando todavía es pronto.

Para ello, separaremos el proyecto en distintas fases:

1. En la primera fase, haremos una segmentación de los leucocitos frente a las demás células sanguíneas que se encuentren en cada muestra del dataset mediante técnicas de visión por computador. También trataremos de normalizar las imágenes de modo que el algoritmo funcione para todo tipo de muestra, independiente de su procedencia (color, calidad...).
2. En la segunda fase, basándonos en los resultados previos, trataremos de encontrar patrones con los que clasificar cada muestra. Para esta fase usaremos técnicas combinadas tanto de visión por computador como de minería de datos.
3. En la tercera fase, por medio de Machine Learning [4], buscaremos generar las mejores clasificaciones posibles haciendo uso de diversos clasificadores (Redes Neuronales, KNN, Árboles de Decisión...). Haremos a su vez un estudio de cada característica para determinar la importancia de cada una en nuestro clasificador.
4. En la cuarta fase, y una vez obtenidos los resultados, crearemos una base de datos en la que recopilaremos toda la información necesaria para la reproducción de los resultados obtenidos (valores de los hiper parámetros utilizados en los clasificadores, valor de las características, clasificadores, porcentajes...). Además, esta base de datos nos servirá para hacer consultas estadísticas sobre los resultados y así poder extraer más información. Con el fin de facilitar su uso, crearemos una interfaz gráfica en donde podremos introducir los datos directamente a la base de datos sin necesidad de saber ningún lenguaje de programación. También se podrá utilizar para generar y almacenar consultas en ficheros.
5. Finalmente, trataremos de repetir la fase 3 por medio de Deep Learning [5]. Para ello tendremos que dividir la fase en dos: formación y aplicación de lo aprendido.

Capítulo 2

Preliminares

En este capítulo vamos a presentar los conceptos necesarios para entender el proyecto, los cuales serán una breve descripción sobre la leucemia, las técnicas informáticas aplicadas y finalmente, el entorno de programación utilizado junto a los requisitos del sistema.

2.1. La leucemia

La leucemia es uno de los tres tipos principales de cáncer de sangre. Esta enfermedad comienza en el tejido blando que hay en los huesos, lugar en donde se forman las células sanguíneas, el cual se denomina médula ósea.

La médula ósea, tal y como ya hemos dicho anteriormente, es la encargada de generar todas las células sanguíneas necesarias para el correcto funcionamiento del cuerpo humano. Entre estas células, nos encontramos con:

- Los glóbulos rojos o hematíes. Son los encargados de llevar el oxígeno a las células.
- Los glóbulos blancos o leucocitos. Nos defienden de las infecciones y otras enfermedades.
- Las plaquetas. Evitan las hemorragias formando un coágulo alrededor de las heridas.

En esta enfermedad, la médula ósea produce glóbulos blancos anómalos y en cantidades incontrolables, los cuales se denominan células blásticas leucémicas o más comúnmente como células de leucemia.

Estas células cancerígenas son capaces de dividirse reproduciéndose a sí mismas y evitando morir al envejecer o dañarse, con lo que impiden la correcta reproducción de las demás células sanguíneas (incluyendo los glóbulos blancos saludables) al acumularse en el torrente sanguíneo [6].

2.1.1. Tipos de leucemias

Las leucemias pueden ser clasificadas de dos formas en función de distintas características [7].

1. En función de la velocidad a la que evoluciona la enfermedad:
 - **Aguda:** Se produce a gran velocidad. La mayoría de los glóbulos blancos que se reproducen son inmaduros.
 - **Crónica:** Se producen muy lentamente. Las células cancerígenas comienzan con un comportamiento correcto que hace que sea más difícil de detectarse. A medida que pasa el tiempo, van obstruyendo la reproducción de las demás células sanguíneas hasta que acaban por convertirse en la mayoría.
2. En función del tipo de célula que afectan:
 - **Mieloide:** Comienza en las células mieloides.
 - **Linfoide:** Comienza en las células linfoides.

Así pues, podemos distinguir cuatro tipos de leucemias:

- **Leucemia mieloide aguda (LMA):** Afecta a las células mieloides y se desarrolla con gran velocidad. Se presenta en personas de avanzada edad, aunque puede afectar también a adolescentes.
- **Leucemia mieloide crónica (LMC):** Comienza afectando lentamente a las células mieloides. Al igual que la LMA, afecta a personas de avanzada edad, aunque también a adolescentes.
- **Leucemia linfoblástica aguda (LLA):** Afecta a las células linfoides y se desarrolla a mayor velocidad. Afecta a personas jóvenes de entre 2 y 5 años.
- **Leucemia linfocítica crónica (LLC):** Comienza afectando lentamente a las células linfoides. A diferencia de la LLA, este tipo de leucemia afecta en su gran mayoría a personas de avanzada edad y muy rara vez en personas jóvenes.

En este proyecto, trataremos la leucemia linfoblástica aguda o LLA.

2.2. Visión por computador

La visión por computador es una técnica dentro del área de la inteligencia artificial que trata de emular la visión humana para utilizar técnicas de conocimiento y así poder tomar decisiones sobre un conjunto de imágenes [8]. Dentro de estas técnicas, existen distintos procesos. Para nuestro proyecto, haremos uso de las tres:

- **Procesos de bajo nivel:** Dentro de estos procesos se encuentran las que tienen como objetivo la edición de la imagen para alterar o realzar alguna de sus características (brillo, rango dinámico...).
- **Procesos de medio nivel:** Entre los procesos de medio nivel, tenemos los que tratan de extraer algún tipo de información de la imagen (bordes, descripciones...).
- **Procesos de alto nivel:** Los procesos de alto nivel tratan de darle sentido al conjunto de objetos encontrados en la imagen, así como analizarla para poder llevar a cabo funciones cognitivas normalmente asociadas con la visión.

En nuestro proyecto, la entrada es un conjunto de imágenes del cual tenemos que extraer toda la información necesaria para poder clasificar cada una de las muestras. Al tratarse de imágenes y tener que simular la visión humana, debemos hacer uso de la visión por computador.

Hemos usado distintos tipos de procesos para tratar las imágenes y poder sacar información de ellas. A continuación, nombraremos y explicaremos las más recalables, llegando a entrar en detalle más adelante:

- Conversión del color de las imágenes (RGB, HSV, HSL, CMYK...) y ampliación del rango dinámico para intentar resaltar y diferenciar entre si los distintos objetos de la muestra.
- Segmentación basada en regiones con el algoritmo de K-Means para poder diferenciar mejor los distintos objetos de la imagen.
- Tratamiento de la intensidad de los píxeles para determinar el porcentaje de colores dentro de una muestra y para suprimir información innecesaria (fondo).
- Detección de formas geométricas para el conteo de leucocitos.
- Reducción de imágenes para poder tratarlas ocupando menos espacio en memoria.
- Recolorear los píxeles de la imagen en base a su intensidad para normalizar las muestras y reducir el ruido.

2.2.1. Conversiones de color

Las conversiones de color de las imágenes sirven para intentar buscar algún espacio de color con el que destacar partes de la imagen que en la imagen original no resaltaban tanto. Lo que se busca con ello es poder encontrar un espacio de color que sea de mayor utilidad a la hora de tratar las imágenes. Además, cada espacio de color se trata con modificaciones. Un ejemplo de esto último sería poner a 0 la tonalidad azul en una imagen RGB. Una de estas conversiones es el Rango Dinámico, el cual expande las intensidades de nuestros histogramas de color para que abarquen todo el rango de 0 a 255.

2.2.2. Segmentación basada en regiones

La segmentación consiste en dividir una imagen en regiones denominadas segmentos con el fin de poder asignar una clase o categoría a cada uno de estos segmentos para poder darles un significado en el mundo real. Para ello, hemos utilizado el algoritmo de K-Means, el cual consiste en agrupar cada píxel de la imagen en base a sus intensidades. Para nuestro proyecto, hemos intentado generar tres regiones distintas: fondo, glóbulos rojos y glóbulos blancos.

2.2.3. Tratamiento de la intensidad de los píxeles

En el tratamiento de la intensidad de los píxeles, buscamos inspeccionar el histograma de las imágenes en busca de patrones como pudieran ser los picos, los posibles umbrales o la diferencia de píxeles entre los picos. Otra utilidad que le podemos dar a las intensidades es calcular el porcentaje de cierto rango para llegar a una aproximación de la diferencia entre el fondo, los glóbulos blancos y los glóbulos rojos.

2.2.4. Detección de formas geométricas

Para la detección de formas geométricas aplicamos la Transformada de Hough. Esta técnica, la cual se usa normalmente como herramienta de detección de curvas, sirve para la detección de figuras en imágenes digitales. En nuestro caso, usaremos esta técnica para lograr detectar y contar la cantidad de leucocitos de la imagen.

2.2.5. Reducción de imágenes

Con la reducción de imágenes intentamos comprobar si las imágenes originales son capaces de perder en resolución sin que esto repercuta en los resultados finales. Con ello lo que buscamos es ver hasta qué punto podemos ahorrar espacio sin que esto conlleve a ninguna penalización.

2.2.6. Recoloración de los píxeles

La recoloración de los píxeles sirve para normalizar todas las muestras dentro de unas mismas intensidades y espacios de color. Con esto lo que buscamos es que cada muestra que obtengamos, independientemente de cómo haya sido tintada tras el frotis de sangre, pueda llegar a tratarse de igual forma que las demás. Un ejemplo de esto sería recolorar el fondo siempre de blanco para así poder distinguirlo más fácilmente y que tenga la misma intensidad, aunque provenga de muestras con distintas intensidades de tintes.

2.3. Aprendizaje Automático

El aprendizaje automático es un campo dentro de la inteligencia artificial que intenta que una máquina sea capaz de aprender patrones para predecir comportamientos futuros con la menor supervisión posible de un humano [9]. El aprendizaje puede ser de distintos tipos:

- **Supervisado:** En este tipo de aprendizaje, partimos de unos datos de entrenamiento los cuales están etiquetados de modo que sabemos cuál debería de ser su salida deseada. El objetivo es crear una función capaz de predecir el valor o comportamiento de un objeto basándonos en los de entrenamiento.
- **No supervisado:** En el aprendizaje no supervisado, a diferencia con el supervisado, no tenemos ningún conocimiento a priori, por lo que tenemos que intentar ajustar el modelo en base a observaciones o suposiciones que sigan

Dentro del aprendizaje supervisado tenemos los siguientes problemas:

- **Clasificación:** El valor a predecir es una clase o categoría entre un número limitado de clases.
- **Regresión:** El valor a predecir es un número dentro de un conjunto infinito de posibles resultados.

En cuanto al aprendizaje no supervisado tenemos el siguiente problema:

- **Clustering:** Consiste en agrupar una serie de vectores en base a unos criterios comunes. En otras palabras, agrupamos dentro de un mismo grupo si comparten propiedades comunes o tienen propiedades cercanas.

Nuestro problema se centrará en la clasificación pese a tener una parte de clustering a la hora de recolorar.

2.3.1. Clasificación

Tal y como hemos comentado anteriormente, la clasificación es una aplicación del aprendizaje supervisado en donde obtenemos una clase o categoría en base a los parámetros de entrada que le introduzcamos.

Dentro de los datos de entrada solemos tener una lista de elementos a clasificar. El problema fundamental de la predicción está en modelar la relación entre las variables de estado para obtener el valor de la variable de control. Esto está directamente relacionado con la separabilidad de las clases. Para clasificar, dividimos nuestros datos en conjuntos de train y test, de donde tras entrenar con un número idóneo de elementos de los cuales sabemos sus clases y generar un modelo, probamos a predecir las clases de un conjunto de test para obtener el porcentaje de acierto y así determinar si clasificamos de forma correcta o no.

Una de las partes más importantes de nuestro proyecto tal y como ya hemos dicho, se trata de clasificar cada muestra en base a las características extraídas gracias a la visión por computador. Para ello, hemos hecho uso de distintos métodos de clasificación, todos ellos supervisados:

- **KNN:** Este método determina la clase de una muestra en función a su cercanía con respecto a los K vecinos más cercanos.
- **Árboles de decisión:** Los árboles de decisión determinan la clase de la muestra en base al valor de sus características con las cuales van mapeando el camino desde el nodo raíz hasta la hoja que determina la clase.
- **Regresión logística:** Es un método que permite estimar la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa.
- **Redes neuronales:** Son sistemas computacionales inspirados en neuronas que tienen nodos interconectados simulando conexiones.
- **SVM:** Este método trata de buscar una separación entre las distintas clases. La separación correcta se logra maximizando el margen que tienen ambas clases respecto a la separación.
- **Bosques aleatorios:** Es una combinación de árboles de decisión no correlacionados que promedia el resultado obtenido por cada árbol.

2.3.2. KNN

KNN o K-Nearest-Neighbor (K vecino más cercano), es un algoritmo de clasificación supervisada que busca en las observaciones más cercanas a la que se está tratando de predecir y clasifica la instancia basada en la mayoría de los datos que le rodean. En otras palabras, lo que hacemos es calcular la “similitud” o “distancia” que tiene un elemento con todos los demás ya clasificados para ver con que clase o grupo tiene mayores coincidencias. Si K fuera igual a 3, tendríamos que ver de qué clase son los 3 elementos más cercanos en cuanto a similitud a nuestro elemento y clasificarlo dentro de la clase mayoritaria dentro de esos 3 elementos vecinos. Es por esto por lo que solemos usar una K impar, ya que, de este modo, no obtendremos empates con problemas binarios como es el caso de este proyecto. Algo negativo de este clasificador, es que utiliza todo el dataset para entrenar cada punto, por lo que requiere un gran uso de memoria. En la figura 1, se muestra un ejemplo de un KNN con $K = 4$ en donde vemos como el elemento será clasificado como parte de la clase azul [10].

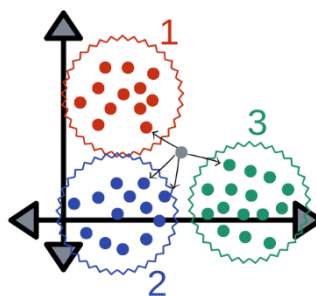


Figura 1. Ejemplo del clasificador KNN

2.3.3. Árboles de decisión

Los árboles de decisión son unos de los algoritmos más utilizados en el aprendizaje automático por su fácil interpretación. Un árbol de decisión es un modelo predictivo que divide el espacio de los predictores agrupando observaciones con valores similares para la variable de respuesta (clase). Los árboles están formados por nodos que van separándose en ramas tras una operación como podría ser: “Valor > 10” o “Color = Amarillo”. Al final, los nodos hoja (nodos finales sin hijos), deberán de englobar las instancias de una única clase. Cuanta menor es la profundidad del árbol, mejor suele ser la clasificación, ya que esto quiere decir que llegamos a una correcta clasificación en menos pasos. En la figura 2, un ejemplo de dos árboles de decisión obtenidos en este proyecto [11].

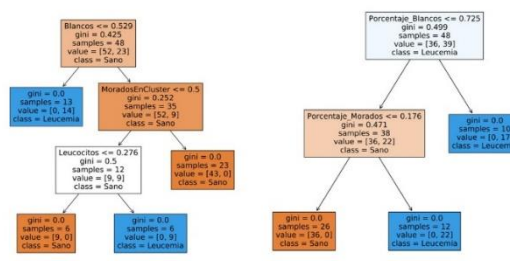


Figura 2. Ejemplo de dos Árboles de Decisión

2.3.4. Regresión logística

La regresión logística es una especie de red neuronal en miniatura. De hecho, se trata de una red neuronal con una sola neurona. Esta técnica, describe y estima la relación entre una variable binaria dependiente y las variables independientes. Es una técnica muy buena a la hora de hacer clasificaciones binarias. Para la clasificación, genera una función sigmoide y estima una clase en base a como se encuentren las variables en relación con la curva generada. Si la salida de la función sigmoide es mayor que 0.5, podemos clasificar el resultado como 1, mientras que, si es menor, lo haremos como 0. Además, podemos usar el valor de la salida para determinar el porcentaje que tiene el paciente de estar dentro de la clase 1 o de la 0 [12]. En la figura 3, se muestra la curva de la función sigmoide de la que se hablaba anteriormente.

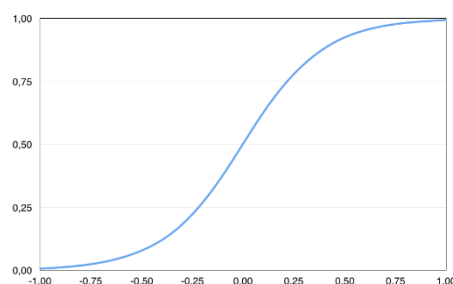


Figura 3. Curva de la función sigmoide de la Regresión Logística

2.3.5. Redes Neuronales

Las redes neuronales son un conjunto de neuronas conectadas entre sí que trabajan en conjunto. Están basadas en las redes neuronales biológicas. En el caso de las artificiales, la suma de las entradas multiplicadas por sus pesos asociados determina el impulso que recibe cada neurona. Las neuronas se encuentran agrupadas en distintos niveles o capas, las cuales son un conjunto de neuronas que reciben como entrada la salida de la capa anterior. Podemos diferenciar tres capas distintas: La capa de entrada, las capas ocultas y la capa de salida. Entrenar una red neuronal consiste en ajustar cada uno de los pesos de las entradas de todas las neuronas lo máximo posible de modo que, si se produce un error muy pronunciado como salida de la última capa, se pueda volver hacia atrás ajustando los pesos para ir obteniendo mejores resultados [13]. Podemos apreciar un ejemplo sencillo de una red neuronal en la figura 4.

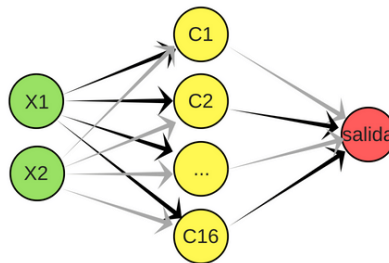


Figura 4. Ejemplo de una red neuronal

2.3.6. SVM

Las SVM o Maquinas de Vectores Soporte, permiten encontrar la forma óptima de clasificar entre varias clases. La clasificación óptima se realiza maximizando el margen de separación entre las clases. Los vectores que definen el borde de esta separación son los vectores de soporte. En ocasiones, las clases no son linealmente separables; es entonces cuando ayudándonos con el kernel, añadimos una nueva dimensión en donde las clases son ya separables. En resumen, lo que hacen las SVM es generar un hiperplano que sea capaz de separar ambas clases maximizando el margen del elemento más cercano de cada clase [14]. En la figura 5 podemos apreciar un ejemplo muy simple de lo que es una SVM.

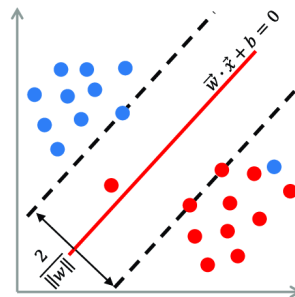


Figura 5. Ejemplo de una red neuronal

2.3.7. Random Forest

El Random Forest o Bosques Aleatorios, es un conjunto (ensemble) de árboles de decisión combinados con bagging. Al hacer uso del bagging, lo que estamos haciendo es que distintos árboles vean distintas porciones de los datos. De este modo, cada árbol es entrenado con distintas muestras de datos dentro de un mismo problema. El resultado de esto es que, a la hora de combinar los resultados, los errores que generan unos árboles se ven compensados por otros y viceversa, por lo que obtenemos una predicción que generaliza mucho mejor. Al final, lo que estamos haciendo es predecir la muestra con la clase mayoritaria de los resultados obtenidos con cada árbol de decisión [15]. En la figura 6, se muestra un ejemplo de lo que sería un random forest.

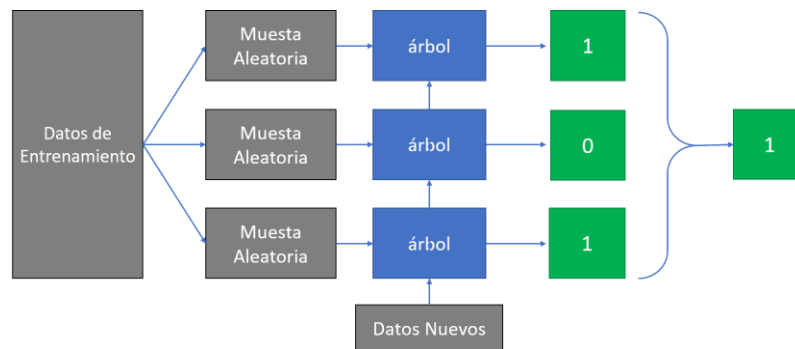


Figura 6. Ejemplo de un random forest

2.4. Entorno de programación

Para el desarrollo del proyecto hemos utilizado distintas herramientas y lenguajes de programación. En este capítulo hablaremos de todas ellas con el fin de que se pueda ejecutar en su totalidad desde cualquier equipo.

2.4.1. Lenguajes de programación

Para la implementación tanto del programa principal como de la interfaz gráfica, usaremos Python (versión 3.7.6) [16], mientras que para la creación e implementación de la base de datos haremos uso del lenguaje SQL [17]. Además, en la interfaz gráfica mezclaremos Python con SQL para hacer las inserciones y consultas.

2.4.2. Herramientas de desarrollo

A la hora de escribir el código del programa principal, se ha programado dentro del entorno informático *Jupyter Notebook* [18], ya que es de gran utilidad para ejecutar y visualizar el resultado celda por celda sin necesidad de tener que ejecutar todo.

Sin embargo, también se han realizado pruebas con el entorno web *Google Colab* [19], ya que, para la fase final, era necesario hacer uso de más memoria por la profundidad de la red de prueba.

Un IDE de multilenguaje con el que poder manejar las partes del programa que van aparte del principal en Python y SQL. Para ello hemos escogido *Visual Studio Code* [20], pero cualquiera nos serviría.

Una herramienta para la visualización y edición de bases de datos. Para ello, al haber utilizado la librería de SQLite3 [21] en Python, hemos usado *DB Browser for SQLite* [22].

2.4.3. Requisitos mínimos

El proyecto ha sido desarrollado en su totalidad bajo el mismo equipo con los siguientes requisitos del sistema:

Sistema Operativo	Windows 10 Home de 64 bits
RAM	16 GB
Espacio Libre Disponible	> 900 GB
Procesador	Intel I7 – 10510-U de hasta 4.90 GHz

Sin embargo, los requisitos mínimos exigidos por cada programa utilizado son los siguientes:

Anaconda Individual Edition

Sistema Operativo	Windows 8, macOS 10.13, Linux
RAM	4 GB
Espacio Libre Disponible	5 GB
Procesador	1.8 GHz

Visual Studio Code 2019

Sistema Operativo	Windows 7
RAM	2 GB (2.5 GB desde Máquina Virtual)
Espacio Libre Disponible	50 GB
Procesador	1.8 GHz

DB Browser for SQLite

Sistema Operativo	Windows 8, macOS 10.13, Linux
RAM	500 MB
Espacio Libre Disponible	100 MB
Procesador	1.8 GHz

Pese a no necesitar más que los requisitos mínimos aquí especificados, se recomienda tener una RAM de al menos 8 GB y unas 100 GB de espacio libre disponible para poder ejecutar sin fallos en memoria el programa correctamente. Además, se ha de tener en cuenta que vamos a trabajar con muchas imágenes.

2.4.4. Librerías

Para la correcta ejecución del programa, debemos importar las siguientes librerías:

- **Numpy**: Librería para crear vectores y matrices grandes, así como para operar con funciones matemáticas de alto nivel.
- **Argparse**: Librería para analizar argumentos enviados al programa.
- **CV2**: Librería para el manejo de funciones relacionadas con visión por computación.
- **Pandas**: Librería utilizada para la manipulación y análisis de datos.
- **Shutil**: Librería para el manejo de ficheros.
- **OS**: Librería utilizada para manejar las rutas.
- **Errno**: Librería para manejar los errores.
- **Matplotlib**: Librería para la visualización de figuras y gráficos a partir de datos.
- **CSV**: Librería para el manejo de ficheros con extensión .csv.
- **Seaborn**: Librería para la visualización de datos.
- **Warnings**: Librería para el manejo de Warnings.
- **Time**: Librería para el manejo del tiempo de ejecución.
- **Sklearn**: Librería para aprendizaje automático.
- **Scipy**: Librería para algoritmos matemáticos.
- **Math**: Librería para el control de operaciones matemáticas.
- **PIL**: Librería para el manejo de imágenes.
- **Xgboost**: Librería para la implementación del algoritmo Extreme Gradient Boosting.
- **Shap**: Librería para la interpretación de la salida de cualquier modelo de aprendizaje automático.
- **Eli5**: Librería para la interpretación de la salida de cualquier modelo de aprendizaje automático.
- **Colorsys**: Librería para la conversión de colores en el espacio RGB.
- **Timeit**: Librería para la medición del tiempo de ejecución.
- **Sqlite3**: Librería para el manejo de funciones de SQLite.
- **Tkinter**: Librería para la implementación de Interfaces Gráficas en Python.
- **Tensorflow**: Librería para el manejo de creación de redes convolucionales.

A estas librerías debemos de sumar las creadas para almacenar funciones.

Capítulo 3

Desarrollo del proyecto

En este capítulo, inspeccionaremos los distintos datasets que analizamos además de mostrar algunas muestras de ellos.

3.1. Datasets candidatos

Pese a que las primeras líneas de código del proyecto fueron escritas al principio del semestre académico, mucho antes de ello, tuve que buscar y leer bastante acerca del problema que quería abordar. Además, era necesario encontrar un conjunto equilibrado de imágenes (para tener el problema balanceado) con muestras tanto de personas sanas como de personas que padecen la enfermedad. Esto último fue complicado de encontrar, ya que hay pocos datasets de imágenes en internet que dispongan de todo lo que buscábamos. Muchos de ellos tan solo tenían imágenes de una de las dos clases, mientras que otros, poseían imágenes de muy mala calidad o de poca cantidad. Esto se debe a que las imágenes que buscábamos normalmente suelen quedarse en el ámbito privado de la institución que las haga, ya que es muy costoso (por las herramientas y métodos científicos que se usan para ello) obtenerlas. Afortunadamente, pudimos contactar con Fabio Scotti, investigador del departamento de ciencia de datos de la Universidad de Milán que había estado trabajando para obtener un conjunto de imágenes de calidad con el que poder investigar acerca de esta enfermedad. El dataset que había desarrollado no era público, por lo que tuvimos que escribirle para poder obtener tanto las imágenes como su permiso. A continuación, mostraré cuales fueron los datasets candidatos en orden no cronológico.

3.1.1. Dataset 1: LISC

El primero de los datasets se llama *LISC: Leukocyte Images for Segmentation and Classification* [23]. Está formado por imágenes de distintos tipos de leucocitos. Está formado por 205 imágenes en las cuales tenemos entre 1 y 3 leucocitos de la misma clase. Un ejemplo de las imágenes de este dataset se puede ver en la Figura 7.

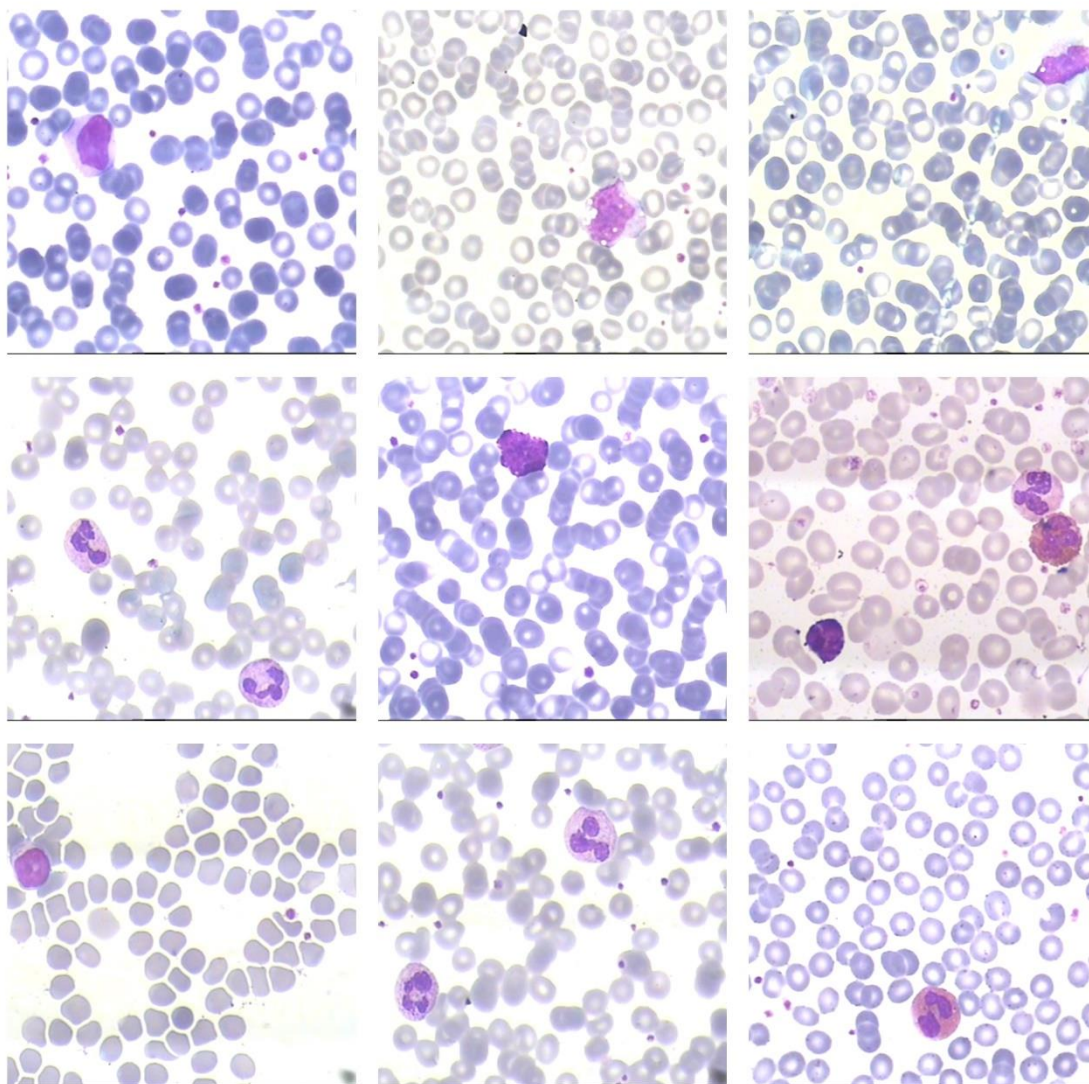


Figura 7. Muestra de algunas de las imágenes del Dataset 1

Obviamente, descartamos este dataset por poseer solo imágenes de Leucocitos en sangre sin especificar si eran de personas enfermas o no.

3.1.2. Dataset 2: SN-AM

El segundo de los datasets se llama *SN-AM Dataset: White Blood cancer dataset of B-ALL and MM for stain normalization* [24]. contiene 22 imágenes sin tratar obtenidas de la sangre de pacientes con leucemia tipo LLA y 30 imágenes sin tratar provenientes de médulas óseas. Un ejemplo de las imágenes de este dataset se puede ver en la Figura 8, en donde se aprecia como las tres últimas imágenes provienen de médulas óseas.

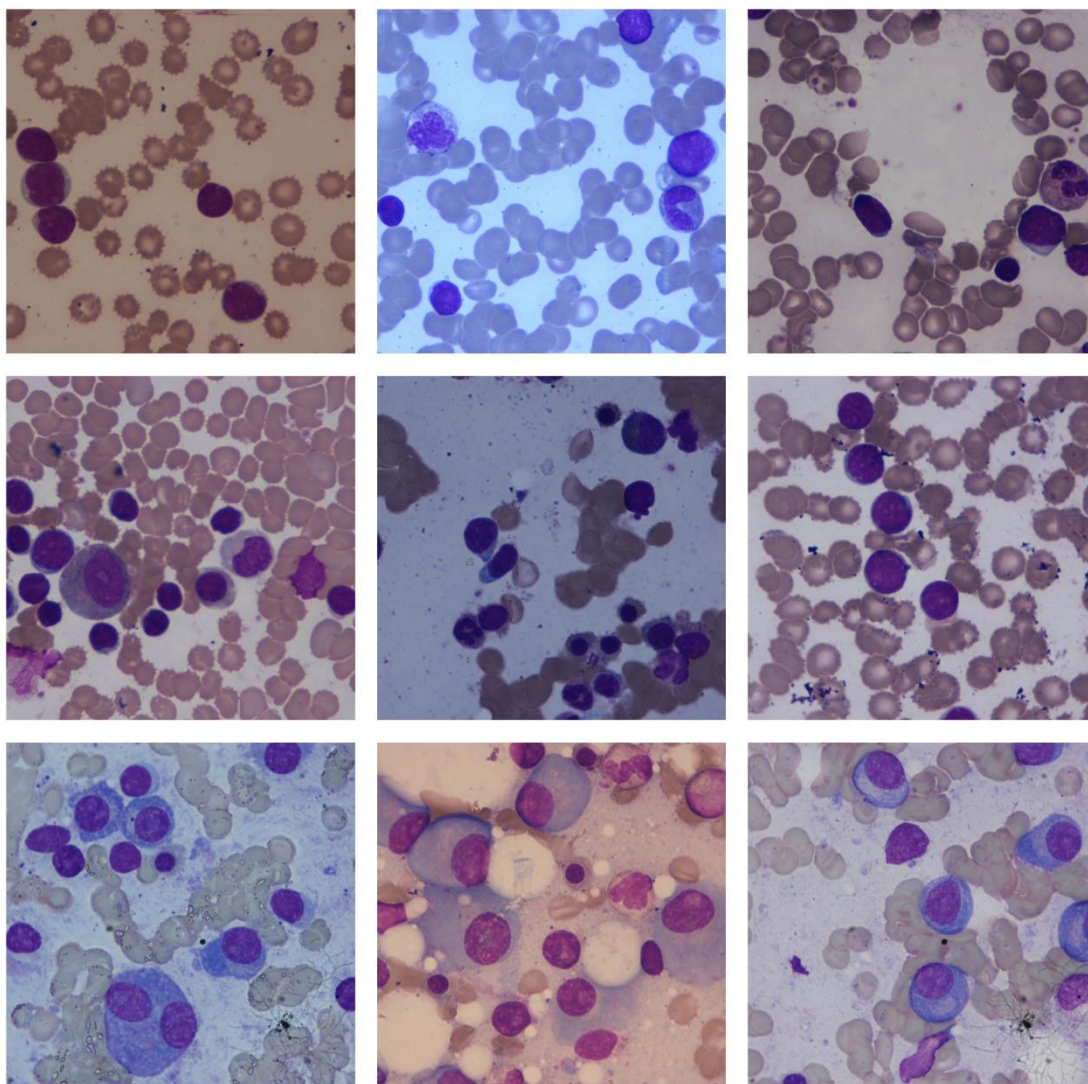


Figura 8. Muestra de algunas de las imágenes del Dataset 2

Este dataset fue descartado por la escasez de imágenes y por la diferencia de resolución de algunas de las imágenes entre sí.

3.1.3. Dataset 3: AML-Cytomorphology_LMU

El tercer dataset, denominado como *AML-Cytomorphology_LMU* [25], separa las muestras en función al tipo de leucocito que aparece en la imagen. A diferencia de los dos anteriores que poseían pocas muestras, este dispone de más de 18.000 imágenes. Sin embargo, no son imágenes completas de una muestra de sangre, sino que se encuentran recortadas para poder clasificar los tipos de leucocitos. Un ejemplo de las imágenes de este dataset se puede ver en la Figura 9.

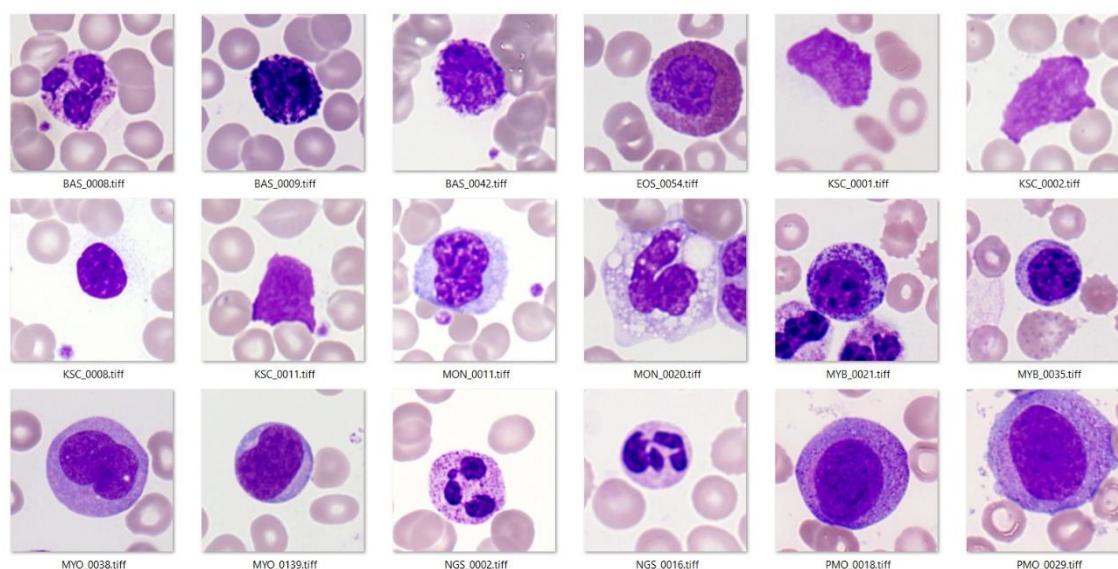


Figura 9. Muestra de algunas de las imágenes del Dataset 3

Este dataset fue descartado por no disponer de imágenes completas. Sin embargo, para líneas futuras, en caso de querer sacar la relación de la enfermedad con los tipos de leucocitos, podría ser de gran utilidad.

3.1.4. Dataset 4: ALL_IDB_Dataset

Finalmente, llegamos al dataset que se escogió como candidato para la realización del proyecto. Este dataset, llamado *ALL_IDB_Dataset* [26], posee un total de 108 imágenes de pacientes sanos y enfermos de leucemia tipo LLA. Cada muestra se encuentra clasificada por su clase en el nombre. Algo que me llamó la atención de este dataset es que se nota como las imágenes han sido obtenidas conjuntamente, es decir, que cada clase proviene de la misma persona, pues los colores de ellas se asemejan en su amplia mayoría e incluso se puede llegar a observar cómo algunas imágenes empiezan en el mismo punto en el que acaban otras. Un ejemplo de las imágenes de este dataset se puede ver en la Figura 10, en donde se observa como las imágenes 1,2 y 9 corresponden al mismo sujeto, mientras que las imágenes 4,5 y 6 son de otra persona. Del mismo modo, se puede ver también como las imágenes 3,7 y 8 son de otro paciente.

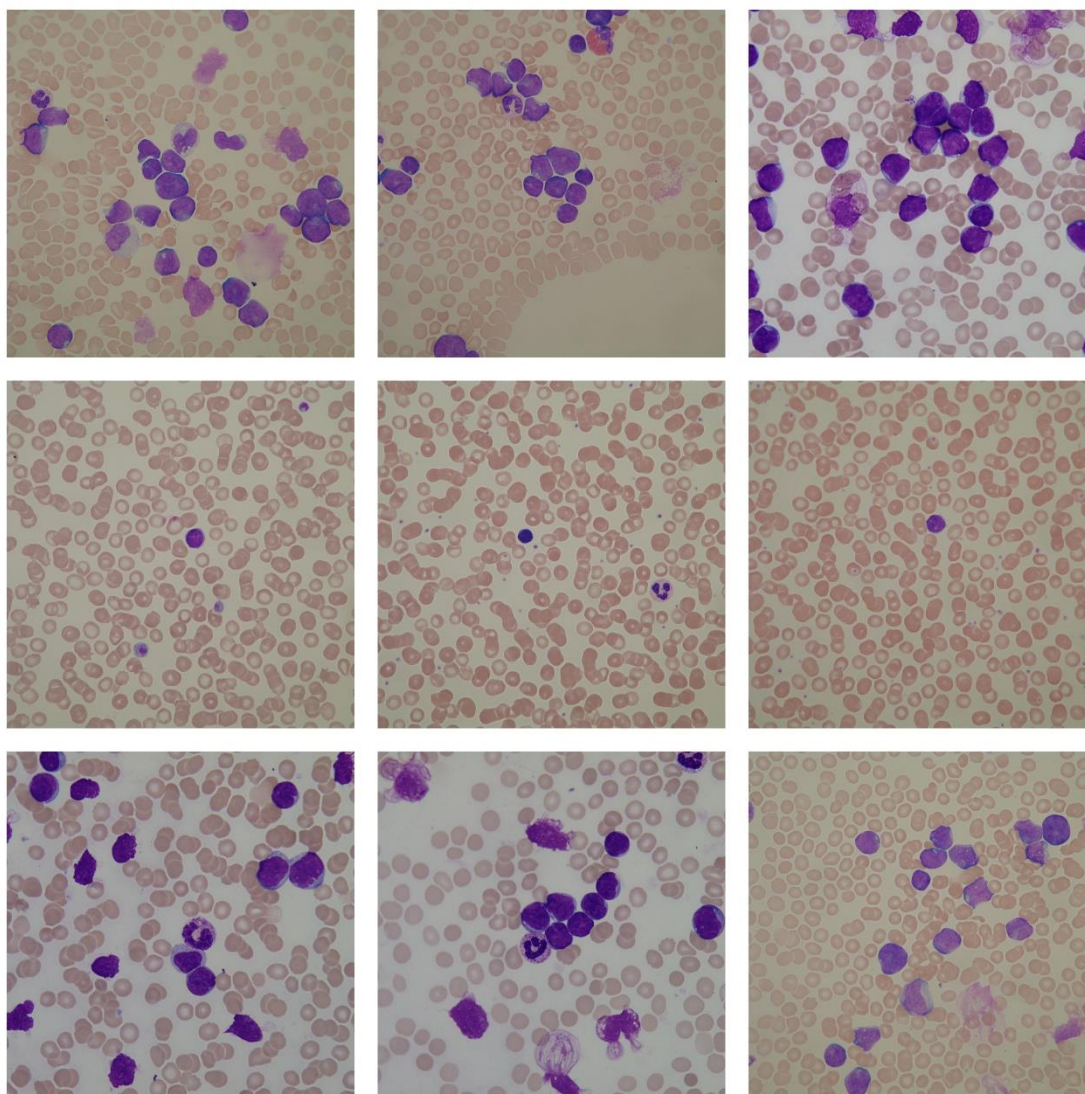


Figura 10. Muestra de algunas de las imágenes del Dataset 4

Capítulo 4

Implementación

En este capítulo, entraremos a explicar el cómo y por qué de cada una de las fases de nuestro código. Aunque será explicado todo como si lo hubiéramos obtenido a la primera, hay que destacar que para nada ha sido así. Cada uno de los pasos conseguidos han tenido por detrás muchos otros intentos e incluso ideas que han sido desechadas.

4.1. Fase 1: Preprocesamiento de imágenes

En la primera fase partimos solamente del dataset conocido como *ALL_IDB_Dataset*, sobre el cual vamos a intentar aplicar técnicas de visión por computación con el fin de poder segmentar los distintos objetos (glóbulos rojos, leucocitos y fondo) de la imagen.

Para ello, empezaremos aplicando el algoritmo de K-Means con $K=3$ sobre todas las muestras del dataset.

4.1.1. Algoritmo de K-Means

El algoritmo de K-Means es un algoritmo de clasificación no supervisada que se encuentra dentro del clustering. Este algoritmo agrupa objetos en K grupos basándose en sus características. El agrupamiento es óptimo cuando la suma de las distancias de cada objeto dentro del mismo grupo respecto a su centroide es mínima. Para ello, usamos la distancia cuadrática. El algoritmo consta de los siguientes pasos:

- **Inicialización:** Una vez escogido el valor de K , establecemos aleatoriamente K centroides dentro del espacio de datos.
- **Asignación a los grupos:** Cada objeto es asignado dentro del grupo del centroide más cercano.
- **Actualización de los centroides:** Se actualizan las posiciones de los centroides tomando como nuevo centroide la posición promedio de los objetos del grupo.

Se repiten los pasos 2 y 3 hasta que los centroides se muevan por debajo de una distancia umbral [27].

En nuestro caso, tal y como hemos dicho antes, aplicaremos este algoritmo con $K=3$, ya que queremos que nos distinga la imagen en tantas clases como objetos queremos diferenciar. Para ello, cada pixel de la imagen será tomado como un objeto a clasificar en base a su intensidad dentro de los tres espectros de color RGB. Tras finalizar el algoritmo, lo que tendremos será una máscara de etiquetas del mismo tamaño que la imagen en donde cada pixel tendrá un valor entero entre 0 y 2, ambos inclusive. Este valor será el que determine dentro de que clase se encuentra el pixel y, por tanto, lo coloreará con la intensidad media de los pixeles de dicha clase.

De esta manera, obtendremos unas imágenes como las que se muestran en la figura 11:

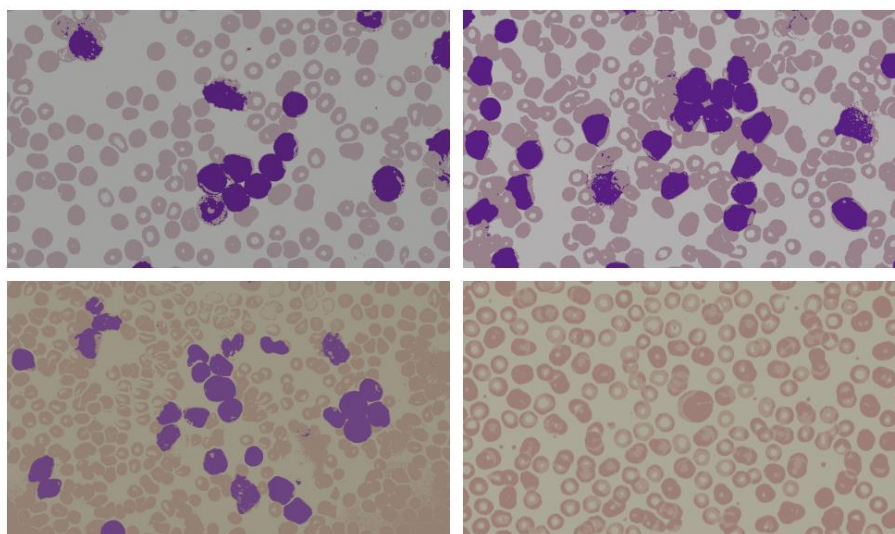


Figura 11. Muestra de algunas de las imágenes tras aplicar K-Means con $K=3$

Tal y como podemos apreciar en la figura, en la imagen de abajo a la derecha, al tratarse de una muestra de un paciente sano y por tanto aparecer un único leucocito (se puede ver en el medio), el algoritmo considera que las tonalidades moradas (relacionadas con los leucocitos) no se encuentran dentro de las tres primeras clases y, por tanto, decide que las clases predominantes son el fondo, los glóbulos rojos y el sombreado de los glóbulos rojos. Esto era de esperar, ya que, al ser un paciente sano, no solo tiene menos leucocitos, sino que, además, hace que pueda generar más glóbulos rojos, influyendo esto en el algoritmo. Por otro lado, podemos observar también como dependiendo de donde provenga la muestra y como haya sido tintada en el laboratorio, tendremos unas intensidades medias u otras para las mismas clases.

Una vez analizado esto, vamos a intentar encontrar alguna solución para ambos problemas (pese a que el primero, tal y como veremos más adelante, más que un problema es una ayuda).

Para intentar que en la imagen resultante del algoritmo K-Means podamos tener siempre los leucocitos independientemente de si la persona está sana o no, probaremos a aplicar K-Means con $K=4$. En la figura 12 se muestran los resultados obtenidos.

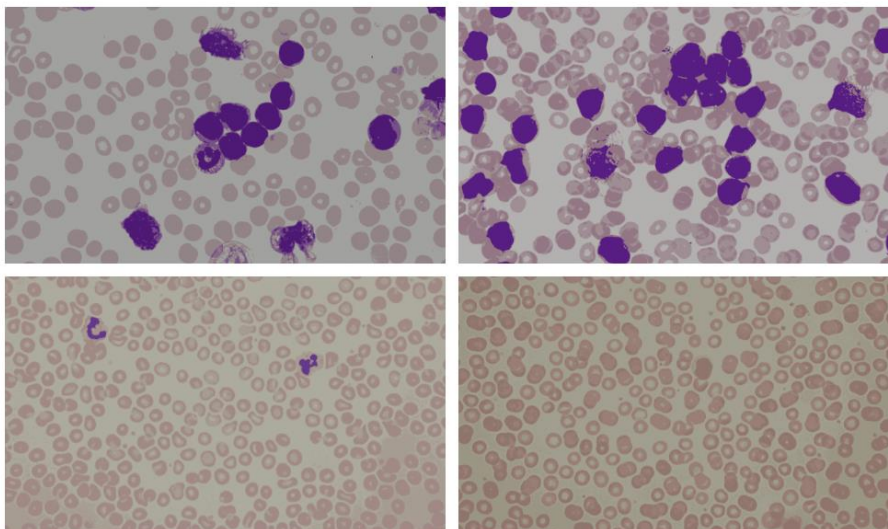


Figura 12. Muestra de algunas de las imágenes tras aplicar K-Means con $K=4$

Tal y como se aprecia, en las imágenes que ya conseguíamos distinguir bien las tres clases lo que nos hace es crear una nueva o bien para el sombreado de los leucocitos o bien para el sombreado de los glóbulos rojos. Sin embargo, en las clases de pacientes sanos, no conseguimos distinguir nuevamente los leucocitos, o al menos su totalidad, ya que hay unas pocas muestras en donde se llega a resaltar los núcleos de los leucocitos tal y como podemos llegar a observar en la tercera imagen de la figura. Pese a ello, no vamos a seguir subiendo el valor de K , ya que no vamos a hacer más que generar intensidades innecesarias que luego nos van a dificultar el trabajo. Es por ello, que descartaremos usar K-Means con $K=4$ y seguiremos aplicando $K=3$.

Otra de las opciones que tenemos para poder mostrar los leucocitos sobre la imagen obtenida por el algoritmo de K-Means con $K=3$, y que como mucho nos generaría 4 intensidades distintas, es aplicar una máscara sobre estas imágenes ya tratadas en donde solamente tengamos a color los píxeles pertenecientes a los leucocitos. Para ello, vamos a generar una máscara gracias al espacio de color HSV en donde eliminaremos de toda imagen original los píxeles que se encuentren dentro de un rango de colores que consideramos que engloba únicamente a los leucocitos (las tonalidades moradas). Tras ello, colorearemos estos píxeles con una única intensidad de morado y eliminaremos cualquier píxel a su alrededor que no se encuentre dentro de esos valores en el espectro RGB. Una vez hecho esto, podremos sobreponer la máscara encima de las imágenes de

K-Means con $K=3$. En la figura 13 podemos ver los pasos que hemos hecho hasta llegar al efecto deseado.

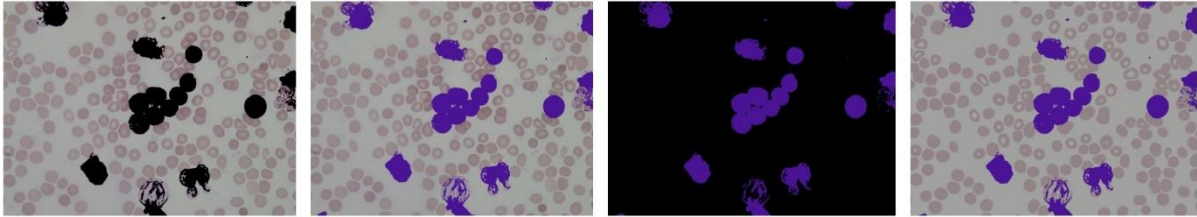


Figura 13. Muestra de los pasos para obtener la imagen

En la figura 14 podemos comprobar que hemos conseguido que todas las imágenes tengan la clase leucocito segmentada.

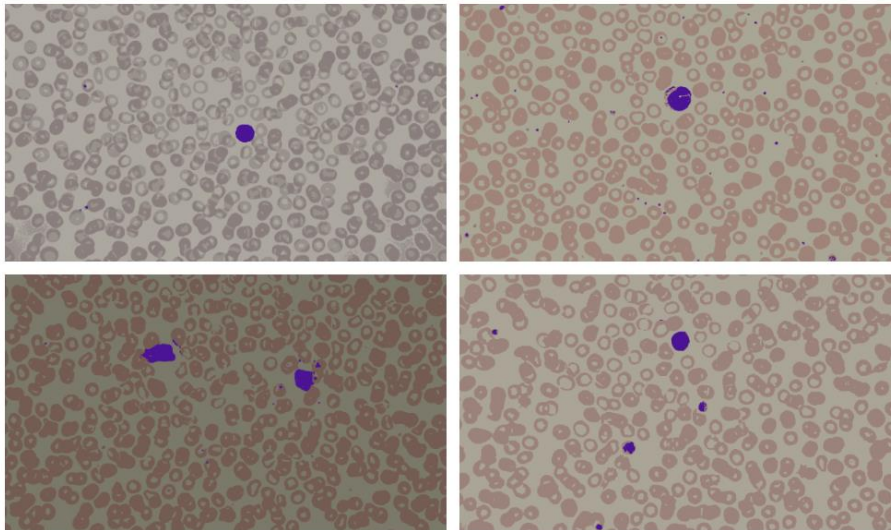


Figura 14. Muestra de las imágenes de pacientes sanos tras aplicar la mascara

Una vez hecho esto, nos encontramos con dos cosas nuevamente. La primera, que tal y como hemos dicho antes, ahora tendremos imágenes con 3 y 4 tonalidades distintas, en donde esta vez sí que distinguimos entre fondo, leucocitos y glóbulos rojos, además de sombreado de glóbulos rojos en las imágenes en donde detectemos 4 tonalidades, por lo que tendremos que tratar de la misma manera tanto glóbulos rojos como su sombreado al ser parte del mismo objeto. Por otro lado, vemos como seguimos teniendo distintas tonalidades para los mismos objetos, es decir que, dependiendo de la procedencia de la muestra, su tinte y la cantidad de objetos que tenga, tendremos las mismas clases coloreadas de unas tonalidades u otras. Como queremos generalizar nuestro algoritmo de modo que sea válido para cualquier imagen tenga el color que tenga, vamos a intentar normalizar las tonalidades de cada clase, de modo que todas las muestras posean las mismas y podamos tratarlas de una manera mucho más fácil.

Para ello, vamos a probar a aplicar las intensidades obtenidas con el K-Means de la primera imagen a todas las demás. Lo que haremos será calcular como ya hemos dicho la primera imagen con el algoritmo de K-Means con $K=3$. Después de esto, en lugar de seguir con el K-Means, lo que vamos a hacer va a ser calcular la distancia de cada pixel de las demás imágenes respecto a estas tres intensidades, de modo que la intensidad con la que tengan una menor distancia sea la que se aplique sobre dicho pixel. Los resultados obtenidos pueden ser apreciados en la figura 15.

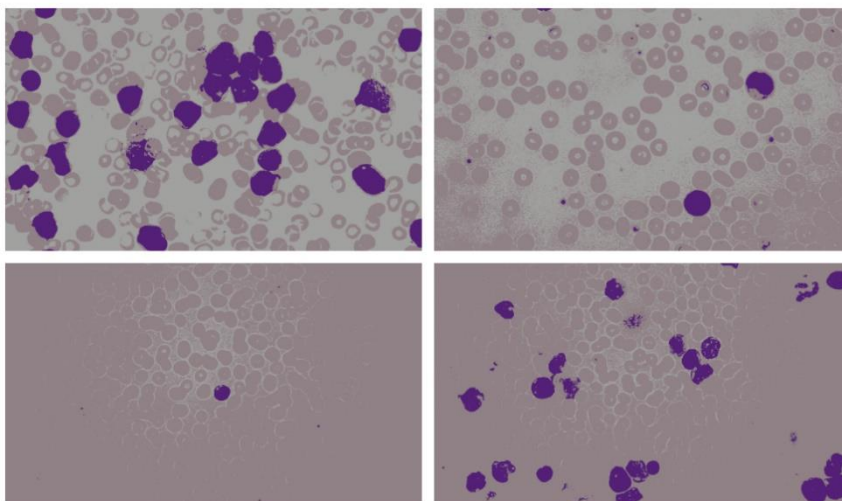


Figura 15. Muestra de las imágenes de la primera prueba de normalización

Podemos apreciar como para las imágenes provenientes del mismo paciente, en este caso, al de arriba a la izquierda, funciona bien. Sin embargo, cuando los colores son muy distintos entre las imágenes, como es en el caso de las dos de abajo, puede ocurrir que el fondo tenga mayor similitud con las intensidades de los glóbulos rojos o viceversa, por lo que acabamos teniendo imágenes como las que se pueden apreciar en la figura. Obviamente, esto no nos va a servir de mucho pese a que cambiemos las intensidades de referencia, ya que siempre va a haber muestras completamente distintas en cuanto a tonalidades, por lo que vamos a tener este problema siempre. Es por ello, que vamos a tener que buscar otro método para normalizar las imágenes.

Otra opción sería intentar “jugar” sobre las imágenes obtenidas del K-Means que tienen la máscara aplicada. Para ello, lo que vamos a hacer va a ser que dependiendo de la suma de sus intensidades en el espectro RGB obtengan un color u otro.

Si tenemos en cuenta que el color negro, al ser la ausencia de color, en RGB se representa como $(0, 0, 0)$ y que el blanco, por ser justo lo contrario, se representa como $(255, 255, 255)$, podemos llegar a la conclusión de que cuanto menor sea la suma de las intensidades en sus tres espectros de color, más oscuro va a ser el color y viceversa.

Teniendo en cuenta que los leucocitos están tintados de morado, los glóbulos rojos y sus sombreados de un rosa oscuro y el fondo de un rosa claro, pese a que las imágenes no comparten las mismas intensidades para los mismos objetos, sí que comparten el mismo orden de suma de sus espectros de color, por lo que podemos aplicar una recoloración en base al orden de estos valores. Como ya hemos dicho antes, tendremos que controlar que, en caso de tener cuatro intensidades, las dos intermedias que serán las que engloban a los glóbulos rojos, reciban el mismo tratamiento. Así pues, tras tratar las muestras de esta manera, obtenemos las imágenes que se pueden apreciar en la figura 16.

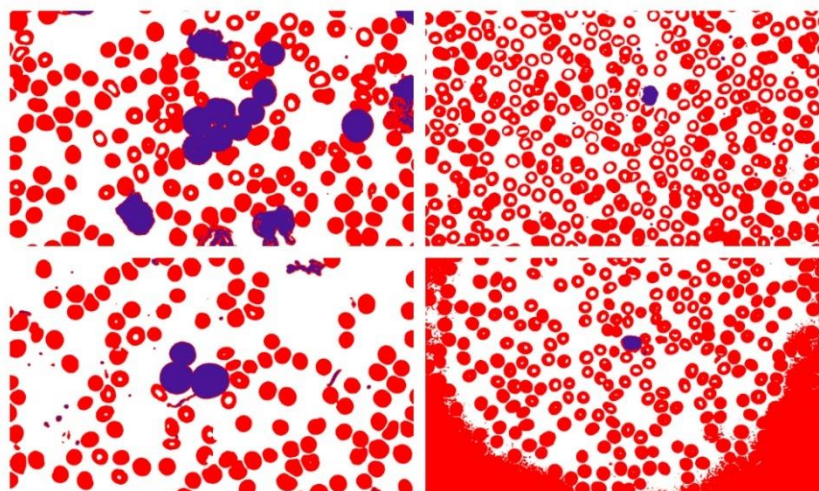


Figura 16. Muestra de las imágenes de la segunda prueba de normalización

Tal y como podemos observar, generaliza muy bien y consigue hacer una buena segmentación. Sin embargo, tenemos dos problemas. El primero de ellos es que tal y como podemos apreciar en la imagen de arriba a la derecha, los glóbulos rojos normalmente suelen tener brillo en su centro, aumentando esto el valor de la suma de los tres espectros de color y haciendo que se tinte como fondo. Sin embargo, al ser este un problema general que engloba todas las muestras, podemos dejarlo así, ya que del mismo modo que en una imagen va a suceder esto, también ocurrirá con las demás, por lo que al final estaríamos equilibrando el error. Por otro lado, el tratamiento que hemos aplicado para los casos en los que tengamos cuatro intensidades distintas no parece ser del todo correcto, ya que como podemos ver en la imagen de abajo a la derecha, por cómo se hizo el frotis de sangre periférico, tenemos las esquinas un tanto más oscuras, lo cual no se aprecia a simple vista en las imágenes originales si no se pone mucho ojo. Esto ocurre en muy pocas muestras y como ya hemos dicho, proviene por cómo se ha sacado la imagen en el laboratorio, por lo que podríamos solucionarlo o bien teniendo más cuidado a la hora de aplicar el frotis de sangre periférico en el laboratorio, o bien modificando el código y haciendo que tuviera en cuenta cuando esa cuarta intensidad es del fondo y cuando es del glóbulo rojo. El problema que tenemos con esto último es que ya de por sí, el generar cada una de estas imágenes aplicando el algoritmo es muy costoso en cuanto a tiempo, aunque es viable (40 segundos por muestra), por lo que si tuviéramos que ir tratando nuevamente los píxeles, aumentaríamos el tiempo de ejecución demasiado. Es por esto último que, al ser un problema que se puede solucionar antes de que se produzca en el laboratorio, y que además solo se aplica a unas pocas muestras, vamos a dejarlo como está.

Ahora que ya tenemos las imágenes segmentadas en regiones y normalizadas, vamos a intentar aplicar distintas conversiones de color con las que intentar buscar algún espacio de color que pueda generalizar mejor el problema de la segmentación que acabamos de realizar.

En la figura 17 se puede observar la muestra IM001_1.jpg junto con los efectos de la conversión de color aplicando distintos espacios de color y sus variaciones: BGR, Rango dinámico ampliado, YUV, HSV, YCC, HLS, CIELab, CIELuv, Ecuilizada en blanco y negro, Ecuilizada en color, BGR con azul, CIELab con verde, YUV con verde, YCC con rojo, BGR con rojo, CIELuv con rojo.

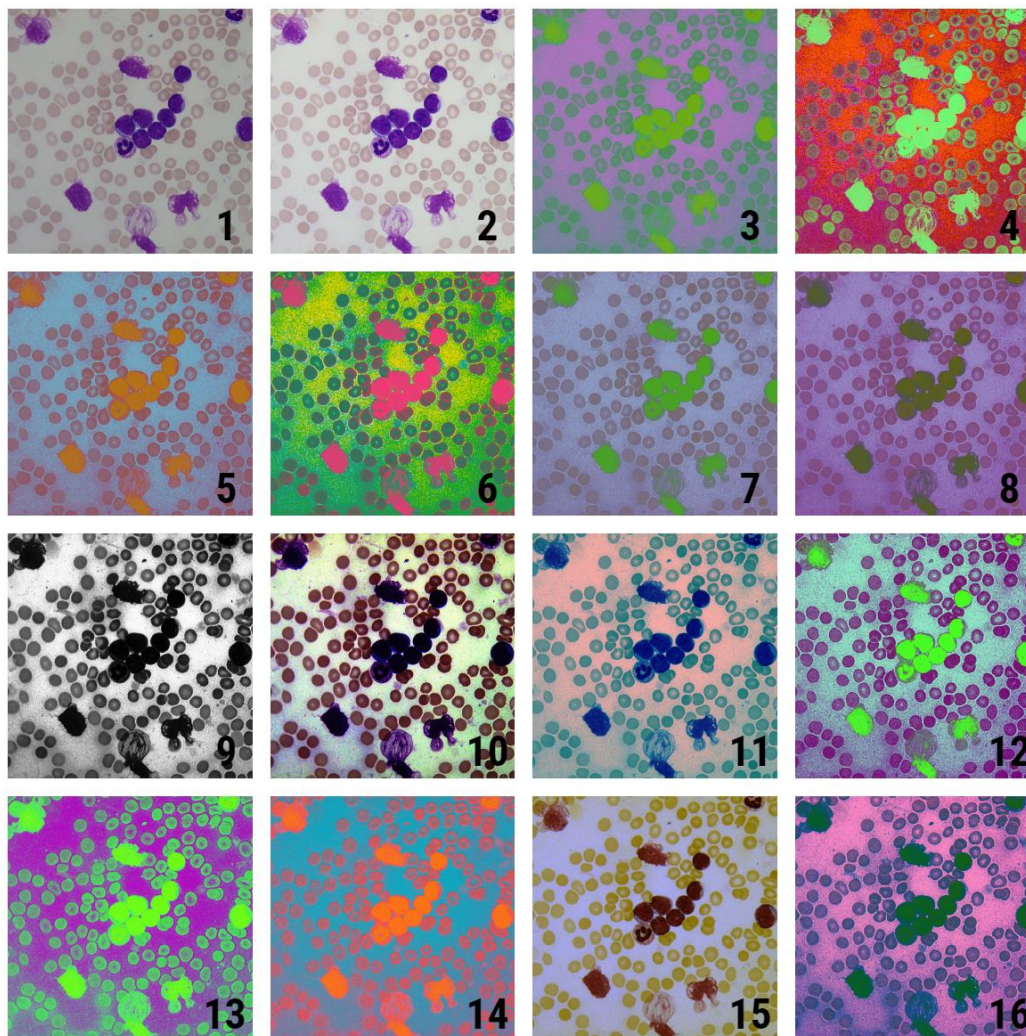


Figura 17. Muestra de la conversión de color de IM001_1.jpg

Si bien es cierto que algunas de las conversiones consiguen hacer que se puedan segmentar visualmente mejor los tres objetos (véase las imágenes 11, 12 y 15 de la figura 17), hay otras que tan solo nos facilitan visualizar mejor dos objetos (véase las imágenes 5, 9, 13 y 14 de la figura 17), ya que tanto glóbulos rojos como blancos acaban teniendo unas tonalidades muy similares entre sí. En cuanto a las que, sí que consiguen lo esperado, si vemos luego el resto de las imágenes del dataset tratadas con dichas conversiones, se aprecia como el tinte que tenga cada muestra inicialmente es de gran importancia, es decir, que en las imágenes de la figura 17 sí que puede parecer que la conversión funciona bien, pero tras aplicarse a imágenes con otras tonalidades, puede que no haga más que dificultar su manejo y llevarnos a ninguna parte. Esto lo podemos ver en la figura 18, en donde se puede ver como la misma conversión de BGR sin rojo funciona distinta para diferentes imágenes de un mismo dataset.

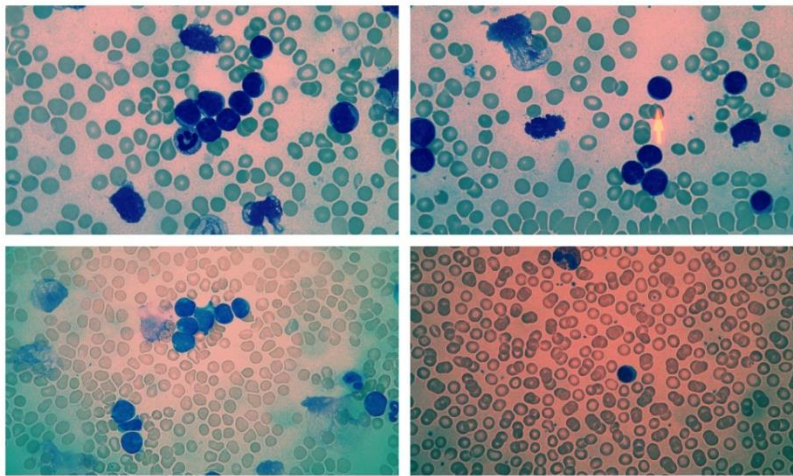


Figura 18. Muestra de la conversión de color BGR sin el espectro de color R

Como ya hemos dicho antes, para las imágenes de arriba de la figura sí que parece funcionar bien, pero cuando vemos las dos de abajo, aparte de notarse aún más el sombreado de la imagen en los bordes, se aprecia como los glóbulos rojos se asemejan mucho al fondo además de que cambia completamente las tonalidades de una a otra. Es por esto por lo que decidimos usar solamente las imágenes con conversiones de color que amplían el rango dinámico (véase la imagen 2 de la figura 17), ya que pensamos que son las únicas que nos pueden ser de utilidad.

4.2. Fase 2: Generación de variables

Una vez hemos tratado ya las muestras de nuestro dataset y obtenido de ellas un nuevo conjunto de imágenes del que sacar información, vamos a generar las variables con las que clasificaremos nuestras muestras.

Para ello, hemos decidido usar el siguiente conjunto de imágenes:

- Dataset original.
- Dataset de K-Means con $K=3$ que tiene la máscara (figura 14).
- Dataset de las imágenes segmentadas en rojo, blanco y morado (figura 16).

Nuestro objetivo en esta fase es intentar obtener el máximo de variables posibles con las que clasificar cada muestra del dataset original.

4.2.1. Variables del Dataset original

De nuestro dataset original obtendremos las siguientes variables:

- **HistMax1:** Es el valor de la intensidad asociada al pico más alto del histograma de la imagen.
- **PixelesMax1:** Se trata de la cantidad de píxeles con la intensidad HistMax1 que tenemos en la imagen.
- **HistMax2:** Se trata del valor del segundo pico más alto del histograma de la imagen. El problema que tenemos en un histograma es que, si calculamos el segundo pico más alto, seguramente nos devuelva una de las intensidades contiguas al primer pico más alto. Para evitar esto, aplicamos una fórmula sobre todas las intensidades de 0 a 255 que pretende que el segundo pico más alto no se encuentre dentro del mismo montículo en el que se encuentra el primer pico. Una vez aplicada, escogeremos la intensidad que maximice el valor de dicha fórmula.
- **PixelesMax2:** Se trata de la cantidad de píxeles con la intensidad HistMax2 que tenemos en la imagen.
- **DiferenciaPicos:** Es la diferencia en valor absoluto entre las intensidades de ambos picos.
- **Leucocitos:** Es la cantidad de leucocitos que tiene cada muestra. Para poder calcularla, generamos una máscara haciendo uso del espacio de color HSV en donde tintamos de negro todo pixel que se encuentre fuera de un rango determinado de intensidades ((60, 150), (60, 255), (0, 255)). De esta manera, conseguimos tener una imagen solo con los leucocitos. Tras esto, aplicamos una Transformada de Hough con la que, tras ajustar los parámetros necesarios, conseguimos identificar cada una de las formas circulares (leucocitos) de la máscara.
- **Porcentaje:** Es el porcentaje de píxeles no negros (leucocitos) de la máscara

Para la variable leucocitos, utilizamos los siguientes parámetros en nuestra Transformada de Hough:

- **minDist:** Es la distancia mínima entre los centroides de los leucocitos. La hemos marcado en 95 píxeles.
- **param1:** Es el valor que marca el límite de detección de borde de Canny. Lo hemos marcado en 70.
- **param2:** Es el valor del umbral límite de espacio Hough. Nosotros lo tenemos como 20.
- **minRadius:** Como su nombre indica, es el radio mínimo que puede detectar. En nuestro caso, es 40.
- **maxRadius:** Al contrario que minRadius, es el radio máximo que puede detectar. Para nosotros será 85.

En la figura 19, podemos observar un histograma obtenido de una de las muestras en donde se aprecian los picos más altos de intensidad de la imagen que tras la maximización generarían los puntos 120 y 13.

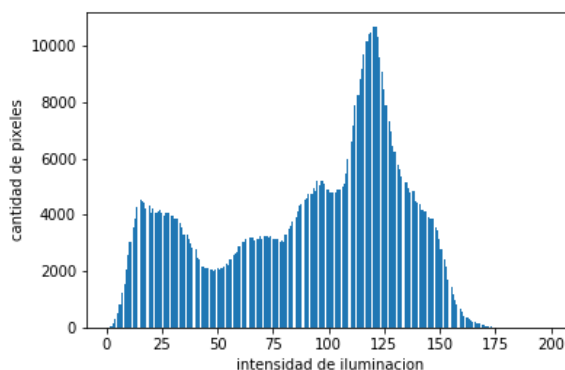


Figura 19. Histograma de la imagen IM001_1.jpg

En la figura 20, tenemos un ejemplo de cómo se obtiene la cantidad de leucocitos de una muestra paso a paso. Además, podemos ver la importancia del porcentaje de píxeles no negros, ya que cuanto mayor sea, más posibilidades de que haya muchos leucocitos y, por tanto, de que el paciente se encuentre dentro de la clase “Leucemia”.

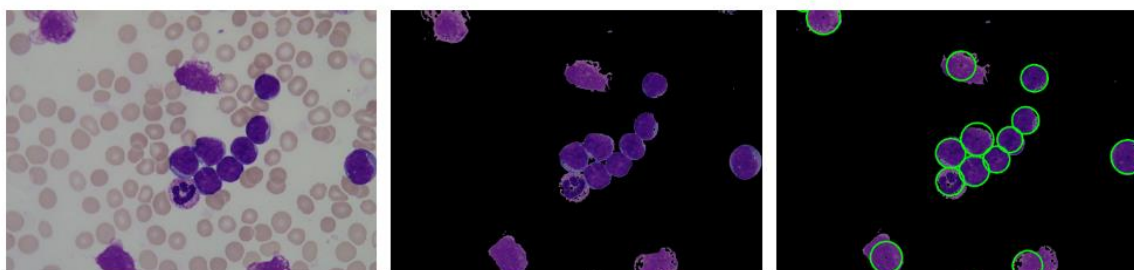


Figura 20. Detección de leucocitos en la IM001_1.jpg

4.2.2. Variables del Dataset en rojo, blanco y morado

Para este segundo dataset, intentaremos aprovechar que tenemos solamente tres tonalidades normalizadas en todas las muestras para sacar la cantidad de píxeles y porcentajes de cada uno de los tres grupos. Estas son nuestras variables:

- **Blancos:** Es la cantidad de píxeles que conforman el fondo.
- **Rojos:** Es la cantidad de píxeles que conforman los glóbulos rojos.
- **Morados:** Es la cantidad de píxeles que conforman los leucocitos.
- **Porcentaje_Blancos:** Es el porcentaje de píxeles blancos.
- **Porcentaje_Rojos:** Es el porcentaje de píxeles rojos.
- **Porcentaje_Morados:** Es el porcentaje de píxeles morados.

4.2.3. Variables del Dataset de K-Means

En el caso de este último dataset, obtendremos las siguientes variables:

- **MoradosEnCluster:** Al igual que hacíamos anteriormente con la variable de porcentaje, en esta lo que hacemos es aplicar la misma metodología sobre la imagen tratada con el algoritmo de K-Means.
- **HistSegMax1:** Es el valor del pico más alto del histograma de la imagen segmentada.
- **PixelesSegMax1:** Se trata de la cantidad de píxeles con la intensidad HistSegMax1 que tenemos en la imagen segmentada.
- **HistSegMax2:** Es el valor del segundo pico más alto del histograma de la imagen segmentada. Aplicaremos la misma metodología que hemos expuesto anteriormente para encontrar el segundo pico más alto.
- **PixelesSegMax2:** Se trata de la cantidad de píxeles con la intensidad HistSegMax2 que tenemos en la imagen segmentada.
- **HistSegMax3:** Es el valor del tercer pico más alto del histograma de la imagen segmentada. Aplicaremos la misma metodología que hemos expuesto anteriormente para encontrar el segundo pico más alto con la diferencia de que esta vez trataremos de que este pico se encuentre fuera de los montículos anteriores.
- **PixelesSegMax3:** Se trata de la cantidad de píxeles con la intensidad HistSegMax3 que tenemos en la imagen segmentada.
- **DiferenciaPicosSeg:** Es la diferencia en valor absoluto entre los picos más altos.

Finalmente, tras la realización de muchas pruebas, se decidió que para el ejemplo final se haría uso de las siguientes 17 variables, que serán las que utilizemos en el código y en las explicaciones que vendrán a continuación:

- ✓ **HistMax1**
- ✓ **PixelesMax1**
- ✓ **HistMax2**
- ✓ **PixelesMax2**
- ✓ **DiferenciaPicos**
- ✓ **Leucocitos**
- ✓ **Porcentaje**
- ✓ **Blancos**
- ✓ **Rojos**
- ✓ **Morados**
- ✓ **Porcentaje_Blancos**
- ✓ **Porcentaje_Rojos**
- ✓ **Porcentaje_Morados**
- ✓ **MoradosEnCluster**
- ✓ **PixelesSegMax1**
- ✓ **PixelesSegMax2**
- ✓ **PixelesSegMax3**

4.3. Fase 3: Clasificación

Nuestro programa principal se encarga de obtener el valor de cada variable para todas las muestras y genera un archivo de extensión CSV en donde las almacena automáticamente junto con el valor de su clase (0 si el paciente está sano y 1 si sufre de leucemia). Un ejemplo de esto se puede ver en la figura 21.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Porcentaje	Leucr	HistMax1	PixelesN	HistN	PixelesI	Difer	PixelesSeg1	PixelesSeg2	PixelesSeg3	Mor.	Blancos	Rojos	Morados	Porcentaje	Porcentaje	Porcentaje	Clase
2																		
3	9.3765378	17	161	134404	132	44482	29	1092337	67435	1182244	0	1321115	805500	215401	56.409	34.393	9.197	1
4																		
5	11.0610664	20	162	107952	129	50538	33	902332	82186	1357498	0	1171423	915775	254818	50.018	39.102	10.88	1
6																		
7	7.48329759	15	159	124085	128	45600	31	1074741	71827	1195448	0	1311873	857851	172292	56.015	36.629	7.357	1
8																		
9	6.63513541	13	153	114747	124	55502	29	984379	77490	1280147	0	1238402	951176	152438	52.878	40.614	6.509	1
10																		
11	16.6663706	36	177	82573	130	41710	47	764546	271245	1306225	0	1003517	955278	383221	42.848	40.789	16.363	1
12																		
13	13.0098999	27	164	91257	127	47496	37	807120	204779	1330117	0	1070011	972877	299128	45.688	41.54	12.772	1
14																		
15	6.23399019	13	154	151685	125	39694	29	1352264	51405	938347	0	1532862	665348	143806	65.451	28.409	6.14	1

Figura 21. Muestra del archivo CSV

Tras haber generado y almacenado las suficientes variables, empezamos la fase de clasificación, en donde trataremos no solo de obtener el mejor porcentaje de acierto posible, sino de utilizar la selección de variables en donde estudiemos el impacto de cada una de ellas en nuestro problema. Es por esto por lo que esta fase se dividirá en distintos apartados.

4.3.1. Clasificación manual

Una vez tenemos operativo nuestro archivo CSV, lo primero que vamos a hacer es una clasificación automática, pero sin hacer uso de ningún tipo de técnica de inteligencia artificial. Para ello, en base al método que utilicemos, generaremos un umbral el cual servirá para clasificar las muestras. Con esto lo que buscamos es ver qué porcentaje de acierto somos capaces de generar sin necesidad de aplicar algoritmos especializados en ello. Los métodos que utilizamos son los siguientes:

1. Umbral como porcentaje más pequeño de las muestras de Leucemia.
2. Umbral como porcentaje más grande de las muestras sanas.
3. Umbral como número de leucocitos más pequeño de las muestras con Leucemia.
4. Umbral como número de leucocitos más grande de las muestras sanas.
5. Umbral como combinación más pequeña entre ambos parámetros para las muestras con Leucemia.
6. Umbral como combinación más grande entre ambos parámetros para las muestras sanas.

En la figura 22 tenemos una muestra de los parámetros que queremos clasificar en base a los porcentajes (métodos 1 y 2).

```

////////////////////////////////////LEUCEMIA:////////////////////////////////////
PORCENTAJES
[ 0.75930357  0.97458959  1.08274221  1.40737891  1.52454376  1.55528188
 1.60220861  1.70208216  2.04443336  2.09627151  2.4902463  2.52402425
 2.56595016  2.68469453  2.91906595  3.78655791  4.50739861  4.72759008
 5.109936  5.19589186  5.51836491  5.53432107  5.81148863  5.89035153
 6.23399019  6.33057356  6.63513541  6.88639283  6.9832027  7.48329759
 7.53902197  8.02398324  8.11342001  8.84616971  8.91571045  9.07487273
 9.3765378  9.76574421  10.10868549  10.20590663  10.27392745  10.28197408
 11.06106639  11.11866832  11.68518662  11.80226803  12.15209365  13.00989985
 16.66637063]
////////////////////////////////////NO LEUCEMIA:////////////////////////////////////
PORCENTAJES
[0.13431907  0.14338493  0.18216372  0.18496513  0.18748045  0.19417405
 0.20659566  0.20737052  0.21495223  0.21830201  0.22062659  0.22386312
 0.23671985  0.25315285  0.26079416  0.27236342  0.28121471  0.29768944
 0.32799244  0.33105016  0.33481717  0.33529401  0.34563541  0.35081506
 0.35377145  0.35726428  0.36792159  0.37252307  0.37496686  0.3832221
 0.38486719  0.38756728  0.39290786  0.4224956  0.43116808  0.43412447
 0.43450594  0.44087172  0.45258403  0.49211383  0.52893162  0.53083301
 0.54278374  0.55165291  0.5599916  0.57092309  0.61069727  0.63036084
 0.64865947  0.70496202  0.72976947  0.78502297  0.84604621  0.84604621
 1.01607442  1.09092593  1.24292374  1.33304  1.42963529]

```

Figura 22. Valor de la variable Porcentajes ordenada para ambas clases

Con el método 1, estaríamos marcando como frontera de decisión el valor del porcentaje más pequeño de las muestras de Leucemia (0.7593). De esta manera, lo que haríamos sería clasificar todas las muestras con el mismo o mayor valor de dicho parámetro dentro de la clase Leucemia, mientras que las demás, irían en la clase No Leucemia. Tal y como podemos observar, estaríamos fallando las ocho últimas muestras de la clase No Leucemia, por lo que tendríamos un porcentaje de acierto del 92.59%.

Lo mismo sucedería para el método 2, en donde marcamos como frontera de decisión el valor del porcentaje más alto de las muestras de No Leucemia (1.4296). Al contrario que en el método anterior, aquí lo que haríamos sería clasificar como Leucemia aquellas muestras cuyo valor fuera superior al umbral y viceversa. En este caso, fallaríamos tan solo las cuatro primeras muestras de la clase Leucemia, por lo que obtendríamos un porcentaje de acierto del 96.29%.

En cuanto a los cuatro métodos restantes, la idea sería la misma que en los dos primeros variando el parámetro con el que se hace la clasificación, pues en los métodos 3 y 4 utilizamos el valor del número de leucocitos detectados, mientras que en los dos últimos métodos hacemos uso de ambas variables al combinarlas multiplicando sus valores entre sí.

Para el ejemplo que vamos a explicar tanto en la memoria como en el código, los resultados obtenidos por los seis métodos son los siguientes:

1. Método 1: 92.59%
2. Método 2: 96.29%
3. Método 3: 91.66%
4. Método 4: 90.74%
5. Método 5: 91.66%
6. Método 6: 90.74%

Tal y como podemos observar, llegamos a unos porcentajes iniciales muy prometedores, ya que tan solo hacemos uso de una y dos variables, lo cual nos hace pensar que tras utilizar las diecisiete variables que hemos decidido usar, llegaremos a unos resultados muy buenos.

Sin embargo, hay que destacar que no es lo mismo fallar cinco muestras de pacientes sanos clasificándolos dentro de la clase Leucemia que fallar cinco de la clase Leucemia dentro de la clase No Leucemia, ya que, en el primero de los casos, nos estaríamos llevando un susto muy desagradable que no deja de ser un susto. Sin embargo, en el segundo caso, posiblemente estemos descuidando a una persona que necesite ser tratada lo antes posible. Con esto quiero decir que no todos los errores son iguales, ya que no todos implican el mismo resultado. De aquí en adelante deberemos tener mucho cuidado con esto.

4.3.2. Clasificación automática

Tras haber hecho una primera aproximación de los resultados gracias a la clasificación manual, empezaremos a manejar distintos clasificadores para lograr una mejor clasificación.

Para ello, dividiremos nuestro conjunto de datos (108 muestras) en dos subconjuntos de train (75 muestras) y de test (33 muestras).

Los clasificadores que vamos a usar son los ya descritos en capítulos anteriores, por lo que no vamos a volver a entrar en detalle en ellos.

Lo primero que haremos será generar los modelos de cada uno de los clasificadores con sus hiper parámetros por defecto y la misma semilla para poder repetir los resultados. Una vez hecho esto, obtenemos los resultados mostrados en la Tabla 1:

Clasificador	Rendimiento en train	Rendimiento en test	FScore
KNN	100.00%	96.96%	95.84%
Arboles de Decisión	100.00%	100.00%	100.00%
Random Forest	100.00%	96.96%	95.84%
Regresión Logística	100.00%	100.00%	100.00%
SVM	84.00%	84.84%	88.88%
Redes Neuronales	58.66%	45.45%	0.00%

Tabla 1. Resultados obtenidos para todos los clasificadores con sus hiper parámetros por defecto

Como podemos observar en este primer acercamiento a los clasificadores, los resultados para los hiper parámetros por defecto están bastante bien salvo en las SVM y las redes neuronales. Además, hemos calculado el FScore esta vez, el cual es la media armónica entre la precisión (fracción de verdaderos positivos predichos entre las muestras predichas como positivas) y el recall (fracción de verdaderos positivos predichos entre los verdaderos positivos) que nos ayuda a saber la calidad de nuestra predicción para poder evitar el problema que explicábamos más arriba. Las fórmulas para ello son las siguientes:

$$\text{Recall} = \frac{\text{True Positives}}{\# \text{ of Actual Positives}} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\# \text{ of Predicted Positives}} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{FScore} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Una vez hecho esto, vamos a probar a hacer una optimización de los hiper parámetros para obtener los mejores resultados posibles para cada clasificador. Para ello, haremos uso de la estrategia denominada como Grid Search, la cual probará todas las combinaciones posibles de los hiper parámetros de cada clasificador y devolverá solo el modelo generado que dé el mejor resultado.

Para esta segunda clasificación, vamos a probar a normalizar nuestros datos, ya que tal y como podemos observar en la figura 17, existe una gran diferencia de rangos entre ellos, la cual podría dificultar la clasificación. Para esto usaremos tres métodos de clasificación distintos: Sin normalización, MinMaxScaler y StandardScaler [28].

- **MinMaxScaler():** Este método de normalización transforma las características escalándolas a un rango dado por defecto entre (0,1) aunque puede personalizarse.
- **StandardScaler():** Este método estandariza los datos eliminando la media y escalándolos de forma que su varianza sea igual a 1.

En cuanto a los hiper parámetros, en las tablas 2,3,4,5,6,7 podemos ver cuales usamos y con qué valores:

- **KNN [29]**

Hiper parámetros	Valores
n_neighbors	1, 3, 5, 7, 9
weights	'uniform', 'distance'
p	1, 2, 1.5, 3

Tabla 2. Hiper parámetros de KNN

- **Árboles de decisión [30]**

Hiper parámetros	Valores
criterion	'gini', 'entropy'
min_samples_split	2, 5, 10
min_samples_leaf	1, 2, 4

Tabla 3. Hiper parámetros de los Árboles de decisión

- **Random Forest [31]**

Hiper parámetros	Valores
n_estimators	10,50,100
max_features	1,'sqrt','log2',None
criterion	'gini','entropy'
max_depth	5,10,None
min_samples_split	2,10,20

Tabla 4. Hiper parámetros del Random Forest

- **Regresión logística** [32]

Hiper parámetros	Valores
C	0.001, 0.01, 0.1, 1, 10, 100, 1000
penalty	l2
solver	lbfgs

Tabla 5. Hiper parámetros de la Regresión Logística

- **SVM** [33]

Hiper parámetros	Valores
C	0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100
gamma	0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100, 'scale', 'auto'

Tabla 6. Hiper parámetros de las SVM

- **Redes neuronales** [34]

Hiper parámetros	Valores
activation	'identity', 'logistic', 'tanh', 'relu'
hidden_layer_sizes	10,50,100,150,200
solver	'lbfgs', 'sgd', 'adam'
learning_rate	'constant', 'invscaling', 'adaptive'
shuffle	True, False

Tabla 7. Hiper parámetros de las Redes Neuronales

Tras la ejecución de las funciones que generan la estrategia de Grid Search, llegamos a los porcentajes de acierto en test que se muestran en la Tabla 8:

Clasificador	Sin normalizar	MinMaxScaler	StandardScaler
KNN	99.09%	99.09%	99.09%
Arboles de Decisión	99.09%	99.09%	99.09%
Random Forest	100.00%	100.00%	100.00%
Regresión Logística	99.09%	99.09%	99.09%
SVM	99.09%	99.09%	100.00%
Redes Neuronales	100.00%	99.09%	99.09%

Tabla 8. Porcentajes de acierto tras el uso del Grid Search

Como se puede apreciar, los porcentajes de acierto en test son muy buenos; tanto, que no bajamos de un 99.09% en ninguno de los casos, ni siquiera sin normalizar. Es por esto por lo que tal y como ya hemos dicho en otras ocasiones, hemos escogido estas variables y estos modelos para mostrar el proyecto, pese a haber hecho con anterioridad otras 24 pruebas que serán expuestas y explicadas en posteriores apartados.

Así pues, tras haber encontrado unos resultados tan prometedores, empezaremos a estudiar el grado de importancia de cada una de las diecisiete variables con el fin de hacer una reducción de éstas y así poder ganar en tiempo de computación.

4.3.3. Selección de variables

Para tener una primera aproximación del número de variables suficientes para realizar nuestra clasificación sin perder información, vamos a calcular el porcentaje de acierto en train y test para toda combinación de números de variables desde uno hasta diecisiete con distintas métricas de calidad como son Chi2 y ANOVA.

ANOVA es una métrica univariable paramétrico el cual ignora las dependencias de las variables. Lo que hace para escoger las mejores variables es crear un ranking en base a su p-valor, el cual se obtiene descomponiendo la varianza [35].

Chi2, en cambio, es un método estadístico que mide el modo en que los valores esperados cercanos son los resultados reales. En este método, las variables escogidas son aleatorias, y tras su uso con muestras también aleatorias, obtenemos una estadística que nos indica cuantos resultados provienen del resultado aleatorio esperado [36].

Al aplicar Chi2, vemos como los porcentajes tanto de train como de test van cambiando desde que utilizamos una sola variable hasta llegar a cinco. A partir de ahí, los resultados se mantienen fijos (98.66% train y 100.00% test). Tenemos que recalcar que para hacer esta prueba hemos usado el clasificador de KNN con la misma partición de los datos que siempre.

En cuando a ANOVA, obtenemos los mismos resultados finales que en Chi2 con la diferencia de que llega a ellos al utilizar tres variables en lugar de cinco.

De esta manera, nos podemos hacer a la idea de primeras de que podemos reducir la cantidad de nuestras variables bastante, ya que conseguimos los mismos resultados con cinco que con diecisiete variables. De todos modos, antes de comprobar la importancia de cada una dentro del grupo de variables, vamos a realizar la misma prueba sin métricas de calidad haciendo un análisis de componentes principales o PCA , lo cual lo que hará será intentar reducir la dimensionalidad del problema perdiendo la menor cantidad posible de información (varianza) [37].

En la figura 23 se puede observar el porcentaje de train (en naranja) y el de test (en azul) desde una variable hasta diecisiete con el PCA.

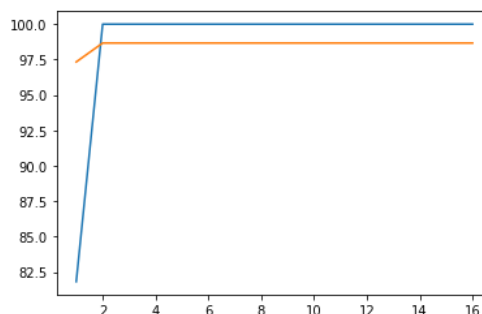


Figura 23. Grafica de rendimiento tras aplicar el PCA de 1 hasta 17 variables

Otra vez llegamos a la misma conclusión de que podemos reducir bastante la cantidad de variables que utilizamos para clasificar.

Ahora sí, vamos a centrarnos en ver cuáles son esas variables que nos resultan redundantes o nos ofrecen poca información. Para ello, empezaremos comparando los valores de cada variable para crear una matriz de correlaciones en donde podremos observar de una manera muy visual la correlación que tienen las variables entre sí. De esta manera, podremos eliminar una variable de cada pareja de variables altamente correlacionadas con un umbral mayor que 0.8 (tanto positiva como negativamente), ya que nos estará dando información redundante o muy pareja a la que puede darnos otra variable. Tras calcular esta matriz, obtenemos la figura 24.

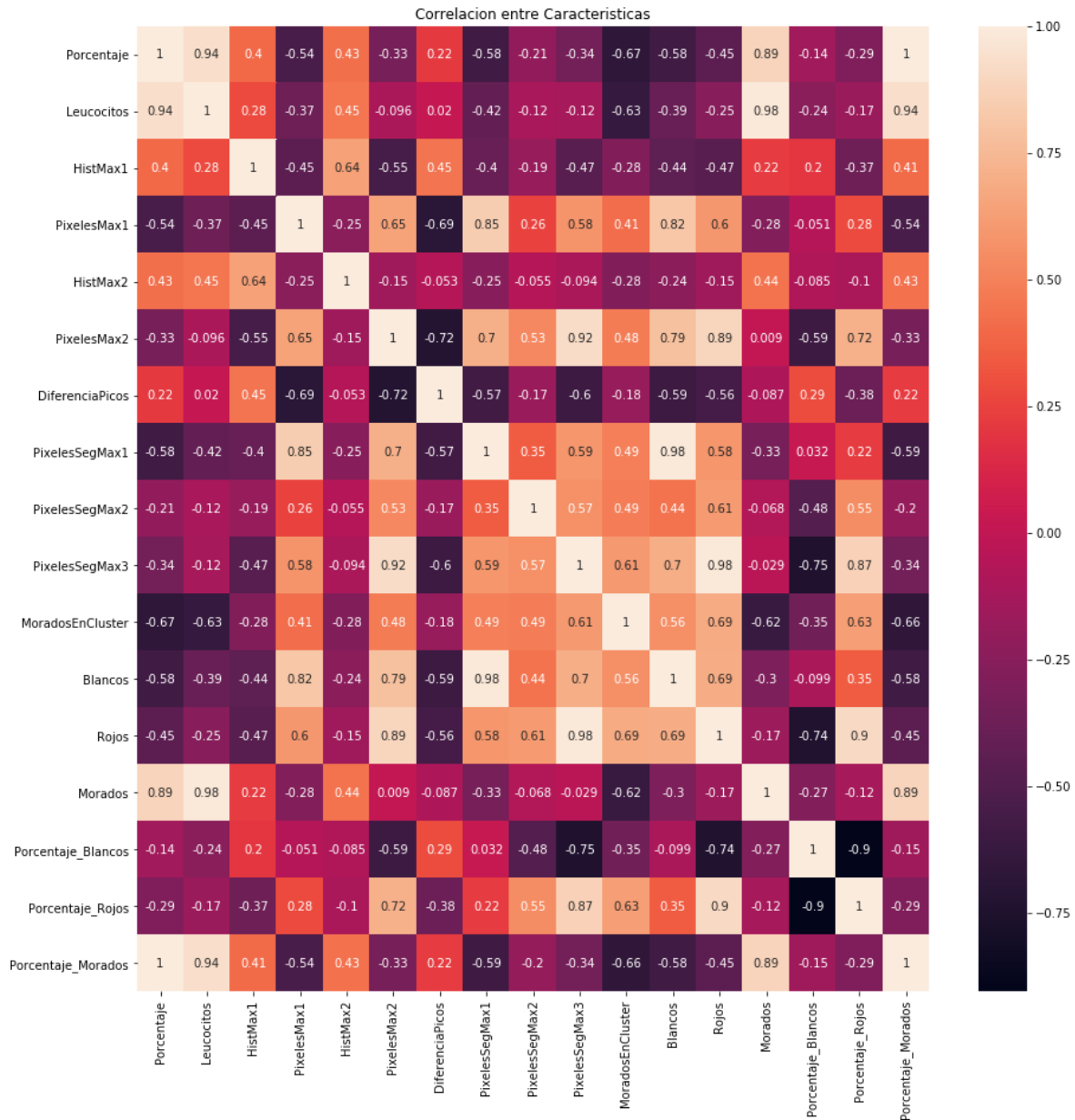


Figura 24. Matriz de correlación de las 17 variables

En la figura 24 podemos apreciar muy bien cuales son las variables que se encuentran altamente correlacionadas entre sí, por lo que podemos probar a eliminarlas de nuestro conjunto de variables y repetir todo el proceso de clasificación automática sobre el conjunto resultante.

Para poder hacerlo aún más visual, hemos impreso los gráficos de correlación de cada variable. En la figura 25, podemos observar la correlación que tiene la variable Porcentaje respecto a las demás.

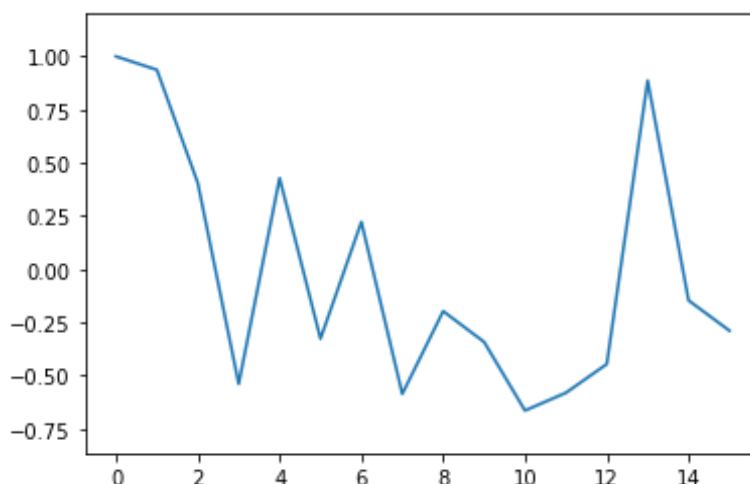


Figura 25. Gráfico de correlación de la variable Porcentaje

Tal y como se puede observar, la variable Porcentaje está altamente correlacionada con la variable número 1 y con la número 13. Estas variables son Leucocitos y Morados, y si lo pensamos, tiene sentido, ya que a mayor valor dentro de la variable Porcentaje, querrá decir que tenemos más píxeles morados en la máscara, lo que implica mayor número de leucocitos y, por tanto, mayor valor dentro de la variable Leucocitos. Del mismo modo pasa con la variable Morados, ya que esta representa la cantidad de píxeles tintados de morado de la imagen tratada con K-Means. Si lo pensamos bien, es normal que se encuentren correlacionadas, ya que están dándonos la misma información de manera indirecta.

Tras examinar aquellas variables que poseen una correlación mayor en valor absoluto a 0.8 con alguna de las variables de entrada, llegamos a que tenemos ocho variables que pueden ser suprimidas (en negrita), las cuales se encuentran correlacionadas con las variables que se marca a continuación:

- **Leucocitos** → Correlación con Porcentaje, Morados y Porcentaje_Morados.
- **PíxelesSegMax1** → Correlación con PíxelesMax1 y Blancos.
- **PíxelesSegMax3** → Correlación con PíxelesMax2, Rojos y Porcentaje_Rojos.
- **Blancos** → Correlación con PíxelesMax1 y PíxelesSegMax1.
- **Rojos** → Correlación con PíxelesMax2, PíxelesSegMax3 y Porcentaje_Rojos.
- **Morados** → Correlación con Porcentaje, Leucocitos y Porcentaje_Morados.
- **Porcentaje Rojos** → Correlación con PíxelesSegMax3, Rojos y Porcentaje_Blancos.
- **Porcentaje Morados** → Correlación con Porcentaje, Leucocitos y Morados.

Tras examinar cada variable eliminada con sus correlacionadas, vemos como de alguna manera tienen que ver entre sí.

Si eliminamos esas ocho variables y repetimos de nuevo la clasificación por Grid Search, llegamos a los resultados de la tabla 9 :

Clasificador	Sin normalizar	MinMaxScaler	StandardScaler
KNN	90.72%	99.09%	99.09%
Arboles de Decisión	98.18%	98.18%	98.18%
Random Forest	100.00%	100.00%	100.00%
Regresión Logística	90.45%	99.09%	99.09%
SVM	86.00%	99.09%	100.00%
Redes Neuronales	82.45%	99.09%	99.09%

Tabla 9. Porcentajes de acierto tras el uso del Grid Search con 9 variables

Hemos marcado en rojo todas aquellas opciones en las que tras la reducción de variables empeoramos el porcentaje. Como se puede apreciar, seguimos pudiendo conseguir un 100.00% de hasta cuatro formas distintas usando dos clasificadores diferentes, por lo que podríamos decir que esta reducción no afecta a la hora de seleccionar un solo modelo para clasificar. Sin embargo, en lo que respecta al conjunto de clasificadores, funciona mucho mejor si usamos las diecisiete variables disponibles, por lo que podríamos escoger cualquier opción.

En mi caso, me quedaré con todas las variables, ya que, si hay tanta diferencia en algunos clasificadores, quiere decir que todas nos están aportando información útil por muy parecida que esta parezca.

Una de las razones por las que sí que podríamos decantarnos por una de las opciones sería ver que ejecuta en mucho menos tiempo. Sin embargo, ambas ejecuciones duran parecido, por lo que no podemos decantarnos por esa opción.

Así pues, vamos a centrarnos en ver de las diecisiete variables iniciales cuales tienen mayor importancia a la hora de hacer la clasificación. En la figura 26 se puede observar la importancia de cada variable tras aplicar un Feature Importance en nuestro modelo generado por el Random Forest, el cual las ordenará basándose en su impureza [38].

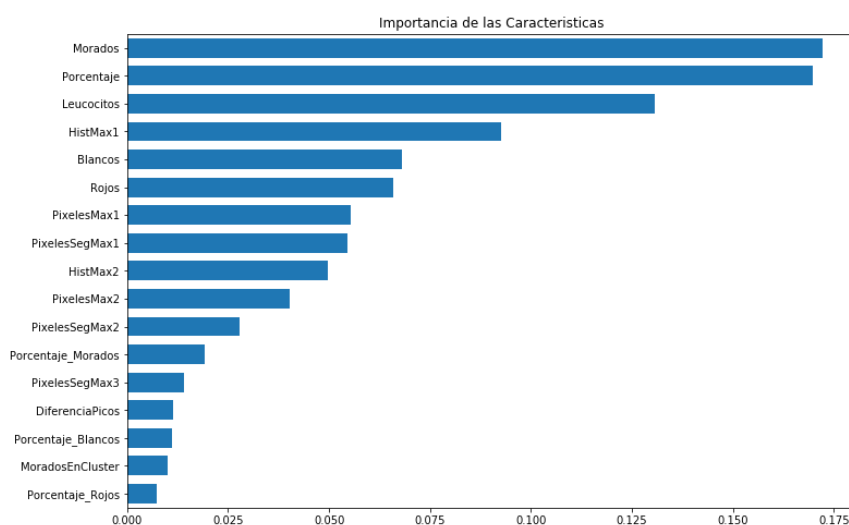


Figura 26. Importancia de cada variable en el Random Forest

En la figura 26 podemos ver como las variables que más peso tienen a la hora de hacer la clasificación son Morados, Leucocitos y Porcentaje, algo que tiene sentido, ya que son tres variables que guardan una alta correlación entre sí.

Para contrastar que dichas variables son de gran importancia, vamos a probar otro método de interpretabilidad denominado SHAP Values, con el cual veremos la separabilidad que proporciona cada variable. En los booleanos (MoradosEnCluster), tan solo tendremos dos colores, mientras que, en las variables numéricas, tendremos todo un gradiente de color. En la figura 27 podemos observar la salida tras calcular sus SHAP Values.

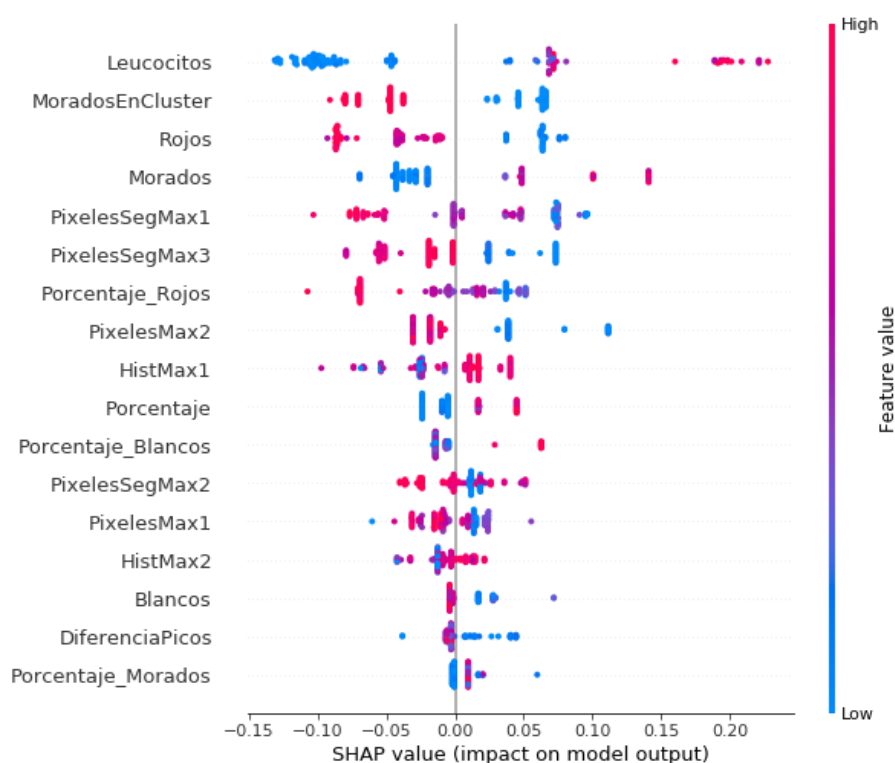


Figura 27. Separabilidad de cada variable tras usar los SHAP Values

En esta figura podemos contrastar la información obtenida en la figura 26, ya que, si nos fijamos, Leucocitos separa muy bien en base a su valor, al igual que Porcentaje y Morados. Sin embargo, podemos observar también contrariedades, ya que según la figura 22, las variables MoradosEnCluster, PixelesMax2, Rojos y PixelesSegMax3 no se encuentran tan arriba como sus SHAP Values parecen indicar, ya que tal y como podemos observar, son capaces de hacer una muy buena separación en base a sus valores. Esto se puede deber a que en la figura 23, tan solo vemos el rango de valores de cada variable gracias al gradiente de color. Sin embargo, esto no nos está indicando a que clase pertenece, ya que podemos tener dos puntos del mismo color en esta figura que en realidad sean de distintas clases. Es por ello por lo que debemos tener cuidado a la hora de interpretar los datos, pues podríamos llegar a este tipo de malinterpretaciones.

Por último, y teniendo en cuenta que nuestro mejor clasificador es Random Forest, vamos a imprimir uno de los árboles que genera para poder interpretar de una manera más visual como se clasifican las muestras. A continuación, se muestra la figura 28 con uno de los árboles generados por el Random Forest, clasificador con el que llegamos al 100% de acierto tan solo con diez árboles.

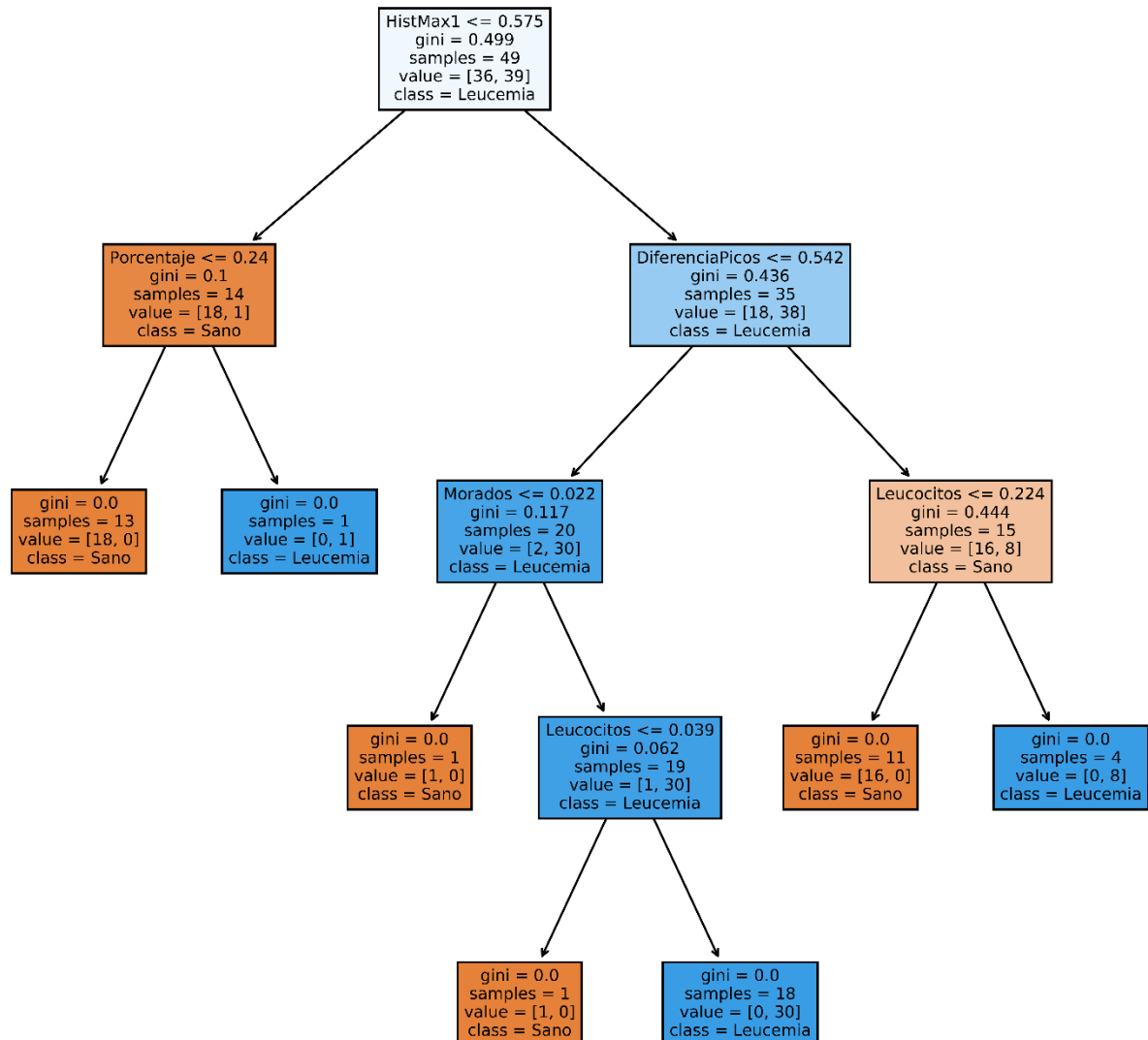


Figura 28. Uno de los diez árboles de decisión del Random Forest

Como podemos ver, las tres variables que hemos visto que eran importantes, aparecen en este árbol, por lo que contrastamos aun más su peso. Este es uno de los árboles más profundos de la colección de árboles generada, pero los podemos encontrar con hasta tres niveles de profundidad contando con el nodo raíz.

4.4. Fase 4: BBDD y UI

Tal y como ya hemos mencionado anteriormente, hasta obtener estos porcentajes, hemos tenido que hacer muchas pruebas. Si tenemos en cuenta que cada prueba genera dieciocho modelos posibles con los hiper parámetros que cada uno posee, vemos como tenemos muchísima información que almacenar para llevar un registro de cada prueba.

Inicialmente, este registro lo llevábamos en una hoja de Excel, pero a medida que íbamos haciendo más y más pruebas, nos hemos dado cuenta de que la hoja de Excel se estaba volviendo demasiado grande y cada vez menos manejable. Es por ello, que hemos decidido construir una base de datos propia en la que poder almacenar todos los resultados de cada prueba, en donde guardaremos datos como los porcentajes obtenidos, que hiper parámetros han sido seleccionados tras el Grid Search, cuando se ha realizado la prueba, quien la ha realizado...

Por otro lado, el objetivo de esta base de datos es también la de recabar información estadística con la que poder mejorar el algoritmo en un futuro. Un ejemplo de esto que estoy diciendo, sería generar una consulta en SQL la cual nos devolviera para cada clasificador los hiper parámetros que aparezcan en todas las pruebas cuyos porcentajes no bajen de 100.00%. De esta manera, podríamos ver si existe algún hiper parámetro que esté relacionado con este resultado y podremos eliminar de las opciones del Grid Search los que no sean devueltos. Esto podría servirnos para mejorar el tiempo de ejecución a la hora de hacer la clasificación, ya que posiblemente reduzcamos considerablemente la cantidad de hiper parámetros, haciendo que la combinatoria entre ellos sea muchísimo menor y, por tanto, más rápida.

Para poder facilitar el uso de la base de datos, hemos creado también una interfaz de usuario con la que poder manejarla de distintas maneras y con diferentes utilidades. La primera de ellas sería poder introducir nuevos valores a los campos de cualquier tabla. La segunda, sería generar una consulta en SQL la cual crearía un fichero en texto plano con toda la información necesaria para la fácil interpretación del resultado.

Además, tanto la interfaz como la base de datos (por los requisitos que exigimos a cada campo de las tablas), tienen un amplio control de errores con el que evitamos introducir datos que puedan estropearnos la base de datos.

La base de datos creada es del tipo relacional, por lo que tenemos que especificar una arquitectura que cumpla con las relaciones entre las distintas tablas que engloba. Posee ocho tablas distintas:

- USUARIOS
- PRUEBAS
- CARACTERISTICAS
- CLASIFICADORES
- PRUEBAS_CARACTERISTICAS
- PRUEBAS_CLASIFICADORES
- PRUEBAS_CLASIFICADOR_PARAMETROS
- CLASIFICADOR_PARAMETROS

La arquitectura y las relaciones de la base son las que se pueden observar en la figura 29.

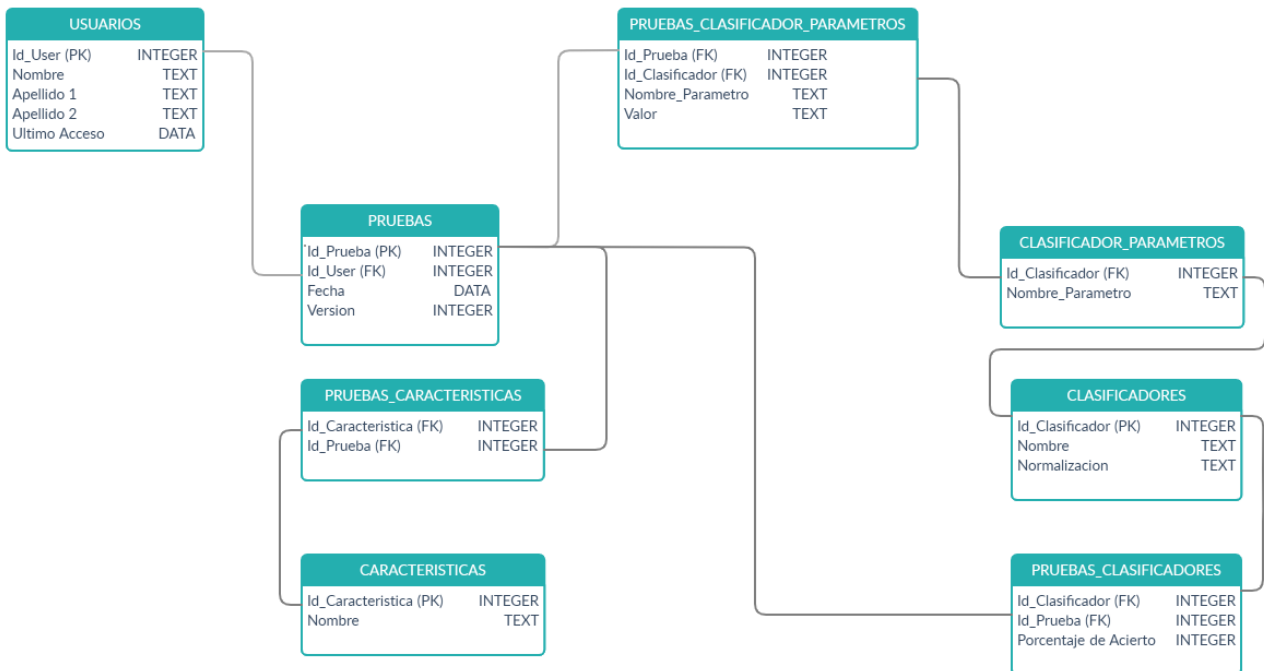


Figura 29. Arquitectura de la BBDD

Los campos que aparecen como PK en la figura anterior son las claves primarias de la tabla, es decir, el identificador único de cada fila. Los que aparecen como FK, en cambio, son los referentes a las claves extranjeras, las cuales son los identificadores de otra tabla que se usan para relacionar las tablas.

Tal y como ya hemos dicho, la BBDD gira entorno a las pruebas. Es por esto por lo que dicha tabla tenga tanta importancia en la arquitectura.

En la actualidad, solamente con los datos generados con las pruebas realizadas durante el proyecto, hemos poblado la tabla con aproximadamente 2500 instancias, todas ellas introducidas a mano dentro de un archivo con extensión .py el cual genera toda la BBDD desde cero. Con esto lo que queremos es tener siempre metidos estos datos por defecto para poder tener una base sólida con la que poder ir poblando poco a poco a medida que vayamos haciendo más pruebas.

En cuanto a nuestra interfaz de usuario, hemos generado un menú lo más sencillo posible para que pueda ser manejado fácilmente por cualquiera. En las figuras 30, 31, 32 y 33, podemos observar cómo sería esta interfaz.

Al abrir la interfaz, tendremos la figura 30 en pantalla, en donde tal y como podemos ver en la figura 31, tenemos distintas acciones que podemos ejecutar, las cuales nos llevarán a otras interfaces personalizadas para la opción escogida. Esto último lo podemos ver en las figuras 32 y 33, en donde se ve como tras escoger una de las distintas acciones, accedemos a un nuevo menú. Finalmente, en la figura 34, vemos el resultado que devolverá la consulta de la figura 32.

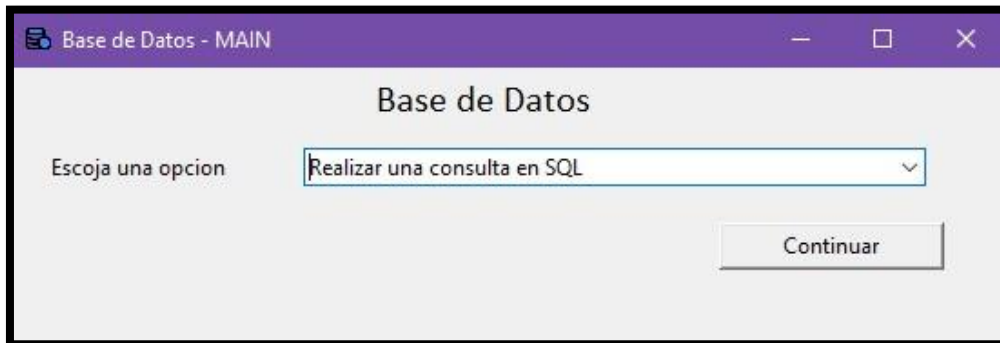


Figura 30. Menú inicial de la UI

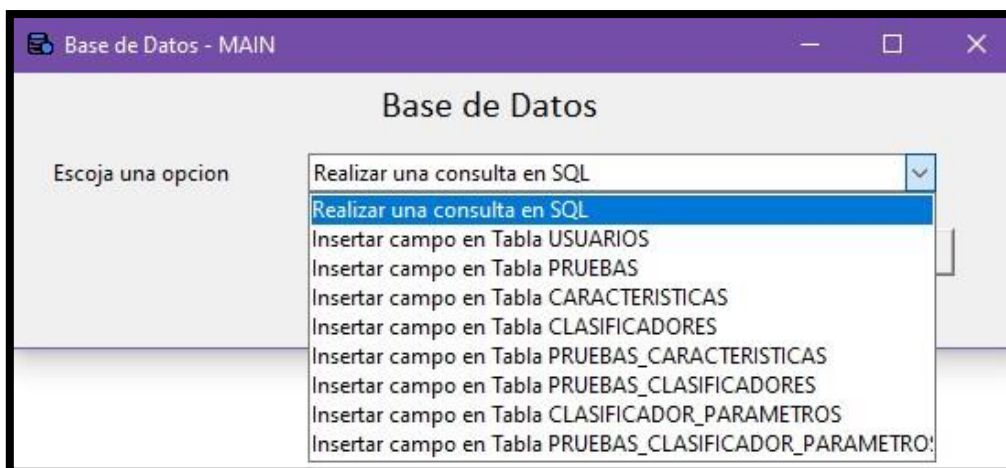


Figura 31. Menú inicial de la UI con la lista de opciones posibles

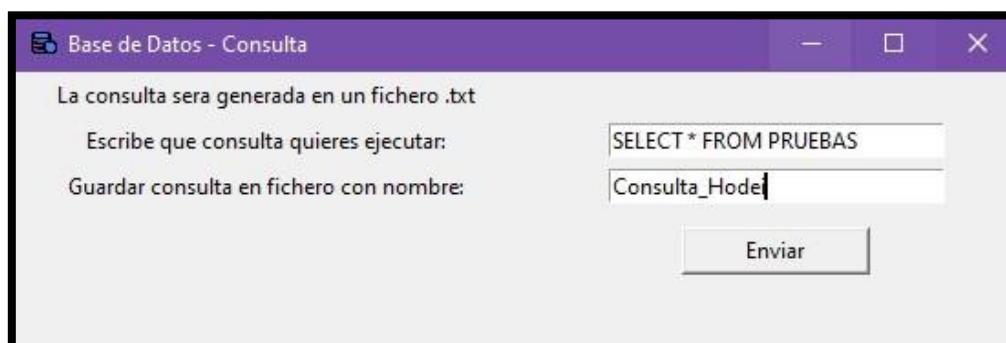


Figura 32. Menú de consultas de la UI

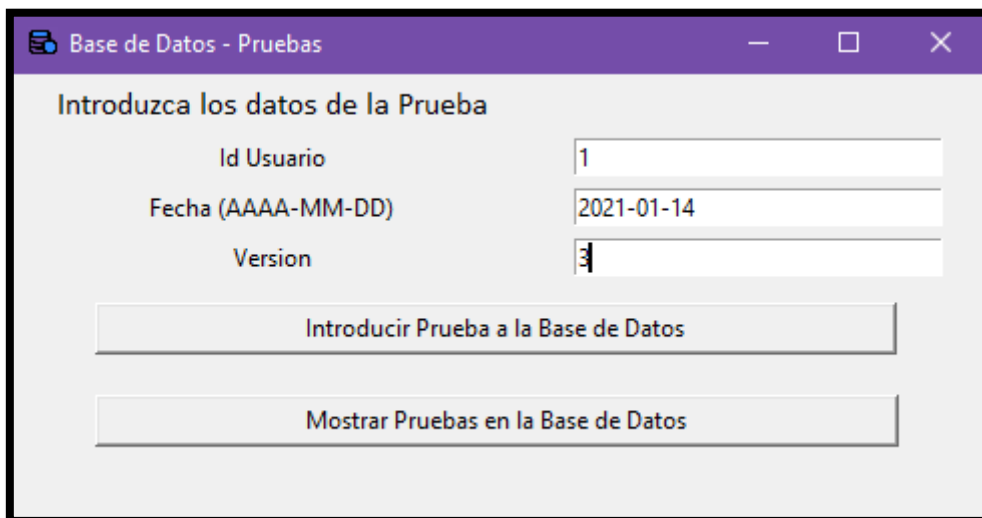
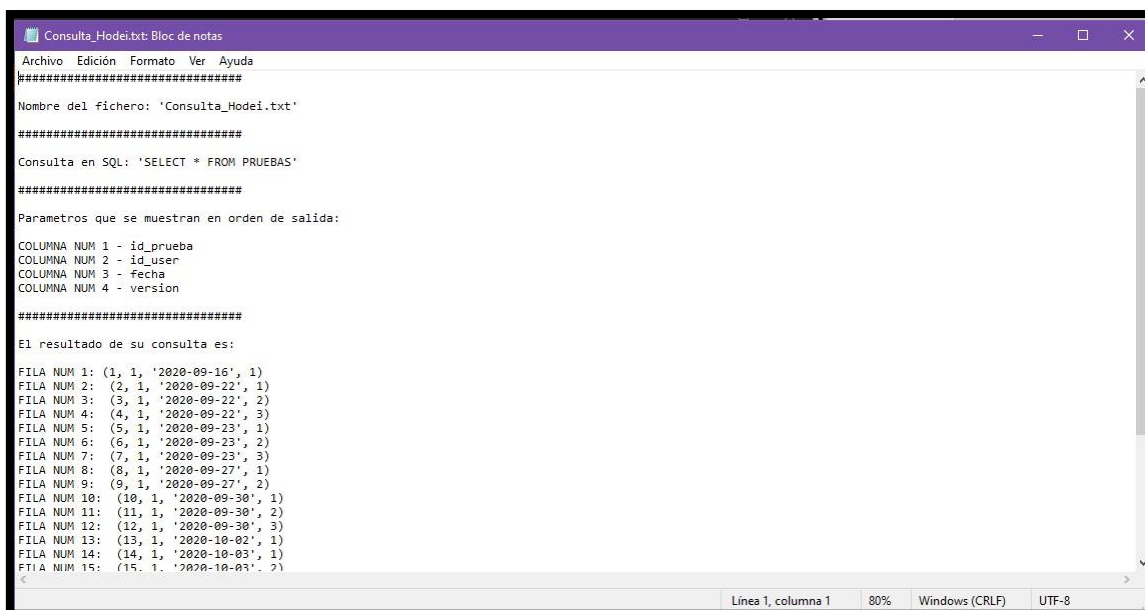


Figura 33. Menú de inserción de campos a la tabla PRUEBAS

Las demás opciones de la UI tienen una interfaz similar a la figura 34 en donde solamente cambian los valores que se piden. Tenemos que resaltar que, para la correcta inserción de los campos dentro de una tabla, se ha de haber introducido anteriormente todos aquellos campos a las demás tablas en las que tengamos dependencias, ya que, de lo contrario, estaríamos saltándonos la estructura de la BBDD, y por ello, nos aparecería un mensaje de error por pantalla indicándonos qué es lo que estamos haciendo mal.

Como ya hemos comentado, en la figura 32 podemos ver como dice que los resultados de la consulta realizada serán impresos dentro de un fichero de extensión .txt. En la figura 34 podemos ver un ejemplo del fichero que nos generaría la consulta realizada en la figura 32.



```

Archivo Edición Formato Ver Ayuda
#####
Nombre del fichero: 'Consulta_Hodei.txt'
#####
Consulta en SQL: 'SELECT * FROM PRUEBAS'
#####
Parametros que se muestran en orden de salida:
COLUMNA NUM 1 - id_prueba
COLUMNA NUM 2 - id_user
COLUMNA NUM 3 - fecha
COLUMNA NUM 4 - version
#####
El resultado de su consulta es:
FILA NUM 1: (1, 1, '2020-09-16', 1)
FILA NUM 2: (2, 1, '2020-09-22', 1)
FILA NUM 3: (3, 1, '2020-09-22', 2)
FILA NUM 4: (4, 1, '2020-09-22', 3)
FILA NUM 5: (5, 1, '2020-09-23', 1)
FILA NUM 6: (6, 1, '2020-09-23', 2)
FILA NUM 7: (7, 1, '2020-09-23', 3)
FILA NUM 8: (8, 1, '2020-09-27', 1)
FILA NUM 9: (9, 1, '2020-09-27', 2)
FILA NUM 10: (10, 1, '2020-09-30', 1)
FILA NUM 11: (11, 1, '2020-09-30', 2)
FILA NUM 12: (12, 1, '2020-09-30', 3)
FILA NUM 13: (13, 1, '2020-10-02', 1)
FILA NUM 14: (14, 1, '2020-10-03', 1)
FILA NUM 15: (15, 1, '2020-10-03', 2)

```

Figura 34. Fichero generado tras la consulta de la figura 32

4.5. Fase 5: Deep Learning

En esta última fase, vamos a intentar hacer la clasificación de un modo distinto sin necesidad de hacer uso de las técnicas antes mencionadas. Para esto, utilizaremos el Deep Learning o Aprendizaje Profundo.

Este método lleva a cabo el proceso de Aprendizaje Automático usando una red neuronal que se compone por muchas capas ocultas. De ahí la palabra “profundo”. En definitiva, el Deep Learning simula una gran red neuronal con distintos métodos de activación [39].

Además, este método, en lugar de recibir los datos para ejecutarlos a través de ecuaciones predefinidas, es capaz de configurar él mismo los parámetros básicos acerca de los datos y entrena a la computadora para que aprenda por cuenta propia reconociendo patrones mediante el uso de muchas capas de procesamiento.

En la figura 35, tenemos un ejemplo de lo que sería una red neuronal profunda diseñada para el aprendizaje profundo y su diferencia respecto a una red neuronal diseñada para el aprendizaje automático.

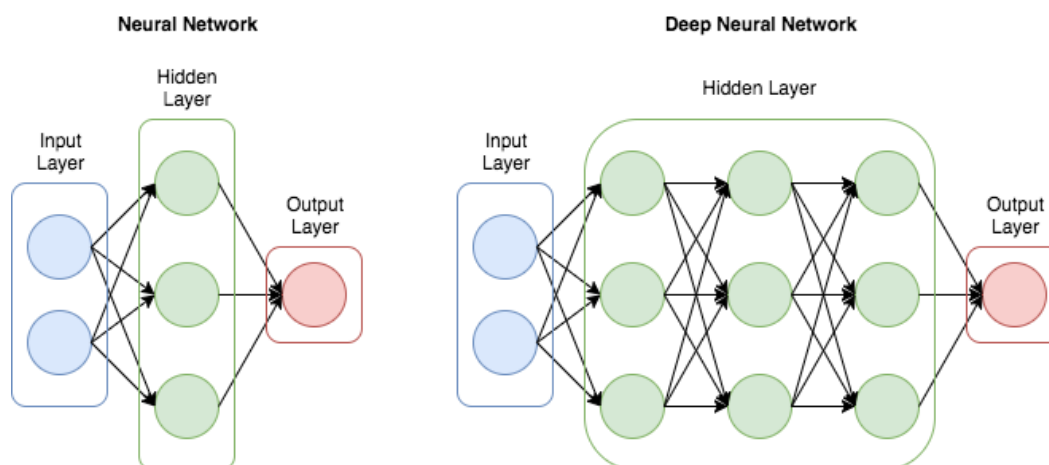


Figura 35. Diferencia entre una red neuronal simple y una profunda

Por muy similar que parezca a una red neuronal convencional, es algo totalmente distinto, por lo que antes de empezar a escribir código para la realización de la clasificación mediante aprendizaje profundo, tenemos que formarnos bien. Para esto, durante estos meses, hemos realizado tres cursos de un total de cinco que imparte *DeepLearning.ai* vía online en la plataforma de *Coursera* dentro de un programa especializado en aprendizaje profundo. Adjunto el enlace a los certificados a continuación:

Neural Networks and Deep Learning:

<https://coursera.org/share/cb49c81b3212ddc608416c6522fe47f4>

Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization:

<https://coursera.org/share/35de54f21a76ec1e2b5357ef751dcfac>

Structuring Machine Learning Projects:

<https://coursera.org/share/00db9e5354d5d173a0b365449f159118>

Una vez que ya hemos recibido una formación básica para la comprensión del aprendizaje profundo, comenzamos a hacer la clasificación con ello.

El primer problema con el que nos encontramos es que nuestras imágenes se encuentran en una resolución de 2592x1944, por lo que, al intentar entrenar una red con tantos datos, gastamos demasiada memoria, tanto en nuestro equipo de trabajo como en la plataforma Google Colab, por lo que tendremos que reducir la resolución de las imágenes para no terminar con los recursos de nuestros equipos.

Hemos generado cinco datasets con distintas resoluciones con las que probar si la memoria era suficiente:

- 1944x1458 (75.00% de resolución respecto a la original)
- 1458x1093 (56.25% de resolución respecto a la original)
- 1093x820 (42.16% de resolución respecto a la original)
- 820x615 (31.63% de resolución respecto a la original)
- 615x461 (23.72% de resolución respecto a la original)

Para la clasificación, lo primero que hemos hecho, ha sido reducir el valor de los píxeles de la imagen haciendo una normalización para tener el rango comprendido entre 0 y 1 en lugar de entre 0 y 255. Una vez hecho esto, hemos generado los conjuntos de train y de test, en donde el tamaño de la matriz de entrada, X viene compuesto no solo por la resolución de la imagen, sino también por el número de espectros de color (3) y la cantidad de muestras. En el caso de la matriz de salida, Y , tendrán el tamaño de una matriz con la cantidad de muestras por el número de clases que tiene el problema a abordar, 2 en nuestro caso.

Tras esto, generamos nuestro modelo asignándole una tasa de aprendizaje de 0.001 para que, aunque aprenda más lentamente, pueda llegar a converger. También le asignamos el valor 100 al número de iteraciones que debe de hacer, es decir, que, tras obtener el primer coste, tendrá que hacer una propagación hacia atrás para reajustar el valor de los pesos y repetirlo haciendo una propagación hacia adelante hasta en cien ocasiones. De este modo, lo que conseguimos es que vaya reduciendo su error a medida que repite el proceso gracias al reajuste de pesos. Además de estos parámetros, también especificamos que el mínimo número de muestras que coja en cada pasada sea igual a doce, ya que tampoco tenemos muchas muestras como para subir ese valor. Para finalizar, hemos utilizado distintas funciones de activación para no estancarnos e ir probando todas las combinaciones posibles.

Como es de esperar, al aplicar un dataset con muy baja resolución, se consiguen los resultados mucho antes que con un dataset de máxima resolución. Es por ello, que vamos a ir probando cada dataset empezando desde el de menor calidad y subiendo poco a poco para ver si nos compensa tener el dataset con mejor calidad o no.

Tras generar y aplicar los modelos explicados para cada dataset, hemos obtenido las figuras 36, 37 y 38, en donde se aprecia no solo el porcentaje final al que se ha llegado sino como ha ido disminuyendo el coste en cada iteración.

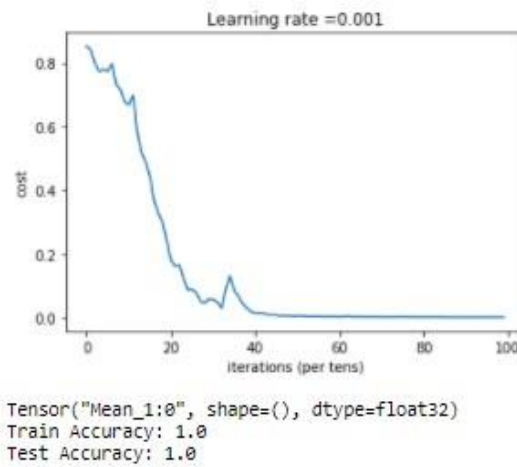


Figura 36. Evolución del coste de error en el dataset 615x461

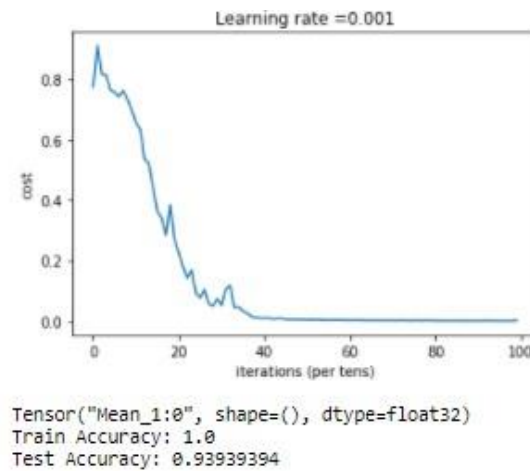


Figura 37. Evolución del coste de error en el dataset 820x615

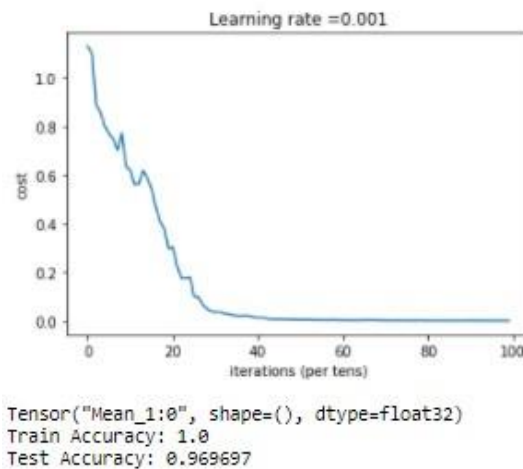


Figura 38. Evolución del coste de error en el dataset 1093x820

Podemos apreciar en estas tres primeras graficas como la red tiene el mismo comportamiento en todas pese a no tener la misma resolución. En todas disminuye el coste de error muy rápidamente teniendo iteraciones en donde tras hacer el reajuste de los pesos, se incrementa. Este incremento, aunque no sea lo esperado, sirve de igual forma para que la red aprenda con qué pesos no debe seguir tras ver como solo empeoran su resultado. Es curioso ver como con la menor resolución posible llegamos a un porcentaje de acierto del 100.00%, algo increíble, y más si tenemos en cuenta que este dataset no llega ni a una cuarta parte de la cantidad de pixeles por imagen que el dataset original. Sin embargo, algo que es curioso es como tras aumentar un nivel más la resolución, bajamos hasta el 93.93% y si seguimos subiendo la calidad, volvemos a subir hasta un 96.96%. Lo que vamos a intentar ahora será probar con mejores resoluciones que, aunque mucho más lentas, pueden llegar a igualar los resultados. Finalmente, también podemos fijarnos en que normalmente, el coste del error converge y se estabiliza sobre la iteración 40, por lo que podemos asignar a los datasets con mayor calidad ese valor como punto final para ahorrar en tiempo. De todos modos, intentaremos ver si es cierta esta última afirmación en la siguiente prueba para poder aplicarlo más adelante. En la figura 39 podemos ver el resultado de la gráfica para el dataset con una resolución de 1458x1093.

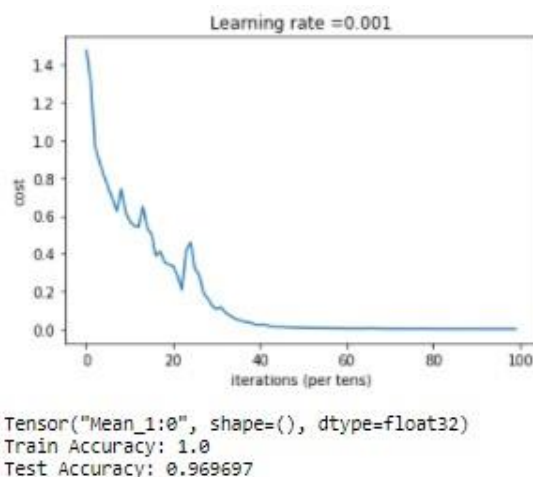


Figura 39. Evolución del coste de error en el dataset 1458x1093

Tal y como hemos dicho, se puede ver como sobre la iteración 40 se estabiliza, por lo que dejaremos que las siguientes pruebas acaben ahí. Al igual que en el último dataset, obtenemos un porcentaje de acierto del 96.96%. Llegados a este punto, probaremos que los dos datasets que nos quedan sean capaces de superar este porcentaje, ya que, de lo contrario, habríamos obtenido ya el mejor resultado posible con la menor de las resoluciones.

Algo que, sí que es curioso, es como cuanto mayor resolución tiene el dataset, mayor coste inicial tenemos.

Tras probar con la siguiente resolución de 1944x1458, hemos logrado finalizar con unos resultados bastante parejos a los anteriores. En la figura 40 se muestra el resultado del dataset de 1944x1458.

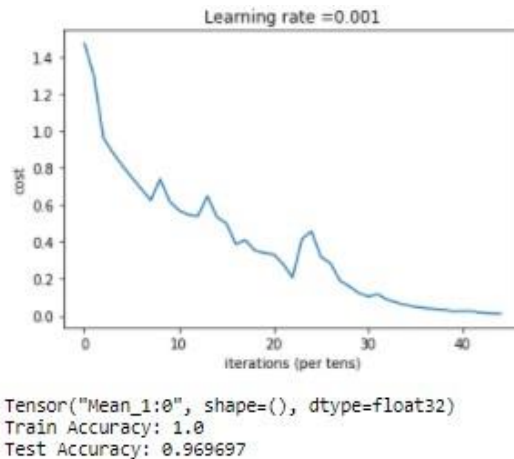


Figura 40. Evolución del coste de error en el dataset 1944x1458

En esta ocasión hemos fijado el máximo de iteraciones en 40 para no malgastar el tiempo. Tal y como podemos ver, llegamos nuevamente al 96.96% de acierto, lo cual nos hace pensar que este será un porcentaje difícilmente superable con los parámetros actuales.

Finalmente, nos queda hacer la prueba con el dataset original de 2592x1944. Sin embargo, tras un largo rato intentándolo, hemos tenido que dar por finalizada la prueba, ya que el equipo de trabajo ha empezado a ralentizarse demasiado hasta el punto de no poder hacer casi nada con él. A continuación, se muestra la figura 41 con el porcentaje de memoria usada durante la realización de esta última prueba fallida.

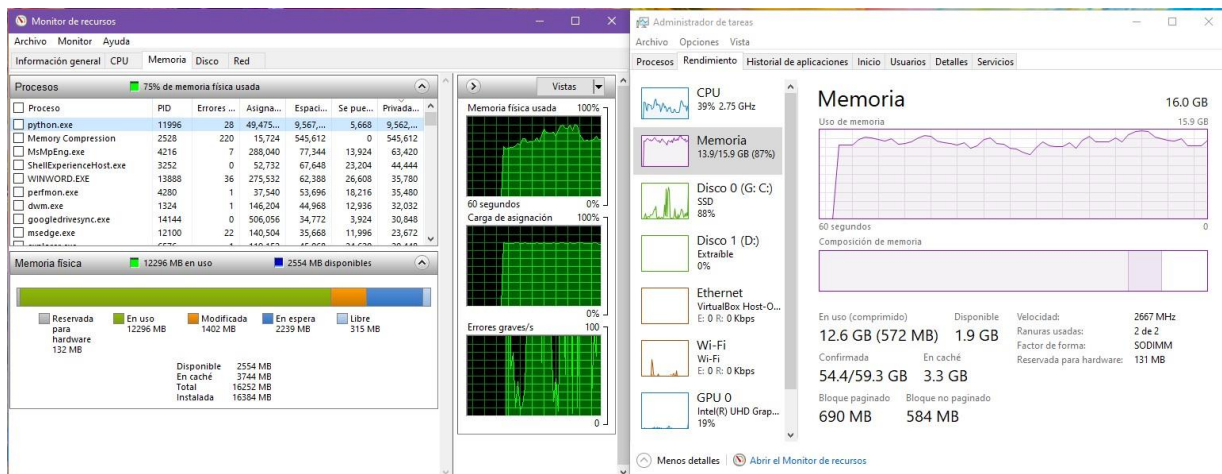


Figura 41. Uso de la memoria durante la prueba 2592x1944

Como podemos observar, nos estaba consumiendo la totalidad de la memoria únicamente en esta tarea, por lo que no veíamos viable continuar con ella.

En definitiva, hemos llegado a la conclusión de que podemos reducir las muestras hasta un 23% de su resolución original y seguir consiguiendo un 100.00% de acierto en test, por lo que el uso del Deep Learning puede ser un gran acierto.

Capítulo 5

Pruebas y validación

Tal y como ya hemos explicado en los apartados anteriores, los modelos obtenidos con los que estamos haciendo todas las explicaciones han sido obtenidas gracias a muchas pruebas que han ido ayudando a crear este proyecto paso a paso.

Al principio, no hacíamos ninguna especie de tratamiento de imagen, por lo que las variables que obteníamos provenían tan solamente de la imagen original. Sin embargo, y para ir aún más allá, empezamos con las variables Leucocitos y Porcentajes; de ahí que en las pruebas de clasificación manual hagamos uso solamente de ellas dos.

En aquella primera prueba con las dos variables, llegamos ya a un 99.09% de acierto en test para SVM y Redes Neuronales, algo que nos hizo ver cómo al contrario que puede ocurrir en otros problemas de clasificación, nosotros ya habíamos conseguido dos variables con las que partir de muy buenos resultados.

De todos modos, desde esa primera prueba con esos 99.09% hasta la final que se muestra tanto en el código como en las explicaciones, ha habido mucho trabajo intermedio, ya que este proyecto no trataba solamente de obtener los mejores resultados posibles, si no de entender el motivo de por qué sucedía lo que estaba sucediendo e intentar mejorarlo en todos los aspectos posibles, tanto en porcentajes como en comprensión o interpretación.

Es por ello, que tras probar con las demás variables que se pueden obtener de la imagen original, y de todas sus combinaciones posibles, llegamos a la conclusión de que quizás era necesario tratar la imagen previamente para poder obtener otras variables con las que ayudarnos.

Una vez ya con nuestra imagen tratada solamente con el K-Means con $K=3$ inicial, vimos como los porcentajes de los 18 clasificadores mejoraban en conjunto llegando incluso al 100.00% en alguna ocasión. Esto nos hizo ver como todavía había mucho margen de mejora, por lo que decidimos seguir probando con todas las combinaciones de variables habidas y por haber.

Finalmente, nos dimos cuenta como también era importante saber que variables utilizar, y no ir probando toda la combinatoria posible entre ellas. Es por esto último que decidimos hacer un estudio de cada una de las variables que íbamos obteniendo. Con esto lo que pretendíamos era además de lo ya explicado, conseguir visualizar los datos con el fin de llegar a una mayor comprensión del problema y así poder avanzar. Fue en este momento cuando decidimos crear la BBDD, ya que nos dimos cuenta de que la dimensión de los datos que teníamos almacenados era demasiado grande como para tenerla escrita.

Lo mismo nos sucedió con la fase del Deep Learning, ya que, pese a ser algo que quería intentar, era totalmente nuevo.

Al principio se hicieron muchas pruebas con esta metodología. Sin embargo, tras ver como no se avanzaba y de cómo aun no comprendíamos que era lo que estaba haciendo en su totalidad, decidimos darle un tiempo al proyecto para poder centrarnos en realizar los cursos de la especialización de Deep Learning antes mencionados.

Una vez terminados los cursos, decidimos incorporar los conocimientos adquiridos en el proyecto, de forma que ya entendemos que es lo que hay que hacer, cómo y por qué se hace.

De esta manera, hicimos pruebas alterando el número de capas ocultas y de funciones de activación de la red, hasta llegar a los ejemplos antes vistos.

Capítulo 6

Conclusiones y líneas futuras

Tras haber finalizado el proyecto, llego a la conclusión de que no todo se basa en sacar un buen porcentaje de acierto, ya que tenemos que estudiar como llegamos hasta él y el motivo de escoger ese método por encima de los demás con el mismo resultado.

Por otro lado, veo como tras haber probado a realizar el proyecto con aprendizaje profundo, funciona muchísimo mejor para todo lo relacionado con imágenes que el aprendizaje automático, ya que no se necesita preprocesar las imágenes como hemos hecho; basta con generar un buen modelo de red neuronal que se encargue de clasificar por sí misma. Me ha parecido además que es algo mucho más rápido, ya que si comparamos el tiempo que necesitamos para hacer la clasificación con aprendizaje automático con el tiempo invertido para el aprendizaje profundo, no hay color.

Otra cosa que he visto también es como no todos los errores son iguales, ya que confundirse por clasificar a una persona enferma dentro del grupo de sanas es mucho más grave que clasificar a una persona sana dentro del grupo de enfermas. Por ello, veo como es muy importante en el aprendizaje supervisado, llevar además de los porcentajes de acierto, el valor de la precisión, recall y FScore.

Ya para finalizar con el apartado de conclusiones, me ha gustado mucho el extraer información de las imágenes del dataset para el aprendizaje automático. No pensaba que se pudieran sacar tantos datos de una imagen con los que poder clasificar la muestra.

Este proyecto me ha ayudado a ver a que me quiero dedicar una vez acabe la carrera, por lo que me gustaría poder seguir investigando acerca de esta enfermedad más a fondo. Por otro lado, estaría bien poder aplicarlo sobre el terreno en casos reales con personas que fueran a donar sangre para además de ir aumentando el dataset, ir probando la efectividad del proyecto e ir modificándolo en base a los resultados.

Estaría bien que en un futuro pudiera servir para facilitar el acceso a este tipo de análisis de sangre a todo el mundo, sin necesidad de tener que estar en ninguna lista de espera o de pagar por pruebas costosas en hospitales privados.

Del mismo modo que cuando vas a donar sangre te llega un análisis con tu nivel de plaquetas, oxígeno en sangre y demás información, estaría bien que tras pasar la prueba por el código nos dijera también dentro de que clase nos encontramos y con qué porcentaje. De esta manera, el servicio sanitario recibiría los datos y se pondría en contacto con nosotros para empezar el tratamiento lo antes posible sin necesidad de tener que empezar a sufrir ninguna clase de síntomas para tener que darnos cuenta. Esto podría ayudar a las personas que padecieran la enfermedad al haberseles sido detectada mucho antes que de normal y cuando todavía no esta tan desarrollada.

En cuanto a las líneas futuras, tal y como he dicho, me gustaría tanto aplicarlo en pacientes reales como identificar qué clase de leucocito tiene mayor relación con la enfermedad detectándolos en la imagen por medio de Deep Learning. También intentaría reducir el tiempo de ejecución del programa eliminando tanto los hiper parámetros que no nos estén sirviendo para los mejores resultados como sus posibles valores. Esto lo podría hacer haciendo uso de la BBDD que hemos creado y ayudaría a reducir la combinatoria en el Grid Search.

Bibliografía

- [1] P.Cervera, “Cáncer, tal vez obra humana”, *RTVE*, 2010. [En línea]. Disponible en: <https://blog.rtve.es/retiaro/2010/10/cancer-tal-vez-obra-humana.html> [Accedido: 26-Dic-2020]
- [2] “Tipos comunes de cáncer”, *NIH*, 2020 [En línea]. Disponible en: <https://www.cancer.gov/espanol/tipos/comunes> [Accedido: 26-Dic-2020]
- [3] “Frotis de sangre”, *MedlinePlus*, 2020 [En línea]. Disponible en: <https://medlineplus.gov/spanish/pruebas-de-laboratorio/frotis-de-sangre/> [Accedido: 26-Dic-2020]
- [4] “¿Qué es el Machine Learning?”, *Iberdrola*, 2019 [En línea]. Disponible en: <https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico> [Accedido: 26-Dic-2020]
- [5] “¿Qué es el Deep Learning?”, *SmartPanel*, 2018 [En línea]. Disponible en: <https://www.smartpanel.com/que-es-deep-learning/> [Accedido: 26-Dic-2020]
- [6] “¿Qué es la leucemia?”, *Fundación Josep Carreras*, 2020 [En línea]. Disponible en: https://www.fcarreras.org/es/que-es-la-leucemia_1585 [Accedido: 26-Dic-2020]
- [7] “Tipos de enfermedades hematológicas”, *Fundación Josep Carreras*, 2020 [En línea]. Disponible en: <https://www.fcarreras.org/es/cancerdelasangre> [Accedido: 26-Dic-2020]
- [8] D.Paternain, “Tema 1 - Introducción al procesamiento de imagen”, *Asignatura de Visión Artificial*, 2019, Universidad Pública de Navarra.
- [9] M.Galar, “Diapositivas de teoría”, *Asignatura de Aprendizaje Formal y Sistemas de Ayuda a la Decisión*, 2019, Universidad Pública de Navarra.
- [10] “Clasificar con K-Nearest-Neighbor ejemplo en Python”, *Aprende Machine Learning*, 2018 [En línea]. Disponible en: <https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/> [Accedido: 11-Ene-2021]
- [11] R.Ferrero, J.L.Lopez, “Qué son los árboles de decisión y para qué sirven”, *Máxima Formación*, 2020 [En línea]. Disponible en: <https://www.maximaformacion.es/blog-dat/que-son-los-arboles-de-decision-y-para-que-sirven/> [Accedido: 11-Ene-2021]
- [12] J.M.Heras, “Regresión Logística para Clasificación”, *IArtificial*, 2020 [En línea]. Disponible en: <https://www.iartificial.net/regresion-logistica-para-clasificacion/> [Accedido: 11-Ene-2021]
- [13] “¿Qué es una red neuronal?”, *IBM*, 2019 [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSLVMB_27.0.0/statistics_mainhelp_ddita/spss/neural_network/nnet_what.html [Accedido: 11-Ene-2021]

- [14] J.Amat, “Máquinas de Vector Soporte (Support Vector Machines, SVMs)”, *Ciencia de Datos*, 2017 [En línea]. Disponible en: https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines [Accedido: 11-Ene-2021]
- [15] J.M.Heras, “Random Forest (Bosque Aleatorio): combinando árboles”, *Ciencia de Datos*, 2020 [En línea]. Disponible en: <https://www.iartificial.net/random-forest-bosque-aleatorio/> [Accedido: 13-Ene-2021]
- [16] “Python 3.7.6”, *Python*, 2019 [En línea]. Disponible en: <https://www.python.org/downloads/release/python-376/> [Accedido: 26-Dic-2020]
- [17] “¿Qué es SQL?”, *styde*, 2018 [En línea]. Disponible en: <https://styde.net/que-es-y-para-que-sirve-sql/> [Accedido: 28-Dic-2020]
- [18] “Jupyter Notebook”, *Jupyter*, [En línea]. Disponible en: <https://jupyter.org/> [Accedido: 28-Dic-2020]
- [19] “¿Qué es Colaboratory?”, *Google Colab*, [En línea]. Disponible en: <https://colab.research.google.com/notebooks/welcome.ipynb?hl=es-EC> [Accedido: 28-Dic-2020]
- [20] “Visual Studio API”, *Visual Studio Code*, 2020 [En línea]. Disponible en: <https://code.visualstudio.com/api> [Accedido: 28-Dic-2020]
- [21] “SQLite3 Python API”, *Python*, 2020 [En línea]. Disponible en: <https://docs.python.org/3/library/sqlite3.html> [Accedido: 8-Ene-2021]
- [22] “DB Browser for SQLite”, *Sqlitebrowser*, 2020 [En línea]. Disponible en: <https://sqlitebrowser.org/> [Accedido: 8-Ene-2021]
- [23] “LISC: Leukocyte Images for Segmentation and Classification”, *ANU College of Engineering and Computer Science*, 2020 [En línea]. Disponible en: <http://users.cecs.anu.edu.au/~hrezatofighi/Data/Leukocyte%20Data.htm> [Accedido: 8-Ene-2021]
- [24] A.Gupta, R.Gupta, “SN-AM Dataset”, *Cancer Imaging Archive*, 2020 [En línea]. Disponible en: <https://wiki.cancerimagingarchive.net/display/Public/SN-AM+Dataset%3A+White+Blood+cancer+dataset+of+B-ALL+and+MM+for+stain+normalization> [Accedido: 8-Ene-2021]
- [25] C.Matek, S.Schwarz, C.Marr, K.Spiekermann, “AML Morphology Dataset”, *Cancer Imaging Archive*, 2020 [En línea]. Disponible en: <https://wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=61080958#61080958171ba531fc374829b21d3647e95f532c> [Accedido: 8-Ene-2021]
- [26] F.Scotti, R.Donida, V.Piuri, “ALL-IDB initiative”, *Unimi*, 2010 [En línea]. Disponible en: <https://homes.di.unimi.it/scotti/all/> [Accedido: 8-Ene-2021]
- [27] Na8, “K-Means en Python paso a paso”, *Aprende Machine Learning*, 2018 [En línea]. Disponible en: <https://www.aprendemachinelarning.com/k-means-en-python-paso-a-paso/> [Accedido: 13-Ene-2021]

- [28] “Preprocessing data”, *Scikit Learn*, 2020 [En línea]. Disponible en: <https://scikit-learn.org/stable/modules/preprocessing.html> [Accedido: 13-Ene-2021]
- [29] “KNeighborsClassifier”, *Scikit Learn*, 2020 [En línea]. Disponible en: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html> [Accedido: 13-Ene-2021]
- [30] “DecisionTreeClassifier”, *Scikit Learn*, 2020 [En línea]. Disponible en: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html> [Accedido: 13-Ene-2021]
- [31] “RandomForestClassifier”, *Scikit Learn*, 2020 [En línea]. Disponible en: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> [Accedido: 13-Ene-2021]
- [32] “LogisticRegression”, *Scikit Learn*, 2020 [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html [Accedido: 13-Ene-2021]
- [33] “SVC”, *Scikit Learn*, 2020 [En línea]. Disponible en: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html> [Accedido: 13-Ene-2021]
- [34] “MLPClassifier”, *Scikit Learn*, 2020 [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html [Accedido: 13-Ene-2021]
- [35] J.A.Sanz, “Tema 3: Preprocesamiento de datos Parte 3, pag.31”, *Asignatura de Descubrimiento del Conocimiento y Minería de Datos*, 2020, Universidad Pública de Navarra.
- [36] “Selección de características basada en filtro”, *Microsoft*, 2020 [En línea]. Disponible en: <https://docs.microsoft.com/es-es/azure/machine-learning/algorithm-module-reference/filter-based-feature-selection> [Accedido: 16-Ene-2021]
- [37] C.Gil, “Análisis de componentes principales (PCA)”, *RPubs*, 2018 [En línea]. Disponible en: https://rpubs.com/Cristina_Gil/PCA [Accedido: 16-Ene-2021]
- [38] “Sklearn.ensemble.RandomForestClassifier”, *Scikit-Learn*, [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=feature%20importances#sklearn.ensemble.RandomForestClassifier.feature_importances_ [Accedido: 18-Ene-2021]

- [39] “Deep Learning: qué es y por qué va a ser una tecnología clave en el futuro de la inteligencia artificial”, *Xataka*, 2020 [En línea]. Disponible en: <https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial> [Accedido: 13-Ene-2021]

