

# Trabajo Fin de Grado

## Grado en Ingeniería de las Tecnologías de Telecomunicación

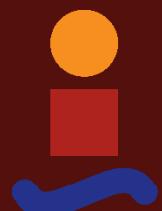
Aplicación de técnicas de aprendizaje profundo (deep learning) a clasificación de imágenes histológicas

Autor: Alejandro Suárez Lamadrid

Tutoras: Begoña Acha Piñero y Carmen Serrano Gotarredona

**Dep. Teoría de la Señal y Comunicaciones**  
**Escuela Técnica Superior de Ingeniería**  
**Universidad de Sevilla**

Sevilla, 2018





Trabajo Fin de Grado  
Grado en Ingeniería de las Tecnologías de Telecomunicación

# **Aplicación de técnicas de aprendizaje profundo (deep learning) a clasificación de imágenes histológicas**

Autor:  
Alejandro Suárez Lamadrid

Tutoras:  
Begoña Acha Piñero y Carmen Serrano Gotarredona  
Catedráticas de Universidad

Dep. Teoría de la Señal y Comunicaciones  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla

Sevilla, 2018



Trabajo Fin de Grado: Aplicación de técnicas de aprendizaje profundo (deep learning) a  
clasificación de imágenes histológicas

Autor: Alejandro Suárez Lamadrid  
Tutoras: Begoña Acha Piñero y Carmen Serrano Gotarredona

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:



# Agradecimientos

---

Este trabajo supone el fin de la que considero que es una importante etapa de mi vida. Durante mis años de formación en la Escuela de Ingeniería no solo he adquirido numerosos conocimientos, sino que también he crecido como persona. Agradezco por tanto a todo el profesorado su labor docente así como a aquellos compañeros con los que he compartido tantas horas de esfuerzo y también diversión.

Agradezco a mis tutoras Begoña y Carmen el haberme hecho descubrir un campo tan interesante como el de la imagen médica e introducirme a la temática de este trabajo, el cual he disfrutado realizando.

Por último, doy las gracias a mis seres queridos su apoyo y ayuda durante estos años, sin los cuales habría sido mucho más difícil pasar por ellos.

Muchas gracias a todos.

*Alejandro Suárez Lamadrid  
Sevilla, 2018*



## Resumen

---

El cáncer es una enfermedad diagnosticada mediante el análisis de muestras histológicas. Una de las características que tienen en cuenta los especialistas y además, indicador temprano de la enfermedad, es el grado de proliferación. Este parámetro es calculado mediante el conteo de mitosis en el tejido a analizar. Este proceso se realiza de manera manual y resulta subjetivo, dando lugar a discrepancias entre patólogos. El desarrollo en los últimos años de técnicas de adquisición, procesado y análisis de imágenes ha permitido numerosos avances en medicina, ayudando a los expertos en su diagnóstico. Este trabajo presenta un método de conteo automático de estas mitosis mediante técnicas de *Deep Learning*. A partir de una base de datos de imágenes histológicas, se ha entrenado una Red Neuronal Convolutacional que posteriormente se ha implementado mediante software, permitiendo analizar imágenes para encontrar figuras mitóticas.



## Abstract

---

Cancer is a disease that is diagnosed using histological samples. Tumor proliferation speed is one of the characteristics used by specialists and also an early stage trace of the disease. This parameter can be calculated by counting the number of mitosis in a tissue. This is a subjective and manual process, leading to disagreements between experts. The recent development in different technologies of images acquisition, processing and analysis has affected medicine, helping the pathologists in their diagnosis. This project presents an automatic method for counting mitosis using *Deep Learning* techniques. A Convolutional Neural Network has been trained using an histological database. This network also has a software implementation, allowing us to process images in order to find those mitosis.



# Índice

---

<i>Resumen</i>	III
<i>Abstract</i>	V
<b>1 Introducción</b>	<b>1</b>
1.1 Motivación médica	1
1.2 Tecnología de partida	2
1.3 Objetivo	2
<b>2 Estado del arte</b>	<b>3</b>
<b>3 Métodos de Deep Learning para el análisis de imagen</b>	<b>5</b>
3.1 Redes de neuronas	5
3.2 Trabajando con imágenes. CNN	7
3.3 Entrenamiento	8
3.4 Bases de datos	10
3.5 Transfer Learning o cómo aprovechar el entrenamiento de una red	10
3.6 Fine Tuning	11
3.7 Conclusiones teóricas en relación al problema del trabajo	11
<b>4 Método propuesto</b>	<b>13</b>
4.1 Material	13
4.1.1 Hardware	13
4.1.2 Software	14
Entrenamiento. MATLAB® como herramienta para el Deep Learning.	14
Entrenamiento. Python y fast.ai.	16
Dataset. TUPAC16   MICCAI Grand Challenge.	19
Implementación. Desarrollo de la aplicación.	21
4.2 Método	22
4.2.1 Planteamiento inicial	22
4.2.2 Mejora del dataset	22
4.2.3 Desbalanceo entre clases	23
Solución propuesta.	23
4.2.4 Selección de candidatos para entrenamiento y validación	27
4.2.5 Entrenamiento del modelo. Última capa	28
4.2.6 Mejora del entrenamiento. Técnicas	31
Learning Rate Finder.	31
Fast.ai Data Augmentation.	32
Descenso del gradiente con reinicios.	32
Fine-Tuning.	33
Learning rate diferencial, imagen médica y entrenamiento final.	33

<b>5 Resultados e implementación software</b>	<b>35</b>
5.1 Resultados del entrenamiento. Validación	35
5.2 Aplicación para la detección de mitosis	37
5.2.1 Exploración de la imagen	37
5.2.2 Predicción	38
Función predict_mitosis	39
5.2.3 Detección múltiple de mitosis	40
Función cluster_box	40
5.2.4 Funciones adicionales	42
Función cuda_enabled	42
Función show_menu	42
5.2.5 Presentación de los resultados	43
Función draw_bb	43
Función draw_mitosis	44
5.2.6 Función principal	44
5.3 Resultados finales. Casos de interés	46
5.3.1 Imagen con presencia de mitosis	48
5.3.2 Imagen sin presencia de mitosis	49
5.3.3 Otros ejemplos de uso	50
Paciente 08. Mitosis con un aspecto diferente	50
Paciente 06. Mayor número de mitosis encontradas	51
Paciente 09. Ausencia de mitosis	52
Paciente 14. Mitosis con un aspecto diferente	52
Paciente 19. Ausencia de mitosis	53
5.4 Resultados finales. Estadísticas de la aplicación	54
5.4.1 Conclusiones sobre las estadísticas	55
<b>6 Conclusiones y líneas futuras</b>	<b>57</b>
<i>Índice de Figuras</i>	59
<i>Índice de Tablas</i>	61
<i>Índice de Códigos</i>	63
<i>Índice alfabético</i>	69
<i>Glosario</i>	69

# 1 Introducción

---

*Whenever I hear people saying AI is going to hurt people in the future I think, yeah, technology can generally always be used for good and bad and you need to be careful about how you build it ... if you're arguing against AI then you're arguing against safer cars that aren't going to have accidents, and you're arguing against being able to better diagnose people when they're sick.*

MARK ZUCKERBERG, 2017

## 1.1 Motivación médica

**E**l cáncer es una enfermedad que representa una de las causas más elevadas de mortalidad en la actualidad. Consiste en una división incontrolada de algunas células del cuerpo y que se diseminan a los tejidos que se encuentran alrededor.

La vida de una célula comprende su nacimiento, crecimiento y división para formar nuevas células, muriendo cuando envejecen o se dañan. Sin embargo en esta enfermedad las células que deberían morir sobreviven y se comienzan a formar células nuevas cuando no son necesarias, dividiéndose además sin interrupción y generando masas denominadas tumores. Son capaces de ignorar las señales que indican a las células que deben detener su división o que comienzan el proceso de apoptosis o muerte celular. También poseen la capacidad de inducir a la formación de vasos sanguíneos que les suministren nutrientes y oxígeno así como de eludir el sistema inmunitario, garantizando su supervivencia.

Se trata de una enfermedad genética causada por cambios en los genes que controlan el funcionamiento de las células. Para cada persona estos cambios genéticos son únicos y no son algo que ocurra únicamente en el inicio de la enfermedad, sino que continúan produciéndose mutaciones del ADN en el desarrollo del tumor. Un punto importante en el transcurso de la enfermedad es el proceso de metástasis, en el que el cáncer se disemina desde el lugar donde comienza a otras partes. Las células que afectan esta nueva zona del cuerpo son del mismo tipo que aquellas del tumor original [1].

Es por todo esto que un diagnóstico temprano es importante para detener el crecimiento del tumor, eliminarlo y evitar el proceso de diseminación. Este diagnóstico es llevado a cabo por patólogos mediante el análisis de imágenes histológicas obtenidas mediante microscopía y que son útiles para estudiar la estructura celular de un tejido y sus funciones. Un indicador relevante en el pronóstico de la enfermedad del cáncer es el grado o índice de proliferación, medida que permite saber el número de células que se están multiplicando. Este valor puede estimarse en este tipo de imágenes mediante el conteo de figuras mitóticas. Pese a que las imágenes se encuentran tintadas con hematoxilina y eosina (H&E) para ayudar en este proceso, no deja de ser algo laborioso por el carácter manual de la tarea además de subjetivo, creando discrepancias entre especialistas. Sin embargo, el desarrollo de las tecnologías digitales y la mejora constante de los microscopios ha propiciado la aparición de herramientas de ayuda al diagnóstico para numerosas tareas clínicas, pese al

recelo que la aplicación de ingeniería a medicina sigue produciendo en algunos expertos [2].

## 1.2 Tecnología de partida

En los últimos años se han popularizado conceptos como *Machine Learning* y *Deep Learning* dentro de las tecnologías de Inteligencia Artificial. Aunque es conocida desde hace décadas, no ha sido hasta hace pocos años que se ha incrementado su uso debido al importante avance que ha sufrido y a la inversión por parte de poderosas corporaciones tecnológicas como Google o Facebook, que han desarrollado las librerías de aprendizaje automático TensorFlow y PyTorch respectivamente. El objetivo de estas bibliotecas es encontrar y descifrar patrones y correlaciones en bases de datos de partida, emulando así el aprendizaje y razonamiento humanos.

Otra de las razones que ha impulsado esta tecnología es la superación de las restricciones de hardware que sufría este área de las ciencias de la computación. El núcleo de funcionamiento es la unidad de procesamiento de gráficos, capaz de realizar una enorme cantidad de operaciones numéricas por segundo. Los avances en la última década en tecnología integrada han reducido los tamaños de los chips y aumentando su capacidad de computación. Además, otras áreas de la computación que se apoyan en el hardware gráfico como las criptomonedas y su reciente popularización, así como de otras no tan relacionadas pero cuyo uso es esencial como en animación o desarrollo de videojuegos, han impulsado la aparición de numerosos dispositivos de la mano de fabricantes como NVIDIA y AMD. Este uso tan generalizado conlleva una reducción de precios, permitiendo a investigadores y al usuario medio por igual aprender y desarrollar aplicaciones de inteligencia artificial.

Por otro lado, la capacidad de unión de este tipo de tecnología a otros como el tratamiento y análisis de imagen y sonido ha abierto un amplio abanico de posibilidades, como el control de la producción en industria, automóviles con conducción autónoma, mejora en sistemas de seguridad por cámaras y numerosas aplicaciones en medicina [3]. Es en este campo donde se centrará el trabajo, donde ya existen proyectos orientados a predecir el riesgo de sufrir una enfermedad y definir mejores tratamientos para los pacientes en base a la recopilación y análisis de datos, ayudando así en sus tareas a los médicos.

## 1.3 Objetivo

Este trabajo propone una herramienta de este tipo basada en redes neuronales convolucionales (CNN) para la detección y conteo automático de figuras mitóticas, con el objetivo de agilizar el tedioso proceso, aportando además precisión en el cálculo, todo con el fin de resultar de utilidad para los especialistas médicos.

## 2 Estado del arte

---

En relación al problema expuesto en este trabajo, existen en el estado del arte algunas aproximaciones basadas en el procesamiento y análisis de imagen. Este proceso consta de métodos de preprocesamiento, extracción de características típicas como color, formas y texturas para así poder clasificar elementos. Aunque estos métodos tradicionales funcionan bien en entornos controlados, en el caso de la imagen médica producen resultados con baja precisión debido a su complejidad. Pese a esto, se ha realizado un repaso de los trabajos existentes en este área, ya que determinados aspectos del procesamiento podrían ser de utilidad para este trabajo.

En 2013, Irshad H. presentó un método automático para la detección de mitosis basado en la extracción de características morfológicas usando diferentes canales y espacios de color en las imágenes a analizar. Incluye métodos como filtros espaciales, umbralización, morfología y detección de bordes. La aplicación final era capaz de extraer 143 características diferentes que eran utilizadas para un clasificador de árbol de decisión. El método fue evaluado en el concurso International Conference of Pattern Recognition de 2012 consiguiendo una precisión de entre 56 % y 70 % con diferentes bases de datos [4].

En 2012 también fueron presentados métodos para la detección y conteo de figuras mitóticas en el Assessment of Mitosis Detection Algorithms (AMIDA). La base de datos de partida correspondía a 11 pacientes de cáncer de mama y el mejor algoritmo tuvo un error comparable al de diferentes expertos patólogos. Los métodos se basaban en el estudio de las imágenes de partida en un espacio de color HED (Hemoglobina, Eosina, Diaminobencidina) [5].

La utilidad de hacer uso de redes neuronales para este problema se hizo evidente pronto, y en 2013 Malon C.D. y Cossatto E. presentaron un método basado en redes neuronales convolucionales para la extracción de características. Estas características contemplaban color, textura e información de forma, demostrando el potencial que esta tecnología tenía mejorando los algoritmos clásicos. Alcanzó unos valores de F1 de 0.659 en una implementación en escáneres de color y de 0.589 en escáneres multiespectrales [6].

Por otro lado, Dan C. y Cires A.G. en 2013 presentan un algoritmo basado en *Deep Learning* capaz de analizar imágenes en espacio de color RGB. La red (DNN) fue entrenada con 50 imágenes en un espacio de color HED, alcanzando una precisión final del 88 % [7].

Recientemente, en 2016 se inició el Tumor Proliferation Assessment Challenge, que concluyó con un elevado número de artículos y métodos propuestos durante el pasado año 2017 [2]. Dada la relación directa de este trabajo con dicho reto al usar la misma base de datos, se detallan a continuación algunos de los métodos que surgieron. Es destacable en primer lugar el trabajo de Salado García J.P. et al de la Universidad del País Vasco, diseñando una arquitectura de red basada en el modelo Lenet entrenada con el *dataset* del reto y que poseía una precisión clasificando del 98.6 %. Para el entrenamiento se eligieron los candidatos con métodos clásicos a partir de la base de datos y dado el desequilibrio entre clases (en el que posteriormente se profundizará en este trabajo) se aplicó un proceso de *data augmentation*. Los resultados de valor F1, el cual fue de 0.541 y de precisión son considerados acordes al estado actual de la materia [8]. En cuanto a este reto, los equipos que alcanzaron un mayor valor F1 fueron los de Lunit Inc. (Corea) con 0.652, el IBM Research

(Zurich y Brasil) con 0.648 y ContextVision (Suecia) con 0.616.

Dejando de lado este reto, resulta interesante el artículo de 2016 por parte de Albarquoni S. et al, donde se propone un modelo de red neuronal convolucional llamado AggNet cuyo objetivo es solucionar el problema de la falta de datos para entrenamiento en la detección de mitosis. Su idea se basa en una colaboración colectiva de usuarios capaces de proporcionar imágenes anotadas con las mitosis marcadas. Se discute si es correcto aplicar el concepto de *crowdsourcing* ya usado para imágenes del mundo real a la imagen médica, con la desventaja de que los usuarios son inexpertos y puede inducir a errores [9].

De este repaso por las recientes investigaciones en relación a la detección de mitosis se concluyen varios aspectos:

- La detección de mitosis es un campo de investigación reciente con margen de mejora.
- La aplicación de tecnologías basadas en redes neuronales convolucionales y *Deep Learning* es muy útil para este problema y mejora significativamente los resultados respecto a métodos clásicos.
- En todos los artículos parece haber unos problemas y preocupaciones similares en relación a la falta de datos para los algoritmos de entrenamiento de redes neuronales. Las soluciones propuestas son variadas y producen resultados positivos.

# 3 Métodos de Deep Learning para el análisis de imagen

---

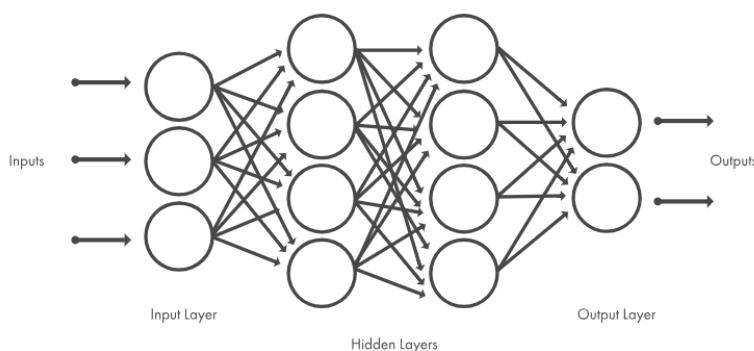
Deep Learning es la tecnología central de este trabajo y la elegida por sus altas capacidades y número de posibilidades para el análisis de imagen. En este capítulo se expone toda la teoría necesaria para comprender en qué consiste y los métodos y conceptos que posteriormente serán introducidos.

Consiste en un acercamiento al funcionamiento del sistema nervioso humano y de cómo aprendemos, a través de modelos computacionales de redes de neuronas las cuales se organizan en capas según la arquitectura y se especializan en encontrar características y patrones en los datos con los que son alimentadas.

## 3.1 Redes de neuronas

Las redes de neuronas son el núcleo de funcionamiento de las tecnologías de *Deep Learning*. Consisten en un modelo basado en un conjunto de unidades básicas llamadas neuronas conectadas entre sí, como puede observarse en la figura 3.1. La interconexión depende del tipo de arquitectura o modelo del que se esté hablando y suelen estar organizadas en capas.

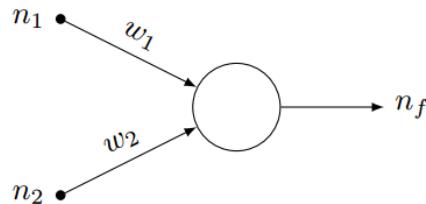
Como se ha comentado, la unidad básica consiste en una neurona la cual se conecta con otras de modo que puede recibir información de varias neuronas y enviar información hacia otras. Sin embargo, el cometido de la neurona no es únicamente el intercambio de información, sino que también la procesa.



**Figura 3.1** Esquema de una red neuronal [10].

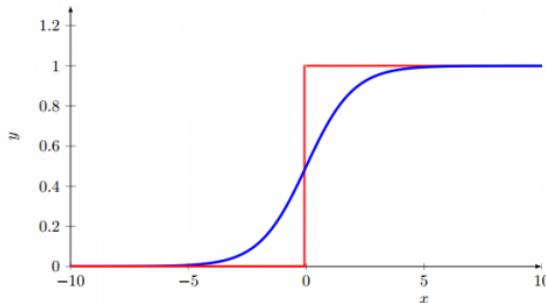
Para comprender mejor todo esto, es mejor centrarse en el tipo de red más simple, también comprendida como la unidad básica funcional de una red neuronal, llamada perceptrón, cuyo esquema queda plasmado en la figura 3.2. El perceptrón consiste en una neurona con varias entradas y una salida. A través de las entradas la neurona recibe valores, los cuales tienen un peso. El concepto de peso es de suma importancia, pues son las variables a encontrar en el entrenamiento de la red y su objetivo es ponderar las entradas de la neurona. Al recibir estos valores ponderados, las neuronas los procesan de una determinada manera, en la mayoría

de casos sumándolos y determinando si superan o no un determinado umbral, conocido como función de activación. Esta función de activación puede consistir en una función escalón en los casos más simples o en una sigmoidal en determinadas redes, como la presente en la figura 3.3. Conocidos por tanto los resultados de un determinado problema y los datos de partida, el proceso de entrenamiento consiste en hallar los pesos que permiten que con los valores ponderados de entrada, se obtenga la salida esperada.



**Figura 3.2** Esquema del perceptrón.

Y es que para entrenar una red que consiga predecir un resultado a partir de unos datos, es necesario previamente tener una base con valores de un problema y sus respectivas salidas. De esta forma, la red ajustará poco a poco sus pesos de modo que intente que ante todas las entradas, la salida sea la correcta.



**Figura 3.3** Función de activación.

Aunque el perceptrón es un ejemplo de red como tal, su alcance está limitado a realizar operaciones básicas booleanas. Conforme aumenta la complejidad de un problema, lo hace del mismo modo la red necesaria para resolverlo. Es un reto importante el diseño de una red, ya que se necesita determinar el número de neuronas necesarias y cómo deben ser las conexiones que existen entre ellas. Surge en este momento el concepto de capa. Al aumentar el número de capas de neuronas, se aumenta el número de operaciones que la red puede realizar y la capacidad de ajustarse al problema. Con dos capas, la red puede además de realizar operaciones básicas representar funciones lineales [11].

Hasta este momento solo se han comentado ejemplos simples de redes, pero el número de capas y formas de conexión dan un número infinito de posibilidades y en este concepto radica el potencial de las redes de neuronas y la inteligencia artificial. En cuanto al tipo de conexión, son de especial importancia las redes unidireccionales, cuyos enlaces no forman bucles. Son comunes al facilitar el estudio de las mismas y evitan además comportamientos caóticos, por lo que en aplicaciones finales son las más utilizadas. Si se habla del tipo de entrenamiento, existen redes supervisadas a las cuales se les proporciona las entradas y las respectivas salidas como ya se ha comentado y las redes no supervisadas a las que solo se les proporciona una entrada. Este tipo de redes están aún en desarrollo pero apuntan a jugar un papel principal en las ciencias de la computación al no depender de intervención humana suministrando datos. Las redes que se tratan en este

trabajo son del tipo unidireccionales y con entrenamiento supervisado, ya que se parte de una base de datos conocidos, por lo que se profundizará en ellas.

Otro punto a tener en cuenta es el número de capas. Comúnmente en las redes se hace distinción entre tres tipos de capas:

- **Capa de entrada:** Aquella que recibe los valores de entrada de la red.
- **Capa de salida:** Capa con los resultados que ofrece la red.
- **Capas ocultas:** Conjunto de capas internas de la red cuyo cálculo de sus pesos proporciona la solución al problema.

Es en las redes con varias capas ocultas donde entra el concepto de redes profundas, capaces de generar predicciones de resultados a partir de conjuntos de datos más complejos.

Llegados a este punto, se puede observar que las redes son herramientas de gran utilidad para procesar valores a partir de un proceso de entrenamiento y que generan un resultado que trata de ajustarse a lo aprendido de la experiencia. Sin embargo, el poder de las redes neuronales va más allá, sobre todo gracias a la aparición de arquitecturas de redes convolucionales (también conocidas como CNN), las cuales no se limitan a recibir cantidades de entrada sino que permiten trabajar con imágenes y están provocando importantes avances en los campos de la visión artificial y el procesamiento del audio.

## 3.2 Trabajando con imágenes. CNN

Las redes neuronales convolucionales son aquellas que tratan de comportarse de manera similar a cómo funcionan las neuronas de la corteza visual del cerebro. Son conocidas algunas funciones del cerebro biológico en cuanto a la visión, donde existen capas enfocadas al reconocimiento de líneas simples, bordes, patrones simples, etc., y que de forma gradual permiten una interpretación completa de lo que el ojo recibe. Se caracterizan por ser redes multicapa cuyas capas consisten en matrices bidimensionales que del mismo modo que el cerebro son capaces de, tras un entrenamiento adecuado, reconocer desde simples formas a texturas o colores para así determinar a qué corresponde lo que en la imagen aparece y realizar una clasificación.

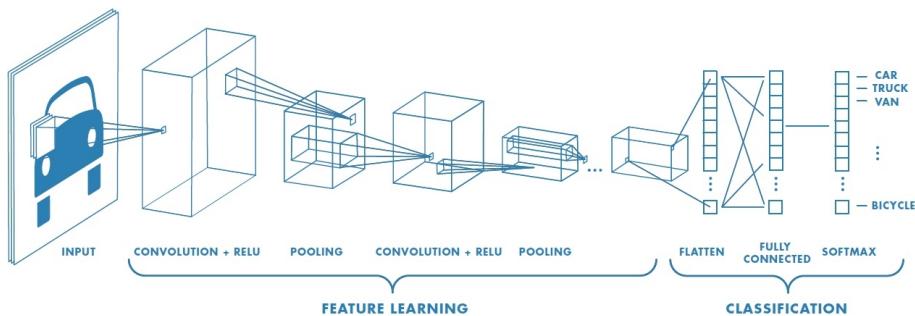
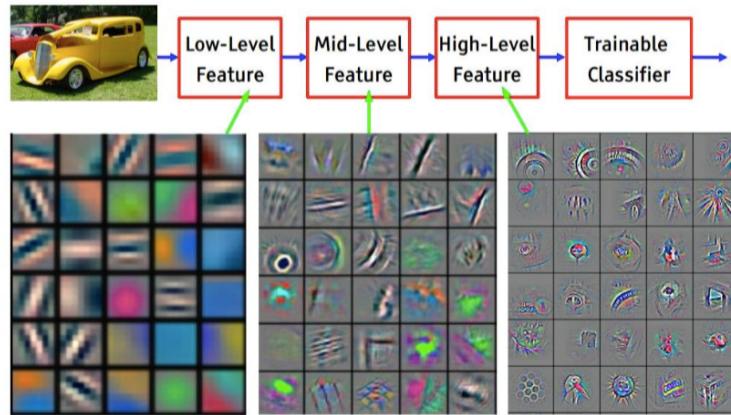


Figura 3.4 Ejemplo de una CNN [10].

Se tienen por tanto un conjunto de capas capaces de reconocer patrones cuya combinación generan los elementos que se desean detectar. La complejidad como es obvio resulta mayor que en las redes introducidas en el apartado anterior, pero también lo es el problema, al estar tratando con una imagen, es decir, un conjunto de elementos bidimensionales.

Las capas convolucionales pueden entonces entenderse como filtros para las imágenes de entrada, cuya convolución con la misma genera una posible activación de la neurona. Un conjunto de activaciones genera una salida correspondiente a una activación, como puede observarse en el ejemplo de la figura 3.5.

Además de las capas convolucionales, las arquitecturas presentan otros tipos de capas, tal y como puede verse en la figura 3.4. En el caso de las capas Pooling están diseñadas para que la red tolere distorsiones, en



**Figura 3.5** Vista de una CNN como filtros [12].

el sentido de permitir desplazamientos relativos entre características extraídas. Con esto se consigue una red más robusta, además de realizar otras funciones que favorecen a los tiempos de computación como la reducción espacial del mapa de características. En el caso de las Fully connected layers su objetivo es permitir una clasificación de los datos de entrada. Mientras que las capas convolucionales almacenan información de características locales, las capas totalmente conectadas incluyen información compuesta de las capas convolucionales.

Dado que el problema del trabajo consiste en imágenes, las redes neuronales convolucionales son la herramienta ideal para este caso. De forma teórica, se tendrá una red capaz de reconocer patrones presentes en mitosis que hayan sido encontrados tras un entrenamiento, generando estos filtros o capas neuronales que se activan en presencia de estas figuras en función de las formas que se presenten, las texturas, los colores y otras características que la red sea capaz de encontrar. Dado que existen dos posibles resultados, que son presencia o no de mitosis, la última capa tendrá dos neuronas que se activarán según la clase que corresponda.

### 3.3 Entrenamiento

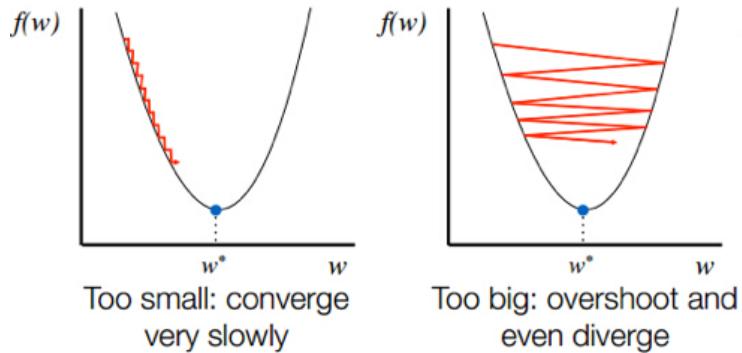
Como se ha comentado, una red no es más que una arquitectura de conexiones de elementos llamados neuronas. Estas conexiones llevan unos pesos desconocidos que ponderan los valores intercambiados por las neuronas. Será el proceso de entrenamiento el que calcule dichos pesos, ajustando la base de datos de partida con sus soluciones en el caso de un entrenamiento supervisado.

El funcionamiento es simple: un conjunto de datos entra en la red por la capa de entrada y la red genera una solución. Si dicha solución es incorrecta, se ajustan los pesos de modo que a la salida se tenga el valor correcto. Esto se hace en base al cálculo de las pérdidas (loss), las cuales son un indicador de la lejanía de las predicciones que hace el modelo a la solución real, similar al significado que tiene el error cuadrático medio usado para regresión lineal. Este proceso continuo de procesamiento, predicción y corrección genera a la larga un conjunto de pesos que se ajustan, siempre que el entrenamiento se haya llevado a cabo de forma adecuada, a los datos de partida [11].

Uno de los algoritmos más destacables de entrenamiento es el de *Backpropagation*, el cual se basa en el método del gradiente y en el que se entiende el problema como una superficie en la cual encontrar un mínimo, es decir, el conjunto de pesos que causa un error mínimo en la predicción. La responsabilidad del error queda repartida entre todos los pesos que afectan a una determinada salida. Si un resultado es incorrecto, se propaga el error hacia atrás generando un cambio en todos los pesos afectados de las capas ocultas. Si el modelo y el entrenamiento son correctos, dicho mínimo es encontrado y por lo tanto se tiene una solución para la red.

En el proceso de entrenamiento existen dos variables importantes a tener en cuenta para alcanzar con éxito las soluciones deseadas:

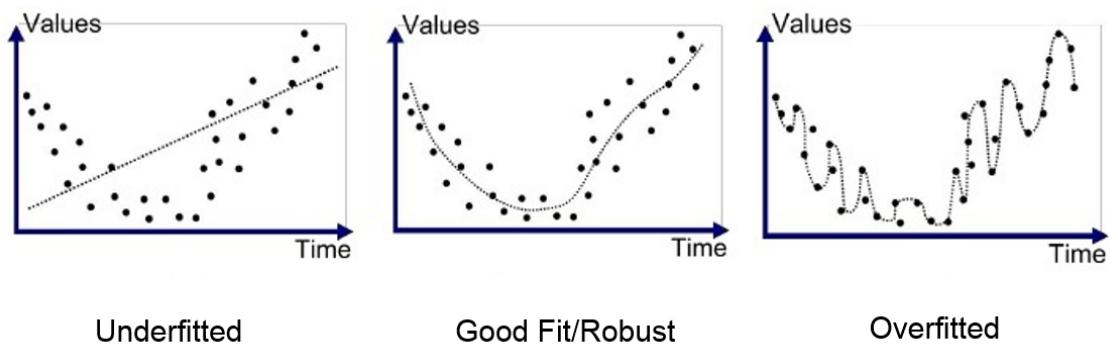
- **Learning Rate:** se conoce como *learning rate* al paso que el algoritmo de descenso del gradiente realiza en el entrenamiento de una red en cada iteración. Es un valor vital para este proceso, ya que de él depende la convergencia del problema. Un paso pequeño genera muchas iteraciones y mayores tiempos de entrenamiento mientras que un paso grande puede provocar que el algoritmo no converja nunca (figura 3.6). La elección de un correcto *learning rate* se abordará en el siguiente capítulo, donde se expone un interesante método para encontrar un valor adecuado.
- **Epochs:** consiste en las veces que el algoritmo recorrerá toda la base de datos de partida para entrenar la red.



**Figura 3.6** Efecto de malos *learning rates* [13].

Un entrenamiento puede finalizar con varios resultados posibles:

- **No convergencia:** el algoritmo es incapaz de encontrar una solución y continua iterando sin fin.
- **Convergencia:** se encuentra una solución al problema, con tres posibilidades, representadas además en la figura 3.7.
  - **Aprendizaje pobre:** la solución tiene errores de aproximación, por lo que las predicciones de la red son malas.
  - **Sobre-aprendizaje:** solución igual de problemática que la anterior. En este caso las predicciones sobre la base de datos son muy buenas, pero la capacidad de generalización de la red es pobre, haciendo que ante nuevos datos con los que no ha sido entrenada sea incapaz de realizar una clasificación acertada. Se puede conocer este efecto también con el nombre de *overfitting*.
  - **Aproximación correcta:** la red genera predicciones con mucha precisión pero a su vez ha desarrollado una capacidad de generalización que le permite predecir correctamente ante nuevos datos de entrada.



**Figura 3.7** Aprendizaje pobre, buen aprendizaje y sobre-aprendizaje [14].

Existen diversas técnicas para evitar las patologías típicas de un mal entrenamiento. Sin embargo, con una correcta elección de parámetros y una base de datos equilibrada, los algoritmos actuales son capaces

de alcanzar buenas soluciones. Aunque esto es en la teoría, en muchos casos como el tema central de este trabajo, la forma que tiene la base de datos y el problema en sí tienen un dimensionamiento complejo que requiere de un tratamiento especial. Se expondrán diversas técnicas para evitar posibles patologías en el siguiente capítulo durante la exposición del método propuesto.

### 3.4 Bases de datos

Una correcta elección de una base de datos para entrenamiento es vital para llegar hasta los resultados deseados. La red debe conocer de igual manera todos aquellos elementos que se quieren predecir, por lo que los prototipos de entrenamiento deben ser equilibrados para todas las clases a detectar y además disponer de una enorme cantidad de datos conlleva, aunque un entrenamiento más largo, soluciones muy buenas. Esto es sin embargo algo poco común en campos como el de la medicina, donde existe escasez de datos, difícil accesibilidad a los mismos y un desequilibrio importante entre clases, ya que los patrones que se quieren predecir (ya sea una enfermedad a partir de datos físicos de un paciente o una figura anormal en una imagen obtenida por técnicas médicas) son elementos aislados y poco comunes. Es por ello que la construcción de una buena base de datos es importante y existen técnicas de mejora que ponen solución a los problemas descritos.

Una de estas técnicas, la cual es casi siempre usada haya o no un problema asociado a la base de datos, es la de *Data Augmentation*. Consiste en aumentar la base de datos a partir de los prototipos conocidos, modificándolos ligeramente para generar nuevos a partir de ellos. En el ejemplo de las redes neuronales aplicadas al reconocimiento en imágenes, es común generar variaciones de los prototipos realizando rotaciones, aplicando aumentos, variando resolución y modificando el espacio de color para crear "nuevas" imágenes con las que entrenar la red. Esta técnica será detallada durante la exposición del método para este trabajo, pues su utilidad es muy alta.

Por último, generalmente los algoritmos de entrenamiento necesitan realizar una separación en la base de datos, usando una parte de ella para el entrenamiento en sí y otra para validar. La parte de validación no es usada para entrenamiento y sirve para verificar si la red realiza predicciones correctas. De igual forma, tras el proceso se predice sobre la base de datos de validación, para comparar el comportamiento frente a datos que la red "conoce" y datos que nunca ha "visto". De esta comparativa se obtienen conclusiones como la precisión de la red o si existe una pobre generalización o por el contrario un sobre-entrenamiento.

### 3.5 Transfer Learning o cómo aprovechar el entrenamiento de una red

El *Deep Learning* puede ser abordado de dos formas: mediante la creación de una arquitectura desde cero, estableciendo diferentes tipos de capas y conexiones, o mediante el aprovechamiento de una red ya existente. Esta segunda opción es quizás la más utilizada a la hora de crear aplicaciones por su mayor simplicidad.

Si se habla de redes convolucionales enfocadas a la visión artificial, existen una gran cantidad de arquitecturas ya diseñadas como AlexNet o ResNet entrenadas de forma predeterminada con un conjunto de imágenes. Están preparadas para predecir un determinado número de objetos y elementos. Sin embargo, aunque el objetivo de un problema determinado no incluya la detección de esos objetos, el conocimiento que ya tiene la red puede ser aprovechado. Volviendo a entender las capas como filtros, eliminar el conocimiento ya almacenado en las primeras capas que distinguen formas, texturas y elementos básicos sería desperdiciar información y tiempo. Es por ello que comúnmente se realiza un proceso de *Transfer Learning*, en el cual se aprovecha la información aprendida de un problema y se aplica a otro relacionado.

Generalmente el proceso conlleva un entrenamiento de las capas finales de la red o la adición de nuevas, lo que provoca una reducción del tiempo entrenando la red. Tras este proceso, normalmente los resultados son buenos si los tipos de imágenes no varían demasiado a aquellas con las que el modelo fue entrenado en su inicio, aunque con las arquitecturas actuales el proceso de *Transfer Learning* es algo a tener siempre en cuenta por sus bondades.

### 3.6 Fine Tuning

Para mejorar aún más los resultados, existe un proceso de *Fine Tuning* cuyo objetivo es ajustar los pesos de algunas de las capas intermedias de la red de forma ligera para afinar en el nuevo problema al que se aplica el *Transfer Learning*. Se inicializa el nuevo modelo a entrenar con las primeras capas "congeladas" para no variar sus pesos (al ser capas de extracción de características más genéricas) y se entrena el resto con la nueva base de datos [15][16].

Esta estrategia junto con la adición de unas capas al final de la red hace que no sea necesario entrenar una red desde cero, reduciendo tiempos y permitiendo aprovechar la información conocida por la red. Es una técnica de uso muy común y que genera muy buenos resultados, mejorando hasta un 10% las predicciones del modelo.

### 3.7 Conclusiones teóricas en relación al problema del trabajo

El objetivo del presente capítulo ha sido por tanto presentar la tecnología central del trabajo y los pilares sobre los que se apoyará el proyecto de detección de mitosis, centrándose en aquellos de los que se hará uso. Se han alcanzado en este punto las siguientes conclusiones:

- Como planteamiento inicial, una vez conocido el problema, la solución consiste en entrenar una red que sea capaz de reconocer figuras mitóticas.
- Existirán dos clases posibles en la salida de la red, que determinará si un elemento es o no mitosis.
- Dada la complejidad de las redes, se optará por hacer uso de una red unidireccional a partir de una arquitectura ya diseñada para la extracción de características en imágenes.
- Esta red será de tipo convolucional y tendrá un entrenamiento supervisado al disponerse de una base de datos de partida que será posteriormente comentada.
- Al usar una arquitectura ya creada, será necesario aplicar conceptos como *Transfer Learning* y *Fine Tuning*, por lo que el método propuesto debe contemplar estas técnicas.
- Al existir actualmente una enorme cantidad de recursos sobre el tema, una selección adecuada del material será vital para el correcto desarrollo del proyecto.



# 4 Método propuesto

---

Una vez conocida la teoría detrás del *Deep Learning* como tecnología central del trabajo, se procede a describir el método propuesto que supone una solución al problema inicial. En primer lugar se describirá todo el material que ha sido utilizado a nivel de *hardware* y *software*, así como la base de datos de partida. Dado que una parte importante del trabajo ha sido la investigación de qué material utilizar, no sólo se comentará lo usado para la versión final sino aquellas herramientas que han sido descartadas, las cuales han tenido la utilidad de evidenciar problemas a resolver y sin las cuales habría sido imposible alcanzar una solución. Se justificará por tanto el porqué de usar unas y no otras y el razonamiento seguido para finalizar con la presentación del método propuesto, detallando todos sus componentes y cómo ha sido desarrollado.

## 4.1 Material

El primer punto a tratar será el material utilizado para el proyecto, tanto *hardware* como *software*. Ambos componentes tienen una importancia mayúscula en el proyecto y conocerlos bien es una tarea previa vital para obtener los resultados deseados. Además, es de gran importancia para poder entender el porqué del método elegido, ya que como se verá más adelante existen una serie de limitaciones.

### 4.1.1 Hardware

El núcleo de funcionamiento de las librerías de *Deep Learning*, tal y como se ha comentado con anterioridad, es la Unidad de Procesamiento Gráfico (GPU). En los campos de la investigación y profesional se utilizan potentes equipos con una o varias GPU de alta gama que alcanzan elevados precios, con el objetivo de reducir los tiempos de entrenamiento de las redes así como de poder crear arquitecturas más complejas. En este caso se investigaron alternativas más económicas, que permitieran manejar redes que fueran interesantes para el problema en cuestión y que por otro lado conllevaran tiempos de entrenamiento cortos, para así poder experimentar variando parámetros hasta alcanzar los resultados deseados. La conclusión de dicha búsqueda, teniendo en cuenta los recursos de los que se disponía, fue la siguiente:

1. Hacer uso de un ordenador con una GPU de gama media. De lograrlo de esta forma no sería necesario realizar ninguna inversión, garantizando además la posibilidad de usar esta tecnología por cualquier persona en detrimento de capacidad de computación, haciendo que los tiempos de espera durante el entrenamiento de las redes neuronales fueran más largos. Otra ventaja es un mayor control del trabajo realizado al estar haciendo uso de herramientas conocidas que además se pueden utilizar en cualquier momento.
2. Aprovechar servicios de *Cloud Computing* como *Paperspace* o *Amazon Web Services*. Estos productos ofrecen potencia de computación, permitiendo contratar por tiempo limitado máquinas con soporte en la nube personalizadas. Existen numerosas alternativas de procesador, memoria y GPUs con precios razonables. La clara ventaja es la posibilidad de estar en condiciones similares a las del ámbito profesional y de la investigación.

La elección final fue la primera, al eliminar las restricciones de limitación de tiempo pese a que conllevara menor eficiencia a la hora de trabajar con redes neuronales. Esta conclusión no habría sido posible sin un buen material *software*, gracias al cual finalmente se consiguió un rendimiento excelente pese a la limitación inicial

como se expondrá en el siguiente punto. Las especificaciones técnicas del equipo final donde se desarrolló la totalidad del proyecto, incluyendo el entrenamiento de redes neuronales y su implementación vía software son las siguientes:

**Tabla 4.1** Especificaciones técnicas del equipo.

Componente	Valor
CPU	Intel Core i5-6400 @ 2.70 GHz
GPU	NVIDIA GeForce GTX 1050 Ti OC 4GB
RAM	8,00 GB
HDD	500 GB
SSD	No
OS	Windows 10

De las anteriores especificaciones es necesario señalar varios puntos:

- Todos los componentes hardware pueden catalogarse como gama media en el año 2017.
- Pese a que la GPU permite *overclocking* (aumento de la velocidad de reloj por encima de los valores establecidos por el fabricante para aumentar el rendimiento) no se ha realizado, por lo que en este sentido existe un margen de mejora respecto a los tiempos de ejecución en labores de entrenamiento.
- El sistema operativo elegido es Windows 10 por motivos de compatibilidad con los drivers de NVIDIA. Pese a que es más cómodo trabajar con sistemas basados en Linux en las tareas de desarrollo, el alto número de componentes *software* y el estado en fase *beta* de algunos de ellos a fecha de realización del proyecto ha provocado que la decisión sea en favor del sistema de Microsoft.
- La existencia de un SSD (Unidad de estado sólido), el cual permite lecturas de disco más rápidas haría que los tiempos de ejecución del software final del proyecto se redujeran por motivos que serán detallados en su respectivo apartado.

#### 4.1.2 Software

Para este proyecto se utilizan numerosos recursos *software*, los cuales serán divididos en dos grandes grupos correspondientes a las dos partes principales del trabajo: entrenamiento de la red e implementación software.

##### Entrenamiento. MATLAB® como herramienta para el Deep Learning.

El proceso de entrenamiento es el más delicado del proyecto, siendo de vital importancia elegir un método adecuado al problema. Este punto comienza con la elección de una arquitectura de red adecuada o la elaboración de una desde cero. Debido a la alta complejidad de las redes neuronales, la opción más adecuada era tomar una arquitectura ya definida y realizar un proceso de *Transfer Learning*, con el objetivo de adaptar una red existente al problema del proyecto.

En primera instancia, se intentó desarrollar el trabajo en MATLAB® versión 2018a, el cual posee interesantes complementos para trabajar con *Deep Learning* [17][18]. De esta temprana versión del proyecto se obtuvieron las siguientes conclusiones tras realizar varias pruebas:

1. **Neural Network Toolbox Model for AlexNet** [19]: este componente de MATLAB® implementa la red AlexNet, premiada en el ILSVRC 2012 con el galardón a la red con mayor rendimiento para clasificar. La arquitectura consiste en 8 capas profundas con 5 capas convolucionales y 3 capas totalmente conectadas. Está entrenada con la base de datos ImageNet [20], compuesta en el año en que se creó esta red de varias decenas de miles de imágenes de mil tipos de clases diferentes. Es típico probar las redes con esta base de datos, que a día de hoy cuenta con más de 14 millones de imágenes. Aunque es rápido entrenarla, los resultados no son tan precisos por lo que aunque no fuera a convertirse en la red elegida para este trabajo, fue interesante como primer contacto con este tipo de tecnologías.

Para ponerla a prueba y conocer sus posibilidades de cara al problema, se entrenó la última capa

de la red con una base de datos inicial de 1000 imágenes, la mitad de perros y la otra mitad de gatos, algo muy típico para practicar. Para el entrenamiento el componente selecciona arbitrariamente las imágenes que forman la base de datos de test y de validación, previa configuración del porcentaje de imágenes que irán a parar a cada una de esas bases de datos. Se estableció un 70 % para el número de imágenes del *training dataset*, por lo que las imágenes restantes se usarían para el *validation dataset*.

Los resultados obtenidos fueron bastante buenos, con un tiempo de entrenamiento de 6 minutos y una precisión del 94 %. Como primera conclusión, cabe destacar que AlexNet limita la dimensión de las imágenes a 224px x 224px, redimensionando aquellas imágenes que no cumplen con ese criterio y haciendo por tanto perder información al reducir la mayoría de ellas y obviando las diferencias en relación de aspecto. Por otro lado, pese a que los 6 minutos pueden parecer pocos, la base de datos de imágenes histológicas como se verá más adelante es mucho mayor y generaba tiempos de espera de hasta 2 horas.

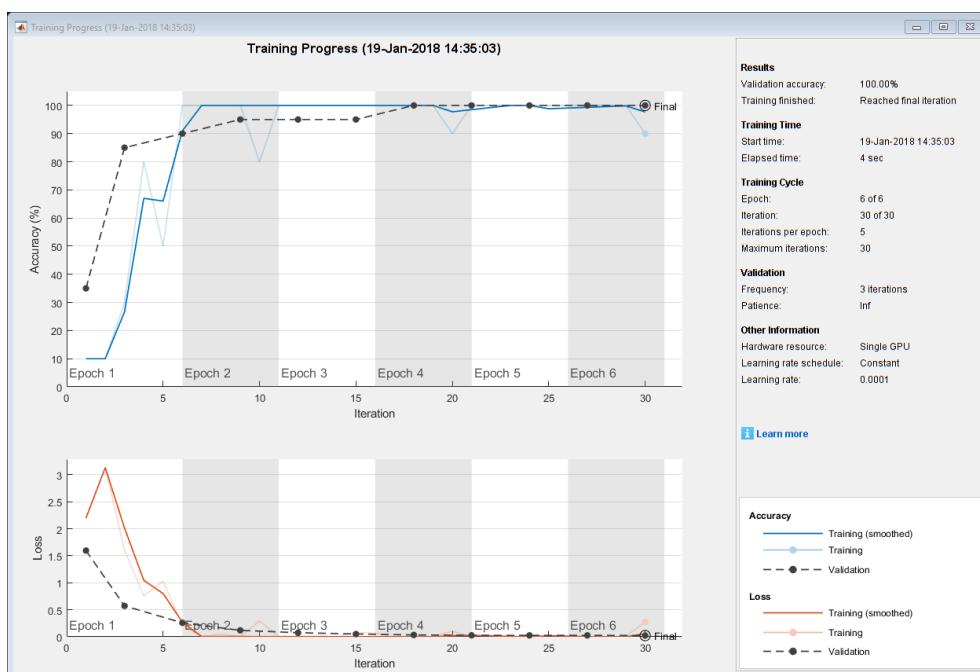
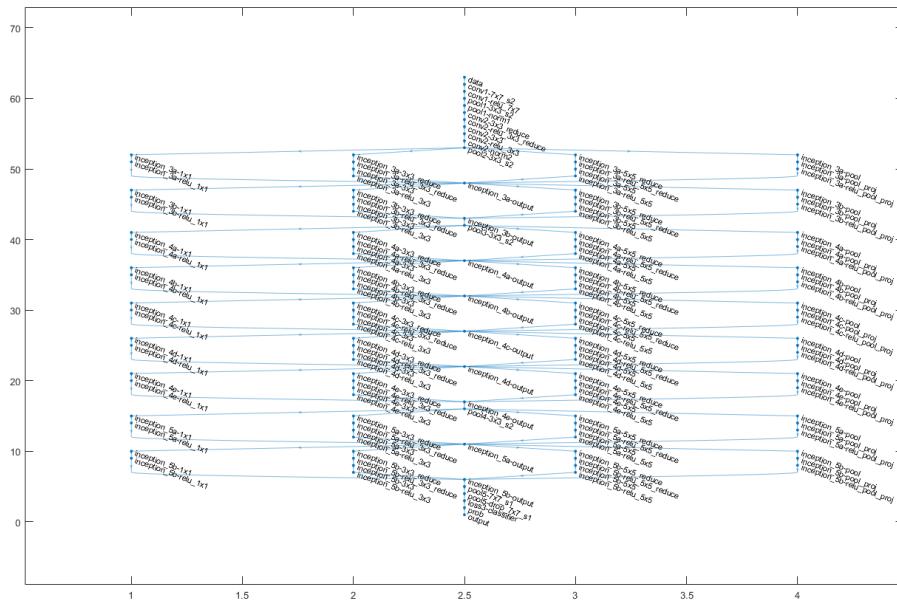


Figura 4.1 Ejemplo de entrenamiento con AlexNet.

- 2. Neural Network Toolbox Model for GoogLeNet Network [21]:** la segunda opción tenida en cuenta al usar MATLAB® fue GoogLeNet, la red creada por Google y que fue participante en el ImageNet Challenge de 2014. Al igual que AlexNet, esta red está entrenada con la misma base de datos aunque promete mayor precisión, también debido a una arquitectura más novedosa.

Es capaz de clasificar imágenes en 1000 clases diferentes. El objetivo inicial con esta red era utilizar el mismo *dataset* de prueba usado en AlexNet para comparar resultados. Finalmente se tuvo que descartar su uso por su incompatibilidad con el *hardware*, ya que el componente era incapaz de detectar la existencia de una GPU y siempre trataba de llevar a cabo el entrenamiento por CPU, lo que conllevaba tiempos de ejecución con previsión de días. Este problema puede ser debido a lo reciente que es este componente.

Del uso de MATLAB® se concluyó que quizás no era la mejor herramienta para este trabajo teniendo en cuenta las limitaciones *hardware*. En este punto del proyecto se inició una búsqueda de alguna alternativa capaz de ofrecer lo que se deseaba. Debido al gran desarrollo que el *Deep Learning* ha sufrido en los últimos años, la cantidad de opciones fuera de MATLAB® es enorme, con múltiples librerías disponibles para diversos lenguajes de programación que ofrecen una mayor flexibilidad y que se ajustaban mejor a las restricciones que se tenían. El resultado de esta investigación fue claro: **Python**.

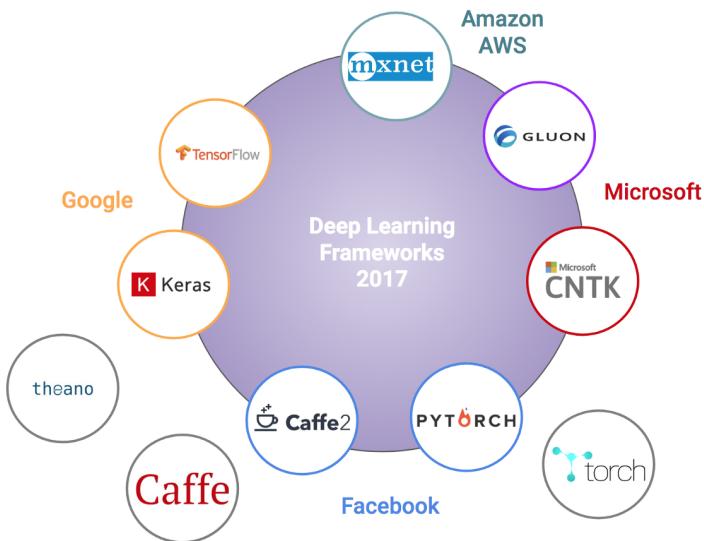


**Figura 4.2** Arquitectura de GoogLeNet.

#### Entrenamiento. Python y fast.ai.

Python es un lenguaje de programación con licencia de código abierto potente, flexible, de código interpretado y multiplataforma. En los últimos años ha vivido un enorme crecimiento gracias a todas sus virtudes, estando presente en los proyectos más importantes de *software* a nivel mundial.

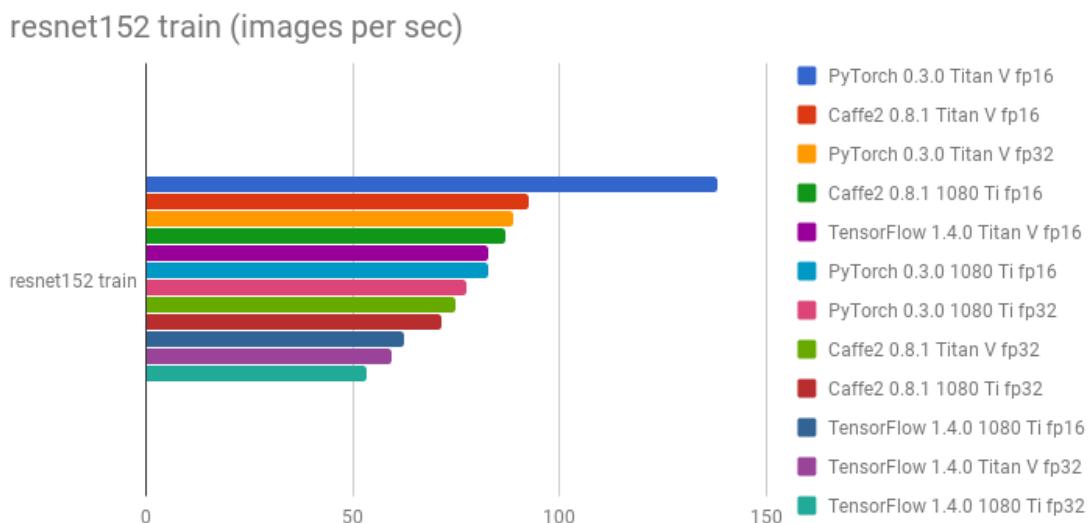
Gracias a esto, también se ha convertido en la base de muchas tecnologías de *Deep Learning* como TensorFlow y PyTorch, desarrolladas por Google y Facebook respectivamente, dotando de potencia de computación y haciendo posibles entornos ligeros. A priori estas características resultaron atractivas y parecían adaptarse a la perfección al problema del que trata este trabajo.



**Figura 4.3** Frameworks más importantes [22].

Uno de los entornos más llamativos fue **fast.ai**, el cual no sólo proporciona librerías con las que manejar datos, redes y realizar entrenamientos, sino que también funciona como punto de encuentro entre expertos y usuarios que quieren aprender, ofreciendo un flujo constante de información interesante. Además sus creadores, han elaborado un completo curso sobre *Deep Learning* y cómo hacer uso de su librería. Sus desarrolladores son Jeremy Howard y Rachel Thomas. Jeremy Howard es desarrollador, emprendedor e investigador en la Universidad de San Francisco. Es además el fundador de Enlitic, la primera compañía en aplicar el *Deep Learning* a la medicina. Rachel Thomas por otro lado es doctorada en matemáticas e ingeniera en Uber [23]. Estos datos despertaron un gran interés a la hora de realizar este trabajo, por lo que a priori fast.ai parecía la elección más adecuada. Además, es un proyecto de libre uso y sus creadores afirman que es una forma de hacer más accesible el *Deep Learning* en un mundo en el que su uso y demanda es cada vez mayor.

El siguiente paso fue instalar dicho entorno junto con todas sus librerías con el objetivo de probarlo. Como resultaba una herramienta con la que no se estaba familiarizado (sí con Python), se realizó un seguimiento de las primeras lecciones del curso, las cuales no sólo mostraron como hacer uso del entorno sino que sirvió para afianzar algunos conceptos sobre esta tecnología [24]. Fast.ai es una librería que hace más accesible el uso de PyTorch [25], el cual puede ser algo complejo debido en parte a su corta edad (primera *release* a finales de 2016), reduciendo la cantidad de código necesario para generar redes y entrenarlas. Pese a que TensorFlow es el entorno más importante y presente en los productos actuales, los creadores de fast.ai se apoyan en PyTorch por su utilidad y mayor rendimiento en el ámbito de la investigación. Durante el año 2017 el uso de PyTorch ha crecido en este campo y su aparición en artículos científicos es más que notable. Además, observando *benchmarks* realizados con las GPUs más potentes de la actualidad, PyTorch ofrece un procesamiento de imágenes por segundo mucho mayor que la competencia, como queda reflejado en la figura 4.4.



**Figura 4.4** Comparativa entre tecnologías con una red ResNet [26].

La arquitectura que fast.ai implementa por defecto es ResNet34, una red que pese a tener una mayor profundidad no hace más complejo el entrenamiento. Fue la ganadora del ILSVRC 2015 y obtuvo un error del 3.57% con el *dataset* ImageNet [27]. Al importarla viene pre-entrenada con dicha base de datos. Existen otras versiones de ResNet con 50 y 152 capas también disponibles en fast.ai.

Por último, para hacer posible su funcionamiento incluye numerosos módulos para Python enfocados a las matemáticas, al manejo de datos y al procesamiento de imagen encapsulados en un entorno virtual.

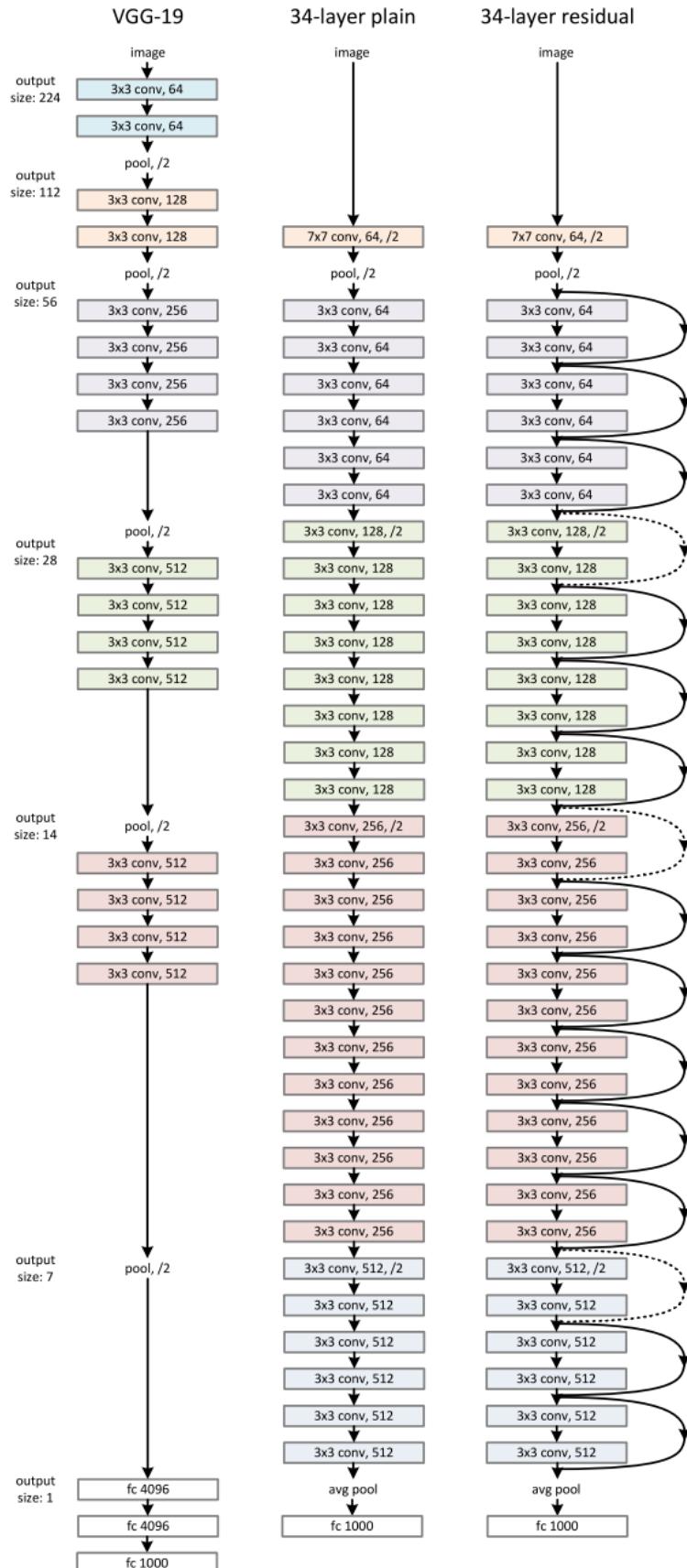


Figura 4.5 Arquitectura de ResNet34.

Al igual que en MATLAB®, se realizó una prueba inicial con fast.ai para estudiar su rendimiento y ver si podía ser de utilidad. Con la instalación incluye un *dataset* de perros y gatos y un cuaderno de Jupyter con un flujo preparado de entrenamiento. Se dejó con los valores por defecto de *learning rate* y *epochs* y realiza el entrenamiento de una capa al final de la red (última capa). El resultado fue el siguiente:

```
In [22]: arch=resnet34
data = ImageClassifierData.from_paths(PATH, tfms=tfms_from_model(arch, sz))
learn = ConvLearner.pretrained(arch, data, precompute=False)
learn.fit(.01, 2)

Epoch [██████████] 100% 2/2 [06:44<00:00, 202.37s/it]

epoch      trn_loss    val_loss   accuracy
0         0.146715   0.123397  0.94873
1         0.130819   0.120039  0.947266

Out[22]: [0.120038696, 0.947265625]
```

**Figura 4.6** Entrenamiento de ResNet34 con el *dataset* de prueba.

Con este caso de prueba se obtuvo una precisión del 94.72 % tras dos *epochs* sobre la base de datos de validación y un tiempo de entrenamiento de 6 minutos y 44 segundos. Aunque parece que no hay apenas diferencia con el uso de MATLAB®, el nuevo *dataset* contenía 23000 imágenes, algo que se asemeja más al del problema que se intenta resolver. Además, en ese tiempo se incluye un procesamiento previo de las imágenes que solo se produce en la primera ejecución del algoritmo. Al ejecutarlo otra vez, teniendo las imágenes preprocesadas en memoria, el resultado fue aún mejor, entrenando la red en 2 minutos y 26 segundos:

```
In [23]: arch=resnet34
data = ImageClassifierData.from_paths(PATH, tfms=tfms_from_model(arch, sz))
learn = ConvLearner.pretrained(arch, data, precompute=False)
learn.fit(.01, 2)

Epoch [██████████] 100% 2/2 [02:26<00:00, 73.07s/it]

epoch      trn_loss    val_loss   accuracy
0         0.149761   0.125301  0.945801
1         0.13127   0.127248  0.945312

Out[23]: [0.12724784, 0.9453125]
```

**Figura 4.7** Segundo entrenamiento de ResNet34 con el *dataset* de prueba.

Aunque el caso de la imagen médica es muy diferente y mucho más complejo que el de una simple base de datos de imágenes de animales, los buenos resultados obtenidos y las amplias posibilidades de configuración hicieron que fast.ai se convirtiera en la herramienta elegida. Además, como permite generar flujos de entrenamiento con pocas líneas de código, contribuye a la legibilidad y al mantenimiento. Finalmente, cabe destacar que fast.ai y sus cursos de aprendizaje datan de finales de 2017 e implementan, además del núcleo PyTorch, numerosos y útiles métodos fruto de investigaciones recientes en el campo de la ciencia de los datos y la computación. Esto resulta importante porque se estaría aplicando por tanto al problema del conteo de mitosis tecnología muy reciente y avanzadas técnicas, que serán descritas en detalle durante la exposición de la solución final.

### Dataset. TUPAC16 | MICCAI Grand Challenge.

El *dataset* usado para la creación del método de detección y conteo automático de mitosis es el proporcionado por la organización del Tumor Proliferation Assessment Challenge [28], un reto consistente en crear un algoritmo capaz de calcular parámetros de proliferación de tumores. Para acceder a estos datos, es necesario poseer una cuenta activada por los organizadores del reto, previa justificación del uso que se le darán a las imágenes. Se proporcionaron varios *datasets*:

- 1. Training dataset:** contiene 500 casos de diferentes pacientes de cáncer de mama procedentes de The Cancer Genoma Atlas. Cada caso es una única imagen almacenada en .svs, formato piramidal para imágenes multirresolución. Esto significa que se tiene en el mismo archivo, una imagen con distintos tamaños, siendo la más grande de la mayoría de ellas de más de 50000 x 50000 px. Cada imagen viene acompañada de una tabla .csv, la cual contiene dos columnas. La primera de ellas indica el número de

mitosis encontradas, mientras que la segunda el cálculo del valor de proliferación molecular. En total, la base de datos de imágenes es de 490 GB.

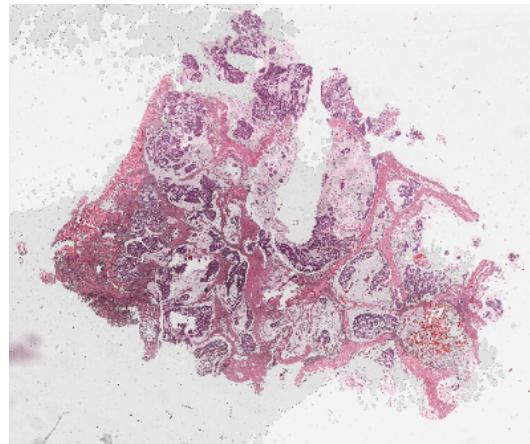


Figura 4.8 Imagen histológica típica proporcionada.

**2. Auxiliary dataset: mitoses:** contiene imágenes histológicas en formato TIFF con anotaciones realizadas por expertos del Departamento de Patología del Hospital Universitario de Utrecht, en Países Bajos. Estas anotaciones indican la posición de las figuras mitóticas. El objetivo de este *dataset* es entrenar al método diseñado de modo que sea capaz de detectar los elementos de interés. Las imágenes provienen de 73 casos diferentes de cáncer de mama. Adicionalmente fueron añadidos 50 casos más procedentes de un centro médico el cual no indica la organización del reto.

En cuanto a la base de datos, cada caso contiene múltiples imágenes correspondientes a regiones de  $2 \text{ mm}^2$  tomadas con un escáner Leica SCN400, el cual tiene un aumento de 40x y una resolución espacial de  $0.25 \mu\text{m}/\text{px}$ . Las anotaciones son fruto del consenso de al menos dos especialistas. Cada imagen viene acompañada de una tabla .csv que respeta el formato de directorios y nomenclatura de las imágenes. En su interior tienen coordenadas del punto central de cada una de las mitosis, indicado con dos números (fila y columna). Es importante señalar que estas tablas tienen por tanto un tamaño variable en función del número de mitosis de cada imagen y, en el caso de no existir ninguna, el archivo será inexistente. En el apartado sobre el desarrollo de la aplicación que implementa la red entrenada se profundizará más sobre este tema. En total el tamaño es de 10 GB.

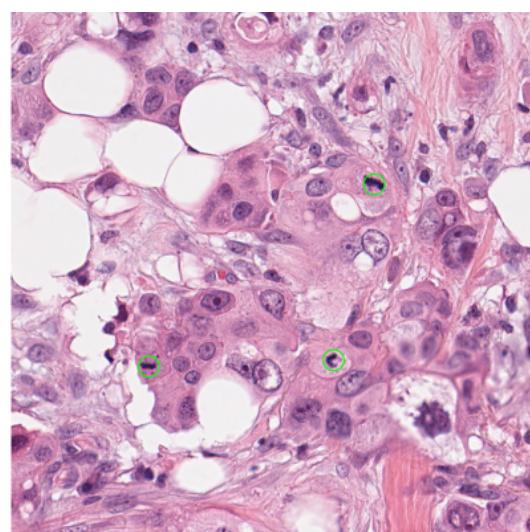


Figura 4.9 Ejemplo de imagen con anotaciones (en verde) sobre mitosis.

3. El último indica regiones de interés dentro de una imagen como las de la primera base de datos, también marcadas por patólogos expertos. Se usa para descartar zonas que no resulten interesantes, de modo que un algoritmo pudiera omitir regiones de estudio en función de su aspecto, para así reducir tiempos de procesamiento. Este *dataset* contiene 321 casos de cáncer de mama y tiene un tamaño de 345 GB.

#### Implementación. Desarrollo de la aplicación.

Tras el entrenamiento de la red ResNet34 con una base de datos, se genera un archivo contenedor de los pesos de las distintas capas de la arquitectura en formato .h5. Con el objetivo de manejar este archivo y hacer una implementación práctica del mismo, se quiso desarrollar una aplicación. El lenguaje de programación elegido fue **Python 3.6**, por las razones ya descritas con anterioridad, y las herramientas adicionales son las siguientes:

- **Spyder**[29]: entorno de desarrollo Python. Posee un visor de archivos y variables, así como integración con una consola de IPython donde realizar pruebas del código que se desarrolle. La elección de este IDE se debe en parte a su parecido en funcionalidades con MATLAB® .
- **fast.ai** [30]: librería para Python que dota de la capacidad de importar pesos de una red de neuronas entrenada a cualquier aplicación con el fin de clasificar imágenes.
- **Pillow** [31]: librería para el manejo de todo tipo de imágenes.
- **Numpy** [32]: paquete para Python de computación científica. Sus posibilidades son innumerables, permitiendo trabajar con matrices N-dimensionales además de implementar álgebra lineal, transformada de Fourier y muchas funciones relacionadas con estadística.
- **Matplotlib** [33]: librería que permite representación 2D y 3D de funciones e imágenes. Muy útil su uso en conjunto con Numpy y otras herramientas científicas como Jupyter.
- **Personalización**: aunque la aplicación final no tiene modo gráfico y funciona en el terminal, se han realizado ajustes que facilitan su uso, como la aplicación de color con **Colorama** [34] y **Termcolor** [35] y el uso de barras de progreso con **tqdm** [36].



**Figura 4.10** Esquema de las herramientas de las que hace uso la aplicación desarrollada.

## 4.2 Método

En este punto se comentará el método propuesto para la detección de mitosis y su conteo. Se desarrollará todo el proceso seguido y las reflexiones y decisiones realizadas sobre el problema inicial.

### 4.2.1 Planteamiento inicial

El primer paso que se siguió en el desarrollo del trabajo fue el entrenamiento de la red con la base de datos de partida. El objetivo era conseguir una red cuya entrada fuera capaz de tomar una imagen histológica y a su salida ofrecer una clasificación.

El primer problema se hizo evidente. Las redes de neuronas relacionan una imagen a una clase. Reflexionando sobre el tipo de imagen que se tiene y el objetivo deseado, no se tiene esa relación, sino una donde existen múltiples clases dentro de una misma imagen ya que existen zonas con figuras mitóticas y zonas donde no las hay. Es por tanto imposible realizar un entrenamiento y una clasificación a partir de las imágenes completas del tejido, por lo que el primer paso es fraccionar dichas imágenes en pequeños cuadros y clasificarlas en dos directorios separados (uno para recortes con presencia de mitosis y otro para aquellos en los que no había) para entrenar la red con fast.ai.

Otro problema que apareció en esta temprana fase del desarrollo del trabajo fue el enorme tamaño de los *datasets*, haciendo casi imposible su manejo por parte del *hardware* ni por los servicios en la nube expuestos con anterioridad. Además, la primera base de datos (Trainig dataset) no ofrecía información sobre la localización de las mitosis, sino únicamente del número de ellas. Se descartó por tanto su uso y se tomó la segunda base de datos, más fácil de utilizar y que sí poseía las posiciones de los elementos que se querían detectar.

Con estas primeras ideas se procedió a generar un *dataset* apropiado, más fácil de manejar y que cumpliera con las restricciones del problema.

### 4.2.2 Mejora del *dataset*

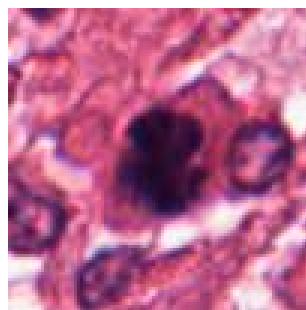
Como era necesario dividir todas las imágenes (656 en total correspondientes a 73 casos diferentes), se generó un proceso automático que lo hiciera. Previo estudio de las imágenes, se determinó que ninguna mitosis excedía un tamaño de 100 x 100 px, por lo que parecían ser unas dimensiones adecuadas para los fragmentos. Además, como las imágenes de esta base de datos son de 2000 x 2000 px, usar ese tamaño para los cuadros simplificaba el problema.

La automatización del proceso se llevó a cabo a través de un script de MATLAB® . La elección de MATLAB® fue debido a que esto se hizo durante la primera etapa del trabajo, mientras se experimentaba con sus componentes de *Deep Learning*. Este script está preparado para trabajar con la estructura de ficheros de la base de datos. El flujo que sigue es el siguiente:

1. Lectura de una imagen.
2. Verificación de si existe una tabla de coordenadas de mitosis asociada a ella.
3. Recorte de la imagen. Los fragmentos son clasificados de manera que las imágenes que poseen un píxel presente en la tabla de coordenadas van a parar a un directorio y el resto a otro.

Este código fue posteriormente mejorado y llevado a Python, ya que al no tener control sobre la posición de las mitosis, algunas aparecían partidas entre dos recortes, por lo que su utilidad para luego entrenar la red se perdía. Sin embargo, los recortes de imágenes donde no había mitosis seguían sirviendo, por lo que fueron salvados.

En este punto se evidenció otro problema, quizás el más importante de todo el proyecto. El directorio con los recortes de mitosis contenía 1552 imágenes. El directorio de recortes con imágenes sin mitosis, sin embargo, 261266.



**Figura 4.11** Ejemplo del recorte de una mitosis.

#### 4.2.3 Desbalanceo entre clases

El problema del desbalanceo de clases es algo típico en determinadas áreas del *Deep Learning*. Como se ha visto con anterioridad, un entrenamiento es mejor si se posee una base de datos equilibrada de imágenes de las diferentes clases. No obstante, en campos como la imagen médica es algo común, donde las características a detectar son normalmente pequeños elementos presentes en un gran espacio.

Se realizó un entrenamiento para probar el resultado de trabajar con este *dataset*, obteniendo muy malos resultados. Dado que el 99,56% de las imágenes componen una clase, la red resultante siempre clasificaba los recortes como pertenecientes a este grupo. Esto es peor incluso si se tiene en cuenta que con los resultados finales se calculará un coeficiente F1 que valora la precisión de la clasificación, dando mayor peso a los verdaderos positivos, que en este caso son las imágenes con mitosis que son bien clasificadas.

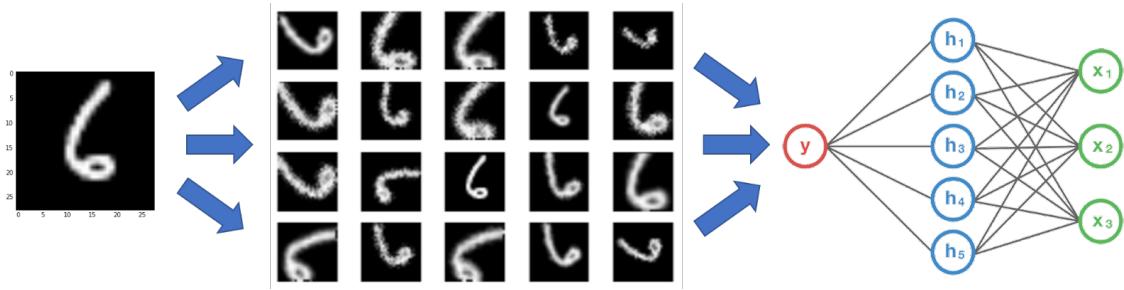
En este punto era evidente que había un problema de dimensionamiento, por lo que se centraron los esfuerzos en estudiar técnicas aplicadas en este tipo de casos [37][38]. Son muchas las técnicas posibles y cada vez surgen nuevas, pero las de más interés fueron las siguientes:

- **Clonado de datos:** una solución sería tan simple como realizar copias de la clase con menor número de elementos. Esto es útil si el desbalanceo es leve, pero en el caso de las mitosis sería necesario generar muchas copias (168 en concreto).  
El tener muchas imágenes similares genera, en *Deep Learning*, el problema de *overfitting*. Esto ocurre cuando una red es capaz de clasificar muy bien las imágenes con las que se entrena, pero incapaz de hacerlo correctamente con imágenes que no ha "visto" nunca. Una de las causas del *overfitting* se debe a que la red, durante el entrenamiento, al procesar muchas veces las mismas imágenes, en vez de reflejarse en los pesos aspectos genéricos de las características a detectar, se reflejan detalles muy concretos que imposibilitan que una imagen nueva y diferente sea bien clasificada.
- **Reducción de la clase mayoritaria:** es posible eliminar elementos de la clase con mayor número de ellos hasta llegar a un equilibrio entre clases.
- **Data Augmentation:** quizás la técnica más utilizada y la que mejores resultados da. Consiste en procesar cada una de las imágenes, generando por cada una de ellas N variaciones. Estas nuevas imágenes se crean siguiendo varios principios, como voltear la imagen original, rotarla, aumentarla o modificar el espacio de color, como puede verse en la figura 4.12. Para hacer uso de esta técnica, es importante reflexionar sobre el tipo de imágenes que la red posteriormente clasificará. Por ejemplo, volviendo al ejemplo de la base de datos de imágenes de perros y gatos, no va a aportar nada positivo añadir una variación con la imagen rotada 180°, ya que a la hora de clasificar una nueva imagen rara vez será de un animal al revés.

#### Solución propuesta.

Para el caso de detección de mitosis se optó por unir las dos últimas técnicas, descartando la primera por el problema que podría causar de *overfitting*. A partir de este momento, se denominará *non-mitosis* a la clase donde no hay mitosis y *mitosis* a la clase donde si hay presencia de ellas.

Por un lado reducir la clase mayoritaria haría más manejable este directorio. Además, si se observan las imágenes donde no hay mitosis, muchas de ellas son muy similares. Por otro lado, se consideró que aplicar

Figura 4.12 Ejemplo de *data augmentation*.

*data augmentation* sería de gran utilidad para conseguir "nuevas" imágenes de mitosis. Se tomó como 10 el número de variaciones que existirían de cada recorte de mitosis, generando un total de 15520 imágenes.

1. **Reducción de la clase *non-mitosis*:** mediante el diseño de un script Python, se seleccionaron los 15520 prototipos de la clase *non-mitosis*. Esta selección fue totalmente aleatoria y cada recorte seleccionado se guardaba en un directorio que alojaría el *dataset* definitivo. A cada recorte se le asignó un nombre compuesto por una cadena de letras y números aleatorios.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Sat Mar 31 10:46:52 2018
4
5 Autor: Alejandro Suárez Lamadrid
6 Título: non_mitosis_selector .py
7
8 """
9
10 from PIL import Image as image
11 import glob
12 import uuid
13 from random import randint
14
15 # Directorio de la clase "non-mitosis"
16 path = 'C:\\Users\\Alejandro\\Documents\\MATLAB\\TFG\\mdataset2\\no\\**'
17 path_folder = glob.glob(path, recursive=True)
18
19 # Directorio donde guardar el dataset resultante
20 save_path = 'C:\\Users\\Alejandro\\Desktop\\TFG\\dataset\\non-mitosis\\'
21
22 # Número de imágenes a seleccionar
23 dataset_num = 15520
24 dataset_len = len(path_folder)
25
26 # Creación del nuevo dataset
27 for i in range(dataset_num):
28
29     random_file = randint(1, dataset_len) # Recorte aleatorio
30     imagen = path_folder[random_file]
31     im = image.open(imagen)
32
33     # Almacenado con nombre aleatorio
34     unique_filename = save_path + str(uuid.uuid4()) + '.jpg'
35
36     im.save(unique_filename)
```

Listing 4.1 Script para reducir la clase *non-mitosis*.

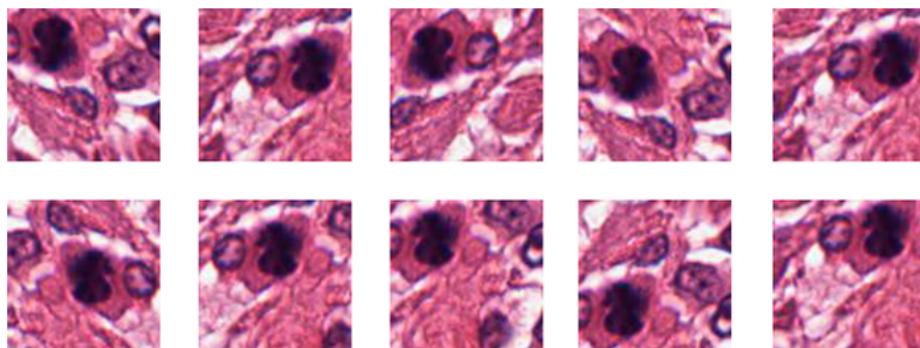
Gracias a este proceso se aseguró una clase que tendría el mismo número de elementos que la clase *mitosis*, que como se verá en el siguiente punto se aumentará hasta los 15520 elementos. Por otro lado, dado que a priori se desconoce cómo selecciona fast.ai las imágenes para el entrenamiento, asignando a cada archivo un nombre aleatorio se mezclan recortes de distintos tejidos, dotando de mayor aleatoriedad al proceso. Esto enriquece el entrenamiento posterior.

**2. Aumento de la clase mitosis por *Data Augmentation*:** partiendo de los 1552 recortes de mitosis de los que se disponía entonces, se quiso generar 10 variaciones para cada uno de ellos. Se escogió esa cifra para evitar posibles casos de *overfitting* y por ser un valor típico en procesos de este tipo.

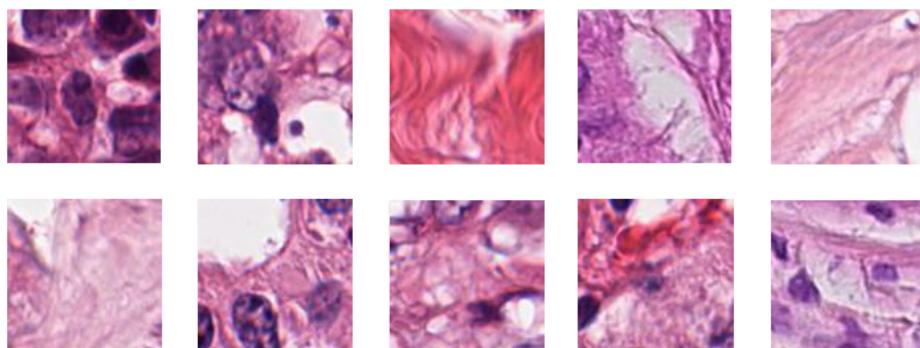
El proceso de *Data Augmentation* se hizo en dos fases. Mientras que fast.ai incorpora técnicas para ello durante el entrenamiento, en un primer lugar se deseó realizar fuera de este entorno para observar cómo serían los resultados y llevar un control del trabajo. Además, el *Data Augmentation* que fast.ai implementa aumenta todas las clases, sin permitir seleccionar una sola de ellas. Es por esto que se creó un script en Python cuyo flujo es el siguiente:

- a) En primer lugar, y del mismo modo que el script inicial que fue desarrollado en MATLAB® para la creación del *dataset*, se localizan las mitosis en las imágenes de partida.
- b) Para cada mitosis, se generan 10 recortes de la misma desplazando la ventana de recorte de 100 x 100 px ligeramente (un máximo de 20 px) en cualquier dirección y de forma arbitraria.
- c) Finalmente, cada uno de los 10 recortes se gira de forma aleatoria, aplicando rotaciones en sentido horizontal, vertical o ambos a la vez. La misma probabilidad hay de que se rote en ningún sentido, que en cualquiera de ellos o en ambos.

Con todo este proceso se consigue un *dataset* de 31040 imágenes, 15520 en cada una de las clases. Se ha logrado por tanto igualar el número de imágenes de cada clase permitiendo un entrenamiento más apropiado, priorizando la riqueza en imágenes de mitosis por ser aquellos elementos que se desea detectar.



**Figura 4.13** Mitosis tratada con técnicas de *data augmentation*.



**Figura 4.14** Ejemplos de elementos en la base de datos sin figuras mitóticas.

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Mar 9 17:45:06 2018
4
5 @author: Alejandro
6 """
7
8 import csv
9 import glob
10 import os.path
11 from random import randint
12
13 from PIL import Image as image
14
15
16 # Directorios donde se tienen los datos de partida
17 ground_path = 'C:\\Users\\Alejandro\\Documents\\MATLAB\\TFG\\Mitosis\\ground_truth\\**'
18 ground_folder = glob.glob(ground_path, recursive=False)
19
20 mitosis_path = 'C:\\Users\\Alejandro\\Documents\\MATLAB\\TFG\\Mitosis\\tejidos\\**'
21 mitosis_folder = glob.glob(mitosis_path, recursive=False)
22
23 # Directorio donde se almacenará el dataset
24 save_path = 'C:\\Users\\Alejandro\\Desktop\\dataset\\'
25
26 # Contadores de mitosis
27 cont = 0
28 mitosis = 0
29
30 for i in range(len(mitosis_folder)):
31
32     # Exploración de los datos de partida
33     mitosis_act = glob.glob((mitosis_folder[i] + "\\**"), recursive=True)
34     ground_act = glob.glob((ground_folder[i] + "\\**"), recursive=True)
35
36     # Tratamiento de cada imagen
37     for j in range(len(mitosis_act)):
38
39         if(j == 0):
40             continue
41
42         # Se preparan las rutas de las referencias
43         im = image.open(mitosis_act[j])
44
45         check = mitosis_act[j]
46         check = check.replace(".tif", ".csv")
47         check = check.replace("tejidos", "ground_truth")
48
49         # Se verifica que existe referencia para una imagen
50         if(os.path.isfile(check)):
51             archivo = open(check)
52             tabla = csv.reader(archivo)
53             tabla = list(tabla)
54             archivo.close()
55             mitosis = len(tabla)
56
57         # Si no hay referencia , no existe mitosis en esa imagen
58         else:
59             mitosis = 0
60
61         # Tratamiento de la imagen
62         for k in range(mitosis):
63
64             s = im.size[1]
65             x = int(tabla[k][0])
66             y = int(tabla[k][1])
67
68             # Para cada imagen se generan 10
69             for l in range(10):
70                 # Desplazamientos aleatorios
71                 randx = randint(20, 80)
```

```

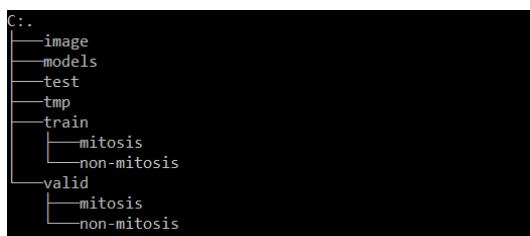
72 randy = randint(20, 80)
73 # Rotaciones aleatorias
74 randtd = randint(0, 1)
75 randlr = randint(0, 1)
76
77 # Generación de la bounding box para recorte de la mitosis
78 box = (max(0, y - randy), max(0, x - randx), min(s, y + (100 - randy)),
79             min(s, x + (100 - randx)))
80 imcrop = im.crop(box)
81
82 # Rotaciones del recorte
83 if (randtd):
84     imcrop = imcrop.transpose(Image.FLIP_TOP_BOTTOM)
85 if (randlr):
86     imcrop = imcrop.transpose(Image.FLIP_LEFT_RIGHT)
87
88 # Se almacenan los recortes en el directorio del dataset
89 save_name = save_path + 'mitosis' + '_' + str(i) + '_' + \
90             str(j) + '_' + str(k) + '_' + str(l) + '.jpg'
91 imcrop.save(save_name)

```

**Listing 4.2** Script para *data augmentation* de la clase *mitosis*.

#### 4.2.4 Selección de candidatos para entrenamiento y validación

Teniendo la base de datos lista para entrenamiento, es necesario previamente crear una estructura de directorios apropiada para fast.ai, de acuerdo a la propia documentación de esta librería. Se añadieron además otros directorios útiles para la aplicación final, con el objetivo de mantener todos los recursos en un mismo lugar.



**Figura 4.15** Estructura de directorios para el entrenamiento.

- **models:** contendrá el archivo de pesos .h5 generado tras el entrenamiento y del que se podrá hacer uso para clasificar imágenes.
- **train:** en su interior se aloja la base de datos de entrenamiento de todas las clases en directorios separados.
- **valid:** contiene las imágenes con las que no se entrena la red y que se usarán para validar su eficacia clasificando.
- El resto de directorios son usados por la aplicación final y se describirán posteriormente.

El siguiente paso fue por tanto seleccionar qué imágenes formarían parte de cada directorio. Es importante tener en cuenta las restricciones que existían ya que seleccionar aleatoriamente imágenes para entrenamiento y validación no era una opción. Si se hacía de esta manera, dado que en el directorio con figuras mitóticas cada una de ellas está presente 10 veces, se podría estar entrenando la red con mitosis con las que luego también se validaría, perdiendo totalmente la utilidad de validar la red. Es por esto que extrajeron 100 mitosis (con sus respectivas 10 variaciones cada una) para la validación del modelo. De esta forma se entraría la red con unas mitosis y se validaría con otras.

Por otro lado, como en el proceso de reducción de la clase *non-mitosis* los recortes sufrieron una selección y nombre aleatorios, la separación en directorios de entrenamiento y validación se realizó sin seguir ningún criterio especial. En total, y para mantener la igualdad con la otra clase, se escogieron 1000 imágenes para validación y el resto para entrenamiento.

El resultado de este proceso, y a modo de resumen del *dataset* definitivo que se utilizó para entrenar y validar el modelo final del proyecto fue el siguiente:

1. **Training dataset (train)**: directorio del cual fast.ai toma las imágenes con las que entrena el modelo ResNet34.
  - **mitosis**: 14520 imágenes, las cuales corresponden a recortes de 1452 mitosis diferentes a las que se les ha aplicado un proceso de *data augmentation* que genera 10 variaciones de cada una.
  - **non-mitosis**: 14520 imágenes, que corresponden a recortes de tejido sin presencia de figuras mitóticas seleccionados de forma aleatoria de todos los casos iniciales.
2. **Validation dataset (valid)**: directorio del cual fast.ai toma las imágenes con las que valida el modelo tras el entrenamiento, indicando aquellas que ha clasificado bien y las que no.
  - **mitosis**: 1000 imágenes que corresponden a 100 mitosis diferentes a aquellas con las que se realiza entrenamiento.
  - **non-mitosis**: 1000 imágenes de recortes de tejido sin presencia de mitosis, que se usan para validar el modelo cuando indica que en una zona no existen figuras mitóticas.

#### 4.2.5 Entrenamiento del modelo. Última capa

Para entrenar el modelo con la base de datos se escogió utilizar la herramienta Jupyter, un proyecto web con el que crear, visualizar y compartir documentos que contienen código ejecutable. La capacidad de lanzar bloques de código independientes y visualizar resultados sobre la marcha hacen de Jupyter un recurso útil para este trabajo. A continuación se detalla todo el proceso seguido.

En primer lugar, se importan los elementos necesarios de la librería fast.ai así como activar el uso de matplotlib por parte de Jupyter para la presentación de resultados.

```
Jupyter plots.

In [1]: %reload_ext autoreload
%autoreload 2
%matplotlib inline

Fastai imports.

In [2]: from fastai.imports import *

In [3]: from fastai.transforms import *
from fastai.conv_learner import *
from fastai.model import *
from fastai.dataset import *
from fastai.sgdr import *
from fastai.plots import *
```

**Figura 4.16** Recursos necesarios para el entrenamiento.

Se indica a fast.ai la localización del dataset, el tamaño en píxeles de las imágenes si se desea y se comprueba si las tecnologías CUDA y CuDNN están habilitadas.

```
Variable PATH con el contenido del dataset y tamaño de las imágenes a procesar. El tamaño está relacionado con la arquitectura de la red.

In [4]: PATH = "C:/Users/Alejandro/Desktop/TFG/dataset_final/"
sz=100

Las siguientes funciones permiten conocer la disponibilidad de la tecnología CUDA de Nvidia. También habilita CuDNN que mejora el rendimiento en el entrenamiento.

In [5]: torch.cuda.is_available()
out[5]: True

In [6]: torch.backends.cudnn.enabled
out[6]: True
```

**Figura 4.17** Recursos necesarios para el entrenamiento.

A continuación se entrena una última capa de la red, perteneciente a las Fully connected layers, con la base de datos indicada. Al método fit de fast.ai se le indican el *learning rate* y las *epochs* deseados. La elección de estos valores son posteriormente justificados. Los resultados fueron los siguientes:

```
In [22]: arch=resnet34
data = ImageClassifierData.from_paths(PATH, tfms=tfms_from_model(arch, sz))
learn = ConvLearner.pretrained(arch, data, precompute=False)
learn.fit(0.01, 2)

Epoch 100% 2/2 [08:45<00:00, 262.87s/it]

epoch      trn_loss    val_loss   accuracy
0          0.404434   0.298156   0.869141
1          0.386694   0.294293   0.871094

Out[22]: [0.2942934, 0.87109375]
```

**Figura 4.18** Entrenamiento de la última capa del modelo.

De la figura anterior se extraen varias conclusiones:

- El tiempo de entrenamiento (con procesamiento previo de las imágenes incluido) fue de 8 minutos y 45 segundos. Este dato reafirma la elección de herramientas realizada.
- Los valores de pérdidas indican errores cometidos en la predicción por parte del modelo. En este caso están indicados por las variables *trn\_loss* (*training loss*) y *val\_loss* (*validation loss*). Es decir, fast.ai entrena el modelo y comprueba su validez clasificando las imágenes de entrenamiento y las de validación. El objetivo es que estos dos valores sean, no sólo lo más reducidos posible, sino también parecidos. Un caso en el que las pérdidas sobre la *training dataset* fueran muy reducidas y las del *validation dataset* más altas sería indicador de *overfitting* o cómo la red es capaz de clasificar bien las imágenes con las que se entrena pero no otras. En este caso, al ser valores muy similares parece que no hay presencia de este problema.
- La precisión del modelo clasificando imágenes alcanza el 87.1 %, un valor que resultó bastante alto para tratarse de imágenes médicas y no de objetos comunes, pero que aún tiene margen de mejora.

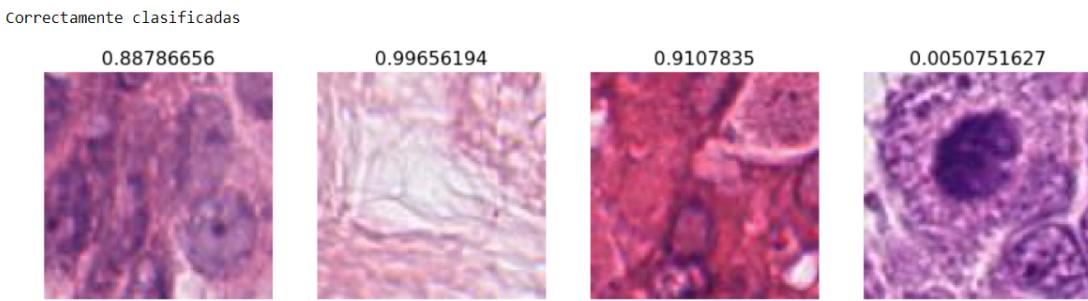
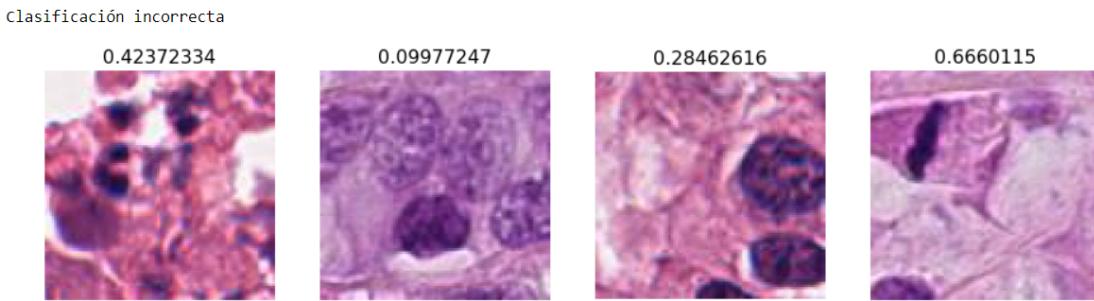
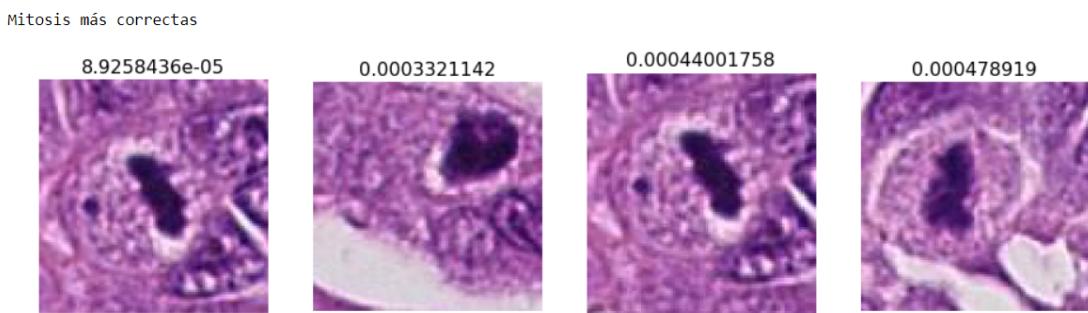
El siguiente paso es realizar con la red entrenada una clasificación del *validation dataset* completo. Para cada imagen el modelo asigna una clase, las cuales numera como 0 y 1 en este caso. Una imagen clasificada como 0 será aquella con presencia de mitosis, mientras que indicará 1 si no la hay. Sin embargo, fast.ai no sólo indica esta clase sino la cercanía a pertenecer a ella. Así, una imagen clasificada con un valor cercano a 0 tendría mayor probabilidad de pertenecer a la clase *mitosis* mientras que un valor cercano a 1 indicaría clase *non-mitosis*.

```
In [24]: data.classes
Out[24]: ['mitosis', 'non-mitosis']

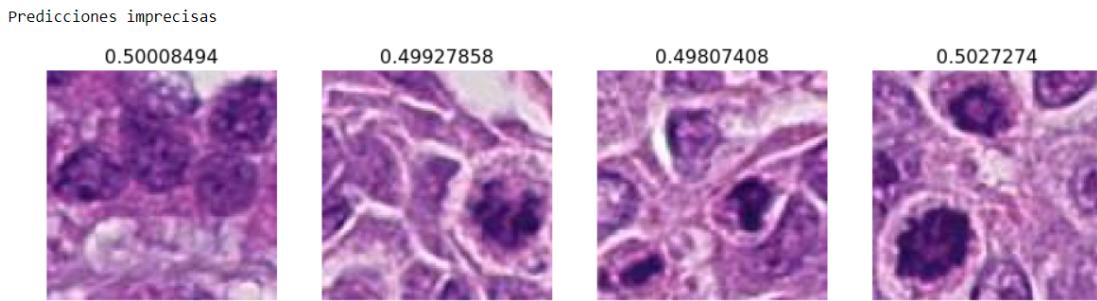
In [25]: log_preds = learn.predict()
log_preds.shape
Out[25]: (2000, 2)
```

**Figura 4.19** Clasificación de los datos para validación.

Gracias a estos valores se tiene de algún modo una puntuación para cada imagen, a partir de la cual establecer un umbral para determinar cuáles corresponden o no a mitosis. Por defecto este umbral se posiciona en el punto medio 0.5, generando los siguientes resultados:

**Figura 4.20** Ejemplos de predicciones correctas.**Figura 4.21** Ejemplos de predicciones incorrectas.**Figura 4.22** Ejemplos de mitosis con mayor puntuación.**Figura 4.23** Ejemplos de clase sin mitosis con mayor puntuación.

Estas figuras muestran bastante bien lo que la red entrenada es capaz de hacer. En un principio parece que tras el entrenamiento sabía diferenciar las clases y lo que claramente eran mitosis y lo que no. Sin embargo, el entrenamiento podía ser más exhaustivo para que la red arrojara mejores predicciones y con mayor certeza, ya que en este momento existían determinadas imágenes con clasificaciones erróneas e imprecisas, es decir, con puntuaciones cercanas al punto medio entre ambas clases.



**Figura 4.24** Ejemplos de predicciones imprecisas.

#### 4.2.6 Mejora del entrenamiento. Técnicas

Con el objetivo de aumentar la precisión y reducir las pérdidas, fast.ai incorpora útiles técnicas de mejora del entrenamiento. Algunas de ellas resultan de gran utilidad para este trabajo en concreto.

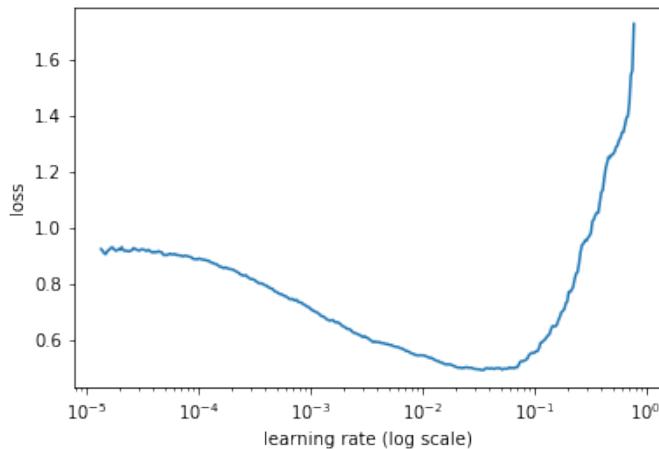
##### Learning Rate Finder.

Función que permite encontrar el *learning rate* más apropiado para cada problema. Entendiendo la solución del problema como el mínimo tras realizar un descenso del gradiente, este parámetro indica el paso que da el algoritmo en cada iteración en su intento de encontrar dicho mínimo. Elegir un valor adecuado es vital, ya que tomando uno demasiado alto podría hacer que el algoritmo no convergiera nunca y uno muy bajo derivaría en tiempos de ejecución demasiado altos.

Esta función inicia el proceso de entrenamiento e incrementa el paso en cada iteración, deteniendo el proceso en el momento en el que la solución empeora (las pérdidas comienzan a aumentar). De este modo se tendría el mayor paso que se puede dar. Sin embargo, para evitar posibles problemas, se toma un valor algo menor. El objetivo de esta técnica no es sólo hacer un entrenamiento más rápido y efectivo, sino también evitar tener que probar diferentes valores.

```
In [40]: learn = ConvLearner.pretrained(arch, data, precompute=False)
In [41]: lr_f=learn.lr_find()
A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.
83% | 376/454 [00:37<00:07, 9.90it/s, loss=2.04]
```

**Figura 4.25** Ejecución del algoritmo y su detención en el momento en el que las pérdidas aumentan.



**Figura 4.26** Evolución de las pérdidas a medida que aumenta el *learning rate*.

El valor elegido finalmente para el *learning rate* fue de 0.01, punto cercano a aquel en el que comienzan a aumentar las pérdidas en la gráfica presente en la figura 4.26.

### Fast.ai Data Augmentation.

La librería ofrece técnicas configurables de *data augmentation*, con las que se puede aplicar pequeños aumentos y rotaciones a las imágenes. Para este caso, se ha aplicado un aumento de hasta el 10% y rotaciones aleatorias a las imágenes de ambas clases, generando 6 nuevos prototipos por cada recorte. Este nuevo uso de *data augmentation* con dichos parámetros no generó *overfitting* como se puede observar en los resultados obtenidos:

```
In [45]: tfms = tfms_from_model(resnet34, sz, aug_tfms=transforms_side_on, max_zoom=1.1)

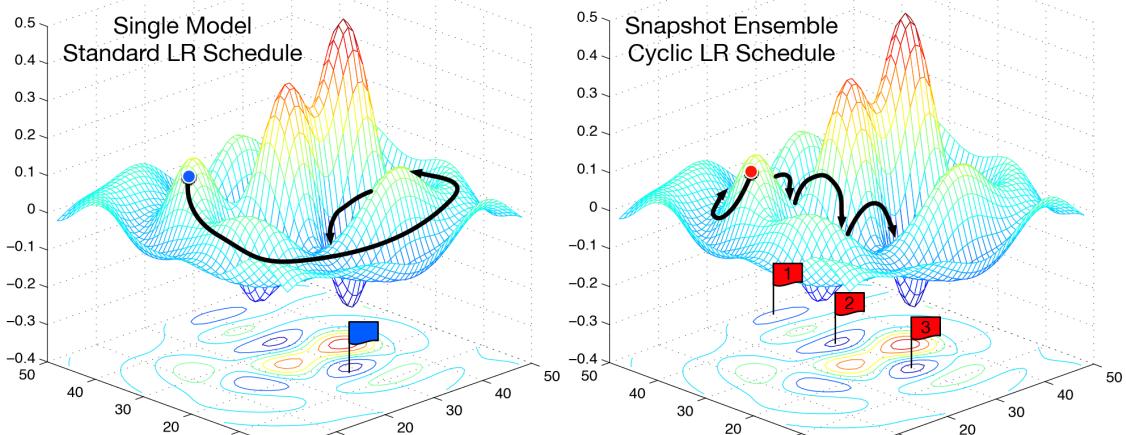
In [46]: def get_augs():
    data = ImageClassifierData.from_paths(PATH, bs=2, tfms=tfms, num_workers=1)
    x, _ = next(iter(data.aug_dl))
    return data.trn_ds.denorm(x)[1]
```

**Figura 4.27** Función de *data augmentation*.

### Descenso del gradiente con reinicios.

El entrenamiento que fast.ai ofrece no mantiene el *learning rate* fijo, sino que lo va reduciendo a medida que se producen iteraciones, con el objetivo de afinar la búsqueda del mínimo. Sin embargo, es posible que el primer mínimo que el algoritmo encuentre sea poco flexible. La flexibilidad en este caso se entiende como la capacidad del modelo de mantenerse preciso y estable ante pequeños cambios en los pesos de la red.

Es por esto que es común utilizar algoritmos con reinicios, en los que el *learning rate* disminuye hasta que se encuentra un mínimo. En este punto el proceso no se detiene, sino que se reinicia de nuevo el paso al valor inicial. De este modo, si el mínimo encontrado se encuentra en un área poco flexible del espacio de pesos, saldrá de ella y se buscará un nuevo mínimo, tal y como queda reflejado en la gráfica de la figura 4.28. Se puede entender una zona como poco flexible viendo el problema como una superficie en la que encontrar mínimos, y el mínimo que se encuentra está localizado en una zona "puntiaguda".



**Figura 4.28** Descenso del gradiente con reinicios.

Para este trabajo, se eligieron inicialmente 3 ciclos de reinicio, con el siguiente resultado obtenido, que además se almacenó como la última capa del modelo definitivo. Como puede observarse, se eleva además la precisión levemente alcanzando un 89.74 %.

```
In [52]: learn.fit(1e-2, 3, cycle_len=1)
A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

epoch      trn_loss    val_loss   accuracy
0         0.379521   0.275181   0.883301
1         0.375426   0.270136   0.891113
2         0.369126   0.26499    0.897461

Out[52]: [0.26498994, 0.8974609375]
```

**Figura 4.29** Entrenamiento de la última capa con esta técnica.

### Fine-Tuning.

Finalmente y para mejorar aún más los resultados se aplica un proceso de *fine-tuning* para adaptar los pesos del resto de capas del modelo al problema en cuestión, aumentando su capacidad de extracción de características sin la necesidad de crear una nueva red desde cero. Además, reemplaza la capa de salida con la anteriormente creada. Esta capa adicional se entrena de nuevo, obteniendo las ventajas de haber adaptado el resto de la red.

Para ejecutar el algoritmo es necesario "descongelar" el modelo, ya que por defecto PyTorch lo bloquea. Este proceso de *fine-tuning* requiere además de un especial cuidado, ya que las primeras capas de la red tienen un propósito de extracción de características más genéricas y se desea evitar que sean destruidos dichos pesos. Es por ello que se aplica la siguiente técnica.

### Learning rate diferencial, imagen médica y entrenamiento final.

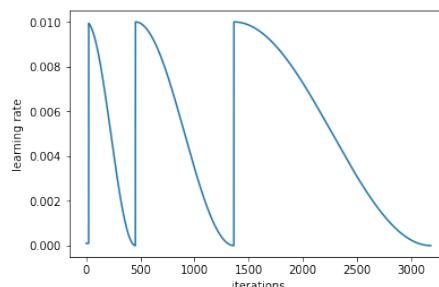
Como no se desea modificar demasiado los pesos de las primeras capas de la red, se utiliza un *learning rate* diferencial, el cual se basa en una función del coseno. El objetivo es tener diferentes pasos para diferentes capas. El efecto que tiene es una menor variación de los pesos de las primeras capas al realizar un menor número de iteraciones. Sin embargo es importante tener cuidado con esta técnica, ya que una elección errónea de parámetros puede generar malos resultados.

```
In [57]: lr = 1e-2
lrs=np.array([lr/9,lr/3,lr])
```

**Figura 4.30** Diferentes *learning-rates* para diferentes capas.

La decisión de tomar los valores de *learning rate* de la figura anterior está justificada por el tipo de problema del que trata el trabajo. Tal y como se indica en la documentación de fast.ai, la elección de este parámetro depende de cada caso, siendo recomendables esos valores en problemas relacionados con la imagen por satélite y médica [24]. Estos campos son muy similares al presentar imágenes sin objetos definidos y más abstractas, por lo que el entrenamiento requiere de un cuidado especial y unos parámetros determinados.

Finalmente se ejecuta el proceso de *fine-tuning* con los parámetros anteriormente indicados y 2 epochs, lo que resulta en un total de 6 reinicios del *learning-rate*, que esta vez tiene la siguiente forma fruto de la última técnica expuesta.



**Figura 4.31** *Learning-rate* diferencial con reinicios.



# 5 Resultados e implementación software

Tras realizar el entrenamiento final del modelo con los parámetros y herramientas descritos en el capítulo anterior, se procede a analizar los resultados obtenidos y a explicar el desarrollo de la aplicación capaz de realizar predicciones sobre imágenes de entrada.

## 5.1 Resultados del entrenamiento. Validación

Como resultado del último entrenamiento aplicando todas las técnicas ya descritas se consiguió una notable reducción de las pérdidas sobre los dos *datasets* y un considerable aumento de la precisión, que alcanzó un valor máximo del 98.39 %.

```
In [58]: learn.fit(lrs, 3, cycle_len=1, cycle_mult=2)

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

epoch    trn_loss    val_loss    accuracy
0       0.108017   0.053999   0.98291
1       0.072684   0.053672   0.978027
2       0.046977   0.041217   0.983887
3       0.040486   0.046912   0.981934
4       0.023887   0.043026   0.983887
5       0.018157   0.045967   0.983398
6       0.01892    0.044617   0.983887

Out[58]: [0.044616796, 0.98388671875]
```

**Figura 5.1** Resultado del último entrenamiento tras aplicar *fine-tuning*.

Las mejoras son importantes respecto al entrenamiento de únicamente la última capa, teniendo una red capaz de clasificar correctamente casi la totalidad de las imágenes.

Para validar se ha usado además una técnica de TTA o *test time augmentation*, que consiste en aplicar *data augmentation* al *validation dataset*, consiguiendo esta vez 4 imágenes a partir de cada una. El resultado de precisión alcanzado fue del 98.5 %. A continuación se expone una tabla comparativa de los distintos entrenamientos llevados a cabo:

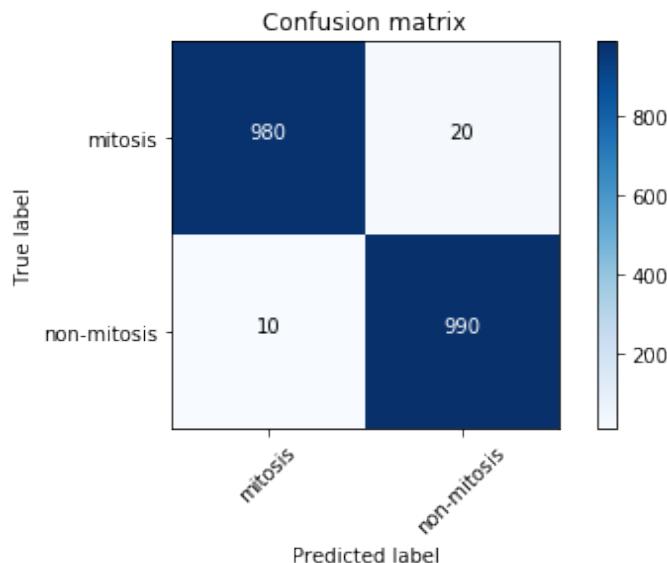
**Tabla 5.1** Comparativa de entrenamientos.

Método	Parámetros	Pérdidas	Precisión
lastlayer	lr=0.01; epochs=2	0.2943	87.11 %
lastlayer	lr=0.01; restarts=3; Data augmentation; LR con reinicios	0.265	89.74 %
lastlayer+fine-tuning	lr=differential; restarts=3; cycles=2; LR diferencial; LR con reinicios	0.0446	98.5 %

A diferencia de la primera exposición de resultados al entrenar únicamente la última capa, para plasmar mejor la capacidad de predicción del modelo generado se representa una matriz de confusión, útil herramienta para en este caso observar el diagnóstico en la forma de:

- **Verdaderos positivos:** imágenes clasificadas como mitosis que sí lo son realmente.
- **Falsos positivos:** imágenes clasificadas como mitosis donde realmente no la hay.
- **Verdaderos negativos:** imágenes clasificadas con ausencia de mitosis y que efectivamente no contienen figuras mitóticas.
- **Falsos negativos:** imágenes clasificadas como carentes de mitosis pero en las que en realidad sí hay alguna presente.

Del total de 2000 imágenes del *validation dataset*, se obtuvo la siguiente matriz de confusión:

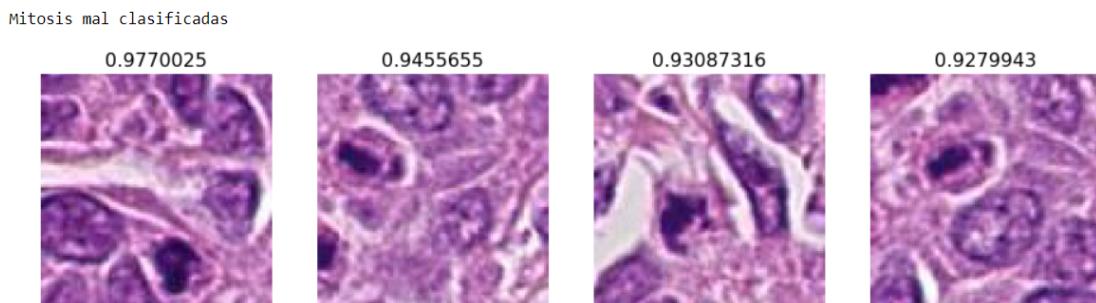


**Figura 5.2** Matriz de confusión.

De esta figura se extraen las siguientes conclusiones:

- De las 1000 imágenes de mitosis, fueron bien clasificadas 980. Posteriormente se verán algunas que no fueron bien clasificadas.
- De las 1000 imágenes sin presencia de figuras mitóticas, fueron bien clasificadas 990, por lo que en las 10 restantes se detectó mitosis sin haberla.

Con estos resultados, solamente queda observar las predicciones incorrectas para su estudio. Dado que la detección de mitosis es el objetivo más importante y principal, son las que se representan únicamente.



**Figura 5.3** Mitosis mal clasificadas.

Viendo las mitosis mal clasificadas en la figura 5.3, todas presentan un tamaño bastante más pequeño a las presentadas con anterioridad, pudiendo esto ser la causa de una clasificación errónea. A pesar de ello, los resultados fueron muy buenos y supusieron una mejora sustancial respecto a otros entrenamientos, por lo que este fue el modelo elegido para la implementación software final.

Con estos datos de la matriz de confusión, se elaboró un script en Python que calcula el Valor-F (F-Score), medida de la precisión que tiene una prueba. Se calcula con los valores estadísticos de precisión y exhaustividad.

$$F1 = 2 * \frac{Precision * Exhaustividad}{Precision + Exhaustividad}$$

$$Precision = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosPositivos}$$

$$Exhaustividad = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosNegativos}$$

$$F1 = 0.9849$$

Con la red ya entrenada y comprobados los resultados que arrojaba, se quiso desarrollar una aplicación capaz de recibir la imagen histológica de un tejido y analizarla para localizar todas las mitosis que hubiera en ella. El desarrollo de este software y los resultados finales del proyecto se detallan en el siguiente punto.

## 5.2 Aplicación para la detección de mitosis

La implementación de la red entrenada se ha realizado con una aplicación desarrollada en Python 3.6, la cual se concibió para cumplir los siguientes requisitos:

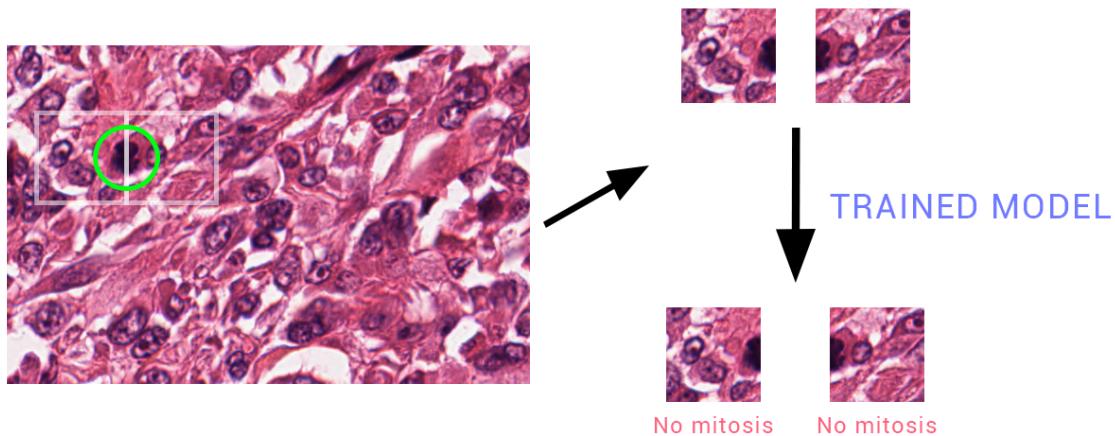
1. Analizar una imagen histológica de entrada y ofrecer a la salida una vista de dicha imagen con la localización de las mitosis encontradas y un conteo de las mismas.
2. Ofrecer una vista en detalle de cada mitosis para su estudio de forma individual.
3. Capacidad de seleccionar una o varias imágenes de entrada para ser procesadas en orden.

### 5.2.1 Exploración de la imagen

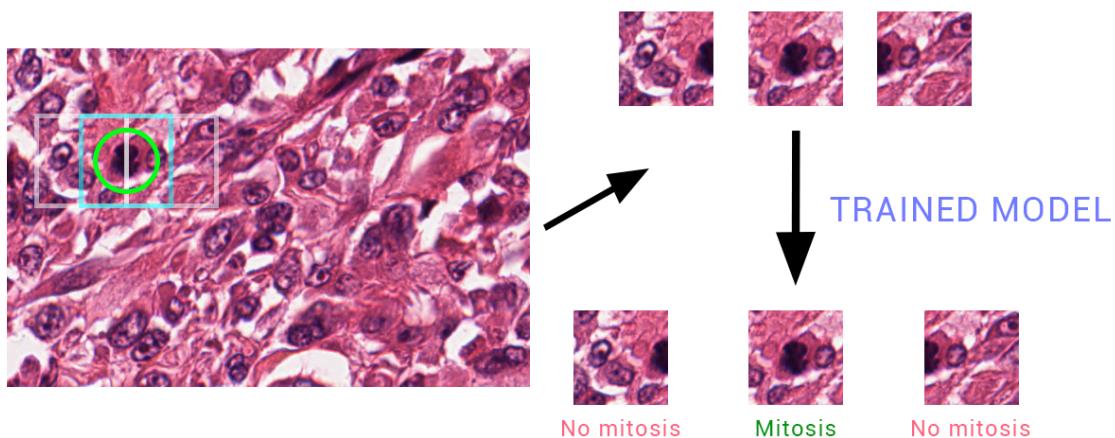
Para realizar dichas tareas, es necesario tener en cuenta que la red no fue entrenada para clasificar imágenes completas, sino pequeños bloques de una misma, por lo que la aplicación tiene que basarse en este mismo principio. Es por ello que el primer paso al cargar la imagen en el programa sería realizar una división en bloques. Sin embargo, si se hace de esta forma, es muy probable que algunas mitosis queden divididas entre varios recortes, lo que podría resultar en una mala clasificación por parte de la red, que no está viendo el elemento completo. Para solucionar este problema, se decidió realizar una exploración por bloques de la imagen pero estableciendo un cierto solapamiento entre ellos, de modo que si en una de las ventanas de exploración una mitosis no se veía completamente, en la siguiente sí.

La ventana será un cuadrado de 100 x 100 px para mantener el tamaño de imagen con el cual la red fue entrenada y realizará una exploración comenzando en la esquina superior izquierda de la imagen, la cual corresponde al píxel (0, 0). Esta exploración será en el sentido horizontal de la imagen. El solapamiento será del 50 %, es decir, la ventana se desplaza cada vez 50 px en sentido horizontal y al llegar al final de la imagen continuará hacia abajo desplazándose otros 50 px, de modo que el solapamiento es en ambas direcciones, evitando dejar alguna mitosis fuera. Se justifica este funcionamiento tras haber realizado numerosas pruebas, concluyendo en los esquemas mostrados en las siguientes figuras.

Se decidió por tanto trabajar de esta forma, ya que se evitaban errores pese a que el procesamiento de una imagen aumentara considerablemente su tiempo de duración al tener que clasificar mayor número de cuadros. En las figuras 5.4 y 5.5 se ponen de manifiesto estas ideas.



**Figura 5.4** Posible exploración sin solapamiento donde no se detecta una mitosis (señalada en verde).



**Figura 5.5** Exploración con solapamiento donde se detecta la mitosis (señalada en verde).

### 5.2.2 Predicción

La siguiente cuestión que hubo que tratar fue el de la clasificación de estos cuadros, ya que por defecto el método que utiliza fast.ai no es el más adecuado. En primer lugar se desarrolló la aplicación con el objetivo de obtener todos los cuadros primero, almacenarlos en un directorio y realizar una llamada a la función de predicción. Esta función devolvería una lista de valores con las puntuaciones de cada clasificación, pero se detectó que el orden en el que daba estos valores era totalmente arbitrario y sin indicar qué valor correspondía a qué cuadro, variando en cada ejecución. Este es un problema documentado y comentado en los foros de fast.ai y requeriría de modificaciones a nivel del núcleo de la librería.

Se optó en este momento por tanto de realizar las predicciones de una en una. Conocida cómo es la exploración de la imagen se puede saber fácilmente el valor dado por la clasificación en cada momento. Cabe destacar que para realizar la predicción la imagen en cuestión debe estar almacenada en el disco, eligiendo para este caso el directorio **test** anteriormente visto en la estructura de directorios para el entrenamiento.

### Función predict\_mitosis

Función que explora una imagen histológica por bloques de 100 x 100 px solapados un 50% y realiza una predicción sobre cada uno de ellos, determinando si contiene o no una figura mitótica. Almacena los resultados en *numpy arrays*, un tipo de variable de tipo matriz que implementa la librería **Numpy**.

Para cada bloque, la función predict de fast.ai arroja un valor de clasificación el cual determina la cercanía a cada una de las dos clases que se tienen. Se establece el umbral de decisión en 0.01, siendo por tanto clasificadas como mitosis aquellos bloques con una puntuación inferior a dicho valor, y clasificando como no mitosis los que lo superen.

La función recibe:

- **f**: imagen del tejido completo cargada en memoria con la librería Pillow.
- **learn**: modelo entrenado de red neuronal cargado en memoria con la librería fast.ai.
- **box\_num**: número de bloques que la función tendrá que analizar.
- **text\_path**: ruta del directorio donde se almacenan temporalmente los bloques para su clasificación.

A su salida devuelve:

- **mitosis\_table**: tabla con los bloques de exploración donde ha sido encontrada una mitosis. Estos bloques están descritos por las coordenadas de 2 de los puntos que los forman.
- **mit\_t**: tabla con los bloques donde se han encontrado mitosis, descritos por un número.
- **probs\_table**: tabla con las puntuaciones fruto de la predicción. Solamente se almacenan aquellas que corresponden a mitosis, es decir, las que se encuentran por debajo del umbral establecido.

---

```

1 def predict_mitosis (f, learn , box_num, test_path):
2
3     mitosis_table = np.array ([]).reshape (0, 4) # Tabla que contendrá las posiciones de las mitosis encontradas
4     mit_t = np.array ([]) # Numeración de cada mitosis
5     probs_table = np.array ([]) # Tabla con los valores de predicción
6     cont = 0
7     umbral = 0.01
8
9     # Bucle de exploración de la imagen que implementa una barra de progreso
10    for i in tqdm(range(box_num[0]), ascii =True):
11        for j in range(box_num[1]):
12            cont = cont + 1
13            box = (50 * j, 50 * i, 50 * j + 100, 50 * i + 100)
14
15            # Recorte de la imagen y almacenamiento
16            imcrop = f.crop(box)
17            save_name = test_path + "temp.jpg"
18            imcrop.save(save_name)
19            imcrop.close ()
20
21            # Predicción sobre el fragmento actual
22            log_preds = learn .predict ( is_test =True)
23            log_preds.shape
24            probs = np.exp(log_preds [:, 1])
25
26            # Umbral establecido para determinar si una figura es mitosis o no
27            if (probs [0] < umbral):
28                box = np. asarray ([ box ])
29                mitosis_table = np. vstack ([ mitosis_table , box ])
30                mit_t = np.append(mit_t, cont)
31                probs_table = np.append(probs_table , probs [0])
32
33    return ( mitosis_table , mit_t, probs_table )

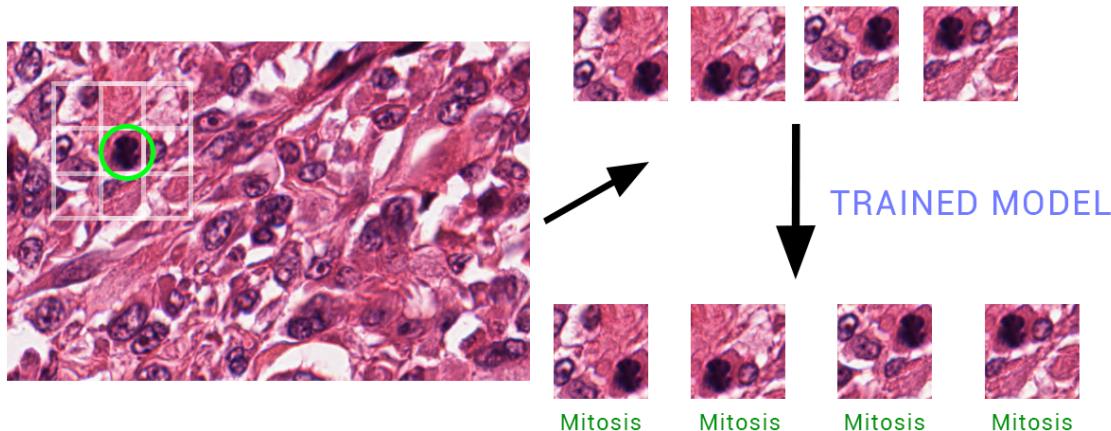
```

---

**Listing 5.1** Función para la exploración y clasificación de imágenes histológicas..

### 5.2.3 Detección múltiple de mitosis

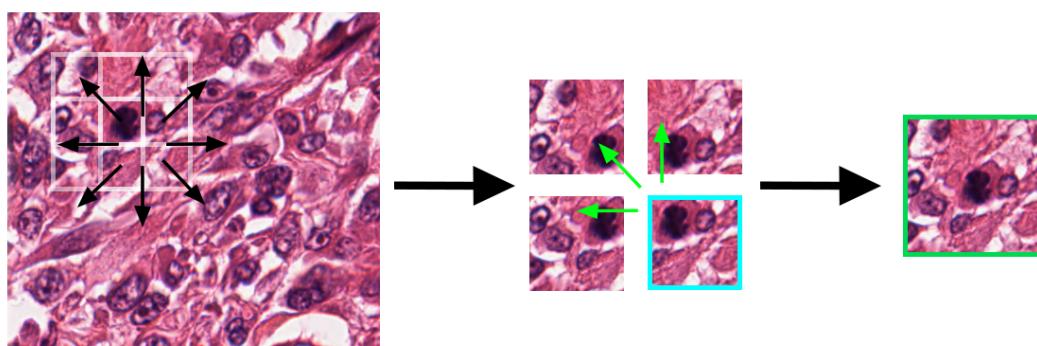
La exploración con solapamiento entre bloques conlleva un problema, y es que al igual que permite detectar elementos que pudieran quedar fragmentados, puede generar que un mismo elemento sea encontrado varias veces. Con el solapamiento elegido es de hecho la situación más frecuente. Por ello fue necesario desarrollar una función que, a partir de la tabla de coordenadas de las mitosis encontradas, detectara aquellos bloques que hacían referencia a una misma figura mitótica. Una vez detectado este fenómeno, que se realiza por cercanía entre bloques, sería necesario eliminar los que estuvieran varias veces o unificarlos, como se muestra en la figura 5.6. Esta última fue la solución elegida, realizando además una media entre las puntuaciones de predicción de cada uno de los bloques a fusionar.



**Figura 5.6** Exploración con solapamiento donde se detecta la misma mitosis varias veces.

#### Función cluster\_box

El algoritmo diseñado para la unión de cuadros se basa en buscar si algún bloque alrededor del mismo se encuentra también en la tabla. En caso afirmativo, se determina que los cuadros afectados hacen referencia al mismo elemento y se unen generando un cuadro mayor. Se añaden las coordenadas que definen este bloque procesado a una nueva lista definitiva que contiene la localización de todas las mitosis, generando una entrada para cada una.



**Figura 5.7** Algoritmo de fusión de cuadros que hacen referencia a la misma mitosis.

La función recibe:

- **mitosis\_table**: tabla con las coordenadas que describen los bloques donde se han encontrado mitosis.
  - **mit\_t**: tabla con los bloques donde fueron encontrados mitosis, numerados.
  - **probs\_table**: tabla con las puntuaciones que la función de predicción de fast.ai otorga a cada bloque. Solo incluye las de aquellos bloques donde hay mitosis.

A su salida devuelve:

- **bb**: tabla con las coordenadas de los bloques donde hay mitosis ya procesados. Aquellos cuadros que inicialmente contenían las mismas mitosis son fusionados creando un cuadro mayor.
  - **p**: tabla con las puntuaciones de predicción otorgadas por fast.ai procesadas. Aquellas puntuaciones de bloques que contenían las mismas mitosis son calculadas de nuevo como la media de todos los valores afectados.

```

1 def cluster_box ( mitosis_table , mit_t, probs_table ):
2
3     bb = np.array ([]).reshape (0, 4) # Tabla que contendrá los bloques procesados
4     p = np.array ([]) # Tabla que contendrá las predicciones procesadas
5     mit_num = mit_t.shape
6
7     # Bucle de procesamiento de bloques contenedores de mitosis
8     for u in range(mit_num[0]):
9         # Bloques alrededor del bloque de estudio
10        comp = np.array ([ mit_t[u], mit_t[u] - 39, mit_t[u] + 39, mit_t[u] - 1,
11                          mit_t[u] + 1, mit_t[u] - 38, mit_t[u] - 40, mit_t[u] + 40,
12                          mit_t[u] + 38])
13
14        # Búsqueda de bloques vecinos
15        searching = np.in1d(mit_t, comp)
16        v = np.where(searching==True)
17        row_idx = np.asarray(v)
18
19        cluster = np.squeeze( mitosis_table [row_idx[:, None]])
20
21        # Fusión de bloques vecinos y cálculo de la nueva predicción
22        if ( cluster .ndim == 2):
23            cl = np.array ([ cluster[:, 0].min(axis=0), cluster[:, 1].min(axis=0),
24                            cluster[:, 2].max(axis=0), cluster[:, 3].max(axis=0)])
25            pl = np.mean(probs)
26        else :
27            cl = cluster
28            pl = probs[0]
29
30        cl_list = np.ndarray . tolist (cl)
31        bb_list = np.ndarray . tolist (bb)
32
33        # Comprueba si el bloque ya existe en la lista definitiva de mitosis
34        if ( cl_list not in bb_list ):
35            bb = np.vstack ([bb, cl ])
36            p = np.append(p, pl)
37
38    return(bb, p)

```

**Listing 5.2** Función para el procesamiento de mitosis encontradas.

### 5.2.4 Funciones adicionales

A las funciones anteriores que describen el funcionamiento principal de la aplicación fueron añadidas dos más que resultan de interés y sirven para realizar comprobaciones iniciales e implementar un menú de usuario respectivamente.

#### Función cuda\_enabled

Esta función es llamada al iniciar la aplicación y comprueba si las tecnologías CUDA y cuDNN están disponibles y habilitadas para su uso. Ambas son necesarias para la correcta importación de la librería fast.ai y por lo tanto para el funcionamiento normal del programa.

No recibe ningún parámetro de entrada.

A su salida devuelve:

- **enabled**: variable booleana cuyo valor es True en el caso de que ambas tecnologías estén disponibles y False en caso contrario.

---

```

1 def cuda_enabled():
2
3     # Inicialización de la aplicación
4     print('Iniciando ...')
5
6     # Se comprueba si CUDA y cuDNN están habilitadas
7     cuda = torch.cuda.is_available()
8     cudnn = torch.backends.cudnn.enabled
9
10    enabled = cuda and cudnn
11    # Impresión del estado
12    if(enabled):
13        color = 'green'
14    else:
15        color = 'red'
16        print('Se necesita CUDA y cuDNN')
17
18    print('**CUDA state: ' + colored(str(cuda), color, attrs=['bold']))
19    print('**cuDNN state: ' + colored(str(cudnn), color, attrs=['bold']))
20
21    return (enabled)

```

---

**Listing 5.3** Función que comprueba la disponibilidad de las tecnologías CUDA y cuDNN.

#### Función show\_menu

Muestra en el terminal un menú con varias opciones a elegir por el usuario. Facilita el manejo del programa y sus opciones son:

1. Analizar el directorio de almacenamiento de imágenes por completo, el cual se encuentra al mismo nivel de directorios que el programa principal en la carpeta **images/**.
2. Analizar una imagen en específico del mismo directorio anterior, a seleccionar en una lista.
3. Salir de la aplicación.

El único parámetro que recibe esta función es la ruta del directorio de imágenes, ya que la utiliza para listar la totalidad de archivos en caso de seleccionar la segunda opción.

A su salida devuelve:

- **option**: elección por parte del usuario.
- **index**: elección de una imagen concreta por parte del usuario.
- **error**: código de error en el caso de uso indebido o error cometido por el usuario.

---

```

1 def show_menu(image_folder):
2     option = 0
3     index = 0
4     error = 0
5
6     # Menú principal
7     print ('\x033[4;30;47m' + '\n----- Detección de mitosis con CNN -----' + '\x033[0;m')
8     print (' Elija una opción: ')
9     print ('\x033[1;36m' + '    1. Analizar toda la carpeta imágenes.' + '\x033[0;m')
10    print ('\x033[1;36m' + '    2. Analizar una imagen específica .' + '\x033[0;m')
11    print ('\x033[1;31m' + '    3. Salir .' + '\x033[0;m')
12
13    option = int(input('Opción deseada: '))
14    print ('')
15
16    # Primera opción, análisis completo
17    if(option == 1):
18        print ('Analizando toda la carpeta ... ')
19
20    # Segunda opción. Se muestra un nuevo menú con todos los archivos y permite elegir uno
21    elif(option == 2):
22        for i, elem in enumerate(image_folder):
23            print ((i + 1), elem)
24        index = int(input('\nElija un archivo de la lista : '))
25
26        if(index > len(image_folder)):
27            error = 1
28            print ('ERROR')
29
30    # Tercera opción. Salir de la aplicación.
31    elif(option == 3):
32        print ('Finalizando ... ')
33    else :
34        error = 1
35        print ('ERROR')
36
37    return(option, index, error)

```

---

**Listing 5.4** Función que muestra el menú principal de la aplicación.

### 5.2.5 Presentación de los resultados

A la salida de la anterior función ya se tienen los resultados finales del método en forma de tablas con las coordenadas de los cuadros donde se localizan las mitosis detectadas y la puntuación que el algoritmo de predicción ofrece. El paso final es presentar dichos resultados para su visualización, para lo cual se ha usado el módulo Matplotlib.

#### Función draw\_bb

Muestra por pantalla la imagen completa y, a partir de las tablas de coordenadas que describen los cuadros donde existen figuras mitóticas, los dibuja sobre la imagen para señalárlas. También coloca sobre cada cuadro el número correspondiente a cada mitosis otorgado por el algoritmo de clasificación, modificado de modo que su cercanía a pertenecer a la clase mitosis se invierte, es decir, la clase mitosis ahora sería valorada como 1 y no mitosis como 0. Se ha hecho de esta forma por resultar más intuitivo para el usuario.

La función recibe:

- **f**: variable que contiene la imagen elegida por el usuario cargada en memoria.
- **bb**: tabla con las coordenadas que describen la situación de las mitosis encontradas.
- **p**: tabla con las puntuaciones correspondientes a cada mitosis otorgada por la función de clasificación.

A su salida no devuelve ninguna variable, pero muestra por pantalla la imagen completa con los elementos indicados previamente.

```
1 def draw_bb(f, bb, p):
2
3     # Número de mitosis encontradas como título para la figura
4     bb_num = bb.shape
5     title = 'Número de posibles mitosis encontradas: ' + str(bb_num[0])
6
7     # Recorrido de las tablas para insertar cuadros sobre la imagen
8     fig, ax = plt.subplots(1)
9     ax.imshow(f)
10    ax.set_title(title)
11    for u in range(bb_num[0]):
12        rect = patches.Rectangle((bb[u][0], bb[u][1]), (bb[u][2] - bb[u][0]),
13                                (bb[u][3] - bb[u][1]), linewidth=2, edgecolor='b',
14                                facecolor='none')
15        ax.add_patch(rect)
16        ax.text(bb[u][0], bb[u][1] - 40, str(1 - round(p[u], 5)), fontsize=7,
17                color='white', bbox={'facecolor': 'black', 'pad': 4})
18
19    # Se muestra la imagen por pantalla
20    plt.show()
21
22    return
```

---

**Listing 5.5** Función para mostrar los resultados finales.

#### Función draw\_mitosis

Una vez mostrada la imagen completa con las mitosis señaladas, el usuario puede ver una vista en detalle de cada figura mitótica por separado para su análisis.

Los parámetros de entrada y salida son exactamente los mismos que la función anterior debido al fin de dicha función.

```
1 def draw_mitosis(f, bb, p):
2
3     bb_num = bb.shape
4     ind = m.ceil(m.sqrt(bb_num[0]))
5
6     # Para cada mitosis, se muestra por pantalla su cuadro correspondiente
7     # en una única figura
8     for i in range(bb_num[0]):
9         ax = plt.subplot(ind, ind, i + 1)
10        cut = f.crop(bb[i])
11        ax.set_title(str(1 - round(p[i], 5)))
12        ax.imshow(cut)
13
14    plt.show()
```

---

**Listing 5.6** Función para mostrar los resultados finales en detalle.

#### 5.2.6 Función principal

Una vez expuestas todas las funciones que hacen posible el funcionamiento de la aplicación, solo queda mostrar cómo es la función principal que se ejecuta para hacer uso del programa. Esta función importa todas las librerías necesarias, verifica que todos los requerimientos se están cumpliendo, carga en memoria la red entrenada, muestra el menú de usuario y ejecuta las funciones descritas anteriormente según la opción elegida.

Esta función no tiene parámetros de entrada ni salida, sino que se trata de un script que incluye el uso de todas las funciones descritas con anterioridad. De este modo el programa principal tiene un código más limpio y permite un mejor mantenimiento, dividiendo y separando el núcleo de funcionamiento de la aplicación en diversas funciones con cometidos bien definidos.

---

```

1 #Import
2 from funciones import *
3
4 #Paths
5 # Directorio raíz
6 path = os.path.dirname(__file__)
7 # Directorio del modelo entrenado
8 cnn_path = "C:\\Users\\Alejandro\\Desktop\\TFG\\mitosis\\dataset_final\\"
9 # Directorio de imágenes
10 image_path = "C:\\Users\\Alejandro\\Desktop\\TFG\\mitosis\\image\\"
11 # Directorio de test
12 test_path = "C:\\Users\\Alejandro\\Desktop\\TFG\\mitosis\\dataset_final\\test\\"
13
14 #Lista de archivos
15 image_folder = os.listdir(image_path)
16 image_folder_s = len(image_folder)
17
18 #Creación de archivo temporal
19 temp = open('C:\\Users\\Alejandro\\Desktop\\TFG\\mitosis\\dataset_final\\test\\temp.jpg', 'w+')
20 temp.close()
21 init(convert=True)
22
23 sz = 100
24 option = 0
25
26 if(cuda_enabled()):
27
28     print ('\nCargando pesos de la red ...')
29     arch = resnet34
30     data = ImageClassifierData.from_paths(cnn_path, tfms=tfms_from_model(arch, sz), test_name = 'test')
31     learn = ConvLearner.pretrained(arch, data, precompute=False)
32     learn.load('224_all')
33
34 while(option != 3):
35     option, index, error = show_menu(image_folder)
36     if(error == 0):
37         if(option == 1):
38             for i in range(image_folder_s):
39                 # Apertura de la imagen
40                 f = image.open(image_path + image_folder[i])
41                 size = f.size
42                 box_num = list(map(lambda x: int(x / 50 - 1), size))
43                 # Obtención de mitosis
44                 print ('Procesando ' + str(image_folder[i]) + ' ')
45                 mitosis_table, mit_t, probs_table = predict_mitosis(f, learn, box_num, test_path)
46                 # Generación de las bounding boxes
47                 bb, p = cluster_box(mitosis_table, mit_t, probs_table)
48                 # Resultados
49                 draw_bb(f, bb, p)
50                 draw_mitosis(f, bb, p)
51
52             f.close()
53
54     elif(option == 2):
55         # Apertura de la imagen
56         f = image.open(image_path + image_folder[index - 1])
57         size = f.size
58         box_num = list(map(lambda x: int(x / 50 - 1), size))
59         # Obtención de mitosis
60         print ('Procesando ' + str(image_folder[index - 1]) + ' ')
61         mitosis_table, mit_t, probs_table = predict_mitosis(f, learn, box_num, test_path)
62         # Generación de las bounding boxes
63         bb, p = cluster_box(mitosis_table, mit_t, probs_table)
64         # Resultado
65         draw_bb(f, bb, p)
66         draw_mitosis(f, bb, p)
67
68     f.close()

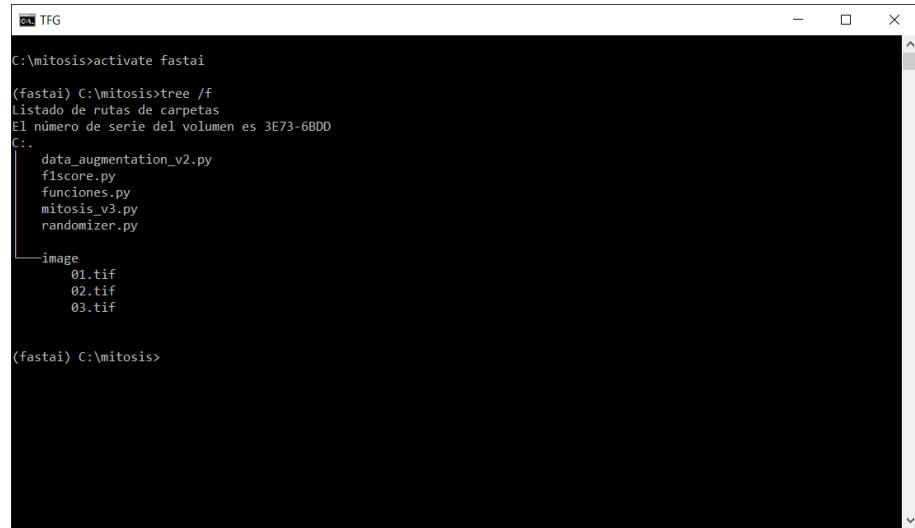
```

**Listing 5.7** Función principal para la aplicación.

### 5.3 Resultados finales. Casos de interés

Por último, solo queda mostrar los resultados obtenidos con el uso de la red entrenada así como de su implementación en forma de aplicación.

En primer lugar, es necesario activar el entorno virtual que fast.ai crea y donde están instaladas todas las librerías que requiere y usar la consola para establecer el directorio actual en la carpeta donde se encuentra alojado todo el código y las imágenes que se van a usar para poner a prueba la red y la aplicación.



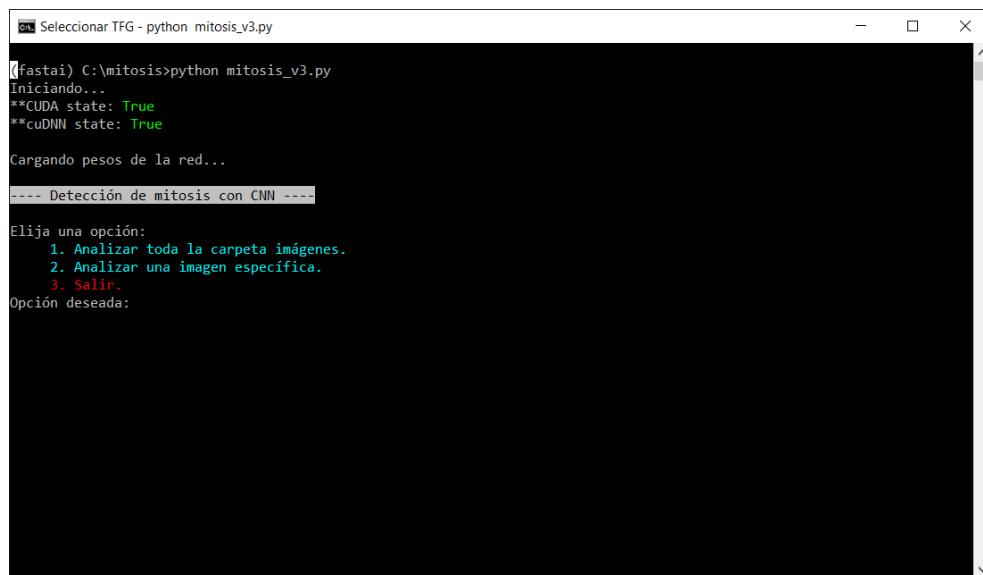
```
C:\mitosis>activate fastai
(fastai) C:\mitosis>tree /f
Listado de rutas de carpetas
El número de serie del volumen es 3E73-6BDD
C:.
    data_augmentation_v2.py
    f1score.py
    funciones.py
    mitosis_v3.py
    randomizer.py
    image
        01.tif
        02.tif
        03.tif

(fastai) C:\mitosis>
```

**Figura 5.8** Activación del entorno virtual y estructura de directorios y archivos.

Como puede observarse en la figura anterior, en el directorio image del cual la aplicación recupera las imágenes histológicas que se van a analizar existen varios archivos con los cuales se va a probar el funcionamiento de la aplicación.

Al ejecutar la aplicación, se muestra un menú como el siguiente:



```
Seleccionar TFG - python mitosis_v3.py
(fastai) C:\mitosis>python mitosis_v3.py
Iniciando...
**CUDA state: True
**cuDNN state: True

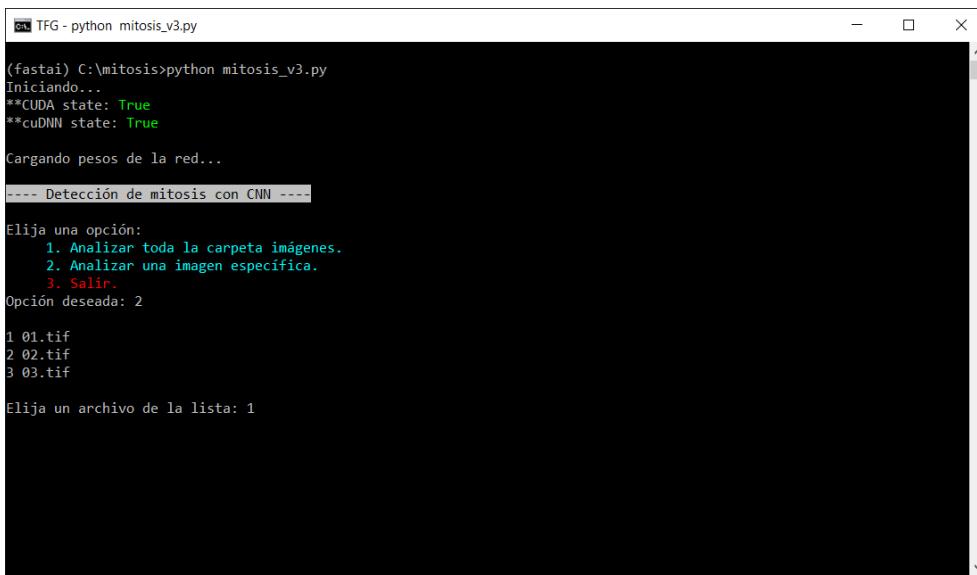
Cargando pesos de la red...
---- Detección de mitosis con CNN ----

Elija una opción:
  1. Analizar toda la carpeta imágenes.
  2. Analizar una imagen específica.
  3. Salir.

Opción deseada:
```

**Figura 5.9** Menú de la aplicación.

Para probar el funcionamiento, se elige la segunda opción para analizar las imágenes por separado.



```
(fastai) C:\mitosis>python mitosis_v3.py
Iniciando...
**CUDA state: True
**cuDNN state: True

Cargando pesos de la red...

---- Detección de mitosis con CNN ----

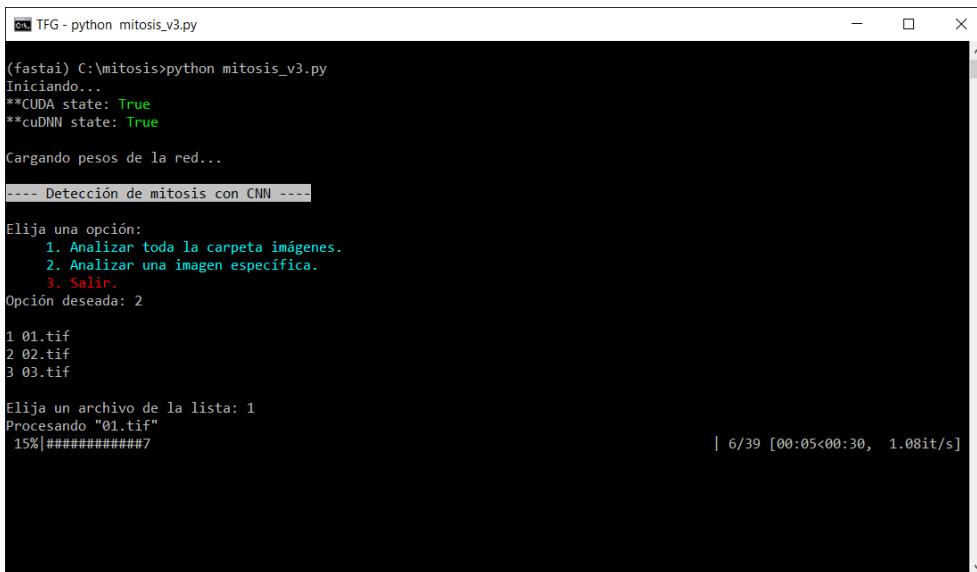
Elija una opción:
 1. Analizar toda la carpeta imágenes.
 2. Analizar una imagen específica.
 3. Salir.
Opción deseada: 2

1 01.tif
2 02.tif
3 03.tif

Elija un archivo de la lista: 1
```

**Figura 5.10** Selección de una imagen histológica.

Al presionar la tecla intro comienza el análisis de la imagen. Para amenizar la espera y comprobar que la aplicación continua su normal funcionamiento se ha incluido una barra de progreso que además muestra el tiempo estimado del proceso.



```
(fastai) C:\mitosis>python mitosis_v3.py
Iniciando...
**CUDA state: True
**cuDNN state: True

Cargando pesos de la red...

---- Detección de mitosis con CNN ----

Elija una opción:
 1. Analizar toda la carpeta imágenes.
 2. Analizar una imagen específica.
 3. Salir.
Opción deseada: 2

1 01.tif
2 02.tif
3 03.tif

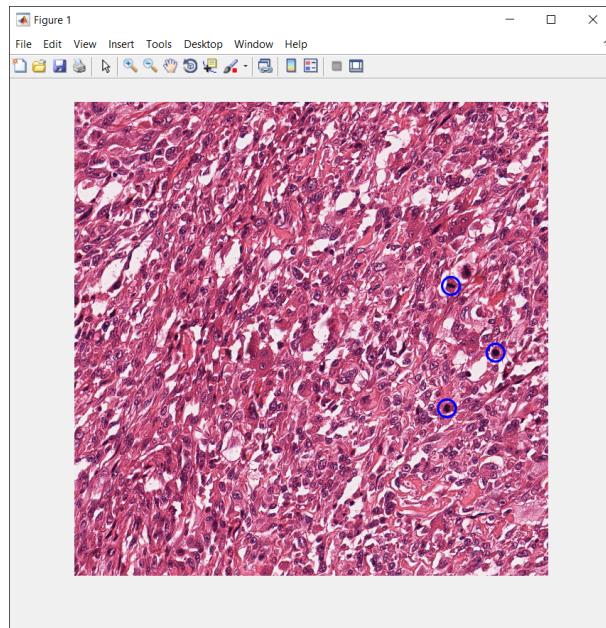
Elija un archivo de la lista: 1
Procesando "01.tif"
15% #####| 6/39 [00:05<00:30, 1.08it/s]
```

**Figura 5.11** Análisis de una imagen.

Como puede observarse en la imagen anterior, el tiempo normal de procesamiento de una imagen, la cual tiene una dimensiones de 2000 x 2000 px supone un tiempo de 30 segundos aproximadamente. Como resultado de dicho proceso, se abre una nueva ventana automáticamente que muestra los resultados arrojados por la red y la aplicación.

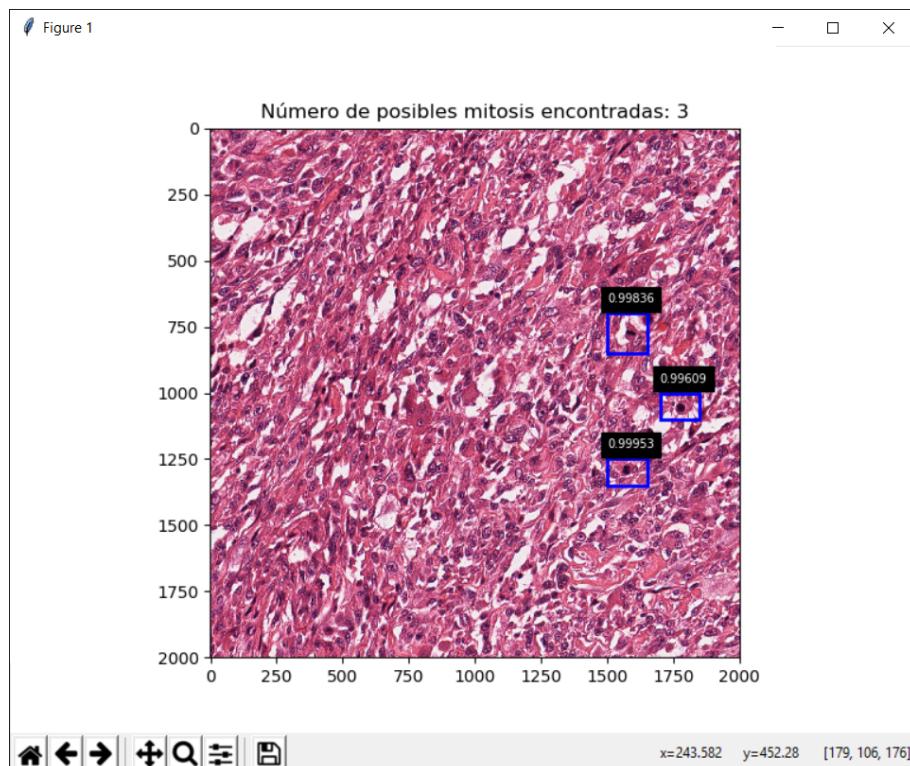
### 5.3.1 Imagen con presencia de mitosis

Dado que en el *dataset* de partida cada imagen viene acompañada de una tabla con las posiciones de las mitosis encontradas por los expertos patólogos, basta con comparar esta imagen con la obtenida como resultado.



**Figura 5.12** Referencia de la primera imagen para comparativa.

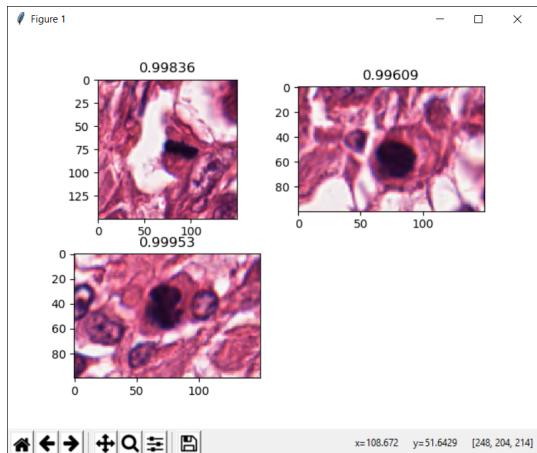
Una vez vista la localización de las mitosis indicadas por la base de datos de partida, se presentan los resultados obtenidos con la aplicación desarrollada en este trabajo.



**Figura 5.13** Resultado del procesamiento y análisis de una imagen histológica con mitosis.

En la figura anterior por tanto, y tras el proceso de análisis y predicción por bloques de la imagen, se han encontrado 3 figuras mitóticas con sus correspondientes puntuaciones de clasificación dadas por la red entrenada.

Si se comparan ambas figuras, se concluye que la red ha detectado correctamente el mismo número de figuras mitóticas y su posición concreta que los especialistas médicos. Si se cierra la ventana con la figura completa, la aplicación muestra una vista en detalle de cada mitosis.

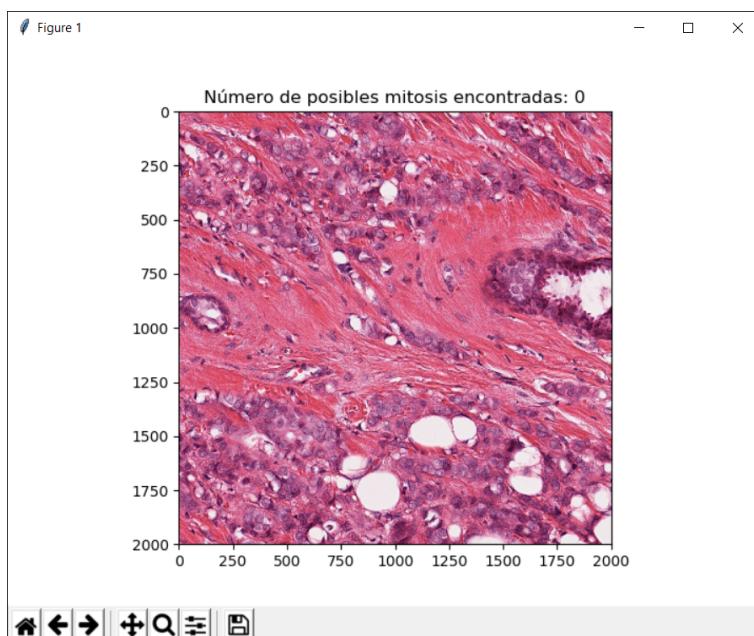


**Figura 5.14** Vista en detalle de cada una de las figuras mitóticas detectadas.

Se está por tanto ante un funcionamiento correcto del método que se ha desarrollado en este trabajo. Sin embargo, es necesario realizar un mayor número de pruebas para verificar la buena capacidad de predicción de la red neuronal.

### 5.3.2 Imagen sin presencia de mitosis

En esta ocasión se utiliza la segunda imagen correspondiente a un tejido sin presencia de mitosis. La aplicación debería por tanto arrojar una predicción en la cual no existiera ningún elemento encontrado.



**Figura 5.15** Resultado del procesamiento y análisis de una imagen histológica sin presencia de mitosis.

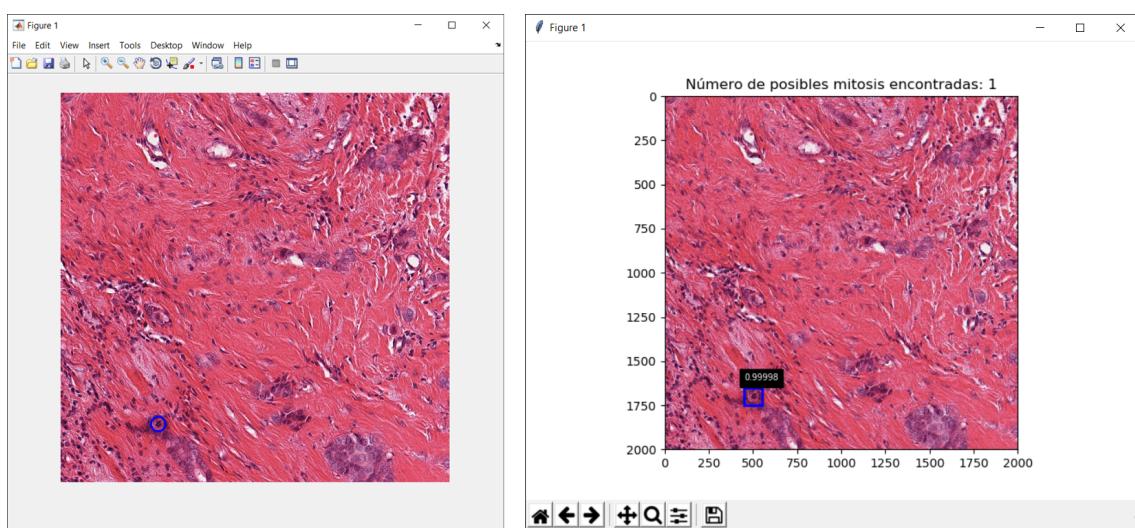
Al igual que la base de datos de partida, la aplicación ha determinado que no existe ninguna figura mitótica en la imagen.

### 5.3.3 Otros ejemplos de uso

Una vez probada la aplicación con dos casos de relevancia que corresponden a presencia y no presencia de mitosis, se repasa el funcionamiento de la aplicación con otras imágenes que han resultado interesantes y las cuales presentan un aspecto diferente a las ya vistas. Se han seleccionado aquellas que pueden suponer un reto por la complejidad de su estructura, por lo que son ideales para poner a prueba la aplicación.

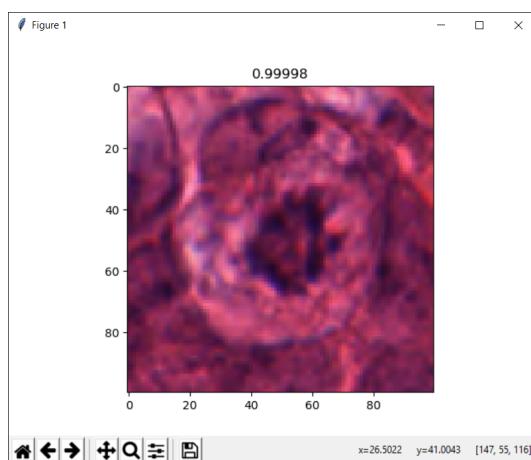
#### Paciente 08. Mitosis con un aspecto diferente

Se muestra a continuación un ejemplo de imagen de tejido que difiere ligeramente a los anteriores, con diferentes tonos de color y la presencia de una mitosis con un aspecto diferente.



**Figura 5.16** Caso 08. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha).

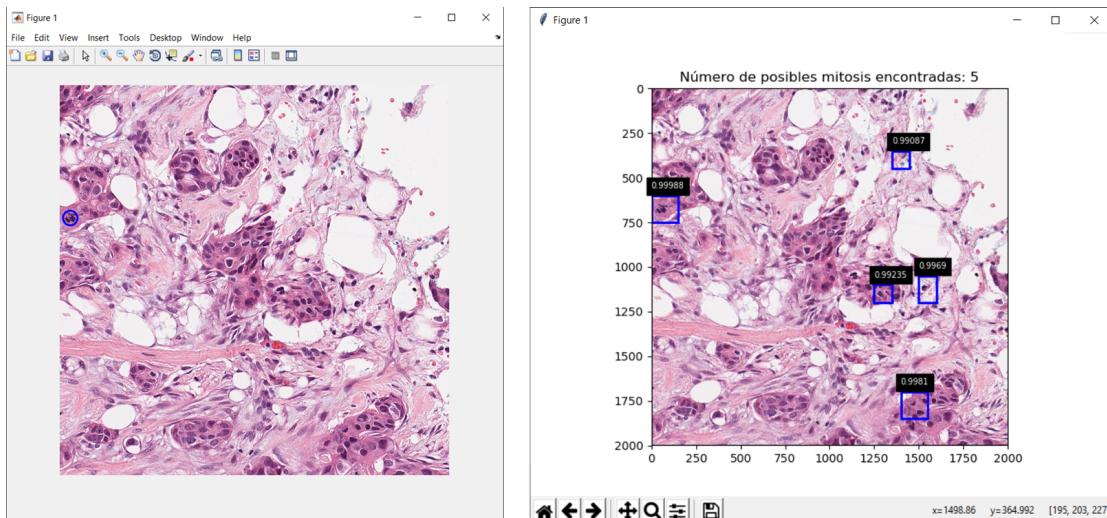
Si se observa la mitosis en detalle se observa por qué es de interés, y es que pese a presentar una estructura diferente a las anteriores la red es capaz de clasificarla correctamente. Con este ejemplo queda claro que la aplicación puede detectar figuras mitóticas que se encuentran en distintas fases.



**Figura 5.17** Caso 08. Vista en detalle de la mitosis detectada.

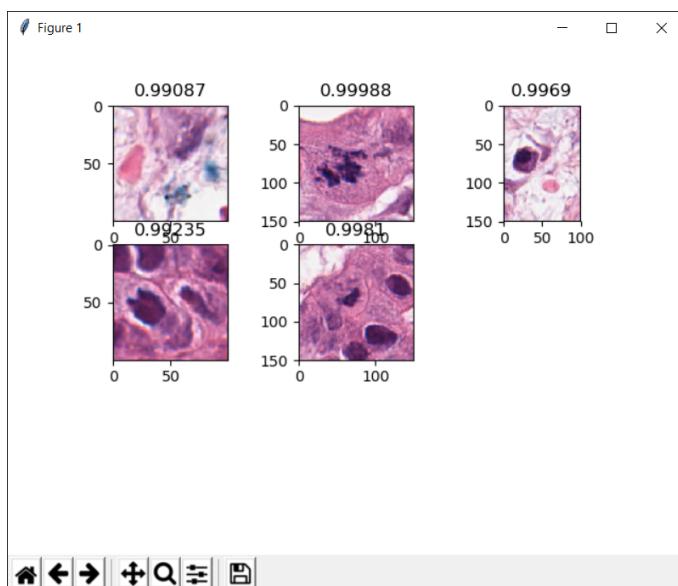
### Paciente 06. Mayor número de mitosis encontradas

El siguiente ejemplo resulta de interés porque en la imagen son detectadas más mitosis que las presentes en la base de datos.



**Figura 5.18** Caso 06. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha).

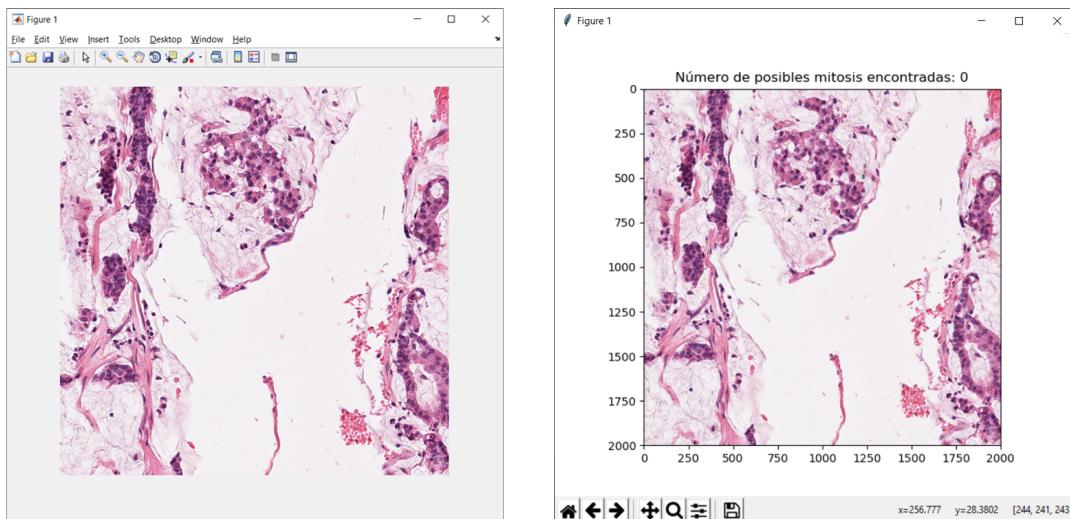
Si se observan en detalle las posibles mitosis detectadas pueden encontrarse elementos que o bien podrían serlo pero no fueron detectadas por los especialistas o bien se asemejan bastante al aspecto de una mitosis. En cualquiera de los casos, la única mitosis presente en la base de datos también lo está en la predicción realizada por la aplicación y además presenta una puntuación mayor que el resto, permitiendo de este modo afinar incluso más en el umbral, discriminando al resto.



**Figura 5.19** Caso 06. Vista en detalle de las mitosis detectadas.

### Paciente 09. Ausencia de mitosis

La siguiente imagen histológica tiene ausencia de figuras mitóticas además de un aspecto similar al caso anterior. Resulta de interés para comparar si se encuentra algún elemento ya que existen en ella numerosas zonas susceptibles de ser clasificadas como mitosis sin serlo.

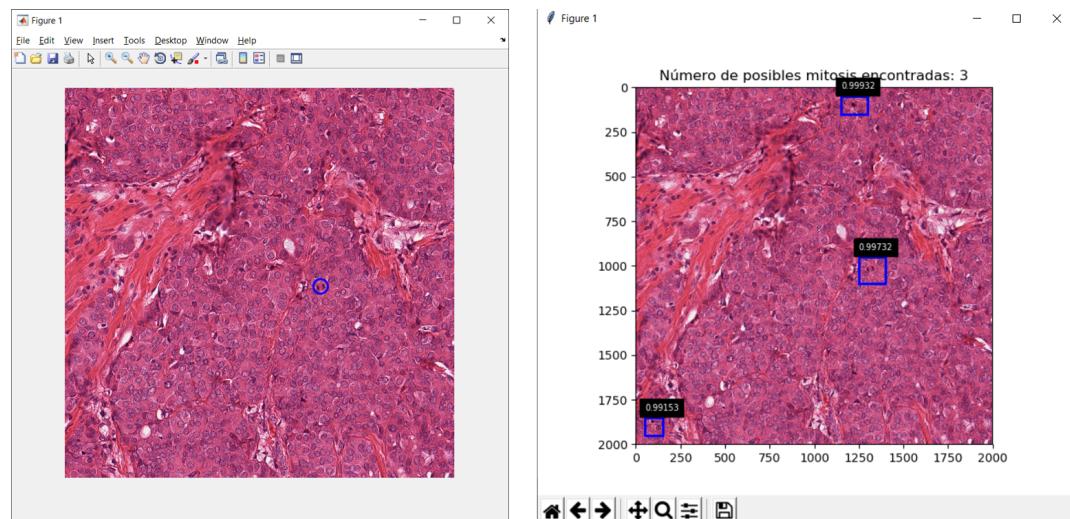


**Figura 5.20** Caso 09. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha).

Al igual que la base de datos, la predicción realizada por el programa no indica presencia de ninguna figura mitótica, por lo que en este caso los resultados son correctos.

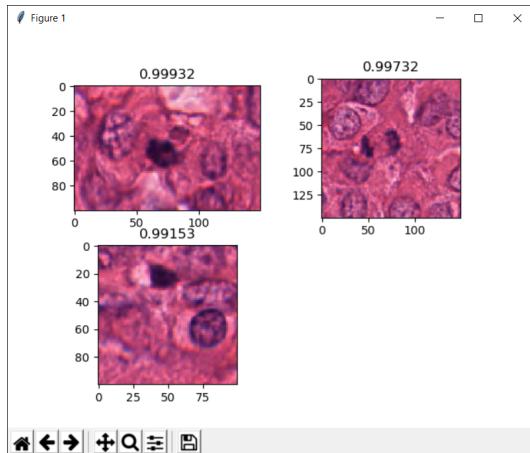
### Paciente 14. Mitosis con un aspecto diferente

En este caso existe una mitosis a detectar la cual se encuentra dividida, es decir, la forman dos elementos cercanos.



**Figura 5.21** Caso 14. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha).

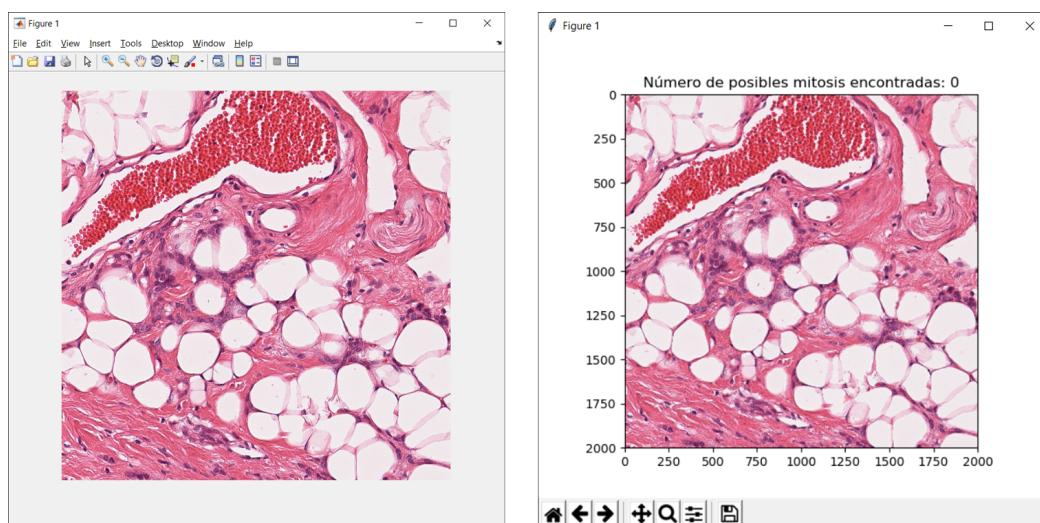
Como puede verse en la figura anterior, un número mayor de posibles mitosis fueron encontradas respecto a las presentes en la base de datos. Sin embargo, aquella que tenía un aspecto diferente y que resultaba de interés para este caso fue correctamente encontrada, como queda reflejado en la siguiente imagen.



**Figura 5.22** Caso 14. Vista en detalle de las mitosis detectadas.

#### Paciente 19. Ausencia de mitosis

Finalmente, se expone el siguiente caso de interés con ausencia de mitosis para probar el método en una imagen con tonalidades muy diferentes.



**Figura 5.23** Caso 19. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha).

Se tiene aquí por tanto otro caso de éxito donde se cumple la igualdad entre lo que indica la base de datos y el resultado del procesamiento a través de la aplicación desarrollada para el proyecto.

## 5.4 Resultados finales. Estadísticas de la aplicación

Para concluir con la exposición de resultados, se han estudiado todas las imágenes de los primeros 23 casos de la base de datos. Se ha realizado de este modo ya que son los casos incluidos desde el principio por la organización del reto y los que mayor riqueza de imágenes tienen. Los casos restantes solamente incluyen una única imagen.

El objetivo una vez más es probar cómo funciona la aplicación y su validez de uso, comparando los resultados que ofrece la clasificación con el método diseñado con la base de datos de partida realizada por expertos de forma manual.

Es importante señalar que estas pruebas se han realizado sobre imágenes que se han usado tanto para entrenamiento como para validación. Aunque no es recomendable comprobar el funcionamiento con datos de entrenamiento, dada la baja cantidad de prototipos de mitosis y la reducida base de datos para validación, se ha decidido realizar de este modo para tener una mayor cantidad de mitosis a detectar.

Para probar la aplicación se han usado 10 imágenes de cada caso, ya que en la mayoría de estos ese era el número que incluían y se ha deseado mantener una igualdad, a excepción de algunos casos los cuales resultaban de interés y se analizaron por completo. Cabe señalar que el valor del umbral se mantuvo sin variaciones durante todas las pruebas. Los resultados fueron los siguientes:

**Tabla 5.2** Resultados del análisis de 23 casos clínicos.

Caso	Mitosis (database)	Mitosis detectadas	Mitosis correctas	Detección (%)
1	73	139	67	91.78 %
2	37	110	37	100 %
3	18	41	17	94.44 %
4	224	567	210	93.75 %
5	6	5	5	83.33 %
6	14	37	14	100 %
7	15	62	15	100 %
8	3	9	3	100 %
9	2	9	2	100 %
10	0	1	0	100 %
11	11	49	10	90.91 %
12	8	27	7	87.5 %
13	3	17	2	66.67 %
14	11	19	8	72.73 %
15	13	53	13	100 %
16	9	14	8	88.89 %
17	6	10	5	83.33 %
18	28	50	26	92.86 %
19	2	2	2	100 %
20	0	1	0	100 %
21	23	89	20	86.96 %
22	17	30	16	94.12 %
23	14	22	12	85.71 %

**Tabla 5.3** Resultados del conjunto de datos.

Mitosis (database)	Mitosis detectadas	Mitosis correctas	Detección (%)	F1-Score
537	1363	499	92.92 %	0.5253

#### 5.4.1 Conclusiones sobre las estadísticas

Tras haber probado en profundidad la aplicación y a la vista de las estadísticas expuestas, se concluye lo siguiente:

- La red era capaz de detectar correctamente casi la totalidad de las mitosis presentes en la base de datos, alcanzando un 92.92% tras probar en 23 casos diferentes (más de 300 imágenes) y un 95.8% sobre la base de datos de validación.
- Existe un alto valor de falsos positivos, fruto de varias circunstancias.
  - Por un lado, el hecho de haber reducido considerablemente la clase sin presencia de mitosis provoca una mala clasificación de zonas donde no hay mitosis. Esto podría solucionarse con una mayor cantidad de mitosis para analizar, pero la pobreza de la base de datos en este sentido impide que sea mejor la precisión.
  - El umbral elegido en la aplicación se mantuvo fijo todo el tiempo. Se ha observado en numerosos casos cómo estos falsos positivos reciben una clasificación ligeramente inferior a las de las mitosis reales, por lo que pueden ser filtrados de mejor manera.
  - Se ha observado que en tejidos con alta presencia de mitosis estos falsos positivos se acentúan. Cuando hay poca o ninguna presencia de estas figuras, los falsos positivos suponen un valor muy bajo y en algunos casos es inexistente.
  - Existen casos como el número 21 donde casi la totalidad de las imágenes están compuestas por figuras muy similares a las mitosis, disparándose el número de falsos positivos.
- Un efecto común en estas pruebas ha sido el hecho de que algunas mitosis que debían ser detectadas no lo fueron por encontrarse en los límites de la imagen y sin aparecer completamente.
- La aplicación se comporta correctamente en casos sin apenas presencia de mitosis y es capaz de detectar aquellos elementos que son de interés.
- El valor final alcanzado de F1 se encuentra muy cercano al del método propuesto por la UPV-EHU [8] en el Tumor Proliferation Assessment Challenge de 2016, el cual fue de 0.541.



## 6 Conclusiones y líneas futuras

---

El uso de tecnologías *Deep Learning* supone la creación de útiles herramientas que tienen cabida en multitud de campos. Se evidencia en este trabajo su potencial en el ámbito de la medicina, permitiendo el desarrollo aplicaciones capaces de detectar elementos difíciles de encontrar incluso para expertos médicos.

Es importante destacar que el uso de estas tecnologías solo suponen una herramienta de ayuda para los especialistas y nunca deben sustituirlos, ya que como se ha observado existen errores de predicción. Sin embargo, es innegable su utilidad automatizando tareas y realizándolas en pocos segundos.

En este trabajo se ha desarrollado una aplicación para la detección y conteo de figuras mitóticas en tejidos de pacientes con cáncer de mama. Partiendo de una base de datos proporcionada por la organización del *Tumor Proliferation Assessment Challenge*, se ha entrenado un modelo de red neuronal ResNet34 aplicando diferentes técnicas para posteriormente implementarlo vía *software* con el desarrollo de una aplicación Python. Finalmente se ha obtenido una precisión del 98.5 % sobre el *dataset* de partida y una detección del 92.92 % de las mitosis en las pruebas realizadas sobre 23 de los casos propuestos, concluyendo con un valor F1 de 0.5253.

Otro punto a señalar es la capacidad para generalizar el método aquí propuesto. Del mismo modo que se ha centrado en el caso de detección de mitosis, el flujo de trabajo a seguir para conseguir detectar otros elementos es similar, siendo importante reflexionar sobre el problema y dimensionarlo correctamente. Además, de la misma forma que el *Deep Learning* sirve para encontrar figuras o patrones de interés en imágenes, también puede ser usado para predecir la aparición de una enfermedad en un paciente en base a sus datos médicos.

Volviendo al caso de la detección y conteo de mitosis, se concluye que los resultados fueron los deseados y que las herramientas utilizadas fueron las apropiadas. Se presenta por tanto un método eficaz y que es posible desplegar en la mayoría de dispositivos, con un entrenamiento de modelo de red neuronal que puede realizarse en poco tiempo y con recursos limitados gracias a la librería para inteligencia artificial expuesta.

Como ventajas de la aplicación son destacables su precisión encontrando los elementos deseados así como el tiempo empleado en dicho proceso. Sin embargo, sería posible mejorar estos puntos con una base de datos más extensa que contemplara un mayor número de mitosis en las diferentes fases en las que pueden encontrarse y con un *hardware* y clasificador más rápidos que pudieran procesar un mayor número de bloques por segundo. Se señalan por tanto como dificultades del proyecto los altos requerimientos *hardware* que necesitan las tecnologías utilizadas y una base de datos complicada de manejar.

En cuanto a las líneas de investigación futuras, campos como el de la inteligencia artificial, *Machine Learning* y *Deep Learning* están sufriendo un enorme y rápido desarrollo en los últimos años causado por el avance tecnológico en general y su popularización. Respecto a sus aplicaciones médicas, su evolución no solo irá encaminada a proyectos de mayor fiabilidad, sino que también deberá hacer frente a detractores y superar el debate ético que su uso conlleva. Sin embargo, es innegable que estas tecnologías ocuparán un importante y útil lugar en nuestras vidas en el futuro y que aún les queda un gran camino por recorrer.



# Índice de Figuras

---

3.1	Esquema de una red neuronal [10]	5
3.2	Esquema del perceptrón	6
3.3	Función de activación	6
3.4	Ejemplo de una CNN [10]	7
3.5	Vista de una CNN como filtros [12]	8
3.6	Efecto de malos <i>learning rates</i> [13]	9
3.7	Aprendizaje pobre, buen aprendizaje y sobre-aprendizaje [14]	9
4.1	Ejemplo de entrenamiento con AlexNet	15
4.2	Arquitectura de GoogLeNet	16
4.3	Frameworks más importantes [22]	16
4.4	Comparativa entre tecnologías con una red ResNet [26].	17
4.5	Arquitectura de ResNet34	18
4.6	Entrenamiento de ResNet34 con el <i>dataset</i> de prueba	19
4.7	Segundo entrenamiento de ResNet34 con el <i>dataset</i> de prueba	19
4.8	Imagen histológica típica proporcionada	20
4.9	Ejemplo de imagen con anotaciones (en verde) sobre mitosis	20
4.10	Esquema de las herramientas de las que hace uso la aplicación desarrollada	21
4.11	Ejemplo del recorte de una mitosis	23
4.12	Ejemplo de <i>data augmentation</i>	24
4.13	Mitosis tratada con técnicas de <i>data augmentation</i>	25
4.14	Ejemplos de elementos en la base de datos sin figuras mitóticas	25
4.15	Estructura de directorios para el entrenamiento	27
4.16	Recursos necesarios para el entrenamiento	28
4.17	Recursos necesarios para el entrenamiento	28
4.18	Entrenamiento de la última capa del modelo	29
4.19	Clasificación de los datos para validación	29
4.20	Ejemplos de predicciones correctas	30
4.21	Ejemplos de predicciones incorrectas	30
4.22	Ejemplos de mitosis con mayor puntuación	30
4.23	Ejemplos de clase sin mitosis con mayor puntuación	30
4.24	Ejemplos de predicciones imprecisas	31
4.25	Ejecución del algoritmo y su detención en el momento en el que las pérdidas aumentan	31
4.26	Evolución de las pérdidas a medida que aumenta el <i>learning rate</i>	31
4.27	Función de <i>data augmentation</i>	32
4.28	Descenso del gradiente con reinicios	32
4.29	Entrenamiento de la última capa con esta técnica	33
4.30	Diferentes <i>learning-rates</i> para diferentes capas	33
4.31	<i>Learning-rate</i> diferencial con reinicios	33
5.1	Resultado del último entrenamiento tras aplicar <i>fine-tuning</i>	35

5.2	Matriz de confusión	36
5.3	Mitosis mal clasificadas	36
5.4	Possible exploración sin solapamiento donde no se detecta una mitosis (señalada en verde)	38
5.5	Exploración con solapamiento donde se detecta la mitosis (señalada en verde)	38
5.6	Exploración con solapamiento donde se detecta la misma mitosis varias veces	40
5.7	Algoritmo de fusión de cuadros que hacen referencia a la misma mitosis	40
5.8	Activación del entorno virtual y estructura de directorios y archivos	46
5.9	Menú de la aplicación	46
5.10	Selección de una imagen histológica	47
5.11	Análisis de una imagen	47
5.12	Referencia de la primera imagen para comparativa	48
5.13	Resultado del procesamiento y análisis de una imagen histológica con mitosis	48
5.14	Vista en detalle de cada una de las figuras mitóticas detectadas	49
5.15	Resultado del procesamiento y análisis de una imagen histológica sin presencia de mitosis	49
5.16	Caso 08. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha)	50
5.17	Caso 08. Vista en detalle de la mitosis detectada	50
5.18	Caso 06. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha)	51
5.19	Caso 06. Vista en detalle de las mitosis detectadas	51
5.20	Caso 09. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha)	52
5.21	Caso 14. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha)	52
5.22	Caso 14. Vista en detalle de las mitosis detectadas	53
5.23	Caso 19. Comparativa entre una detección manual (izquierda) y automática con la aplicación desarrollada (derecha)	53

# Índice de Tablas

---

4.1	Especificaciones técnicas del equipo	14
5.1	Comparativa de entrenamientos	35
5.2	Resultados del análisis de 23 casos clínicos	54
5.3	Resultados del conjunto de datos	54



# Índice de Códigos

---

4.1	Script para reducir la clase <i>non-mitosis</i>	24
4.2	Script para <i>data augmentation</i> de la clase <i>mitosis</i>	26
5.1	Función para la exploración y clasificación de imágenes histológicas.	39
5.2	Función para el procesamiento de mitosis encontradas	41
5.3	Función que comprueba la disponibilidad de las tecnologías CUDA y cuDNN	42
5.4	Función que muestra el menú principal de la aplicación	43
5.5	Función para mostrar los resultados finales	44
5.6	Función para mostrar los resultados finales en detalle	44
5.7	Función principal para la aplicación	44



# Bibliografía

---

- [1] Instituto Nacional del Cáncer NIH, *¿Qué es el cáncer?*.  
<https://www.cancer.gov/espanol/cancer/naturaleza/que-es> (Consultado: Junio de 2018)
- [2] Tumor Proliferation Assessment Challenge 2016, *Introduction*.  
<http://tupac.tue-image.nl/> (Consultado: Junio de 2018)
- [3] NVIDIA, *Deep Learning - Advances in medicine*.  
<http://www.nvidia.com/object/deep-learning-in-medicine.html> (Consultado: Junio de 2018)
- [4] Irshad H., *Automated mitosis detection in histopathology using morphological and multi-channel statistics features.* , Journal for Pathology Informatics, Mayo 2013.
- [5] Mitko P., *Assessment of Mitosis Detection Algorithms*. AMIDA13.  
<http://amida13.isi.uu.nl> (Consultado: Junio de 2018)
- [6] Malon C.D., Cosatto E., *Classification of mitotic figures with convolutional neural networks and seeded blob features.* , Journal for Pathology Informatics, 2013.
- [7] Dan C., Cires A.G., *Mitosis Detection in Breast Cancer Histology Images with Deep Neural Networks.* , International Conference on Medical Image Computing and Computer-Assisted Intervention, 2013.
- [8] Salado García J.P., Picón Ruiz A., Bereciartua Pérez A., Irusta Zarandona U., *Contaje de mitosis en imágenes histológicas mediante redes neuronales convolucionales.* , XXXV Congreso Anual de la Sociedad Española de Ingeniería Biomédica, 2017.
- [9] Albarquoni S., Baur C., Achilles F., Belagiannis V., Demirci S., Navab N., *AggNet: Deep Learning From Crowds for Mitosis Detection in Breast Cancer Histology Images.* , IEEE Transactions on Medical Imaging, 2016.
- [10] Mathworks, *Redes Neuronales Convolucionales - Tres cosas que es necesario saber*.  
<https://es.mathworks.com/solutions/deep-learning/convolutional-neural-network.html>  
(Consultado: Junio de 2018)
- [11] Martínez de Dios J.R., *Redes de Neuronas para Reconocimiento de Patrones.* , Visión Artificial, Lección 4.5, ETSI Universidad de Sevilla, 2017.
- [12] Diaz-Aviles E., *A Glimpse into Deep Learning for Recommender Systems*.  
<https://medium.com/> (Consultado: Junio de 2018)
- [13] Varma S., Das S., *Training Neural Networks.* , Chapter 7, Part 1.  
<http://srdas.github.io/DLBook/GradientDescentTechniques.html> (Consultado: Junio de 2018)
- [14] Bhande A., *What is underfitting and overfitting in machine learning and how to deal with it*.  
<https://medium.com/> (Consultado: Junio de 2018)

- [15] Yu F., *A Comprehensive guide to Fine-tuning Deep Learning Models in Keras.*  
<https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html>  
(Consultado: Junio de 2018)
- [16] CV-Tricks, *Fine-tuning Convolutional Neural Network on own data using Keras Tensorflow*  
<http://cv-tricks.com/keras/fine-tuning-tensorflow/> (Consultado: Junio de 2018)
- [17] Hicklin J., *Deep Learning with MATLAB: Transfer Learning in 10 Lines of MATLAB Code*, MathWorks, 2017.  
<https://es.mathworks.com/videos/series/deep-learning-with-MATLAB.html>  
(Consultado: Junio de 2018)
- [18] Nehemiah A., Leung V., *Deep Learning for Computer Vision with MATLAB*, MathWorks.
- [19] MathWorks, *Pretrained AlexNet convolutional neural network.*  
<https://es.mathworks.com/help/nnet/ref/alexnet.html> (Consultado: Junio de 2018)
- [20] ImageNet,  
<http://image-net.org/index> (Consultado: Junio de 2018)
- [21] MathWorks, *Pretrained GoogLeNet convolutional neural network.*  
<https://es.mathworks.com/help/nnet/ref/googlenet.html> (Consultado: Junio de 2018)
- [22] den Bakker I., *Battle of the Deep Learning frameworks - Part I: 2017, even more frameworks and interfaces.*  
<https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i>  
(Consultado: Junio de 2018)
- [23] Howard J., Thomas R., *About fast.ai.*  
<http://www.fast.ai/about/> (Consultado: Junio de 2018)
- [24] Howard J., Thomas R., *Deep Learning for Coders.*  
<http://course.fast.ai/> (Consultado: Junio de 2018)
- [25] *Tensors and Dynamic neural networks in Python with strong GPU acceleration.*  
<https://pytorch.org/> (Consultado: Junio de 2018)
- [26] Sako Y., Ludwiczuk B., *Benchmark on Deep Learning Frameworks and GPUs.*  
<https://github.com/u39kun/deep-learning-benchmark> (Consultado: Junio de 2018)
- [27] He K., Zhang X., Ren S., Sun J., *ResNet-34 - Deep Residual Learning for Image Recognition.*  
<https://www.kaggle.com/pytorch/resnet34/> (Consultado: Junio de 2018)
- [28] Tumor Proliferation Assessment Challenge 2016, *Dataset*  
<http://tupac.tue-image.nl/node/3> (Consultado: Junio de 2018)
- [29] Spyder - The Scientific Python Development Environment.  
<https://github.com/spyder-ide> (Consultado: Junio de 2018)
- [30] fast.ai - Making neural nets uncool again.  
<http://www.fast.ai/> (Consultado: Junio de 2018)
- [31] Pillow - The friendly PIL fork.  
<http://python-pillow.org/> (Consultado: Junio de 2018)
- [32] Numpy - Scientific computing.  
<http://www.numpy.org/> (Consultado: Junio de 2018)
- [33] Matplotlib.  
<https://matplotlib.org/> (Consultado: Junio de 2018)
- [34] Colorama.  
<https://pypi.org/project/colorama/> (Consultado: Junio de 2018)

- [35] Termcolor.  
<https://pypi.org/project/termcolor/> (Consultado: Junio de 2018)
- [36] tqdm.  
<https://tqdm.github.io/> (Consultado: Junio de 2018)
- [37] Upasana, *How to handle Imbalanced Classification Problems in machine learning.*  
<https://www.analyticsvidhya.com/blog/2017/03/imbalance-classification-problem/>  
(Consultado: Junio de 2018)
- [38] Wang S., Liu W., Wu J., Cao L., Meng Q., Kennedy P.J. *Training deep neural networks on imbalanced data sets.* , International Joint Conference on Neural Networks (IJCNN), 2016.



