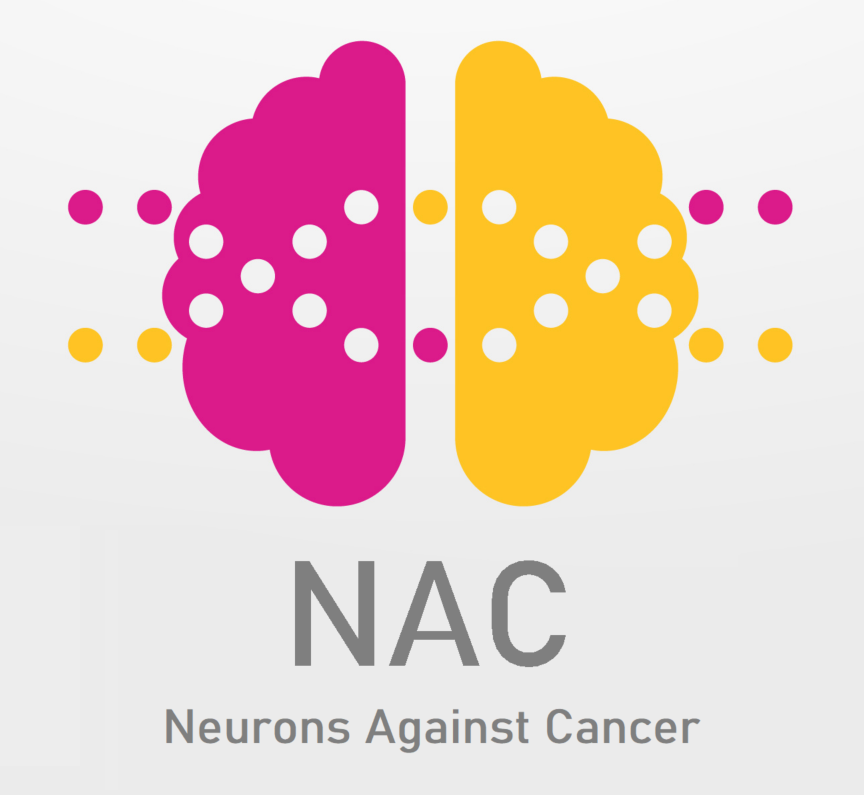
**Detección de tumor de Colon a través de imágenes histopatológicas**



Master in Big Data Science, 2020-21

Autores **Carlos-Maria Ainchil Cayuela**

**Malena Biedma Rodríguez**

**Montserrat Puiggròs Maldonado**

Tutores [Nombre y Apellidos]

Fecha de presentación 03 / 06 / 2021

**Contenido**

[**Índice de tablas y figuras**](#_cyyupuxhk91a) **4**

[Introducción](#_30j0zll) **6**

[**Grupo de Investigación**](#_6qdae0tq4006) **6**

[**El interlocutor con la empresa es Víctor Raúl Moreno Aguado. Es doctor en medicina y cirugía por la Universitat Autònoma de Barcelona. Trabaja como investigador en el Instituto de Investigación Biomédica de Bellvitge (IDIBELL) y el Institut Català de la Salut (ICO), y colabora con la Universitat de Barcelona y el Hospital de Bellvitge. Colidera el grupo de investigación del Cáncer colorrectal (CRC) del IDIBELL y participa en los consorcios internacionales de investigación sobre cáncer de colon MOTRICOLOR y FIGI (insertar nota al pie)**](#_9w0ameo34vej) **6**

[**https://idibell.cat/es/investigacion/area-de-cancer/programa-de-mecanismos-moleculares-y-terapia-experimental-en-oncologia-oncobell/cancer-colorectal/**](#_l0yfueqioeq2) **6**

[**Descripción del problema**](#_d6iodr303ic) **6**

[**Cáncer de colon**](#_k31ekfjweq65) **6**

[**Imágenes histopatológicas**](#_k31ekfjweq65) **7**

[**Diagnosticado el cáncer por patólogos**](#_s284jubqeeo7) **7**

[**Introducción al Deep Learning**](#_98ij027itrw8) **7**

[**Que es**](#_no7vwi1g5ko) **7**

[**Que puede aportar al campo del diagnóstico del cáncer**](#_no7vwi1g5ko) **7**

[**Estado del arte**](#_nei8w94xw77q) **7**

[**Arquitectura y Tecnología**](#_j3govxbbscly) **7**

[**Dataset**](#_4xxmfowhfxtt) **7**

[**Ubicación y estructura de los datos en el servidor**](#_n46v7kj3ckau) **8**

[**Descripción y tratamiento del dataset**](#_g8wzucd7m6ne) **9**

[**Extracción de la capa según la resolución escogida**](#_irl6ib5t6uqd) **11**

[**Verificación de la resolución**](#_duk23lfvo0m7) **13**

[**Tecnología usada**](#_qo62x37liifg) **17**

[**EDA**](#_hn0w2ekhppbb) **18**

[**Tratamiento estadístico de las imágenes**](#_ocetnlnyoo0t) **18**

[**Estratificación**](#_wrljkeasl8q4) **18**

[**Modelo: Aplicación de Deep Learning al problema**](#_8to74hgoot1d) **18**

[**Clasificador**](#_yqilh9i28e2w) **19**

[**Transferencia específica: Paper seleccionado**](#_4z2zrjbka39a) **19**

[**Modelo 1**](#_78kqkzd4jt20) **19**

[**Transferencia general: Arquitecturas estándares**](#_9jhhesmtftsw) **21**

[**Modelo 2: MobileNet - Modelo 3: Inception-v3**](#_wo1kd97lmx03) **22**

[**Hyperparametrización**](#_vf9vad9cpai1) **25**

[**Comparativa: MobileNet vs. Inception-v3**](#_4z2zrjbka39a) **25**

[**Modo de trabajo (jupyter → py / github / excel hyperparametrización / test paralelo petó…, tensorflow / keras)**](#_bwqld46fpuo9) **25**

[**Localizador - LO QUITAMOS y lo ponemos a futuro ???**](#_g5jlx7t2hi31) **25**

[**Modelo 4 - Localización**](#_gw4w0vkysqd6) **25**

[**Conclusiones**](#_ookhbt6do2wj) **25**

[**Propuesta de valor**](#_837t1quz3ur) **25**

[**Posibles mejoras**](#_7faw6vxdzltm) **25**

[**Aplicaciones futuras**](#_7faw6vxdzltm) **26**

[**Ética**](#_9zchjrvtlbht) **26**

[**Protección de datos**](#_xia8s87ao69h) **26**

[**“Valores humanos”**](#_xia8s87ao69h) **26**

[**Planificación del proyecto ???**](#_4vi0o1u4jhnj) **27**

[**Bibliografía**](#_4vi0o1u4jhnj) **27**

[**Anexo**](#_hg49vf38f4q2) **28**

[**Scripts de extracción de las imágenes**](#_g3kbebrnh9c5) **28**

# Índice de tablas y figuras

**Índice de tablas**

XXX

**Índice de figuras**

XXX

**Agradecimientos**

# Introducción

## Grupo de Investigación

Los autores de este trabajo de fin de máster son:

Carlos-Maria Ainchil Cayuela

* Graduado en economía por la Universitat Pompeu Fabra con un posgrado en empresa por la Universitat Autònoma de Barcelona y un máster en economía por la *University of York* (Reino Unido). Ha desarrollado su carrera como auditor interno en diferentes sectores.

Malena Biedma Rodríguez

* Graduada en administración y dirección de empresas por la Universitat de Barcelona con mención en contabilidad y fiscal. Tiene amplia experiencia profesional en el ámbito de la auditoría y la contabilidad siendo ésta su dedicación actual.

Montserrat Puigròs Maldonado

* Ingeniera informática por la Universitat Pompeu Fabra de Barcelona con un máster en tecnologías multimedia por la misma universidad. Actualmente trabaja como *Research Engineer* en el Barcelona Supercomputer Center - Centro Nacional de Supercomputación.

### 

### El interlocutor con la empresa es Víctor Raúl Moreno Aguado. Es doctor en medicina y cirugía por la Universitat Autònoma de Barcelona. Trabaja como investigador en el Instituto de Investigación Biomédica de Bellvitge (IDIBELL) y el *Institut Català de la Salut* (ICO), y colabora con la Universitat de Barcelona y el Hospital de Bellvitge. Colidera el grupo de investigación del Cáncer colorrectal (CRC) del IDIBELL y participa en los consorcios internacionales de investigación sobre cáncer de colon MOTRICOLOR y FIGI (insertar nota al pie)

<http://www.ub.edu/comint/pdi/concursos/20170323/moreno.pdf>

### <https://idibell.cat/es/investigacion/area-de-cancer/programa-de-mecanismos-moleculares-y-terapia-experimental-en-oncologia-oncobell/cancer-colorectal/>

## Descripción del problema

### Cáncer de colon

El cáncer es una enfermedad que se caracteriza por la transformación de las células, que proliferan de manera anormal e incontrolada (\*\*drae). En este trabajo se tratará el cáncer de colon que consiste en el crecimiento anormal de células en la zona del colon. Este cáncer es el tercero más común y el segundo con mayor mortalidad.

<https://www.who.int/es/news-room/fact-sheets/detail/cancer>

<https://www.cancer.gov/espanol/tipos/colorrectal>

### Imágenes histopatológicas

* Tinción de las imágenes, técnica usada en nuestras imágenes
* Definición imágenes histopatológicas
* Tipo de tinción

### Diagnosticado el cáncer por patólogos

## Introducción al Deep Learning

### Que es

### Que puede aportar al campo del diagnóstico del cáncer

Review Nature ⇒ <https://www.nature.com/articles/s41568-020-00327-9.pdf>

## Estado del arte

Explicación de los papers que hemos encontrado:

<https://drive.google.com/drive/u/1/folders/1Drix16OeyZ3JvCB4aH4lFXxFc-iU1ANk>

## 

# Arquitectura y Tecnología

# Dataset

El dataset ha sido facilitado por el Dr. Víctor Moreno, en el marco de una colaboración que está realizando con un grupo de “Los Ángeles, California”. Al tratarse de datos sensibles relacionados con pacientes, estos no pueden moverse de las instalaciones del IDIBELL y deben ser tratados en las mismas.

El dataset está formado por 1331 imágenes histopatológicas recopiladas a lo largo de más de 20 años y las primeras fueron casos de pacientes que enfermaron en el año 2000. Cabe mencionar que el tratamiento de dichas imágenes no ha cambiado sustancialmente, ya que la técnica de tinción usada, la Hematoxilina-eosina también conocida como tinción general, no ha sufrido cambios significativos durante estos años (Depto. de Biología Funcional y Ciencias de la Salud. Facultad de Biología. Universidad de Vigo, 2020; Wikipedia - HE, 2021).

El nombre de las imágenes es el identificador del paciente y no están todos los pacientes representados en todos los grupos de imágenes.

## Ubicación y estructura de los datos en el servidor

Como se ha comentado en la sección “Arquitectura”, para acceder a los datos es necesario conectarse al servidor facilitado y este acceso será mediante un docker específico, dependiendo de la tarea que se quiera realizar. En este caso, al querer tratar los archivos del dataset, se accede mediante el docker “GPU”. Este docker realiza la conexión a través de Jupyter, con el que podemos realizar distintas tareas a través de la terminal (sistema usado para realizar el tratamiento de los datos, de las imágenes) o del Jupyter Notebook.

En el servidor se puede encontrar un directorio con las imágenes /home/jovyan/images/ (en adelante images) y un directorio de trabajo /home/jovyan/work/ (en adelante work) que será donde se podrán crear, modificar y ejecutar los archivos.

En el directorio images se encuentra un archivo que contiene los metadatos, llamado CRCpath.csv, y las 1331 imágenes facilitadas para el estudio y distribuidas en diferentes carpetas, mostradas en la Tabla X:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cantidad | % | Peso |
| T | **702** | **52** | **1.3T** |
| N | **264** | **20** | **220G** |
| TN | **362** | **27** | **727G** |
| U | **3** | **0** | **4.8G** |
| **Total** | **1331** | **100** | **2.2T** |

Tabla X: Información archivos TIF

Donde:

* N es el directorio Normal que contiene imágenes con tejido sano.
* T es el directorio Tumor que contiene imágenes con tejido tumoral.
* TN es el directorio TumorNormal que contiene imágenes con tejido tumoral y sano.
* U es el directorio Undefined que contiene las imágenes que no están definidas.

## Descripción y tratamiento del dataset

Como ya se ha comentado anteriormente, el dataset consta de 1331 imágenes, las cuales están en formato TIF (Tagged Image File Format), un formato flexible para gestionar datos dentro de un mismo fichero, el cual incluye etiquetas con especificaciones y se utiliza especialmente para imágenes, ya que utiliza un método de compresión sin pérdidas de información (Wikipedia - TIFF, 2021).

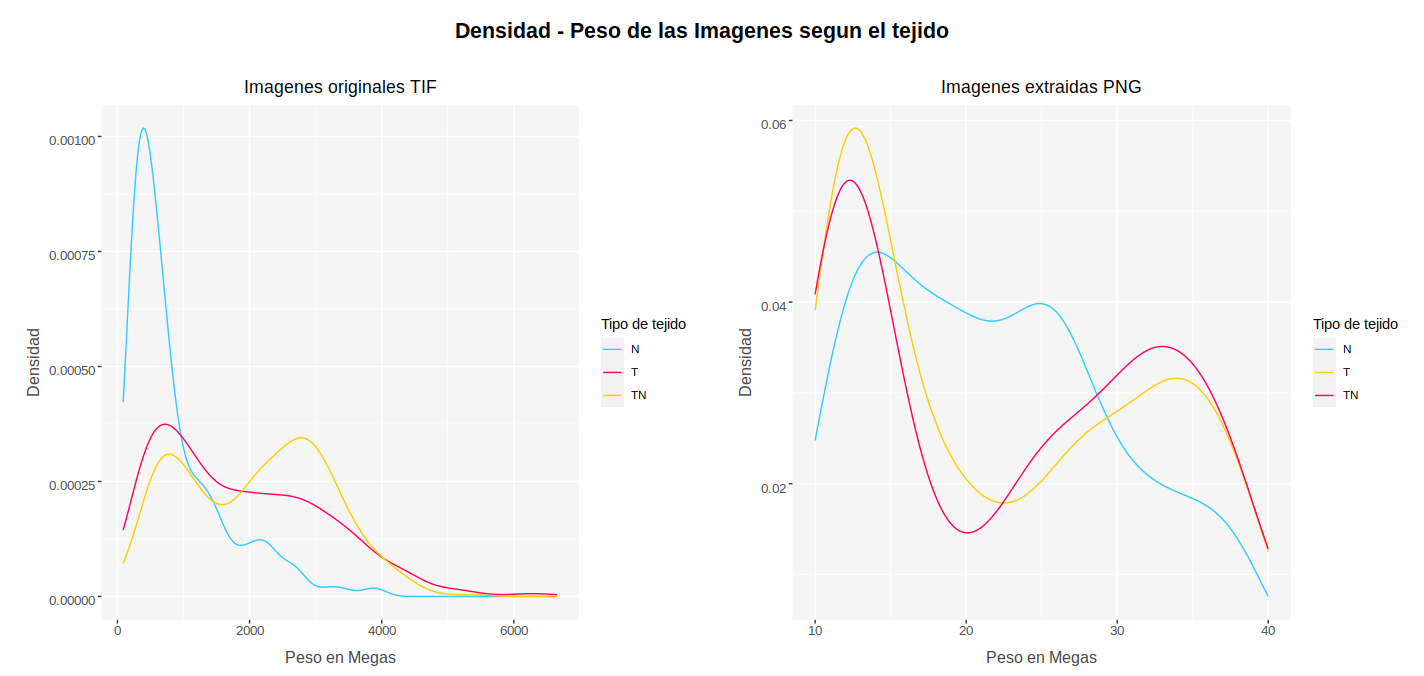
Las imágenes TIF de este dataset constan de 11 capas, cada una de ellas con una resolución distinta. A continuación se muestra un ejemplo de cómo se ven las capas en una imagen TIF [Figura X]:

|  |
| --- |
| # Code Blocks: **auto → mono-blue**  # auto → atom-one-light / paraiso-light / routeros (base) mpuiggros@uic.es:~/work/data/T\_10M$ identify /home/jovyan/images/TN/I21198.tif /home/jovyan/images/TN/I21198.tif[0] TIFF64 1251x3685 1251x3685+0+0 8-bit sRGB 2.77733GiB 0.000u 0:00.007 /home/jovyan/images/TN/I21198.tif[1] TIFF64 1251x3685 1251x3685+0+0 8-bit Grayscale Gray 0.000u 0:00.007 /home/jovyan/images/TN/I21198.tif[2] TIFF64 81920x64512 81920x64512+0+0 8-bit sRGB 0.000u 0:00.005 /home/jovyan/images/TN/I21198.tif[3] TIFF64 40960x32256 40960x32256+0+0 8-bit sRGB 0.000u 0:00.004 /home/jovyan/images/TN/I21198.tif[4] TIFF64 20480x16128 20480x16128+0+0 8-bit sRGB 0.000u 0:00.004 /home/jovyan/images/TN/I21198.tif[5] TIFF64 10240x8064 10240x8064+0+0 8-bit sRGB 0.000u 0:00.004 /home/jovyan/images/TN/I21198.tif[6] TIFF64 5120x4032 5120x4032+0+0 8-bit sRGB 0.000u 0:00.004 /home/jovyan/images/TN/I21198.tif[7] TIFF64 2560x2016 2560x2016+0+0 8-bit sRGB 0.000u 0:00.004 /home/jovyan/images/TN/I21198.tif[8] TIFF64 1280x1008 1280x1008+0+0 8-bit sRGB 0.000u 0:00.004 /home/jovyan/images/TN/I21198.tif[9] TIFF64 640x504 640x504+0+0 8-bit sRGB 0.000u 0:00.004 /home/jovyan/images/TN/I21198.tif[10] TIFF64 320x252 320x252+0+0 8-bit sRGB 0.000u 0:00.004 /home/jovyan/images/TN/I21198.tif[11] TIFF64 160x126 160x126+0+0 8-bit sRGB 0.000u 0:00.004 |

**Figura X:** Información de cada capa del formato TIF

El peso total del dataset es de 2.2TB de memoria, lo que implica que se debe tener en cuenta que se está tratando con Big Data en el momento de procesar los datos.

Tras una primera valoración de los datos, estudiando el peso de la imágenes, se vió que el peso de las imágenes fluctuaba entre 1MB, la imagen con menor peso, a 6.5GB, la imagen con un peso mayor [Figura X]. Es decir, que había una gran variabilidad del peso de las imágenes teniendo en cuenta que era un conjunto de imágenes similares con el mismo tipo de información. Tras esta observación, se comprobó que la transmisión de las imágenes fue realizada correctamente, no hubo ningún problema referente a la transmisión de datos.



**Figura X:** Densidad: Peso de las imágenes según el tejido

### Extracción de la capa según la resolución escogida

Una vez comprobado que los archivos no estaban corruptos, se inició el proceso de extracción de las capas. Se determinó que la extracción de las imágenes sería en formato PNG (Portable Network Graphics), un formato gráfico basado en un algoritmo de compresión que no sufre pérdidas por bitmaps, es decir, permite almacenar imágenes con una mayor profundidad de contraste (Wikipedia - PNG, 2021).

Primero se determinó que la imagen extraída debería tener un peso mínimo de 10MB (peso mínimo óptimo para poder realizar el modelo), con lo que se hicieron un conjunto de extracciones de diferentes resoluciones (capas) en diferentes imágenes escogidas aleatoriamente para determinar la resolución que más se aproximaba a un peso de 10MB.

Una vez determinada la resolución, se debe llevar a cabo la extracción de una capa para cada imagen. Se debe tener en cuenta que estamos trabajando con un gran volumen de datos (Big Data), así que cualquier operación que sea necesaria se tendrá que ver el coste que conlleva para poder adecuar y organizar el trabajo.

Considerando que la extracción de una capa en una imagen son alrededor de 6 segundos [Figura X]:

|  |
| --- |
| # Conversión de una capa a formato PNG time -p convert /home/jovyan/images/TN/I21198.tif[6] I21198\_6.png real 6.65 user 5.14 sys 0.27 |

**Figura X:** Extracción de una capa con el tiempo transcurrido

Calculamos el tiempo total que implicaría la extracción de una capa para todas las imágenes:

1331 \* 6 = 7986 segundos = 2.2 horas

Es decir, si se mira estrictamente el tiempo de ejecución, se tardarían alrededor de 2.2 horas en extraer una capa de cada imagen. De modo que hacerlo manualmente, contando el tiempo de búsqueda del nombre de la imagen y la preparación del comando (suponiendo que son alrededor de 10 segundos), se estima que llevaría alrededor de una 3.5 horas todo el proceso. De modo que para facilitar y agilizar el trabajo se decidió crear un script para poder llevar a cabo la extracción (facilitando también la repetición del proceso en un futuro debido a cambios de criterio o a la extracción incorrecta o fallida).

El script llamado extract\_img.sh (ver Anexo), a parte de automatizar la extracción de una capa (la escogida según la resolución adecuada en cada imagen), usa la paralelización de ejecuciones para reducir el tiempo de ejecución. Así que para ejecutar el script se le tienen que pasar 2 parámetros:

* Parámetro 1: El nombre, sin añadir la ruta, del directorio donde están las imágenes en formato tif.
* Parámetro 2: El número de procesadores deseados para poder paralelizar el proceso.

Con el script se consigue reducir el tiempo de ejecución notablemente. Para la extracción de 362 imágenes (las correspondientes a la carpeta TN), ejecutando el script con 10 CPUs se extraen las imágenes en 225 segundos (es decir 3,75 minutos), pero en cambio si se ejecuta con 20 CPUs el tiempo se reduce a 172 segundos (2,9 minutos). A continuación se muestra un ejemplo de dos ejecuciones (Figura X), la primera paralelizando con 10 CPUs y la segunda con 20 CPUs, las dos sobre la carpeta TN que, como ya se ha comentado, contiene 362 imágenes.

|  |
| --- |
| # CON 10 threads mpuiggros@uic.es:~/work/data$ time -p ./extract\_img.sh TN 10 &> log\_TN real 225.98 user 906.86 sys 47.33 |

|  |
| --- |
| # CON 20 threads mpuiggros@uic.es:~/work/data$ time -p ./extract\_img.sh TN 20 &> log\_TN real 172.37 user 907.55 sys 46.99 |

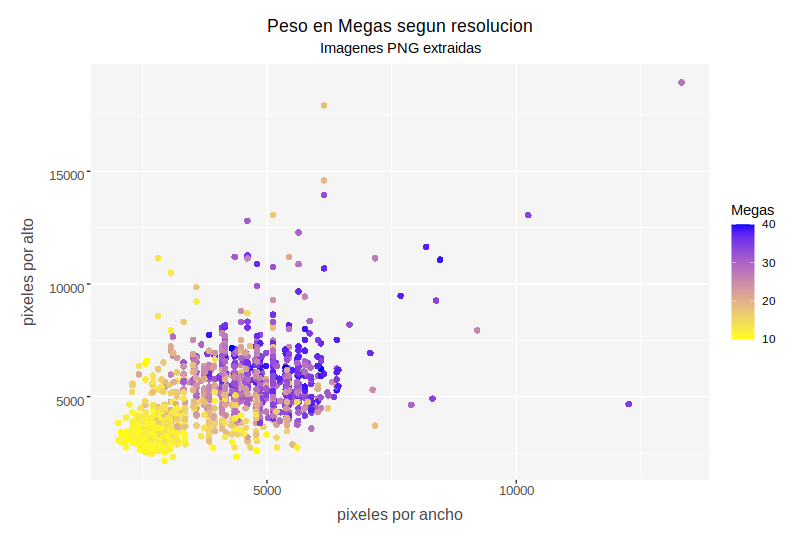
**Figura X:** Ejemplo del tiempo de ejecución

Para almacenar el output, el programa crea un directorio con el nombre “{parametro1}\_10M”, donde se guardarán las imágenes de la capa seleccionada de cada imagen TIF en PNG, y dichas imágenes conservan el nombre original del archivo TIF añadiendo el número de la capa correspondiente (ejemplo: I21198\_6.png, donde I21198 es el nombre del archivo TIF y 6 la capa extraída de dicho archivo).

### Verificación de la resolución

Una vez extraídas las capas según la resolución escogida, es necesario verificar que la imagen de la resolución escogida tiene un peso mínimo de 10MB. En caso contrario, se deberá extraer la capa más cercana que cumpla el requisito (la capa anterior).

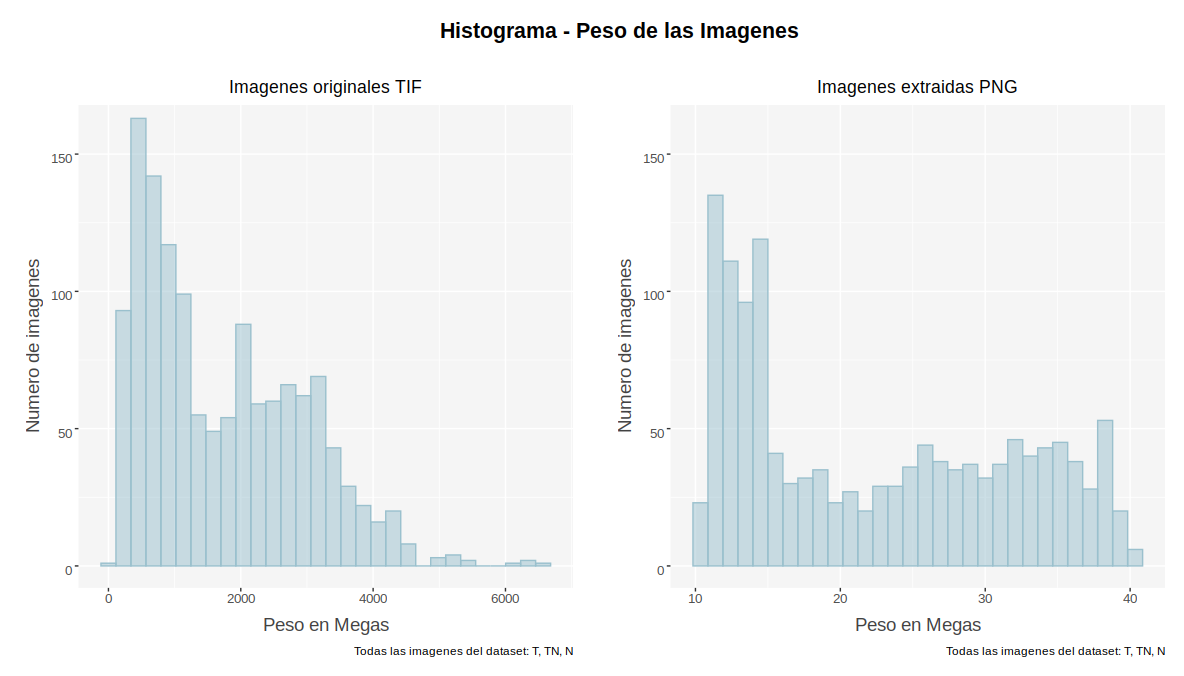
Para llevar a cabo este proceso, también se ha realizado un script, llamado extract\_img\_next\_layer.sh (ver Anexo), donde el input es el listado de las imágenes que se tienen que corregir, de las cuales detecta el número de capa para escoger una capa menor que contendrá la siguiente resolución mayor de la imagen original TIF. De este modo se podrá extraer una imagen en PNG con un peso mayor. Este proceso, también paralelizado del mismo modo que el anterior, se repetirá las veces necesarias hasta llegar a obtener todas las imágenes con el peso deseado, con un peso mínimo de 10MB. A continuación, en la Figura X, se puede observar como han quedado los archivos PNG extraídos, tanto en resolución como en peso.



**Figura X**: Peso en megas de las imágenes PNG según resolución.

Se puede observar que a mayor resolución mayor es el peso de la imagen, que es el comportamiento esperado, pero se ve que hay imágenes que teniendo una resolución elevada el peso no es muy alto. Esto podría indicar que hay imagenes de peor calidad, como por ejemplo que el color no esté muy definido o que la parte proporcional de color (es decir de tejido) sea muy pequeña. Esto se deberá tener en cuenta al hacer el EDA de las imágenes, para poder obtener un comportamiento adecuado en el modelo.

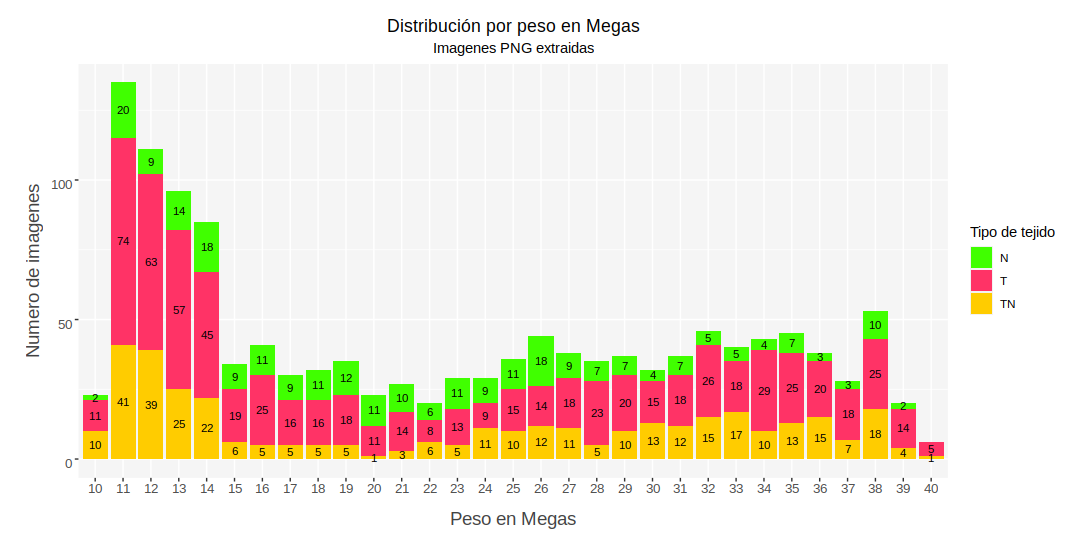
También es interesante observar la distribución de las frecuencias del peso, tanto en las imágenes en formato TIF cómo en PNG Figura X.1.



**Figura X.1:** Distribución de las frecuencias del peso

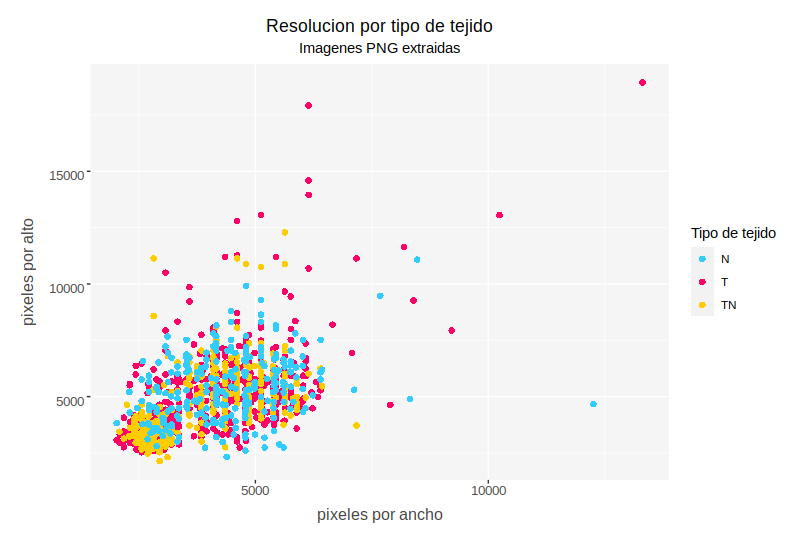
Las figuras anteriores muestran que hay una gran variabilidad en cuanto al peso en las imágenes TIF (las imágenes originales), van desde 1 MB a más de los 6000 MB, pero se puede observar que el rango más común está aproximadamente entre los 500 y 3000 MB. Esto se debe tener en consideración porque al hacer la extracción de las imágenes puede que tenga alguna influencia en la resolución y calidad, cosa que podría llegar a influenciar al modelo. Por otro lado, vemos que las imágenes PNG extraídas oscilan entre los 10 y 40 MB, teniendo en cuenta que la extracción de las imágenes se ha hecho extrayendo la resolución con un peso de 10 MB o la inmediatamente superior.

También se puede observar la distribución por peso según el tipo de tejido Figura X.2 y la resolución según el tipo de tejido Figura X.3.



**Figura X.2:** Distribución por peso según el tipo de tejido

Se observa que cada tipo de tejido tiene imágenes de todos los pesos, con lo que no hay ninguna concentración ni relación entre la imagen según el tipo de tejido y el peso del archivo. En los tres casos, el mayor conjunto de imágenes estaría entre los pesos de 11 y 14 MB. También observamos que hay un mayor número de imágenes de tejido Tumoral (Tumor y TumorNormal) que de tejido sano (Normal). Viendo estos resultados se puede decir que hay representación de todos los tipos de tejidos en los diferentes pesos y que no tiene porque ser un factor determinante para el modelo. Pero lo que sí que se tiene que tener en cuenta, es que las diferentes clases tienen variedad en el número de imágenes (la clase Tumor es mayoritaria), cosa que se estudiará en el EDA.



**Figura X.3:** Resolución según el tipo de tejido

Con la gráfica anterior se muestra que las resoluciones de las imágenes se concentran entre las resoluciones 2500x2500 y 7000x7000 aproximadamente. Vemos que la concentración no se distribuye a lo largo de la diagonal, con lo que podemos afirmar que la mayoría de las imágenes no son cuadradas, sino que tienden a ser alargadas o apaisadas. También se aprecia algún outlier, algunas imágenes con resoluciones muy superiores al resto. Con lo referente a los distintos tipos de tejidos, se puede decir que la variedad de resolución afecta a los tres grupos de forma similar, no hay concentraciones específicas agrupando los tejidos según las resoluciones. La forma de la imagen, que no es cuadrada, se deberá de tener en consideración al crear el modelo.

### Tecnología usada

Todo este proceso se ha llevado a cabo utilizando el lenguaje de programación bash, ya que es el lenguaje más directo para poder tratar todo lo referente a la información que atañe al sistema operativo, como es el peso de las imágenes, el listado de los ficheros, la conversión de formato, etc.

Como recordatorio a lo que se ha comentado anteriormente, el nombre de los scripts utilizados se muestran a continuación, así como el modo de ejecución.

1. extract\_img.sh

|  |
| --- |
| $> ./extract\_img.sh [N/TN/T] [#CPUs] &> log\_file |

1. extract\_img\_next\_layer.sh

|  |
| --- |
| $> ./extract\_img\_next\_layer.sh dir/list\_name\_file [#CPUs] |

## EDA

# Tratamiento estadístico de las imágenes

## Estratificación

# Modelo: Aplicación de Deep Learning al problema

La detección de cáncer a través de una imagen histopatológica la llevan a cabo los patólogos, que en un día pueden llegar a ver decenas de pacientes. La mayor parte de los diagnósticos son benignos, pero en el momento que se diagnostica un caso de cáncer se tiene que actuar con celeridad y el patólogo acaba trabajando estrechamente con el oncólogo (La vida en rosa, 2018). De modo que un patólogo, no sólo diagnostica si el paciente tiene cáncer, sino que se involucra en el procedimiento, cosa que hace que invierta mucho tiempo y recursos en cada caso.

La propuesta que se realiza para agilizar este proceso es habilitar un sistema que pueda reconocer con gran especificidad si la imagen histopatológica se trata de un tejido sano, de forma que el patólogo focalice todo su tiempo y recursos a los casos dudosos de cáncer o a aquellos que el diagnóstico es claro.

De modo que la propuesta realizada es crear un clasificador binario supervisado para determinar si las imágenes histopatológicas son normales, solo se aprecia tejido sano, o imágenes tumorales, donde el tejido es parte o íntegramente cancerígeno. Dicha clasificación se realizará mediante la extracción de características de alto nivel mediante diferentes capas mediante la técnica de Deep Learning (Wikipedia - Deep Learning, 2021), concretamente mediante redes neuronales convolucionales (conocida como CNN, Convolutional Neural Networks).

## Clasificador

Para llevar a cabo el clasificador se plantea utilizar la técnica del “*transfer learning*”, una de las técnicas más importantes del Deep Learning, donde un modelo desarrollado para realizar una tarea específica es reutilizado como punto de partida para otro modelo con un propósito diferente (*A Gentle Introduction to Transfer Learning for Deep Learning*, 2017). Por ejemplo, un modelo entrenado para reconocer coches puede ser usado como pre-entreno de un modelo para reconocer tractores. Se pueden encontrar diferentes modelos pre-entrenados con grandes conjuntos de datos (como ImageNet) donde se han entrenado los parámetros de la red neuronal, con los que se tiene un modelo entrenado, que se pueden usar posteriormente con un modelo de propósito distinto (Towards data science - What is TL, 2017).

### Transferencia específica: Paper seleccionado

#### Modelo 1

En el caso del clasificador de imágenes Normales y Tumorales, se ha querido utilizar la técnica de transfer learning para poder aprovechar el conocimiento específico ya generado en el campo. Para ello, después de realizar una búsqueda en la literatura específica sobre modelos aplicados en imágenes histopatológicas de cáncer, se optó por replicar la metodología utilizada en el paper “*Accurate diagnosis of colorectal cancer based on histopathology images using artificial intelligence*”, donde se predice con alrededor del 95% y 99% de sensitividad y de especificidad respectivamente (Wang, 2021) si la imágen es tumoral o no. Este paper fue escogido por los resultados, por la similitud del estudio y porque proporcionaban el código aplicado.

De modo que se empezó descargando el código proporcionado en el github <https://github.com/csu-bme/DeepPathology-CRC> y se trasladó al entorno facilitado para desarrollar el modelo (un servidor con sistema operativo ubuntu y 2 GPUs al cual se accede mediante la conexión a un docker con Jupyter). Es importante conocer la tecnología utilizada en el paper así como el software y las versiones para poder llegar a replicar el estudio y aplicarlo de este modo a los datos propios.

La tecnología del paper se basa en el Python v2.7.15, el Tensorflow-GPU v1.8.0 y la arquitectura aplicada del modelo es la Inception-v3. Por otro lado, en el sistema facilitado, el Python instalado tenía la versión Python v3.8.8 mientras que el TensorFlow-GPU era la versión Tensorflow v2.3. De modo que se optó por equiparar las versiones de ambos sistemas. Al estar en un entorno Conda, se optó por utilizar la opción de entornos virtuales para realizar las instalaciones y así evitar posibles conflictos entre las diferentes versiones. Se llevó a cabo la instalación de Python v2.7.15 sin problemas pero al intentar instalar el Tensorflow-GPU v1.8.0 tuvimos varios errores:

* La versión de CUDA 10.0, necesaria para el TensorFlow v1.8, no era compatible con la que tenía el sistema, la CUDA 9.0, de modo que se actualizó a la versión CUDA 11.2, pero tampoco dió resultado y la alternativa encontrada era utilizar una versión equivalente de TensorFlow v1.8, el tf-nightly, (Github - RTX3080 install RuntimeError, 2021) pero éste tenía problemas de dependencias que entraban en conflicto con las librerías del sistema. Al ver la imposibilidad de instalar la misma versión de Tensorflow por temas de incompatibilidad con las GPUs disponibles, se optó por instalar la versión de TensorFlow 1.7.0, que se logró instalar después de solventar los problemas de dependencias.

Una vez preparado el sistema con el software pertinente se prosiguió con la ejecución del código proporcionado por el paper. El cual se divide en 3 partes:

1. **1\_convert\_data**

El código de esta parte tiene como objetivo transformar las imágenes originales con formato JPEG, JPG o PNG, a imágenes TFRecord que es un formato para almacenar una secuencia de registros binarios (TensorFlow - TFRecord, 21). Esta parte consta de un solo script con 172 líneas de código.

1. **2\_training**

El código de esta parte, que consta de cuatro archivos y 747 líneas en total, realiza el entrenamiento del modelo partiendo de la arquitectura Inception-v3 con los pesos obtenidos del dataset ImageNet.

1. **3\_predict**

Por último está la parte de predicción donde, a partir de unas imágenes, te predice la categoría a la que pertenecen. Esta parte está compuesta por 14 archivos y 2544 líneas de código.

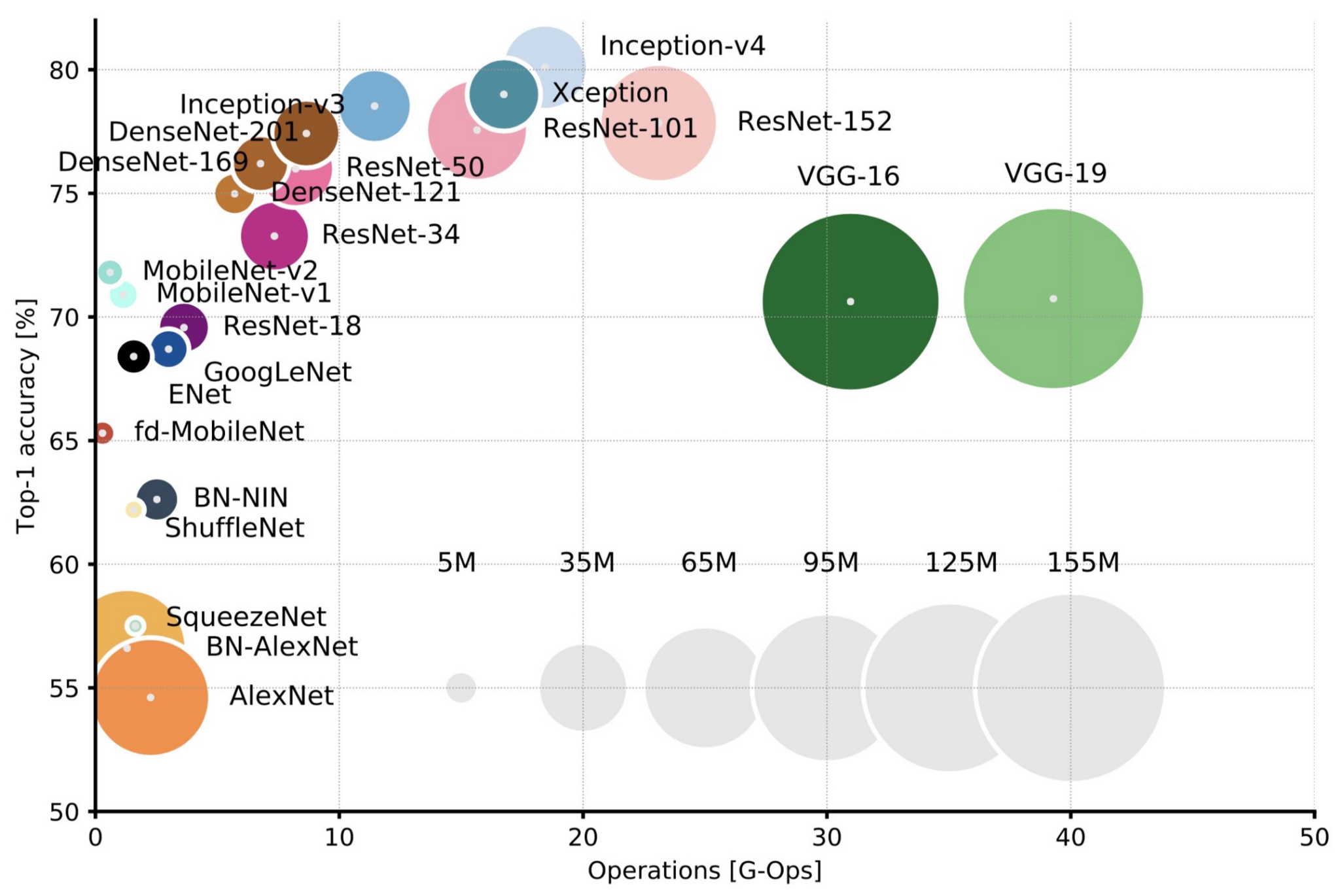
Inicialmente se escogió aleatoriamente un pequeño grupo de las imágenes a identificar y se intentó ejecutar el código dado, pero hubieron varios problemas:

* Aunque en teoría el código aceptaba imágenes en formato PNG, se obtuvieron varios errores que se acabaron resolviendo transformando las imágenes del dataset a JPG. Seguidamente se encontraron problemas de incompatibilidad de versiones de TensorFlow, ya que el paper usaba la versión TensorFlow 1.8 y en el sistema se pudo instalar la verisión TensorFlow 1.7. Un ejemplo de este tipo de errores sería intentar llamar la función exists(file) cuando se detectaba Exists(file), había un cambio en el nombre de la función. De modo que después de solucionar varios errores de compatibilidad de versiones se logró ejecutar el código y obtener así la conversión de las imágenes JPG a TFRecord.

Seguidamente se procedió a utilizar la tercera parte del código, la predicción, ya que no se quería reentrenar el modelo. Pero para realizar la predicción aprovechando el conocimiento previo del modelo se tuvieron que cargar el archivo de checkpoint, que estaba en formato zip dividido (varios zip consecutivos forman uno de solo más grande), el cual se tuvo que descomprimir. Este archivo proporciona a la arquitectura del modelo (en este caso la Inception-v3) el conocimiento previo, aunque este formato es complejo de manejar (en las versiones posteriores de TensorFlow la transferencia está simplificada). Una vez descomprimido se tuvo que cambiar la parte del código pertinente para que cogiera dichos pesos y se intentó correr el código para poder llegar a obtener el modelo de predicción. La ejecución, como pasó anteriormente, empezó a tener varios errores de versiones y viendo que el código tenía 2544 líneas (14 veces más código que la primera parte) se decidió que no era viable resolver cada una de las incidencias que podían ir saliendo, ya que el tiempo estaría invertido en solucionar unos problemas que no atañen al objetivo del máster, así que se decidió probar una estrategia diferente.

### Transferencia general: Arquitecturas estándares

Hay un gran abanico de arquitecturas estándares disponibles para aplicar el transfer learning. En la Figura X se puede ver una comparativa de las arquitecturas según su accuracy y el coste computacional.



**Figura X:** Comparación de las redes

(Fuente de la imagen: Medium - Analysis of deep neural networks, 2018)

Analizando este gráfico, y teniendo en cuenta las arquitecturas facilitadas por Keras (TensorFlow - Arquitecturas, 2020), se seleccionaron dos tipos para compararlos y poder escoger el mejor para llevar a cabo el modelo. La MobileNet-v2, ya que tiene una accuracy bastante buena y un nivel de operaciones muy bajo, y la Inception-v3 ya que es una de las más utilizadas, como en el paper que se seleccionó para hacer la prueba del transfer learning específico. Como el objetivo es comparar ambas arquitecturas, estas serán las únicas diferencias en el código, con lo que cada uno de los pasos están en ambas por igual.

Como en este caso no cabía la posibilidad de tener incompatibilidades con otro código, se optó por trabajar con las últimas versiones tanto de Python, la v3.8.8, como de TensorFlow, la v2.4.1, es decir, la instalación predeterminada del sistema.

#### Modelo 2: MobileNet - Modelo 3: Inception-v3

Inicialmente se empezó a trabajar con la arquitectura MobileNet-v2 partiendo de un ejemplo existente (TensorFlow MobileNetv2 example, 21), porque el coste computacional es menor que el de la arquitectura Inception-v3, cosa que facilitaba el proceso de creación del modelo.

Primero se replicó el ejemplo íntegro usando las imágenes dadas por este, imágenes de perros y gatos, para corroborar que dicho software funcionaba sin problemas en el sistema proporcionado para realizar el proyecto. Una vez estaba en funcionamiento se aplicó el modelo al dataset de cáncer, primero con un subconjunto de las imágenes y seguidamente con el dataset completo, siempre en formato PNG, sin hacer ninguna conversión adicional.

Como el modelo seleccionaba las categorías de las imágenes a través de los directorios, estas se tuvieron que organizar según el grupo al que pertenecían (perros o gatos). De modo que para aplicarlo al dataset del cáncer de colon se replicó la estructura de los directorios. Primero se separaron aleatoriamente por fases, según si eran imágenes de train (70% de las imágenes), validación (10% de las imágenes) o test (20% de las imágenes) y dentro de cada grupo, dos directorios separando las imágenes Normales de las Tumorales (N y T) de forma aleatoria aplicando los porcentajes indicados anteriormente.

Para poder realizar el modelo inicialmente se preparó todas las variables relacionadas con el sistema: librerías, detección de las GPUs, prevención de la manipulación completa de la memoria por parte de la GPU usada, etc. Se tuvieron problemas con la selección de las GPUS, ya que escogía siempre la GPU:0 y daba problemas de memoria debido a que estaba ocupada con otro proceso, de modo que se fijó la selección de la GPU:1 para evitar dichos problemas (Figura X). Para poder controlar el tiempo de ejecución se decidió imprimir la fecha y la hora al inicio y al final del proceso.

|  |
| --- |
| **import** matplotlib.pyplot **as** plt **import** numpy **as** np **import** os **as** os **import** argparse **as** argparse # parser input arguments script **import** tensorflow **as** tf  **from** tensorflow.keras.preprocessing **import** image\_dataset\_from\_directory **from** datetime **import** datetime **from** tensorflow.keras.callbacks **import** TensorBoard **from** sklearn.metrics **import** confusion\_matrix  # Get date / time to control it now = datetime.now() dt\_string = now.strftime("%d/%m/%Y %H:%M:%S")  # Get a specific GPU gpus = tf.config.experimental.list\_physical\_devices('GPU') **if** gpus:  **try**: # Restrict TensorFlow to only use the first GPU  tf.config.experimental.set\_visible\_devices(gpus[1], 'GPU')  **except** RuntimeError **as** e:  print(e) # Visible devices must be set at program startup  # To prevent get all GPU memory os.environ['TF\_FORCE\_GPU\_ALLOW\_GROWTH'] = 'true' AUTOTUNE = tf.data.experimental.AUTOTUNE  # Init seed tf.random.set\_seed(9)  print("\n\nDATE:", dt\_string, "\n\n") print("Tensorflow version ", tf.\_\_version\_\_, "\n") print("Num GPUs Available: ", len(tf.config.list\_physical\_devices('GPU'))) |

**Figura X:** Preparación del sistema

Como se tenía en mente aplicar una hyperparametrización para poder mejorar el modelo, y con el sistema proporcionado no se podía aplicar un método automático, se decidió preparar el código para poderse ejecutar mediante la línea de comandos (la terminal) pasándole los argumentos específicos de la hyperparametrización (Figura X) (ver apartado Hyperparametrización para ver más información). De modo que se crearon distintas variables, como por ejemplo el ID de la ejecución, el número de EPOCHS, el IMAGE\_SIZE, etc. para poder darles valores específicos en cada ejecución.

|  |
| --- |
| # Execution example: python model\_MobileNet-v2.py -hp 2 -e 20 &> model\_MobileNet-v2\_HP-2.log &  ## Get arguments: if no arguments use the default ones parser = argparse.ArgumentParser(description= "Create a list of germline mutations from vcf (allele 1, allele2)") parser.add\_argument('-od', dest='OUT\_DIR', default='result\_files', help='Output directory') parser.add\_argument('-oi', dest='OUT\_IMG', default='original\_img\_example.pdf', help='File name image examples pdf') parser.add\_argument('-oa', dest='OUT\_ACC', default='accuracy\_graph.pdf', help='File name accuracy graphic pdf') parser.add\_argument('-op', dest='OUT\_PRED', default='out\_predicted\_img.pdf', help='File name predicted imgs - graphic pdf') parser.add\_argument('-oc', dest='OUT\_CONF', default='out\_confusion\_matrix.pdf',  help='File name confusion matrix - graphic pdf') parser.add\_argument('-hp', dest='HYPER\_PARAM', default='0', help='ID hyperparam', type=str) parser.add\_argument('-e', dest='EPOCHS', default=10, help='Epochs number', type=int) parser.add\_argument('-s', dest='IMG\_SIZE\_VAL', default=160, help='Input pixels image (i.e. 160)', type=int) parser.add\_argument('-lr', dest='LEARNING\_RATE', default=0.0001, help='Learning rate (i.e. 0.0001)', type=float)  # Get the argument values args = parser.parse\_args() OUT\_DIR = args.OUT\_DIR OUT\_IMG = args.OUT\_IMG OUT\_ACC = args.OUT\_ACC OUT\_PRED = args.OUT\_PRED OUT\_CONF = args.OUT\_CONF HYPER\_PARAM = args.HYPER\_PARAM EPOCHS = args.EPOCHS IMG\_SIZE\_VAL = args.IMG\_SIZE\_VAL LEARNING\_RATE = args.LEARNING\_RATE  TB\_DIR = "logs/TensorBoard" # If Tensorboard folder doesn't exist, then create it. **if** **not** os.path.isdir(TB\_DIR):  os.makedirs(TB\_DIR)  print("Created folder : ", TB\_DIR)  # If folder doesn't exist, then create it. **if** **not** os.path.isdir(OUT\_DIR):  os.makedirs(OUT\_DIR)  print("Created folder : ", OUT\_DIR)  # Creating out path img examples out\_path = OUT\_DIR + "/HP-" + HYPER\_PARAM + "\_"+ OUT\_IMG  # Creating out path accuracy out\_path\_accuracy = OUT\_DIR + "/HP-" + HYPER\_PARAM + "\_"+ OUT\_ACC  # Creating out path predicted images out\_path\_predicted\_img = OUT\_DIR + "/HP-" + HYPER\_PARAM + "\_"+ OUT\_PRED  # Creating out path confusion matrix out\_path\_confusion\_matrix = OUT\_DIR + "/HP-" + HYPER\_PARAM + "\_"+ OUT\_CONF  # Creating out path model out\_path\_model\_h5 = OUT\_DIR + "/HP-" + HYPER\_PARAM + "\_model\_MobileNet-v2.h5"  print("\n-------------------------------------------------") print("Arguments") print("Output directory: ", OUT\_DIR) print("Output pdf images: ", OUT\_IMG) print("Output accuracy pdf graph:", OUT\_ACC) print("") print("Path img\_examples: ", out\_path) print("Path accuracy graphics: ", out\_path\_accuracy) print("Path predicted imgs: ", out\_path\_predicted\_img) print("Path confusion matrix: ", out\_path\_confusion\_matrix) print("") print("HYPER\_PARAM ID: ", HYPER\_PARAM) print("EPOCHS number: ", EPOCHS) print("IMAGE\_SIZE: ", IMG\_SIZE\_VAL) print("LEARNING\_RATE: ", LEARNING\_RATE) print("-------------------------------------------------\n") |

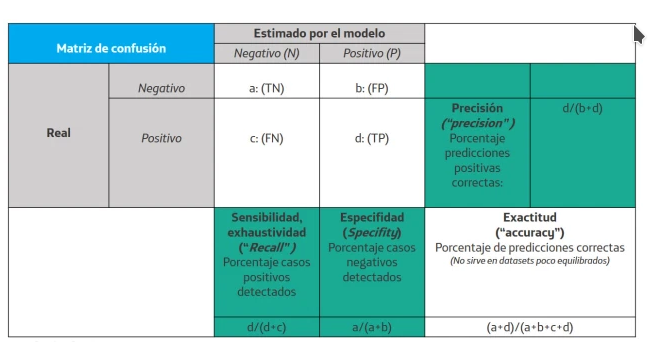
**Figura X:** Preparación de argumentos para la hyperparametrización

Después de la preparación del sistema y de los argumentos creados para poder mejorar el modelo, se llevó a cabo la lectura del dataset de train y validación (Figura X).

|  |
| --- |
| # tumor - normal PATH = '/home/jovyan/work/model/transfer\_learning/NAC/data\_test\_meu\_tot/' train\_dir = os.path.join(PATH, 'train') validation\_dir = os.path.join(PATH, 'validation')  print("\n :: Paso 1. Definir datasets\n") BATCH\_SIZE = 32 IMG\_SIZE = (IMG\_SIZE\_VAL, IMG\_SIZE\_VAL)  train\_dataset = image\_dataset\_from\_directory(train\_dir,  shuffle=**True**,  batch\_size=BATCH\_SIZE,  image\_size=IMG\_SIZE)  validation\_dataset = image\_dataset\_from\_directory(validation\_dir,  shuffle=**True**,  batch\_size=BATCH\_SIZE,  image\_size=IMG\_SIZE)  # Getting class names from dataset class\_names = train\_dataset.class\_names |

**Figura X:** Lectura del dataset de train y validación

Una vez se comprobó que el código también funciona correctamente con las imágenes de cáncer de colon, se empezó el proceso de hyper parametrización, que se explicará más adelante.



<https://empresas.blogthinkbig.com/como-interpretar-la-matriz-de-confusion-ejemplo-practico/>

**Exactitud (ACCURACY)-** Si recordamos, la **exactitud** (o “*accuracy*“) representa el porcentaje de predicciones correctas frente al total. Por tanto, es el cociente entre los casos bien clasificados por el modelo (verdaderos positivos y verdaderos negativos, es decir, los valores en la diagonal de la matriz de confusión), y la suma de todos los casos.

Sin embargo, *cuando un conjunto de datos es poco equilibrado, no es una métrica útil.* Por ejemplo, si lo que intentamos predecir es una enfermedad rara, y nuestro algoritmo clasifica a todos los individuos como sanos, podría ser muy preciso (incluso un 99%), pero también, totalmente inútil.

Para poder probar un modelo con nuestros datos escogimos un ejemplo e intentamos replicarlo.

Con este modelo utilizamos las imágenes png directamente extraidas de las imágenes originales TIF, inicialmente no se ha hecho ninguna reconversión.

Este ejemplo usa la arquitectura MobileNet-v2

También utilizamos una normalización del color de la imagen entre [-1,1] ya que son imágenes recopiladas a lo largo del tiempo (20 años) donde el proceso de tinción, aunque se ha usado la misma técnica, las personas y los recursos utilizados para ello son distintos.

La primera ejecución de prueba se ha hecho sobre un pequeño conjunto de imágenes cogidas aleatoriamente en ambos grupos (T/N). Con este “toy example” hemos obtenido una accuracy del 0.8. Tenemos que ver cómo se comporta aplicando todas las imágenes.

Al añadir el TensorBoard el incremento de tiempo de ejecución es del

##### Hyperparametrización

### Comparativa: MobileNet vs. Inception-v3

Comparación con el excel hyperparametrización

### Modo de trabajo (jupyter → py / github / excel hyperparametrización / test paralelo petó…, tensorflow / keras)

* jupyter nbconvert --to script test\_1.ipynb
* Excel : Explicar como hemos trabajado con el modelo

El entorno no era muy “user-friendly” → hemos hecho un procedimiento para que todo el equipo pudiera trabajar cómodamente sin necesidad de tener conocimientos de sistemas operativos.

## Localizador - LO QUITAMOS y lo ponemos a futuro ???

### Modelo 4 - Localización

# 

# 

# Conclusiones

# Propuesta de valor

## Posibles mejoras

* Lectura de las imágenes en TFRecords
* Añadir RNN (referència papers) para mejorar el modelo actual
* Añadir ejecución multi-GPU → Mejora de recursos
* Estudiar los optimizadores y ver si hay alguno que se adapta mejor

<https://keras.io/api/optimizers/>

* [SGD](https://keras.io/api/optimizers/sgd)
* [RMSprop](https://keras.io/api/optimizers/rmsprop)
* [Adam](https://keras.io/api/optimizers/adam) → Utilizado
* [Adadelta](https://keras.io/api/optimizers/adadelta)
* [Adagrad](https://keras.io/api/optimizers/adagrad)
* [Adamax](https://keras.io/api/optimizers/adamax)
* [Nadam](https://keras.io/api/optimizers/Nadam)
* [Ftrl](https://keras.io/api/optimizers/ftrl)

## Aplicaciones futuras

* ??? Crear clasificadores de detección de daños de la imagen? Para automatizar y agilizar el EDA de un nuevo set de imágenes??
* Localizador del tumor?
* Relacionar la clasificación de la imagen con un nuevo modelo de detección de mutaciones? → Paper Santi
* Detección del grado del cáncer

<https://www.juntscontraelcancer.cat/cancer/diagnostic-de-cancer/>

# Ética

## Protección de datos

* ID de las imagenes = paciente
* no relacionar id con otros estudios
* no intentar averiguar nombre del paciente

## “Valores humanos”

# Planificación del proyecto ???

# 

# Bibliografía

Depto. de Biología Funcional y Ciencias de la Salud. Facultad de Biología. Universidad de Vigo. (2020, 06 23). *Tinciones Generales*. Técnicas histológicas. https://mmegias.webs.uvigo.es/6-tecnicas/5-general.php#he

*A Gentle Introduction to Transfer Learning for Deep Learning*. (2017, 12 20). Machine Learning Mastery. https://machinelearningmastery.com/transfer-learning-for-deep-learning/

Github - RTX3080 install RuntimeError. (2021, 10 1). *RTX3080 install RuntimeError*. Github. https://github.com/tensorflow/tensorflow/issues/43701

La vida en rosa. (2018, 11 14). *Patólogos: la vida tras un microscopio*. https://revistalvr.es/patologos-la-vida-tras-microscopio/

Medium - Analysis of deep neural networks. (2018, 09 08). *Analysis of deep neural networks*. Analysis of deep neural networks. https://culurciello.medium.com/analysis-of-deep-neural-networks-dcf398e71aae

TensorFlow - Arquitecturas. (2020, 12 14). *Module: tf.keras.applications*. Module: tf.keras.applications. https://www.tensorflow.org/api\_docs/python/tf/keras/applications

TensorFlow - TFRecord. (21, 05 20). *TFRecord y tf.train.Ejemplo*. TFRecord y tf.train.Ejemplo. https://www.tensorflow.org/tutorials/load\_data/tfrecord

Towards data science - What is TL. (2017, 11 17). *What is Transfer Learning?* Towards data science. https://towardsdatascience.com/what-is-transfer-learning-8b1a0fa42b4

Wang, K. S. (2021). Accurate diagnosis of colorectal cancer based on histopathology images using artificial intelligence. *BMC Medicine*, *10*(1186), s12916-021-01942-5. https://bmcmedicine.biomedcentral.com/articles/10.1186/s12916-021-01942-5

Wikipedia - Deep Learning. (2021, 05 09). *Deep learning*. Deep learning. https://en.wikipedia.org/wiki/Deep\_learning

Wikipedia - HE. (2021, 04 10). *H&E stain*. H&E stain. https://en.wikipedia.org/wiki/H%26E\_stain

Wikipedia - PNG. (2021, 02 04). *PNG*. PNG. https://ca.wikipedia.org/wiki/PNG

Wikipedia - TIFF. (2021, 05 15). *TIFF*. TIFF. https://en.wikipedia.org/wiki/TIFF

# Anexo

## Scripts de extracción de las imágenes

**Script *extract\_img.sh***

|  |
| --- |
| *# EXECUTIONS Examples* *#time -p ./extract\_img.sh TN 10 &> log\_TN* *#time -p ./extract\_img.sh T 10 &> log\_T*  *# input* dir\_samples=$1 *# TN / T ==> MANDATORI* threads=$2; *# number threads*  *#############################################* *# Checking input arguments*  *# Checking if there is input argument 1* if [ -z "$1" ]; then  threads=5;  echo -e "\nImage folder name is needed (TN / T)\n"  exit -1 fi *# Checking if there is input argument 2* if [ -z "$2" ]; then  threads=5; fi echo -e "\nNumber of threads = $threads\n"  out\_dir="${dir\_samples}\_10M" mkdir -p $out\_dir *# Creating dir*  *#############################################* *# Getting ~10M resolutions from original files*  echo -n -e "\nDATE get ALL resolutions: "; date echo "" echo -e "\nGetting ALL resolutions from original files..." echo -e "\tFile: resolution\_aprox10M\_${dir\_samples}" for i in `ls /home/jovyan/images/${dir\_samples}/\*`; do identify $i | grep -P " [2-3][0-9][0-9][0-9]x"; done > resolution\_aprox10M\_${dir\_samples} echo -e "DONE!\n"  *#######################################* *# Preparing cmd with resolution ~ 10M*  echo -n -e "\nDATE get 10M resolution: "; date echo "" echo -e "\nPreparing commands..." echo -e "\tFile: resolution\_aprox10M\_${dir\_samples}\_cmd" for i in `cat resolution\_aprox10M\_${dir\_samples} | cut -d' ' -f1`; do id=`echo $i | sed 's/.\*\[//g' | sed 's/\]//g'`; filename=`echo $(basename $i) | cut -d'.' -f1`; echo "convert $i $out\_dir/${filename}\_${id}.png"; done > resolution\_aprox10M\_${dir\_samples}\_cmd echo -e "Comands DONE!\n"   *#############################################* *# PARALLEL execution*  echo -n -e "\n\nDATE: extracting Images "; date echo ""  count=1; tmp\_file="tmp\_file\_${dir\_samples}" rm $tmp\_file PID\_LIST="" cat resolution\_aprox10M\_${dir\_samples}\_cmd | while read line; do  echo $line  $line &  PID="$!"  echo "$PID" >> $tmp\_file  PID\_LIST+="$PID ";   if [ $count == $threads ]; then  count=1  echo "Waiting jobs IDs:"  for wait\_PID in `echo $PID\_LIST`; do  echo $wait\_PID  wait $wait\_PID  done  PID\_LIST=""  else  count=`expr $count + 1`  fi done echo -n -e "\nDATE: Images DONE! "; date |

**Script extract\_img\_next\_layer.sh**

|  |
| --- |
| *# EXECUTIONS example* *# ./extract\_img\_next\_layer.sh T\_10M/list\_T\_10M\_menor\_10M\_nextLayer\_1 T\_10M 4* *# ./extract\_img\_next\_layer.sh TN\_10M/list\_TN\_10M\_menor\_10M\_nextLayer\_1 TN\_10M 4* *# ./extract\_img\_next\_layer.sh N\_10M/list\_N\_10M\_menor\_10M\_nextLayer\_1 N\_10M 4*   *# input* file\_nextLayer=$1 *# list\_T\_10M\_menor\_10M\_nextLayer\_1 ==> MANDATORI* out\_dir=$2 *# T\_10M / TN\_10M / N\_10M* threads=$3; *# number threads*  in\_dir=`echo $out\_dir | cut -d'\_' -f1` dir\_samples="/home/jovyan/images/$in\_dir"  iteration\_layer=`echo $file\_nextLayer | sed 's/.\*\_//g'` cmd\_file="resolution\_aprox10M\_${in\_dir}\_cmd\_NextLayer\_${iteration\_layer}"  *##############################################################* *# Getting file name and layer for Preparing cmd with nextLayer*  echo -e "\nPreparing commands NextLayer..."  for line in `cat $file\_nextLayer`; do name=`echo $line | cut -d'\_' -f1`; layer=`echo $line | cut -d'\_' -f2 | cut -d'.' -f1`; nextLayer=`expr $layer - 1`; echo "convert $dir\_samples/$name.tif[$nextLayer] $out\_dir/${name}\_$nextLayer.png"; done > $cmd\_file  echo -e "Comands DONE!\n"  *#############################################* *# PARALLEL execution*  echo -n -e "\n\nDATE: extracting Images "; date echo ""  count=1; tmp\_basename=`basename $file\_nextLayer` tmp\_file="tmp\_file\_${tmp\_basename}" rm $tmp\_file PID\_LIST="" cat $cmd\_file | while read line; do  echo $line  $line &  PID="$!"  echo "$PID" >> $tmp\_file  PID\_LIST+="$PID ";   if [ $count == $threads ]; then  count=1  echo "Waiting jobs IDs:"  for wait\_PID in `echo $PID\_LIST`; do  echo $wait\_PID  wait $wait\_PID  done  PID\_LIST=""  else  count=`expr $count + 1`  fi done echo -n -e "\nDATE: Images DONE! "; date |