TRABAJO DE FIN DE MÁSTER

# Máster Interuniversitario en Estadística e Investigación Operativa UPC-UB

**Título: Deep Learning para la detección de patologías de cáncer de piel y generación de imágenes de tejidos humanos**

**Autor: Sandra Redondo Hernández**

**Directores: Ferran Reverter Comes y Esteban Vegas Lozano**

**Departamento: Genética, Microbiología y Estadística. Sección Estadística.**

**Universidad: Universidad Politécnica de Catalunya**

**Convocatoria: Enero 2020**



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT DE BARCELONA

Facultad de Matemáticas y Estadística
Universidad Politécnica de Cataluña

Máster en Estadística e Investigación Operativa
Trabajo final de máster – Área: Data Science

# Deep Learning para la detección de patologías de cáncer de piel y generación de imágenes de tejido humano

Sandra Redondo Hernández

Directores: Ferran Reverter Comes y Esteban Vegas Lozano
Departamento: Genética, Microbiología y Estadística. Sección Estadística.

Este trabajo no hubiera sido posible sin mis padres ya que me han enseñado día tras día a superar mis metas y que el esfuerzo es la herramienta para el éxito. Además mi hermana Miriam por estar siempre a mi lado y apoyarme tanto en lo bueno como en lo malo. Mis amigos por darme ánimos y fuerzas en cada uno de mis proyectos. A todos los docentes que me han formado a lo largo de mi vida académica. Y finalmente a mis tutores que me han guiado por el maravilloso mundo de la Estadística y del Machine Learning.

# Prólogo

En los tiempos que corren, la ciencia del dato domina todos los ámbitos tanto económicos como científicos y tecnológicos. Sus aplicaciones van desde lo más banal como por ejemplo modificar una imagen según el estilo de un pintor hasta conseguir detectar patologías medicas a partir de una imagen. En consecuencia, para conocer lo que sucede hoy en día como por ejemplo los descubrimientos y los avances en el ámbito científico es necesario conocer el Machine Learning y la Estadística.

# Resumen

Las redes neuronales se han hecho un hueco en el mundo de la resolución de problemas de clasificación y predicción, entre otros. Gracias a las redes neuronales hoy en día es posible detectar e identificar objetos, predecir la siguiente secuencia en un video e incluso generar imágenes y música, además solucionan toda una serie de problemas en el ámbito biomédico como por ejemplo la detección de cáncer a partir de imágenes de tejido humano.

En este trabajo se investigará una serie de modelos que existen para hacer frente a un problema tan grave como es el cáncer de piel, el principal tipo de cáncer a nivel mundial. Se plantearán dos casos de uso relacionados con el cáncer de piel, el primero clasificar mediante redes neuronales convolucionales si una imagen de tejido humano es cancerígena o no y en segundo lugar generar imágenes de tejido humano que presenten las distintas patologías mediante redes generativas antagónicas, estas imágenes en un futuro se pueden utilizar para entender mejor los datos o bien incrementar el tamaño muestral de los estudios.

Para la elaboración del presente trabajo se utilizarán las imágenes extraídas vía dermatoscopia que almacena el ISIC (*International Skin Imaging Collaboration*).

La dermatoscopia[1] es una técnica de imagen que elimina el reflejo superficial de la piel. Al eliminar el reflejo de la superficie, se mejora la visualización de los niveles más profundos de la piel. Investigaciones han demostrado que, cuando la utilizan los dermatólogos expertos, la dermatoscopia proporciona una precisión diagnóstica mejorada, en comparación con la fotografía estándar.



**Palabras clave:** Redes Neuronales Convolucionales, Redes Generativas Antagónicas, Dermatoscopia, Cáncer de Piel


[1] ISIC

# Abstract

Neural networks have found their place in the world of solving classification and prediction problems, among others. Thanks to them today it is possible to detect and identify objects, predict the following sequence in a video and even generate images and music, and solve a whole series of problems in the biomedical field such as cancer detection from images of human tissue.

In this work we will investigate the different models that exist to deal with a problem as serious as skin cancer, the main type of cancer worldwide. Two cases of use related to skin cancer will be considered, the first to classify by means of convolutional neural networks if an image of human tissue is carcinogenic or not and secondly to generate images of human tissue that present the different pathologies through antagonistic generative networks. These future images can be used to better understand the data or increase the sample size of the studies.

For the elaboration of this work, the images extracted via dermatoscope stored by the ISIC (International Skin Imaging Collaboration) will be used.

Dermatoscopy[2] is an imaging technique that eliminates the superficial reflection of the skin. By eliminating the reflection of the surface, the visualization of the deeper levels of the skin is improved. Research has shown that, when used by expert dermatologists, dermatoscopy provides improved diagnostic accuracy, compared to standard photography.



**Key words:** Convolutional neuronal networks, Generative Antagonic Networks, Skin Cancer, Dermoscopy


[2] ISIC

# Notación

- Algoritmo del gradiente estocástico (SGD)
- Retropropagación (en inglés backpropagation)
- Juego minimax
- Equilibrio de Nash
- Convolución
- Interpolación bilineal
- Función de perdida
- Redes neuronales
- Redes generativas antagónicas

# Índice

# Ilustraciones

# Tablas

# 1 Introducción

## 1.1 Contexto y motivación

En el contexto actual, después de grandes avances tecnológicos que han facilitado el acceso a la población de ordenadores potentes o bien a servidores con hardware inigualables como por ejemplo AWS, EC2 y Google Colab entre otros, el Machine Learning y sobre todo el Deep Learning han encontrado el momento idóneo para incorporarse en la automatización de tareas en la sociedad.

El Deep Learning se esconde detrás de una multitud de herramientas de uso diario de la población, aunque no todo el mundo lo sabe. Un ejemplo de ello es el sistema de *Google reCAPTCHA* que consiste en identificar objetos o números de una imagen para determinar si el individuo es un humano o un robot, este sistema está basando en redes convolucionales. Por otro lado, *NVIDIA's Hyperrealistic Face Generator*, es un generador de imágenes de caras ultra realista que se puede aplicar en sectores tanto de publicidad o marketing como en videojuegos, su trasfondo son las redes generativas adversarias (GAN). Otra de sus muchas aplicaciones sonlos vehículos autónomos, los diagnósticos de cáncer de piel o de mama, los asistentes de voz de Apple, Google y Amazon, los traductores, etc.

La motivación este trabajo es recorrer el mundo del Machine Learning, especialmente en el Deep Learning aplicado en la "visión por computador" o *computer visión* y conocer las maravillas que se pueden hacer gracias a las redes convolucionales o GANs, desde generar imágenes a partir de un ruido aleatorio, transferir el estilo de Van Gogh a una imagen[3] e incluso detectar enfermedades. Además, desde la perspectiva estadística, las redes neuronales generativas adversarias presentan una serie aplicaciones muy importantes. En todo estudio de investigación, obtener una base de datos con las características deseadas es muy difícil, un ejemplo de ello sería el tamaño muestral, el cual puede limitar en muchas ocasiones las aplicaciones de ciertos modelos. Esta casuística se puede solucionar gracias a las GAN, si dispones de poca muestra, pero suficiente como para modelizar una GAN, podrás generar nueva muestra teniendo la certeza que los datos reales y los generados comparten espacio vectorial.

[3] DeepDream - Google

## 1.2 Objetivos

En este trabajo se estudiarán imágenes de cáncer de piel proporcionadas por el ISIC (*The International Skin Imaging Collaboration)*.

El proyecto ISIC fue creado con el objetivo de unir fuerzas entre todos los organismos y entidades relacionados con el tratamiento y diagnóstico de cáncer de piel para así unir el conocimiento sobre el melanoma y además desarrollar estándares de las imágenes de piel.

El Centro de Cáncer Memorial Sloan Kettering coordina el proyecto, varias sociedades profesionales participan en él, una de ellas es la International Dermoscopy Society (IDS) y la International Society for Digital Imaging of the Skin (ISDIS).

Las imágenes de las lesiones cutáneas se pueden utilizar para educar a los profesionales y al público sobre el reconocimiento del melanoma, así como para ayudar directamente en el diagnóstico del melanoma a través de la teledermatología, el apoyo a la decisión clínica y el diagnóstico automatizado. Actualmente, la falta de estándares para las imágenes dermatológicas perjudica la calidad y la utilidad de las imágenes de lesiones cutáneas, por esa razón el ISIC está desarrollando unos estándares para poder abordar las tecnologías, técnicas y terminología utilizadas en el diagnóstico con imágenes de la piel.

Con el presente trabajo se quiere conocer y estudiar las distintas herramientas existentes para el tratamiento de imágenes médicas, como es el caso las imágenes de cáncer de piel, centrándose en el Machine Learning y en especial el Deep Learning.

El Machine Learning va cogido de la mano de la computación y por esa razón se estudiarán los métodos y herramientas de software y de hardware que hacen posible la ejecución de los distintos algoritmos que se utilizaran, además se optimizara el coste computacional de ellos.

Principalmente se estudiarán las redes neuronales para el tratamiento de imágenes destacándose tres métodos: las redes neuronales convolucionales CNNs y las redes generativas antagónicas GANs y las redes generativas antagónicas convolucionales profundas DCGANs.

Así pues, se llevará a cabo un modelo de clasificación de imágenes a partir de una red neuronal convolucional para predecir si una lesión de piel es benigna o maligna. Además, se diseñará una red generativa adversaria que permita generar imágenes de piel permitiendo aumentar la muestra de imágenes que se dispone.

## 1.3 Enfoque y metodología

A partir de la definición de los objetivos y del hardware disponible se han desarrollado las siguientes estrategias relativas a tres aspectos clave del proyecto: una relativa a las características de los datos y a la obtención de estos, otra relativa al software y una tercera con respecto a la arquitectura de la red.

### Imágenes y fuente de datos

Las imágenes utilizadas en este estudio provienen del archivo de imágenes de cáncer de piel de ISIC *(International Skin Imaging Collaboration: Melanoma Project), e*ste proyecto tiene como objetivo facilitar la aplicación de imágenes de la piel para ayudar a reducir la mortalidad por melanoma.

El dataset está constituido por 23.906 imágenes con sus correspondientes archivos .json que contiene el metadata con información de si es benigna o maligna, el diagnostico, la edad, el sexo, la parte del cuerpo de la lesión, la dimensión, el historial familiar, la clase de melanoma, etc.

El archivo ISIC contiene imágenes de diferentes fuentes:

*Tabla 1. ISIC Datasets*

| | Fuente | % |
|---|---|---|
| Dataset | HAM10000 | 41,89 |
| | SONIC * | 38,70 |
| | MSK-2 | 6,42 |
| | MSK-1 | 4,60 |
| | MSK-4 | 3,96 |
| | UDA-1 | 2,33 |
| | MSK-3 | 0,94 |
| | MSK-5 | 0,46 |
| | 2018 JID Editorial Images | 0,42 |
| | UDA-2 | 0,25 |
| | Dermoscopedia (CC-BY) | 0,02 |

* Se descartaron las imágenes de la fuente *SONIC* porque presentaban adhesivos encima de la piel y todas las imágenes de esta fuente eran de tipo benigno, en consecuencia, los modelos de aprendizaje supervisado detectarían y asociarían este patrón (tener o no adhesivo) con la categoría de respuesta. Esta fuente representa alrededor del 50% del total.

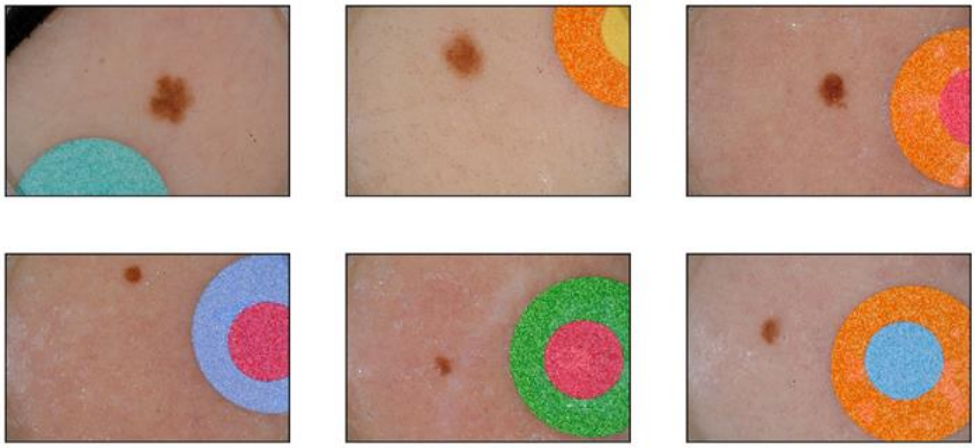


*Ilustración 1. Ejemplo imágenes fuente SONIC*

La distribución de la categoría de respuesta:

*Tabla 2. Distribución de la categoría de respuesta*

| | Tipo | % |
|---|---|---|
| Benign or Malignant | Benign | 69,07 |
| | Malignant | 15,60 |
| | Unknown* | 15,20 |
| | Indeterminate* | 0,10 |
| | Indeterminate/benign* | 0,02 |
| | Indeterminate/malignant* | 0,01 |

* Las imágenes correspondientes a “Indeterminate”, “Indeterminate/ benign”, “Indeterminate/ Malignant” y “Unknown” se han descartado de la modelización de la red convolucional pero no en el desarrollo de la red generativa adversaria ya que contra mayor número de imágenes mayor calidad.

### Hardware y software

En este apartado se describe de manera general el hardware y el software utilizado.

Se ha utilizado el framework para desarrollo de redes neuronales Tensorflow 2 que está integrado con Keras y además permite la utilización de los recursos de la GPU. Por otra parte, para su correcta utilización se ha instalado los drivers de la GPU de Nvidia, CUDA Toolkit 10, CUPTI, cuDNN SDK y TensorRT 5.

Para la realización de este proyecto se ha utilizado un ordenador con la siguiente configuración de hardware:

- Nvida Geforce RTX 2060 6GB
- Intel Core i7-9759H 2.60GHz
- RAM 16GB
- HDD SSD 1TB

En cuanto al software:

- Windows 10 como sistema operativo.
- Entorno virtual con Python 3.6.8 gestionado con conda y pip.
- Tensorflow 2
- Resto de paquetes utilizados para el desarrollo de la aplicación como numpy, matplotlib, PIL, etc...
- Jupyter notebooks para la creación de informes.

Las elecciones del sistema operativo y de Tensorflow como framework de desarrollo están determinadas por el tipo de hardware disponible (GPUs Nvidia). Conda se eligió sobre otros gestores de entornos ya que representa una opción más completa, y que además cumple funciones de gestión de paquetes y es compatible con pip. Para mostrar los resultados se prefirió el uso de informes interactivos ya que permiten una mayor flexibilidad a la hora de mostrar imágenes, animaciones o gráficas que los ficheros Python tradicionales.

El código utilizado para la elaboración del presente trabajo esta disponible en el repositorio de GitHub sredondoh/TFM.

# 2 Caso de uso: Cáncer de piel

El cáncer de piel es la forma más común de cáncer, representa globalmente al menos el 40% de los casos[4]. El tipo más común es el cáncer de piel no melanoma, que se da en al menos 2-3 millones de personas por año[5].

El cáncer de piel es una afección por la que se forman células malignas (cancerígenas) las diversas capas de la piel.

Hay tres tipos principales de cáncer de piel*: Cáncer de piel de células basales (BCC), cáncer de piel de células escamosas (SCC) y melanoma.* Los dos primeros, junto con una serie de cánceres de piel menos comunes, se conocen como cáncer de piel no melanoma (NMSC).

Los NMSC tienen un crecimiento lento y tienden a dañar solo el tejido que lo rodea, es poco probable que se extienda a áreas distantes o provoque la muerte a diferencia de los melanomas que son los más agresivos.

### Signos y síntomas

Hay una variedad de síntomas diferentes de cáncer de piel. Estos incluyen cambios en la piel que no sanan, úlceras en la piel, piel descolorida y cambios en los lunares existentes, como bordes irregulares del lunar y agrandamiento del lunar.

Cáncer de piel de células basales

El cáncer de piel de células basales (CCB) generalmente se presenta como una protuberancia elevada, lisa y perlada en la piel de la cabeza, el cuello o los hombros expuesta al sol. Algunas veces se pueden ver pequeños vasos sanguíneos (llamados telangiectasias) dentro del tumor. Con frecuencia se forman costras y sangrado en el centro del tumor. A menudo se confunde con una llaga que no cicatriza. Esta forma de cáncer de piel es la menos mortal y con el tratamiento adecuado puede eliminarse por completo, a menudo sin dejar cicatrices.

Cáncer de piel de células escamosas

El cáncer de piel de células escamosas (SCC) es comúnmente un parche rojo, escamoso y engrosado en la piel expuesta al sol. Algunos son nódulos firmes y duros y cúpula con forma de queratoacantomas. Es peligroso, pero no tan peligroso como un melanoma.

[4] Jou PC, Feldman RJ, Tomecki KJ (June 2012). "UV protection and sunscreens: what to tell patients". Cleveland Clinic Journal of Medicine.
[5] World Cancer Report 2014. World Health Organization. 2014. pp. Chapter 5.14. ISBN 978-9283204299.

Melanoma

La mayoría de los melanomas presentan distintos colores, desde tonos de marrón a negro. Una pequeña cantidad de melanomas son de color rosa, rojo o carnoso, estos se denominan melanoma amelanótico y tienden a ser más agresivos. Las señales de advertencia de melanoma maligno incluyen cambios en el tamaño, forma, color o elevación de un lunar. Otros signos son la aparición de un nuevo lunar durante la edad adulta o dolor, picazón, ulceración, enrojecimiento alrededor del sitio o sangrado en el sitio.

Otros

Los carcinomas de células de Merkel a menudo son protuberancias de color rojo, morado o de la piel que no crecen rápidamente, que no son dolorosas y que no duelen ni pican. Pueden confundirse con un quiste u otro tipo de cáncer.

La base de imágenes del ISIC presenta casos de todos los tipos de cancer de piel extraidos a traves del dermatoscopio. La dermatoscopia es una técnica de imagen que elimina el reflejo superficial de la piel, al eliminarlo se mejora la visualización de los niveles más profundos de la piel.

A continuación se muestran diferentes ejemplos de cada diagnositico extraidos de la base de datos.

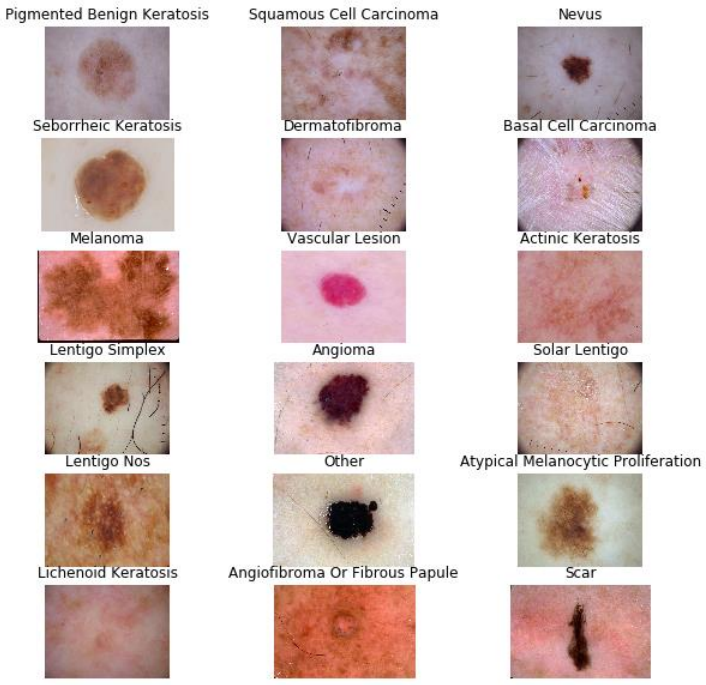



*Ilustración 2. Diagnosis cáncer piel*

*Tabla 3. Tamaño muestral por diagnóstico*

| Diagnosis | Número | % |
|---|---|---|
| Nevus | 9315 | 63,05 |
| Melanoma | 2169 | 14,68 |
| Pigmented benign keratosis | 1099 | 7,44 |
| Basal cell carcinoma | 586 | 3,97 |
| Seborrheic keratosis | 419 | 2,84 |
| Unknown | 249 | 1,69 |
| Squamous cell carcinoma | 226 | 1,53 |
| Vascular lesion | 142 | 0,96 |
| Atypical melanocytic proliferation | 133 | 0,90 |
| Actinic keratosis | 132 | 0,89 |
| Dermatofibroma | 122 | 0,83 |
| Lentigo NOS | 71 | 0,48 |
| Solar lentigo | 57 | 0,39 |
| Lentigo simplex | 27 | 0,18 |
| Angioma | 15 | 0,10 |
| Other | 10 | 0,07 |
| Angiofibroma or fibrous papule | 1 | 0,01 |
| Lichenoid keratosis | 1 | 0,01 |
| Scar | 1 | 0,01 |

Como podemos observar, la mayoría de las imágenes son nevus, comúnmente conocidos como lunares, en esta clase están recogidos los lunares no malignos. La clase melanoma hace referencia a los tumores melánicos o pigmentados, tal como se ha comentado anteriormente, estos son los que mayor mortalidad presentan. Las demás clases pertenecen a NMSC.

# 3 Deep Learning. Análisis y generación de imágenes

El Deep Learning para la visión artificial o también conocido como “computer visión”, se fundamenta en dos pilares: las redes neuronales convolucionales y el algoritmo de backpropagation que aunque ya se conocían en 1989 no tomaron importancia hasta 2012 por tres motivos:

- Hardware
- Disponibilidad de bases de datos y puntos de referencia
- Avances en algoritmia

El Deep Learning se basa en los avances prácticos más que por la teoría, por ese motivo, aunque los algoritmos ya estaban inventados, el uso y la popularidad del Deep Learning se estancó entre 1990s y 2000s por culpa de la falta de datos disponibles y la falta de avances en hardware.

Entre 1990 y 2010, la comercialización de CPUs creció de forma exponencial, como resultado, hoy en día es posible ejecutar modelos pequeños de Deep Learning en cualquier ordenador, esto sería impensable hace 25 años.

En cuanto al hardware, compañías como Nvidia y AMD han hecho posible el desarrollo de modelos de Deep Learning que antes eran irrealizables por su alto coste computacional gracias a las GPUs. Estas empresas que se dedican al hardware del mundo de los videojuegos, mientras invertían su capital y esfuerzo en desarrollar redes de procesadores en paralelo o unidades de procesado grafico (GPUs) para hacer frente a los gráficos de los videojuegos y a la serie de cálculos necesarios, hicieron posible la ejecución de modelos de Deep Learning de forma implícita.

Los videojuegos necesitan unos procesadores capaces de renderizar las imágenes y hacer cálculos de forma inmediata para que la experiencia del jugador sea lo mejor posible, de la misma forma el Deep Learning requiere hacer una serie de transformaciones y operaciones matriciales rápidas. Así pues dos sectores aparentemente inconexos crearon una simbiosis que hoy en día no deja de crecer.

En cuanto a algoritmos, en 2014, 2015 y 2016 se desarrollaron métodos para ayudar al cálculo del gradiente del algoritmo de backpropagation, como por ejemplo el batch normalization.

Hoy en día, Google ha presentado TPU (tensor processing unit) una serie de chips desarrollados específicamente para ejecutar modelos de redes neuronales profundas, haciendo que el tiempo de ejecución sea 10 veces más rápido y más eficiente que las GPUs.

Por esta serie de motivos y gracias a los resultados satisfactorios que presentan, hoy en día las redes neuronales son una herramienta fundamental en muchos ámbitos.

## 3.1 Redes Neuronales Convolucionales (CNNs)

Las redes neuronales convolucionales como su nombre indica se fundamentan en la convolución. La convolución es el termino matemático que define la operación de integrar el producto de dos funciones al ser desplazadas una sobre la otra (y una de ellas volteada), este caso hablamos de convolución discreta. Así pues, en las CNNs mediante la combinación de una matriz/vector de entrada (tensor[6]) y otra que se desplaza sobre ella (filtro o kernel), aplicando una serie de operaciones en cada una de las posiciones posibles, se obtiene una nueva matriz, este procedimiento se realiza en cada uno de los canales de color de la imagen.

### 3.1.1 Convoluciones

Sean dos funciones $f$ y $g$ se define la convolución discreta de $f$ y $g$ para el conjunto de valores enteros como:

$$(f * g)[n] = \sum_{x=-\infty}^{\infty} f(x)g(n-x) \tag{1}$$

Si el input es bidimensional, como es el caso de las CNNs, la convolución se traduce a:

$$(f * g)[n,m] = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} f(x,y)g(n-x,m-y) \tag{2}$$

De forma que la convolución discreta aplicada en el caso de las CNNs donde el input es un tensor 3D, para cada canal se calcula:

$$feature\ map\ = \sum_{y=0}^{m} \left( \sum_{x=0}^{n} input(x-a, y-b) kernel(x,y) \right) \tag{3}$$

Para cada elemento del input o pixel (a,b) se calcula la formula anterior donde la función $f$ definida en la formula (2) equivale al kernel y la función $g$ (2) es la matriz de entrada o input.

[6] Generalización de vectores y matrices en un número de dimensiones.

Para cada canal, el *kernel* de dimensión $n\ x\ m$ se desplaza sobre la matriz de entrada, de manera que en cada una de las posiciones se calcula el producto escalar entre la matriz de entrada y el kernel, así pues se obtiene cada uno de los elementos de la matriz de salida, también llamada *feature map*. Finalmente el resultado es un tensor 3D definido por cada uno de los feature maps de cada canal.

La dimensión del *feature map* dependerá de tres variables: el *stride (*saltos del kernel sobre la matriz, por ejemplo, de uno a uno, de dos en dos, etc.) y como se desplaza cuando alcanza los bordes de la matriz de entrada (*padding*) y el número de canales del input.

Como norma general la dimensión del feature map resultante será:

$$\begin{gathered}\text{Input} = (n, n, n_c) \\ \text{Kernel} = (f, f, n_c) \\ \text{Output} = \left(\left(\frac{n+2p-f}{s}+1\right), \left(\frac{n+2p-f}{s}+1\right), n_c\right)\end{gathered} \tag{4}$$

Donde $p$ se refiere a la dimensión del padding, $s$ al stride, $n$ al número de pixeles de largo o ancho, $n_c$ hace referencia al número de canales y $f$ es la dimensión del kernel.

De la forma en que está definida la convolución y según el número de stride escogido, los pixeles o elementos del input que están ubicados en los extremos exteriores participan menos en el cálculo de la convolución y por lo tanto se puede perder información. Para evitar este problema y además evitar la modificación de la ratio de aspecto del input o la reducción de la dimensión del output, se utiliza el padding que consiste en añadir ceros en el contorno de la matriz de entrada.

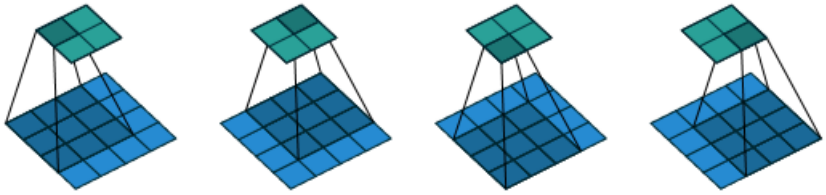


*Ilustración 3. Padding cero, strides 1x1*

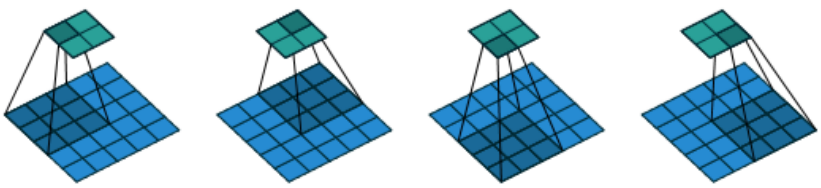


*Ilustración 4. Padding cero, strides 2x2*

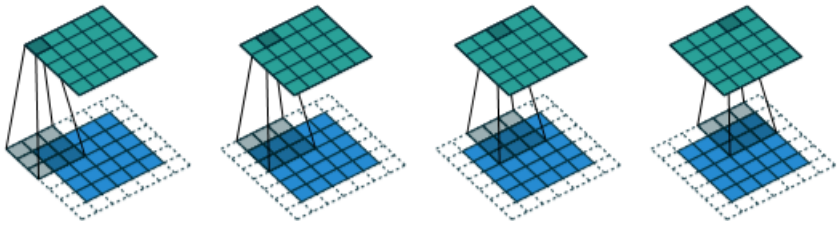


*Ilustración 5. Padding 1, strides 1x1*

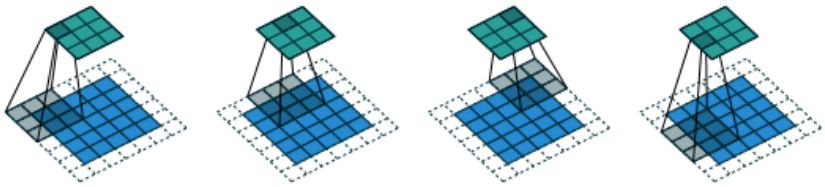


*Ilustración 6. Padding 1, strides 2x2*

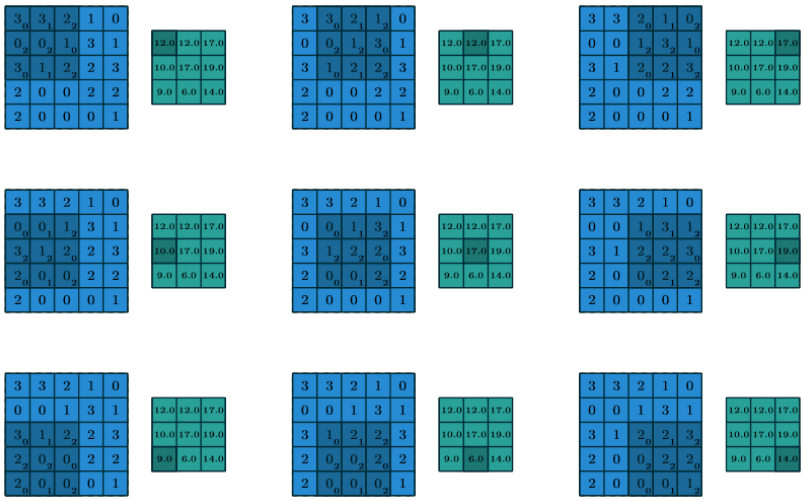


*Ilustración 7. Ejemplo convolución discreta*

La Ilustración 7 muestra en color azul claro el input o tensor, donde se le aplica un kernel con los siguientes valores:

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 2 | 2 | 0 |
| 0 | 1 | 2 |

Para cada zona oscurecida, se calcula el producto entre cada elemento del kernel y los del input y se suma de forma que se obtiene el resultado del elemento del output.

El *kernel* se puede interpretar como un filtro que mediante un entrenamiento va modificando sus pesos para detectar distintas características de la imagen.

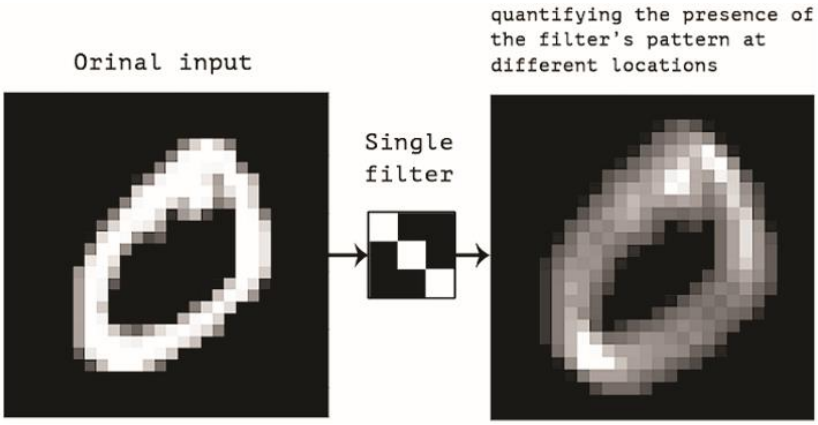



*Ilustración 8. Ejemplo feature map de un input con un canal de color*

Las redes neuronales convolucionales suelen presentar múltiples capas convolucionales, en cada una de ellas se define un kernel o filtro.

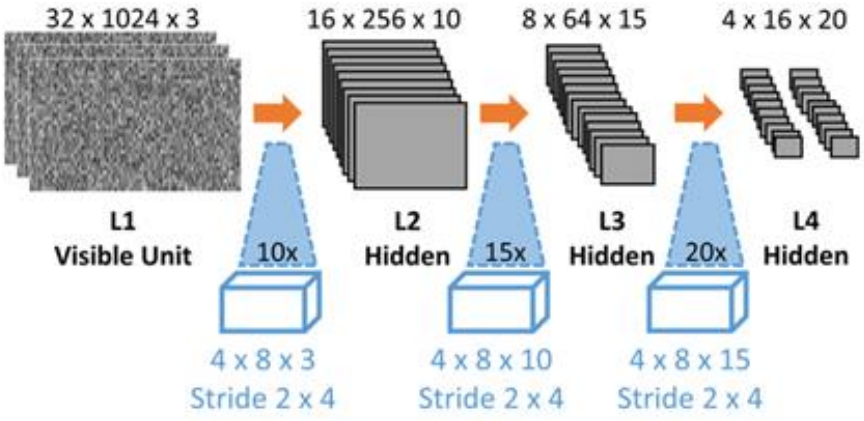



*Ilustración 9. Ejemplo CNNs con tres capas convolucionales*

En el caso de las CNNs los valores del *kernel* se inicializan al azar, normalmente ruido blanco, y mediante el entrenamiento estos pesos van cambiando hasta que obtienen su combinación óptima.

### 3.1.2 Convoluciones traspuestas

Las convoluciones traspuestas son operaciones que permiten hacer el proceso opuesto a la convolución discreta. Sea la matriz resultante de aplicar la convolución si se aplica la convolución traspuesta sobre ella, se obtiene la matriz inicial donde se aplicó la convolución. En cuanto a los pesos, estos se conectan manteniendo los mismos patrones que en la convolución.

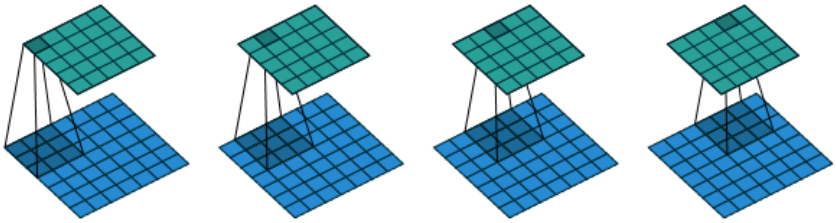


*Ilustración 10. Convolución traspuesta*

En la Ilustración 10. se representa la convolución traspuesta de una matriz de input (verde) 5x5 con un kernel 3x3 con padding =2 y strides unitarias. Es equivalente a hacer una convolución con un kernel 3x3 sobre un input (azul) 7x7 usando strides unitarios y sin padding.

Cabe destacar que no es lo mismo una deconvolución que una convolución traspuesta, en el primer caso, se obtiene de nuevo los valores existentes antes de aplicar la convolución, por lo tanto, es la función inversa; mientras que con las otra se obtiene una matriz de salida igual a la original pero no se obtienen los mismos valores.

Las convoluciones traspuestas son fundamentales en las DCGANs a partir de un vector n-dimensional se aplican convoluciones traspuestas hasta conseguir una matriz de dimensión igual al tamaño de la imagen deseada.

### 3.1.3 Optimización y estimación de los parámetros en las CNNs

Las CNNs son redes neuronales, pero con la peculiaridad de las convoluciones.

Podemos esquematizar una red neuronal con una capa oculta de la siguiente manera:

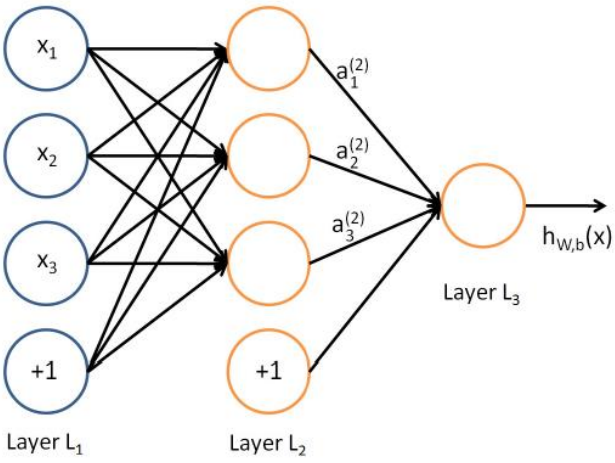



*Ilustración 11. Ejemplo red neuronal*

La capa L1 es el input o tensor 1D (vector), en el caso de las CNNs en vez de ser un vector es un array, como se puede observar en lo único que se diferencia una CNNs de una red neuronal feedforward es el filtro (kernel) y la forma como se conectan las capas.

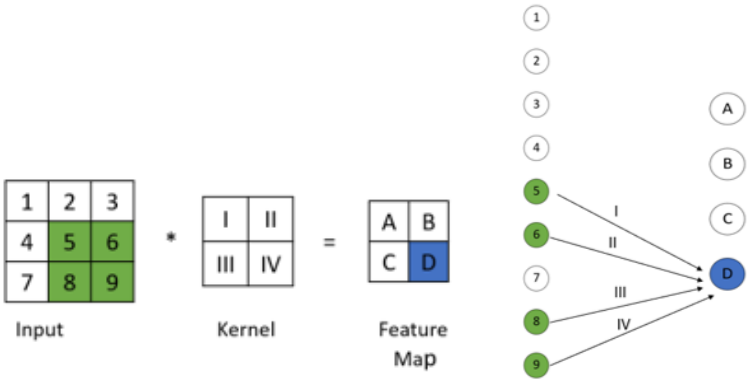



*Ilustración 12. Ejemplo CNNs*

Como se puede observar en la Ilustración 12. Ejemplo CNNs las redes convolucionales son idénticas a las redes neuronales feedforward pero se diferencian en las conexiones.

El problema de optimización en las redes neuronales es encontrar los pesos (I, II, III, IV) tal que minimicen una función de coste.

Por ejemplo, siguiendo el caso de la Ilustración 11:

Sea $\theta_{ij}^{(l)}$ el peso del input/nodo $j$ al nodo $i$ en la capa $l$, $z^{(l+1)}$ el resultado de multiplicar los valores de la capa $l$ con los pesos $\theta_{i,j}^{l}$ y $a^{l+1}$ los valores que toma la capa $l$ después de haber aplicado la función de activación $f$ sobre $z^{(l+1)}$.

- Los valores de la capa $L_2$ vienen definidos de la siguiente manera $z^{(2)} = \Theta^{(1)}x$ o de forma equivalente:

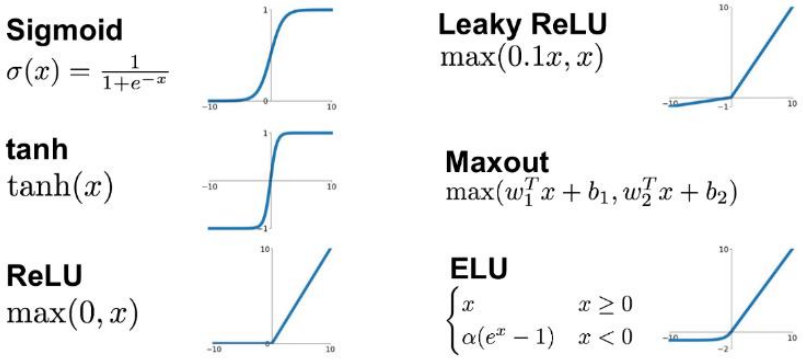
$$\begin{aligned} z_1^{(2)} &= \theta_{10}^{(1)} + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3 \\ z_2^{(2)} &= \theta_{20}^{(1)} + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3 \\ z_3^{(2)} &= \theta_{30}^{(1)} + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3 \end{aligned} \quad (5)$$

  Donde $x_j$ es el input, $\Theta^{(1)}$ es la expresión matricial de los pesos que tiene dimensión 3x4

- Sobre los valores de L2 se les aplica una función de activación $f$.

$$a^{(2)} = f(z^{(2)}) \quad (6)$$

La función de activación varía según las características del problema, por ejemplo la sigmoide y tanh se utilizan cuando el output es binario mientras que las demás cuando el output es numérico.




* En este caso de estudio se utilizará la función sigmoide en la capa de output del clasificador, para las capas intermedias ReLu ya que su salida son matrices numéricas no binarias. Para el caso específico de las redes generativas adversarias se utiliza la función tanh en la capa de output del generador y en las demás ReLu, para el discriminador se utiliza Leaky Relu.

- En la capa L3 se obtienen los valores de forma análoga a L2 pero con otros pesos ($\Theta^{(2)} \in \mathbb{R}_{1x4}$).

$$z^{(3)} = \Theta^{(2)} a^{(2)} \tag{7}$$

- Finalmente, se construye la función $h_\Theta(x)$ que depende de las thetas definidas previamente.

$$h_\Theta(x) = a^{(3)} = f(z^{(3)}) \tag{8}$$

De forma general:

$$z^{(l+1)} = \begin{bmatrix} z_1^{(l+1)} \\ z_2^{(l+1)} \\ \vdots \\ z_{s_{l+1}}^{(l+1)} \end{bmatrix} = \begin{bmatrix} \theta_{10}^{(l)} & \cdots & \theta_{1s_l}^{(l)} \\ \vdots & \ddots & \vdots \\ \theta_{s_{l+1}0}^{(l)} & \cdots & \theta_{s_{l+1}s_l}^{(l)} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ a_1^{(l)} \\ \vdots \\ a_{s_l}^{(l)} \end{bmatrix} \tag{9}$$

$$a^{(l+1)} = \begin{bmatrix} a_1^{(l+1)} \\ a_2^{(l+1)} \\ \vdots \\ a_{s_{l+1}}^{(l+1)} \end{bmatrix} = f\left(z^{(l+1)}\right) = \begin{bmatrix} f\left(z_1{}^{(l+1)}\right) \\ f\left(z_1{}^{(l+1)}\right) \\ \vdots \\ f\left(z_{s_{l+1}}^{(l)}\right) \end{bmatrix} \tag{10}$$

Así pues, el objetivo será minimizar la función de perdida de $h_\Theta(x)$ que depende de todo el conjunto de pesos de la red neuronal. Esta minimización se realiza mediante el descenso del gradiente y el algoritmo de backpropagation.

A continuación, se muestra la formulación para el ejemplo de la ilustración 10 con respuesta binaria.

El problema de optimización consiste en encontrar los pesos $\Theta$ tal que minimicen la función de perdida.

La función para minimizar es $l(h_\theta(x), y)$ donde $h_\theta(x)$ se define de la siguiente forma:

$$\begin{gathered} h_\Theta(x) = a^{(3)} \\ a^{(3)} = f(z^{(3)}) = f(\Theta^{(2)} a^{(2)}) \\ a^{(2)} = f(z^{(2)}) = f(\Theta^{(1)} x) \\ h_\Theta(x) = f(\Theta^{(2)} f(\Theta^{(1)} x)) \end{gathered} \tag{11}$$

Se utiliza la función de perdida por partes:

$$\begin{aligned} l(h_\theta(x), y) &= -log h_\theta(x) && si\ y = 1 \\ l(h_\theta(x), y) &= -log(1 - h_\theta(x)) && si\ y = 0 \end{aligned} \tag{12}$$

O de forma equivalente,

$$l(h_\theta(x), y) = -y log h_\theta(x) - (1 - y) log(1 - h_\theta(x)) \tag{13}$$

Por lo tanto, la función de coste a minimizar:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} log h_\theta\left(x^{(i)}\right) + (1 - y^{(i)}) log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] \tag{14}$$

El objetivo será encontrar los valores de $\theta$ tal que minimicen esta función. esto se resuelve gracias al *gradient descent* que permite encontrar la dirección de paso para minimizar la función.

En cada iteración del algoritmo los valores de $\theta$ se actualizan de la siguiente forma:

$$\Theta^{n+1} = \Theta^n - \eta \nabla J(\theta) \tag{15}$$

Para cada elemento:

$$\theta_j^{n+1} = \theta_j^n - \frac{\partial}{\partial \theta_j} J(\theta) \tag{16}$$

Donde $\eta$ es la longitud de paso o learning rate.

Habitualmente, en las redes neuronales no se utiliza el gradient descent debido a su gran coste computacional, en vez de este se utiliza el gradiente estocástico.

El gradiente estocástico en vez de calcular la derivada para cada uno de los puntos se calcula sobre el punto i escogido al azar:

$$\theta_j^{n+1} = \theta_j^n - \frac{\partial}{\partial \theta_j} J(\theta; x_i) \quad (17)$$

Tal como se ha comentado anteriormente, la única diferencia que presentan las redes neuronales de clasificación binaria respecto la regresión logística es que depende de los pesos definidos en las capas anteriores, por lo tanto es necesario no solamente optimizar los pesos de la última capa si no todos los anteriores.

En las redes neuronales se utiliza el algoritmo de *backpropagation*[7] para calcular los diferentes gradientes de los pesos $\theta$ de cada una de las capas.

El algoritmo de backpropagation consiste en calcular el error total de la red neuronal y analizar cómo cambia el error cuando se cambia el valor de uno de los elementos. La magnitud de cambio se calcula diferenciando la función de coste respecto cada elemento de la red neuronal, así pues se obtiene una estimación de la contribución de cada elemento a la función de coste total.

Así pues, todas las redes neuronales se basan en este principio, las únicas variaciones son la arquitectura de ellas.

### 3.1.4 Arquitectura de las CNNs

Normalmente, las redes neuronales convolucionales para el tratamiento de imágenes tienen como input un array o tensor 3D (channel, width, height), donde channel es el número de canales de color siendo 1 escala de grises o 3 color (también se puede trabajar sobre un canal específico: rojo, azul o verde), width es la anchura en pixeles y height la altura. En este caso, se aplica el kernel sobre cada uno de los canales de la imagen, extrayendo en cada capa un feature *map* 3D, donde cambiara el número de canales, la anchura y la altura según la arquitectura.

Las redes neuronales convolucionales presentan tres tipos de capas: las convoluciones, pooling y fully-connected.

[7] Learning representations by back-propagation errors. David E. Rumelhart (1986)

Las capas de convolución, tal como se ha comentado anteriormente, presentan la característica que a partir de un kernel que recorre el array de entrada se obtiene un array de salida, en consecuencia, las redes neuronales presentan al principio este tipo de capa.

Otro tipo de capa es el pooling, con este tipo de capa se transforma el tensor de entrada a uno con diferente dimensión, pero no se estiman pesos, si no se utiliza un kernel y transformaciones matemáticas como el máximo o la media para obtenerla.

Finalmente las capas fully-connected, después de la serie de transformaciones necesarias se obtiene un tensor de dimensión $n\ x\ m$ y k canales los cuales finalmente se transforman en un vector de dimensión $n\ x\ m\ x\ k$, este procedimiento se le llama aplanado o flatten. A partir de esta capa se hace una arquitectura igual a las redes neuronales simples.

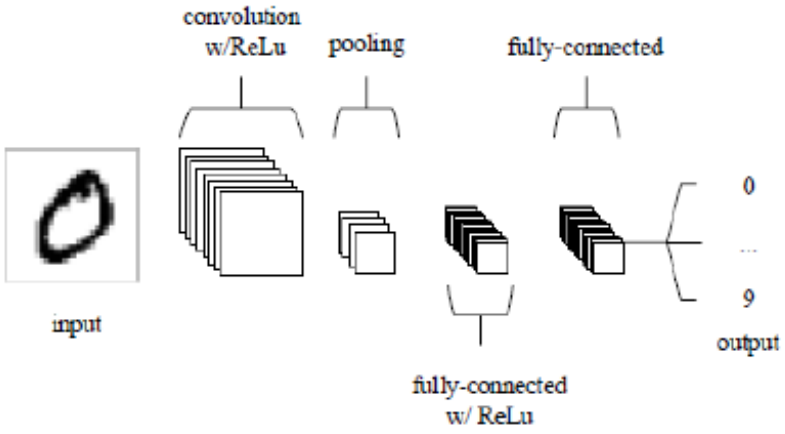



*Ilustración 13. Ejemplo Arquitectura CNN*

### 3.1.5 Características CNNs

En las redes neuronales convolucionales se destaca que las capas densas detectan patrones globales en el espacio del input, mientras que las capas de convolución detectan patrones locales, tal como se puede observar en la Ilustración 14.

Los patrones que aprenden las CNNs son invariantes a traslaciones, es decir, si detectan un patrón en la esquina de la imagen, este podrá ser reconocido en cualquier parte de la imagen presente o futura. Además, pueden detectar patrones de jerarquía espacial, debido a su arquitectura, la primera capa convolucional detectara patrones locales, la siguiente detectara patrones a partir del *feature map* que se obtiene de la anterior, así sucesivamente, de forma que se construye una jerarquía.

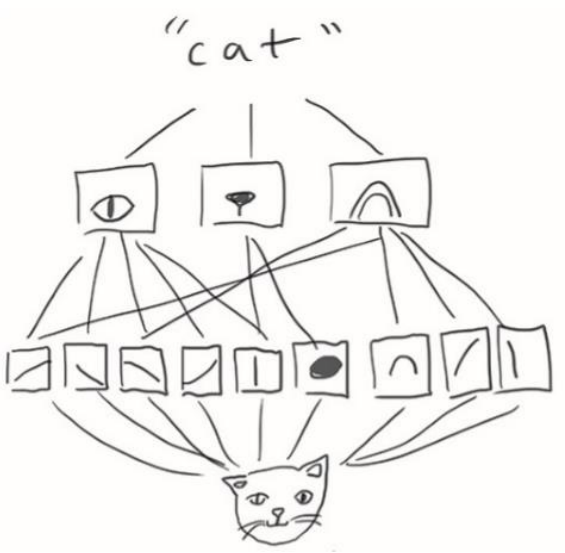



*Ilustración 14. Patrones jerarquía espacial. CNNs*

Además, las CNNs a diferencia de las redes neuronales, permiten observar la *caja negra*, es decir, se puede analizar la relación entre el input y el output, en otras palabras como la red toma las decisiones.

Por ejemplo, se puede visualizar en cada capa cada uno de los feature maps resultantes, así pues se puede analizar en cada uno de ellos que elementos la red destaca y observar como de forma secuencial la red va destacando los pixeles significativos para determinar la respuesta.

Además, con las CNNs es posible visualizar la importancia de cada píxel a la hora de predecir la respuesta, es decir como de importante ha sido cada uno de los pixeles en el momento de realizar la predicción. Un ejemplo de ello es la Ilustración 15, donde se destaca las zonas que han permitido a la red neuronal identificar el tipo de elefante que aparece en la imagen.

*Ilustración 15. Heatmap de activación de clase*

## 3.2 Redes generativas antagónicas (GANs)

Las redes generativas antagónicas o también conocidas como GANs son un tipo de red neuronal no supervisada propuesta por Ian J. Goodfellow en 2014 capaz de generar nuevos datos a partir de estimar la distribución subyacente de los datos con los que se entrena.

La lógica es sencilla, mediante dos agentes, un generador y un discriminador, el primero intenta engañar al segundo, así pues, durante el entrenamiento, este consigue ser capaz de imitar las características de los datos reales, siendo el objetivo obtener muestras creíbles en vez de minimizar un error. En consecuencia, el error nos indicará la dirección de mejora para ambos agentes pero solo será una medida de performance de la GANs.

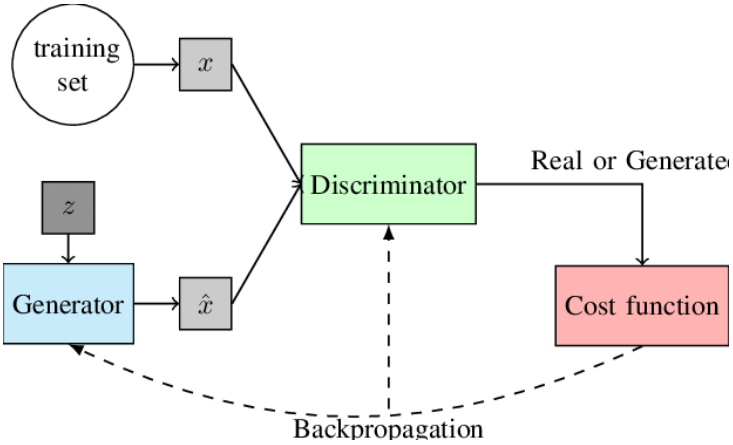



*Ilustración 16. Diagrama GANs*

El entrenamiento es un proceso antagónico en el que se entrena simultáneamente un modelo generativo que intenta representar la distribución de los datos y un modelo discriminativo que estima la probabilidad de que la muestra generada prevenga de la distribución real y no del generador.

El generador tiene como input un vector aleatorio o un punto aleatorio en un espacio latente, este lo transforma en los datos sintéticos.

El discriminador, toma como input los datos tanto reales como ficticios y debe predecir si provienen del generador o no, además sirve como función de pérdida del generador pero con la diferencia que en cada iteración esta función varía.

En las GANs la función de pérdida del discriminador o del generador no solo depende de los propios parámetros sino que también de los parámetros del otro agente.

Por ejemplo, el discriminador tratará de minimizar $\mathcal{L}_{disc}(\theta_G, \theta_D)$ pero solo puede hacerlo manipulando sus propios pesos $\theta_D$, de forma análoga lo hará el generador.

La solución será un equilibrio de Nash $(\theta_G, \theta_D)$ que es un mínimo local de $\mathcal{L}_{disc}(\theta_G, \theta_D)$ dado $\theta_D$ y mínimo local de $\mathcal{L}_{disc}(\theta_G, \theta_D)$ dado $\theta_G$.

$$\mathcal{L}_{disc} = -\frac{1}{2N}\sum_{i}^{N}\log\big(D(y_i)\big) - \frac{1}{2N}\sum_{i}^{N}\log\big(1 - D(\hat{y}_i)\big) \tag{18}$$

Con esta formulación el discriminador tiende a dar una probabilidad cercana a 1 cuando las muestras son reales mientras que da 0 cuando son muestras del generador.

Desde la perspectiva estadística,

$$\mathcal{L}_{disc} = -\frac{1}{2}\mathbb{E}_{y\sim P_{data}}\big[log\big(D(y)\big)\big] - \frac{1}{2}\mathbb{E}_{z\sim P_z}\Big[log\Big(1 - D\big(G(z)\big)\Big)\Big] \tag{19}$$

Donde $G(z)$ son los datos generados por el generador y $D(y)$ son los datos reales.

De forma equivalente la función de pérdida del generador es $\mathcal{L}_{gen} = -\mathcal{L}_{disc}$.

Así pues mientras el discriminador intenta minimizar $\mathcal{L}_{disc}$, el generador intenta maximizarla, en consecuencia es un juego de suma cero.

Las GANs son un sistema donde a diferencia de los otros modelos de Deep Learning no tienen una función a minimizar, realmente el objetivo de la GAN no es conseguir un mínimo si no un equilibrio entre las dos fuerzas.

Definiendo $V(D, G) = -2\mathcal{L}_{disc}$, se puede reformular como un juego minimax de dos jugadores en el que el discriminador quiere maximizar la probabilidad de distinguir si los datos son reales o no y el objetivo del generador es minimizarla. La formulación es la siguiente:

$$\min_G \max_D V(D, G) = \min_{\theta_G} \max_{\theta_D}\Big(\mathbb{E}_{y\sim P_{data}}[log\ D(y)] + \mathbb{E}_{z\sim P_z}\Big[log\Big(1 - D\big(G(z)\big)\Big)\Big]\Big) \tag{20}$$

La solución posible a este problema es que el generador G sea capaz de reproducir exactamente los datos, es decir que su error sea 0 y que la probabilidad de que el discriminador D identifique de donde provienen los datos sea ½, es decir que no sea capaz.

Esta formulación presenta un problema: mientras que la función de coste del clasificador presenta un gradiente no nulo cuando el discriminador se equivoca, cuando acierta la función de pérdida se satura y en consecuencia el generador se queda sin gradiente.

Por lo tanto es necesario evitar que el discriminador se vuelva muy bueno ya que si esto sucede el generador no podrá remontar en el juego minimax. Para solucionar este problema se cambia algunas de las etiquetas (real o generado) de los datos de entrenamiento del discriminador, así pues las muestras generadas obtienen la categoría real y viceversa consiguiéndose que las dos redes tengan un gradiente bien definido.

Como se ha podido observar, pese ser una idea simple llegar a la solución óptima suele ser complicado en las GANs ya que tienden a presentar distintos problemas como por ejemplo el *colapso, la no-convergencia y la desaparición del gradiente.*

El colapso en una GAN se refiere a que el generador no es capaz de generar datos con la misma variabilidad que los originales y en consecuencia solo genera unas cuantos de un tipo. La no convergencia se debe a que cuando uno de los dos agentes toma ventaja respecto el otro uno se queda atrás y las oscilaciones afectan a la calidad de los datos generados, así pues, esto se puede resolver utilizando ruido aleatorio en el generador y usando dropout en el discriminador, tal como se explica más adelante.

Finalmente, *sparse gradient*, se debe a que el gradiente después de mucho entrenamiento tiende a tomar valores muy pequeños, por lo tanto, el crecimiento es lento y se suele estancar la mejora.

## 3.3 Redes Generativas Antagónicas Convolucionales Profundas (DCGANs)

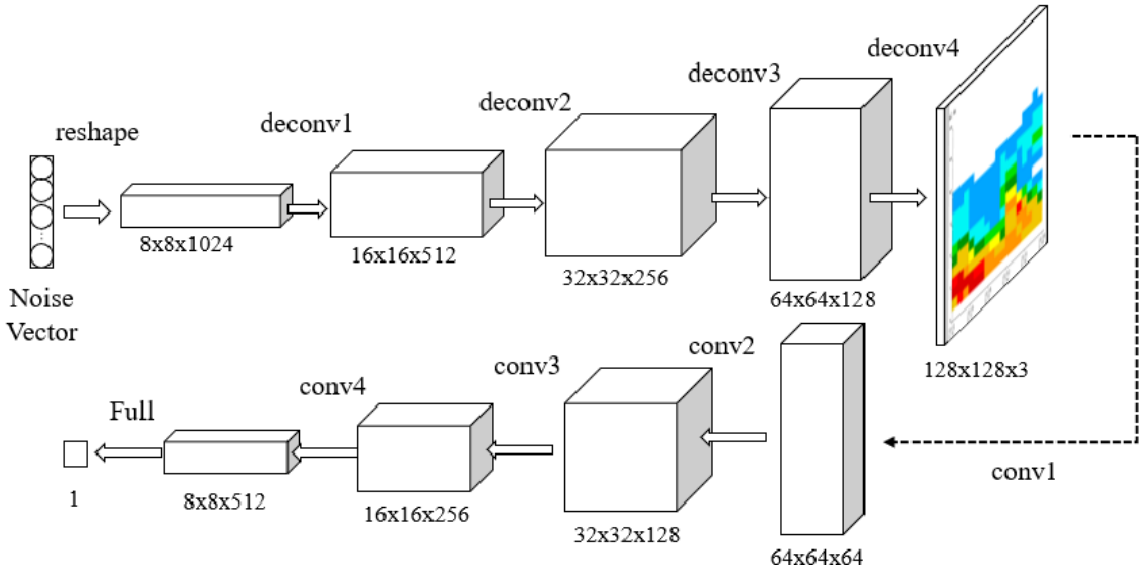



*Ilustración 17. Arquitectura DCGAN*

Tras el descubrimiento de las GANs se desarrollaron múltiples arquitecturas y tipos de red basadas en las GANs, una de ellas son las Deep Convolutional GANs o DCGANs.

Este tipo de red neuronal se desarrolló en 2015 por Radford et al. que combina el entrenamiento antagónico con las redes convolucionales que mejoraba las arquitecturas que había hasta la fecha para hacer frente a las problemáticas de reconocimiento de imagen o *computer visión.*

Como se puede observar en la Ilustración 17. Arquitectura DCGAN, las DCGANs parten de un vector aleatorio n-dimensional, es decir un punto aleatorio en el espacio euclídeo de dimensión n que definirá el espacio latente vectorial al que pertenecen las imágenes una vez el generador este entrenado, esta es una de las hipótesis del modelo. A partir del vector de ruido se reconstruye en un formato 2D tensor, característico de las imágenes (width, height, channels), así de forma sucesiva se van haciendo deconvoluciones hasta conseguir, en la última capa, un tensor con la dimensión de las imágenes de muestra.

Esta imagen generada por el generador se le pasa al discriminador que presenta las características típicas de una CNN ya que el input es una imagen, finalmente el discriminador determina si la imagen es real o ficticia del mismo modo que las GANs.

Las características en arquitectura que presentan las DCGANs son las siguientes:

- El generador se construye a partir de un vector n-dimensional aleatorio que mediante deconvoluciones transforma el tensor 1D a un tensor 3D hasta conseguir en la última capa un tensor 3D con las mismas características que las imágenes de muestra.
- El discriminador es una red neuronal convolucional que tiene como input las imágenes reales y generadas, para evitar el colapso se intercambian las etiquetas de las imágenes.
- Uso del Batch normalization estabiliza el aprendizaje ya que normalizando el input en cada unidad ayuda al gradiente y así pues evita el colapso del generador. No se utiliza Batch Normalization en la última capa del generador ni en la capa de input del discriminador.
- Se utiliza la función de activación Relu en las capas del generador excepto en el output que se utiliza Tanh, gracias a esto la red aprende, satura y cube el mapa de color con la distribución de muestra más rápido.
- Los puntos del espacio latente se extraen de una distribución normal no de una uniforme.
- Se substituye el pooling por convoluciones con stride, así pues la red aprende por si misma a hacer el upsampling en el caso del generador o downsamling en el discriminador.

- Se añade dropout en el discriminador y se añade ruido en las etiquetas del discriminador para favorecer la convergencia del modelo.
- El uso de max pooling y de funciones de activación ReLU tienden a generar *gradient sparsity*, para evitarlo se utiliza capas convolucionales con stride para hacer el downsampling y utilizar leaky ReLU como activación, esta última a diferencia de la normal, permite un pequeño número de valores negativos.

Las DCGANs además de permitir generar imágenes realistas también permiten realizar operaciones aritméticas como por ejemplo pasar de una imagen generada a otra de forma sucesiva como se puede observar en la Ilustración 18.

*Ilustración 18. Interpolación lineal entre dos imágenes generadas*

Por ejemplo, sean dos puntos aleatorios en el espacio latente obteniéndose dos imágenes, se puede generar una interpolación lineal o uniforme entre esos dos puntos en el espacio latente.

Por otro lado, también se pueden realizar operaciones vectoriales con los puntos del espacio latente. Por ejemplo, si se extraen 100 puntos del espacio latente y se analizan las características de las imágenes generadas, escogiendo la característica deseada en una de ellas se puede utilizar su vector para manipular.

Por ejemplo, siguiendo con el caso de las caras generadas, si encontramos un hombre con gafas, un hombre sin gafas y una mujer sin gafas, haciendo las convenientes operaciones se puede obtener una mujer con gafas como se muestra en la Ilustración 19.

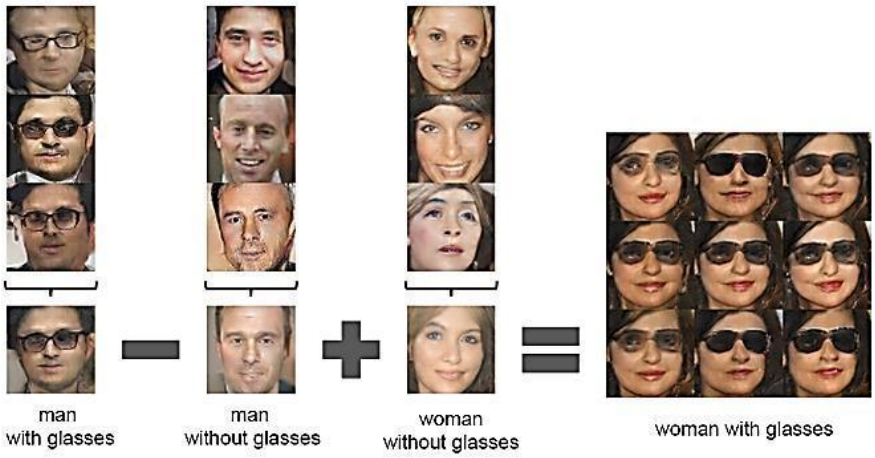



*Ilustración 19. Operaciones DCGANs*

En conclusión, las GANs permiten moverse por el espacio latente y gracias a ello se puede avanzar por diferentes direcciones y visualizar los cambios que se producen en la imagen según la dirección que se toma. Además, las DCGANs permiten realizar operaciones aritméticas con los vectores conceptuales, por ejemplo, sumar dos imágenes a partir de sus vectores permite que los conceptos presentes en las imágenes se sumen de manera que la imagen resultante presente las características esperadas.

# 4 Aplicación del caso de estudio y resultados

Las imágenes fueron extraídas del archivo del ISIC con un tamaño total de 24 GB, son imágenes extraídas mediante dermatoscopio.

Tal y como se comentó en el apartado 1.3, las imágenes provienen de diferentes fuentes de datos en consecuencia presentan diferentes tamaños de ancho y largo, a continuación, se explica la serie de transformaciones sobre las imágenes necesarias para entrenar la red neuronal.

El clasificador se ha realizado mediante una submuestra balanceada entre fotos benignas y malignas, así pues se ha utilizado toda la muestra de imágenes malignas y se ha extraído un conjunto aleatorio de la misma dimensión de imágenes benignas, el número total de imágenes es 4.394.

Para la selección de los hiperparámetros, como por ejemplo el número de epoch o la arquitectura de la red se ha dividido la muestra en tres partes: train, validación y test, repartidas en un 70%, 20% y 10% respectivamente. Así pues, la muestra de prueba no se hará servir en ningún momento para elegir hiperparámetros tan solo para determinar la precisión final, mientras que se utilizará la muestra de validación para determinar las configuraciones que presentan mejores resultados.

Por otro lado, para el entrenamiento de la red generativa antagónica se ha utilizado toda la muestra disponible ya que el objetivo es generar imágenes realistas de piel humana, cabe destacar que para este tipo de modelo no se realiza ninguna partición de la muestra.

## 4.1 Procesamiento de la imagen

Las imágenes utilizadas son a color (3 canales RGB) con formato .jpg y presentan diferentes dimensiones de ancho y alto.

*Tabla 4. Distribución ancho-largo de las imágenes*

| | Mínimo | Máximo |
|---|---|---|
| Width | 576px | 6780px |
| Height | 450px | 4519px |

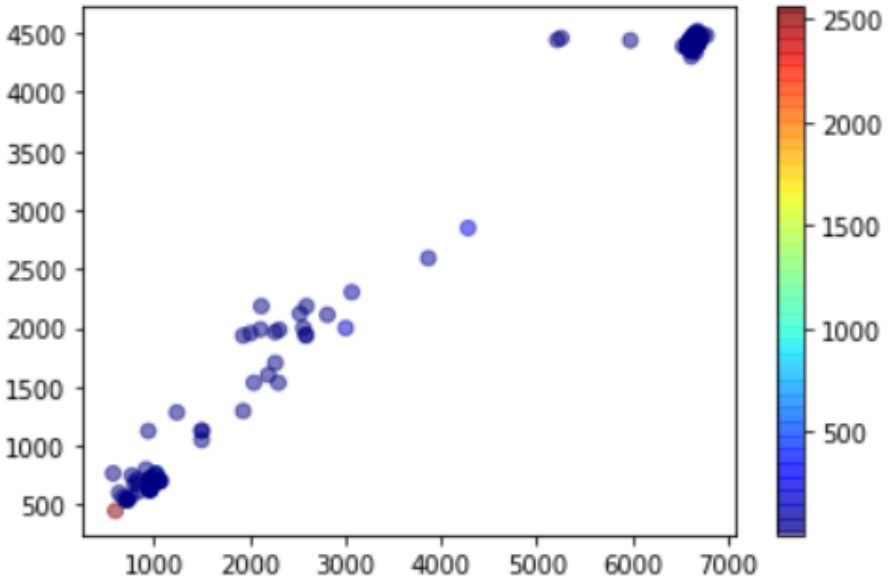



*Ilustración 20. Distribución de la dimensión de pixeles*

La mayoría de las imágenes, aproximadamente el 60% son de 600x400px.

Para la correcta utilización de las imágenes ha sido necesario hacer en determinados casos un "downsampling" para homogenizar la dimensión de las imágenes a 256x256px. Esta metodología consiste en hacer una extrapolación o interpolación de los pixeles en caso del cómo sería un k-nearest neighbors o incluso redes neuronales convolucionales.

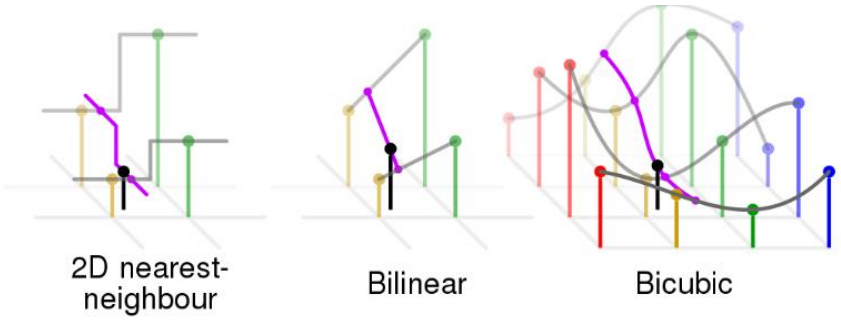



*Ilustración 21. Tipo de resampling de imágenes*

En este trabajo se ha utilizado el método de *interpolación bilineal* ya que es el mejor método para realizar downsampling tal y como se informa en la documentación oficial de tensorflow.

Se tiene que destacar que al aplicar esta transformación se modifica el *aspect ratio* de la imagen y en consecuencia puede provocar perturbaciones en los resultados.

### Transformación imágenes a tensores

Una vez decidido el tamaño de las imágenes que se va a usar, se procede a transformar las imágenes a un formato que la red neuronal pueda procesar, es decir a tensores.

Las imágenes se transforman en matrices de 256x256 con valores numéricos entre [0,255] y 3 canales (Rojo, Verde y Azul) que posteriormente se normalizan a valores entre 0 y 1.

Para una ejecución óptima es necesario definir correctamente el *input pipeline* ya que de esto depende el tiempo de ejecución y por lo tanto la factibilidad computacional de los modelos.

Como se ha comentado anteriormente, gracias a las GPU y TPU se puede reducir radicalmente el tiempo requerido para ejecutar un "training step". Para conseguir el mínimo tiempo de ejecución es necesario una canalización de entrada eficiente que proporcione la información (datos) de la siguiente iteración antes de que el actual haya terminado.

Para elegir qué modelo de datos es más convenientes se ha llevado a cabo un estudio de la performance entre la función *keras.preprocessing vs. tf.data (tensorflow)*, el primero es un procedimiento conservador mientras que el segundo es más flexible y eficiente.

### 4.1.1 Comparativa keras vs. tensorflow

**Pipeline**

1. Extracción: Se leen los datos de la memoria (*NumPy*) o del almacenamiento local (HDD o SSD) o remoto.

2. Transformación: Se utiliza la CPU para analizar y realizar las operaciones de preprocesamiento de los datos, como el shuffle (mezclar o desordenar las imágenes), el batching y las transformaciones específicas del caso de estudio, en este caso la normalización y descompresión de las imágenes.

3. Load: Cargar los datos transformados en los dispositivos de aceleración (por ejemplo, GPU (s) o TPU (s)) que ejecutan el modelo de aprendizaje automático.

**Tf.keras.preprocessing**

El training del modelo es bastante rápido gracias a los aceleradores (TPU o GPU) y debido a eso donde se suele generar cuellos de botella es en la siguiente iteración donde se requiere el uso de CPU. Mientras la CPU está preparando los datos (2), el acelerador está inactivo, por el contrario, mientras el acelerador está entrenando al modelo (3), la CPU está inactiva.

En consecuencia, el tiempo training es la suma del tiempo de procesamiento previo de la CPU y el tiempo de entrenamiento del acelerador.

Sin pipelines, la CPU y la GPU / TPU permanecen inactivas la mayor parte del tiempo:

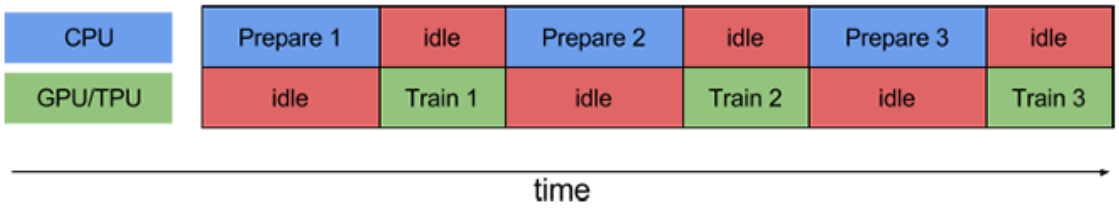



*Ilustración 22. Pipeline datos*

**Tf.Data**

La API *tf.data* permite definir un input pipelines que utilice eficazmente la CPU. El procedimiento de *tf.Data* se diferencia del *tf.keras.preprocesing* porque mientras el acelerador realiza el training step N, la CPU prepara los datos para el paso N + 1. Gracias a esto, se reduce el tiempo de extracción y transformación de los datos y por ello se minimiza el tiempo de training step.

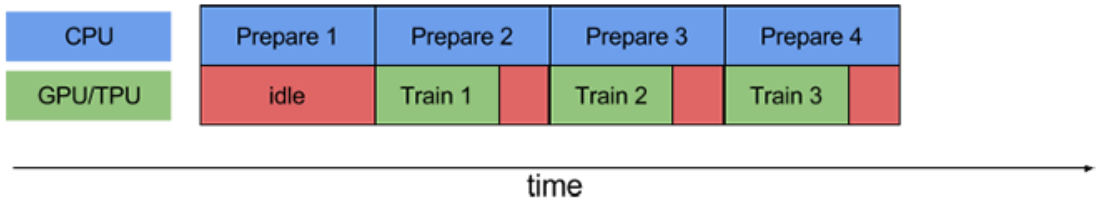



*Ilustración 23. Pipeline datos con CPU*

La API *tf.data* permite desacoplar el proceso de lectura de los datos (2) del proceso de tratamiento de los datos (3), así pues prepara los datos antes de que se soliciten mediante una serie de subprocesos y almacenado en el buffer la información, cabe destacar que el número de elementos almacenados en el buffer debe ser igual o superior a la dimensión del batch, este valor se puede ajustar manualmente o de forma dinámica con *prefetch.*

Gracias a que los elementos de entrada son independientes entre sí, el preprocesamiento puede ser paralelizado a través de múltiples núcleos del CPU.

Con este fin, la API *tf.data* ofrece la función *tf.data.Dataset.map* que aplica una función definida por el usuario a cada elemento del conjunto de datos de entrada. Si se utiliza esta función es necesario que se especifique el nivel de paralelismo (*num_parallel_calls*) que dependerá de las características del hardware, de los datos y de la función definida por el usuario. Un valor que se suele utilizar es el número que núcleos del CPU, también puedes definir que este valor sea escogido por tensorflow *on demand*.

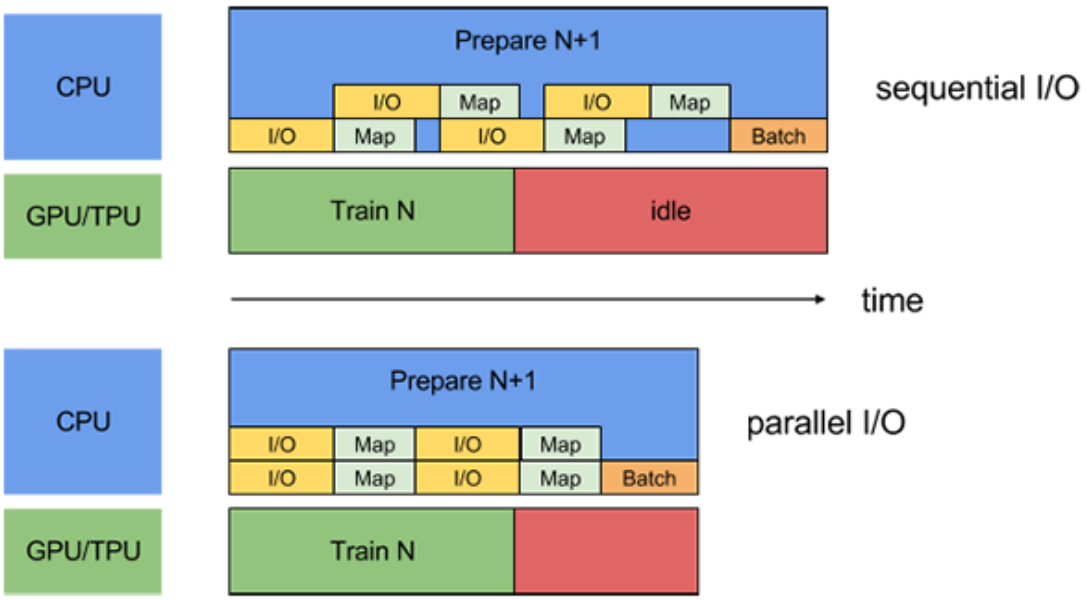



*Ilustración 24. Data pipeline 2 procesos en paralelo*

Finalmente, la API *tf.Data* proporciona alguna de las funciones típicas para que se realicen de forma combinada con las explicadas anteriormente, estas funciones ofrecen un mejor rendimiento ya que las simples se ven afectadas por el orden con el que se ejecutan. Por ejemplo, indicar shuffle antes de repeat o después, en el primer caso se garantiza el orden a costa del rendimiento y el segundo ofrece un mejor rendimiento.

La función *shuffle_and_repeat* combina las dos funciones de manera que no afecta al rendimiento. Otros ejemplos de funciones combinadas serian *map_and_batch* o *padded_batch_and_drop_remainder.*

**Comparativa**

Para comprar ambos procedimientos se calculará el tiempo de procesamiento de 100 steps, haciendo una media del tiempo por step y tiempo total.

Los resultados son los siguientes:

*Tabla 5. Performance data pipeline*

| | Imágenes/segundo | 100 batch (32 imágenes/batch) |
|---|---|---|
| *tf.data* | 133,76 | 95,69 segundos |
| *tf.keras* | 11,14 | 1148,67 segundos |

Como se puede observar, el modelo de Tensorflow Data procesa 100 batch de 32 imágenes cada uno en 95,69 segundos mientras que el modelo de lectura de imágenes de Tensorflow Keras tarda 1148,67 segundos, por lo tanto, *tf.Data* es 10 veces más rápido que *tf.keras*, por este motivo se utilizará el modelo Tensorflow Data. El efecto de utilizar keras implica que el entrenamiento del modelo sea más lento.

### 4.1.2 Data augmentation

Disponer de un número de imágenes elevado no siempre es posible, un ejemplo de ello es este caso de estudio.

Para hacer frente la falta de imágenes y así poder evitar el sobreajuste (overfitting en inglés) en las redes neuronales que se llevan a cabo en el presente trabajo, se ha utilizado una técnica llamada *data augmentation*.

Data augmentation se puede realizar sobre *data space* o bien sobre *feature space,* el primero hace referencia al conjunto de valores de la muestra de los datos, mientras que el segundo se refiere a todo el espacio de los datos tanto los muestrales como los que no. Un ejemplo claro de data augmentation en el *feature space* sería el uso de las GANs para generar datos realistas mientras que en el espacio muestral sería el uso de transformaciones de las imágenes como por ejemplo aplicar de forma aleatoria transformaciones como girar la imagen, cambiar la escala de color, aplicar zoom o rotarla, este método también se llama data wrapping. Un ejemplo de ello se muestra en la Ilustración 25.

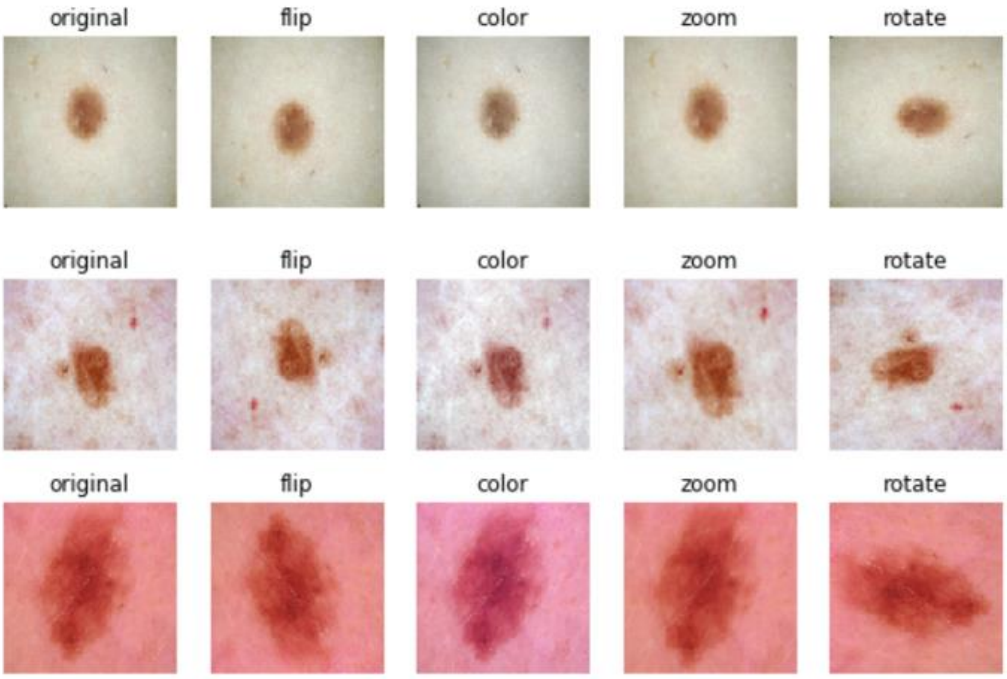



*Ilustración 25. Ejemplos data augmentation*

En el siguiente esquema se muestra el conjunto de herramientas de *data augmentation* aplicado en las imágenes.

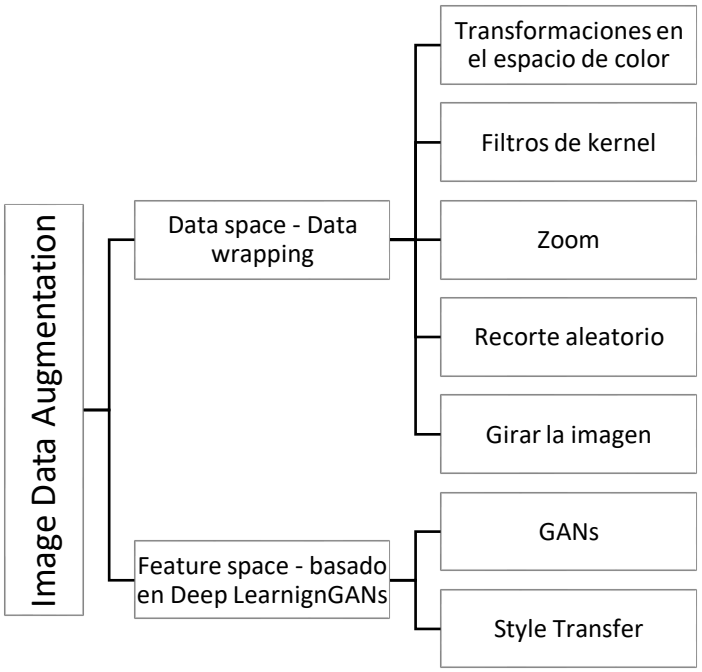



*Ilustración 26. Esquema data augmentation*

Estudios han demostrado que la técnica de data augmentation actúa como un regularizador previniendo así el overfitting en las redes neuronales y además en situaciones de clases desbalanceadas permite obtener mejores resultados.

## 4.2 CNNs para clasificar imágenes

Se define la siguiente arquitectura de red neuronal convolucional para clasificar las imágenes entre benignas y malignas.

Para definir el número de epoch óptimo se define la siguiente lógica:

1. Utilizar un valor elevado hasta conseguir el overfitting.
2. Identificar el punto en el cual el modelo deja de aprender y fijar ese valor como el número máximo de epoch.
3. Re-entrenar el modelo.
4. Cuando el modelo alcance el valor máximo de accuracy en validación, se guardan los pesos y el número de epoch.

Así pues, con esta metodología se obtiene un modelo más generalizable y sin overfitting.

El valor óptimo se guarda automáticamente con la función *keras.callbacks.modelcheckpoints*, en cada iteración se guarda un "checkpoint" con los pesos (weights) si el resultado del accuracy en la muestra de validación es el mayor que se ha encontrado. Finalmente se define el modelo con los pesos guardados.

Se ha utilizado como función de perdida la entropía cruzada binaria (cross-entropy) (21) y el optimizador RMSprop que está indicado para redes neuronales con mini-batch como es el caso. El optimizador RMSprop consiste en calcular el gradiente estocastico en cada uno de los mini-batch, después en vez de calcular la media se calcula la media móvil de los cuadrados del gradiente para cada peso y después se divide por la raíz de la media de los cuadrados. Finalmente, la métrica de evaluación utilizada es el accuracy.

$$Loss = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i \log p(y_i) + (1 - y_i) log(1 - \log p(y_i))\right] \qquad (21)$$

### 4.2.1 Modelo base

**Características**

- Input 256x256, 3 canales
- Mini batch: 64 imágenes (parámetro fijo)
- Epoch inicial: 100
- Max pooling

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 256, 256, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 128, 128, 32)      4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               33554944
_________________________________________________________________
dense_4 (Dense)              (None, 64)                32832
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 33,611,425
Trainable params: 33,611,425
Non-trainable params: 0
```

*Ilustración 27. Modelo 1 CNN*

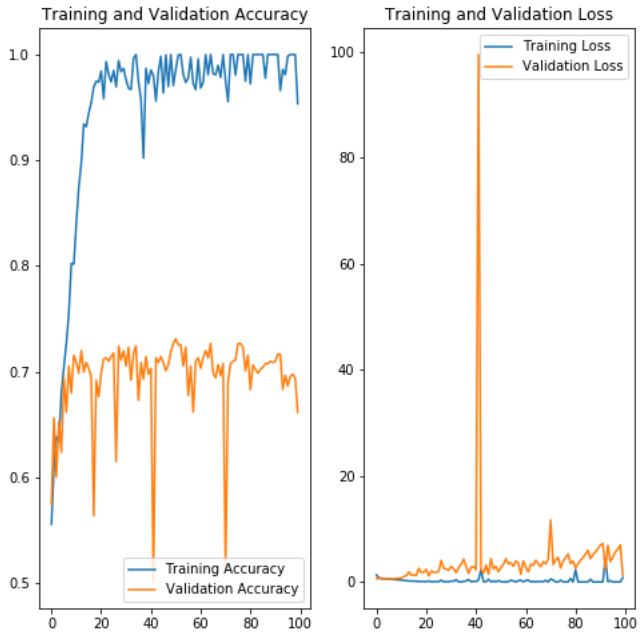



*Ilustración 28. Evolución accuracy y loss del modelo 1*

Como se puede observar en Ilustración 28, en las primeras iteraciones tanto la muestra de train como la de validación presentan la misma tendencia creciente en el accuracy y decreciente en la función de perdida (loss) por lo tanto el modelo está aprendiendo. Sin embargo, muy rápidamente el accuracy en la muestra de validación se estanca mientras que en la muestra de entrenamiento aumenta y de la misma forma sucede con la función de loss, mientras el train decrece, la validación sube. Por lo tanto, el modelo presenta overfitting.

### 4.2.2 Control del overfitting

Para corregir el problema de overfitting se introducen dos conceptos el dropout y regularizadores del kernel.

#### Dropout

El dropout es una técnica que ayuda al entrenamiento de redes neuronales que presentan overfitting, es decir que después de un número de iteraciones el modelo deja de aprender y memoriza el input y en consecuencia no generaliza bien.

Este método consiste en eliminar o fijar su valor a cero de forma aleatoria a alguno de los pesos estimados, en la Ilustración 29. Ejemplo dropout se muestra un ejemplo del kernel en el cual se le aplica un dropout del 50% es decir la mitad de los valores se fijan a cero de forma aleatoria.

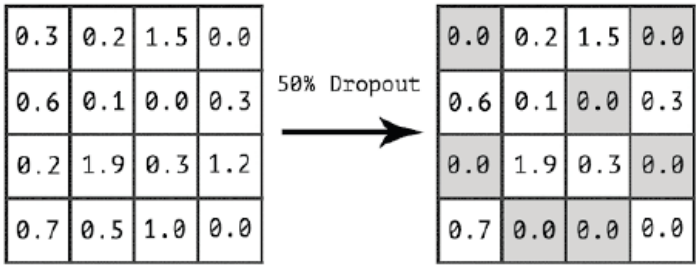



*Ilustración 29. Ejemplo dropout*

#### Regularizadores del kernel

Los valores que toma el kernel se van definiendo de forma iterativa a partir del gradiente y de la función de pérdida, para evitar el overfitting y que algunos de estos valores definan el modelo y por lo tanto predominen en la clasificación se aplican restricciones a los valores que puede tomar el kernel.

En la siguiente fórmula se muestra como se ve modificada la función de coste aplicando una regularización L2.

$$Cost\ function\ =\ Loss\ +\frac{\lambda}{2m}*\sum\|w\|^2 \qquad (22)$$

La función de coste a minimizar es la función de perdida más una penalización según los valores que toma el kernel $w$, a mayor valor toman, mayor penalización sufre la función

de coste, la penalización dependerá del factor $\frac{\lambda}{2m}$ que se define según cuanto se quiere penalizar.

Siguiendo con la metodología, se llevará a cabo los siguientes modelos, el primero con dropout del 20%, el segundo con regularización L2 con factor de 0.001 y finalmente con dropout y regularización.

### 4.2.3 Modelo base con dropout

El modelo base presentaba overfitting con el objetivo de combatirlo se ha añadido en las capas dropout del 20%.

#### Características

- Input 256x256, 3 canales
- Mini batch: 64 imágenes (parámetro fijo)
- Epoch: 100
- Max pooling
- Dropout 20%

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 256, 256, 16)      448
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 128, 128, 16)      0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 128, 128, 32)      4640
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 64, 64, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 32, 32, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               33554944
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                32832
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 65
=================================================================
Total params: 33,611,425
Trainable params: 33,611,425
Non-trainable params: 0
```

*Ilustración 30. Modelo 2*

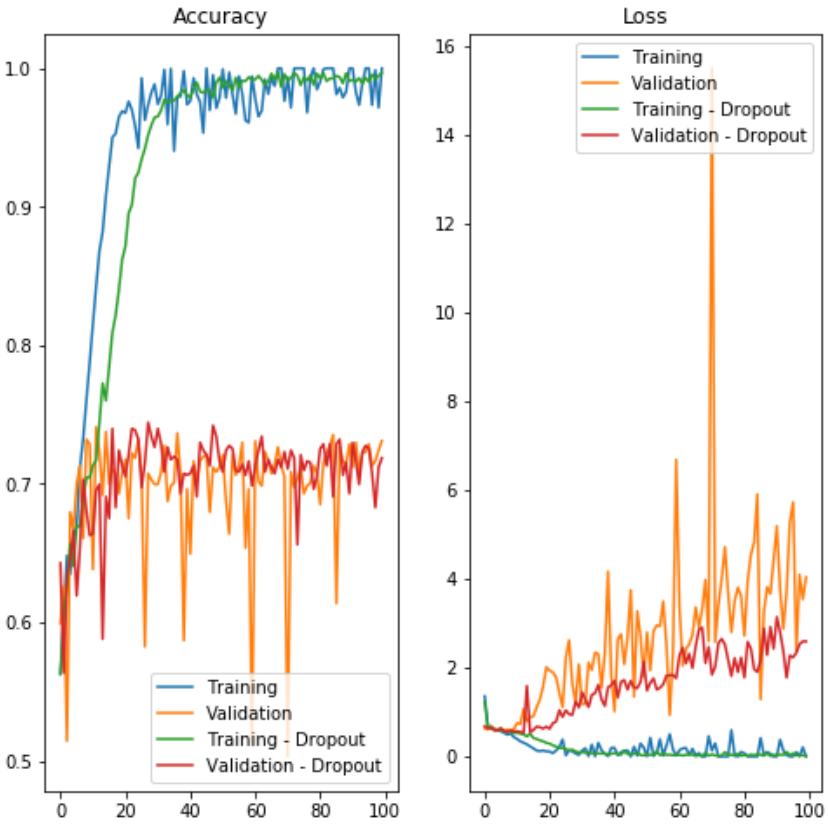



*Ilustración 31. Training modelo con dropout*

Como se puede observar en la Ilustración 31, añadiendo un dropout del 20% no es suficiente para prevenir el overfitting. Vemos que la función de loss en la validación con dropout es ligeramente inferior que sin él, por lo tanto el dropout está afectando positivamente al modelo ya que el objetivo final es que la tendencia de la función de perdida en la muestra de train y validación sea lo más parecido posible y así pues que el accuracy en la muestra de validación este próximo al de entrenamiento.

### 4.2.4 Modelo base con regularización del kernel

El modelo con dropout del 20% sigue presentando problemas de overfitting por ese motivo se estimará el modelo con regularizadores del kernel L2 con factor 0.001.

#### Características

- Input 256x256, 3 canales
- Mini batch: 64 imágenes (parámetro fijo)
- Epoch: 100
- Max pooling
- Regularización L2 ratio 0.001

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 256, 256, 16)      448
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 128, 128, 32)      4640
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 32, 32, 64)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_15 (Dense)             (None, 512)               33554944
_________________________________________________________________
dense_16 (Dense)             (None, 64)                32832
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 65
=================================================================
Total params: 33,611,425
Trainable params: 33,611,425
Non-trainable params: 0
```

*Ilustración 32. Modelo base con regularización*

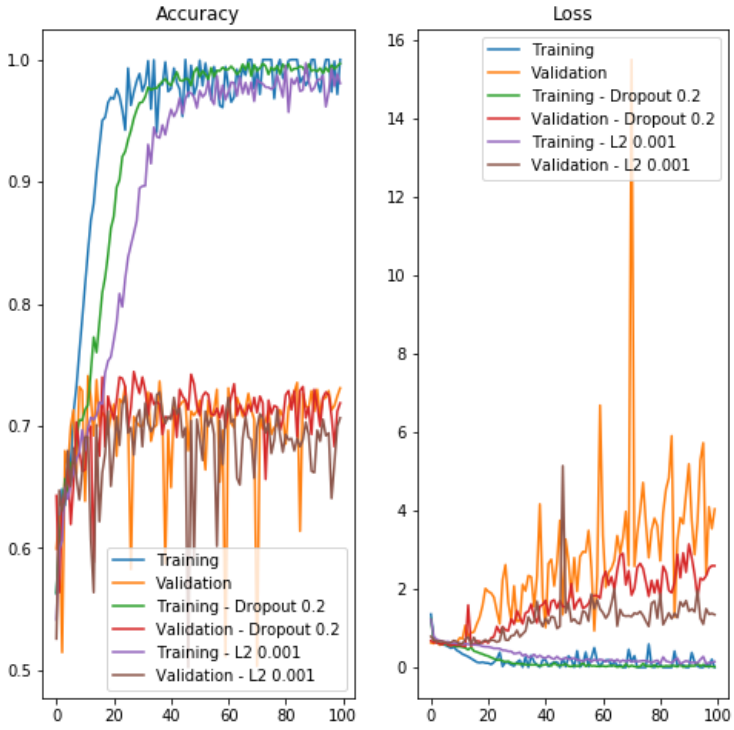



*Ilustración 33. Training modelo con regularizaciones*

A partir de los resultados, al añadir regularizaciones en cada capa del modelo se observa que el accuracy sigue estancándose y no mejora, pero la función de perdida de la muestra de validación está cada vez más cerca de la muestra de entrenamiento.

Finalmente se añadirá al modelo con regularización L2 dropout.

### 4.2.5 Modelo base con regularización y dropout

Finalmente, para escoger que arquitectura se utilizará se contempla la última combinación, modelo base con regularizador L2 y dropout.

#### Características

- Input 256x256, 3 canales
- Mini batch: 64 imágenes (parámetro fijo)
- Epoch: 100
- Max pooling
- Dropout 20%
- Regularización L2 factor 0.001

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 16)      0
_________________________________________________________________
dropout (Dropout)            (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 65536)             0
_________________________________________________________________
dense (Dense)                (None, 512)               33554944
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                32832
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 33,611,425
Trainable params: 33,611,425
Non-trainable params: 0
```

*Ilustración 34. Modelo con dropout y regularización*

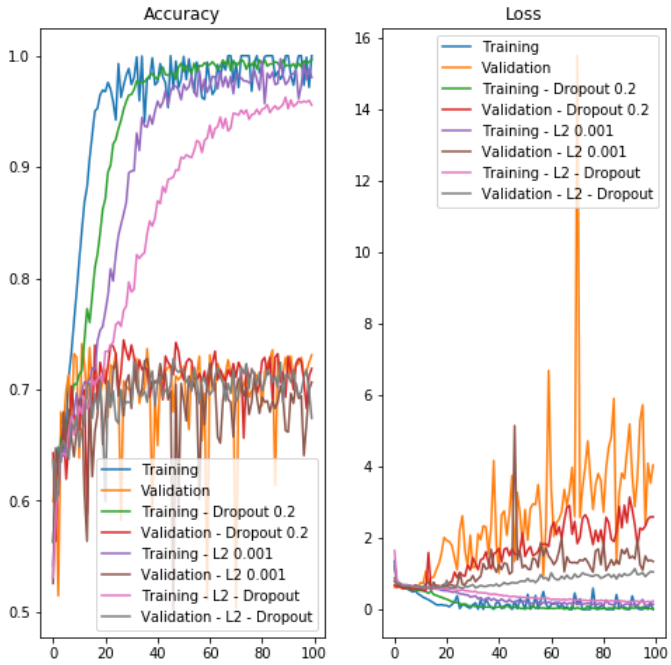



*Ilustración 35. Training modelo con dropout y regularizaciones*

Se observa que no se ha podido mejorara el accuracy ni tampoco evitar el overfitting pero en cuanto a la función de perdida, el modelo con dropout y regularizador L2 presenta una función de perdida en la muestra de validación más próxima a la de entrenamiento.

### 4.2.6 Modelo base con data augmentation, regularización y dropout

Finalmente, como última herramienta para prevenir el overfiting e intentar mejorar el rendimiento del accuracy en la muestra de validación se utilizará data augmentation tal como se ha comentado anteriormente en el apartado 4.1.2.

Un problema que puede generar el uso de data augmentation dentro del ámbito biomédico es que puede generar distorsiones en las imágenes que dificulten a la red identificar los patrones de cada clase. Por ejemplo, si la red neuronal destaca el color para identificar de que clase es utilizar esta técnica perjudica la performance del modelo.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 256, 256, 16)      448
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 128, 128, 16)      0
_________________________________________________________________
dropout_15 (Dropout)         (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 128, 128, 32)      4640
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 64, 64, 32)        0
_________________________________________________________________
dropout_16 (Dropout)         (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 32, 32, 64)        0
_________________________________________________________________
dropout_17 (Dropout)         (None, 32, 32, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               33554944
_________________________________________________________________
dropout_18 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                32832
_________________________________________________________________
dropout_19 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 65
=================================================================
Total params: 33,611,425
Trainable params: 33,611,425
Non-trainable params: 0
```

*Ilustración 36. Modelo con regularización, dropout y data augmentation*

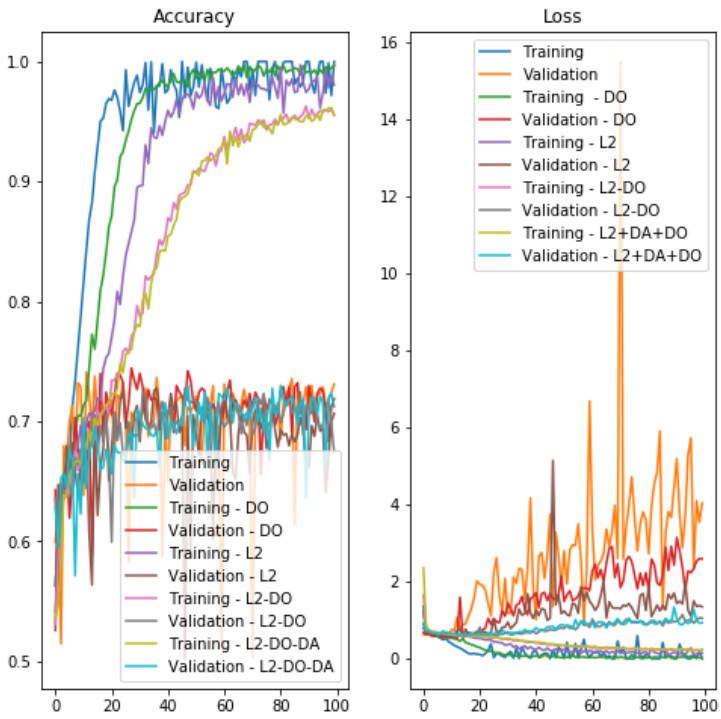



*Ilustración 37. Training modelo con dropout, regularizacion y data augmentation*

Con data augmentation se consigue reducir aún más el overfitting, como se puede observar la función de perdida de la muestra de validación más cercana a su respectiva función de perdida de la muestra de entrenamiento es el modelo con data augmentation, regularización L2 y dropout (L2+DA+DO).

### 4.2.7 Resultados

Después de haber analizado los modelos y las características gráficas toca decidir qué modelo escogemos para realizar la clasificación.

Tal como se ha comentado anteriormente, los modelos se han estimado con la función de keras callbacks que guardaba los pesos en los cuales el modelo maximizaba el accuracy por lo tanto para cada uno de ellos tenemos la configuración que optimiza el accuracy en la muestra de validación.

*Tabla 6. Resultados modelo*

| | Accuracy (en %) | Sensibilidad (en %) | Epoch |
|---|---|---|---|
| Modelo base | 69 | 74 | 11 |
| Modelo base con dropout | 73 | 78 | 27 |
| Modelo base con regularización | 70 | 78 | 36 |
| Modelo base con regularización y dropout | 73 | 81 | 46 |
| Modelo base con regularización, dropout y data augmentation | 71 | 68 | 47 |

El modelo que mayor accuracy presenta en la muestra de validación es el modelo con dropout del 20% y regularización L2 presentando un 73% de acierto y la sensibilidad en que clasifica una imagen de piel maligna es del 81%.

En este caso de estudio no solamente es importante el accuracy si no también es la sensibilidad en que se asigna a una imagen maligna la etiqueta maligna ya que no tiene las mismas consecuencias clasificar una imagen benigna como maligna que viceversa, por este motivo se escoge el modelo con regularización y dropout ya que tiene mejor sensibilidad y es casi el mismo nivel de accuracy.

## Evaluación del modelo

*Tabla 7. Matriz de confusión - contaje*

| | Benigno | Maligno |
|---|---|---|
| Benigno | 299 | 162 |
| Maligno | 81 | 354 |

*Tabla 8. Métricas de performance*

| | % |
|---|---|
| Accuracy | 72,88 |
| Error rate | 27,12 |
| Precisión | 68,60 |
| Sensibilidad | 81,37 |
| Especificad | 64,86 |

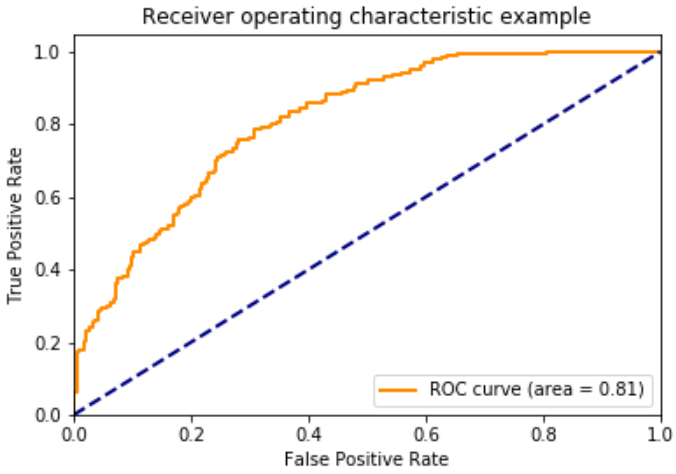



*Ilustración 38. Curva ROC*

El modelo escogido presenta un accuracy del 72,88% además presenta un AUC de 81%.

Finalmente, vamos a analizar los resultados en la muestra de prueba.

*Tabla 9. Matriz de confusión*

| | Benigno | Maligno |
|---|---|---|
| Benigno | 146 | 84 |
| Maligno | 41 | 177 |

En la muestra de prueba se obtiene un accuracy del 72,10 %.

### 4.2.8 Visualización activaciones de las capas intermedias

Tal como se comentó anteriormente, las redes neuronales convolucionales transforman la imagen de entrada a partir del kernel que se va definiendo en el entrenamiento así pues, se obtienen sucesivos tensores 3D con una dimensión height, weight y canal definidos por la arquitectura de la CNNs.

En este apartado se mostrará cómo va evolucionando la imagen de entrada en cada uno de los canales de cada capa convolucional.

**Input**

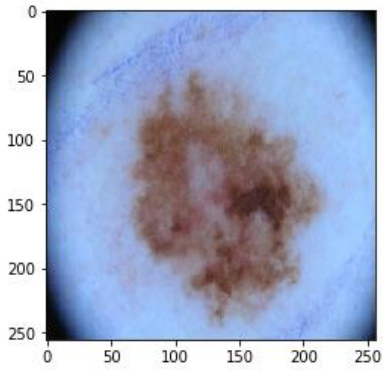



*Ilustración 39. Imagen de muestra maligna*

Esta imagen pertenece a un melanoma maligno, a continuación vamos a observar cómo va cambiando por cada canal de las capas convolucionales del modelo anteriormente definido.

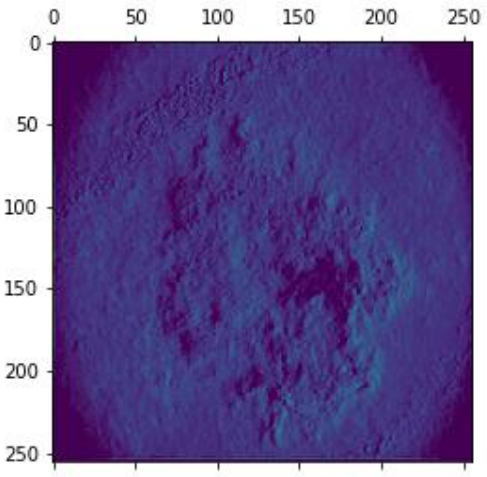



*Ilustración 40. Imagen maligna capa 1 canal 8*

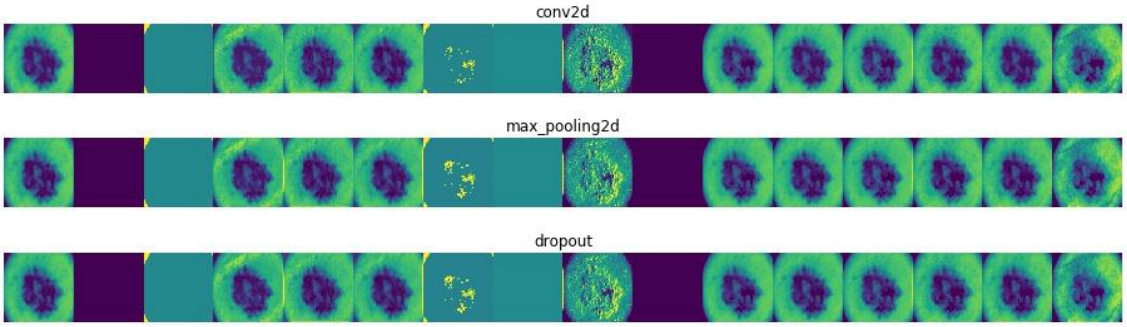



*Ilustración 41. Feature map de la primera convolución*

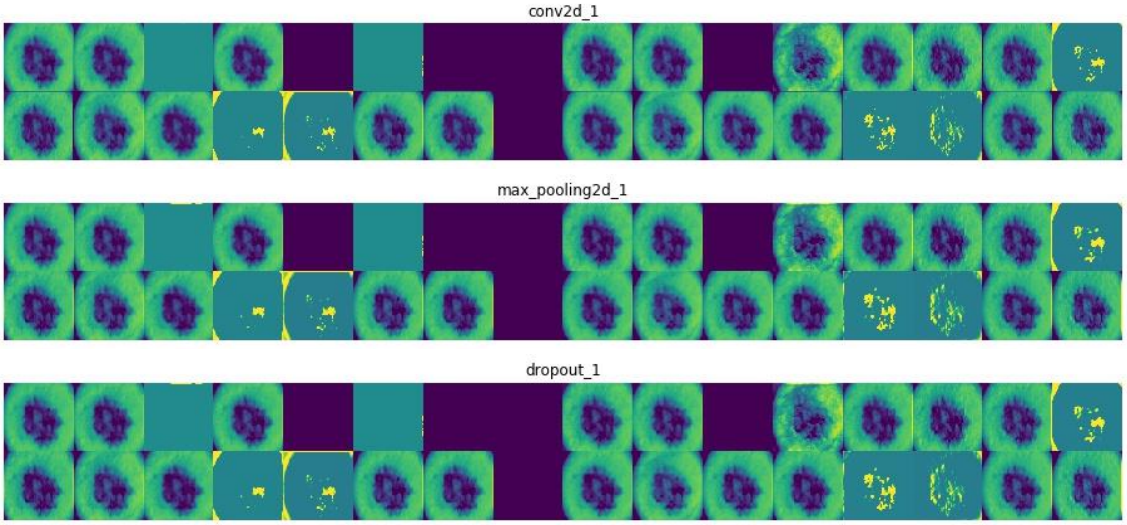
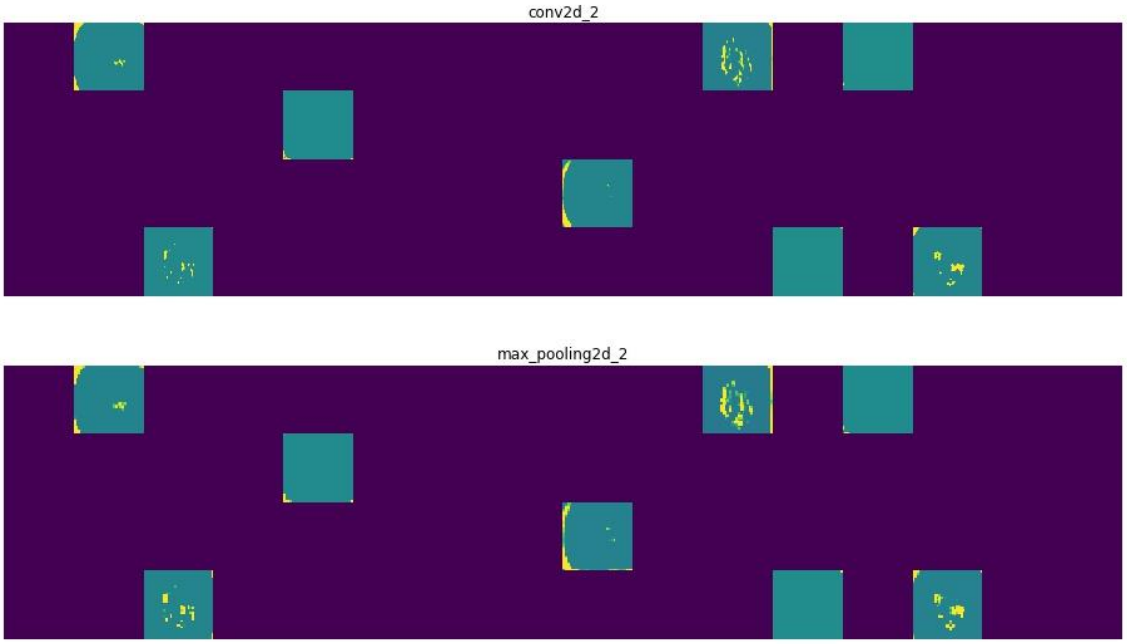



*Ilustración 42. Feature map de la segunda convolución*



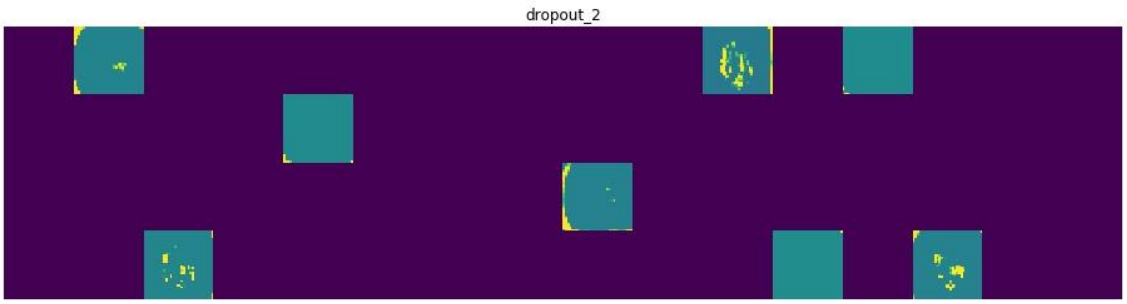



*Ilustración 43. Feature map de la tercera convolución*

Como se puede observar en cada capa la CNNs va profundizando en las características de la lesión de piel. En la primera capa, se destaca la forma o el contorno, a medida que pasan las sucesivas capas, se detectan elementos más concretos. La última convolución aparentemente capta el contorno interior de la lesión.

En la primera capa, casi todos los filtros se activan con la imagen de entrada pero a medida que nos adentramos en las capas posteriores, la mayoría de los feature maps están en negro, esto sucede cuando no se da la activación, es decir el patrón que detecta el filtro no se encuentra en la imagen.

## 4.3 GANs para la creación de imágenes de piel

En este apartado se realizará una red generativa antagónica para generar imágenes piel humana.

Por motivos computacionales, se generarán imágenes de 128 x 128 pixeles.

### Características Generador

- Dimensión espacio latente: 100
- Kernel 5x5
- Strides 2x2
- Batch normalization
- Leaky ReLu

Se ha diseñado el generador de la siguiente manera:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32768)             3276800
_________________________________________________________________
batch_normalization (BatchNo (None, 32768)             131072
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32768)             0
_________________________________________________________________
reshape (Reshape)            (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 64, 64, 16)        12800
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 64, 16)        64
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 128, 128, 3)       1200
=================================================================
Total params: 3,421,936
Trainable params: 3,356,368
Non-trainable params: 65,568
```

*Ilustración 44. Generador*

Partiendo de una normal n-dimensional, se sacará puntos al azar. Al principio cuando el generador no ha entrenado si se representa un punto al azar tiene el siguiente aspecto:

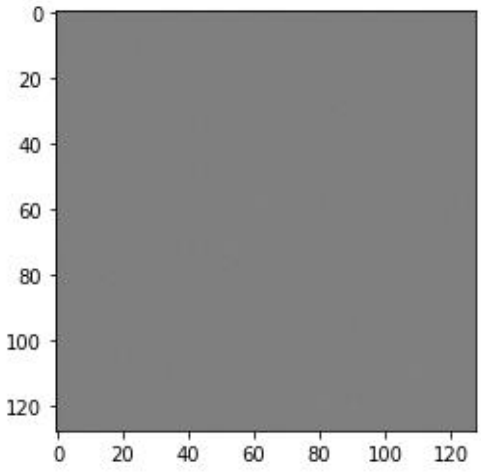



*Ilustración 45. Ejemplo ruido aleatorio*

Esta imagen que al principio es solo ruido, se ira transformando de forma iterativa en imágenes realistas.

### Características Discriminador

- Mini batch: 64 imágenes de 128x128 y 3 canales
- Kernel 5x5
- Strides 2x2
- Dropout 30%
- Leaky ReLu

Se ha diseñado el discriminador de la siguiente manera:

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 64, 64, 64)        4864
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64, 64, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 128)       204928
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 131072)            0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 131073
=================================================================
Total params: 340,865
Trainable params: 340,865
Non-trainable params: 0
```

*Ilustración 46. Discriminador*

A partir de las imágenes reales y generadas, se define el discriminador que tiene como input una imagen de 128x128 pixeles y 3 canales. El output del discriminador será una probabilidad entre 0 y 1 que la imagen sea real o generada.

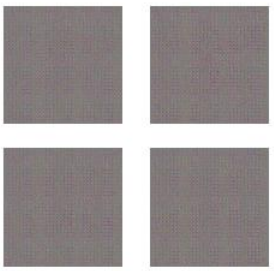
La función de pérdida del discriminador se define como la suma de la función de perdida de las imágenes reales más la de las generadas. Mientras que la función de pérdida del generador es la asociada a la función de pérdida de las imágenes generadas, tal como se comentó en el apartado 3.2.

Las funciones de perdida escogidas es la entropía cruzada binaria o “Binary cross entropy”.

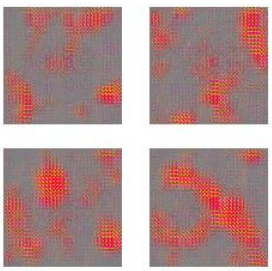
Para optimizar el generador y el discriminador se ha utilizado la función Adam que es una modificación del Optimizador RMSProp (Root Mean Square Propagation) que se basa en una ratio de aprendizaje adaptativa. Por ejemplo, si los parámetros están muy dispersos (sparse) la ratio de aprendizaje aumentará.

### Resultados

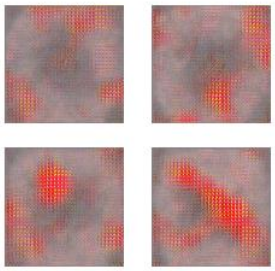
Después de 1000 iteraciones, aproximadamente 8h de ejecución 28s por epoch, la evolución de las imágenes generadas por el generador es el siguiente:



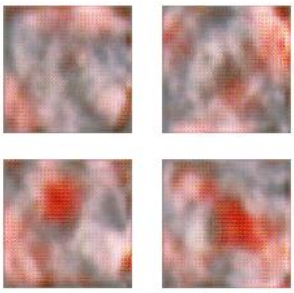
Iteración 1



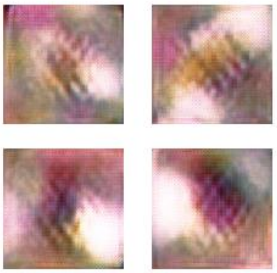
Iteración 5



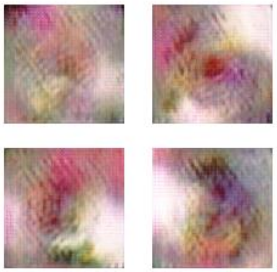
Iteración 10



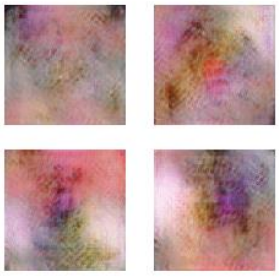
Iteración 20



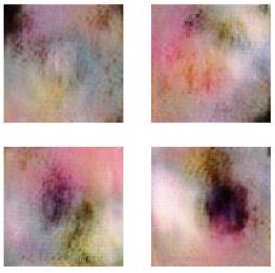
Iteración 50



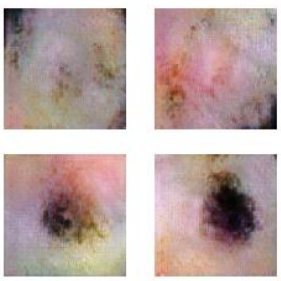
Iteración 100



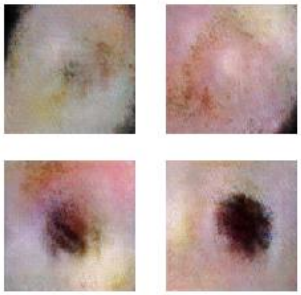
Iteración 250



Iteración 500



Iteración 750



Iteración 1000

### 4.3.1 Evaluación de los resultados

Para evaluar que las imágenes generadas comparten espacio vectorial con las imágenes de muestra se ha utilizado un método de reducción de la dimensionalidad: el análisis de componentes principales.

El análisis de componentes principales es una técnica de reducción lineal de la dimensionalidad de la matriz de datos, forma parte del conjunto de análisis factoriales. Esta técnica consiste en reducir la dimensión de la matriz de datos a partir de la matriz de covariancia de la cual se extraen unos nuevos ejes o variables tales que maximicen la variabilidad explicada de la proyección de nube de puntos sobre ellos, estos ejes se les llama componentes.

El objetivo pues es encontrar aquel conjunto de nuevas variables o ejes que al representar los individuos en ese plano la inercia proyectada sea máxima.

Para obtener los componentes principales, en primer lugar se define la combinación lineal de variables que maximicen la inercia total, este es el primer componente el cual es el que mayor variabilidad explica. Los siguientes componentes se calculan teniendo en cuenta que han de estar incorrelacionado con el primer componente y que explique la máxima variabilidad.

Una vez obtenido los componentes principales se transforma los valores de los individuos o filas en cada uno de los componentes que son combinaciones lineales de las variables originales.

Así pues, en este apartado se proyectarán en el primer plano factorial (componente 1 y 2) las imágenes reales y las generadas, analizando si forman grupos inconexos o bien están unidos, si se da el último caso se concluirá que los datos generados comparten espacio vectorial con los datos reales.

Las imágenes generadas constituyen cada una matriz de 128 filas y 128 columnas además cada una de ellas tiene 3 canales (rojo, azul y verde). Para llevar a cabo el análisis factorial se realiza un flatten (pasar del tensor 3D a un vector), obteniendo para cada imagen un vector de dimensión 49.152 (128x128x3).

Se escogen al azar 100 imágenes reales y se generan 100 mediante la GAN, después se ha transformado cada imagen a vector y se ha convertido en una base de datos de 200 filas y 49.152 columnas. Sobre esta matriz de datos se ha realizado el PCA, a continuación se muestra el primer plano factorial:

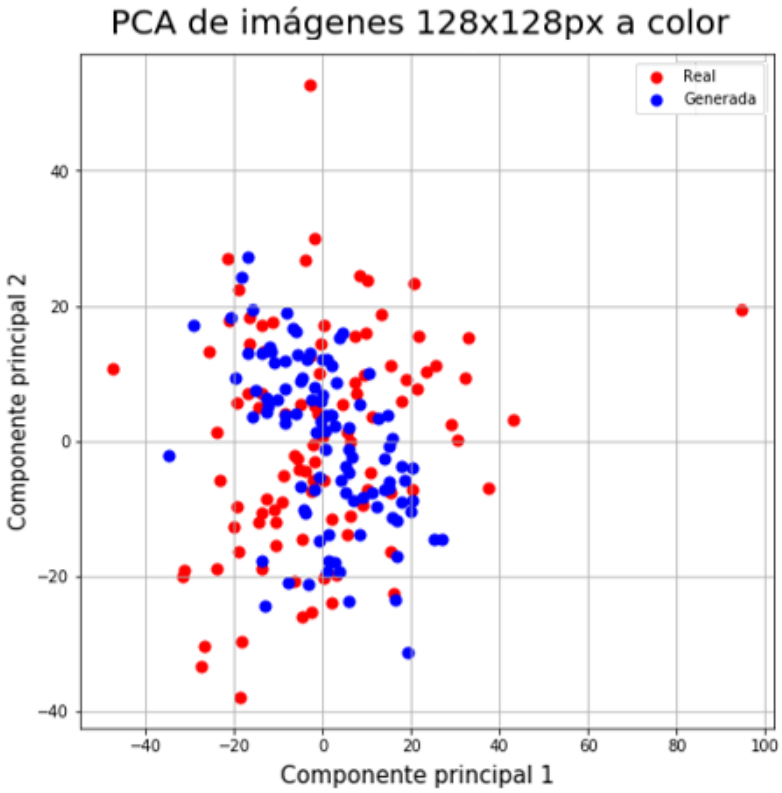



*Ilustración 47. PCA de las imágenes reales y generadas*

El primer plano factorial explica el 48,34% de la variabilidad total de la nube, se puede observar que el grupo de puntos asociado a las imágenes reales no se puede separar o discriminar del grupo de puntos de las imágenes generadas, por lo tanto se concluye que el generador ha conseguido definir el espacio latente de las imágenes.

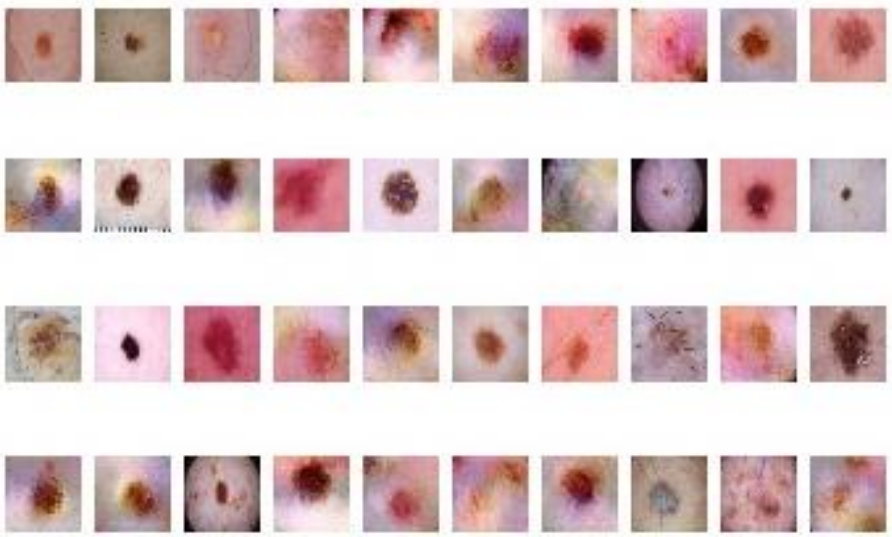


*Ilustración 48. Imágenes reales y generadas*

En la Ilustración 48, se muestran 20 imágenes reales versus 20 imágenes generadas con la GAN.

# 5 Conclusiones

Las redes neuronales permiten resolver multitud de problemas. Como se ha podido observar en el siguiente trabajo, las redes neuronales convolucionales son capaces de clasificar si una imagen de piel humana presenta o no características de cáncer de piel. Además, se ha conseguido generar imágenes realistas de tejido humano usando las redes generativas antagónicas.

Gracias a los avances tecnológicos y científicos hoy en día es posible estimar los pesos de una red neuronal de grandes dimensiones, como es el caso, en un tiempo mínimo de 6 segundos por epoch, además de la serie de optimizadores del input y del cálculo como son las GPUS.

En este trabajo se ha utilizado una base de datos de 23Gb, que se leían, transformaban y almacenaban en tensores en apenas 40 segundos, este procedimiento y rapidez hubiera sido impensable hace 10 años. El hardware cada vez más permite a la sociedad acercarse al machine learning y por lo tanto que no esté controlado por aquellos individuos que disponen del capital suficiente para adquirir hardware potente.

Aunque los resultados obtenidos son satisfactorios cabe destacar que si se hubiera tenido mayor tamaño muestral hubiera sido posible llevar a cabo mayor número de aplicaciones de redes neuronales como por ejemplo realizar un modelo predictivo del diagnóstico de la lesión de piel yendo más allá de clasificar benigno o maligno, dando el diagnostico si la imagen presenta melanoma, nevo melanocítico, carcinoma de células basales, queratosis actínica o queratosis benigna entre otros, aprovechando la información del historial médico del paciente.

Un ejemplo de ello sería una red neuronal con múltiple input, de las cuales una de ellas correspondería a una red neuronal convolucional que trataría las imágenes y por otro lado, una red que interpretara las características del historial médico del paciente (imagen), así pues una vez entrenadas las redes se unen para dar un único output, el diagnostico.

Otro ejemplo que no se ha podido llevar a cabo por el tamaño muestral y la falta de hardware pero que sería muy interesante a nivel científico y personal es realizar una red generativa antagónica condicional que fuera capaz de generar imágenes de piel con cáncer así pues seriamos capaces de analizar mediante interpolación lineal las sucesivas transformaciones que definen una lesión cancerígena hacia una que no lo es.

Finalmente, quiero destacar el problema de overfitting que suelen presentar los modelos de Machine Learning.

Como se ha podido observar en el apartado de control del overfitting, existen numerosas herramientas que ayudan a reducirlo y evitarlo, pero a veces el modelo no es capaz de aprender más, es decir colapsa. En esta situación aunque se utilicen diferentes arquitecturas y controles el overfitting, la performance no se puede.

Como conclusión, a medida que pase el tiempo y se vaya estableciendo la cultura del dato, cada vez será más fácil resolver los distintos problemas y desafíos de la sociedad.

# 6 Referencias


Aggarwal, C. C. (2018). Neural networks and deep learning. *Springer*.

Arjovsky, M. (2017). Towards principled methods for training generative adversarial. *arXiv:1701.04862*.

Arjovsky, M. (2017). Wasserstein GAN. *arXiv:1701.07875*.

Borji, A. (2018). Pros and Cons of GAN Evaluation Measures. *arXiv:1802.03446v5*.

Brownlee, J. (19 de Agosto de 2019). A Gentle Introduction to StyleGAN the Style Generative Adversarial Network.

Chollet, F. (2017). *Deep Learning with R.* Manning Publications.

Goodfellow, I. J. (2014). Generative Adversarial Networks. *arXiv:1406.2661*.

GrégoireMontavon. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing 73*, 1-15.

ISIC. (23 de Diciembre de 2019). *ISIC Archive*. Obtenido de https://www.isic-archive.com/

Isola, P. (2016). Image-to-image translation with conditional adversarial networks. *arXiv:1611.07004*.

Jou, P. (2012). UV protection and sunscreens: what to tell patitents. *Cleveland Clinic Journal of Medicine*.

Keras. (23 de Diciembre de 2019). *Keras*. Obtenido de https://keras.io/

Lipton, Z. C. (2017). Precise recovery of latent vectors from generative adversarial networks. *arXiv:1702.04782*.

Liu, Z. (2015). Deep learning face attributes in the wild. *In Proceedings of International Conference on Computer Vision (ICCV)*.

Pieters, M. (2018). Comparing Generative Adversarial Network. *arXiv:1803.09093v1*.

Radford, A. (2015). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* arXiv:1511.06434.

Rumelhart, D. (19). Learning Internal Representations. En D. Rumelhart, *Learning Internal Representations* (págs. 319-362).

Sebastien, C. W. (2016). Understanding Data Augmentation for Classification: When to Warp? *IEEE*.

Sjöstrand, J. J. (2018). *Cell Image Transformation Using.* Lund University.

Tensorflow. (23 de Diciembre de 2019). *Tensorflow.* Obtenido de https://www.tensorflow.org/

Visin, V. D. (2016). A guide to convolution arithmetic for deep learning. *arXiv:1603.07285*.

World Health Organization. (2014). *World Cancer Report 2014.*

Yu Lia, C. H. (2019). Deep learning in bioinformatics: Introduction, application, and perspective. *Methods*, 4-21.