
# Detección del carcinoma de glándula mamaria mediante termografía infrarroja usando redes neuronales profundas

**Nombre Estudiante**
**Francisco Javier Fernández Ovies**
Máster en Bioinformática y Bioestadística
Área de Machine Learning

**Nombre Director, Consultor**
**Edwin Santiago Alférez Baquero**
PhD in Biomedical Engineering

Enero de 2019

**FICHA DEL TRABAJO FINAL**

| | |
|---|---|
| **Título del trabajo:** | Detección del carcinoma de glándula mamaria mediante termografía infrarroja usando redes neuronales profundas. |
| **Nombre del autor:** | Francisco Javier Fernández Ovies |
| **Nombre del consultor/a:** | Edwin Santiago Alférez Baquero |
| **Nobre del PRA:** | Ferran Prados Carrasco |
| **Fecha de entrega (mm/aaaa)** | 02/01/2018 |
| **Titulación** | Máster en Bioinformática y Bioestadística |
| **Área del Trabajo Final:** | Machine Learning |
| **Idioma del trabajo:** | Español |
| **Palabras clave:** | Breast cancer, thermography, deep neuronal network. |


**Resumen del Trabajo:**

El objetivo principal de este trabajo es la evaluación de una red neuronal convolucional que facilite la detección precoz del cáncer de mama utilizando imágenes de termografía infrarroja.

En la literatura se han presentado resultados satisfactorios en estudios previos utilizando otros algoritmos. Los principales retos de la utilización de redes neuronales son dos, disponer de un gran número de imágenes y el tiempo de procesamiento. El planteamiento es utilizar las funciones de alto nivel implementadas en la librería fast.ai, sobre la plataforma Pytorch, que han obtenido resultados excelentes en la clasificación de imágenes, tal como lo avalan sus éxitos en las competiciones de Kaggle, referencia mundial en las tecnologías Machine Learning.

Las imágenes se organizaron en los tres grupos habituales: training, validation y test; lo que nos permitió contrastar los resultados utilizando diferentes arquitecturas pre-entrenadas (resnet18, resnet34, resnet50, resnet152, vgg16 y vgg19) permitiendo agilizar el tiempo de procesamiento, y valorar su clasificación mediante matrices de confusión, obteniendo el mejor resultado con resnet34, con un accuracity en el test de 0.985.

**Abstract:**

The main objective of this work is the evaluation of a convolutional neuronal network that facilitates the early detection of breast cancer using infrared thermography images.

The literature shows satisfactory results in previous studies using other algorithms. The main challenges of using neural networks are two, having a large number of images and processing time. The approach is to use the high-level functions implemented in the fast.ai library, on the Pytorch platform, which have obtained excellent results in the image classification, as endorsed by their successes in the Kaggle competitions, a world reference in technologies Machine Learning.

The images are organized in the three usual groups: training, validation and testing; what has told us to contrast the results using different pre-trained architectures (renet18, resnet34, resnet50, resnet152, vgg16 and vgg19) use the processing time, assess their classification by means of confusion matrices, obtain the best result with resnet34, with a accuracy in the 0.985 test.

## Índice:

# Lista de Figuras

# Lista de tablas

# Lista de códigos

# 1. Introducción

## 1.1 Contexto y justificación del trabajo

El cáncer de mama es una de las principales causas de mortalidad femenina en los países desarrollados. La detección precoz es de suma importancia [13], aumentando las tasas de supervivencia de los pacientes y redundando en la economía de los sistemas sanitarios.

La imagen médica mediante diferentes tecnologías: rayos X, ultrasonidos y resonancia magnética; ha sido fundamental en la detección precoz del cáncer de mama.

La termografía infrarroja se ha mostrado como una herramienta económica y no invasiva para la detección precoz del cáncer de mama, aunque en su defecto tiene un elevado número de falsos positivos y falsos negativos, entorno al 10%, además de no localizar con precisión el área afectada [3, 4].

La detección de regiones con gradientes altos de temperatura y el análisis geométrico a partir de puntos característicos, permitiendo detectar de forma automática regiones de interés (ROI) para cada seno [6], y la consiguiente detección de asimetrías [7, 8], han sido un avance importante.

La automatización del proceso anterior con resultados positivos se ha puesto de relieve en los trabajos de Silva et al. [1] y U. Acharya et al. [2], utilizando características de textura y el clasificador de máquinas de vectores de soporte (SVM). Así como la utilización del clasificador K-NN por Mejía et al. [7] o SVM-RFB por Sadek Ali et al. [9] y Sathish et al. [10].

Es importante destacar los trabajos para la implementación de bases de datos de acceso abierto, en este caso de termografía infrarroja de mama, que permitan desarrollar y comparar diversas técnicas de análisis de imágenes [5].

Otros estudios involucran al desarrollo de simulaciones numéricas para estudiar los efectos térmicos de las propiedades tumorales de Satish G. Kandlikar et al. [11]. El uso de redes neuronales convolucionales (Convolutional Neural Network - CNN) se encuentra muy extendido en el análisis de imágenes por sus excelentes resultados, pero sin embargo no constan estudios en los que se haya aplicado esta tecnología a termografías y más concretamente en el carcinoma de mama. Es objetivo de este trabajo valorar la utilización de las CNN en la catalogación de pacientes con cáncer de mama a partir de termografías.

## 1.2 Objetivos generales y específicos del trabajo

### 1.2.1 Objetivo principal:

Evaluar las Redes Neuronales Convolucionales (CNN) para la detección precoz del cáncer de mama basándose en imágenes de termografía.

### 1.2.2 Objetivos específicos:

- Desarrollar un procedimiento que permita clasificar pacientes sanas y enfermas en cáncer de mama utilizando CNN:
  - Explorar imágenes termográficas de mama.
  - Crear un conjunto de base de datos de imágenes termográficas para el entrenamiento y evaluación de la CNN.
  - Estudiar la librería de Fast.ai [27] para clasificación de imágenes.
  - Clasificar la base de datos mediante las funciones de alto nivel de Fast.ai.

- Aplicar el procedimiento a diferentes arquitecturas:
  - Realizar una selección aleatoria de las imágenes, organizándolas en los habituales grupos de training, test y validation.
  - Aplicar el procedimiento a diferentes arquitecturas de CNN.
  - Evaluar los resultados aplicando el procedimiento con el grupo de test a la arquitectura que haya mostrado mayor precisión.

## 1.3 Enfoque y método seguido

Para la realización de esta investigación se han tenido en cuenta los conocimientos previos del autor. El enfoque y método se muestran en la Tabla 1.

| Áreas con experiencia previa | Áreas sin experiencia previa |
|---|---|
| Algoritmos de clasificación. | Redes neuronales convolucionales. |
| Conocimientos básicos sobre redes neuronales. | Pytorch. |
| | Librería Fast.ai |
| Programación básica en Python. | |

*Tabla 1. Enfoque y método seguido por el trabajo*

**Metodología:**

Se trata de realizar una clasificación sencilla de pacientes sanos y enfermos. Los enfermos están diagnosticados mediante biopsia o mamografía (BI-RADS 4 o mayor).

Para alcanzar el objetivo se elaborará un procedimiento basado en redes neuronales convolucionales, que trate de automatizar la clasificación a partir de las termografías, facilitando la detección precoz del cáncer de mama, permitiendo complementarse con otros métodos de diagnóstico más precisos.

Para la implementación de dicho procedimiento se utiliza la librería de alto nivel Fast.ai [27] que está basada en la librería de PyTorch [29].

Este estudio se realizará sobre un conjunto de datos balanceado construido a partir del banco público de imágenes termográficas de mama que nos ofrece Visual Lab, desarrollado por L. F. Silva et al. [1], combinándolo con un dataset obtenido a partir del historial clínico de los pacientes que nos ofrece dicha web.

Las imágenes se organizan en tres grupos: training, validation y test. Permitiendo de esta forma, contrastar diferentes arquitecturas de redes convolucionales pre-entrenadas que aceleren los cálculos. De esta manera, se puede evaluar su desempeño y analizar la clasificación mediante matrices de confusión.

Este trabajo se implementa en servidores en la nube, en concreto Paperspace [30], que nos facilita servidores especializados en Machine Learning con todas las herramientas y estándares de la industria preinstalados, como Tensorflow, Torch, Caffe y Keras; así como las características necesarias para trabajar con GPUs (Unidades de Procesamiento Gráfico), las cuales resultan idóneas para el trabajo con redes neuronales convolucionales. Todo esto hace que sea un servidor perfectamente adaptado a los requisitos de implementación de las librerías Fast.ai.

La programación se realizará en Python mediante Jupyter Notebooks [31], por resultar especialmente adecuado en la elaboración de documentos de forma interactiva, compaginando la introducción de código con la visualización de resultados y gráficos.

La formación necesaria para la utilización de las librerías de Fast.ai se puede adquirir a través del propio Fast.ai gracias a su curso online sobre Deep Leraning con formato MOOC .

El Pipeline que para desarrollar este trabajo es el siguiente:

- Estudiar las publicaciones sobre investigación con imágenes termográficas.
- Adquirir la formación necesaria en CNN y la librería Fast.ai
- Configurar un servidor en Paperspace e implementar las librerías necesarias.
- Descargar los datos termográficos de Visual Lab.
- Realizar un dataset con la información sobre los pacientes disponible en Visual Lab.
- Convertir los datos termográficos en imágenes y hacer una selección de las imágenes y/o pacientes, limitándose a las tomas frontales, y en la que se tendrá en cuenta las condiciones en que fueron registradas, descartando las que ofrecen poca nitidez, las que muestran un único seno y las de pacientes que antes de la toma hayan utilizado desodorante por interferir en los diagnósticos.
- Dado que el conjunto de imágenes es desbalanceado, se debe aplicar técnicas para equilibrarlo, ya sea sub-muestreando las imágenes, repitiéndolas, tomando en consideración imágenes termográficas dinámica o cualquier otro procedimiento que se considere oportuno.
- Subir las imágenes al servidor en Paperspace.
- Desarrollar un procedimiento para realizar una selección aleatoria y balanceada de las imágenes, organizándolas en tres grupos: training, validation y test.
- Configurar adecuadamente los parámetros e hiperparámetros de las funciones y redes neuronales de la librería de Fast.ai.
- Realizar el entrenamiento, validación y prueba de clasificación con diferentes arquitecturas.
- Contrastar las matrices de confusión y los accuracy.
- Realizar la clasificación utilizando el grupo de test con la arquitectura que haya resultado más precisa.
- Analizar los resultados finales.

## 1.4 Planificación con hitos y temporización

### 1.4.1 Calendario

El desarrollo y temporización de las diferentes tareas que han permitido el desarrollo del Pipeline lo mostramos en el siguiente diagrama de Gantt:

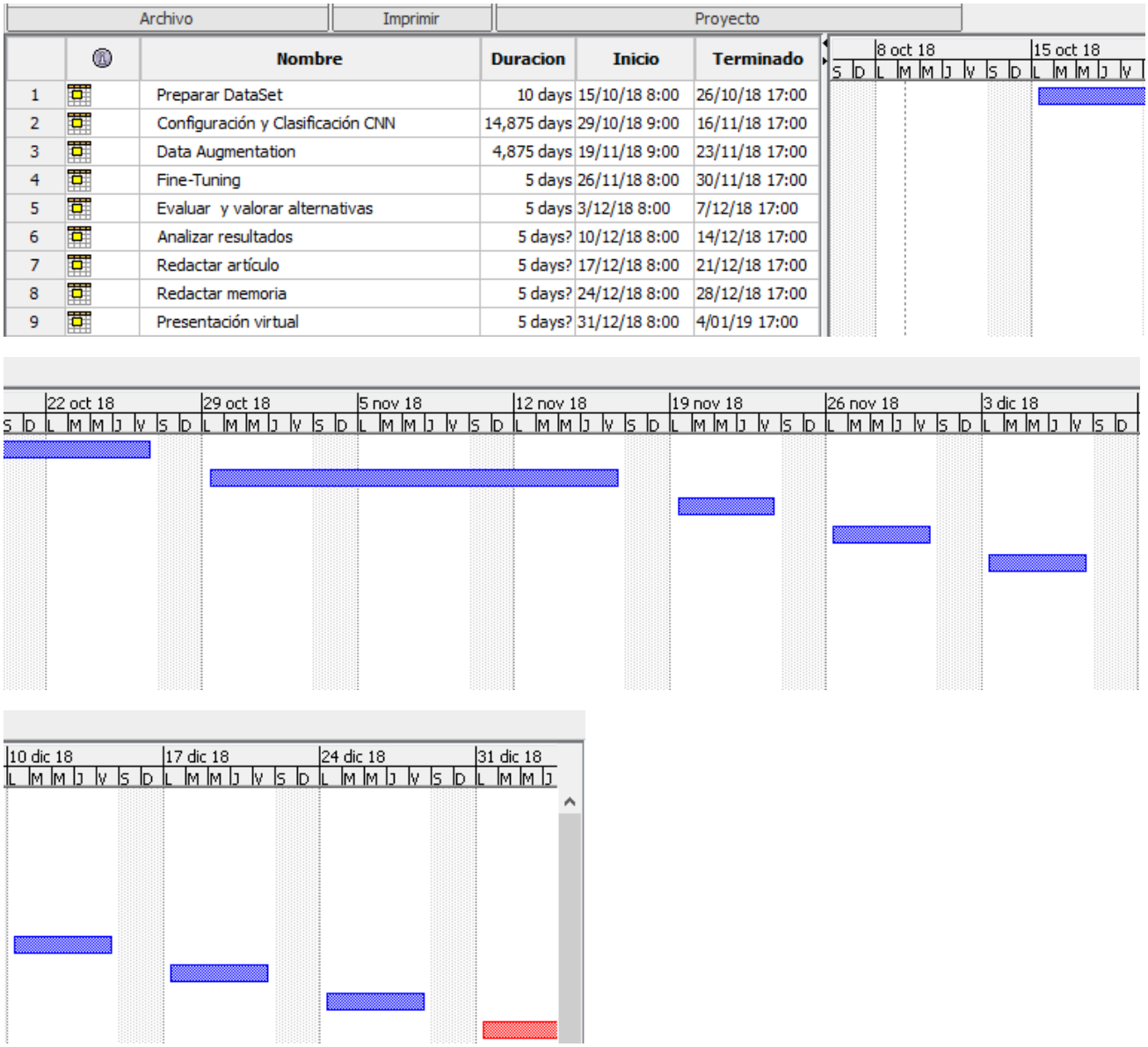



*Figura 1 – Diagrama de Gabntt de las tareas desarrolladas*

### 1.4.2 Tareas

El tiempo del que se dispone es de 300 h repartido de la siguiente manera:

- Definición del plan de trabajo: 60 h.
- Desarrollo del trabajo de investigación: 180 h.
- Elaboración de la memoria, presentación y defensa: 60 h.

Las tareas asociadas a cada objetivo específico se detallan en la Tabla 2, en concordancia con el diagrama temporal anterior:

| Tarea | Fin del plazo |
|---|---|
| **ELABORACIÓN DE UN PROCEDIMIENTO DE CLASIFICACIÓN** | |
| Preparar DataSet (PEC01): | 26 de Octubre |
| - Explorar imágenes termográficas, entendiendo su significado y la patología. | |
| - Elaborar una DataSet con los datos de los pacientes. | |
| - Descargar las imágenes. | |
| Configuración y Clasificación CNN: | 23 de Noviembre |
| - Estudiar los principios de las CNN, conocer el paquete Pytorch, | |
| - Estudio de la librería Fast.ai. | |
| - Configuración del servidor Paperspace | |
| - Realización de una primera clasificación | |
| Aumentar el número de imágenes: | 7 de Diciembre |
| - Evaluar las limitaciones de los resultados previos | |
| - Descargar los datos termográficos (por resultar más cómodo) permitiendo aumentar el número de imágenes disponibles. | |
| - Conversión a imágenes de los datos termográficos, subir al servidor y selección de las apropiadas | |
| - Aplicar técnica de Data Augmentation. | |
| Analizar resultado: | 14 de Diciembre |
| - Analizar resultados y plantear la valoración final mediante otras arquitecturas. | |
| **APLICAR EL PROCEDIMIENTO A DIFERENTES ARQUITECTURAS:** | |
| Contrastar arquitecturas: | 21 de Diciembre |
| - Realización de cálculos con las arquitecturas propuestas. | |
| Redacción de la Memoria: | 31 de Diciembre |
| - Redacción y revisiones de la memoria. | |
| Presentación virtual: | 10 de Enero |
| - Elaboración de presentación. | |
| - Grabación de la presentación. | |

*Tabla 2 - Lista de tareas desarrolladas*

### 1.4.3 Herramientas

Consideramos oportuno referenciar las herramientas que se han utilizado en el desarrollo del trabajo:

Software:

- Microsoft Word para editar los textos.
- ProjectLibre para temporización de los objetivos.
- Python V.3.x incluyendo diferentes librerías bajo sistema Linux.
- Jupyter Notebook para la programación de código.

- Librería Fast.ai para utilizar funciones de alto nivel de redes neuronales convolucionales.
- PyTorch para dar soporte a las librerías Fast.ai.
- OBS para la grabación de la presentación.

Hardware:

- Servidor Paperspace [30] con toda la plataforma de Machine Learning necesaria.

### 1.4.4 Análisis de riesgos

En el desarrollo de todo proyecto es necesario una valoración de los riesgos que puedan afectar negativamente a la consecución del mismo, planificando alternativas ante posibles contingencias, tal como detallamos en los siguientes puntos.

**1. Costos del equipamiento:**

El equipo de trabajo es personal, no requiere ningún tipo de inversión y cualquier problema con el mismo entraría dentro de las previsiones de reposición.

En cuanto al servidor propuesto para desarrollar el trabajo, localizado en la plataforma Paperspace, resulta económico y proporciona los recursos necesarios para este tipo de estudios. Una opción no tan potente y más económica es Kaggle, totalmente gratuito, se planteó utilizarlo como plataforma intermedia, pero finalmente se decidió aprovechar las posibilidades de Paperspace y ganar tiempo trabajando desde el principio con él.

El software implementado en Paperspace es totalmente libre, basado en las librerías Python y fast.ai. Tal como se ha señalado la capacidad de computación que proporciona es apropiada para este tipo de trabajos, disponiendo de GPUs que resultan idóneas para cálculos con CNNs.

**2. Problemas técnicos:**

No se debe obviar los problemas inherentes al trabajo con tecnología, pudiendo resultar de lo más variado: problemas de conexión de red, limitación en la capacidad de cálculo de los procesos, corrupciones del sistema… Todo esto obliga a poner las medidas necesarias para solventarlos con la mayor agilidad posible. Se pondrá especial atención a las convenientes y regulares copias de seguridad, evitando perder el trabajo desarrollado.

Cualquier contingencia de carácter técnico es sensible con los tiempos programados, obligando a ampliarlos con una mayor dedicación, siempre en la medida de las posibilidades.

La mayor dificultad ha sido el acceso a los datos de Vision Lab, tanto de las imágenes como de los datos de los pacientes, además de no ser una conexión fluida tan poco se facilita un descarga grupal, bien del conjunto de imágenes o de una selección de interés.

Por las dificultades señaladas anteriormente, se decidió descargar mediante un sencillo script los datos termográficos (en formato .txt), para acto seguido procesarlo como imágenes utilizando un código Python, facilitando el análisis y procesamientos posteriores.

En cualquier caso, es justo destacar y agradecer la posibilidad de acceder públicamente a las fuentes de Vision Lab, disponiendo así de datos que de otra manera sería difícil de conseguir.

**3. Cumplimiento de calendario:**

A medida que se avanzó en el trabajo, se profundizó en el conocimiento del problema, abriéndose nuevas perspectivas no consideradas, obligando a realizar los ajustes correspondientes, sin perder

de vista los objetivos específicos. Todo esto debe estar previsto en los tiempos de dedicación, permitiéndo encajar los nuevos procedimientos en la planificación temporal propuesta.

**4. Causas externas:**

En lo que respecta a la dedicación, ha sido necesario compaginar el trabajo y la vida personal con el tiempo dedicado a dicho trabajo, imponiéndose para ello una cantidad de horas semanales, sin escatimar en hacerlo fines de semana, festivos y periodos vacacionales para poder cumplir con la carga programada.

Otro riesgo eran posibles enfermedades que nos obligarían nuevamente a aumentar los plazos, por suerte no se dio el caso.

## 1.5 Breve resumen de los productos obtenidos

Los productos resultantes de este trabajo son:

- Una memoria en la que se detallan todos los pasos, procesos y resultados obtenidos durante el desarrollo del trabajo
- Se incluyen los códigos desarrollados para facilitar la reproducción de los resultados.

## 1.6 Estructuración del trabajo

A continuación, se detallan los diferentes capítulos en los que se ha estructurado el trabajo:

**Capítulo 1. Introducción:** se describe y justifica el trabajo, indicando los objetivos y tareas asociadas, con un calendario que detalla la programación temporal.

**Capítulo 2. Conjunto de datos:** se muestran los datos con los que se trabaja, obtenidos de Vision Lab, correspondientes a las termografías y los datos registrados de cada paciente.

**Capítulo 3. Red neuronal y red neuronal convolucional:** se hace un repaso a los principios sobre los que se sustenta la tecnología empleada en este trabajo.

**Capítulo 4. Desarrollo del trabajo:** se describen los diferentes pasos implementados sobre Jupyter Notebook para aplicar las librerías de Fast.ai al objetivo del trabajo, la clasificación de pacientes enfermos y sanos a partir de sus termografías. Se aplica el procedimiento a varias arquitecturas y a continuación sobre la que se ha mostrado más precisa, para hacer una evaluación y discusión final de los resultados.

**Capítulo 5. Conclusión:** valoración personal como resultado del desarrollo del trabajo.

**Capítulo 6. Referencias bibliográficas:** incluyendo artículos, webs de referencia.

**Capítulo 7. Anexos:** incluye un glosario y el conjunto de códigos utilizados.

# 2. Conjunto de datos

## 2.1 Origen de los datos

Se trata de un conjunto de 5.604 ficheros de datos termográficos descargados del servidor público Vision Lab, procesados posteriormente para obtener las imágenes infrarrojas que los representan. En la Figura 2 tenemos una muestra de las mismas.

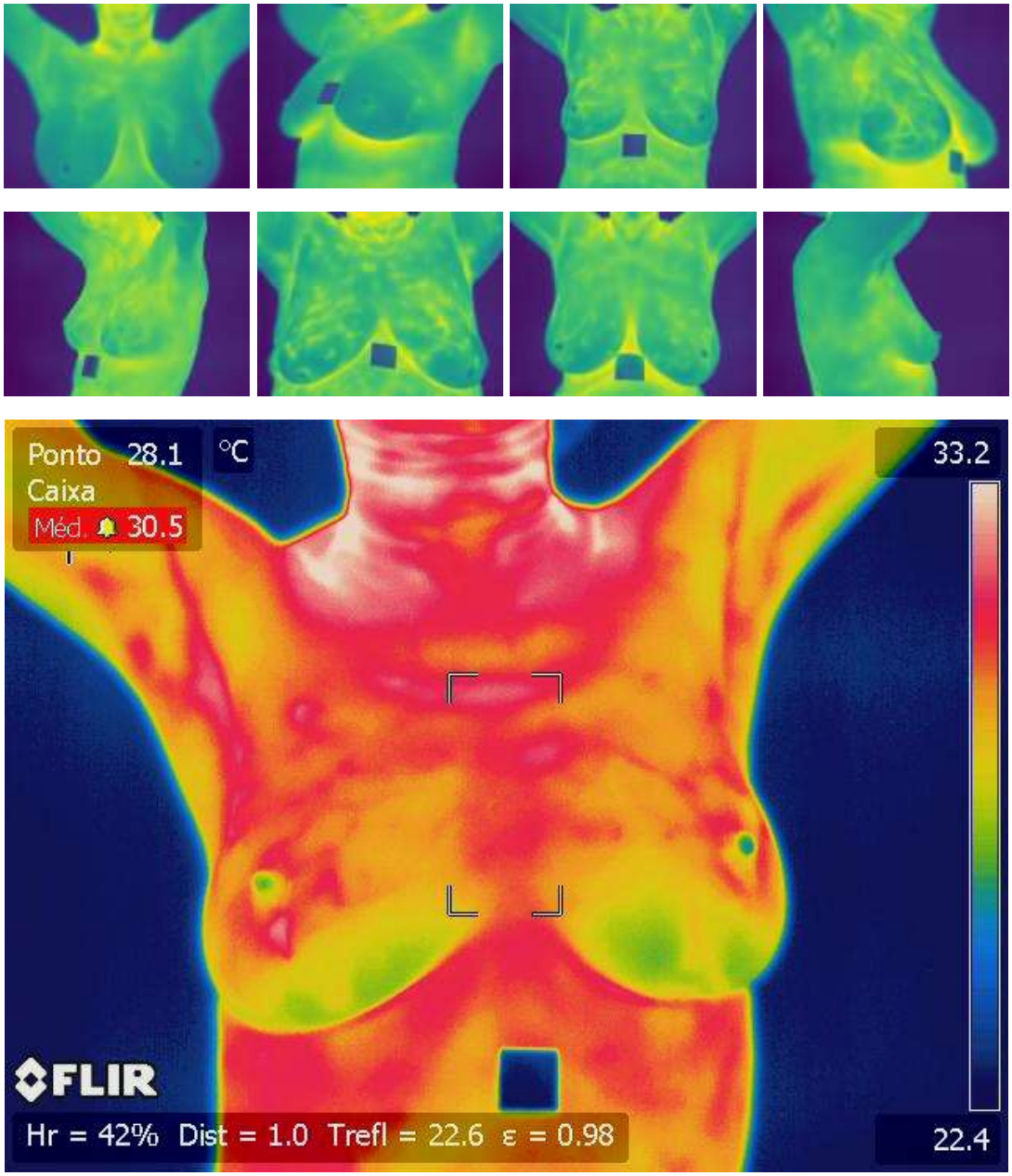



*Figura 2 - Muestra de termografías utilizadas en el estudio*

## 2.2 Exploración del conjunto de datos

El conjunto de imágenes con las que se trabaja tienen un tamaño de 640×480 píxeles, obtenidas a partir de los datos termográficos. Cada termografía infrarrojas de mama es un mapa de calor de 640x480 = 307.200 temperaturas, que se ha normalizado para poder representarlas tal como se muestra en la Figura 2. Es importante tener presente que las imágenes termográficas son una representación en forma de imagen de los datos termográficos.

Conviene aclarar las diferencias de color en dicha figura, las ocho primeras muestran una selección de las utilizadas en el estudio una vez ajustadas, contrastando con los colores de la termografía que les acompaña, descargada directamente del servidor de Vision Lab, lo cual se explica ya que las escalas utilizadas son diferentes, en nuestro caso las temperaturas más elevadas tienen un tono verde mientras que en el de Vision Lab son rojas.

Para cada paciente se han obtenido diferentes tipos de tomas, tanto dinámicas (secuencia temporal de tomas) como estáticas (una única toma). A continuación y a modo de ejemplo se indica la terminología empleada en uno de los archivos, facilitándonos su comprensión. La cadena que identifica nuestro ejemplo sería T0287.1.1.D.2015-07-20.18.txt, y el significado de sus términos el siguiente:

**T0:** todas comienzan con esta referencia.

**287:** es el número de paciente.

**1.1.:** se refiere a vista frontal, si fuese 1.2 sería “right 45º”, 1.3 “right 90º”, 1.4 “left 45º”, 1.5 “left 90º”

**D:** es “dynamic”, es decir la termografía dinámica (la mayoría son frontales), es una secuencia de imágenes temporales, con S se refiere a “static”.

**2015-07-20.18:** fecha y hora tomada.

A su vez, para este trabajo se ha creado una tabla con la información de cada paciente, incluyendo su diagnóstico, lo que facilita la organización de los datos.

Posteriormente, con las imágenes ya obtenidas en su totalidad, se realiza una preselección de las mismas. Se limita el estudio a las tomas frontales, se descartan las imágenes termográficas de las pacientes que aplicaron desodorante (por afectar en gran medida a los diagnósticos), también se omiten las que presentan un solo seno (debido a extirpación) y aquellas que se muestran con poca nitidez. La selección resultante se organizó en dos grupos: Healthy con 2.411 imágenes y Sick con 534.

El siguiente paso fue realizar una selección aleatoria balanceada de las imágenes con 500 para Sick y 500 para Healthy. Se organizaron en tres carpetas: training, validation y test, incluyendo en cada una de ella dos carpetas Sick y Healthy, organizando así la selección aleatoria. Cumpliendo con los requisitos de estructura que necesita Fast.ai para aplicar sus funciones.

# 3. Red neuronal artificial y red neuronal convolucional

En este capítulo se realiza una introducción al marco conceptual y tecnológico sobre el que se aborda este trabajo para tener una mayor claridad de cómo afrontarlo. En las siguientes líneas se sintetiza la base de conocimiento adquirido durante la formación.

## 3.1 Redes Neuronales

La primera referencia a redes neuronales se hace en el artículo de McCulloch y Pitts [14], donde se plantean dos enfoques, uno centrado en el proceso biológico del cerebro y otro en las aplicaciones en Inteligencia Artificial. Trabajo al que había precedido unos años antes la hipótesis de aprendizaje de Hebb [15], basada en la plasticidad neuronal. La primera creación de una red de Hebb la realizan Farley y Wesley A. Clark en el MIT en 1954.

Posteriormente, en 1958 Rosenblatt [16] crea el perceptrón, un algoritmo de reconocimiento de patrones en una red de aprendizaje con dos capas, pero incapaz de crear un circuito de o-exclusiva, hasta la propuesta de Paul Werbos en 1975 que citaremos posteriormente.

La primera publicación sobre redes multicapa corresponde a Ivakhnenko y Lapa en 1965 [17], el denominado “método de agrupamiento para el manejo de datos”.

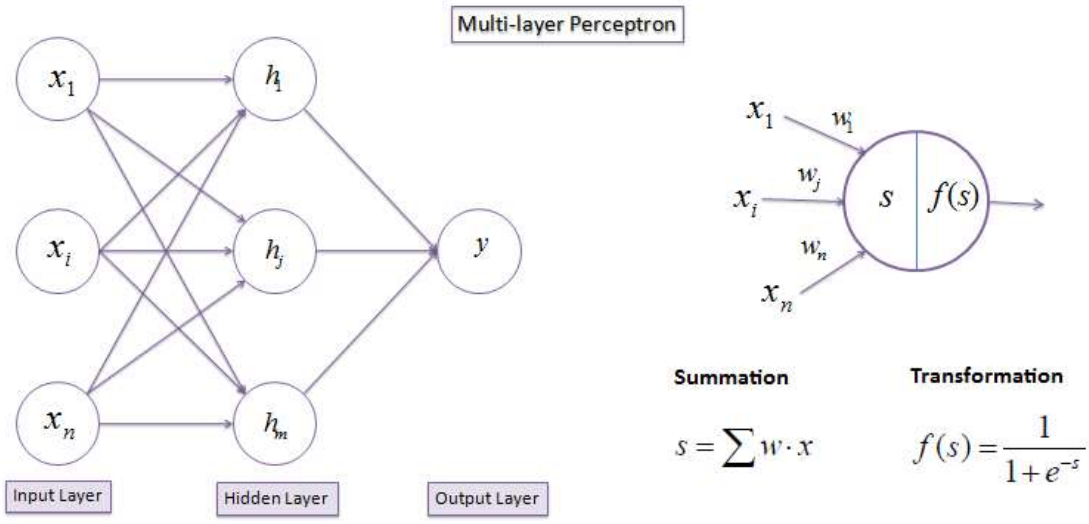



*Figura 3 – Perceptrón simple y multicapa (Imagen obtenida de saedsayad.com [32]*

El estudio de las redes neuronales artificiales se estancó en 1969 con la publicación de Marvin Minsky y Seymour Papert [18], donde se pone de relieve dos cuestiones fundamentales en los desarrollos con redes neuronales, la primera era la incapacidad de los perceptrones para procesar el o-exclusivo ya comentado, y la segunda la limitada capacidad de los ordenadores para atender los requisitos de procesamiento de las redes neuronales.

Sin embargo, en 1975 Paul Werbos [19] resuelve el problema del o-exclusivo con el algoritmo de propagación hacia atrás, en el que se propone evaluar la diferencia entre el resultado obtenido y el resultado deseado, con el objetivo de cambiar los “pesos” de las redes neuronales. Si a esto sumamos la mejora en el procesamiento de los ordenadores, nos encontraos con un clima idóneo para el renacimiento de las redes neuronales.

En las siguientes líneas se profundiza un poco más en los aspectos técnicos de lo que debe ser una red neuronal.

El elemento clave de una red neuronal artificial es la neurona, una unidad de procesamiento que tiene conexiones con otras neuronas, algunas de estas conexiones son estímulos de entrada, sobre los que hace un cálculo interno para obtener un valor de salida, equivalente a lo que conocemos como función matemática.

Esa función interna no es más que la suma ponderada de las entradas, con un peso asignado a cada una de ellas (ver Figura 3), muy similar a una regresión lineal. En analogía con esta, la función también dispone de un parámetro independiente conocido como sesgo o bias, que se puede asimilar como una entrada más, vinculada al valor uno en este caso. La aplicación de este principio a funciones binarias equivale gráficamente a dividir el espacio mediante una recta que separa las dos clases.

Las neuronas se pueden organizar de diferentes maneras, por ejemplo, en capas, cada una recibe información de todas las neuronas de la capa anterior, y aportan información a todas las neuronas de la capa siguiente.

Esta arquitectura facilita la adquisición de conocimiento jerarquizado. Las primeras capas elaboran un conocimiento especializado y en las posteriores se va produciendo un conocimiento más abstracto. Cuantas más capas más complejo se hace el conocimiento. Organización que da el nombre al Deep Learning.

Matemáticamente se está concatenando/sumando varias operaciones de regresión lineal, lo que da como resultado final una regresión lineal, que equivale a una única neurona. Para evitar esto, se necesita que resulte algo diferente a una línea recta, apareciendo el concepto de función de activación para alcanzar dicho objetivo.

La función de activación interviene sobre la suma ponderada en cada una de las neuronas, distorsionando el valor de salida, añadiéndole deformaciones no lineales, permitiendo así concatenar varias neuronas.

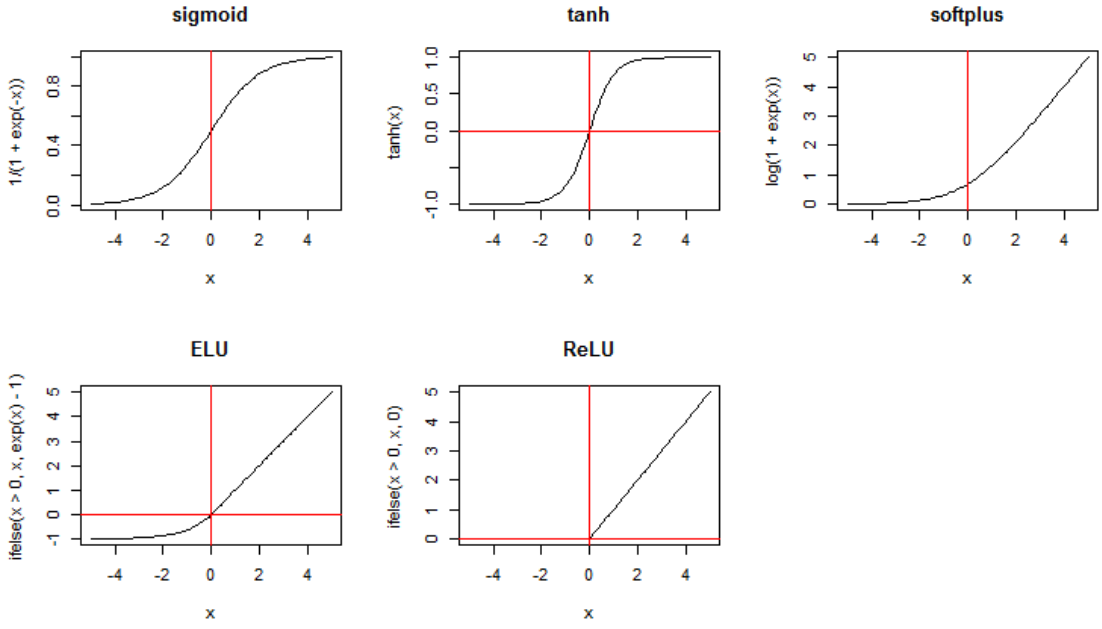



*Figura 4 - Funciones de activación de las neuronas (imagen obtenida de mql5 [33])*

Algunos ejemplos de función de activación son: función escalonada, sigmoide, tangente hiperbólica, unidad rectificada lineal (relu). Cada una de estas funciones además de la no linealidad aportan otra serie de ventajas. Combinando en una capa diferentes tipos de neuronas con sus

propias funciones de activación conseguiremos hacer aquellas clasificaciones que nos resulten interesantes.

En 1986 el algoritmo que se venía utilizando para entrenar el perceptrón no era aplicable a redes neuronales, apareciendo el artículo de Rumelhart, Hinton y Williams [23], para introducir el algoritmo del descenso del gradiente o backpropagation, mostrando como una red neuronal ajusta sus parámetros, logrando de esta manera aprender.

El descenso del gradiente es una estrategia para obtener los parámetros de una regresión lineal, evaluando el error del modelo en un punto. Calculando las derivadas parciales en dicho punto se obtiene un vector, que indica la pendiente de la función, hacia donde se incrementa el error. Tomando la dirección contraria se consigue ir reduciendo el error, equivalente a descender por una montaña. Así se evalúa hacia atrás el error de cada parámetro, minimizando el error en su conjunto y logrando el entrenamiento de la red.

En una red neuronal profunda el número de capas es dos o tres, en una red neuronal profunda puede superar los 150. Otra característica es el gran número de datos etiquetados se necesitan para entrenar la red.

En 1992 se introduce el max-pooling, técnica basada en el submuestreo que dividen los datos en grupos de un tamaño determinado, transmitiendo solamente el valor máximo, técnica que ayuda al reconocimiento de objetos tridimensionales. Posteriormente en 2010 la utilización de GPUs permitieron acelerar el entrenamiento mediante backpropagation, alcanzando así un rendimiento alto.

Para completar esta revisión a la génesis, es necesario tener presentes los tres paradigmas de aprendizaje existentes:

- Aprendizaje supervisado: partiendo de los datos de entrenamiento se crea una función capaz de predecir la salida a partir de cualquier entrada válida.
- Aprendizaje no supervisado: el modelo se ajusta a las observaciones, se diferencia del supervisado en no disponer de un conocimiento a priori.
- Aprendizaje por refuerzo: el algoritmo de aprendizaje recibe algún tipo de valoración sobre la adecuación de la respuesta.

Algunos aspectos críticos de las redes neuronales que es necesario indicar serían los siguientes:

- Requieren diversidad de entrenamiento.
- No reflejan aprendizaje neurona real, la propagación hacia atrás es su base, mecanismo que no se da en las neuronas naturales.
- Requiere un alto cálculo de procesamiento.

## 3.2 Redes Neuronales Convolucionales

Las redes neuronales convolucionales se basan en el Neocognitrón de Kunihiko Fukushia et al. [20], mejorado posteriormente por LeCun, Yann et al. [21] introduciendo el método de aprendizaje de propagación hacia atrás, logrando entrenar el sistema de forma adecuada. En 2012 fueron perfeccionadas por Ciseran, Dan et al. [22]**,** y fueron implementadas para una GPU, consiguiendo resultados sorprendentes.

La arquitectura de las CNNs está orientada principalmente a la visión artificial. En esencia una red CNN convoluciona las características aprendidas con los datos de entrada, empleando capas convolucionales en 2D. Esto las hace muy adecuadas para trabajar con datos 2D, como es el caso de las imágenes.

La compleja estructura de capas ocultas de las CNNs permiten extraer las características necesarias para la clasificación de imágenes sin necesidad de identificación previa. Cada una de las capas va aumentando la complejidad de las características detectadas: bordes, formas, etc.

Conviene hacer la diferencia entre Machine Learning y Deep Learning. En el Machine Learning se seleccionan previamente las características y el clasificador, en aprendizaje profundo la extracción de características y la modelización son automáticas. En Machine Learning se llega a una convergencia a medida que aumenta la fuente de datos, mientras que en Deep Learning se va mejorando el aprendizaje.

Por último, cabe destacar tres formas de realizar aprendizaje profundo:

- Entrenamiento desde cero: partiendo de un conjunto elevado de datos adecuadamente etiquetados, se diseña la arquitectura que aprenda las características y el modelo. Se tarda tiempo en entrenar estas redes, días e incluso semanas.
- Transferencia de aprendizaje: se parte de un modelo y se proporcionan nuevos datos, tras los ajustes oportunos se logra gestionar una tarea nueva. Requiere menos datos y tiempo.
- Extracción de características: es el menos habitual, consistiendo en extraer las características identificadas por alguna de las capas, pudiendo utilizarlas para un proceso de Machine Learning.

La aceleración mediante GPUs reduce considerablemente los tiempos de aprendizaje.

Un concepto a tener presente en el estudio de CNNs es el de arquitectura, entendiendo por tal a la topología, estructura o patrón de conexionado que esta tiene. Son varias las arquitecturas de CNN que se han desarrollado mejorando sucesivamente los resultados de clasificación, por citar sólo algunas destacaríamos AlexNet, VGG, GoogLeNet y ResNet; cada una de ellas con particulares técnicas que continuamente se van innovando.

## 3.3 Framewoks de deep learning

Python y Redes Neuronales convolucionales se conjugan en TensorFlow, se trata de una biblioteca open source desarrollada por Google para trabajar con tensores apoyándose en diagramas de flujo o grafos, en los que los nodos representan operaciones matemáticas y los arcos tensores. Su arquitectura facilita la realización de cálculos en varias CPUs o GPUs.

Los nodos se identifican con los dispositivos computacionales y se ejecutan de forma asíncrona y en paralelo, una vez todos los tensores están disponibles.

En definitiva, un grafo de TensorFlow es una descripción de cálculos, y debe ser lanzado dentro de una sesión. Cada sesión coloca las operaciones del grafo en los diferentes dispositivos, ya sean CPU o GPU.

TensorFlow nos otorga una gran flexibilidad para trabajar con redes neuronales, y en concreto con redes neuronales convolucionales. Como parte de TensorFlow tenemos TensorBoard que nos permite visualizar los grafos facilitando el entendimiento del flujo de cálculo de nuestros modelos.

Una alternativa a TensorFlow es Keras, una librería de alto nivel que utiliza otras como TensorFlow, CNTK y Theano, facilitando mucho el trabajo con redes neuronales.

Otra opción es la librería Pytorch, que conjuga la facilidad de Keras con la potencia de TensorFlow, también destinado al cálculo numérico utilizando programación de tensores. Su gran ventaja es la sencillez para trabajar con redes neuronales y la posibilidad de trabajar con grafos dinámicos en

vez de estáticos, lo que permite modificar las funciones, con lo que variaría el cálculo del gradiente, facilitando la depuración de código.

Pytorch tiene soporte para trabajar sobre GPUs utilizando la API CUDA, que conecta la CPU con la GPU, gracias a un desarrollo de NVIDIA.

## 3.4 Fast.ai

Es necesario destacar la labor de Kaggle [26], una comunidad de científicos de datos creada por Google Inc., que supera el millón de colaboradores compartiendo formación, conjuntos de datos y modelos, incentivando el desarrollo mediante competiciones para resolver desafíos entorno a la ciencia de datos.

Son muchos los proyectos de éxito que han surgido de la iniciativa de Kaggle, en lo que concierne a nuestro trabajo destacaríamos **Fast.ai [27]**, una iniciativa de Rachel Thomas y Jeremy Howard, cuyo objetivo principal es terminar con la idea de que la Inteligencia Artificial es complicada y requiere de grandes expertos, siendo su pretensión acercarla al gran público para resolver problemas variados. Como dice la propia Rachel, “Quiero que usted se una a la fuerza de trabajo de la inteligencia artificial (IA) y tengo un plan para conseguirlo”, terminando así con los estereotipos de expertos, grandes cantidades de datos y equipos potentes [24]. Para lograr estos objetivos imparten formación gratuita a través de su MOOC [28].

Con los principios de Fast.ai hemos abordado nuestro trabajo, aprovechando la potencia de su librería, que ha conseguido entre otros éxitos vencer en las competiciones de Kaggle.

# 4. Desarrollo del trabajo

Como se ha indicado en secciones anteriores, para el entrenamiento de las redes neuronales artificiales profundas se ha de contar con un gran número de imágenes catalogadas y equipos con alta capacidad de procesamiento. La comunidad colabora intercambiando los pesos de arquitecturas entrenadas previamente para aprovecharlas en investigaciones más específicas. Los pesos de las capas más profundas contienen descriptores más específicos que se van especializando en función del conjunto de datos con el que se va trabajando. Las funciones de alto nivel de Fast.ai facilitan todos los cálculos necesarios para lograr excelentes rendimientos en la construcción de un modelo.

A continuación se detalla los pasos del procedimiento desarrollado en este trabajo, utilizando la arquitectura restnet34, que goza de un gran reconocimiento. Se recurrirá a una selección aleatoria de las termografías de mama. Posteriormente se repetirá el mismo procedimiento con otras arquitecturas, permitiendo contrastar los resultados. Finalmente se aplicará al grupo test la arquitectura que haya resultado más precisa.

## 4.1 Configuración y preparación de datos

El primer paso a realizar es descargar las imágenes Vision Lab, al ser un número reducido (175 Healthy y 41 Sick), se decidió descargar las termografías de mama (varias tomas, tanto estáticas como dinámicas) resultando un número superior, en total 5.604 imágenes térmicas.

En paralelo se realizó una tabla en el que se registraron datos de los pacientes tal como figuraban en Vision Lab.

Tras diferenciar los archivos Healthy de los Sick se convirtieron a imágenes por dos motivos fundamentales: el primero ver sus características y el segundo para poder aplicar las funciones de Fast.ai tal como se propone en su Mooc.

Para la conversión se utilizó un sencillo procedimiento escrito en Python, teniendo en cuenta que el tamaño de las imágenes es de 480x640:

```
import os
os.chdir(PATH)
files = os.listdir()
for f in files:
	fichero = open(“txt/”+f)
	datos = fichero.read()
	datos = datos.split()
	imagen = np.array(datos)
	imagen.shape = (480, 640)
	imagen = imagen.astype(float)
	imagen = imagen/40
	plt.imsave(“imágenes/”+f[:4]+”jpg”, imagen)
```

*Código 1- Conversión de datos termográficos a imágenes*

Tras convertir los datos termográficos a imágenes se realiza un filtrado para seleccionar las que resultaban adecuadas para el estudio. En primer lugar, se seleccionan las que corresponden a tomas frontales, tanto las estáticas como las dinámicas, limitándose a las que muestran dos senos y eliminando todas aquellas de pacientes que han utilizado desodorante por interferir en las predicciones. Finalmente se omiten las que se registraron con poca nitidez.

Como resultado se obtienen 2.411 imágenes Healthy y 534 Sick, procediendo a subirlas al servidor de Paperspace, donde se organizan en dos carpetas, Healthy y Sick. A continuación, se programó

un código en Python, el cual permite hacer una selección aleatoria balanceada de 500 imágenes Healthy y 500 Sick, destinando 400 a training y 100 a validación en cada grupo, para una primera valoración y desarrollo del procedimiento de referencia.

Señalar que este código ofrece la posibilidad de incluir un tercer grupo para test. Posteriormente el procedimiento se aplicará a diferentes arquitecturas, en ese momento las termografías se organizaron en tres carpetas, una para test en la que se reserva el 20%, otra para validación con un 20% del 80% restante, y finalmente el resto en una para training; en números redondos serían 320 para training, 80 para validación y 100 para test; esto para cada grupo Healthy y Sick.

```
Import os, sutil, random

#random.seed(5)

PATH_IMAGENES = "../../../datajavi/TotalImagenes"
PATH_DESTINO = "../../../datajavi/SeleccionAleatoria"
cantidad = 500
cantidad_test = 0
cantidad_valid = 100

# Funciones para seleccionar:

def selecciona_imagens(PATH_IMAGENES, PATH_DESTINO, cantidad):
    lista_imagenes = os.listdir(PATH_IMAGENES)
    seleccion_lista_imagenes = random.sample(lista_imagens, cantidad)
    lista_destino_a_borrar = os.listdir(PATH_DESTINO)
    for imagen in lista_destino_a_borrar:
            os.remove(PATH_DESTINO + "/" + imagen)
    for imagen in selección_lista_imagenes:
            sutil.copy(PATH_IMAGENES + "/" + imagen, PATH_DESTINO)

def mover_imagenes(PATH_IMAGENES, PATH_DESTINO, cantidad):
    lista_imagenes = os.listdir(PATH_IMAGENES)
    seleccion_lista_imagenes = random.sample(lista_imagenes, cantidad)
    lista_destino_a_borrar = os.listdir(PATH_DESTINO)
    for imagen in lista_destino_a_borrar:
            os.remove(PATH_DESTINO + "/" + imagen)
    for imagen in selección_lista_imagenes:
            sutil.move(PATH_IMAGENES + "/" + imagen, PATH_DESTINO)

selecciona_imagenes(PATH_IMAGENES + "Healthy", PATH_DESTINO + "/train/Healthy", cantidad)
selecciona_imagenes(PATH_IMAGENES + "Sick", PATH_DESTINO + "/train/Sick", cantidad)
mover_imagenes(PATH_DESTINO + "/train/Healthy", PATH_DESTINO + "test/Healthy", cantidad_test)
mover_imagenes(PATH_DESTINO + "/train/Sick", PATH_DESTINO + "test/Sick", cantidad_test)
mover_imagenes(PATH_DESTINO + "/train/Healthy", PATH_DESTINO + "valid/Healthy", cantidad_valid)
mover_imagenes(PATH_DESTINO + "/train/Sick", PATH_DESTINO + "valid/Sick", cantidad_valid)
```

*Código 2 - Organización de las imágenes en training, validation y test*

Las funciones de alto nivel de Fast.ai requieren organizar las imágenes en dos directorios train y valid, y también que en cada uno de ellos haya directorios con cada una de las clases en las que queremos organizar las imágenes (Healthy y Sick). Siguiendo estas indicaciones se estructura la información en carpetas. Los dos grupos Healthy y Sick se reparten en las proporciones indicadas dentro de las carpetas train y valid respectivamente.

Con las imágenes ya organizadas estamos en disposición de aplicar las funciones de Fast.ai. Lo primero es configurar adecuadamente los parámetros necesarios: recargas automáticas, permitir

el grafismo en línea en los Jupyter Notebook, indicar el path donde tenemos localizadas las imágenes (donde también se almacenarán datos temporales y resultados finales), el tamaño de ajuste de imagen para lograr un entrenamiento rápido e importar las librerías necesarias para ejecutar el código.

```
# Configuramos los parámetros para reLoading automático
# plotting en línea

% reload_ext autoreload
% autoreload 2
% matplotlib inline

# Cargamos las librerías necesarias

from fastai.imports import *
from fastai.conv_learner import *
from fastai.transforms import *
from fastai.model import *
from fastai.dataset import *
from fastai.sgdr import *
from fastai.plots import *

# Indicamos el path y el tamaño al que se ajustarán las imágenes garantizando un
entrentamiento rápido.

PATH = "../../../datajavi/SeleccionAleatoria/"
sz = 224

# Verificar la disposición de CUDA que nos permite aprovechar el procesamiento GPU

torch.cuda.is_available()
```

*Código 3- Configuración del sistema, importación de librerías*

## 4.2 Revisión de la organización de los datos:

Para garantizar la correcta organización de las imágenes en la estructura requerida por las funciones Fast.ai, es recomendable hacer una sencilla revisión inicial, en la que se compruebe la existencia de las carpetas train y valid, incluyendo los subdirectorios de las clases en las que queremos organizar las pacientes (Sick y Healthy). Tal como refleja los siguientes comandos en línea, y sus resultados a modo de ejemplo:

```
os.listdir(PATH)
```

['test', 'valid', 'models', 'train', 'tmp']

```
os.listdir('f '{path}valid')
```

['Healthy', 'Sick']

*Código 4 - Verificación de organización y disponibilidad de datos*

## 4.3 Creación del modelo:

Como punto de partida se utilizó un modelo de red neuronal convolucional pre-entrenado en ImageNet (1.2 millones de imágenes y 1000 clases), el ya comentado resnet34.

La forma de aplicar las librerías de alto nivel de Fast.ai es la siguiente:

```
arch = resnet34
data = ImageClassifierData.frompaths(PATH, tfms = tfms_from_model(arch, sz), bs = 8)
learn = ConvLearner.pretarained(arch, data, precompute = True)
learn.fit(0.01, 2)
```

*Código 5 - Cálculo del primer modelo con arquitectura resnet34*

Tras indicar la arquitectura, se configuran los datos y el objeto de aprendizaje, tal como muestra el código 5. Por un lado tenemos **ImageClassifierData.frompaths**, que lee datos de la dirección que se le pasa como primer parámetro, creando un conjunto de datos listos para el proceso de entrenamiento. Por su parte **tfms** se encarga del cambio de tamaño, el recorte y la normalización de las imágenes. A su vez **ConvLearner.pretarained** construye el modelo de aprendizaje a partir de un modelo pre-entrenado, en este caso el citado resnet34.

Es el momento de aplicar **learn.fit**, función que realiza los cálculos de clasificación y en la que se deben indicar dos hiperparámetros, el primero es el rango de aprendizaje y segundo al número de épocas. Conviene señalar que los hiperparámetos son propiedades del modelo que no se adquieren en el proceso de aprendizaje. El rango de aprendizaje determina la rapidez o la lentitud con la que se desea actualizar los pesos de las CNNs, es uno de los parámetros más difíciles de establecer y con el que hay que ser especialmente minucioso, ya que afecta significativamente al resultado y al rendimiento del modelo. Por su parte las épocas son los números de ciclos en los que se repiten los cálculos sobre todos los elementos a clasificar.

Con un número reducido de líneas y muy poco tiempo de cálculo, las funciones de Fast.ai realizan una clasificación y dan un accuracy :

| Epoch | trn_loss | val_loss | accuracy |
|---|---|---|---|
| 0 | 0.528624 | 0.221165 | 0.925 |
| 1 | 0.460429 | 0.200873 | 0.94 |

*Código 6 - Resultados del cálculo para el primer modelo*

Tal como se ve, la precisión alcanzada sobre el grupo de validación es un prometedor 94%.

**Apunte sobre la tasa de aprendizaje:**

Indicar que para ayudar a determinar la tasa de aprendizaje óptima se dispone del método learn.lr_find(), detallado en el artículo Cyclical Learning Rates for Training Neural Networks [12], donde se propone aumentar la velocidad de aprendizaje desde un valor muy pequeño hasta que la pérdida deje de disminuir.

```
learn = ConvLearner.pretrained(arch, data, precompute = True)
lrf = learn.lr_find()
```

*Código 7 - Preparación para cálculo de la tasa de aprendizaje*

El objeto learn dispone de un atributo sched, que a su vez incluye diferentes funciones de trazado y un programador que ayudan visualmente a la determinación de la tasa de aprendizaje. En la Figura 4 se ve la pérdida en función de la tasa de aprendizaje, debiendo fijarnos en la zona en la que deja de disminuir, en ese punto se redondea a la potencia de 10 más próxima, en este caso 1e-2, estableciendo en ese valor la tasa de aprendizaje:

```
learn.sched.plot()
```

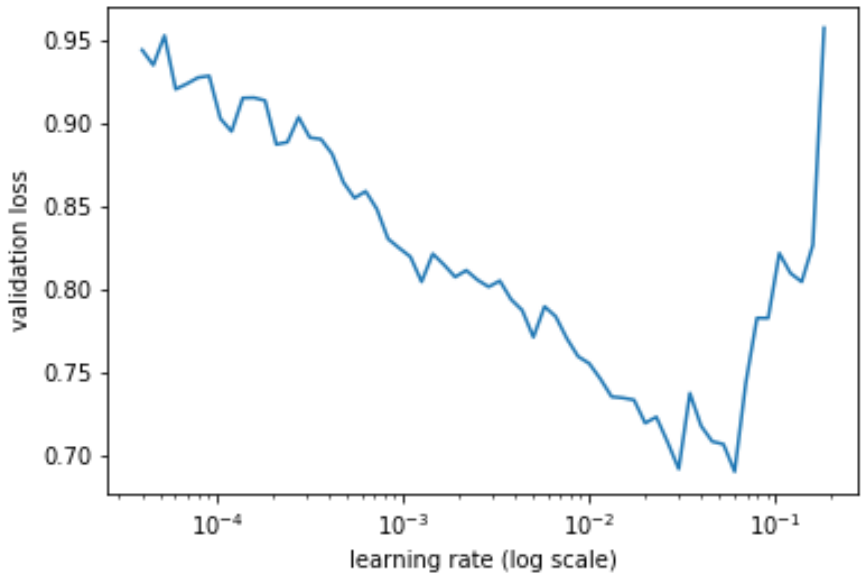



*Figura 5 - Pérdida en función de la tasa de aprendizaje*

## 4.4 Analizando resultados

Antes de seguir avanzando y afinando el procedimiento, es recomendable realizar un análisis que nos permita valorar los primeros resultado obtenidos, basándose en una visualización de las siguientes estimaciones:

1. Obtener una selección aleatoria de imágenes correctamente etiquetadas.
2. Obtener una selección aleatoria de imágenes incorrectamente etiquetadas.
3. Obtener las imágenes etiquetadas con mejor precisión en cada clase.
4. Obtener las imágenes etiquetadas con peor precisión en cada clase.
5. Obtener las imágenes etiquetadas con mayor incertidumbre, que son aquellas con una probabilidad próxima a 0.5.

Las predicciones para el conjunto de validación en escala logarítmica mediante el siguiente comando:

```
log_preds = learn.predict()
```

*Código 8 - Predicciones para el conjunto de validación*

En el objeto log_preds se obtiene una serie de probabilidades, cuanto más próximas se encuentran a 0 mayor probabilidad tienen de ser Sick, y cuanto más próximas se encuentran a 1 mayor probabilidad tienen de ser Healthy. Valores en torno a 0.5 dan mayor grado de incertidumbre.

Para la realización del análisis propuesto anteriormente Fat.ai se facilita una serie de funciones que debemos incorporar al procedimiento:

```
preds = np.argmax(log_preds, axis = 1)
probs = np.exp(log_preds[:,1])

def rand_by_correct(is_correct): return np.random.choice(np.where(mask)[0], min(len(preds), 4), replace =
False)

def rand_by_correct(is_correct): return rand_by_mask((preds == data.val_y) == is_correct)

def plots(ims, figsize = (12,6), rows = 1, titles = None):
```

```
        f = plt.figure(figsize)
        for i in rango(len,(ims)):
        sp = f.add_subplot(rows, len(ims)//rows, i+1)
        sp.axis('Iff')
        if titles is not None: sp.set_title(titles[i], fontsize = 16)
        plt.imshow(ims[i])

def load_img_id(ds, idx): return np.array(PIL.Image.open(PATH+ds.fnames[idx]))

def plot_val_with_title(idxs, title):
        imgs = [load_img_id(data.val_ds, x) for x in idxs]
        title_probs = [probs[x] for x in idxs]
        print(title)
        return plots(imgs, rows = 1, titles = title_probs, figsize = (16, 8)) if len(imgs) > 0 else print('Not
Found.')

def most_bymask(mask, mult):
        idxs = np.where(mask)[0]
        return idxs[np.argsort(mult * probs[idxs])[:4]]

def most_by_correct(y, is_correct):
        mult = -1 if (y == 1) == is correct else 1
        return most_by_mask(((preds == data.val_y) == is_correct) & (data.val_y == y), mult)
```

*Código 9 - Preparación para mostrar resultados*

Ya se está en disposición de realizar el análisis aplicando las funciones correspondientes, comenzando por las correctamente clasificadas:

```
plot_val_with_title(rand_by_correct(True)), "Correctly classified")
```

Correctly classified

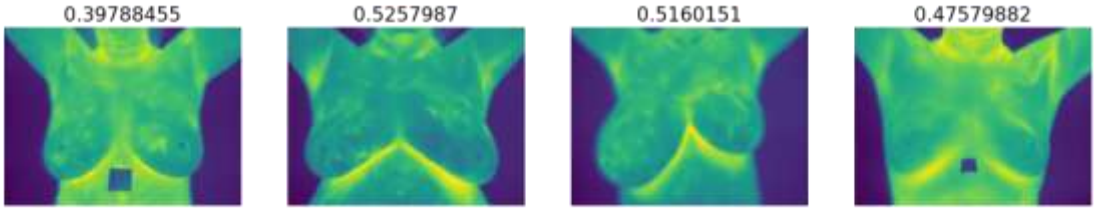



*Figura 6 - Imágenes correctamente clasificadas*

Y ahora las incorrectamente clasificadas:

```
plot_val_with_title(rand_by_correct(False)), "Incorrectly classified")
```

Incorrectly classified

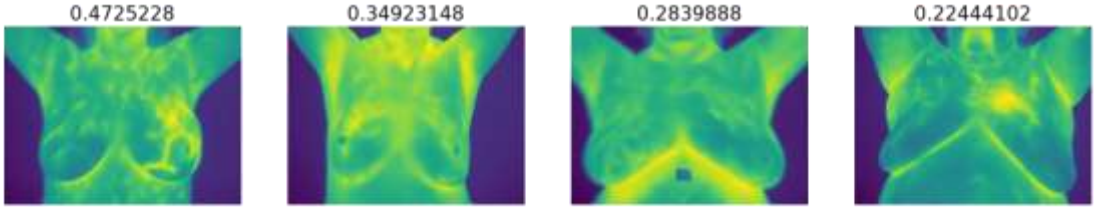



*Figura 7 - Imágenes incorrectamente clasificadas*

Continuando con el análisis obtenemos los diferentes grupos:

```
plot_val_with_title(most_by_correct(0, True), “Most correct Healthy”)
```

Most correct Healthy

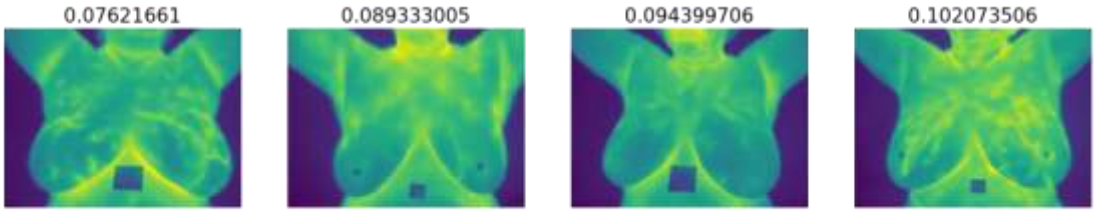



*Figura 8 - Imágenes Healthy más correctamente clasificadas*

```
plot_val_with_title(most_by_correct(1, True), “Most correct Sick”)
```

Most correct Sick

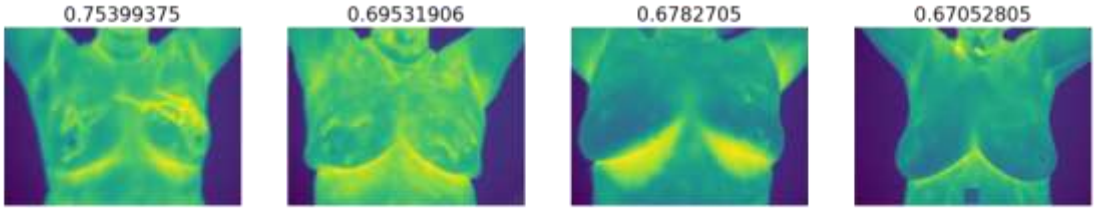



*Figura 9 - Imágenes Sick más correctamente clasificadas*

```
plot_val_with_title(most_by_correct(0, False), “Most incorrect Healthy”)
```

Most incorrect Healthy

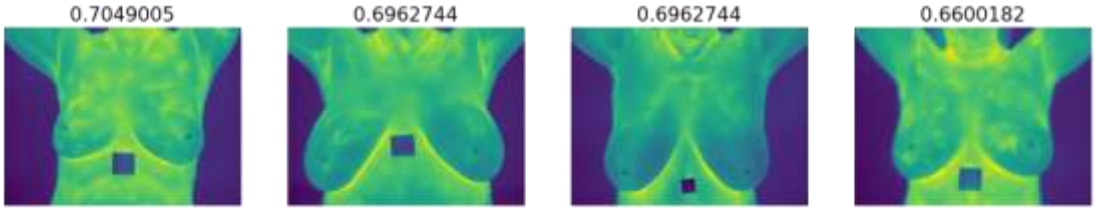



*Figura 10 - Imágenes Healthy más incorrectamente clasificadas*

```
plot_val_with_title(most_by_correct(1, False), “Most incorrect Sick”)
```

Most incorrect Sick

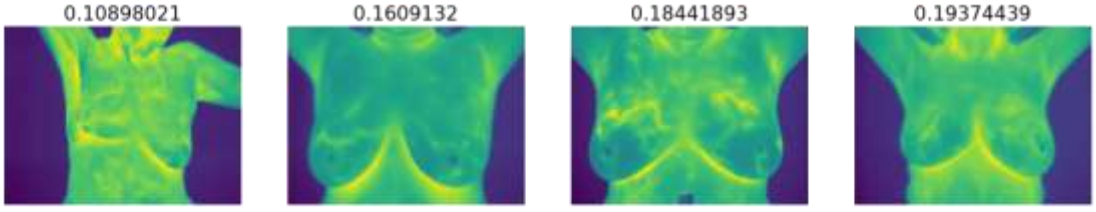



*Figura 11 - Imágenes Sick más incorrectamente clasificadas*

```
Most_uncertain = np.argsort(np.abs(probs -0.5))[:4]
plot_val_with_title(most_uncertain, “Most uncertain predictions”)
```

Most uncertain predictions

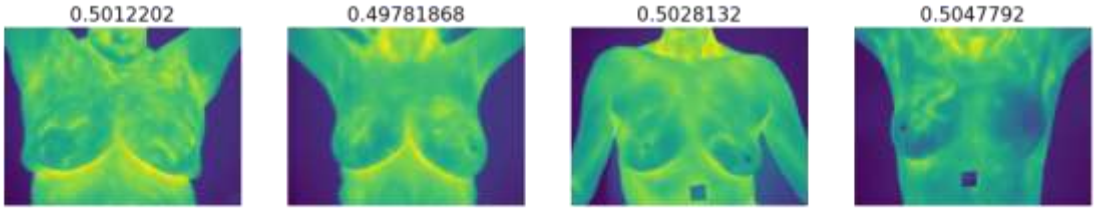



*Figura 12 - Imágenes con mayor incertidumbre en la clasificación*

## 4.5 Mejora del modelo

Continuando con el desarrollo del procedimiento, el siguiente paso es aprovechar las funciones que facilita Fast.ai para introducir técnicas habituales en redes neuronales convolucionales, como son Data Augmentation y Fine-Tuning; lo que ayudará a mejorar as precisiones

### 4.5.1 Data Augmentation

Esta técnica consiste en la realización de operaciones aleatorias de rotación, traslación y zoom sobre las imágenes. Con esto se logra evitar el overfitting o sobre ajuste, que no es más que evitar que la red se ajuste a un conjunto de imágenes concreto, sin ser capaz de generalizar.

Al pasar versiones modificadas de las imágenes con las operaciones citadas, se logrará que la red aprenda y sea capaz de extraer características nuevas, consiguiendo generalizar mucho mejor.

En Fast.ai esto lo posibilita el parámetro **aug_tfms** de la función **tfms_from_model**. Así por ejemplo se puede especificar el zoom hasta una escala concreta mediante el parámetro **max_zoom**.

```
tfms = tfms_from_model(resnet34, sz, aug_tfms=transforms_side_on, max_zoom=1.1)
def get_augs():
    data = ImageClassifierData.from_paths(PATH, bs=2, tfms=tfms, num_workers=1)
    x,_ = next(iter(data.aug_dl))
    return data.trn_ds.denorm(x)[1]
ims = np.stack([get_augs() for i in rango(6)])
plots(ims, rows = 2)
```

*Código 10 - Aplicación de la técnica Data Augmentation*

En la Figura 13 se muestra un ejemplo de imágenes obtenidas mediante Data Augmentation.

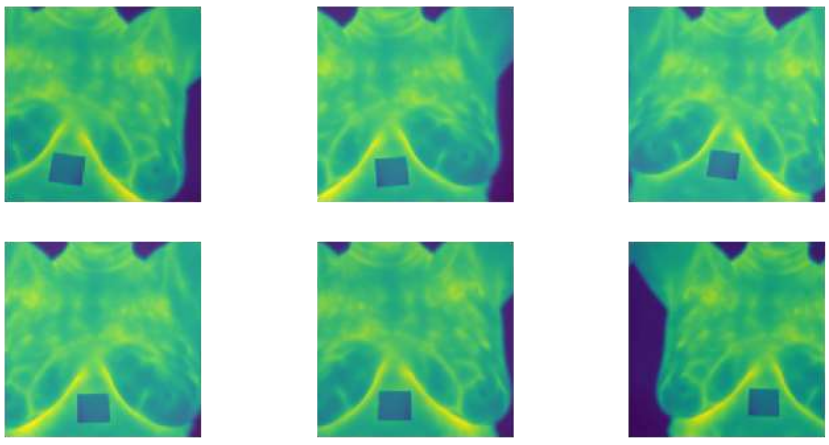


*Figura 13 - Muestra de imágenes obtenidas con Data Augmentation*

Lo que se debe hacer ahora es crear un objeto data que incluya las nuevas imágenes, pudiendo así realizar un nuevo ajuste del modelo al igual que antes:

```
data = ImageClassifierData.from_paths(PATH, tfms=tfms)
learn = ConvLearner.pretrained(arch, data, precompute=True)
learn.fit(1e-2, 1)
```

Resultando:

| Epoch | trn_loss | val_loss | accuracy |
|---|---|---|---|
| 0 | 0.54735 | 0.283614 | 0.905 |

*Código 11 - Cálculo incluyendo los datos obtenidos mediante Data Augmentation*

### 4.5.2 Descongelar capas

De forma predeterminada todas las capas menos la última están congeladas, actualizándose sólo los pesos en la última capa, para descongelarlos y permitir que se actualicen se debbe modificar el parámetro precompute:

```
learn.precompute = False
```

*Código 12 - Descongelación de los pesos*

También se puede introducir el hiperparámetro **cycle_len**, indicando así la utilización del "stochastic gradient descent with restarts (SGDR)", una variante de learning rate annealing. Con este procedimiento lo que se hace es disminuir gradualmente la tasa de aprendizaje a medida que avanza el entrenamiento, dicho de otra manera, reducir los pasos a medida que los pesos se aproximan a su valor óptimo.

Esto se realiza porque nos podemos encontrar con zonas en las que pequeños cambios de los pesos suponen grandes pérdidas, para evitarlo, de vez en cuando se aumenta la tasa de aprendizaje, forzando al modelo a saltar a zonas diferentes en el espacio de los pesos.

El número de épocas entre cada restablecimiento de la tasa de aprendizaje se establece mediante este parámetro cycle_len, y el número de veces que sucede, es el número de ciclos, indicado en el segudo parámetro.

El nuevo ajuste quedaría de la siguiente manera:

```
learn.fit(1e-2, 3, cycle_len = 1)
```

*Código 13 - Nuevo ajuste con los pesos descongelados*

Resultando:

| Epoch | trn_loss | val_loss | accuracy |
|---|---|---|---|
| 0 | 0.362536 | 0.261552 | 0.905 |
| 1 | 0.365487 | 0.236783 | 0.93 |
| 2 | 0.333971 | 0.221858 | 0.935 |

*Código 14 - Resultados con los pesos descongelados*

A modo de curiosidad en la figura 13 vemos el resultado de reajustar la tasa de aprendizaje tres veces:

```
learn.sched.plot_lr()
```

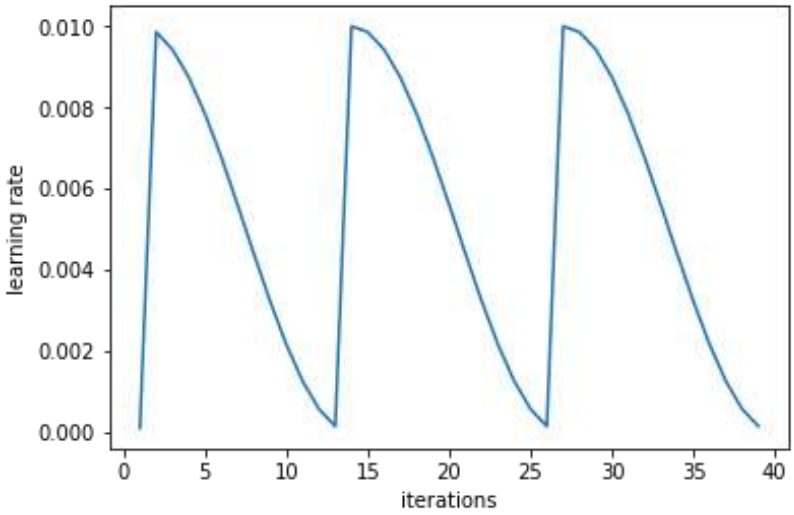



*Figura 14 - Comportamiento de las tasas de aprendizaje*

### 4.5.3 Fine-tuning con tasa de aprendizaje diferencial

Siguiendo con el refinamiento del procedimiento el siguiente paso es recurrir a la técnica de Fine-tuning, la cual se basa en descongelar los pesos de las capas convolucionales, permitiendo a la red entrenarse en su totalidad. La función de Fast.ai que nos facilita esto es **unfreeze**.

A su vez y recordando que los pesos pueden tener información más o menos generalizada e interesante en función del nivel de la capa en la que se encuentren, en Fast.ai se suele aplicar lo que se conoce como **"tasa de aprendizaje diferencial"**, consistente en aplicar un array con tres tasas de aprendizaje distintas. El primero de los valores del array se aplica a las primeras capas, el segundo a las intermedias y el tercero a las últimas. Los valores del array serán sucesivamente mayores, teniendo así en cuenta el conocimiento diferencial que tienen las capas, siendo las últimas las que menos modificaciones requerirán por tener un conocimiento más generalizado.

También hay que señalar la utilización del parámetro **cycle_mult** con valor 2, indicando con esto que se dobla el ciclo en cada paso.

El nuevo ajuste se realizaría de la siguiente manera:

```
learn.unfreeze()

lr = np.array([1e-4, 1e-3,1e-2])
learn.fit(lr, 3, cycle_len = 1, cycle_mult = 2)
```

*Código 15 - Aplicar tasas de aprendizaje diferenciales*

Resultando:

| epoch | trn_loss | val_loss | accuracy |
|---|---|---|---|
| 0 | 0.530134 | 0.26339 | 0.915 |
| 1 | 0.360913 | 0.212397 | 0.93 |
| 2 | 0.272829 | 0.154322 | 0.94 |
| 3 | 0.220812 | 0.125904 | 0.965 |
| 4 | 0.171927 | 0.107653 | 0.965 |
| 5 | 0.137743 | 0.095862 | 0.97 |
| 6 | 0.11579 | 0.090694 | 0.97 |

```
[array(0.09069]), 0.97]
```

*Código 16 - Resultado de aplicar tasas de aprendizaje diferencial*

Veamos gráficamente el efecto que tiene el parámetro cycle_mult sobre la tasa de aprendizaje:

```
learn.sched.plot_lr()
```

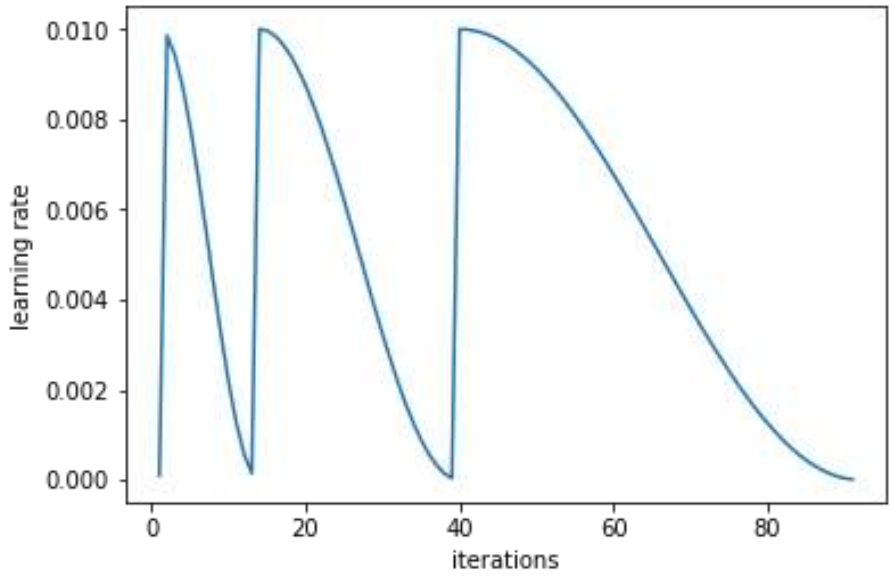



*Figura 15 - Comportamiento de la tasa de aprendizaje doblando ciclo*

Un último método que se puede aplicar es el conocido como **"aumento de tiempo de prueba"** TTA, consistente en hacer predicciones sin limitarse al conjunto de validación, incluyendo también versiones obtenidas por aumentos aleatorios de las ya existentes. Por defecto se utiliza un promedio obtenido de la imagen original y cuatro versiones aumentadas aleatoriamente. Con esto se obtiene en algunos casos mejoras de entre el 10% y 20% del error.

```
log_preds,y = learn.TTA()
probs = np.mean(np.exp(log_preds), 0)

accuracy_np(probs, y)
```

*Código 17 - Aplicación del aumento de tiempo de prueba (TTA)*

### 4.5.4 Matriz de confusión

Completado todo el procedimiento ya podemos hacer la valoración de la clasificación obtenida mediante la matriz de confusión correspondiente, aprovechando que **Scikit-learn** tiene una función para este cometido:

```
preds = np.argmax(probs, axis=1)
probs = probs[:,1]
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, preds)
```

*Código 18 - Cálculo de la matriz de confusión*

Gráficamente se obtiene:

```
plot_confusion_matrix(cm, data.classes)
```

*Código 19 - Representar matriz de confusión*

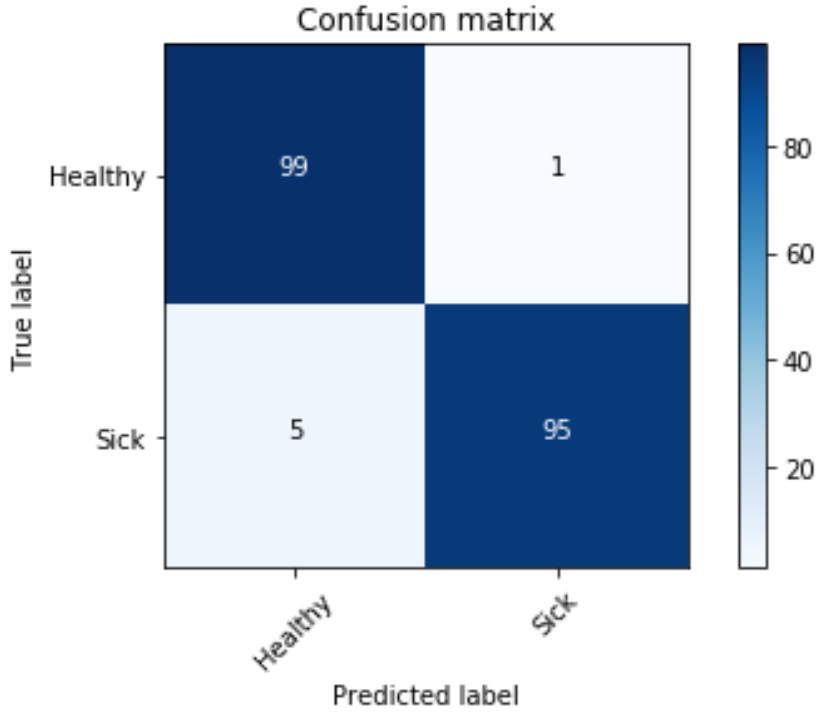



*Figura 16 - Matriz de confusión*

La exactitud es bastante optimista, con 1 Healthy y 5 Sicks erróneos, resultado que ya anticipa buenas predicciones para la utilización de las redes neuronales convolucionales en la clasificación de carcinoma de mama a partir de termografías infrarrojas.

### 4.5.5 Revisión de imágenes

Para terminar, con el objetivo de obtener una mayor claridad de los resultados obtenidos, es recomendable visionar aquellos que han resultado erróneos, aplicando nuevamente las funciones introducidas anteriormente, ayudándonos a ver posibles razones:

```
plot_val_with_title(most_by_correct(0, False), "Most incorrect Healthy")
```

*Figura 17 - Representar los Healthy más incorrectos*

Most Incorrect Healthy

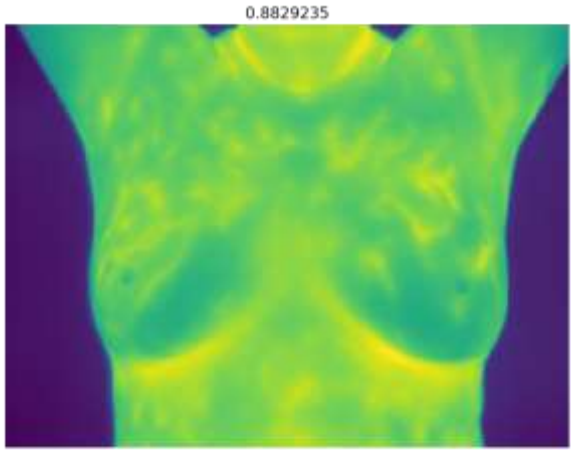



*Figura 18 - Healthy más incorrectos*

```
plot_val_with_title(most_by_correct(1, False), "Most incorrect sick")
```

*Código 20 - Representar los Sick más incorrectos*

Most incorrect sick

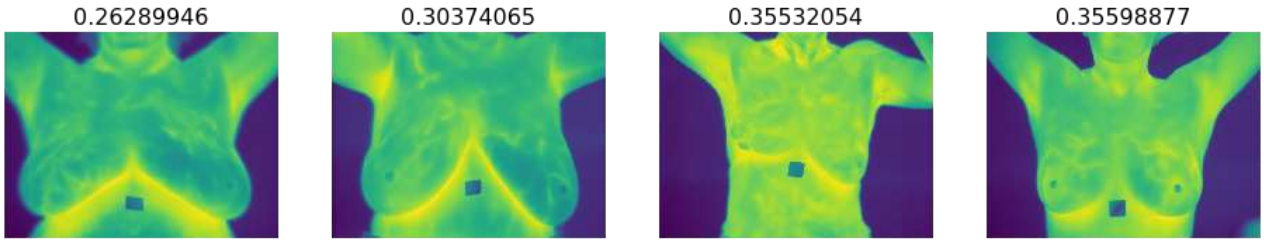



*Figura 19 - Sicks más incorrectos*

### 4.5.6 Resultados con otras arquitecturas

Desarrollado ya en el punto 4.5 el procedimiento de referencia, lo que corresponde ahora es extender el estudio a otras arquitecturas, aplicando la habitual organización de las imágenes en un grupo de training, otro de validación y un tercero de test que ya indivamos. El método consiste en aplicar el procedimiento con cada una de las arquitecturas utilizando los grupos de training y validación. A continuación para la arquitectura que de un mejor resultado se aplicará nuevamente el procedimiento pero esta vez utilizado el grupo de test.

Recordar que lo primero es obtener un conjunto aleatorio de imágenes balanceadas, 500 Healthy y 500 Sick, organizando a su vez cada clase en los tres grupos citados con las siguientes proporciones: 320 training, 80 validación y 100 test. Aprovechando las funciones creadas específicamente para este objetivo y que se detallaron en el punto 4.1.

Las arquitecturas sobre las que se aplica el estudio son: resnet18, resnet34, resnet50, resnet152, vgg16 y vgg19.

Es necesario destacar que al ajustar los parámetros en Jupyter Notebook, las tres últimas arquitecturas dieron problemas de memoria cuando se trataba de aplicar la técnica de FineTuning, por lo que se tuvo que omitir esta técnica en estas tres arquitecturas.

Mostramos los accuracity y las matrices de confusión resultantes:

| | resnet18 | resnet34 | resnet50 | resnet152 | vgg16 | vgg19 |
|---|---|---|---|---|---|---|
| **accuracy** | 0.9625 | 0.975 | 0.975 | 0.975 | 0.9375 | 0.9112 |

*Tabla 3 - Precisión de las diferentes arquitecturas*

Las tablas de confusión serían:

| Resnet18 | Healthy | Sick |
|---|---|---|
| Healthy | 74 | 6 |
| Sick | 0 | 80 |
| Accuracy: | 0.9625 | |

| Resnet34 | Healthy | Sick |
|---|---|---|
| Healthy | 76 | 4 |
| Sick | 0 | 80 |
| Accuracy: | 0.975 | |

| Resnet50 | Healthy | Sick |
|---|---|---|
| Healthy | 76 | 4 |
| Sick | 0 | 80 |
| Accuracy: | 0.975 | |

| Resnet152 | Healthy | Sick |
|---|---|---|
| Healthy | 77 | 3 |
| Sick | 1 | 79 |
| Accuracy: | 0.975 | |

| Vgg16 | Healthy | Sick |
|---|---|---|
| Healthy | 70 | 10 |
| Sick | 0 | 80 |
| Accuracy: | 0.9375 | |

| Vgg19 | Healthy | Sick |
|---|---|---|
| Healthy | 74 | 6 |
| Sick | 8 | 72 |
| Accuracy: | 0.9112 | |

*Tabla 4 - Matrices de confusión de las diferentes arquitecturas*

## Valoración con la arquitectura más precisa

Tras aplicar el procedimiento a las diferentes arquitecturas, se puede hacer una valoración final utilizando el modelo con resultados más precisos, resnet34, tal como se refleja en la tabla 3, siendo además la que ha manifestado una mayor agilidad en los tiempos de cálculo.

Aplicando una vez más el procedimiento para esta arquitectura, pero esta vez sobre el grupo test, obtenemos como resultado la siguiente matriz de confusión:

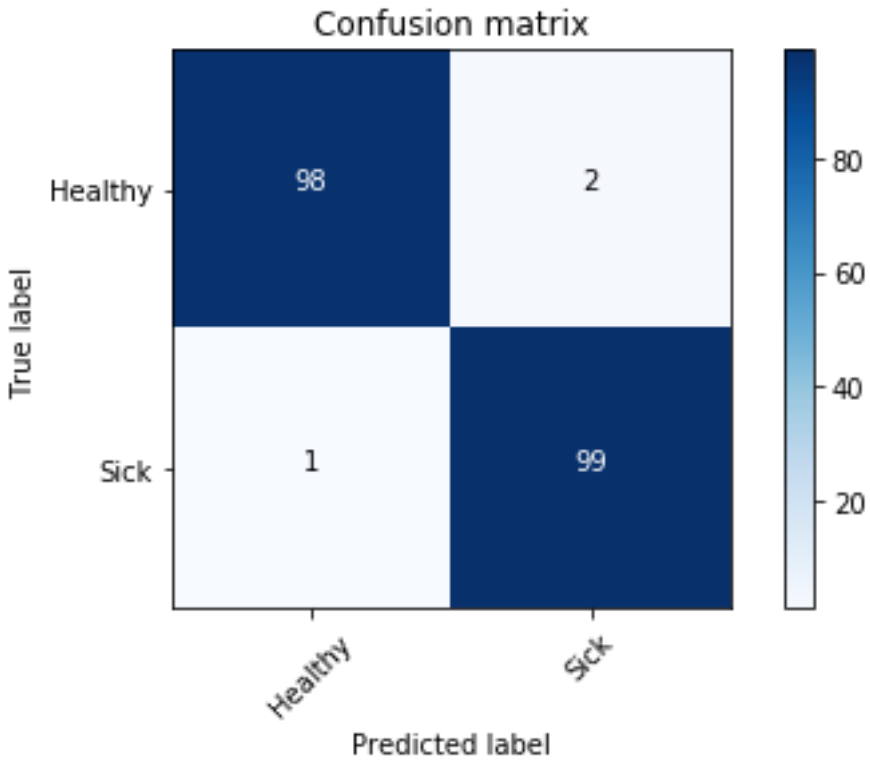



*Figura 20 - Matriz de confusión para la arquitectura resnet34 utilizando los datos de test*

Lo que nos da una precisión del 98.5 %, validando a las redes neuronales convolucionales como una técnica apropiada para realizar clasificación de carcinoma de mama a partir de termografías infrarrojas.

## 4.6 Comparación con otros artículos

En la investigación previa no se había localizado estudio alguno en el que se hayan utilizado CNN para la detección de carcinoma de mama mediante la clasificación de imagen termográfica infrarroja. Este puede ser considerado un primer estudio que abra el camino y también ayude a complementar estudios afines.

Igualmente es justo recordar los resultados obtenidos con otros algoritmos en estudios previos ya citados en el punto 1.1. Podemos citar las investigaciones de Silva et al., 2014 [1] y Acharya et al. [2], empleando el clasificador SVM; o la de Mejía et al. [7] utilizando el clasificador K-NN obteniendo una precisión del 94.44%; o los trabajos de Sadek Ali et al. [9] utilizando SVM-RFB; así como los de Sathish et al. [10] utilizando SVM con precisiones entre el 83.75% y el 90%. Las precisiones del 98.5% alcanzaras en este trabajo con CNN, ponen a esta tecnología en una posición optimista para su utilización en la clasificación de carcinoma utilizando termografías.

## 4.7 Discusión final

Durante el progreso de este trabajo se han desarrollado los dos objetivos específicos planteados inicialmente:

- Creación de un procedimiento que permita clasificar pacientes sanas y enfermas en cáncer de mama. Partiendo de un conjunto de termografías y aplicando la librería de Fast.ai.
- Contrastar el procedimiento aplicándolo a diferentes arquitecturas.

Los resultados obtenidos son satisfactorios, logrando una precisión alta en los pronósticos, que alcanzan el 98.5%, a pesar de partir de un número reducido de imágenes. Todo esto ha permitido alcanzar, y con optimismo, el objetivo general inicial, evaluar las redes neuronales convolucionales para la detección precoz del cáncer de mama basándose en imágenes de termografía.

Las CNN se han mostrado como una alternativa real a la detección precoz del carcinoma de mama mediante la clasificación automática de termografías infrarrojas, ampliando y reforzando los estudios previos que mediante otras técnicas lograron resultados prometedores utilizando este tipo de fuentes.

El objetivo en esta línea de estudios no es otro que lograr diagnósticos lo suficientemente fiables, económicos y no invasivos, que permitan aumentar las tasas de supervivencia de las pacientes y abaratar los costes de los sistemas sanitarios.

Como resultado final nos encontramos con tres protagonistas muy a tener en cuenta, las termografías como fuente fiable, no invasiva y económica; el Machine Learning que se pone a disposición de la sociedad de forma muy accesible gracias a iniciativas como las de Fast.ai, y por último las CNN como instrumento de apoyo a un detección precoz y fiable del carcinoma de mama.

Por otro lado profundizar en el análisis de imagen termográfica es hacerlo por extensión en el de la imagen médica, aportando técnicas reutilizables en otro tipo de imágenes médicas.

### 4.7.1 Futuros estudios:

Se propone extender el procedimiento a otro conjunto de pacientes, afinar en el ajuste de las arquitecturas, permitiendo entre otras mejoras resolver los problemas de memoria que mostraron algunas ellas, tratando también de ampliarlo a otras nuevas. Todo ello con el objetivo final de elaborar una publicación en la que se muestren los resultados de estos trabajos.

También se puede valorar la utilización directa de los tensores en vez de las imágenes, aprovechando las funciones que nos facilita Fast.ai, **ImageClassifierData.from_arras**, parece la adecuada.

Un objetivo deseable sería el desarrollo de una aplicación que automatice el diagnóstico a partir de termografías. Valorando la extensión del procedimiento a otro tipo de dolencias en las que la termografía pueda aportar información sensible.

# 5. Conclusiones

Tanto el objetivo general como el específico se han cumplido satisfactoriamente. Las redes neuronales convolucionales se han mostrado como una herramienta útil para la detección precoz de carcinoma de mama a partir de termografías infrarrojas.

Los logros personales alcanzados durante el desarrollo del trabajo serían los siguientes.

- Descubrir las posibilidades de la termografía infrarroja y sus fundamentos.
- Conocer la problemática del carcinoma de pecho, especialmente en lo concerniente a su diagnóstico.
- Profundizar en la programación con Python.
- Introducirse en la programación y uso de las redes neuronales convolucionales.
- Conocer las librerías de Fast.ai y la potencia de sus funciones de alto nivel, facilitando la clasificación automática de imágenes mediante redes neuronales convolucionales.
- Introducirse en el conocimiento de la librería Pytorch, soporte fundamental de Fast.ai. Mostrándose como una alternativa seria otras equivalentes, como son TensorFlow y Keras.
- Utilización de los servidores Paperspace, que aportan una plataforma especialmente configurada para proyectos de MachineLearning a un precio económico.
- Descubrir las posibilidades que ofrece la termografía infrarroja y sus fundamentos.

# 6. Bibliografía

[1] [2014] A new database for breast research with infrared image. L. F. Silva, D. C. M. Saade, G. O. Sequeiros, A. C. Silva, A. C. Paiva, R. S. Bravo, and A. Conci.
Banco de imágenes Visual Lab: http://visual.ic.uff.br/dmi

[2] [2012] Thermography Based Breast Cancer Detection Using Texture Features and Support Vector Machine. U. Rajendra Acharya & E. Y. K. Ng & Jen-Hong Tan & S. Vinitha Sree.

[3] [2013] Breast thermography from an image processing viewpoint_ A survey. Tiago B.Borchartt a,n, AuraConci, RitaC.F.Lima, RogerResmini, AngelSanchez.

[4] [2014] Detección temprana del cáncer de mama mediante la termografía en Ecuador. María G. Pérez, Aura Conci, Aida Aguilar, Ángel Sánchez, Víctor H. Andaluz.

[5] [2014] Automatic segmentation of thermal imageas to support breast cáncer diagnosis. Steve Rodríguez Guerrero, Humberto Loaiza, Andrés David Retrepo.

[6] [2014] Interval symbolic feature extraction for thermography breast cáncer detection. Marcus C. Araújo, Rita C.F. Lima, Renata M.C.R. de Souza

[7] [2015] Automatic Segmentation and Analysis of Thermograms Using Texture Descriptors for Breast Cancer Detection. Tatiana M. Mejía, María G. Pérez, Víctor H. Andaluz, Aura Conci.

[8] [2015] Breast Abnormality Detection Through Statistical Feature Analysis Using Infrared Thermograms. Usha Rani Gogoi, Gautam Majumdar, Mrinal Kanti Bhowmik, Anjan Kumar Ghosh, Debotosh Bhattacharjee,

[9] [2015] Detection of Breast Abnormalities of Thermograms based on a New Segmentation Method. Mona Abdelbaset Sadek Ali, Aboul Ella Hassanien, Tarek Gaber, Lincoln Silva.

[10] [2016] Asymmetry analysis of breast thermograms using automated segmentation and texture features. Dayakshini Sathish, Surekha Kamath, Keerthana Prasad, Rajagopal Kadavigere, Roshan J. Martis.

[11] [2017] Infrared imaging technology for breast cancer detection – Current status, protocols and new directions. Satish G. Kandlikar, Isaac Pérez-Raya, Pruthvik A. Raghupathi, José-Luís González-Hernández, Donnette Dabydeen, Lori Medeiros, Pradyumn a Phatak.

[12] [2015] Cyclical Learning Rathes for Training Neuronal Networks. Leslie N. Smith.

[13] [2005] Breast J. 2006 Jan-Feb; 12 Suppl 1: S3-15. PMID: 16430397 [PubMed - indexed for MEDLINE]. Anderson, B.O., R. Shyyan, A. Eniu, R.A. Smith, C.H. Yip, N.S. Bese, L.W. Chow, S. Masood, S.D. Ramsey, R.W. Carlson.

[14] [1940] Hebb, D. O.; Penfield, W. Human behaviour after extensive bilateral removal from the frontal lobes. Archives of Neurology and Psychiatry 44: 421-436.

[15] [1943] McCulloch, Warren; Walter Pitts. «A Logical Calculus of Ideas Immanent in Nervous Activity». Bulletin of Mathematical Biophysics.

[16] [1958] Rosenblatt, F. The perceptrón: A Probabilistic Model For Information Storage And Organization In the Brain. Psychological Review 65. 386-408.

[17] [1967] Ivakhnenko, O.G., Lapa, V.G. Cybernetics and Forecasting Techniques (Modern Analytic and Computational Methods in Science and Mathematics, v.8. ed.). American Elsevier.

[18] [1969] Minsky, Marvin; Papert, Seymour. Perceptrons: An Introduction to Computational Geometry. MIT Press.

[19] [1975] Werbos, P.J. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University.

[20] [1980] Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position".

[21] [1998] LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition".

[22] [2011] Ciresan, Dan; Ueli Meier; Jonathan Masci; Luca M. Gambardella; Jurgen Schmidhuber (2011). "Flexible, High Performance Convolutional Neural Networks for Image Classification".

[23] [1986] Learning representations by back-propagating errors (David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams.

[24]: Ponencia de Rachel Thomas sobre Fast.ai:

https://www.technologyreview.es/s/10106/la-start-que-intenta-demostrar-que-la-ia-esta-al-alcance-de-todos

[25]: ¿Qué es una red neuronal?

https://www.youtube.com/watch?v=eNIqz_noix8

[26]: Kaggle: https://en.wikipedia.org/wiki/Kaggle

[27]: Fast.ai: http://www.fast.ai

[28]: Mooc Fast.ai: https://course.fast.ai

[29]: Pytorch: https://pytorch.org

[30]: Paperspace: https://www.paperspace.com

[31]: Jupyter: https://jupyter.org/

[32]: Saedsayad: http://www.saedsayad.com/artificial_neural_network_bkp.htm

[33] [2017]: Vlarimir Perervenko. Neurodes profundas (Parte IV). Creación, entrenamiento y simulación de un modelo de neurored:

https://www.mql5.com/es/articles/3473#activation

# 7. Anexos
## 7.1 Glosario

**BI-RADS:** Breast Imaging Reporting and Data System, método para clasificar informe mamográfico, considerado el estándar en el diagnóstico de patología mamaria.

**CNN:** Convolutiona network / redes neuronales convolucionales. Red neuronal multicapa con capas convolucionadas, cada una de ellas especializada en una tarea, con especial aplicación en el tratamiento de imágenes.

**ConvNet:** Convolutiona network / redes neuronales convolucionales.

**Data Augmentation:** técnica para aumentar el conjunto de imágenes mediante transformaciones (tamaño, rotación, desplazamientos).

**Deep Leaning:** algoritmos de aprendizaje automático (Machine Learning) destinados a realizar clasificaciones/abstraciones de alto nivel en datos utilizando arquitecturas compuestas de transformaciones no lineales múltiples.

**Descenso del gradiente:** algoritmo de optimización muy extendido en el campo de las redes neuronales, en definitiva un método general de minimización.

**DTC:** Decision tree classifier / áboles de decisión. Modelo de predicción utilizado en Machine Learning.

**Fine tuning:** ajustes de pesos en redes convolucionales, permitiendo entrenar partes de la red utilizando características aprendidas con bases de datos más grandes.

**Función de activación:** función que regula la salida de una red neuronal, en virtud de los valores de entrada y de los pesos de dicha neurona.

**GPU:** Graphics procesing unit / tarjeta gráfica, que ofrece grandes ventajas de cálculo en las redes convolucionales, a través de librerías como CUDA en el caso de NVidia.

**Kaggle:** comunidad de científicos de datos que ha alcanzado gran importancia en el impulso de la inteligencia artificial.

**K-NN:** método de clasificación no paramétrico supervisado.

**Machine Learning:** campo de la inteligencia artificial destinado a que las máquinas consigan adquirir conocimiento.

**Overfiting:** sobre ajuste, sobre entrenamiento. Efecto de sobreentrenar a un algoritmo de aprendizaje con datos para los que se conoce el resultado, pudiendo quedar ajustado a una característica muy específica.

**Perceptrón:** red neuronal artificial o unidad básica de inferencia en forma de discriminador lineal, permitiendo desarrollar un algoritmo para seleccionar subgrupos a partir de grupos más grandes.

**Pipeline:** procedimiento.

**Pooling layer:** capa de muestreo. Reduce la cantidad de parámetros quedándose con las características más comunes.

**Pytorch:** framework de Python diseñada para realizar cálculos numéricos haciendo uso de la programación con tensores. Permitiendo la utilización de GPUs para acelerar los cálculos.

**Red Hebb:** red neuronal basada en la teoría de Hebb respecto a la excitación de neuronas e intercambio de información.

**Rango de aprendizaje:** rapidez o lentitud con la que se actualizan los pesos de una red neuronal.

**RFC:** Random forest classifier / bloques aleatorios.

**ROI:** regiones de interés, partes de un volumen mayor sobre las que se realizan determinados cálculos.

**SVM:** Suporta vector machine / máquinas de soporte vectorial. Conjunto de algoritmos de aprendizaje supervisado.

**TensorFlow:** biblioteca de código abierto para aprendizaje automático desarrollada por Google.

**Termografía:** técnica destinada a registrar mediante imágenes las temperaturas de distintas partes del cuerpo.

**Toma dinámica:** registro de imágenes tomadas secuencialmente en el tiempo.

**Toma estática:** registro de una única imagen.

**TTA:** aumento del tiempo de prueba, consistente en emplear los datos obtenidos por Data Augmentation en el momento de inferencia.

## 7.2 Jupyter Notebook - Procedimiento de referencia

A continuación mostramos el código implementado en Jupyter Notebook como procedimiento de referencia, ta como aplicamos en el primer modelo resnet34 con una selección trainin y validation. Procedimiento que extendimos posteriormente al resto de modelos con una selección training, validation y test; particularizando en cada modelo como corresponde.

---

# Procedimiento para detección de carcinoma mamario mediante clasificación de termografías infrarrojas usando redes neuronales profundas.

La idea es intentar realizar una clasificación sencilla de pacientes sanos y enfermos. Los enfermos han sido diagnosticados mediante biopsia o mamografía (BI-RADS 4 o mayor).

Este código automatizala el proceso a partir de las termografías, facilitando la detección precoz del cáncer de mama, permitiendo complementarse con otros métodos de diagnóstico más precisos.

Para alcanzar el objetivo aplicaremos la tecnología Deep Learning mediante redes neuronales convolucionales, utilizando Python y las funciones de alto nivel de la librería fast.ai., las cuales requieren del framework Pythorch. Todo esto se debe configurar adecuadamente sobre servidor, siendo la elección en este trabajo de hacerlo sobre los servidores en la nube de Paperspace.

El estudio se realizará sobre un dataset desbalanceado obtenido de un banco de imágenes termográficas públicas (http://visual.ic.uff.br/dmi (http://visual.ic.uff.br/dmi)) (L. F. Silva et al., 2014), combinándose con historial clínico. Recurriremos al estudio habitual con un grupo de entrenamiento y otro de validación, evaluando así la eficacia del mismo. En este procedimiento también tenemos la opción de organizar las imágenes en tres grupos: entrenamiento, validación y testeo; para poder posteriormente contrastar diferentes arquitecturas de redes neuronales convolucionales o incluso algoritmos.

Los datos termográficos los descargamos en formato txt, decidiendo pasarlos a formato jpg para aplicar las funciones de fast.ai, conversión que hemos realizado en local mediante el código siguiente:

Con una imagen:

```
f = file.open("nombre_fichero.txt")
imagen = f.read()
imagen.shape = (480,640)

imagenf = imagen.astype(float) # Pasamos los datos de temperatura a float
imagenf = imagenf/40 # Normalizamos a una temperatura máxima de 40º
plt.imshow(imagenf,cmap="gray") # Mostramos la imagen en formato de grises
plt.imsave("nombre.jpg",imagenf) # Guardamos la imagen
```

Para todas las imágenes:

```
for f in files:
   datos = open("txt/"+f)
   fichero.read()
   datos = datos.split()
```

```
imagen = np.array(datos)
imagen.shape = (480,640)
imagenf = imagen.astype(float)
imagenf = imagenf/40
 plt.imsave ("imagenes/"+f[:-4]+"jpg",imagenf)
```

Tambien hemos creado un Data Frame con información sobre cada uno de los pacientes, a modo de muestra realizaremos algún estudio preliminar:

# Análisis de los Datos:

Los datos de nuestra fuente los recogimos en una hoja excel, pudiendo asceder a los mismos mediante Pandas:

In [1]:

```
import pandas as pd
path = "../../../datajavi/BD/BDTermografiasV30.xlsx"
df = pd.read_excel(path)
df.shape
```

Out[1]:

```
(235, 23)
```

Observamos que tiene 235 registros y 23 campos. Veamos la cabecera de nuestro DataFrame:

In [2]:

```
df.head()
```

Out[2]:

| | ID | Date | Years | Status | Race | Diagnosis | Habits | Mammography | Radiotherapy | Plast |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 2012-10-08 | 70 | widow | pardo | Healthy | Low | Yes | No | No |
| **1** | 2 | 2012-10-30 | 63 | married | pardo | Healthy | Low | Yes | No | No |
| **2** | 3 | 2012-10-30 | 65 | single | black | Healthy | Low | Yes | No | No |
| **3** | 4 | 2012-10-30 | 64 | married | black | Healthy | No | Yes | No | No |
| **4** | 5 | 2012-10-30 | 63 | married | white | Healthy | Low | Yes | No | No |

5 rows × 23 columns

Realicemos algún análisis gráfico:

In [3]:

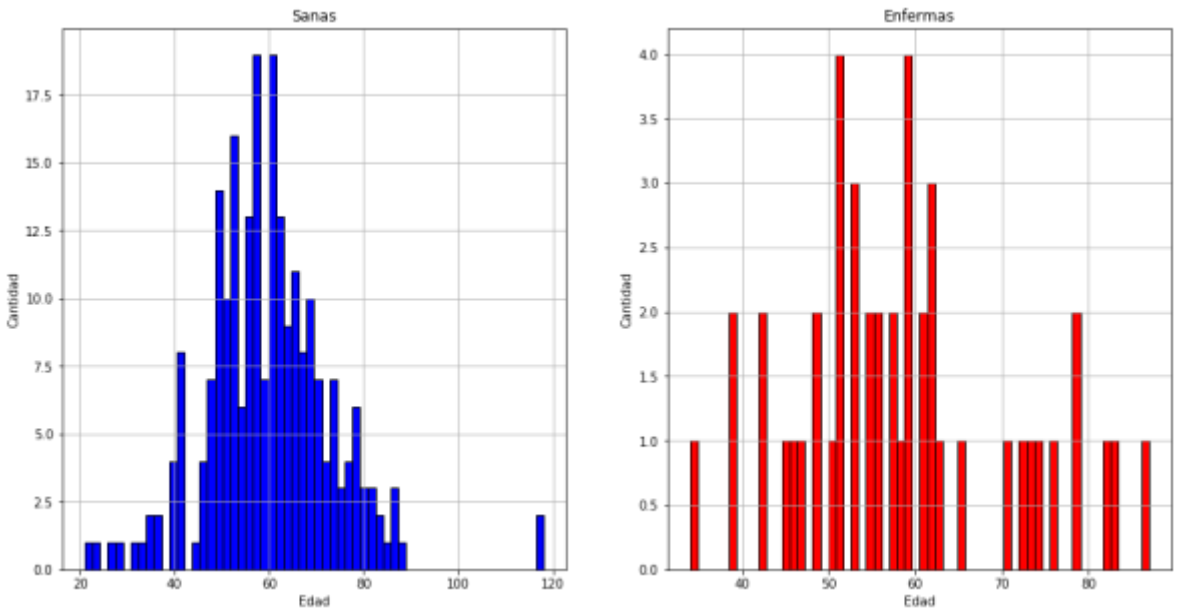
```
import matplotlib.pyplot as plt
plt.figure(figsize=(16, 8))

plt.subplot(111)
plt.title('Porcentajes enfermas y sanas en la población estudiada')
df.Diagnosis.value_counts(normalize = "Sick").plot(kind="bar",alpha=0.5)
plt.show()
```

```
<Figure size 1600x800 with 1 Axes>
```

Veamos la distribución de la población por edad en ambos grupos:

In [4]:

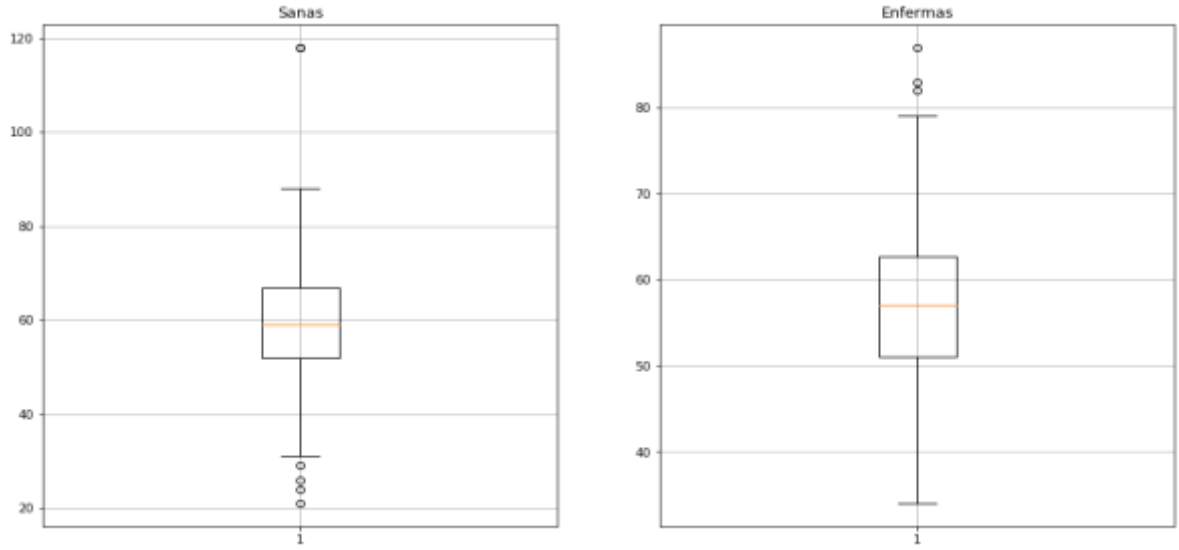
```
plt.figure(figsize=(16, 8))

plt.subplot(121)
plt.title('Sanas')
plt.xlabel('Edad')
plt.ylabel('Cantidad')
plt.hist(df.Years, bins=60, alpha=1, edgecolor = 'black',  linewidth=1, facecolor='blue
plt.grid(True)

plt.subplot(122)
plt.title('Enfermas')
plt.xlabel('Edad')
plt.ylabel('Cantidad')
plt.hist(df_sicks.Years, bins=60, alpha=1, edgecolor = 'black',  linewidth=1, facecolor
plt.grid(True)

# plt.legend(loc='top_right')


plt.show()
```



Veamos mediante boxplot la edad de pacientes sanas y enfermas:

In [5]:

```
plt.figure(figsize=(16, 8))

plt.subplot(121)
plt.title('Sanas')
plt.boxplot(df.Years)
plt.grid(True)

plt.subplot(122)
plt.title('Enfermas')
plt.boxplot(df_sicks.Years)
plt.grid(True)

plt.show()
```




# Aplicación del Procedimiento:

Tenemos el total de imágenes en la carpeta TotalImagenes, diferenciando Healthy y Sick en sendas carpetas; previamente hemos eliminado las borrosas, las que tenían un sólo seno o las que habían aplicado desodorante, ya que son imágenes que ofrecen problemas.

Hemos considerado hacer una selección balanceada de 500, destinando 400 a training y 100 a validación, sin destinar ninguna a test ya que no vamos a contrastar con otro método.

Nuestro interés a posteriori es contrastar diferentes arquitecturas, por lo que organizaremos las carpetas en tres grupos, con 320 para training, 80 par validation y 100 para test. El grupo test lo reservaremos para el modelo con mejor resultado.

Con todo esto respetamos los requisitos de estructura de la librería fast.ai para estudios sobre un conjunto de imágenes:

```
import os, shutil, random

#random.seed(5)

 PATH_IMAGENES = "../../../datajavi/TotalImagenes"
```

```
PATH_DESTINO = "../../../datajavi/SeleccionAleatoria"
cantidad = 500
cantidad_test = 0
cantidad_valid = 100

# Funciones para seleccionar:

def selecciona_imagenes(PATH_IMAGENES, PATH_DESTINO, cantidad):
    lista_imagenes = os.listdir(PATH_IMAGENES)
     seleccion_lista_imagenes = random.sample(lista_imagenes, cantidad)
     lista_destino_a_borrar = os.listdir(PATH_DESTINO)
    for imagen in lista_destino_a_borrar:
        os.remove(PATH_DESTINO+"/"+imagen)
    for imagen ineleccion_lista_imagenes:
        shutil.copy(PATH_IMAGENES+"/"+imagen, PATH_DESTINO)

def mover_imagenes(PATH_IMAGENES, PATH_DESTINO, cantidad):
    lista_imagenes = os.listdir(PATH_IMAGENES)
    seleccion_lista_imagenes = random.sample(lista_imagenes, cantidad)
    lista_destino_a_borrar = os.listdir(PATH_DESTINO)
    for imagen in lista_destino_a_borrar:
        os.remove(PATH_DESTINO+"/"+imagen)
    for imagen inseleccion_lista_imagenes:
        shutil.move(PATH_IMAGENES+"/"+imagen, PATH_DESTINO)

selecciona_imagenes(PATH_IMAGENES+"/Healthy", PATH_DESTINO+"/train/Healthy",cantidad)
selecciona_imagenes(PATH_IMAGENES+"/Sick", PATH_DESTINO+"/train/Sick", cantidad)
mover_imagenes(PATH_DESTINO+"/train/Healthy", PATH_DESTINO+"/test/Healthy",
               cantidad_test)
mover_imagenes(PATH_DESTINO+"/train/Sick", PATH_DESTINO+"/test/Sick", cantidad_test)
mover_imagenes(PATH_DESTINO+"/train/Healthy", PATH_DESTINO+"/valid/Healthy",
               cantidad_valid)
mover_imagenes(PATH_DESTINO+"/train/Sick", PATH_DESTINO+"/valid/Sick",
               cantidad_valid)
```

Ya estamos en disposición de introducir el código:

In [7]:

```
# Activamos la representación gráfica en línea del Notebook
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

Lo primero es importar las librerías necesarias.

In [8]:

```
# Importamos las librerías necesarias
from fastai.imports import *
```

```
/home/paperspace/anaconda3/envs/fastai/lib/python3.6/site-packages/sklearn/e
nsemble/weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is
```

```
an internal NumPy module and should not be imported. It will be removed in a
future NumPy release.  from numpy.core.umath_tests import inner1d
```

In [9]:

```
from fastai.transforms import *
from fastai.conv_learner import *
from fastai.model import *
from fastai.dataset import *
from fastai.sgdr import *
from fastai.plots import *
```

PATH es el path donde tenemos nuestros datos, sz es el tamaño al que se ajustarán nuestras imágenes para asegurar un entrenamiento rápido, lo ponemos en 224.

In [10]:

```
PATH = "../../../datajavi/SeleccionAleatoria/"
sz=224
```

Es importante trabajar con GPUs NVidia, y el framework de programación que nos lo permite es CUDA, debiendo para ello asegurarnos que el siguiente comando devuelve TRUE.

In [11]:

```
torch.cuda.is_available()
```

Out[11]:

```
True
```

NVidia provee funciones especiales de aceleración para deep learning en un paquete llamado CuDNN. Aunque no es estrictamente necesario mejorará significativamente el resultado del entrenamiento.

## Revisión de la organización

Es conveniente verificar que los datos están organizados como exigen las funciones de Fast.ai; es decir en dos carpetas train y valid; y cada una de ellas incluyen a su vez con otras dos referidas a las clases en las que queremos realizar la clasificación, en nuestro caso Healthy y Sick.

In [12]:

```
os.listdir(PATH)
```

Out[12]:

```
['test','valid','models',train','tmp']
```

In [13]:

```
os.listdir(f'{PATH}valid')
```

Out[13]:

```
['Healthy', 'Sick']
```

## Construcción del modelo

La librería Fast.ai nos permite construir nuestro modelo de forma sencilla y muy ágil, con tan sólo tres líneas, que entre otras cosas nos permiten elegir la arquitectura de red convolucional.

In [14]:

```
arch=resnet34 # En arch indicamos la arquitectura que aplicamos a nuestro modelo
data = ImageClassifierData.from_paths(PATH, tfms=tfms_from_model(arch, sz), bs=8)
learn = ConvLearner.pretrained(arch, data, precompute=True)
learn.fit(0.01, 2)
```

```
Epoch                                    100% 2/2 [00:01<00:00, 1.13it/s]

epoch       trn_loss   val_loss   accuracy
0           0.508364   0.141058   0.97
1           0.478382   0.036208   0.985
```

Out[14]:

```
[array([0.03621]), 0.985]
```

En muy poco tiempo de cálculo obtenemos nuestro modelo y su precisión.

La función clave es learn.fit, encargada de realizar todos los cálculos y construir el modelo, requiriendo de tan sólo dos parámetros, el primero corresponde al rango de aprendizaje y el segundo al número de épocas  número de ciclos en los que se repiten los cálculos sobre todos los elementos a clasificar).

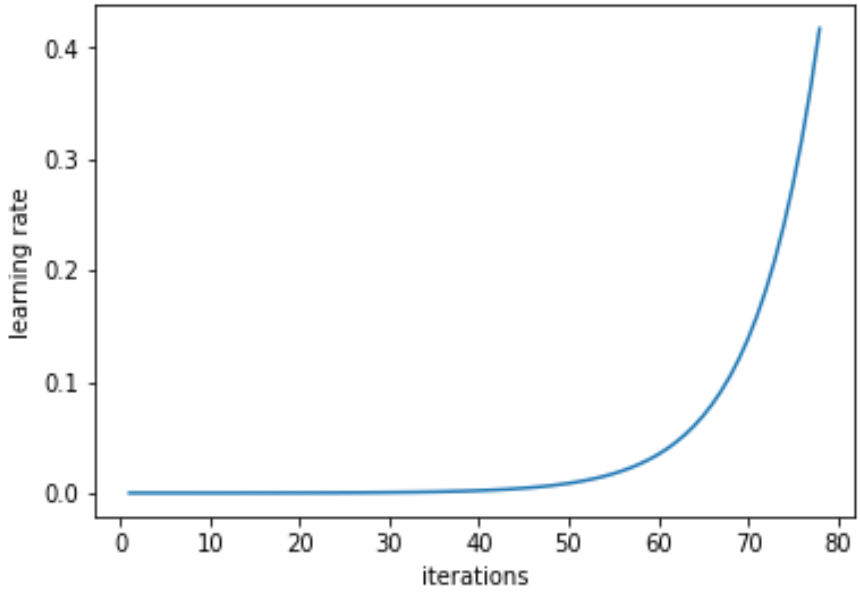
El rango de aprendizaje, es un parámetro propio de las redes neuronales, determina la rapidez o la lentitud con la que desea actualizar los pesos, siendo además difícil de establecer, ya que afecta significativamente al modelo.

Para ayudarnos a determinar el rango de aprendizaje Fast.ai nos facilita la función learn.lr_find(). El objeto learn contiene a su vez el objeto sched, que aporta una serie de funciones gráficas para facilitar la determinación de la tasa de aprendizaje. Veamos el procedimiento.

In [15]:

```
learn = ConvLearner.pretrained(arch, data, precompute=True)
lrf=learn.lr_find()
learn.sched.plot_lr()
```

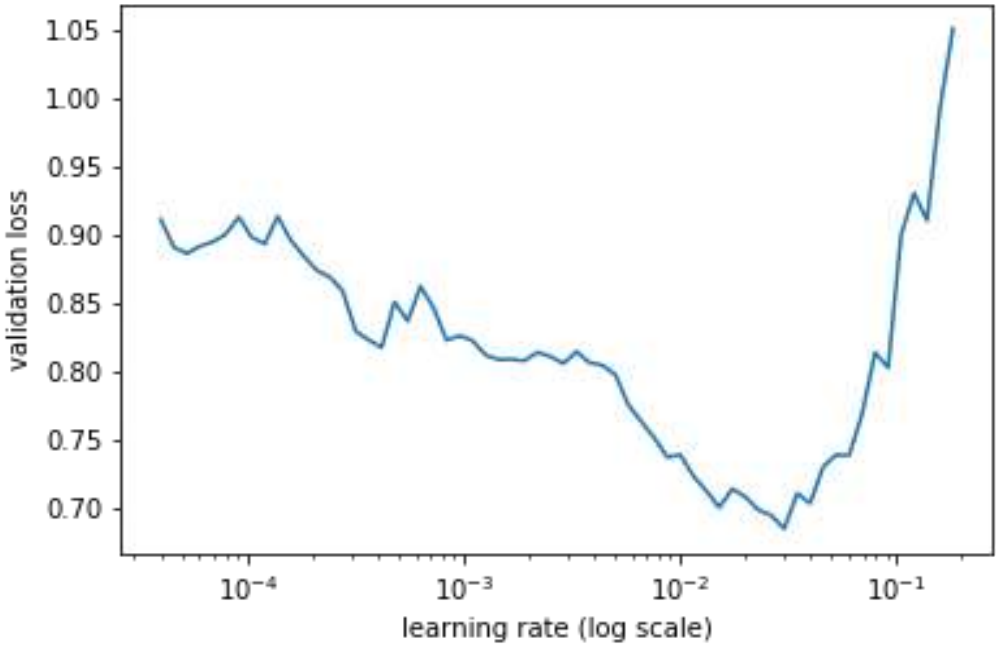
```
Epoch                                    0% 0/1 [00:00<?, ?it/s]

59%|█████▉    | 59/100 [00:00<00:00, 54.29it/s, loss=0.685]
```




Pero lo que realmente nos interesa es ver la gráfica de pérdida en función de la tasa de aprendizaje para ver dónde nuestra pérdida deja de disminuir, redondeando a la potencia de 10 anterior más próxima:

In [16]:

```
1 learn.sched.plot()
```




En nuestro caso elegiríamos lr=1e-2 (0.01).

## Analizando la clasificación

Lo que corresponde en este punto es hacer un análisis de la clasificación realizada por nuestro modelo, para ello es necesario introducir una serie de funciones que nos ayuden a evaluar las diferentes clasificaciones

In [17]:

```
# Obtenemos predicciones para el conjunto de validación en escala logarítmica
log_preds = learn.predict( )

preds = np.argmax(log_preds, axis=1)
probs = np.exp(log_preds[:,1])
```

In [18]:

```
def rand_by_mask(mask): return np.random.choice(np.where(mask)[0], min(len(preds),
4), replace = False)
def rand_by_correct(is_correct): return rand_by_mask((preds ==
data.val_y)==is_correct)
```

In [19]:

```
def plots(ims, figsize=(12,6), rows=1,titles=None):
   f = plt.figure(figsize=figsize)
   for i in range(len(ims)):
         sp = f.add_subplot(rows, len(ims)//rows, i+1)
         sp.axis('Off')
         if titles is not None: sp.set_title(titles[i], fontsize=16)
         plt.imshow(ims[i])
```

In [20]:

```
def load_img_id(ds, idx): return np.array(PIL.Image.open(PATH+ds.fnames[idx]))

    def plot_val_with_title(idxs, title):
    imgs = [load_img_id(data.val_ds,x) for x in idxs]
    title_probs = [probs[x] for x in idxs]
    print(title)
    return plots(imgs, rows=1, titles=title_probs, figsize=(16,8)) if len(imgs)>0
    else print('Not Found')
```

Completamos con otro par de funciones:

In [21]:

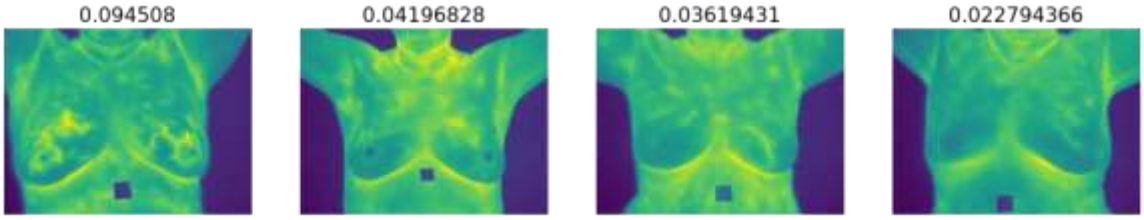
```
    def most_by_mask(mask, mult):
    idxs = np.where(mask)[0]
    return idxs[np.argsort(mult * probs[idxs])[:4]]

    def most_by_correct(y, is_correct):
    mult = -1 if (y==1)==is_correct else 1
    return most_by_mask(((preds == data.val_y)==is_correct) & (data.val_y == y),
    mult)
```

Y ya estamos en disposición de ver las clasificaciones de los diferentes grupos:

In [22]:

```
# 1. A few correct labels at random
plot_val_with_title(rand_by_correct(True), "Correctly classified")
```

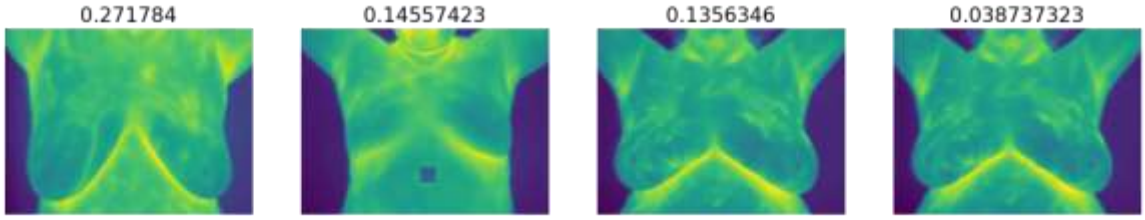
Correctly classified




In [23]:

```
# 2. A few incorrect labels at random
plot_val_with_title(rand_by_correct(False), "Incorrectly classified")
```

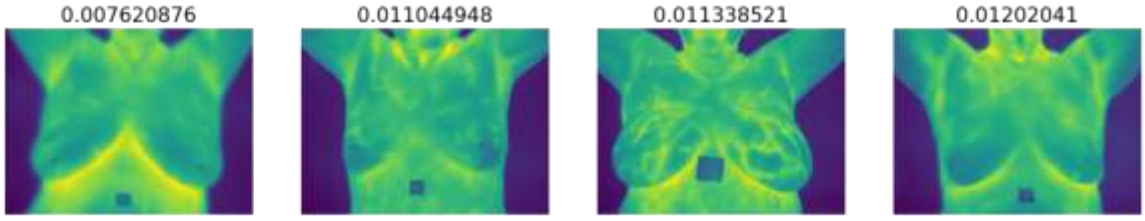
Incorrectly classified




In [24]:

```
plot_val_with_title(most_by_correct(0, True), "Most correct Healthy")
```

Most correct Healthy




In [25]:

```
plot_val_with_title(most_by_correct(1, True), "Most correct sick")
```

Most correct sick
Not Found.

In [26]:

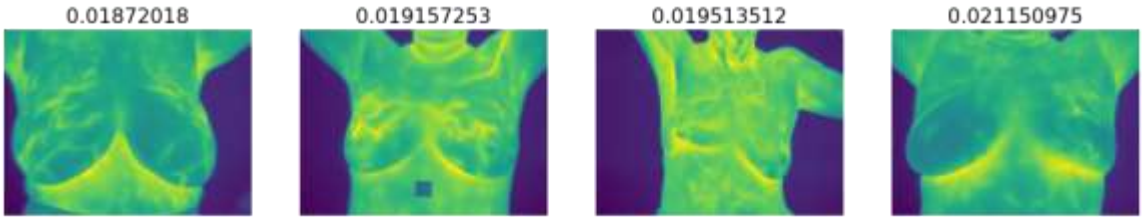
```
plot_val_with_title(most_by_correct(0, False), "Most incorrect healthy")
```

```
Most incorrect healthy
Not Found.
```

In [27]:

```
plot_val_with_title(most_by_correct(1, False), "Most incorrect sick")
```

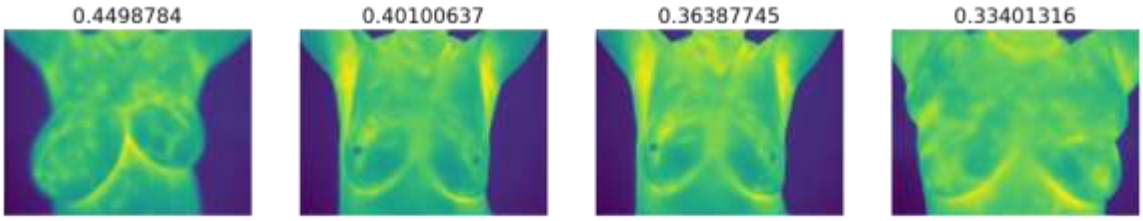
```
Most incorrect sick
```




In [28]:

```
most_uncertain = np.argsort(np.abs(probs -0.5))[:4]
plot_val_with_title(most_uncertain, "Most uncertain predictions")
```

```
Most uncertain predictions
```




# Mejorando nuestro modelo

## Data augmentation

Para mejorar nuestro modelo podemos utilizar la técnica conocida como Data Augmentation, basada fundamentalmente en la aplicación de diferentes transformaciones a nuestras imágenes de forma aleatoria, entre las que se incluyen: rotación, translación y zoom.

Esta técnica nos permite evitar el overfitting o sobreajuste, término que se identifica con la incapacidad de generalizar. Con las modificaciones comentadas, ayudamos a que sea capaz de generalizar mejor.

Fast.ai nos lo facilita mediante el parámetro "aug_tfms" en la funcion "tfms_from_model". Para el caso concreto del zoom tenemos el parámetro "max_zoom".

In [29]:

```
tfms = tfms_from_model(resnet34, sz, aug_tfms=transforms_side_on, max_zoom=1.1)
```

In [30]:

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

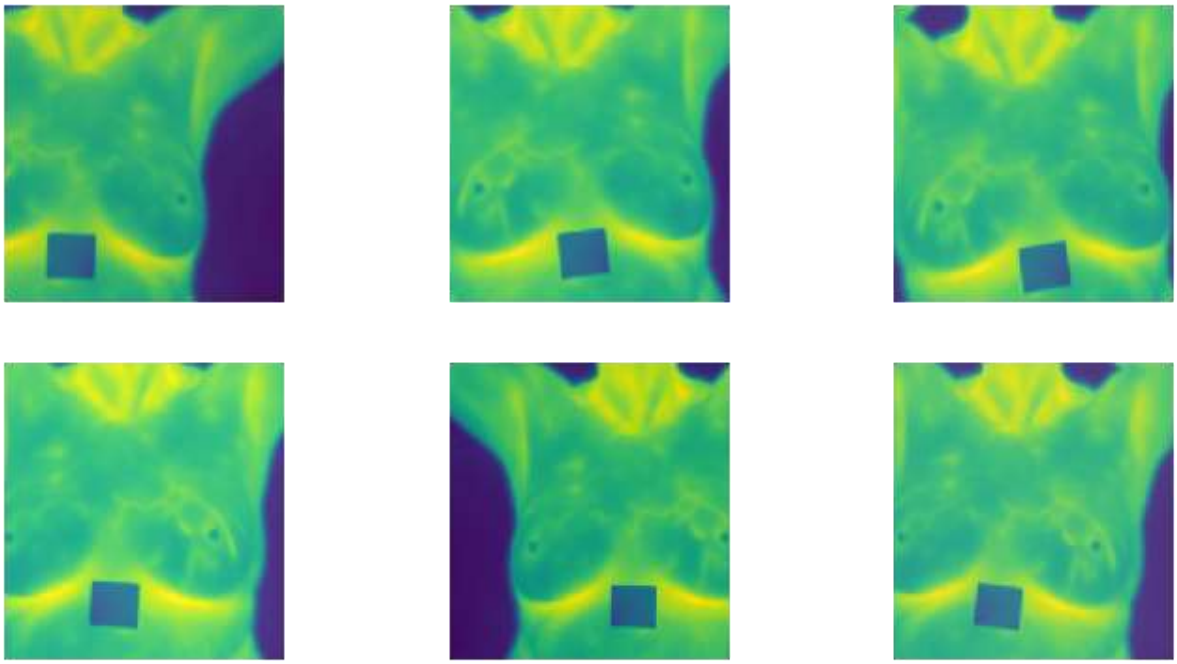
In [31]:

```
ims = np.st ack([get_augs() for i in range(6)])
```

Mostremos una selección de las modificiones para verlo con más claridad:

In [32]:

```
plots(ims, rows=2)
```



Ahora lo que tenemos que hacer es crear un objeto "data" que incluya las nuevas imágenes obtenidas con Data augmentation. A continuación aplicamos el modelo y vemos la mejora.

In [33]:

```
  data = ImageClassifierData.from_paths(PATH, tfms=tfms)
  learn = ConvLearner.pretrained(arch, data, precompute=True) In [42]:
```

```
1 learn.fit(1e-2, 1)
```

```
Epoch                                    100% 1/1 [00:00<00:00, 5.05it/s]

 epoch             trn_loss   val_loss   accuracy
       0           0.58724    0.402866   0.84
```

Out[33]:

```
[array([0.40287]), 0.84]
```

También hay que señalar que por defecto todas las capas menos la última tienen los pesos "congelados", sólo se están actualizando los pesos de la última capa. Lo que podemos hacer es descongelar los pesos antes de realizar un nuevo ajuste. Esto se logra modificando el siguiente parámetro:

In [34]:

```
1 learn.precompute=False
```

Podemos realizar un nuevo ajuste del mismo modo que hicimos anteriormente, pero ahora introduciremos un nuevo parámetro, "cycle_len", para indicar la utilización del descenso del gradiente con reinicios (SGDR), consistente en disminuir gradualmente la tasa de aprendizaje a medida que avanza el entrenamiento, que resulta especialmente útil cuando nos vamos acercando a los valores óptimos de los pesos, queriendo dar pasos más pequeños.

Pero también nos podemos encontrar con zonas en las que pequeños cambios de los pesos suponen grandes cambios en la pérdida, para animar al modelo a encontrar zonas más precisas, de vez en cuando se aumenta la velocidad de aprendizaje, es decir, se reinicia, haciendo que se salte a zonas diferentes.

Concretando, el número de épocas entre cada restablecimiento de la tasa de aprendizaje se indica con "cycle_len", y el número de veces que se produce "número de ciclos" se pasa en el segundo parámetro a la funcion "fit".

In [35]:

```
1 learn.fit(1e-2, 3, cycle_len=1)
```

```
Epoch                                    100% 3/3 [00:14<00:00, 4.78s/it]

 epoch       trn_loss   val_loss   accuracy
 0           0.394446   0.244209   0.895
 1           0.35615    0.246833   0.92
 2           0.338643   0.212681   0.915
```

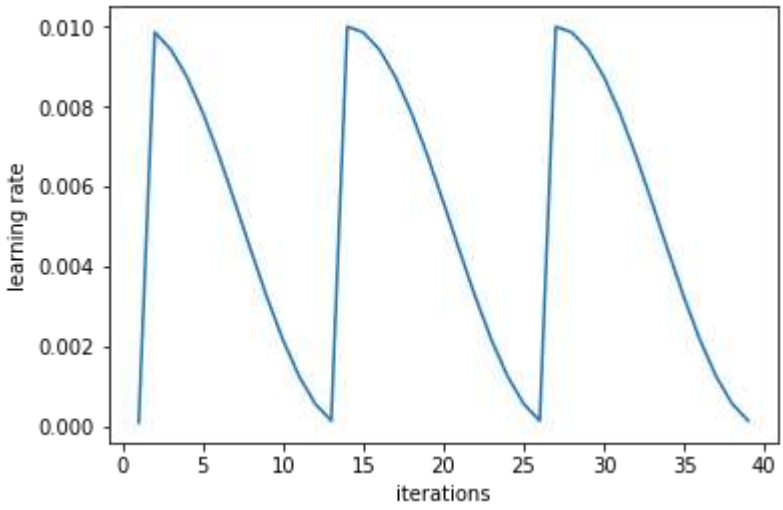
Out[35]:

```
[array([0.21268]), 0.915]
```

Como curiosidad y para mayor claridad veamos gráficamente como varía la tasa de aprendizaje en cada

[36]:

```
learn.sched.plot_lr()
```




iteración

Conviene conocer los parámetros para salvar y recuperar nuestro modelo.

learn.save('224_lastlayer')

learn.load('224_lastlayer')

## Fine-tuning con rango de aprendizaje diferencial

Aún podemos mejorar más nuestro modelo, para ello utilizaremos otra técnica muy habitual en redes neuronales convolucionales, la conocida como Fine-tuning. La cual consiste en descongelar los pesos de las capas convolucionales permitiendo a la red en su conjunto entrenarse. Esto lo conseguiremos con la función unfreeze.

Pero no hay que olvidar que los pesos pueden tener información de sumo interés. Por todo ello, lo que se hace es aplicar lo que se denomina "tasa de aprendizaje diferencial", consistente en aplicar un array con tres tasas de aprendizaje distintas. El primero de los valores del array se aplica a las primeras capas, el segundo a las intermedias y el tercero a las últimas. Los valores del array serán sucesivamente mayores; para tener en cuenta que las primeras capas respecto a las últimas tienen conocimientos más generales que no requieren de grandes modificaciones.

Señalar que también utilizaremos el parámetro cycle_mult con valor 2, lo que supone doblar el ciclo en cada paso.

In [36]:

```
learn.unfreeze()
lr=np.array([1e-4,1e-3,1e-2])
learn.fit(lr, 3, cycle_len=1, cycle_mult=2)
```

Epoch 100% 7/7 [01:00<00:00, 8.60s/it]

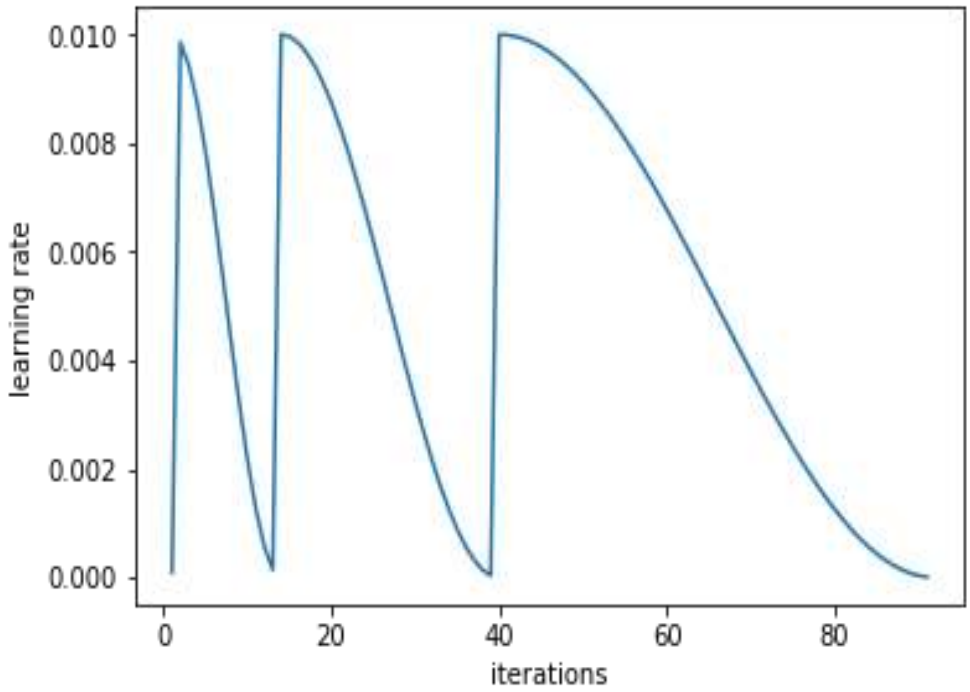
```
epoch  trn_loss   val_loss   accuracy
0      0.491768   0.222324   0.94
1      0.355266   0.170815   0.935
2      0.279037   0.110505   0.955
3      0.218481   0.057111   0.99
4      0.175096   0.039831   0.985
5      0.146585   0.02911    0.995
6      0.129172   0.028368   0.995
```

```
[array([0.02837]), 0.995]
```

Veamos nuevamente como se muestra gráficamente la tasa de aprendizaje y como le afecta el nuevo parámetro cycle_mult:

In [37]:

```
1 learn.sched.plot_lr()
```




Por último también podemos utilizar el conocido como aumento en tiempo de prueba o TTA, para hacer predicciones no sólo en el conjunto de validación, sino incluir también las imágenes obtenidas por aumentos aleatorios. Por defecto toma la imagen original y cuatro versiones aumentadas de forma aleatoria, hace la predicción promedio y es la que utiliza finalmente. Esto se consigue con la función TTA:

In [38]:

```
log_preds,y = learn.TTA()
probs = np.mean(np.exp(log_preds),0)
```

In [39]:

```
accuracy_np(probs, y)
```

```
0.985
```

# Analizando resultados

## Matriz de confusión

In [40]:

```
preds = np.argmax(probs, axis=1)
probs = probs[:,1]
```

Para el análisis final recurriremos a la habitual matriz de confusión, aprovechando las posibilidades que nos da en este sentido Scikit-learn:

In [41]:

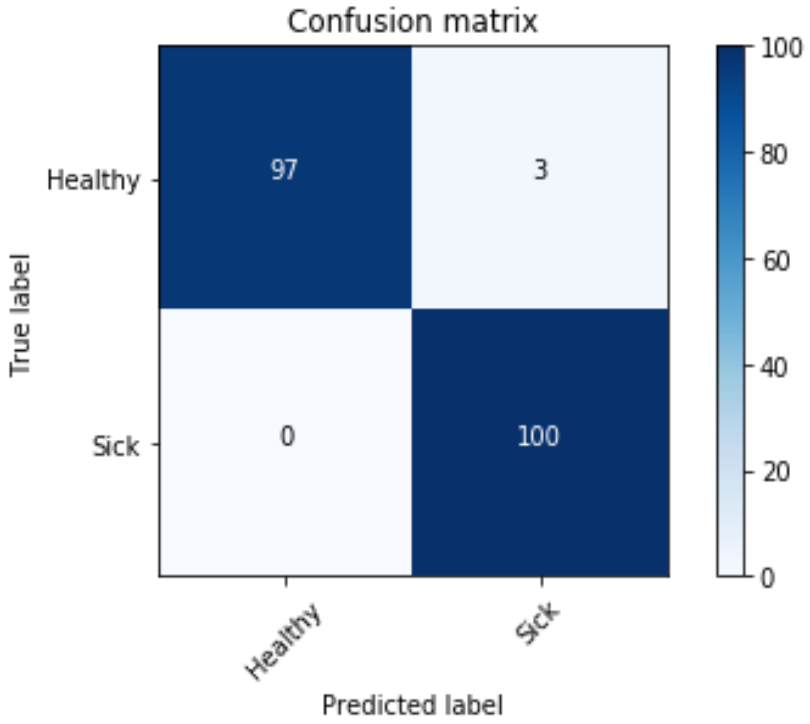
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, preds)
```

Veámoslo gráficamente:

In [42]:

```
plot_confusion_matrix(cm, data.classes)
```

```
[[ 97   3]
 [  0 100]]
```



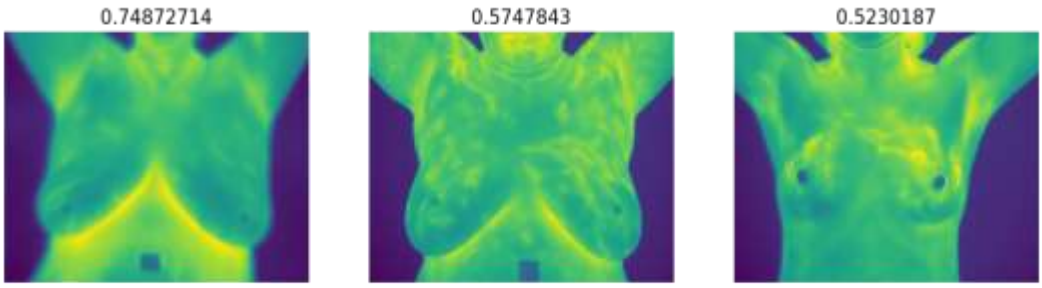

## Analizando las imágenes:

Veamos finalmente las imágenes que dan problemas:

In [43]:

```
1  plot_val_with_title(most_by_correct(0, False), "Most incorrect Healthy")
```

```
Most incorrect Healthy
```




In [44]:

```
1  plot_val_with_title(most_by_correct(1, False), "Most incorrect sick")
```

```
Most incorrect sick
Not Found.
```